Progress of Concurrent Objects with Partial Methods (Extended Version)

HONGJIN LIANG and XINYU FENG*, Nanjing University, China and University of Science and Technology of China, China

Various progress properties have been proposed for concurrent objects, such as wait-freedom, lock-freedom, starvation-freedom and deadlock-freedom. However, none of them applies to concurrent objects with partial methods, i.e., methods that are supposed *not* to return under certain circumstances. A typical example is the `lock_acquire` method, which must *not* return when the lock has already been acquired.

In this paper we propose two new progress properties, partial starvation-freedom (PSF) and partial deadlock-freedom (PDF), for concurrent objects with partial methods. We also design four patterns to write abstract specifications for PSF or PDF objects under strongly or weakly fair scheduling, so that these objects contextually refine the abstract specifications. Our Abstraction Theorem shows the equivalence between PSF (or PDF) and the progress-aware contextual refinement. Finally, we generalize the program logic LiLi to have a new logic to verify the PSF (or PDF) property and linearizability of concurrent objects.

CCS Concepts: • **Theory of computation → Hoare logic**; **Program specifications**; **Program verification**; **Abstraction**;

Additional Key Words and Phrases: Concurrent Objects, Progress, Refinement, Partial Methods

ACM Reference Format:
Hongjin Liang and Xinyu Feng. 2018. Progress of Concurrent Objects with Partial Methods (Extended Version). 1, 1 (January 2018), 149 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

A concurrent object consists of shared data and a set of methods which provide an interface for client threads to access the shared data. Linearizability [Herlihy and Wing 1990] has been used as a standard definition of the correctness of concurrent object implementations. It describes safety and functionality, but has no requirement about termination of object methods. Various progress properties, such as wait-freedom, lock-freedom, starvation-freedom and deadlock-freedom, have been proposed to specify termination of object methods. In their textbook Herlihy and Shavit [2008] give a systematic introduction of these properties.

Recent work [Filipović et al. 2009] has shown the equivalence between linearizability and a contextual refinement. The result has been further extended [Gotsman and Yang 2011; Liang and Feng 2016; Liang et al. 2013] to show that, when progress properties are taken into account, one may have the corresponding progress-aware contextual refinement to reestablish the equivalence. The equivalence results allow us to build abstractions for linearizable objects so that safety and progress of the client code can be reasoned about at a more abstract level.

However, *none* of these progress-related results applies to concurrent objects with partial methods! A method is *partial* if it is supposed *not* to return under certain circumstances. A typical example is the `lock_acquire` method, which must *not* return when the lock has already been acquired. Concurrent objects with partial methods simply do not satisfy any of the aforementioned

*Corresponding author.

Authors' address: Hongjin Liang, hongjin@nju.edu.cn; Xinyu Feng, xyfeng@nju.edu.cn, State Key Laboratory for Novel Software Technology, Nanjing University, 163 Xianlin Road, Nanjing, 210023, China , University of Science and Technology of China, 443 Huangshan Road, Hefei, 230000, China.

2018. XXXX-XXXX/2018/1-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

progress properties, and people do not know how to give progress-aware abstract specifications for them either. The existing studies on progress properties and progress-aware contextual refinements have been limited to concurrent objects with total specifications.

As an awkward consequence, we cannot treat lock implementations as objects when we study progress of concurrent objects. Instead, we have to treat `lock_acquire` and `lock_release` as *internal* functions of other lock-based objects. Also, without a proper progress-aware abstraction for locks, we have to redo the verification of `lock_acquire` and `lock_release` when they are used in different contexts [Liang and Feng 2016], which makes the verification process complex and painful. Note that locks are not the only objects with partial methods. Concurrent sets, stacks and queues may also have methods that intend to block. For instance, it may be sensible for a thread attempting to pop from an empty stack to block, waiting until another thread pushes an item. The reasoning about these objects suffers from the same problems too when progress is concerned.

We face the following key challenges to address these problems.

- We need definitions of new progress properties for these objects, and the definitions need to describe the situations in which permanent blocking is allowed. It is important to note that, although deadlock-freedom and starvation-freedom have been used as progress properties for "blocking" algorithms [Herlihy and Shavit 2008], they allow permanent blocking only when the scheduling is unfair. They can specify concurrent objects implemented using locks, but they do not apply to lock objects themselves. For objects like locks, blocking may also be caused by inappropriate method invocations by the client. For instance, if a thread of the client fails to call `lock_release` after acquiring the lock, the calls to `lock_acquire` by other threads will be always blocked. Similarly, for a stack object with a partial pop method, if no client threads call push, the calls to pop will be permanently blocked at an empty stack. The question is, how to distinguish the blocking behaviors caused by "bad" clients with those caused by bad object implementations, and blame the objects only for the blocking in the latter case.
- The abstractions for objects with partial methods should be able to distinguish the objects with different progress guarantees under different scheduling conditions. A natural abstraction for partial methods is the blocking primitive **await**$(B)\{C\}$. It is blocked if B does not hold, and executes C atomically if B holds (in this case, we say the code **await**$(B)\{C\}$ is *enabled*). A specification in the form of **await**$(B)\{C\}$ can characterize both the atomicity of the functionality and the fact that the method is partial. However, it is not sufficient to serve as a progress-aware abstraction for the following two reasons.
  - Different implementations of the same **await** block may exhibit different progress properties, requiring different abstractions. For instance, the ticket lock algorithm [Mellor-Crummey and Scott 1991] has stronger progress guarantees than the test-and-set lock algorithm [Herlihy and Shavit 2008]. Therefore when progress is concerned it is impossible to use the same partial specification (e.g., **await**`(l=0){l := cid}`, where `cid` is the ID of the current thread) as an abstraction for the `lock_acquire` methods in both algorithms (even though it may work for both if we consider linearizability only).
  - Even the same implementation may require different abstractions for different scheduling. The blocking primitive **await**$(B)\{C\}$ behaves differently under strongly fair and weakly fair scheduling. The former ensures the execution of the primitive as long as it is enabled a sufficient number of times, but the latter requires the primitive to be *always* enabled to ensure its execution. On the other hand, the distinction between strong and weak fairness is meaningful only if there are blocking primitives. A low-level program consisting of non-blocking primitive instructions only (like most machine instructions) behaves the same

under both scheduling. Such a program cannot have the same abstraction with blocking primitives under different scheduling.

As a result, if we consider m kinds of progress properties (e.g., to distinguish ticket locks and test-and-set locks) and the 2 choices of strongly fair and weakly fair scheduling, we may need as many as $2 \times m$ kinds of abstractions for the same functionality. Can we find general patterns for these abstractions?

In this paper, we specify and verify progress of concurrent objects with partial methods. We define progress properties and abstraction patterns under strongly and weakly fair scheduling. Then we prove that given a linearizable object implementation Π, the contextual refinement between Π and its abstraction Π' under a certain kind of fair scheduling is equivalent to the progress property of Π. We also provide a program logic to verify the contextual refinement between Π and Π', which ensures linearizability and the progress property of Π.

Our work is based on earlier work on abstraction for concurrent objects and concurrency verification, but makes the following new contributions:

- We propose progress properties, *partial starvation-freedom (PSF)* and *partial deadlock-freedom (PDF)*, for concurrent objects with partial methods. They identify the execution scenarios in which the partial methods are blocked due to inappropriate invocation sequences made by "bad" clients, and allow the object methods to be blocked permanently in these special scenarios. Ticket locks and test-and-set locks satisfy PSF and PDF respectively. Traditional starvation-freedom and deadlock-freedom for objects with total methods can be viewed as specializations of PSF and PDF respectively, if we view total methods as special cases of partial ones that are always enabled to return.
- We design four general patterns for abstractions for concurrent objects with PSF and PDF progress properties under strongly and weakly fair scheduling, respectively. We start with the basic blocking primitive **await**(B)\{C\} and define syntactic wrappers that transform it into non-atomic object specifications which can be refined by the object implementations in the progress-aware contextual refinement. We give distinguished wrappers for different combinations of fairness and progress properties.
- We prove the equivalence between PSF (or PDF) and the progress-aware contextual refinement, where the abstraction is generated by the wrapper, under strong or weak fairness. The equivalence results (called the abstraction theorem) allow us to verify safety and liveness properties of client programs at a high abstraction level, by replacing concrete object implementations with the specifications. Using the natural transitivity of the contextual refinement, it is also possible to verify linearizability and PSF (or PDF) of nested concurrent objects.
- We design a program logic to verify objects with PSF or PDF progress properties. Our logic is a generalization of the LiLi logic for starvation-free and deadlock-free objects [Liang and Feng 2016]. It also provides inference rules for the **await**(B)\{C\} statement under strong and weak fairness, so that **await** commands can also be used in object implementations. The soundness of our logic ensures the progress-aware contextual refinement, and linearizability and PSF (or PDF) under different fairness. We have applied the program logic to verify ticket locks [Mellor-Crummey and Scott 1991], test-and-set locks [Herlihy and Shavit 2008], bounded partial queues with two locks [Herlihy and Shavit 2008] and Treiber stacks [Treiber 1986] with possibly blocking pop methods.

In the rest of this paper, we first give an informal overview of the background and our key ideas in Sec. 2. Then we introduce the object language in Sec. 3, and linearizability and the basic contextual refinement in Sec. 4. We propose our new progress properties in Sec. 5, and give the progress-aware contextual refinement and the abstraction theorem in Sec. 6. We present the logic

```
L_initialize(){  l := 0; }

L_acq(){
1  local b := false;
2  while(!b){ b := cas(&l, 0, cid); }
}

L_rel(){
3  l := 0;
}
```

(a) test-and-set lock implementation

```
inc(){ L_acq(); x:=x+1; L_rel(); }
```

(b) counter with a test-and-set lock

```
tkL_initialize(){ owner := 0; next := 0; }

tkL_acq(){
1  local i, o;
2  i := getAndInc(&next);
3  o := owner; while(i!=o){ o := owner; }
}

tkL_rel(){
4  owner := owner + 1;
}
```

(c) ticket lock implementation

```
inc_tkL(){ tkL_acq(); x:=x+1; tkL_rel(); }
```

(d) counter with a ticket lock

```
INC(){x:=x+1;}
```

(e) atomic spec. INC

Fig. 1. Counters with locks.

in Sec. 7 and show the examples we have verified in Sec. 8. Finally, we discuss related work and conclude in Sec. 9.

2 BACKGROUND AND OVERVIEW OF KEY IDEAS

Below we first give an overview of linearizability, starvation-freedom, deadlock-freedom and contextual refinement. Then we analyze the problems in defining progress of concurrent objects with partial methods, and explain our solutions informally.

2.1 Background

A concurrent object usually satisfies linearizability, a standard safety criterion, and certain progress property, describing when and how method calls of the object are guaranteed to terminate.

Linearizability. A concurrent object is linearizable, if each method call appears to take effect instantaneously at some moment between its invocation and return [Herlihy and Wing 1990]. Intuitively, linearizability requires the implementation of each method to have the same effect as an atomic specification.

Consider the two implementations of the counter object in Fig. 1(b) and (d). We assume that every primitive command is executed atomically. A counter provides a method `inc` for incrementing the shared data `x`. Both implementations use locks to synchronize the increments. Intuitively they have the same effect as the atomic specification `INC()` in Fig. 1(e), so they are linearizable.

The locks themselves could also be viewed as standalone objects. For instance, the test-and-set lock object in Fig. 1(a) provides the methods `L_acq` and `L_rel` for a thread to acquire and release the lock `l`. Here `cid` represents the current thread's ID, which is a positive number. The counter's implementation code in Fig. 1(b) can be viewed as a client of this lock object. The lock object is linearizable, because `L_acq` and `L_rel` both update `l` atomically (if they indeed return). They

produce the same effects as the atomic operations `L_ACQ` and `L_REL` (defined below), respectively:

```
L_ACQ(){ l := cid; }      L_REL(){ l := 0; }                    (2.1)
```

However, linearizability does not characterize progress properties of the object implementations. For instance, the following counter object is still linearizable, even if its method never terminates.

```
inc'(){ L_acq(); x:=x+1; L_rel(); while(true) skip; }
```

Progress properties. Various progress properties have been proposed for concurrent objects, such as wait-freedom and lock-freedom for non-blocking implementations, and starvation-freedom and deadlock-freedom for lock-based implementations. These properties describe conditions under which a method call is guaranteed to successfully finish in an execution. The two implementations of the counter in Fig. 1(b) and (d) satisfy deadlock-freedom and starvation-freedom respectively.

We use the definitions given by Herlihy and Shavit [2011]. Both deadlock-freedom and starvation-freedom assume fair scheduling, i.e., every thread gets eventually executed. For the counters in Fig. 1(b) and (d), fairness ensures that every thread holding the lock will eventually release the lock.

Deadlock-freedom requires "minimal progress" in fair executions, i.e., there always exists some method call which can finish under fair scheduling, while starvation-freedom requires "maximal progress" in fair executions, i.e., every method call should eventually finish under fair scheduling.

The counter in Fig. 1(b) is deadlock-free, because the test-and-set lock (see Fig. 1(a)) guarantees that eventually some thread will succeed in getting the lock via the **cas** instruction at line 2, and hence the method call of `inc` in that thread will eventually finish. It is not starvation-free, because there might be a thread that continuously fails to acquire the lock. For the following client program (2.2), the **cas** instruction executed by the left thread could always fail if the right thread infinitely often acquires the lock.

```
inc(); print(1);   ||   while(true) inc();                      (2.2)
```

The counter in Fig. 1(d) implemented with a ticket lock is starvation-free. Figure 1(c) shows the details of the ticket lock implementation. It uses two shared variables `owner` and `next`, which are equal initially. The threads attempting to acquire the lock form a waiting queue. In `tkL_acq`, a thread first atomically increments `next` and reads its old value to a local variable `i` (line 2). It waits until the lock's `owner` equals its ticket number `i` (line 3), then it acquires the lock. In `tkL_rel`, the thread releases the lock by incrementing `owner` (line 4). Then the next waiting thread (the thread with ticket number `i+1`, if there is one) can acquire the lock. We can see that the ticket lock implementation ensures the first-come-first-served property, and hence every thread calling `inc_tkL` can eventually acquire the lock and finish its method call.

Deadlock-freedom and starvation-freedom are progress properties for the so-called "blocking implementations" [Herlihy and Shavit 2008], such as the counters in Fig. 1(b) and (d), where a thread holding a lock will block other threads requesting the lock. However, they do not apply to lock objects, e.g., the ones in Fig. 1(a) and (c). We will explain the problems in detail in Sec. 2.2.

Contextual refinement and the abstraction theorem. It is difficult to use linearizability and progress properties directly in modular verification of client programs of an object, because their definitions fail to describe how the client behaviors are affected. To verify clients, we would like to abstract away the details of the object implementation. This requires a notion of object correctness, telling us that the client behaviors will not change when we replace the object methods' implementations with the corresponding abstract operations (as specifications).

Contextual refinement gives the desired notion of correctness. Informally, an object implementation Π is a contextual refinement of a (more abstract) implementation Π', if every observable

behavior of any client program using Π can also be observed when the client uses Π' instead. Then, when verifying a client of Π, we can soundly replace Π with its abstraction Π'.

There has been much work (e.g., [Filipović et al. 2009; Gotsman and Yang 2012; Liang et al. 2013]) studying abstraction theorems, which relate linearizability and progress properties with contextual refinements. It has been proved that linearizability of Π is equivalent to a contextual refinement between Π and its *atomic specification* Γ, where the observable behaviors are finite traces of I/O events. When taking progress properties into account, the corresponding contextual refinement should be sensitive to termination or divergence (non-termination). For instance, deadlock-freedom or starvation-freedom of linearizable objects is shown equivalent to a contextual refinement which observes (possibly infinite) full traces of I/O events in fair executions. Then, a client which diverges with Π in a fair execution must also have a diverging execution when using the abstraction Π'. Deadlock-free and starvation-free objects could be distinguished by different abstractions. The abstraction for starvation-free objects is the atomic specification Γ, while for deadlock-free ones the abstraction has to be non-atomic [Liang and Feng 2016].

The counter implementation `inc_tkL()` in Fig. 1(d) is a progress-aware contextual refinement of the atomic counter `INC` in Fig. 1(e), but `inc()` in Fig. 1(b) is not. To see the difference, consider the client program (2.2). Under fair scheduling, the client calling `inc()` may generate an empty I/O event trace because it may not print out 1. However, the empty trace cannot be generated when replacing `inc()` with `inc_tkL()` or `INC()`, because the resulting program must print out 1.

2.2 Problems and Our Solutions

The existing progress properties and the corresponding contextual refinement are proposed for concurrent objects with total methods only, i.e., methods that should always return when executed sequentially. They do not apply to objects with partial methods, such as the lock objects in Fig. 1(a) and (c), which intend to block at certain situations. We have outlined the key challenges in reasoning about progress properties of objects with partial methods in Sec. 1. We give more detailed explanations here.

2.2.1 Atomic Specifications Need to Be Partial. The specifications defined in (2.1) can characterize the atomic behaviors of lock objects, but they fail to specify that `L_ACQ` should be partial in the sense that it should be blocked when the lock is unavailable.

To address the problem, we introduce the *atomic partial specification* Γ, where each method specification is in the form of $\mathbf{await}(B)\{C\}$. For the lock objects, we can define the atomic partial specification Γ as follows.

```
L_ACQ'(){  await (l = 0) { l := cid };  }        L_REL(){ l := 0; }      (2.3)
```

The **await** block naturally specifies the atomicity of method functionality, just like the traditional atomic specification $\langle C\rangle$ (which can be viewed as syntactic sugar for $\mathbf{await}(\mathtt{true})\{C\}$), therefore Γ may serve as a specification for linearizable objects. It also shows the fact that the object method is partial, with explicit specification of the enabling condition B. Below we use the atomic partial specification as the starting point to characterize the progress of objects.

2.2.2 Deadlock-Freedom and Starvation-Freedom Do Not Apply. We need new progress properties for objects with partial methods. Consider the following client program (2.4) using the test-and-set lock in Fig. 1(a). One of the method calls never finishes.

```
L_acq();  ||  L_acq();                                   (2.4)
```

It shows that the test-and-set lock object does not satisfy the traditional deadlock-freedom or starvation-freedom property we just presented. Neither does the ticket lock object in Fig. 1(c).

Table 1. Client (2.5) with different locks. "Yes" means it must print out 1, "No" otherwise.

	spec. (2.3)	ticket locks (Fig. 1(c))	test-and-set locks (Fig. 1(a))
Strong Fairness	Yes	Yes	No
Weak Fairness	No	Yes	No

The problem is that `L_acq` intends to block when the lock is not available. The non-termination in the above example (2.4) is just the intention of a correct lock implementation; otherwise the lock cannot guarantee mutual exclusion.

Our solution. We define two new progress properties for objects with partial methods, which we call partial starvation-freedom (PSF) and partial deadlock-freedom (PDF). PSF requires that in each fair execution trace by any client with the object Π, either each method invocation eventually returns (as required in starvation-freedom), or each pending method invocation must be continuously *disabled*. The latter case intuitively says that this non-termination is caused by the "bad" client, e.g., by inappropriate invocations of the methods. Similarly, PDF requires that in each fair execution trace by any client with the object Π, either there always *exists* a method invocation that eventually returns (as in deadlock-freedom), or each pending method invocation must be continuously disabled.

But how do we formally say that a method is disabled? When we informally say this, we actually refer to the enabling condition B in **await**(B)$\{C\}$ in the object's atomic partial specification Γ. However, we may not be able to infer such a condition from the concrete implementation Π. To address this problem, our definitions of $\mathsf{PSF}_\Gamma(\Pi)$ and $\mathsf{PDF}_\Gamma(\Pi)$ are parameterized with the specification Γ (the actual definitions take more parameters, as shown in Sec. 5).

We can prove the lock objects in Fig. 1(a) and (c) satisfy PDF and PSF respectively. We can also show that starvation-freedom and deadlock-freedom are special cases of our PSF and PDF respectively, by instantiating the parameter Γ with specifications in the form of **await**(true)$\{C\}$.

2.2.3 Atomic Partial Specifications Are Insufficient for Progress-Aware Abstractions. Although the atomic partial specification Γ describes the atomic functionality and the enabling condition of each method, it is insufficient to serve as a progress-aware abstraction for the following reasons.

First, the progress of the **await** command itself is affected by the fairness of scheduling, such as strong fairness and weak fairness.

- *Strong fairness*: Every thread which is infinitely often enabled will execute infinitely often. Then, **await**(B)$\{C\}$ is not executed only if B is continuously false after some point in the execution trace.
- *Weak fairness*: Every thread which is eventually always enabled will execute infinitely often. Then, **await**(B)$\{C\}$ may not be executed when B is infinitely often false. This fairness notion is weaker than strong fairness.

As a result, the choice of fair scheduling will affect the behaviors of a program or a specification with **await** commands. To see this, we consider the following client program (2.5).

$$[\_]_{\text{ACQ}};\ [\_]_{\text{REL}};\ \texttt{print(1);} \quad || \quad \texttt{while(true)\{}\ [\_]_{\text{ACQ}};\ [\_]_{\text{REL}};\ \texttt{\}} \tag{2.5}$$

where $[\_]_{\text{ACQ}}$ and $[\_]_{\text{REL}}$ represent holes to be filled with method calls of lock acquire and release, respectively. Table 1 shows the behaviors of the client with different locks. If the client calls the abstract specifications in (2.3), it must execute `print(1)` under strongly fair scheduling, but may not do so under weakly fair scheduling. This is because the call of `L_ACQ'` could be infinitely often enabled and infinitely often disabled in an execution, making its termination sensitive to the fairness of scheduling.

Table 2. Wrappers for atomic specifications.

	PSF	PDF
Strong Fairness	$\mathrm{wr}_{\mathrm{PSF}}^{\mathrm{sfair}}(\mathbf{await}(B)\{C\})$	$\mathrm{wr}_{\mathrm{PDF}}^{\mathrm{sfair}}(\mathbf{await}(B)\{C\})$
Weak Fairness	$\mathrm{wr}_{\mathrm{PSF}}^{\mathrm{wfair}}(\mathbf{await}(B)\{C\})$	$\mathrm{wr}_{\mathrm{PDF}}^{\mathrm{wfair}}(\mathbf{await}(B)\{C\})$

Also note that the two fairness notions coincide when the program does not contain blocking operations. Therefore, regardless of strongly or weakly fair scheduling, the client (2.5) using ticket locks always executes `print(1)`, but it may not do so if using test-and-set locks instead (see Table 1).

As a result, for the same object implementation, we may need *different abstractions under different scheduling*. As shown in Table 1, the specification (2.3) cannot serve as the specification of the test-and-set locks under both strong fairness and weak fairness.

Second, even under the same scheduling, *different implementations demonstrate different progress, therefore need different abstractions*. As shown in Table 1, the different lock implementations have different behaviors, even under the same scheduling.

For the above two reasons, we need different abstractions for different combinations of fairness and progress. For PSF and PDF under strong and weak fairness respectively, we may need four different abstractions. Can we systematically generate all of them?

Our solution. We define code wrappers over the basic blocking primitive $\mathbf{await}(B)\{C\}$ to generate the abstractions. The code wrappers are syntactic transformations that transform $\mathbf{await}(B)\{C\}$ into possibly non-atomic object specifications which can be refined by the object implementations in the progress-aware contextual refinement. As shown in Table 2, the four wrappers correspond to all combinations of fairness and progress. The definitions are shown in Sec. 6. Here we only give some high-level intuitions using the lock objects as examples.

First, we observe that the lock specification (2.3) can already serve as an abstraction for ticket locks under strong fairness, or for test-and-set locks under weak fairness. In general, the wrapper $\mathrm{wr}_{\mathrm{PSF}}^{\mathrm{sfair}}$ can be an identity function, i.e., the atomic partial specifications are already proper abstractions for PSF objects (not only for locks) under strong fairness. But $\mathrm{wr}_{\mathrm{PDF}}^{\mathrm{wfair}}$ is subtle. The atomic partial specifications are insufficient as abstractions for general PDF objects under weak fairness, which we will explain in detail in Sec. 6.

Second, as we have seen from Table 1, the lock specification (2.3) does not work for PSF locks under weak fairness nor for PDF locks under strong fairness. Then the role of the wrapper $\mathrm{wr}_{\mathrm{PSF}}^{\mathrm{wfair}}$ (or $\mathrm{wr}_{\mathrm{PDF}}^{\mathrm{sfair}}$) is to generate the same PSF (or PDF) behaviors even though the fairness of scheduling is weaker (or stronger).

To guarantee PSF, the idea is to create some kind of "fairness" on termination, i.e., every method call can get the chance to terminate. Given weakly fair scheduling, this requires the enabling condition of the abstraction to continuously remain true. As a result, a possible way to define $\mathrm{wr}_{\mathrm{PSF}}^{\mathrm{wfair}}(\texttt{L\_ACQ'})$ is to keep a queue of threads requesting the lock, and a thread can acquire the lock only when it is at the head of the queue.

To support PDF under strongly fair scheduling, we have to allow non-termination even if the enabling condition is infinitely often true. For the client (2.5), the call of `L_ACQ'` in the specification (2.3) under strongly fair scheduling always terminates. Then $\mathrm{wr}_{\mathrm{PDF}}^{\mathrm{sfair}}$ needs to incorporate with some kind of delaying mechanisms, so that the termination of `L_ACQ'` of the left thread could be delayed every time when the right thread succeeds in acquiring the lock.

$$
\begin{array}{rrcl}
(\mathit{MName}) & f, g \ldots & & (\mathit{PVar})\ x, y, z \ldots \\
(\mathit{Expr}) & E & ::= & x \mid n \mid E + E \mid \ldots \qquad (\mathit{BExp})\ B ::= \mathsf{true} \mid \mathsf{false} \mid E = E \mid \neg B \mid \ldots \\
(\mathit{Stmt}) & C & ::= & x := E \mid x := [E] \mid [E] := E \mid \mathbf{print}(E) \mid x := \mathbf{cons}(E, \ldots, E) \mid \mathbf{dispose}(E) \\
 & & \mid & \mathbf{skip} \mid x := f(E) \mid \mathbf{return}\ E \mid C; C \mid \mathbf{if}\ (B)\ C\ \mathbf{else}\ C \mid \mathbf{while}\ (B)\{C\} \\
 & & \mid & \mathbf{await}(B)\{C\} \\
(\mathit{ODecl}) & \Pi, \Gamma & ::= & \{f_1 \rightsquigarrow (\mathcal{P}_1, x_1, C_1), \ldots, f_n \rightsquigarrow (\mathcal{P}_n, x_n, C_n)\} \\
(\mathit{Prog}) & W & ::= & \mathbf{let}\ \Pi\ \mathbf{in}\ \hat{C}_1 \,\|\, \ldots \,\|\, \hat{C}_n \qquad (\mathit{Thrd})\ \hat{C} ::= C \mid \mathbf{end}
\end{array}
$$

Fig. 2. Syntax of the programming language.

2.2.4 Other Results. We also have the following new results in addition to the new progress properties and code wrappers.

Abstraction theorem. We prove the abstraction theorem, saying that our new progress properties PSF and PDF (together with linearizability) are equivalent to contextual refinements where the abstractions are generated by the corresponding wrappers. On the one hand, the theorem justifies the abstractions generated by our wrappers, showing that they are refined by linearizable and PSF (or PDF) object implementations. On the other hand, it also justifies our formulation of PSF and PDF by showing that they imply progress-aware contextual refinements.

The abstraction theorem also allows us to verify safety and progress properties of whole programs (consisting of clients and objects) in a modular way — after proving linearizability and PSF (or PDF) of an object Π with respect to its atomic partial specification Γ, we can replace Π with the abstraction generated by applying the corresponding wrapper over Γ, and then reason about properties of the whole program at the high abstraction level.

Program logic. Finally we design a program logic as the proof method for verifying PSF and PDF objects. It ensures linearizability and PSF (or PDF) of an object Π with respect to its atomic partial specification Γ. The logic is a generalization of our previous program logic LiLi for starvation-free and deadlock-free objects [Liang and Feng 2016], plus new inference rules for the **await** statement under strong and weak fairness. We will explain the details in Sec. 7.

3 THE LANGUAGE

Figure 2 shows the syntax of the language. A *program* W consists of an *object declaration* Π and n parallel threads $\hat{C}$ as *clients* sharing the object. To simplify the language, we assume there is only one object in each program. Each Π maps method names f_i to annotated method implementations $(\mathcal{P}_i, x_i, C_i)$, where x_i and C_i are the formal parameter and the method body respectively, and the assertion $\mathcal{P}_i$ is an annotated precondition over the object state to ensure the safe execution of the method. It is defined in Fig. 3 and is used in the operational semantics explained below. A thread $\hat{C}$ is either a command C, or an **end** flag marking termination of the thread. The commands include the standard ones used in separation logic, where $x := [E]$ and $[E] := E'$ read and write the heap at the location E respectively, and $x := \mathbf{cons}(E, \ldots, E)$ and $\mathbf{dispose}(E)$ allocate and free memory cells respectively. In addition, we have method call $(x := f(E))$ and return ($\mathbf{return}\ E$) commands. The $\mathbf{print}(E)$ command generates *externally observable events*, which are used to define trace refinements in Sec. 4. The $\mathbf{await}(B)\{C\}$ command is the only *blocking* primitive in the language. It blocks the current thread if B does not hold, otherwise C is executed *atomically* together with the testing of B.

$$
\begin{array}{rlcl@{\qquad}rlcl}
(\mathit{ThrdID}) & \mathsf{t} & \in & \mathit{Nat} & (\mathit{Store}) & s,\mathbb{s} & \in & \mathit{PVar} \rightharpoonup \mathit{Val} \\
(\mathit{Heap}) & h,\mathbb{h} & \in & \mathit{Nat} \rightharpoonup \mathit{Val} & (\mathit{Data}) & \sigma,\Sigma & ::= & (s,h) \\
(\mathit{CallStk}) & \kappa,\mathbb{k} & ::= & \circ \mid (s_l,x,C) & (\mathit{ThrdPool}) & \mathcal{K},\mathbb{K} & ::= & \{\mathsf{t}_1 \rightsquigarrow \kappa_1,\ldots,\mathsf{t}_n \rightsquigarrow \kappa_n\} \\
(\mathit{PState}) & \mathcal{S},\mathbb{S} & ::= & (\sigma_c,\sigma_o,\mathcal{K}) & (\mathit{LState}) & \varsigma,\delta & ::= & (\sigma_c,\sigma_o,\kappa)
\end{array}
$$

$$
\begin{array}{rlcl@{\qquad}rlcl}
(\mathit{ExecCtxt}) & \mathbf{E} & ::= & [\,] \mid \mathbf{E};C \\
(\mathit{Pre}) & \mathcal{P} & \in & \mathscr{P}(\mathit{Data}) & (\mathit{AbsFun}) & \varphi & \in & \mathit{Data} \rightharpoonup \mathit{Data}
\end{array}
$$

$$
\begin{array}{rlcl}
(\mathit{Event}) & e & ::= & (\mathsf{t},f,n) \mid (\mathsf{t},\mathbf{ret},n) \mid (\mathsf{t},\mathbf{obj}) \mid (\mathsf{t},\mathbf{obj},\mathbf{abort}) \mid (\mathsf{t},\mathbf{out},n) \\
 & & \mid & (\mathsf{t},\mathbf{clt}) \mid (\mathsf{t},\mathbf{clt},\mathbf{abort}) \mid (\mathsf{t},\mathbf{term}) \mid (\mathbf{spawn},n) \\
(\mathit{BIdSet}) & \Delta & \in & \mathscr{P}(\mathit{ThrdID}) \qquad (\mathit{PEvent})\ \iota ::= (e,\Delta_c,\Delta_o) \\
(\mathit{ETrace}) & \mathcal{E} & ::= & \epsilon \mid e::\mathcal{E} \quad \text{(co-inductive)} \qquad (\mathit{PTrace})\ T ::= \epsilon \mid \iota::T \quad \text{(co-inductive)}
\end{array}
$$

$$
\mathsf{en}(\hat{C}) \stackrel{\text{def}}{=} \begin{cases} B & \text{if } \exists \mathbf{E},C'.\ \hat{C} = \mathbf{E}[\mathbf{await}(B)\{C'\}] \\ \mathsf{true} & \text{otherwise} \end{cases}
$$

$$
(\sigma_o,\kappa) \models B \ \text{ iff } \ [\![B]\!]_{((\sigma_o.s)\uplus(\kappa.s_l))} = \mathsf{true} \wedge \kappa \neq \circ \qquad\qquad \sigma_c \models B \ \text{ iff } \ [\![B]\!]_{\sigma_c.s} = \mathsf{true}
$$

$$
\begin{array}{rcl}
\mathsf{btids}(\mathbf{let}\ \Pi\ \mathbf{in}\ \hat{C}_1 \parallel \ldots \parallel \hat{C}_n,(\sigma_c,\sigma_o,\mathcal{K})) & \stackrel{\text{def}}{=} & (\{\mathsf{t} \mid \mathcal{K}(\mathsf{t}) = \circ \wedge \neg(\sigma_c \models \mathsf{en}(\hat{C}_\mathsf{t}))\}, \\
 & & \{\mathsf{t} \mid \mathcal{K}(\mathsf{t}) \neq \circ \wedge \neg((\sigma_o,\mathcal{K}(\mathsf{t})) \models \mathsf{en}(\hat{C}_\mathsf{t}))\})
\end{array}
$$

Fig. 3. States and event traces.

We make the following assumptions to simplify the technical setting. There are no regular function calls in either clients or objects. Therefore $x := f(E)$ can only be executed in client code to call object methods, and **return** E always returns from object methods to clients. Each object method takes only one argument and each method body ends with a **return** command. Object methods never execute the **print**(E) command and therefore do *not* generate external events. The command C in **await**$(B)\{C\}$ cannot contain **await**, **print**, and method calls and returns. It cannot contain loops either so that it always terminates.

Operational semantics. The operational semantics rules shown in Fig. 4 consist of three parts, including state transitions made by the whole program, by individual threads, and by clients or object methods, respectively. We define program states $\mathcal{S}$ in Fig. 3, where we use two sets of notations to represent states at the concrete and the abstract levels respectively when we study refinement. To ensure that the client code does not touch the object data, in $\mathcal{S}$ we separate the data accessed by clients (σ_c) and by object methods (σ_o). $\mathcal{S}$ also contains a *thread pool* $\mathcal{K}$ mapping thread IDs t to the corresponding method *call stacks* κ. Recall that the only function call allowed in the language is the method invocation made by a client and there are no nested function calls, therefore each κ is either empty ($\circ$, which means the thread is executing the *client* code), or contains only *one* stack frame (s_l,x,C), where s_l is the local store for the local variables of the method, x is the (client) variable recording the return value, and C is the continuation (the remaining client code to be executed after the return of the method).

Figure 4(a) shows that the execution of the program W follows the non-deterministic interleaving semantics, which is defined based on thread transitions defined in Fig. 4(b). Each transition over program configurations is labelled with a program event ι, a triple in the form of (e,Δ_c,Δ_o). The event e is generated by thread transitions. As defined in Fig. 3, (t,f,n) records the invocation of the method f with the argument n in the thread t, and $(\mathsf{t},\mathbf{ret},n)$ is for a method return with the return value n. $(\mathsf{t},\mathbf{obj})$ and $(\mathsf{t},\mathbf{clt})$ record a regular object step and a regular client step respectively, while $(\mathsf{t},\mathbf{obj},\mathbf{abort})$ and $(\mathsf{t},\mathbf{clt},\mathbf{abort})$ are for aborting of the thread in the object and client code

$$\frac{(\hat{C}_i,(\sigma_c,\sigma_o,\mathcal{K}(i))) \xrightarrow{e}_{i,\Pi} (\hat{C}_i',(\sigma_c',\sigma_o',\kappa'))) \qquad \mathcal{K}' = \mathcal{K}\{i \rightsquigarrow \kappa'\} \quad \text{btids}(\textbf{let } \Pi \textbf{ in } \hat{C}_1 \,\|\, \ldots \hat{C}_i' \ldots \|\, \hat{C}_n,(\sigma_c',\sigma_o',\mathcal{K}')) = (\Delta_c,\Delta_o)}{(\textbf{let } \Pi \textbf{ in } \hat{C}_1 \,\|\, \ldots \hat{C}_i \ldots \|\, \hat{C}_n,(\sigma_c,\sigma_o,\mathcal{K})) \xmapsto{(e,\Delta_c,\Delta_o)} (\textbf{let } \Pi \textbf{ in } \hat{C}_1 \,\|\, \ldots \hat{C}_i' \ldots \|\, \hat{C}_n,(\sigma_c',\sigma_o',\mathcal{K}'))}$$

$$\frac{\hat{C}_i = \textbf{skip} \qquad \mathcal{K}(i) = \circ \qquad \hat{C}_i' = \textbf{end} \qquad e = (i,\textbf{term}) \quad \text{btids}(\textbf{let } \Pi \textbf{ in } \hat{C}_1 \,\|\, \ldots \hat{C}_i \ldots \|\, \hat{C}_n,(\sigma_c,\sigma_o,\mathcal{K})) = (\Delta_c,\Delta_o)}{(\textbf{let } \Pi \textbf{ in } \hat{C}_1 \,\|\, \ldots \hat{C}_i \ldots \|\, \hat{C}_n,(\sigma_c,\sigma_o,\mathcal{K})) \xmapsto{(e,\Delta_c,\Delta_o)} (\textbf{let } \Pi \textbf{ in } \hat{C}_1 \,\|\, \ldots \hat{C}_i' \ldots \|\, \hat{C}_n,(\sigma_c,\sigma_o,\mathcal{K}))}$$

$$\frac{(\hat{C}_i,(\sigma_c,\sigma_o,\mathcal{K}(i))) \xrightarrow{e}_{i,\Pi} \textbf{abort}}{(\textbf{let } \Pi \textbf{ in } \hat{C}_1 \,\|\, \ldots \hat{C}_i \ldots \|\, \hat{C}_n,(\sigma_c,\sigma_o,\mathcal{K})) \xmapsto{(e,\emptyset,\emptyset)} \textbf{abort}}$$

(a) program transitions

$$\frac{\Pi(f) = (\mathcal{P},y,C) \qquad \sigma_o \in \mathcal{P} \qquad [\![E]\!]_{s_c} = n \qquad x \in dom(s_c) \qquad \kappa = (\{y \rightsquigarrow n\},x,\mathbf{E}[\,\textbf{skip}\,])}{(\mathbf{E}[\,x := f(E)\,],((s_c,h_c),\sigma_o,\circ)) \xrightarrow{(\mathrm{t},f,n)}_{\mathrm{t},\Pi} (C,((s_c,h_c),\sigma_o,\kappa))}$$

$$\frac{f \notin dom(\Pi) \quad \text{or} \quad \sigma_o \notin \Pi(f).\mathcal{P} \quad \text{or} \quad [\![E]\!]_{s_c} \textit{ undefined} \quad \text{or} \quad x \notin dom(s_c)}{(\mathbf{E}[\,x := f(E)\,],((s_c,h_c),\sigma_o,\circ)) \xrightarrow{(\mathrm{t},\textbf{clt},\textbf{abort})}_{\mathrm{t},\Pi} \textbf{abort}}$$

$$\frac{\kappa = (s_l,x,C) \qquad [\![E]\!]_{s_l} = n \qquad s_c' = s_c\{x \rightsquigarrow n\}}{(\mathbf{E}[\,\textbf{return } E\,],((s_c,h_c),\sigma_o,\kappa)) \xrightarrow{(\mathrm{t},\textbf{ret},n)}_{\mathrm{t},\Pi} (C,((s_c',h_c),\sigma_o,\circ))}$$

$$\frac{\kappa = (s_l,x,C) \qquad [\![E]\!]_{s_l} \textit{ undefined}}{(\mathbf{E}[\,\textbf{return } E\,],((s_c,h_c),\sigma_o,\kappa)) \xrightarrow{(\mathrm{t},\textbf{obj},\textbf{abort})}_{\mathrm{t},\Pi} \textbf{abort}}$$

$$\frac{[\![E]\!]_{s_c} = n}{(\mathbf{E}[\,\textbf{print}(E)\,],((s_c,h_c),\sigma_o,\circ)) \xrightarrow{(\mathrm{t},\textbf{out},n)}_{\mathrm{t},\Pi} (\mathbf{E}[\,\textbf{skip}\,],((s_c,h_c),\sigma_o,\circ))}$$

$$\frac{(C,(s_o \uplus s_l,h_o)) \longrightarrow_{\mathrm{t}} (C',(s_o' \uplus s_l',h_o')) \qquad dom(s_l) = dom(s_l')}{(C,(\sigma_c,(s_o,h_o),(s_l,x,C_1))) \xrightarrow{(\mathrm{t},\textbf{obj})}_{\mathrm{t},\Pi} (C',(\sigma_c,(s_o',h_o'),(s_l',x,C_1)))} \qquad \frac{(C,\sigma_c) \longrightarrow_{\mathrm{t}} (C',\sigma_c')}{(C,(\sigma_c,\sigma_o,\circ)) \xrightarrow{(\mathrm{t},\textbf{clt})}_{\mathrm{t},\Pi} (C',(\sigma_c',\sigma_o,\circ))}$$

$$\frac{(C,(s_o \uplus s_l,h_o)) \longrightarrow_{\mathrm{t}} \textbf{abort}}{(C,(\sigma_c,(s_o,h_o),(s_l,x,C_1))) \xrightarrow{(\mathrm{t},\textbf{obj},\textbf{abort})}_{\mathrm{t},\Pi} \textbf{abort}} \qquad \frac{(C,\sigma_c) \longrightarrow_{\mathrm{t}} \textbf{abort}}{(C,(\sigma_c,\sigma_o,\circ)) \xrightarrow{(\mathrm{t},\textbf{clt},\textbf{abort})}_{\mathrm{t},\Pi} \textbf{abort}}$$

(b) thread transitions

$$\frac{[\![B]\!]_s = \textbf{true} \qquad (C,(s,h)) \longrightarrow_{\mathrm{t}}^{*} (\textbf{skip},(s',h'))}{(\mathbf{E}[\,\textbf{await}(B)\{C\}\,],(s,h)) \longrightarrow_{\mathrm{t}} (\mathbf{E}[\,\textbf{skip}\,],(s',h'))} \qquad \frac{[\![B]\!]_s = \textbf{true} \qquad (C,(s,h)) \longrightarrow_{\mathrm{t}}^{*} \textbf{abort}}{(\mathbf{E}[\,\textbf{await}(B)\{C\}\,],(s,h)) \longrightarrow_{\mathrm{t}} \textbf{abort}}$$

(c) local thread transitions

Fig. 4. Selected operational semantics rules.

respectively. The output event $(\mathrm{t},\textbf{out},n)$ is generated by the $\textbf{print}(E)$ command. $(\mathrm{t},\textbf{term})$ records the termination of the thread t. We also introduce a special event $(\textbf{spawn},n)$, which is inserted at the beginning of each event trace to record the creation of n threads at the beginning of the whole

program execution. Its use is shown in Sec. 5. An *event trace* $\mathcal{E}$ is a (possibly infinite) sequence of events, and a *program trace* T is a (possibly infinite) sequence of labels ι.

The sets Δ_c and Δ_o in the label record the IDs of threads that are blocked in the client code and object methods respectively. They are generated by the function btids defined in Fig. 3. Recall that a thread t is executing the client code if its call stack is empty, i.e., $\mathcal{K}(\mathrm{t}) = \circ$. We also define $\mathsf{en}(\hat{C})$ as the enabling condition for $\hat{C}$, which ensures that $\hat{C}$ can execute at least one step unless it has terminated. Here the execution context $\mathbf{E}$ defines the position of the command to be executed next.

The second rule in Fig. 4(a) shows that **end** is used as a flag marking the termination of a thread. A termination event $(\mathrm{t}, \mathbf{term})$ is generated correspondingly.

The first two rules in Fig. 4(b) show that method calls can only be executed in the client code (i.e., when the stack κ is empty), and it is the clients' responsibility to ensure that the precondition $\mathcal{P}$ (defined in Fig. 3) of the method holds over the object data. If $\mathcal{P}$ does not hold, the method invocation step aborts. Similarly, as shown in the subsequent rules, the **return** command can only be executed in the object method, and the **print** command can only be in the client code. Other commands can be executed either in the client or in the object, and the transitions are made over σ_c and σ_o respectively. In Fig. 4(c) we show the operational semantics for $\mathbf{await}(B)\{C\}$. Note that there is no transition rule when B is false, which means that the thread is blocked. Transition rules of other commands are standard and omitted here.

More discussions about partial methods. There are actually two reasons that make a method partial. The first is due to non-termination when the method is called under certain conditions. The second is due to abnormal termination, i.e., the method aborts or terminates with incorrect states or return values. Since the goal of this work is to study progress, the paper focuses on the first kind of partial methods. In our language, we specify the two kinds of partial methods differently. For the first kind, we use the enabling condition B in $\mathbf{await}(B)\{C\}$ to specify when the method should *not* be blocked. For the second kind, we use the annotated precondition $\mathcal{P}$ to specify the condition needed for the method to execute safely and to generate correct results. For instance, although the lock's release method `L_REL` in specification (2.3) always terminates, it needs an annotated precondition `l=cid` to prevent client threads not owning the lock from releasing it.

4 LINEARIZABILITY AND BASIC CONTEXTUAL REFINEMENT

In this section we formally define linearizability [Herlihy and Wing 1990] of an object Π with respect to its abstract specification Γ. As explained in Sec. 2.2, Γ is an *atomic partial specification* for Π. It has the same syntax with Π (see Fig. 2), but each method body in Γ is always an await block $\mathbf{await}(B)\{C\}$ (followed by a **return** E command). We also assume that the methods in Γ are safe, i.e., they never abort.

History events, externally observable events, and traces. We call (t, f, n), $(\mathrm{t}, \mathbf{ret}, n)$ and $(\mathrm{t}, \mathbf{obj}, \mathbf{abort})$ *history* events, and $(\mathrm{t}, \mathbf{out}, n)$, $(\mathrm{t}, \mathbf{obj}, \mathbf{abort})$ and $(\mathrm{t}, \mathbf{clt}, \mathbf{abort})$ *externally observable* events. In Fig. 5 we define $\mathcal{T}[\![W, \mathcal{S}]\!]$ as the *prefix closed* set of finite traces T generated during the execution of $(W, \mathcal{S})$. $\mathcal{H}[\![W, \mathcal{S}]\!]$ contains the set of histories projected from traces in $\mathcal{T}[\![W, \mathcal{S}]\!]$. Here $\mathsf{get\_hist}(T)$ returns a subsequence $\mathcal{E}$ consisting of the history events projected from the corresponding labels in T. Similarly $O[\![W, \mathcal{S}]\!]$ contains the set of externally observable event traces projected from traces in $\mathcal{T}[\![W, \mathcal{S}]\!]$, where $\mathsf{get\_obsv}(T)$ is a subsequence $\mathcal{E}$ consisting of externally observable events only.

As defined below, an event trace $\mathcal{E}$ is linearizable with respect to $\mathcal{E}'$, i.e., $\mathcal{E} \preceq^{\mathsf{lin}} \mathcal{E}'$, if they have the same sub-trace when projected over individual threads (projection represented as $\mathcal{E}|_{\mathrm{t}}$), and $\mathcal{E}$ is a permutation of $\mathcal{E}'$ but preserves the order of non-overlapping method calls in $\mathcal{E}'$. Here we use $\mathsf{is\_inv}(e)$ (or $\mathsf{is\_ret}(e_2)$) to represent that e is in the form of (t, f, n) (or $(\mathrm{t}, \mathbf{ret}, n)$).

$$\mathcal{T}[\![W,\mathcal{S}]\!] \stackrel{\text{def}}{=} \{T \mid \exists W',\mathcal{S}'.\ (W,\mathcal{S}) \overset{T}{\longmapsto}{}^{*} (W',\mathcal{S}') \vee (W,\mathcal{S}) \overset{T}{\longmapsto}{}^{*} \textbf{abort}\}$$

$$\mathcal{H}[\![W,\mathcal{S}]\!] \stackrel{\text{def}}{=} \{\mathcal{E} \mid \exists T.\ \mathcal{E} = \text{get\_hist}(T) \wedge T \in \mathcal{T}[\![W,\mathcal{S}]\!]\}$$

$$O[\![W,\mathcal{S}]\!] \stackrel{\text{def}}{=} \{\mathcal{E} \mid \exists T.\ \mathcal{E} = \text{get\_obsv}(T) \wedge T \in \mathcal{T}[\![W,\mathcal{S}]\!]\}$$

$$\text{match}(e_1,e_2) \stackrel{\text{def}}{=} \text{is\_inv}(e_1) \wedge \text{is\_ret}(e_2) \wedge (\text{tid}(e_1) = \text{tid}(e_2))$$

$$\frac{}{\text{seq}(\epsilon)} \qquad \frac{\text{is\_inv}(e)}{\text{seq}(e :: \epsilon)} \qquad \frac{\text{match}(e_1,e_2) \quad \text{seq}(\mathcal{E})}{\text{seq}(e_1 :: e_2 :: \mathcal{E})} \qquad \frac{\forall \text{t}.\ \text{seq}(\mathcal{E}|_{\text{t}})}{\text{well\_formed}(\mathcal{E})}$$

$$\frac{\text{well\_formed}(\mathcal{E})}{\mathcal{E} \in \text{extensions}(\mathcal{E})} \qquad \frac{\mathcal{E}' \in \text{extensions}(\mathcal{E}) \quad \text{is\_ret}(e) \quad \text{well\_formed}(\mathcal{E}'{+}{+}[e])}{\mathcal{E}'{+}{+}[e] \in \text{extensions}(\mathcal{E})}$$

$$\text{truncate}(\epsilon) \stackrel{\text{def}}{=} \epsilon \qquad \text{truncate}(e :: \mathcal{E}) \stackrel{\text{def}}{=} \begin{cases} e :: \text{truncate}(\mathcal{E}) & \text{if is\_ret}(e) \text{ or } \exists i.\ \text{match}(e,\mathcal{E}(i)) \\ \text{truncate}(\mathcal{E}) & \text{otherwise} \end{cases}$$

$$\text{completions}(\mathcal{E}) \stackrel{\text{def}}{=} \{\text{truncate}(\mathcal{E}') \mid \mathcal{E}' \in \text{extensions}(\mathcal{E})\}$$

$$\circledcirc \stackrel{\text{def}}{=} \{\text{t}_1 \rightsquigarrow \circ, \ldots, \text{t}_n \rightsquigarrow \circ\}$$

$$\Gamma \triangleright (\Sigma,\mathcal{E}) \text{ iff } \exists n, C_1, \ldots, C_n, \sigma_c.\ (\mathcal{E} \in \mathcal{H}[\![(\textbf{let } \Gamma \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \Sigma, \circledcirc)]\!]) \wedge \text{seq}(\mathcal{E})$$

Fig. 5. Auxiliary definitions for linearizability.

Definition 4.1 (Linearizable Histories). $\mathcal{E} \preceq^{\text{lin}} \mathcal{E}'$ iff both the following hold.

(1) $\forall \text{t}.\ \mathcal{E}|_{\text{t}} = \mathcal{E}'|_{\text{t}}$.

(2) There exists a bijection $\pi : \{1,\ldots,|\mathcal{E}|\} \to \{1,\ldots,|\mathcal{E}'|\}$ such that $\forall i.\ \mathcal{E}(i) = \mathcal{E}'(\pi(i))$ and
$$\forall i,j.\ i < j \wedge \text{is\_ret}(\mathcal{E}(i)) \wedge \text{is\_inv}(\mathcal{E}(j)) \implies \pi(i) < \pi(j)\,.$$

Definition 4.2 says Π is linearizable with respect to Γ and the state abstraction function φ (see Fig. 3) if, for any trace $\mathcal{E}$ generated by Π with the initial object data σ, the corresponding complete trace $\mathcal{E}_c$ is always linearizable with some *sequential* trace $\mathcal{E}'$ generated by Γ with initial object data Σ such that $\varphi(\sigma) = \Sigma$. Some of the key notations are defined in Fig. 5. We use $\circledcirc$ to represent the initial thread pool where each thread has an empty call stack. $\text{completions}(\mathcal{E})$ appends matching return events for some pending invocations in $\mathcal{E}$, and discards the other pending invocations, so that in the resulting trace every invocation has a matching return. We use ++ for list concatenation, and $[e_1,\ldots,e_n]$ for a list consisting of a sequence of elements. We use $\text{tid}(e)$ for the thread ID in e.

Definition 4.2 (Linearizability of Objects). The object implementation Π is linearizable with respect to Γ, written as $\Pi \preceq^{\text{lin}}_{\varphi} \Gamma$, iff

$$\forall n, C_1, \ldots, C_n, \sigma_c, \sigma, \Sigma, \mathcal{E}.\ \mathcal{E} \in \mathcal{H}[\![(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma, \circledcirc)]\!] \wedge (\varphi(\sigma) = \Sigma)$$
$$\implies \exists \mathcal{E}_c, \mathcal{E}'.\ (\mathcal{E}_c \in \text{completions}(\mathcal{E})) \wedge (\Gamma \triangleright (\Sigma,\mathcal{E}')) \wedge (\mathcal{E}_c \preceq^{\text{lin}} \mathcal{E}')$$

Abstraction of linearizable objects. Filipović et al. [Filipović et al. 2009] has shown that linearizability is equivalent to a contextual refinement. As defined below, Π contextually refines Γ under the state abstraction function φ if, for any clients $C_1 \ldots C_n$ as the execution context, and for any initial object data related by φ, executing Π generates no more externally event traces than executing Γ. Theorem 4.4 shows the equivalence between linearizability and the contextual refinement.

Definition 4.3 (Basic Contextual Refinement). $\Pi \sqsubseteq^{\text{fin}}_{\varphi} \Gamma$ iff

$$\forall n, C_1, \ldots, C_n, \sigma_c, \sigma, \Sigma.\ (\varphi(\sigma) = \Sigma) \implies$$
$$O[\![(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma, \circledcirc)]\!] \subseteq O[\![(\textbf{let } \Gamma \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \Sigma, \circledcirc)]\!]\,.$$

Theorem 4.4 (Basic Equivalence for Linearizability). $\Pi \preceq^{\mathsf{lin}}_{\varphi} \Gamma \iff \Pi \sqsubseteq^{\mathsf{fin}}_{\varphi} \Gamma$.

5 PROGRESS PROPERTIES

In this section we define the new progress properties, partial starvation-freedom (PSF) and partial deadlock-freedom (PDF), for objects with partial methods.

We first define the trace set $\mathcal{T}_\omega[\![W, \mathcal{S}]\!]$ in Fig. 6. It contains the (possibly infinite) *whole* execution traces T generated by $(W, \mathcal{S})$ but with a special label $((\mathbf{spawn}, |W|), \Delta_c, \Delta_o)$ inserted at the beginning. Here we use $|W|$ to represent the number of threads in W. The event $(\mathbf{spawn}, |W|)$ is used to define fairness, as explained below. Δ_c and Δ_o records the threads blocked in clients and object methods respectively (see the definition of btids in Fig. 3). At the beginning of an execution, Δ_o must be an empty set since no threads are in method calls. The whole execution trace T may be generated under three cases, i.e., the execution of $(W, \mathcal{S})$ diverges, aborts or gets stuck (terminates or is blocked). We write $(W, \mathcal{S}) \overset{T}{\longmapsto}^{\omega} \cdot$ for an infinite execution. In this case, the length of T must be infinite, written as $|T| = \omega$.

Strong and weak fairness. As defined in Fig. 6, a trace T is strongly fair, represented as $\mathsf{sfair}(T)$, if each thread either terminates, or is executed infinitely many times if it is *infinitely often* enabled (i-o-enabled). We know a thread is enabled if it is not in the blocked sets Δ_c and Δ_o. $T(j)$ represents the j-th element in the trace T. Similarly, $\mathsf{wfair}(T)$ says that T is a weakly fair trace. It requires that each thread either terminates, or is executed infinitely many times if it is *always* enabled after certain step on the trace (e-a-enabled).

Thread progress and program progress. We use prog-t(T) in Fig. 6 to say that every method call eventually terminates. It ensures that each individual thread calling a method eventually returns. prog-p(T) says that there is always at least one method call that terminates. It ensures that the whole program is making progress. Here $\mathsf{pend\_inv}(T)$ represents the set of method invocation events that do not have matching returns. $T(1..i)$ represents the prefix of T with length i.

Partial starvation-freedom (PSF) and partial deadlock-freedom (PDF). We want to define PSF as a generalization of starvation-freedom. We say an object Π is *partially starvation-free* if, under fair scheduling (with strong or weak fairness), each method call eventually returns (as required in starvation-freedom), unless it is eventually always disabled (i.e., it is not supposed to return in this particular execution context). In the latter case the non-termination is caused by inappropriate invocations of the methods in the client code and the object implementation should *not* be blamed.

Although the idea is intuitive, it is challenging to formalize it. This is because when we say a method is disabled we are thinking at an abstract level, where the abstract disabling condition cannot be syntactically inferred based on the low-level object implementation Π. For instance, the lock implementations in Fig. 1 use non-blocking commands only, so they are always enabled to execute one more step at this level, although we intend to say at a more abstract level that the `L_acq()` operation is disabled when the lock is unavailable.

To address this problem, we refer to the abstract object specification Γ when defining the progress of a concrete object Π. Recall that method specifications in Γ are in the form of $\mathbf{await}(B)\{C\}$, so we know that the method is disabled when B does not hold.

We formalize the idea as Def. 5.1. Under the scheduling fairness χ (where $\chi \in \{\mathsf{sfair}, \mathsf{wfair}\}$, as defined in Fig. 6), we say the object Π is PSF with respect to an abstract specification Γ and a state abstraction function φ, i.e., $\mathsf{PSF}^{\chi}_{\varphi,\Gamma}(\Pi)$, if any χ-fair trace T generated by $((\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n), (\sigma_c, \sigma, \circledcirc))$ either aborts, or satisfies prog-t, or we could blame the client for the blocking of each pending invocation.

$$\mathcal{T}_\omega[\![W,\mathcal{S}]\!] \stackrel{\text{def}}{=} \{((\mathbf{spawn},|W|),\Delta_c,\Delta_o)::T \mid \text{btids}(W,\mathcal{S}) = (\Delta_c,\Delta_o) \wedge (((W,\mathcal{S}) \xrightarrow{T}{}^{\omega} \cdot) \vee ((W,\mathcal{S}) \xrightarrow{T}{}^{*} \mathbf{abort}) \vee \exists W',\mathcal{S}'.\ ((W,\mathcal{S}) \xrightarrow{T}{}^{*} (W',\mathcal{S}')) \wedge \neg(\exists \iota.\ (W',\mathcal{S}') \xrightarrow{\iota} \_)) \}$$

$$|\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n| \stackrel{\text{def}}{=} n \qquad \text{tnum}(((\mathbf{spawn},n),\Delta_c,\Delta_o)::T) \stackrel{\text{def}}{=} n$$

$$\text{evt}(\iota) \stackrel{\text{def}}{=} e \quad \text{if } \iota = (e,\Delta_c,\Delta_o) \qquad \text{bset}(\iota) \stackrel{\text{def}}{=} \Delta_c \cup \Delta_o \quad \text{if } \iota = (e,\Delta_c,\Delta_o)$$

$$\text{i-o-enabled}(\mathrm{t},T) \ \text{ iff } \ \forall i.\ \exists j \geq i.\ \mathrm{t} \notin \text{bset}(T(j)) \qquad \text{“infinitely often”}$$

$$\text{e-a-enabled}(\mathrm{t},T) \ \text{ iff } \ \exists i.\ \forall j \geq i.\ \mathrm{t} \notin \text{bset}(T(j)) \qquad \text{“eventually always”}$$

$$\text{sfair}(T) \ \text{ iff } \ |T| = \omega \implies \forall \mathrm{t} \in [1..\text{tnum}(T)].\ \text{evt}(\text{last}(T|_{\mathrm{t}})) = (\mathrm{t},\mathbf{term}) \vee (\text{i-o-enabled}(\mathrm{t},T) \implies |(T|_{\mathrm{t}})| = \omega)$$

$$\text{wfair}(T) \ \text{ iff } \ |T| = \omega \implies \forall \mathrm{t} \in [1..\text{tnum}(T)].\ \text{evt}(\text{last}(T|_{\mathrm{t}})) = (\mathrm{t},\mathbf{term}) \vee (\text{e-a-enabled}(\mathrm{t},T) \implies |(T|_{\mathrm{t}})| = \omega)$$

$$\text{pend\_inv}(T) \stackrel{\text{def}}{=} \{e \mid \exists i.\ e = \text{evt}(T(i)) \wedge \text{is\_inv}(e) \wedge \neg\exists j > i.\ \text{match}(e,\text{evt}(T(j))) \}$$

$$\text{abt}(T) \ \text{ iff } \ \exists i.\ \text{is\_abt}(\text{evt}(T(i)))$$

$$\text{prog-t}(T) \ \text{ iff } \ \text{pend\_inv}(T) = \emptyset$$

$$\text{prog-p}(T) \ \text{ iff } \ \forall i,e.\ e \in \text{pend\_inv}(T(1..i)) \implies \exists j > i.\ \text{is\_ret}(\text{evt}(T(j)))$$

$$\text{e-a-disabled}(\mathrm{t},T) \ \text{ iff } \ \exists i.\ \forall j \geq i,\iota = T(j).\ \mathrm{t} \in \text{bset}(\iota) \qquad \text{“eventually always”}$$

$$\text{well-blocked}(T,(W_a,\mathcal{S}_a)) \ \text{ iff } \ \exists T_a.\ T_a \in \mathcal{T}_\omega[\![W_a,\mathcal{S}_a]\!] \wedge (\text{get\_hist}(T) = \text{get\_hist}(T_a)) \wedge (\forall e.\ e \in \text{pend\_inv}(T_a) \implies \text{e-a-disabled}(\text{tid}(e),T_a))$$

$$\mathcal{O}_\chi[\![W,(\sigma_c,\sigma_o)]\!] \stackrel{\text{def}}{=} \{\mathcal{E} \mid \exists T.\ T \in \mathcal{T}_\omega[\![W,(\sigma_c,\sigma_o,\circledcirc)]\!] \wedge \chi(T) \wedge \text{get\_obsv}(T) = \mathcal{E}\} \qquad \chi \in \{\text{sfair},\text{wfair}\}$$

Fig. 6. Fairness and progress.

In the last case, we must be able to find a trace T_a generated by the execution of the abstract object Γ (with the abstract object state Σ related to σ by φ) such that it has the *same* method invocation and return history with T, and every pending invocation in this abstract trace T_a is eventually always disabled. See the definition of well-blocked in Fig. 6 for the formal details.

Definition 5.1 (Partially Starvation-Free Objects). $\text{PSF}^{\chi}_{\varphi,\Gamma}(\Pi)$ iff

$$\forall n,C_1,\ldots,C_n,\sigma_c,\sigma,\Sigma,T.\ T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n),(\sigma_c,\sigma,\circledcirc)]\!] \wedge (\varphi(\sigma) = \Sigma) \wedge \chi(T) \implies \text{abt}(T) \vee \text{prog-t}(T) \vee \text{well-blocked}(T,((\mathbf{let}\ \Gamma\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n),(\sigma_c,\Sigma,\circledcirc)))\,.$$

We also define PDF in Def. 5.2. It is similar to PSF, but requires prog-p instead of prog-t.

Definition 5.2 (Partially Deadlock-Free Objects). $\text{PDF}^{\chi}_{\varphi,\Gamma}(\Pi)$ iff

$$\forall n,C_1,\ldots,C_n,\sigma_c,\sigma,\Sigma,T.\ T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n),(\sigma_c,\sigma,\circledcirc)]\!] \wedge (\varphi(\sigma) = \Sigma) \wedge \chi(T) \implies \text{abt}(T) \vee \text{prog-p}(T) \vee \text{well-blocked}(T,((\mathbf{let}\ \Gamma\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n),(\sigma_c,\Sigma,\circledcirc)))\,.$$

The above definitions consider the three factors that may affect the termination of a method call: the scheduling fairness χ, the object implementation Π which determines whether its traces satisfy prog-t or prog-p, and the execution context $C_1 \parallel \ldots \parallel C_n$ which may make inappropriate method invocations so that well-blocked holds. Consider the lock objects in Fig. 1(a) and (c) and

the following client program (5.1). The initial value of the shared variable x is 0.

```
[_]ACQ; print(0); [_]REL;      ||  [_]ACQ; print(2);
x:=1;                          ||  while(x=1){
[_]ACQ; print(1); [_]REL;      ||    [_]REL; [_]ACQ; print(3); }
```
(5.1)

The client can produce a trace satisfying prog-t when it uses the ticket lock. It first executes the left thread until termination and then executes the right thread. Then every method call terminates, printing out 0, 1, 2 and an infinite number of 3. Thus prog-t holds. When the test-and-set lock is used instead, the same client can produce a trace satisfying prog-p but not prog-t. In the execution, the second call to `L_acq` in the left thread never finishes. It prints out 0, 2 and an infinite number of 3, but not 1. Such an execution is not possible when the client uses the ticket lock, under fair scheduling. This shows how different object implementations affect termination of method calls. Note that neither of the two execution traces satisfies well-blocked, because every method call in the traces either terminates or is enabled infinitely often.

On the other hand, the client (5.1) can produce a well-blocked trace no matter it uses the ticket lock or the test-and-set lock. It executes the right thread first until termination and then executes the left thread. Then the first call to lock acquire of the left thread is always blocked, and only 2 is printed during the execution. The non-termination of the method call is caused by the particular execution context, in which the method call is not supposed to return, regardless of the object implementations. This is why the same well-blocked condition is used in both definitions of PSF and PDF for both strongly and weakly fair executions of the object implementation.

PSF (or PDF) and starvation-freedom (or deadlock-freedom) coincide if we require each **await** block in Γ is in the special form of **await**(true)$\{C\}$ — Since the methods in Γ are always enabled, well-blocked$(T, ((\textbf{let}\ \Gamma\ \textbf{in}\ C_1 \parallel \ldots \parallel C_n), \mathcal{S}_a))$ now requires that there is no pending invocation in T. This is stronger than both prog-t(T) and prog-p(T). Therefore we can remove the disjunction branch about well-blocked in Defs. 5.1 and 5.2, resulting in definitions equivalent to starvation-freedom and deadlock-freedom respectively.

6 PROGRESS-AWARE ABSTRACTION OF OBJECTS

In this section we study the abstraction of linearizable and PSF (or PDF) objects. Similar to Theorem 4.4, we want theorems showing that linearizability along with PSF (or PDF) of an object Π is equivalent to a contextual refinement between Π and some abstract object Π', where Π' can be syntactically derived from the atomic specification Γ.

We first define the progress-aware contextual refinement for objects under different fairness χ of scheduling ($\chi \in \{\text{sfair}, \text{wfair}\}$). As Def. 6.1 shows, Π contextually refines Π' under the χ-fair scheduling if, in any execution context, Π generates no more externally observable event traces than Π'. The set of event traces $O_\chi[\![(\textbf{let}\ \Pi\ \textbf{in}\ C_1 \parallel \ldots \parallel C_n), (\sigma_c, \sigma)]\!]$ is defined in Fig. 6, where each event trace $\mathcal{E}$ is extracted from the χ-fair trace T in $\mathcal{T}_\omega[\![(\textbf{let}\ \Pi\ \textbf{in}\ C_1 \parallel \ldots \parallel C_n), (\sigma_c, \sigma, \circledcirc)]\!]$. The refinement is *progress-aware* because we use the whole execution trace T here, from which we can tell whether the corresponding execution terminates or not.

Definition 6.1 (Progress-Aware Contextual Refinement).

$$\Pi \sqsubseteq^{\chi}_{\varphi} \Pi' \ \text{ iff } \ \forall n, C_1, \ldots, C_n, \sigma_c, \sigma, \Sigma.\ \varphi(\sigma) = \Sigma \implies O_\chi[\![(\textbf{let}\ \Pi\ \textbf{in}\ C_1 \parallel \ldots \parallel C_n), (\sigma_c, \sigma)]\!] \subseteq O_\chi[\![(\textbf{let}\ \Pi'\ \textbf{in}\ C_1 \parallel \ldots \parallel C_n), (\sigma_c, \Sigma)]\!]$$

Wrappers for atomic specifications. As explained in Sec. 2, one of the major contributions of this paper is to define wrappers for atomic partial specifications Γ, which transform the method specification **await**$(B)\{C\}$ in Γ into a (possibly non-atomic) abstract specification for each combination of progress (PSF or PDF) and fairness (sfair or wfair), as shown in Table 2.

$$\mathsf{wr}^{\chi}_{\mathsf{Prog}}(\Gamma)(f) \stackrel{\text{def}}{=} (\mathcal{P}, x, \mathsf{wr}^{\chi}_{\mathsf{Prog}}(\mathbf{await}(B)\{C\}); \mathbf{return}\ E) \quad \text{if } \Gamma(f) = (\mathcal{P}, x, \mathbf{await}(B)\{C\}; \mathbf{return}\ E)$$

$$\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PSF}}(\mathbf{await}(B)\{C\}) \stackrel{\text{def}}{=} \mathbf{await}(B)\{C\}$$

$$\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\mathbf{await}(B)\{C\}) \stackrel{\text{def}}{=} \texttt{listid := listid++[(cid,'}B\texttt{')]}; \ \mathbf{await}(B \wedge \texttt{cid} = \mathbf{enhd}(\texttt{listid}))\{C;\ \texttt{listid := listid\cid;}\}$$

$$\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\mathbf{await}(B)\{C\}) \stackrel{\text{def}}{=} \mathbf{while}\ (\mathsf{done})\{\}; \ \mathbf{await}(B \wedge \neg\mathsf{done})\{C; \mathsf{done} := \mathsf{true};\};\ \ \mathsf{done} := \mathsf{false}; \ \mathbf{while}\ (\mathsf{done})\{\};$$

$$\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PDF}}(\mathbf{await}(B)\{C\}) \stackrel{\text{def}}{=} \mathbf{await}(B \wedge \neg\mathsf{done})\{C; \mathsf{done} := \mathsf{true};\};\ \ \mathsf{done} := \mathsf{false}; \ \mathbf{await}(\neg\mathsf{done})\{\}$$

$$\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\varphi)(\sigma) \stackrel{\text{def}}{=} \begin{cases} \sigma' \uplus \{\texttt{listid} \rightsquigarrow \epsilon\} & \text{if } \varphi(\sigma) = \sigma' \\ \textit{undefined} & \text{if } \sigma \notin \mathit{dom}(\varphi) \end{cases} \qquad \mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PSF}}(\varphi) \stackrel{\text{def}}{=} \varphi$$

$$\mathsf{wr}^{\chi}_{\mathsf{PDF}}(\varphi)(\sigma) \stackrel{\text{def}}{=} \begin{cases} \sigma' \uplus \{\mathsf{done} \rightsquigarrow \mathbf{false}\} & \text{if } \varphi(\sigma) = \sigma' \\ \textit{undefined} & \text{if } \sigma \notin \mathit{dom}(\varphi) \end{cases}$$

Fig. 7. Definition of wrappers.

Before introducing the definition of the wrappers in Fig. 7, we first show our abstraction theorem (Theorem 6.2) for linearizable and PSF (or PDF) objects. It establishes the equivalence between the progress-aware contextual refinement and linearizability with PSF (or PDF).

Theorem 6.2 (Abstraction Theorem). *Let* $\mathsf{Prog} \in \{\mathsf{PSF}, \mathsf{PDF}\}$ *and* $\chi \in \{\mathsf{sfair}, \mathsf{wfair}\}$*, then*

$$\Pi \preceq^{\mathsf{lin}}_{\varphi} \Gamma \wedge \mathsf{Prog}^{\chi}_{\varphi,\Gamma}(\Pi) \iff \Pi \sqsubseteq^{\chi}_{\widehat{\varphi}} \mathsf{wr}^{\chi}_{\mathsf{Prog}}(\Gamma)\ ,$$

where $\widehat{\varphi} = \mathsf{wr}^{\chi}_{\mathsf{Prog}}(\varphi)$*, and the wrappers for* Γ *and* φ *are defined in Fig. 7. We also assume that the variables* `listid` *and* done *introduced in the wrapper code are fresh, i.e.,* $\texttt{listid}, \mathsf{done} \notin \mathsf{FV}(\{\Pi, \Gamma, \varphi\})$.

To prove the theorem, we define compositional operational semantics that can generate separate traces for objects and clients, and build simulations using the object semantics. Detailed proofs are given in Appendix A.

Next we introduce the definition of the wrappers in detail. The wrapper $\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PSF}}$ is simply an identity function. It maps the atomic specification $\mathbf{await}(B)\{C\}$ to itself. This is because under *strongly fair scheduling* $\mathbf{await}(B)\{C\}$ will eventually be executed unless it is eventually always disabled. This is exactly what we need for PSF of linearizable objects, which requires that the invocation of each method eventually returns, unless the corresponding high-level atomic operation $\mathbf{await}(B)\{C\}$ is eventually always disabled (as specified by well-blocked in Fig. 6).

Wrapper for PSF under weakly fair scheduling. Under weakly fair scheduling, however, we cannot guarantee that $\mathbf{await}(B)\{C\}$ is eventually executed even if B holds infinitely often. Therefore it alone cannot satisfy PSF. That's why we define $\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\mathbf{await}(B)\{C\})$, which guarantees that the atomic operation is eventually executed if B holds infinitely often. We introduce a blocking queue `listid` in the object state, which is a sequence of $(\mathsf{t}, `B')$ pairs, showing that the thread t requests to execute an atomic operation with the enabling condition $B$. Note that the enabling condition $B$ is recorded *syntactically* in `listid`, represented as $`B'$. The operator $\mathbf{enhd}(\texttt{listid})$ returns the first

thread on the list whose enabling condition is true. It evaluates the syntactic enabling conditions 'B' recorded in `listid` on the fly. Note that different pairs in `listid` may have different enabling conditions B. In the code generated by $\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\textbf{await}(B)\{C\})$, we first append the current thread ID and the enabling condition 'B' at the end of the list. In the subsequent command the thread waits until both B holds and $\texttt{cid} = \textbf{enhd}(\texttt{listid})$[1]. Then it atomically executes C and deletes the current thread in the queue.

This wrapper guarantees that C is eventually executed when B becomes infinitely often true because we know $B \wedge \texttt{cid} = \textbf{enhd}(\texttt{listid})$ will be *eventually always* true, and then the weakly fair scheduling guarantees the execution of C. This is because, whenever B becomes true, either $\texttt{cid} = \textbf{enhd}(\texttt{listid})$ holds or there is a pair $(\texttt{t}', B')$ such that $B' \wedge \texttt{t}' = \textbf{enhd}(\texttt{listid})$ holds. In the first case, other threads trying to execute the object methods must be blocked at the **await** command. Therefore B cannot be changed to false by other threads. Therefore $B \wedge \texttt{cid} = \textbf{enhd}(\texttt{listid})$ is always true until the current thread executes the atomic block. In the second case $\texttt{t}'$ must be able to finish its method, following the same argument above. Therefore there will be one less thread waiting in front of the current thread `cid`. Since B becomes true infinitely often, we eventually reach the first case.

As a result, the wrapper does not terminate in a weakly fair execution only if B is eventually always false. In that case the execution trace is well-blocked (see Fig. 6), still satisfying PSF.

One may argue that the abstraction generated by the wrapper is not very useful because it may not be much simpler than the object implementation. For instance, if we consider the `acquire` method of locks, the abstraction is almost the same as queue locks or ticket locks. But we want to emphasize that our wrapper is a general one that works for any object method implementation with an atomic specification in the form of $\textbf{await}(B)\{C\}$. Therefore we know the method's progress-aware abstraction can always be in this form, no matter how complex its implementation is.

Wrapper for PDF under weakly fair scheduling. For the right column in Table 2, we first introduce the wrapper at the bottom right corner. The definition of PDF says a method can be non-terminating if (1) it is eventually always disabled, as specified by well-blocked (see Fig. 6); or (2) there are always other method calls terminating, as specified by prog-p. Note that the second condition allows the method to be non-terminating even if it is eventually always enabled under weakly fair scheduling. As an example, the Treiber stack with a partial pop in Fig. 8 demonstrates one such scenario. The pop method is blocked when the stack is empty. It is linearizable with respect to the following specification

$$\textbf{await}(\texttt{S} \neq \texttt{nil})\{\texttt{tmp} := \texttt{head(S)}; \texttt{S} := \texttt{tail(S)};\}; \quad \textbf{return}\ \texttt{tmp}; \tag{6.1}$$

where `S` is the abstraction of the stack and `tmp` is a thread-local temporary variable.

In the following execution context (6.2),

```
pop();    ||    while(true){ push(0); }
```
(6.2)

the call of the concrete method pop may never terminate because its **cas** command may always fail, although the enabling condition at the abstract level ($\texttt{S} \neq \texttt{nil}$) is eventually always true. However, if we replace the method implementation with the specification (6.1), pop must terminate under weakly fair scheduling. This shows that the concrete implementation cannot contextually refine this simple specification (6.1).

Our first attempt to address this problem is to introduce a new object variable done (initialized to `false`), and let the wrapper $\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PDF}}$ transform $\textbf{await}(B)\{C\}$ into:

$$\textbf{await}(B \wedge \neg\texttt{done})\{C;\ \texttt{done} := \texttt{true}\}; \quad \texttt{done} := \texttt{false}; \tag{6.3}$$

[1] Actually the conjunct B in the **await** condition in the wrapper could be omitted, because B must be true when $\texttt{cid} = \textbf{enhd}(\texttt{listid})$ holds.

```
initialize(){  Top := null; }

push(v){
 1  local x, b, t;
 2  b := false;
 3  x := cons(v, null);
 4  while (!b) {
 5    t := Top;
 6    x.next := t;
 7    b := cas(&Top, t, x);
 8  }
}
```

```
pop(){
 9  local x, b, t, v;
10  b := false;
11  while (!b) {
12    t := Top;
13    if (t != null) {
14      v := t.data;
15      x := t.next;
16      b := cas(&Top, t, x);
17    }
18  }
19  return v;
}
```

```
initialize'(){  initialize(); done := false;}
push'(v){ push(v); DLY_NOOP}        pop'(v){ tmp := pop(); DLY_NOOP; return tmp}
DLY_NOOP ≝ await(¬done){done := true}; done := false;

r0 := pop'();
print(r0);
```

```
push'(1);
push'(2);
r1 := pop'();
print(r1);
while(true){ push'(0) };
```

Fig. 8. Treiber stacks with partial pops.

Therefore the resulting **await** command may not be executed even if B is always true, because done can be set to `true` infinitely often when other threads finish the atomic block. Also note done is reset to `false` at the end of each **await** command, therefore the condition ¬done cannot always disable the **await** command, which may cause deadlock. As a result, there is always some thread that can finish the wrapper (i.e., prog-p holds) unless the B-s of all the pending invocations are eventually always false (i.e., well-blocked holds), thus PDF holds.

However, this is not the end of the story. If the code (6.3) fails to terminate, C must not be executed and no effects (over the object data) are generated. However, it is possible for PDF methods to finish C and make the effects visible to other threads but fail to terminate. As an example we define the `push'` and `pop'` methods in Fig. 8 as a new implementation of the Treiber stack. They call the push and pop methods respectively and then execute the code snippet `DLY_NOOP` before they return. `DLY_NOOP` simply waits until done becomes `false` and then atomically sets it to `true`, and finally resets it to `false`. The only purpose of `DLY_NOOP` is to allow the methods to be delayed by other threads or to delay others.

Then we consider the client code shown at the bottom of Fig. 8. Under weakly fair scheduling it is possible that the call of `pop'()` by the left thread never terminates but the thread on the right prints out 1. That is, although the `pop'()` on the left does not terminate, it does generate effects over the stack and the effects happen before the `pop'()` on the right. Such an external event trace cannot be generated if we replace the concrete `push'()` and `pop'()` methods with the abstract method code generated using the wrapper (6.3) defined above. Thus the contextual refinement between the concrete code and the wrapped specification does not hold.

Our solution is to append an **await** command at the end of (6.3), so that the resulting code $\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PDF}}(\mathbf{await}(B)\{C\})$ (see Fig. 7) may finish C but still be blocked at the end.

Wrapper for PDF under strongly fair scheduling. Much of the effort to define $\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PDF}}(\mathbf{await}(B)\{C\})$ is to allow the resulting code to be non-terminating even if B is eventually always true. We need to do the same to define $\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\mathbf{await}(B)\{C\})$, but it is more challenging with *strongly fair scheduling* because $\mathbf{await}(\neg\mathsf{done})\{\}$ cannot be blocked under strongly fair scheduling if done is infinitely often true. Therefore we use **while**-loops in $\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\mathbf{await}(B)\{C\})$ to allow the method to be delayed when done is infinitely often true[2]. Note that **while** (done)$\{\}$ terminates when done is false.

Wrappers for the state abstraction function. Since the program transformations by the wrappers introduce new object variables such as `listid` and done, we need to change the state abstraction function φ accordingly, which is defined as $\mathsf{wr}^{\chi}_{\mathsf{Prog}}(\varphi)$ in Fig. 7 ($\chi \in \{\mathsf{sfair}, \mathsf{wfair}\}$ and $\mathsf{Prog} \in \{\mathsf{PSF}, \mathsf{PDF}\}$).

More discussions. There could be different ways to define the wrappers to validate the Abstraction Theorem 6.2. We do not intend to claim that our definitions are the simplest ones (and it is unclear how to formally compare the complexity of different wrappers), but we would like to point out that, although some of the wrappers look complex, the complexity is partly due to the effort to have general wrappers that work for any atomic specifications in the form of $\mathbf{await}(B)\{C\}$. It is possible to have simpler wrappers for specific objects. For instance, the lock specification Γ in (2.3) defined in Sec. 2.2 can already serve as an abstraction for the test-and-set lock object Π_{TAS} (which is a PDF lock) under weakly fair scheduling, i.e., $\Pi_{\mathsf{TAS}} \sqsubseteq^{\mathsf{wfair}}_{\varphi} \Gamma$ holds.

7 PROGRAM LOGIC

We extend the program logic LiLi [Liang and Feng 2016] to verify progress properties of concurrent objects with partial methods. LiLi is a rely-guarantee style program logic to verify linearizability and starvation-freedom/deadlock-freedom of concurrent objects. It establishes progress-aware contextual refinements between concrete object implementations and abstract (total) specifications.

The key ideas of LiLi to verify progress are the following:

- A thread can be blocked, relying on the actions of other threads (i.e., its environment) to make progress. To ensure it eventually progresses, we must guarantee that the environment actions that the thread waits for eventually occur.
- To avoid circularity in rely-guarantee reasoning, each thread specifies a set of *definite actions* $\mathcal{D}$, which are state transitions specified in the form of $P \rightsquigarrow Q$. The thread guarantees that, whenever a definite action $P \rightsquigarrow Q$ is "enabled" (i.e., the assertion P holds), the transition must occur so that Q eventually holds, regardless of the environment behaviors.
- A blocked thread must wait for a set of definite actions of other threads, and the size of the set must be decreasing (so that the thread is eventually unblocked).
- A thread may delay the progress of others, i.e., to make other threads to execute more steps than they need when executed in isolation. To ensure deadlock-freedom, LiLi disallows a thread to be delayed infinitely often without whole system progress. This is achieved by using tokens as resources and each delaying action must consume a token.

These ideas to reason about blocking and delay are general enough for verifying objects with partial methods, but we have to first generalize LiLi in the following two aspects:

[2]Actually the conjunct ¬done in the **await** condition in the wrapper could be removed, because the loop **while** (done)$\{\}$ before the **await** block can already produce the non-terminating behaviors when other threads finish the method infinitely often (i.e., done is infinitely often true). Here we keep the conjunct ¬done to make the wrapper more intuitive.

$$
\begin{array}{rrcl}
(\mathit{RelAssn}) & P,Q,J & ::= & B \mid \mathsf{emp} \mid E \mapsto E \mid E \Mapsto E \mid [\![p]\!] \mid P * Q \mid P \wedge Q \mid P \vee Q \mid \ldots \\
(\mathit{FullAssn}) & p,q & ::= & P \mid \mathsf{arem}(C) \mid \Diamond(E) \mid \blacklozenge(E_k,\ldots,E_1) \mid p * q \mid p \wedge q \mid \ldots \\
(\mathit{RelAct}) & R,G & ::= & P \ltimes_k Q \mid [P] \mid \lfloor G \rfloor_0 \mid \mathcal{D} \mid G \wedge G \mid G \vee G \mid \ldots \\
(\mathit{DAct}) & \mathcal{D} & ::= & P \rightsquigarrow Q \mid \forall x.\, \mathcal{D} \mid \mathcal{D} \wedge \mathcal{D}
\end{array}
$$

$$
\mathfrak{S} ::= (\sigma,\Sigma) \qquad u ::= (n_k,\ldots,n_1) \qquad w \in \mathit{Nat}
$$

$$
\begin{array}{ll}
\mathsf{Enabled}(P \rightsquigarrow Q) \stackrel{\text{def}}{=} P & \langle \mathcal{D} \rangle \stackrel{\text{def}}{=} \mathcal{D} \wedge (\mathsf{Enabled}(\mathcal{D}) \ltimes \mathsf{true}) \\
\mathsf{Enabled}(\forall x.\, \mathcal{D}) \stackrel{\text{def}}{=} \exists x.\, \mathsf{Enabled}(\mathcal{D}) & \mathcal{D}' \preccurlyeq \mathcal{D} \text{ iff } (\mathsf{Enabled}(\mathcal{D}') \Rightarrow \mathsf{Enabled}(\mathcal{D})) \\
\mathsf{Enabled}(\mathcal{D}_1 \wedge \mathcal{D}_2) \stackrel{\text{def}}{=} \mathsf{Enabled}(\mathcal{D}_1) \vee \mathsf{Enabled}(\mathcal{D}_2) & \qquad\qquad \wedge\, (\mathcal{D} \Rightarrow \mathcal{D}')
\end{array}
$$

Fig. 9. Assertions and models.

- LiLi does not have **await** commands. There **while**-loops are the only commands that affect progress. Now we have to reason about **await** in object code, which may affect progress as well. It is interesting to see that **await** can be reasoned about similarly as **while**-loops.
- We also have $\mathbf{await}(B)\{C\}$ as partial specifications. Since we want *termination-preserving* refinement, we do not have to guarantee progress of the concrete object methods when the partial specification is disabled.

As an extension of LiLi, our logic borrows LiLi's key ideas and most of the logic rules. Due to the space limit, the full details of the logic are given in Appendix B. Below we only show the major extensions.

7.1 Assertions

The assertions, shown in Fig. 9, are the same as those in LiLi. P and Q are assertions over relational states $\mathfrak{S}$, which are pairs of concrete and abstract object states (see Fig. 9). We use relational assertions because our logic establishes refinement between the concrete object implementation and the abstract specification. As in separation logic, we use $E \mapsto E$ and $E \Mapsto E$ to specify memory cells at the concrete and abstract level respectively. emp specifies empty states, and the separating conjunction $P * Q$ specifies two disjoint parts of $\mathfrak{S}$, which satisfy P and Q respectively.

The full assertions p and q specify triples in the form of $(\mathfrak{S},(u,w),C)$. In addition to the relational state $\mathfrak{S}$, the assertions also describe the numbers (u,w) of available tokens, and the high-level (abstract) method code C that remains to be refined by the low-level (concrete) code. The assertion $\mathsf{arem}(C)$ specifies such high-level code in the triple.

Tokens and multi-level delaying actions. As explained above, LiLi uses tokens as resources to prevent infinitely many execution steps and infinitely many delaying actions. It requires that each round of a **while**-loop consumes a $\Diamond$-token, and each delaying action consumes a $\blacklozenge$-token. LiLi also stratify delaying actions into multiple levels. The delay caused by high-level actions may lead to executions of more low-level ones (and non-delaying actions), but *not* vice versa. Therefore we use w to represent the number of $\Diamond$-tokens, and use the vector u for the numbers of the k-level $\blacklozenge$-tokens, as defined in Fig. 9. They are described by the assertions $\Diamond(E)$ and $\blacklozenge(E_k,\ldots,E_1)$ respectively.

Rely/guarantee conditions and definite actions. The rely/guarantee conditions R and G specify stratified transitions between relational states, i.e., they are assertions over $(\mathfrak{S},\mathfrak{S}',k)$, where k is the level of stratified delaying actions. $P \ltimes_k Q$ specifies a level-k delaying transition from P to Q. The subscript k can be omitted when $k = 0$. $[P]$ specifies identity transitions with the initial states satisfying P. $\lfloor G \rfloor_0$ can be satisfied only by level-0 transitions in G. Definite actions $\mathcal{D}$ is a special form of rely/guarantee condition. $P \rightsquigarrow Q$ specifies a transition where the final state satisfies Q if the initial state satisfies P.

As in LiLi, all assertions are implicitly parameterized with a thread ID t.

7.2 Logic Rules

Figure 10 shows the key logic rules. As the top rule of the logic, the OBJ rule says that, to verify Π satisfies its specification Γ with the object invariant P, one needs to specify the rely/guarantee conditions R and G, and the definite actions $\mathcal{D}$, and then prove that each individual object method implementation refines its specification. Here Γ must be an *atomic partial specification*.

For each method, we take the object invariant P and the annotated preconditions $\mathcal{P}$ and $\mathcal{P}'$ (in Π and Γ) as preconditions. We also assign $\blacklozenge$-tokens in the precondition ($\blacklozenge(E_k,\ldots,E_1)$) to constrain the number of delaying actions executed in the method. $\mathsf{arem}(C')$ says that the high-level code which remains to be refined at this point is C'. At the end we need to re-establish the object invariant P, and show that there is no more high-level code that needs to be refined (i.e., $\mathsf{arem}(\textbf{skip})$), which means the method body indeed refines the specification C'.

The object invariant P should also ensure that the annotated pre-conditions $\mathcal{P}$ and $\mathcal{P}'$ are either both true or both false. That is, whenever P holds, it is either safe to call the methods at both the concrete and abstract levels, or unsafe to do so at both levels. The other side conditions in the rule are the same as those in LiLi and irrelevant to our extensions, so we omit the explanations here.

The WHL rule. The rule for **while**-loops is almost the same as the WHL rule in LiLi, with the changes highlighted in gray boxes. We verify the loop body with a precondition p', which needs to be derived from the loop invariant p and the loop condition B. In two cases we must ensure that there are no infinite loops:

- the definite action $\mathcal{D}$ is enabled (see Fig. 9 for the definition of $\mathsf{Enabled}(\mathcal{D})$). Then the loop must terminate to guarantee that the definite action $\mathcal{D}$ definitely occurs.
- the current thread is not blocked. Here we need to find a condition Q that ensures the current thread can make progress without waiting for actions of other threads.

The second premise of the rule says, in either of the two cases above we must consume a $\lozenge$-token for each round of the loop, as p' has one less token than $p \wedge B$.

On the other hand, if the current thread is blocked (Q does not hold) and it is not in the middle of a definite action, the loop can run an indefinite number of rounds to wait for the environment actions. It does not have to consume tokens. However, we must ensure the thread cannot be blocked forever, i.e., Q cannot be always false. This is achieved by the *definite-progress* condition introduced in LiLi. We show a generalized definition in Def. 7.1, with the changes highlighted in gray boxes.

Definition 7.1 (Definite Progress). $\mathfrak{S} \models (R, G\colon \mathcal{D} \xrightarrow{f} (Q, B_h))$ iff the following hold for any t:

(1) either $\mathfrak{S} \models Q_\mathsf{t}$, or $\mathfrak{S} \models \neg B_h$, or there exists t' such that $\mathsf{t}' \neq \mathsf{t}$ and $\mathfrak{S} \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}'})$;
(2) for any $\mathsf{t}' \neq \mathsf{t}$ and $\mathfrak{S}'$, if $(\mathfrak{S}, \mathfrak{S}', 0) \models R_\mathsf{t} \wedge (\langle \mathcal{D}_{\mathsf{t}'} \rangle \vee ((\neg B_h) \ltimes B_h))$, then $f_\mathsf{t}(\mathfrak{S}') < f_\mathsf{t}(\mathfrak{S})$;
(3) for any $\mathfrak{S}'$, if $(\mathfrak{S}, \mathfrak{S}', 0) \models R_\mathsf{t} \vee G_\mathsf{t}$, then $f_\mathsf{t}(\mathfrak{S}') \leq f_\mathsf{t}(\mathfrak{S})$.

Here f is a function that maps the relational states $\mathfrak{S}$ to some metrics over which there is a well-founded order $<$.

The definite progress condition $(R, G\colon \mathcal{D} \xrightarrow{f} (Q, B_h))$ tries to ensure Q is eventually always true, unless B_h is eventually always false. It requires the following conditions to hold:

(1) Either Q holds, which means that the low-level code is no longer blocked; or the high-level specification $\textbf{await}(B_h)\{C'\}$ is disabled, so that the low-level code does not have to progress to refine the high-level code; or one of the definite actions in $\mathcal{D}$ that the current thread t

$$\frac{\begin{array}{c}\text{for all } f \in dom(\Pi):\quad \Pi(f) = (\mathcal{P}, x, C) \qquad \Gamma(f) = (\mathcal{P}', y, C') \qquad P \Rightarrow (\mathcal{P} \wedge \mathcal{P}') \vee (\neg\mathcal{P} \wedge \neg\mathcal{P}') \\ \mathcal{D}, R, G \vdash \{P \wedge (\mathcal{P} \wedge \mathcal{P}') \wedge (x = y) \wedge \mathsf{arem}(C') \wedge \blacklozenge(E_k, \ldots, E_1)\}\, C\, \{P \wedge \mathsf{arem}(\mathbf{skip})\} \\ \forall \mathsf{t}, \mathsf{t}'.\ \mathsf{t} \neq \mathsf{t}' \Longrightarrow G_{\mathsf{t}} \Rightarrow R_{\mathsf{t}'} \qquad \mathsf{wffAct}(R, \mathcal{D}) \qquad P \Rightarrow \neg\mathsf{Enabled}(\mathcal{D})\end{array}}{\mathcal{D}, R, G \vdash \{P\}\Pi : \Gamma} \ (\textsc{obj})$$

$$\frac{\begin{array}{c}p \wedge B \Rightarrow p' \qquad p \wedge B \wedge (\mathsf{Enabled}(\mathcal{D}) \vee Q) \Rightarrow p' * (\Diamond \wedge \mathsf{emp}) \qquad \mathcal{D}, R, G \vdash \{p'\} C \{p\} \\ p \wedge B \Rightarrow J \wedge \mathsf{arem}(\mathbf{await}(B')\{C'\}) \qquad \mathsf{Sta}(J, R \vee G) \qquad J \Rightarrow (R, G\colon \mathcal{D}' \xrightarrow{f} (Q, B')) \\ \mathcal{D}' \leqslant \mathcal{D} \qquad \mathsf{wffAct}(R, \mathcal{D}')\end{array}}{\mathcal{D}, R, G \vdash \{p\}\mathbf{while}\ (B)\{C\}\{p \wedge \neg B\}} \ (\textsc{whl})$$

$$\frac{\begin{array}{c}p \wedge \mathsf{Enabled}(\mathcal{D}) \Rightarrow B \qquad \mathcal{D}, \mathsf{Id}, G \vdash \{p \wedge B\}\langle C\rangle\{q\} \qquad \mathsf{Sta}(\{p, q\}, R) \\ \mathcal{D}' \leqslant \mathcal{D} \qquad \mathsf{wffAct}(R, \mathcal{D}') \qquad p \Rightarrow \exists B', C'.\ \mathsf{arem}(\mathbf{await}(B')\{C'\}) \wedge (R\colon \mathcal{D}' \circ\!\!\xrightarrow{f} (B, B'))\end{array}}{\mathcal{D}, R, G \vdash_{\mathsf{wfair}} \{p\}\mathbf{await}(B)\{C\}\{q\}} \ (\textsc{await-w})$$

$$\frac{\begin{array}{c}p \wedge \mathsf{Enabled}(\mathcal{D}) \Rightarrow B \qquad \mathcal{D}, \mathsf{Id}, G \vdash \{p \wedge B\}\langle C\rangle\{q\} \qquad \mathsf{Sta}(\{p, q\}, R) \\ \mathcal{D}' \leqslant \mathcal{D} \qquad \mathsf{wffAct}(R, \mathcal{D}') \qquad p \Rightarrow \exists B', C'.\ \mathsf{arem}(\mathbf{await}(B')\{C'\}) \wedge (R\colon \mathcal{D}' \bullet\!\!\xrightarrow{f} (B, B'))\end{array}}{\mathcal{D}, R, G \vdash_{\mathsf{sfair}} \{p\}\mathbf{await}(B)\{C\}\{q\}} \ (\textsc{await-s})$$

Fig. 10. The key extensions of inference rules.

waits for is enabled in some thread t'. Here $\mathcal{D}$ can be viewed as a set of n definite actions in the form of $\mathcal{D}_1 \wedge \ldots \wedge \mathcal{D}_n$ parameterized with thread IDs.

(2) There is a well-founded metric f that becomes strictly smaller whenever (a) an environment thread t' executes a definite action in $\mathcal{D}$, or (b) an environment action has turned the high-level command from disabled to enabled. Case (a) requires that the number of definite actions waited by the current thread must be strictly decreasing. Therefore eventually there are no enabled definite actions. By condition (1) we know eventually either Q or $\neg B_h$ is true. Case (b) further requires that the high-level command cannot be infinitely often disabled and then enabled during the loop. Therefore either B_h is eventually always true or it is eventually always false. In the former case we know Q must be eventually always true by the above condition (1). In the latter the loop does not have to terminate because the execution is well-blocked (see Fig. 6).

(3) The value of f over program states cannot be increased by any level-0 actions (i.e., non-delaying actions).

Note that the last two conditions do not prevent delaying actions (level-k actions where $k > 0$) from increasing the value of f, but such an increase can only occur a finite number of times because each delaying action consumes a $\blacklozenge$-token. The effects of delaying actions are shown in the ATOM rule, which is the same as in LiLi. Since the way delaying action is handled is orthogonal to our extension for partial methods, we omit the rule here.

To ensure the definite progress condition always holds, we need to find an invariant J which is preserved by any program step (by the current thread or by the environment), and require that J implies definite progress, given the currently remaining high-level command $\mathbf{await}(B')\{C'\}$. Note that to simplify the presentation we treat $\mathsf{arem}(\mathbf{skip})$ as $\mathsf{arem}(\mathbf{await}(\mathsf{true})\{\mathbf{skip}\})$ so that it can be reasoned about in the same way.

The WHL rule also allows us to use $\mathcal{D}'$, a subset of $\mathcal{D}$, to prove definite progress, which is useful to simplify the proofs. See the definition of $\mathcal{D}' \leqslant \mathcal{D}$ in Fig. 9. $\mathcal{D}'$ also needs to satisfy $\mathsf{wffAct}(R, \mathcal{D}')$. This premise is taken from LiLi and we do not explain it here.

Since we have highlighted the changes over the WHL rule in LiLi, we can see that the WHL rule in LiLi is a specialization of ours when the high-level code is always in the form of **await**(true){C'}.

Rules for **await** *commands.* Our logic introduces two new rules, AWAIT-W and AWAIT-S, to verify **await** commands in the *object implementation* under weakly fair and strongly fair scheduling. We use the subscripts of the judgment to distinguish the scheduling.

Naturally the AWAIT-W rule combines the ATOM rule in LiLi and the WHL rule. If **await**(B){C} is enabled, we can simply treat C as an atomic block $\langle C \rangle$ and apply the ATOM rule of LiLi to verify it. In this case we do not need to consider the interference and take Id as the rely condition (where Id is a shorthand notation for [true], which specifies arbitrary identity transitions).

Similar to the WHL rule, if the definite action $\mathcal{D}$ is enabled, then **await**(B){C} must be enabled at this point (see the first premise of the AWAIT-W rule). This is because we require that, when enabled, the definite action $\mathcal{D}$ must be fulfilled regardless of environment behaviors. Therefore the current thread cannot be blocked.

Finally, we require that, even if the command is blocked, it must be eventually enabled unless the corresponding high-level specification is blocked too. So if we view the enabling condition B the same as the condition Q we use in the WHL rule, we require the same *definite progress* condition, except that things are simpler here because **await**(B){C} finishes in one step once enabled, unlike loops which take multiple steps to finish even if Q holds. Therefore we do not need the invariant J used in the WHL rule, and we do not need to consider actions in G in the definite progress condition. We can use a simpler condition $(R\colon \mathcal{D}' \circ\!\!\xrightarrow{f} (B, B'))$ defined below, which simply instantiates G with Id and Q with B in $(R, G\colon \mathcal{D}' \xrightarrow{f} (Q, B'))$ (see Def. 7.1).

Definition 7.2 (Definite Progress for Await).

- $\Im \models (R\colon \mathcal{D} \circ\!\!\xrightarrow{f} (B_l, B_h))$ iff $\Im \models (R, \mathsf{Id}\colon \mathcal{D} \xrightarrow{f} (B_l, B_h))$.
- $\Im \models (R\colon \mathcal{D} \bullet\!\!\xrightarrow{f} (B_l, B_h))$ iff the following hold for any t:
  (1) either $\Im \models B_l$, or $\Im \models \neg B_h$, or there exists t′ such that $\mathsf{t}' \neq \mathsf{t}$ and $\Im \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}'})$;
  (2) for any $\mathsf{t}' \neq \mathsf{t}$ and $\Im'$, if $(\Im, \Im', 0) \models R_{\mathsf{t}} \wedge (\langle \mathcal{D}_{\mathsf{t}'} \rangle \vee ((\neg B_h) \ltimes B_h))$, then $f_{\mathsf{t}}(\Im') < f_{\mathsf{t}}(\Im)$;
  (3) for any $\Im'$, if $(\Im, \Im', 0) \models R_{\mathsf{t}} \wedge ((\neg B_l) \ltimes (\neg B_l))$, then $f_{\mathsf{t}}(\Im') \leq f_{\mathsf{t}}(\Im)$.

The AWAIT-S rule for strongly fair scheduling looks almost the same as the AWAIT-W rule, with a slightly different definite progress condition $(R\colon \mathcal{D}' \bullet\!\!\xrightarrow{f} (B, B'))$, which is also shown in Def. 7.2 with the difference highlighted in the gray box. The key difference here is that the low-level enabling condition B (represented as B_l in Def. 7.2) does not have to be stable once it becomes true. Under strongly fair scheduling we know the **await** block will be executed as long as it is enabled infinitely often. Therefore in condition (3) we only need to ensure that f does not increase if the enabling condition B_l remains false, but we allow f to increase whenever we see B_l holds.

The WHL rule, AWAIT-W rule and AWAIT-S rule are the only new command rules we introduce to reason about partial methods and blocking primitives. All the other command rules are taken directly from LiLi, which are omitted here.

7.3 Soundness of the Logic

The two AWAIT rules actually give us two program logics, for strongly fair and weakly fair scheduling respectively. To distinguish them, we use $\mathcal{D}, R, G \vdash_\chi \{P\}\Pi : \Gamma$ to represent the verification using the logic for χ-scheduling ($\chi \in \{\mathsf{sfair}, \mathsf{wfair}\}$), where the corresponding AWAIT rule is used.

Theorem 7.3 shows that our logic is sound in that it guarantees linearizability and partial deadlock freedom (PDF) of concurrent objects. It also ensures partial starvation freedom (PSF) if the rely/guarantee conditions specify only level-0 actions, as required by $R \Rightarrow \lfloor R \rfloor_0$ and $G \Rightarrow \lfloor G \rfloor_0$. That is, none of the object actions of a thread could delay the progress of other threads. With the specialized R and G, we can derive the progress of each single thread, which gives us PSF.

THEOREM 7.3 (SOUNDNESS). *If* $\mathcal{D}, R, G \vdash_\chi \{P\}\Pi : \Gamma$ *and* $\varphi \Rightarrow P$, *then*

(1) both $\Pi \preceq^{\mathsf{lin}}_{\varphi} \Gamma$ *and* $\mathsf{PDF}^{\chi}_{\varphi,\Gamma}(\Pi)$ *hold; and*

(2) if $R \Rightarrow \lfloor R \rfloor_0$ *and* $G \Rightarrow \lfloor G \rfloor_0$, *then* $\mathsf{PSF}^{\chi}_{\varphi,\Gamma}(\Pi)$ *holds.*

where $\chi \in \{\mathsf{sfair}, \mathsf{wfair}\}$, *and* $\varphi \Rightarrow P \stackrel{\text{def}}{=} \forall \sigma, \Sigma.\, (\varphi(\sigma) = \Sigma) \Longrightarrow (\sigma, \Sigma) \models P$.

Proofs of the theorem are in Appendix B. We first prove the logic establishes the progress-aware contextual refinements, and then apply the Abstraction Theorem 6.2 to ensure linearizability and the progress properties. The proof structure is similar to the one for LiLi.

8 EXAMPLES

We have applied the program logic to verify ticket locks [Mellor-Crummey and Scott 1991], test-and-set locks [Herlihy and Shavit 2008], bounded partial queues with two locks [Herlihy and Shavit 2008] (where the locks are implemented using the specification (2.3)) and Treiber stacks [Treiber 1986] with partial pop methods. Perhaps interestingly, we also use our logic to prove that, for the atomic partial specification Γ for locks, the wrapping of Γ (as the object implementation) respects Γ itself as the atomic specification under the designated fairness conditions, i.e., $\left(\mathcal{D}, R, G \vdash \{P\}\mathsf{wr}^{\chi}_{\mathsf{Prog}}(\Gamma) : \Gamma\right)$ holds for certain $\mathcal{D}$, R, G and P, and for different combinations of fairness χ and progress Prog. This result validates our wrappers and program logic. It shows $\mathsf{Prog}^{\chi}_{\varphi,\Gamma}(\mathsf{wr}^{\chi}_{\mathsf{Prog}}(\Gamma))$ holds, i.e., each wrapper itself satisfies the corresponding progress property. Below we show the proofs for test-and-set locks, ticket locks, and simple locks implemented using **await** which guarantee PSF under weak fairness. The proofs of other examples are given in Appendix C.

8.1 Test-and-Set Locks

In Fig. 11, we verify PDF of the test-and-set locks using our logic with the atomic partial specifications `L_ACQ'` and `L_REL` defined in (2.3). To distinguish the variables at the two levels, below we use capital letters (e.g., `L`) in the specifications and small letters (e.g., `l`) in the implementations.

As we explained in Sec. 3, the method `L_rel` and the specification `L_REL` have annotated preconditions (`l = cid`) and (`L = cid`), respectively. That is, it is not allowed to call `L_rel` (or `L_REL`) when the thread does not hold the lock. The annotated precondition for `L_acq` and `L_ACQ'` is true. In Fig. 11, we define the assertion lock as the object invariant P used in the OBJ rule. Then the method `L_acq` is verified with the precondition lock, and `L_rel` is verified with the precondition lock $\wedge$ (`l = cid`) which is reduced to $\mathsf{lockedBy}_{\mathtt{cid}}$, as shown in Fig. 11.

To verify `L_acq`, we make the following key observations. When the **cas** at line 3 succeeds, `L_ACQ'` must be enabled and can be executed correspondingly. And at the time when the **cas** fails, `L_ACQ'` must be disabled. The progress of `L_acq` relies on that the environment thread holding the lock could eventually release the lock, i.e., turning the current thread's `L_ACQ'` from disabled to enabled. But such an action is not "definite", since the client thread may never call the `L_rel`

$\mathsf{lock} \stackrel{\text{def}}{=} \exists s.\, \mathsf{lock}_s \qquad \mathsf{lock}_s \stackrel{\text{def}}{=} (1 = \mathsf{L} = s)$

$\mathsf{unlocked} \stackrel{\text{def}}{=} \mathsf{lock}_0$

$\mathsf{locked} \stackrel{\text{def}}{=} \exists t.\, \mathsf{lockedBy}_t$

$\mathsf{lockedBy}_t \stackrel{\text{def}}{=} \mathsf{lock}_t \wedge (t \neq 0)$

$R_t \stackrel{\text{def}}{=} \bigvee_{t' \neq t} G_{t'} \qquad G_t \stackrel{\text{def}}{=} Acq_t \vee Rel_t$

$Acq_t \stackrel{\text{def}}{=} \mathsf{unlocked} \ltimes_1 \mathsf{lockedBy}_t$

$Rel_t \stackrel{\text{def}}{=} \mathsf{lockedBy}_t \ltimes_0 \mathsf{unlocked}$

$\mathcal{D} \stackrel{\text{def}}{=} \mathsf{false} \leadsto \mathsf{true}$

$J \stackrel{\text{def}}{=} \mathsf{lock}$

$Q \stackrel{\text{def}}{=} \mathsf{unlocked}$

$$f(\mathfrak{S}) = \begin{cases} 1 & \text{if } \mathfrak{S} \models \mathsf{locked} \\ 0 & \text{if } \mathfrak{S} \models Q \end{cases}$$

```
L_acq(){
    {lock ∧ ♦ ∧ arem(L_ACQ')}
1   local b := false;
    {((¬b) ∧ lock ∧ ♦ ∧ ◊ ∧ arem(L_ACQ'))
        ∨ (b ∧ lockedBy_cid ∧ arem(skip))}
2   while (!b) {
        {((unlocked ∧ ♦) ∨ (locked ∧ ♦ ∧ ◊))
         ∧ arem(L_ACQ')}
3     b := cas(&l, 0, cid);
4   }
    {lockedBy_cid ∧ arem(skip)}
}
L_rel(){
    {lockedBy_cid ∧ arem(L_REL)}
5   l := 0;
    {lock ∧ arem(skip)}
}
```

Fig. 11. Proofs for the test-and-set lock.

method. The definite action $\mathcal{D}$ for this object can be defined as false $\leadsto$ true, saying that there is no definite action that a thread needs to complete.

The action Acq_t (corresponding to the successful **cas** at line 3) is a delaying action (defined with level 1). When thread t succeeds in **cas**, termination of other threads' `L_acq` can be delayed, as allowed by PDF. The thread t has to pay a ♦-token, given in the precondition of `L_acq`.

The definite progress condition $(R, G : \mathcal{D} \xrightarrow{f} (Q, \mathtt{L=0}))$ now says that thread t is either at a state that it itself can progress (i.e., Q holds), or blocked at the abstract level (i.e., `L=0` does not hold). The metric $f_t(\mathfrak{S})$ decreases when an environment thread releases `L`, but can be reset (which means thread t is delayed) if an environment thread successfully acquires the lock.

By the Soundness Theorem 7.3, we know the test-and-set lock object satisfies the PDF property, and contextually refines the abstraction generated by the corresponding PDF wrappers in Fig. 7, under strongly and weakly fair scheduling.

8.2 Ticket Locks

In Fig. 12, we prove the ticket lock object satisfies PSF. We introduce some write-only auxiliary variables to help the verification. First, we introduce an array `ticket` to help specify the queue of the threads requesting the lock. Each array cell `ticket[i]` records the ID of the unique thread getting the ticket number `i` (see line 2). Second, we introduce a lock bit `l` to make the lock acquirement and lock release explicit (see lines 4 and 5).

We then define the object invariant $\mathsf{lock}(s, tl, n_1, n_2)$. It says that the lock bits `l` and `L` are equal, n_1 and n_2 are the values of `owner` and `next` respectively, and tl is the list of the threads recorded in `ticket`$[n_1]$, `ticket`$[n_1 + 1], \ldots,$ `ticket`$[n_2 - 1]$ (as specified by $\mathsf{tickets}(tl, n_1, n_2)$).

The guarantee condition G_t describes the possible atomic actions of thread t. Req_t adds t at the end of tl of the threads requesting the lock and also increments `next`. It corresponds to line 2 in the code at the top of Fig. 12. Acq_t sets the lock bits to t, explicitly indicating the lock acquirement (see line 4). It is also a definite action (see the definition of $\mathcal{D}_t$) since thread t must acquire the lock if its loop at line 3 terminates. Rel_t increments `owner` to dequeue the thread t which currently holds the lock, and resets the lock bits (see line 5). All actions are at level 0. There are no delaying actions.

```
tkL_acq(){
1  local i, o;
2  <i := getAndInc(&next); ticket[i] := cid >;
3  o := owner;  while (i != o) { o := owner; }
4  l := cid;
}
```

```
tkL_rel(){
5  <owner := owner+1; l := 0 >;
}
```

$$\mathsf{lock}(s, tl, n_1, n_2) \stackrel{\text{def}}{=} (\mathtt{l} = \mathtt{L} = s \wedge (s = \mathsf{head}(tl) \vee s = 0)) * ((\mathtt{owner} = n_1) * (\mathtt{next} = n_2) \wedge (n_1 \le n_2)) * \mathsf{tickets}(tl, n_1, n_2)$$

$$G_\mathtt{t} \stackrel{\text{def}}{=} \mathit{Req}_\mathtt{t} \vee \mathit{Acq}_\mathtt{t} \vee \mathit{Rel}_\mathtt{t} \qquad \mathcal{D}_\mathtt{t} \stackrel{\text{def}}{=} \forall tl, n_1, n_2.\ \mathsf{lock}(0, \mathtt{t} :: tl, n_1, n_2) \rightsquigarrow \mathsf{lock}(\mathtt{t}, \mathtt{t} :: tl, n_1, n_2)$$

$$\mathit{Req}_\mathtt{t} \stackrel{\text{def}}{=} \exists s, tl, n_1, n_2.\ \mathsf{lock}(s, tl, n_1, n_2) \ltimes \mathsf{lock}(s, tl{+}{+}[\mathtt{t}], n_1, n_2 + 1)$$

$$\mathit{Acq}_\mathtt{t} \stackrel{\text{def}}{=} \exists tl, n_1, n_2.\ \mathsf{lock}(0, \mathtt{t} :: tl, n_1, n_2) \ltimes \mathsf{lock}(\mathtt{t}, \mathtt{t} :: tl, n_1, n_2)$$

$$\mathit{Rel}_\mathtt{t} \stackrel{\text{def}}{=} \exists tl, n_1, n_2.\ \mathsf{lock}(\mathtt{t}, \mathtt{t} :: tl, n_1, n_2) \ltimes \mathsf{lock}(0, tl, n_1 + 1, n_2)$$

$$J_\mathtt{t} \stackrel{\text{def}}{=} \exists s, n_1, n_2, tl_1, tl_2.\ \mathsf{tlocked}_{tl_1, \mathtt{t}, tl_2}(s, n_1, \mathtt{i}, n_2) \wedge (\mathtt{o} \le n_1)$$

$$Q_\mathtt{t} \stackrel{\text{def}}{=} \exists n_2, tl_2.\ \mathsf{lock}(0, \mathtt{t} :: tl_2, \mathtt{i}, n_2) \wedge (\mathtt{o} \le \mathtt{i}) \qquad f(\Xi) = \begin{cases} 2k+1 & \text{if } \Xi \models (\mathtt{i} - \mathtt{owner} = k) * (\mathtt{l} = 0) \\ 2k & \text{if } \Xi \models (\mathtt{i} - \mathtt{owner} = k) * (\mathtt{l} \neq 0) \end{cases}$$

Fig. 12. Proofs for the ticket lock (with auxiliary code in gray).

By applying the WHL rule of our logic, we need to prove the definite progress condition $J \Rightarrow (R, \mathsf{Id} : \mathcal{D} \xrightarrow{f} (Q, \mathtt{L{=}0}))$ for the loop at line 3. Here J, Q and f are defined at the bottom of Fig. 12. In the definition of $J_\mathtt{t}$, we use $\mathsf{tlocked}_{tl_1, \mathtt{t}, tl_2}(s, n_1, \mathtt{i}, n_2)$ to say that t is requesting the lock and its ticket number is i. Here tl_1 is the list of the threads which are waiting ahead of t, and tl_2 is for the threads behind t. $Q_\mathtt{t}$ specifies the case when tl_1 is empty. In this case the lock bits must be 0 and tlocked is reduced to lock, as shown at the bottom of Fig. 12.

The metric $f_\mathtt{t}(\Xi)$ is determined by the number of threads ahead of t in the waiting queue and the status of the lock bits. It decreases when an environment thread $\mathtt{t}'$ does the definite action $\mathcal{D}_{\mathtt{t}'}$, setting the lock bits to $\mathtt{t}'$. It also decreases when $\mathtt{t}'$ releases the lock and increments owner, turning $(\mathtt{L} \neq 0)$ to $(\mathtt{L} = 0)$. Thus we can prove $J \Rightarrow (R, \mathsf{Id} : \mathcal{D} \xrightarrow{f} (Q, \mathtt{L{=}0}))$.

By the Soundness Theorem 7.3, we know the ticket lock object satisfies the PSF property, and contextually refines the abstraction generated by the corresponding PSF wrappers, under both strongly and weakly fair scheduling. The detailed formal proofs are given in Appendix C.1.

8.3 Simple PSF Locks with Await Blocks

Figure 13 shows the proofs of a simple lock object implemented with an **await** statement which guarantees PSF under weak fairness. The acquire method is simply $\mathsf{wr}_{\mathsf{PSF}}^{\mathsf{wfair}}(\mathbf{await}(\mathtt{l{=}0})\{\mathtt{l{:=}cid}\})$. The release method resets the lock bit l directly. It has the annotated precondition $(\mathtt{l} = \mathtt{cid})$. We still verify the object in our logic with the specifications L_ACQ' and L_REL defined in (2.3).

We first define the object invariant P used in the OBJ rule. It is defined based on lock, which requires l to have the same value as the abstract lock L. The queue listid records the threads currently waiting for the lock. Here $\mathsf{diff}(tb)$ says that the threads in tb are all different. Then the object invariant $P_\mathtt{t}$ further requires that the current thread t is not recorded in listid. It is preserved before and after t calls a method.

The object has three kinds of possible actions (see the definition of G). $\mathit{Req}_\mathtt{t}$ appends the thread t at the end of listid to request the lock (line 1). $\mathit{Acq}_\mathtt{t}$ acquires the lock if the lock is available and

$P_t \stackrel{\text{def}}{=} \exists s, tb.\ \mathsf{lock}_s(tb) \wedge (t \notin tb)$ where $tb ::= \epsilon \mid (t, \text{'l=0'}) :: tb$

$\mathsf{lock}_s(tb) \stackrel{\text{def}}{=} (l = L = s) * (\mathtt{listid} = tb) \wedge \mathsf{diff}(tb)$

$\mathsf{unlocked}(tb) \stackrel{\text{def}}{=} \mathsf{lock}_0(tb)$ $\quad \mathsf{lockReq}_t \stackrel{\text{def}}{=} \exists s, tb.\ \mathsf{lock}_s(tb) \wedge (t \in tb)$

$\mathsf{locked}_t(tb) \stackrel{\text{def}}{=} \mathsf{lock}_t(tb) \wedge (t \neq 0) \wedge (t \notin tb)$ $\quad \mathsf{locked}_t \stackrel{\text{def}}{=} \exists tb.\ \mathsf{locked}_t(tb)$

$G_t \stackrel{\text{def}}{=} Req_t \vee Acq_t \vee Rel_t$ $\quad \mathcal{D}_t \stackrel{\text{def}}{=} \forall tb.\ \mathsf{unlocked}((t, \text{'l=0'}) :: tb) \rightsquigarrow \mathsf{locked}_t(tb)$

$Req_t \stackrel{\text{def}}{=} \exists s, tb.\ (\mathsf{lock}_s(tb) \wedge (t \notin tb)) \ltimes \mathsf{lock}_s(tb +\!\!+ [(t, \text{'l=0'})])$

$Acq_t \stackrel{\text{def}}{=} \exists tb.\ \mathsf{unlocked}((t, \text{'l=0'}) :: tb) \ltimes \mathsf{locked}_t(tb)$ $\quad Rel_t \stackrel{\text{def}}{=} \exists tb.\ \mathsf{locked}_t(tb) \ltimes \mathsf{unlocked}(tb)$

$$f_t(\Im) \stackrel{\text{def}}{=} \begin{cases} 2k+1 & \text{if } \exists s, tb, tb'.\ (\Im \models \mathsf{lock}_s(tb +\!\!+ [(t, \text{'l=0'})] +\!\!+ tb') \wedge s \neq 0) \wedge |tb| = k \\ 2k & \text{if } \exists tb, tb'.\ (\Im \models \mathsf{unlocked}(tb +\!\!+ [(t, \text{'l=0'})] +\!\!+ tb')) \wedge |tb| = k \end{cases}$$

```
acquire(){
   {P_cid ∧ arem(L_ACQ')}
1  listid := listid ++ [(cid, 'l=0')];
   {lockReq_cid ∧ arem(L_ACQ')}
2  await (l = 0 /\ cid = enhd(listid)) {
     {∃tb. unlocked((cid,'l=0') :: tb) ∧ arem(L_ACQ')}
3    l := cid;  listid := listid \ cid;
     {∃tb. locked_cid(tb) ∧ arem(skip)}
4  }
   {locked_cid ∧ arem(skip)}
}
```

```
release(){
   {locked_cid ∧ arem(L_REL)}
5  l := 0;
   {P_cid ∧ arem(skip)}
}
```

Fig. 13. Proofs for the simple PSF lock under weak fairness.

t is at the head of `listid` (lines 2–4). Rel_t releases the lock (line 5). Here Acq_t is also the definite action of thread t (see the definition of $\mathcal{D}$). None of the actions are delaying actions.

To verify the **await** statement at lines 2–4, we apply the AWAIT-W rule in Fig. 10, and prove:

$$\mathsf{lockReq} \Rightarrow (R: \mathcal{D} \overset{f}{\hookrightarrow} (l = 0 \wedge \mathtt{cid} = \mathtt{enhd(listid)}, L = 0)) . \tag{8.1}$$

The metric f is defined at the top of Fig. 13. We can see that $f_t(\Im)$ decreases when an environment thread t′ performs a definite action, since $\mathcal{D}_{t'}$ will remove t′ that is waiting ahead of the thread t. Also $f_t(\Im)$ decreases when t′ releases the lock, turning $(L \neq 0)$ to $(L = 0)$. Thus (8.1) holds.

By the Soundness Theorem 7.3, we know this simple lock satisfies PSF under weak fairness.

9 RELATED WORK AND CONCLUSION

There has been much work on the relationships between linearizability, progress properties and contextual refinement (e.g., [Filipović et al. 2009; Gotsman and Yang 2011, 2012; Liang et al. 2013]), and on verifying progress properties or progress-aware refinement (e.g., [Boström and Müller 2015; da Rocha Pinto et al. 2016; Gotsman et al. 2009; Hoffmann et al. 2013; Jacobs et al. 2015; Tassarotti et al. 2017]). But none of them studies objects with partial methods as we do. On the other hand, our ideas might be general enough to be integrated with these verification methods to support blocking primitives and partial methods. For instance, Tassarotti et al. [2017] propose a higher-order logic based on Iris [Jung et al. 2015] for fair refinements. Our wrappers and reasoning method may be applied there to support higher-order refinement reasoning with blocking primitives. The logic by Boström and Müller [2015] ensures that no thread will be blocked forever. It supports special built-in blocking primitives for locking, message passing and thread join. Their obligation-based

reasoning strategies may be applied to **await** blocks too, to verify that the client threads of **await** will not be permanently blocked.

In our previous work we propose the program logic LiLi [Liang and Feng 2016] to verify starvation-free and deadlock-free objects. This work is inspired by several ideas from LiLi:

- The soundness of LiLi ensures a progress-aware contextual refinement, which gives starvation-freedom or deadlock-freedom, if fed with different abstractions generated by specific code wrappers. Here we take a similar approach, and define new wrappers to generate abstractions for PSF and PDF objects.
- LiLi sorts progress properties in two dimensions called blocking and delay, and distinguish starvation-freedom and deadlock-freedom by whether delay is permitted. Here the difference between our PSF and PDF also lies in the delay dimension.
- The program logic proposed in this paper is a generalization of LiLi. Both logics use tokens to support delay, and use similar definite progress conditions to support blocking.

However, there are two main problems with LiLi, which are addressed in this paper:

- LiLi does not provide abstractions for objects with partial methods. When using LiLi to verify lock-based algorithms (such as the counters shown in Fig. 1(b) and (d) in this paper), one has to inline the implementations of locks, losing the modularity of verification. Here we define progress-aware abstractions for objects with partial methods, allowing us to verify their clients in a modular way.
- The inference rules of LiLi do not apply to objects with partial methods, such as the objects in Sec. 8 in this paper. We have explained the reasons and our solutions in Sec. 7.

Schellhorn et al. [2016] propose a proof method for verifying starvation-freedom. Their approach is based on a special predicate which describes the waiting-for relations among the threads. However, their work has similar problems as LiLi, and cannot apply to the examples considered in this paper.

Gu et al. [2016] verify progress of the ticket lock implementation as part of their verified kernel. Their specification of the lock relies on the behaviors of clients. It requires that the client owning a lock must eventually release it. Then they prove that the acquire method always terminates with the cooperative clients. It is unclear how the approach can be applied for general objects with partial methods.

Conclusions and more discussions. We have studied the progress of objects with partial methods in three aspects. First, we define new progress properties, partial starvation-freedom (PSF) and partial deadlock-freedom (PDF). Second, we design wrappers to generate abstractions for PSF and PDF objects under strongly or weakly fair scheduling. Third, we develop a program logic to verify PSF and PDF.

Although our program logic verifies both linearizability and progress properties, it is focused more on the latter. Existing work [Khyzha et al. 2017; Liang and Feng 2013; Turon et al. 2013] has shown that linearizability itself can be challenging to verify, and special mechanisms are needed for very fine-grained objects with non-fixed linearization points (LPs). Our logic cannot verify these objects, but our conjecture is that the mechanisms handling non-fixed LPs (as in [Liang and Feng 2013]) are orthogonal to our progress reasoning, and they can be integrated into our logic if needed.

The logic follows LiLi's ideas of definite actions and stratified tokens to reason about progress. They can be viewed as special strategies implementing the general principle for termination reasoning, that is to find a well-founded metric that keeps decreasing during the program execution. These ideas and rules give a concrete guide to users on how to construct the metric and the proofs. Although we have tried to make them as general as possible, and they have been shown applicable to many non-trivial algorithms (see [Liang and Feng 2016] and Appendix C), they may not be

complete and it would be unsurprising if there are examples that they cannot handle. As future work, we would like to verify more examples to explore the scope of the applicability.

The specifications of linearizable objects must be *atomic*, but sometimes we may want to give non-atomic specifications to object methods. We can apply our wrappers to every occurrence of the **await** blocks in the non-atomic specifications to establish progress-aware refinements. We suspect that our logic can still be used to verify such refinements (as in [Liang et al. 2014]). Another potential limitation may be due to the use of the pure Boolean expression B in $\mathbf{await}(B)\{C\}$, which may limit the expressiveness of the specifications. However, our technical development does not rely on this setting. Everything may still hold if we replace B with the more expressive state assertions.

Other interesting future work includes automating the verification process. One of the key problems is to infer the definite actions and prove the definite progress conditions. There have been efforts to synthesize the ranking functions for loop termination (see [Cook et al. 2011] for an overview), which may provide insights for automating the definite progress proofs. In addition we might be able to follow the ideas in automated rely-guarantee reasoning (e.g., [Calcagno et al. 2007]) to automate the verification in our rely-guarantee logic.

ACKNOWLEDGMENTS

We would like to thank anonymous referees for their suggestions and comments on earlier versions of this paper. This work is supported in part by grants from National Natural Science Foundation of China (NSFC) under Grant Nos. 61379039, 61502442 and 61632005.

REFERENCES

Pontus Boström and Peter Müller. 2015. Modular Verification of Finite Blocking in Non-terminating Programs. In *Proceedings of the 29th European Conference on Object-Oriented Programming (ECOOP 2015)*. 639–663.

Cristiano Calcagno, Matthew J. Parkinson, and Viktor Vafeiadis. 2007. Modular Safety Checking for Fine-Grained Concurrency. In *Proceedings of the 14th International Symposium on Static Analysis (SAS 2007)*. 233–248.

Byron Cook, Andreas Podelski, and Andrey Rybalchenko. 2011. Proving Program Termination. *Commun. ACM* 54, 5 (2011), 88–98.

Pedro da Rocha Pinto, Thomas Dinsdale-Young, Philippa Gardner, and Julian Sutherland. 2016. Modular Termination Verification for Non-blocking Concurrency. In *Proceedings of the 25th European Symposium on Programming Languages and Systems (ESOP 2016)*. 176–201.

Xinyu Feng. 2009. Local rely-guarantee reasoning. In *POPL*. 315–327.

Ivana Filipović, Peter O'Hearn, Noam Rinetzky, and Hongseok Yang. 2009. Abstraction for Concurrent Objects. In *Proceedings of the 18th European Symposium on Programming (ESOP 2009)*. 252–266.

Alexey Gotsman, Byron Cook, Matthew J. Parkinson, and Viktor Vafeiadis. 2009. Proving that Non-Blocking Algorithms Don't Block. In *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2009)*. 16–28.

Alexey Gotsman and Hongseok Yang. 2011. Liveness-Preserving Atomicity Abstraction. In *Proceedings of the 38th International Conference on Automata, Languages and Programming (ICALP 2011)*. 453–465.

Alexey Gotsman and Hongseok Yang. 2012. Linearizability with Ownership Transfer. In *Proceedings of the 23rd International Conference on Concurrency Theory (CONCUR 2012)*. 256–271.

Ronghui Gu, Zhong Shao, Hao Chen, Xiongnan (Newman) Wu, Jieung Kim, Vilhelm Sjöberg, and David Costanzo. 2016. CertiKOS: An Extensible Architecture for Building Certified Concurrent OS Kernels. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI 2016)*. 653–669.

Maurice Herlihy and Nir Shavit. 2008. *The Art of Multiprocessor Programming*. Morgan Kaufmann.

Maurice Herlihy and Nir Shavit. 2011. On the Nature of Progress. In *Proceedings of the 15th International Conference on Principles of Distributed Systems (OPODIS 2011)*. 313–328.

Maurice Herlihy and Jeannette Wing. 1990. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Trans. Program. Lang. Syst.* 12, 3 (1990), 463–492.

Jan Hoffmann, Michael Marmar, and Zhong Shao. 2013. Quantitative Reasoning for Proving Lock-Freedom. In *Proceedings of the 28th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2013)*. 124–133.

Bart Jacobs, Dragan Bosnacki, and Ruurd Kuiper. 2015. Modular Termination Verification. In *Proceedings of the 29th European Conference on Object-Oriented Programming (ECOOP 2015)*. 664–688.

Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer. 2015. Iris: Monoids and Invariants as an Orthogonal Basis for Concurrent Reasoning. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2015)*. 637–650.

Artem Khyzha, Mike Dodds, Alexey Gotsman, and Matthew J. Parkinson. 2017. Proving Linearizability Using Partial Orders. In *Proceedings of the 26th European Symposium on Programming (ESOP 2017)*. 639–667.

Hongjin Liang and Xinyu Feng. 2013. Modular Verification of Linearizability with Non-Fixed Linearization Points. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2013)*. 459–470.

Hongjin Liang and Xinyu Feng. 2016. A Program Logic for Concurrent Objects Under Fair Scheduling. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2016)*. 385–399.

Hongjin Liang, Xinyu Feng, and Zhong Shao. 2014. Compositional Verification of Termination-preserving Refinement of Concurrent Programs. In *Proceedings of the Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (CSL-LICS 2014)*. 65:1–65:10.

Hongjin Liang, Jan Hoffmann, Xinyu Feng, and Zhong Shao. 2013. Characterizing Progress Properties of Concurrent Objects via Contextual Refinements. In *Proceedings of the 24th International Conference on Concurrency Theory (CONCUR 2013)*. 227–241.

John M. Mellor-Crummey and Michael L. Scott. 1991. Algorithms for Scalable Synchronization on Shared-memory Multiprocessors. *ACM Trans. Comput. Syst.* 9, 1 (1991), 21–65.

Matthew Parkinson, Richard Bornat, and Cristiano Calcagno. 2006. Variables as Resource in Hoare Logics. In *LICS*. 137–146.

Gerhard Schellhorn, Oleg Travkin, and Heike Wehrheim. 2016. Towards a Thread-Local Proof Technique for Starvation Freedom. In *Proceedings of the 12th International Conference on Integrated Formal Methods (IFM 2016)*. 193–209.

Joseph Tassarotti, Ralf Jung, and Robert Harper. 2017. A Higher-Order Logic for Concurrent Termination-Preserving Refinement. In *Proceedings of the 26th European Symposium on Programming (ESOP 2017)*. 909–936.

R. Kent Treiber. 1986. *System Programming: Coping with Parallelism.* Technical Report RJ 5118. IBM Almaden Research Center.

Aaron Turon, Jacob Thamsborg, Amal Ahmed, Lars Birkedal, and Derek Dreyer. 2013. Logical Relations for Fine-Grained Concurrency. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2013)*. 343–356.

Aaron Turon and Mitchell Wand. 2011. A separation logic for refining concurrent objects. In *POPL*. 247–258.

A PROOFS OF ABSTRACTION THEOREM

In this section, we prove Theorem 6.2. We first define compositional operational semantics that can generate separate traces for objects and clients. Then, the definitions of contextual refinement, PSF, PDF and fairness conditions can all be reduced to equivalent ones based on object traces only. We also build simulations based on the object-local semantics.

A.1 Auxiliary Semantics

Before showing the compositional semantics, we first define the type of local traces $\widehat{T}$.

$$\begin{array}{llll} (\mathit{LEvent}) & \widehat{\iota} & ::= & (e, \Delta) \\ (\mathit{LTrace}) & \widehat{T} & ::= & \epsilon \mid \widehat{\iota} :: \widehat{T} \qquad \text{(co-inductive)} \end{array}$$

For any "global" trace T, we use get_clt(T) to get a client-local trace, which contains only client events (i.e., $(\mathrm{t}, \mathbf{out}, n)$, $(\mathrm{t}, \mathbf{clt})$, $(\mathrm{t}, \mathbf{term})$ and $(\mathrm{t}, \mathbf{clt}, \mathbf{abort})$) and history events (i.e., (t, f, n), $(\mathrm{t}, \mathbf{ret}, n)$ and $(\mathrm{t}, \mathbf{obj}, \mathbf{abort})$), with the accompanied Δ_c for each event. Similarly get_obj(T) gives an object-local trace, which contains only object events (i.e., $(\mathrm{t}, \mathbf{obj})$ and $(\mathrm{t}, \mathbf{obj}, \mathbf{abort})$) and history events, with the accompanied Δ_o for each event.

Client-local semantics. We define the client-local language as follows.

$$\begin{array}{llll} (\mathit{CLStmt}) & D & ::= & \hat{C} \mid \mathbf{incall}(x) \mid \mathbf{incall}(x); C \mid \mathbf{fstuck} \\ (\mathit{CLProg}) & \widehat{C} & ::= & D \,\|\, \ldots \,\|\, D \end{array}$$

$$\frac{(D_i,\sigma_c) \overset{e}{\circ\!\!\longrightarrow}_i (D_i',\sigma_c') \qquad \Delta_c = \text{btids-c}(D_1 \parallel \ldots D_i' \ldots \parallel D_n,\sigma_c')}{(D_1 \parallel \ldots D_i \ldots \parallel D_n,\sigma_c) \overset{(e,\Delta_c)}{\oplus\!\!\longrightarrow} (D_1 \parallel \ldots D_i' \ldots \parallel D_n,\sigma_c')}$$

$$\frac{D_i = \textbf{skip} \qquad D_i' = \textbf{end} \qquad e = (i,\textbf{term}) \qquad \Delta_c = \text{btids-c}(D_1 \parallel \ldots D_i \ldots \parallel D_n,\sigma_c)}{(D_1 \parallel \ldots D_i \ldots \parallel D_n,\sigma_c) \overset{(e,\Delta_c)}{\oplus\!\!\longrightarrow} (D_1 \parallel \ldots D_i' \ldots \parallel D_n,\sigma_c)}$$

$$\frac{(D_i,\sigma_c) \overset{e}{\circ\!\!\longrightarrow}_i \textbf{abort}}{(D_1 \parallel \ldots D_i \ldots \parallel D_n,\sigma_c) \overset{(e,\emptyset)}{\oplus\!\!\longrightarrow} \textbf{abort}}$$

(a) program transitions

$$\frac{[\![E]\!]_{s_c} = n \qquad x \in dom(s_c)}{(\mathbf{E}[\, x := f(E)\,],(s_c,h_c)) \overset{(\mathrm{t},f,n)}{\circ\!\!\longrightarrow}_{\mathrm{t}} (\mathbf{E}[\,\textbf{incall}(x)\,],(s_c,h_c))} \qquad \frac{[\![E]\!]_{s_c}\ \textit{undefined} \quad \text{or} \quad x \notin dom(s_c)}{(\mathbf{E}[\, x := f(E)\,],(s_c,h_c)) \overset{(\mathrm{t},\textbf{clt},\textbf{abort})}{\circ\!\!\longrightarrow}_{\mathrm{t}} \textbf{abort}}$$

$$\frac{}{(\mathbf{E}[\,\textbf{incall}(x)\,],\sigma_c) \overset{(\mathrm{t},\textbf{obj})}{\circ\!\!\longrightarrow}_{\mathrm{t}} (\mathbf{E}[\,\textbf{incall}(x)\,],\sigma_c)} \qquad \frac{}{(\mathbf{E}[\,\textbf{incall}(x)\,],\sigma_c) \overset{(\mathrm{t},\textbf{obj})}{\circ\!\!\longrightarrow}_{\mathrm{t}} (\textbf{fstuck},\sigma_c)}$$

$$\frac{}{(\mathbf{E}[\,\textbf{incall}(x)\,],\sigma_c) \overset{(\mathrm{t},\textbf{obj},\textbf{abort})}{\circ\!\!\longrightarrow}_{\mathrm{t}} \textbf{abort}} \qquad \frac{n \in Val \qquad s_c' = s_c\{x \rightsquigarrow n\}}{(\mathbf{E}[\,\textbf{incall}(x)\,],(s_c,h_c)) \overset{(\mathrm{t},\textbf{ret},n)}{\circ\!\!\longrightarrow}_{\mathrm{t}} (\mathbf{E}[\,\textbf{skip}\,],(s_c',h_c))}$$

$$\frac{[\![E]\!]_{s_c} = n}{(\mathbf{E}[\,\textbf{print}(E)\,],(s_c,h_c)) \overset{(\mathrm{t},\textbf{out},n)}{\circ\!\!\longrightarrow}_{\mathrm{t}} (\mathbf{E}[\,\textbf{skip}\,],(s_c,h_c))} \qquad \frac{(C,\sigma_c) \longrightarrow_{\mathrm{t}} (C',\sigma_c')}{(C,\sigma_c) \overset{(\mathrm{t},\textbf{clt})}{\circ\!\!\longrightarrow}_{\mathrm{t}} (C',\sigma_c')}$$

(b) thread transitions

$$\text{btids-c}(D_1 \parallel \ldots \parallel D_n,\sigma_c) \overset{\text{def}}{=} \{\mathrm{t} \mid \neg(\sigma_c \models \text{en}(D_{\mathrm{t}}))\}$$

Fig. 14. Client-local operational semantics rules.

A client-local thread D extends $\hat{C}$ with two new constructs **incall** and **fstuck**. The client-local operational semantics rules are shown in Fig. 14. When the client thread calls an object method, it goes to **incall**. **incall** nondeterministically returns, or loops, or leads to an object **abort** or **fstuck**. **fstuck** indicates that the method call blocks. It does not have next steps in the client-local semantics.

Then we define the client-local trace set $\mathcal{T}_\omega^{\mathrm{c}}[\![\widehat{C},\sigma_c]\!]$ as follows.

$$\begin{aligned}\mathcal{T}_\omega^{\mathrm{c}}[\![\widehat{C},\sigma_c]\!] \overset{\text{def}}{=} \{&((\textbf{spawn},|\widehat{C}|),\text{btids-c}(\widehat{C},\sigma_c)) :: \text{get\_clt}(\widehat{T}) \mid \\ &((\widehat{C},\sigma_c) \overset{\widehat{T}}{\oplus\!\!\longrightarrow}^{\omega} \cdot) \vee ((\widehat{C},\sigma_c) \overset{\widehat{T}}{\oplus\!\!\longrightarrow}^{*} \textbf{abort}) \\ &\vee \exists \widehat{C}',\sigma_c'.\ ((\widehat{C},\sigma_c) \overset{\widehat{T}}{\oplus\!\!\longrightarrow}^{*} (\widehat{C}',\sigma_c')) \wedge (\neg\exists \widehat{\iota}.\ (\widehat{C}',\sigma_c') \overset{\widehat{\iota}}{\oplus\!\!\longrightarrow} \_)\}\end{aligned}$$

$$|D_1 \parallel \ldots \parallel D_n| \overset{\text{def}}{=} n$$

Also, we write $\widehat{C}|_{\mathrm{t}}$ to get the code of the thread t:

$$(D_1 \parallel \ldots \parallel D_n)|_{\mathrm{t}} \overset{\text{def}}{=} D_{\mathrm{t}}$$

$$\frac{(O_i,(\sigma_o,\mathcal{K}(i))) \overset{e}{\diamond\!\!\longrightarrow}_{i,\Pi} (O_i',(\sigma_o',\kappa')) \quad \mathcal{K}' = \mathcal{K}\{i \rightsquigarrow \kappa'\} \quad \Delta_o = \text{btids-o}(O_1 \,\|\, \ldots O_i' \ldots \,\|\, O_n,(\sigma_o',\mathcal{K}'))}{(O_1 \,\|\, \ldots O_i \ldots \,\|\, O_n,(\sigma_o,\mathcal{K})) \overset{(e,\Delta_o)}{\diamond\!\!\longrightarrow}_{\Pi} (O_1 \,\|\, \ldots O_i' \ldots \,\|\, O_n,(\sigma_o',\mathcal{K}'))}$$

$$\frac{(O_i,(\sigma_o,\mathcal{K}(i))) \overset{e}{\diamond\!\!\longrightarrow}_{i,\Pi} \textbf{abort}}{(O_1 \,\|\, \ldots O_i \ldots \,\|\, O_n,(\sigma_o,\mathcal{K})) \overset{(e,\emptyset)}{\diamond\!\!\longrightarrow}_{\Pi} \textbf{abort}}$$

(a) program transitions

$$\frac{}{(\textbf{inclt},(\sigma_o,\circ)) \overset{(\text{t},\textbf{clt})}{\diamond\!\!\longrightarrow}_{\text{t},\Pi} (\textbf{inclt},(\sigma_o,\circ))} \qquad \frac{}{(\textbf{inclt},(\sigma_o,\circ)) \overset{(\text{t},\textbf{clt})}{\diamond\!\!\longrightarrow}_{\text{t},\Pi} (\textbf{cstuck},(\sigma_o,\circ))}$$

$$\frac{}{(\textbf{inclt},(\sigma_o,\circ)) \overset{(\text{t},\textbf{clt},\textbf{abort})}{\diamond\!\!\longrightarrow}_{\text{t},\Pi} \textbf{abort}} \qquad \frac{\Pi(f) = (\mathcal{P},y,C) \quad \sigma_o \in \mathcal{P} \quad n \in \mathit{Val} \quad \kappa = (\{y \rightsquigarrow n\},\_,\_)}{(\textbf{inclt},(\sigma_o,\circ)) \overset{(\text{t},f,n)}{\diamond\!\!\longrightarrow}_{\text{t},\Pi} (C,(\sigma_o,\kappa))}$$

$$\frac{\kappa = (s_l,\_,\_) \qquad [\![E]\!]_{s_l} = n}{(\textbf{E}[\,\textbf{return}\ E\,],(\sigma_o,\kappa)) \overset{(\text{t},\textbf{ret},n)}{\diamond\!\!\longrightarrow}_{\text{t},\Pi} (\textbf{inclt},(\sigma_o,\circ))} \qquad \frac{\kappa = (s_l,\_,\_) \qquad [\![E]\!]_{s_l}\ \mathit{undefined}}{(\textbf{E}[\,\textbf{return}\ E\,],(\sigma_o,\kappa)) \overset{(\text{t},\textbf{obj},\textbf{abort})}{\diamond\!\!\longrightarrow}_{\text{t},\Pi} \textbf{abort}}$$

$$\frac{(C,(s_o \uplus s_l,h_o)) \longrightarrow_{\text{t}} (C',(s_o' \uplus s_l',h_o')) \qquad \mathit{dom}(s_l) = \mathit{dom}(s_l')}{(C,((s_o,h_o),(s_l,\_,\_))) \overset{(\text{t},\textbf{obj})}{\diamond\!\!\longrightarrow}_{\text{t},\Pi} (C',((s_o',h_o'),(s_l',\_,\_)))}$$

(b) thread transitions

$$\text{btids-o}(O_1 \,\|\, \ldots \,\|\, O_n,(\sigma_o,\mathcal{K})) \overset{\text{def}}{=} \{\text{t} \mid \mathcal{K}(\text{t}) \neq \circ \wedge \neg((\sigma_o,\mathcal{K}(\text{t})) \models \text{en}(O_\text{t}))\}$$

Fig. 15. Object-local operational semantics rules.

Object-local semantics. We define the object-local language as follows:

$$\begin{array}{llll} (\mathit{OStmt}) & O & ::= & C \mid \textbf{inclt} \mid \textbf{cstuck} \\ (\mathit{OProg}) & \widetilde{C} & ::= & O \,\|\, \ldots \,\|\, O \end{array}$$

An object-local thread O extends C with two new constructs **inclt** and **cstuck**. We do not need the **end** flag since in object-local semantics we do not care about the termination of client threads.

The object-local operational semantics rules are shown in Fig. 15. A thread always starts executions from **inclt**. Then **inclt** nondeterministically calls an arbitrary method (with arbitrary arguments), or loops, or leads to a client **abort** or **cstuck**. **cstuck** indicates that the thread blocks inside the client code. It does not have next steps in the object-local semantics.

Then the object-local trace set $\mathcal{T}_\omega^{\text{o}}[\![\Pi,\sigma_o]\!]$ is defined as follows. To execute the object Π, we spawn an arbitrary number of object-local threads. Each thread starts its executions from **inclt**.

$$\begin{array}{lll} \mathcal{T}_\omega^{\text{o}}[\![\Pi,\sigma_o]\!] & \overset{\text{def}}{=} & \{((\textbf{spawn},n),\emptyset) :: \text{get\_obj}(\widehat{T}) \mid n \in \mathit{Nat} \\ & & \quad \wedge\ \exists \widetilde{C},O_1,\ldots,O_n.\ (\widetilde{C} = O_1 \,\|\, \ldots \,\|\, O_n) \wedge (\forall i.\ O_i = \textbf{inclt}) \\ & & \quad \wedge\ (((\widetilde{C},(\sigma_o,\circledcirc)) \overset{\widehat{T}}{\mapsto}_{\Pi}^{\omega} \cdot) \vee ((\widetilde{C},(\sigma_o,\circledcirc)) \overset{\widehat{T}}{\mapsto}_{\Pi}^{*} \textbf{abort}) \\ & & \qquad \vee\ \exists \widetilde{C}',\sigma_o',\mathcal{K}'.\ ((\widetilde{C},(\sigma_o,\circledcirc)) \overset{\widehat{T}}{\mapsto}_{\Pi}^{*} (\widetilde{C}',(\sigma_o',\mathcal{K}'))) \wedge (\neg\exists \widehat{\imath}.\ (\widetilde{C}',(\sigma_o',\mathcal{K}')) \overset{\widehat{\imath}}{\mapsto}_{\Pi} \_))\} \end{array}$$

Compositionality of the semantics. The following "Decomposition" and "Composition" Theorems describe the compositionality of the semantics. The Decomposition Theorem A.1 says that every trace T generated from the whole program using the semantics in Fig. 4 can be decomposed to a client-local trace $\widehat{T}_c$ and an object-local trace $\widehat{T}_o$. The Composition Theorem A.2 says that a client-local trace and an object-local trace, if they are coherent, can be composed to a full trace.

The coherence between $\widehat{T}_c$ and $\widehat{T}_o$ requires that the histories projected from them are the same. Besides, when both $\widehat{T}_c$ and $\widehat{T}_o$ are finite, we also require fin_coherent$(\widehat{T}_c, \widehat{T}_o)$ holds. We define fin_coherent as follows.

$$\begin{array}{l}\mathsf{fin\_coherent}(\widehat{T}_c, \widehat{T}_o)\ \mathsf{iff}\\ \quad \forall \mathsf{t}.\ (|\widehat{T}_c| \neq \omega) \wedge (|\widehat{T}_o| \neq \omega) \wedge \neg\mathsf{abt}(\widehat{T}_c) \wedge \neg\mathsf{abt}(\widehat{T}_o) \wedge \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T}_c)]\\ \quad \Longrightarrow (\mathsf{term\text{-}c}(\widehat{T}_c|_\mathsf{t}) \vee \mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}_c))) \ \wedge\ (\mathsf{term\text{-}o}(\widehat{T}_o|_\mathsf{t}) \vee \mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}_o)))\end{array}$$

$$\mathsf{term\text{-}c}(\widehat{T})\ \ \mathsf{iff}\ \ (\mathsf{evt}(\mathsf{last}(\widehat{T}|_\mathsf{t})) = (\mathsf{t}, \mathbf{term})) \vee \mathsf{is\_inv}(\mathsf{evt}(\mathsf{last}(\widehat{T})))$$

$$\mathsf{term\text{-}o}(\widehat{T})\ \ \mathsf{iff}\ \ \mathsf{is\_ret}(\mathsf{evt}(\mathsf{last}(\widehat{T}))) \vee |\widehat{T}| = 0$$

Theorem A.1 (Decomposition). *If* $T \in \mathcal{T}_\omega[\![\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \sigma_o, \circledcirc)]\!]$, *then there exist* $\widehat{T}_c$ *and* $\widehat{T}_o$ *such that*

$$\widehat{T}_c \in \mathcal{T}_\omega^c[\![C_1 \,\|\, \ldots \,\|\, C_n, \sigma_c]\!], \quad \widehat{T}_o \in \mathcal{T}_\omega^o[\![\Pi, \sigma_o]\!], \quad n = \mathsf{tnum}(\widehat{T}_o),$$
$$\widehat{T}_c = \mathsf{get\_clt}(T), \quad \widehat{T}_o = \mathsf{get\_obj}(T).$$

Theorem A.2 (Composition). *If* $\widehat{T}_c \in \mathcal{T}_\omega^c[\![C_1 \,\|\, \ldots \,\|\, C_n, \sigma_c]\!]$, $\widehat{T}_o \in \mathcal{T}_\omega^o[\![\Pi, \sigma_o]\!]$, $\mathsf{get\_hist}(\widehat{T}_c) = \mathsf{get\_hist}(\widehat{T}_o)$, $\mathsf{fin\_coherent}(\widehat{T}_c, \widehat{T}_o)$ *and* $n = \mathsf{tnum}(\widehat{T}_o)$, *then there exists* T *such that*

$$T \in \mathcal{T}_\omega[\![\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \sigma_o, \circledcirc)]\!], \quad \widehat{T}_c = \mathsf{get\_clt}(T), \quad \widehat{T}_o = \mathsf{get\_obj}(T).$$

Client-local and object-local fairness. We also define fairness conditions for client-local and object-local traces.

$$\mathsf{sfair\text{-}o}(\widehat{T})\ \ \mathsf{iff}\ \ \mathsf{abt}(\widehat{T}) \vee \forall \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T})].\ \mathsf{term\text{-}o}(\widehat{T}|_\mathsf{t}) \vee (\mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}))) \vee \mathsf{e\text{-}a\text{-}dis}(\mathsf{t}, \widehat{T}) \vee (|(\widehat{T}|_\mathsf{t})| = \omega)$$

$$\mathsf{wfair\text{-}o}(\widehat{T})\ \ \mathsf{iff}\ \ \mathsf{abt}(\widehat{T}) \vee \forall \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T})].\ \mathsf{term\text{-}o}(\widehat{T}|_\mathsf{t}) \vee (\mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}))) \vee \mathsf{i\text{-}o\text{-}dis}(\mathsf{t}, \widehat{T}) \vee (|(\widehat{T}|_\mathsf{t})| = \omega)$$

$$\mathsf{sfair\text{-}c}(\widehat{T})\ \ \mathsf{iff}\ \ \mathsf{abt}(\widehat{T}) \vee \forall \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T})].\ \mathsf{term\text{-}c}(\widehat{T}|_\mathsf{t}) \vee (\mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}))) \vee \mathsf{e\text{-}a\text{-}dis}(\mathsf{t}, \widehat{T}) \vee (|(\widehat{T}|_\mathsf{t})| = \omega)$$

$$\mathsf{wfair\text{-}c}(\widehat{T})\ \ \mathsf{iff}\ \ \mathsf{abt}(\widehat{T}) \vee \forall \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T})].\ \mathsf{term\text{-}c}(\widehat{T}|_\mathsf{t}) \vee (\mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}))) \vee \mathsf{i\text{-}o\text{-}dis}(\mathsf{t}, \widehat{T}) \vee (|(\widehat{T}|_\mathsf{t})| = \omega)$$

$$\mathsf{e\text{-}a\text{-}dis}(\mathsf{t}, \widehat{T})\ \ \mathsf{iff}\ \ \exists i.\ \forall j \geq i.\ \mathsf{t} \in \mathsf{bset}(\widehat{T}(j))$$

$$\mathsf{i\text{-}o\text{-}dis}(\mathsf{t}, \widehat{T})\ \ \mathsf{iff}\ \ \forall i.\ \exists j \geq i.\ \mathsf{t} \in \mathsf{bset}(\widehat{T}(j))$$

Notice the difference between the definitions of e-a-dis above and e-a-disabled in Fig. 6. If T is finite, e-a-disabled(t, T) may hold while e-a-dis(t, T) must not. Thus, we could rewrite the definition of well-blocked using e-a-dis as follows:

$$\begin{array}{l}\mathsf{well\text{-}blocked}(T, (W_a, \mathcal{S}_a)) \iff\\ \quad \exists T_a.\ T_a \in \mathcal{T}_\omega[\![W_a, \mathcal{S}_a]\!] \wedge (\mathsf{get\_hist}(T) = \mathsf{get\_hist}(T_a))\\ \quad \wedge\ (|T_a| = \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(T_a) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e), T_a)))\end{array}$$

The fairness conditions for the full traces T can also be rewritten in disjunctive forms, as the above fairness definitions for client-local and object-local traces. This is shown in the following Lemmas A.3 and A.4.

Lemma A.3. *Suppose* $T \in \mathcal{T}_\omega[\![\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \sigma_o, \circledcirc)]\!]$. *Then,* $\mathsf{sfair}(T)$ *is equivalent to* $\mathsf{abt}(T) \vee \forall \mathsf{t} \in [1..\mathsf{tnum}(T)].\ (\mathsf{evt}(\mathsf{last}(T|_\mathsf{t})) = (\mathsf{t}, \mathbf{term})) \vee (|(T|_\mathsf{t})| = \omega) \vee (\mathsf{t} \in \mathsf{bset}(\mathsf{last}(T))) \vee \mathsf{e\text{-}a\text{-}dis}(\mathsf{t}, T)$.

Lemma A.4. *Suppose* $T \in \mathcal{T}_\omega[\![\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \sigma_o, \circledcirc)]\!]$. *Then,* $\mathsf{wfair}(T)$ *is equivalent to*

$$\mathsf{abt}(T) \vee \forall \mathrm{t} \in [1..\mathsf{tnum}(T)].\ (\mathsf{evt}(\mathsf{last}(T|_{\mathrm{t}})) = (\mathrm{t}, \mathbf{term})) \vee (|(T|_{\mathrm{t}})| = \omega) \vee (\mathrm{t} \in \mathsf{bset}(\mathsf{last}(T))) \vee \text{i-o-dis}(\mathrm{t}, T).$$

Lemmas A.5, A.6, A.7 and Lemmas A.8, A.9, A.10 show the compositionality of fairness conditions. Finally, Lemma A.11 shows that e-a-dis for an object-local trace implies e-a-dis of the corresponding full trace.

Lemma A.5. *If* $T \in \mathcal{T}_\omega[\![\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n, (\sigma_c, \sigma_o, \circledcirc)]\!]$, $\mathsf{sfair}(T)$ *and* $\widehat{T_c} = \mathsf{get\_clt}(T)$, *then* $\mathsf{sfair\text{-}c}(\widehat{T_c})$.

Lemma A.6. *If* $T \in \mathcal{T}_\omega[\![\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n, (\sigma_c, \sigma_o, \circledcirc)]\!]$, $\mathsf{sfair}(T)$ *and* $\widehat{T_o} = \mathsf{get\_obj}(T)$, *then* $\mathsf{sfair\text{-}o}(\widehat{T_o})$.

Lemma A.7. *If* $T \in \mathcal{T}_\omega[\![\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n, (\sigma_c, \sigma_o, \circledcirc)]\!]$, $\widehat{T_c} = \mathsf{get\_clt}(T)$, $\widehat{T_o} = \mathsf{get\_obj}(T)$, $\mathsf{sfair\text{-}c}(\widehat{T_c})$ *and* $\mathsf{sfair\text{-}o}(\widehat{T_o})$, *then* $\mathsf{sfair}(T)$.

Lemma A.8. *If* $T \in \mathcal{T}_\omega[\![\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n, (\sigma_c, \sigma_o, \circledcirc)]\!]$, $\mathsf{wfair}(T)$ *and* $\widehat{T_c} = \mathsf{get\_clt}(T)$, *then* $\mathsf{wfair\text{-}c}(\widehat{T_c})$.

Lemma A.9. *If* $T \in \mathcal{T}_\omega[\![\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n, (\sigma_c, \sigma_o, \circledcirc)]\!]$, $\mathsf{wfair}(T)$ *and* $\widehat{T_o} = \mathsf{get\_obj}(T)$, *then* $\mathsf{wfair\text{-}o}(\widehat{T_c})$.

Lemma A.10. *If* $T \in \mathcal{T}_\omega[\![\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n, (\sigma_c, \sigma_o, \circledcirc)]\!]$, $\widehat{T_c} = \mathsf{get\_clt}(T)$, $\widehat{T_o} = \mathsf{get\_obj}(T)$, $\mathsf{wfair\text{-}c}(\widehat{T_c})$ *and* $\mathsf{wfair\text{-}o}(\widehat{T_o})$, *then* $\mathsf{wfair}(T)$.

Lemma A.11. *If* $T \in \mathcal{T}_\omega[\![\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n, (\sigma_c, \sigma_o, \circledcirc)]\!]$, $\widehat{T_o} = \mathsf{get\_obj}(T)$, $|\widehat{T_o}| = \omega$ *and* $\text{e-a-dis}(\mathrm{t}, \widehat{T_o})$, *then* $\text{e-a-dis}(\mathrm{t}, T)$.

A.2 Proofs of Theorem 6.2-1 (PSF under strong fairness)

By Theorem 4.4, the goal is reduced to the following:

$$\Pi \sqsubseteq^{\mathsf{fin}}_{\varphi} \Gamma \wedge \mathsf{PSF}^{\mathsf{sfair}}_{\varphi,\Gamma}(\Pi) \iff \Pi \sqsubseteq^{\mathsf{sfair}}_{\varphi} \Gamma\,.$$

We first define the object version of partial starvation-freedom.

Definition A.12. $\mathsf{PSF\text{-}O}^{\chi}_{\varphi,\Gamma}(\Pi)$, iff

$$\begin{aligned}&\forall n, \sigma, \Sigma, T.\ \ \widehat{T} \in \mathcal{T}^{\mathrm{o}}_{\omega}[\![\Pi, \sigma]\!] \wedge (\varphi(\sigma) = \Sigma) \wedge \chi\text{-o}(\widehat{T})\\ &\Longrightarrow \mathsf{abt}(\widehat{T}) \vee \text{prog-t}(\widehat{T}) \vee \text{well-blocked-o}(\widehat{T}, (\Gamma, \Sigma))\,.\end{aligned}$$

Here well-blocked-o is defined as follows:

$$\begin{aligned}&\text{well-blocked-o}(\widehat{T}, (\Gamma, \Sigma))\ \text{ iff}\\ &\quad \exists \widehat{T_a}.\ \widehat{T_a} \in \mathcal{T}^{\mathrm{o}}_{\omega}[\![\Gamma, \Sigma]\!] \wedge (\mathsf{get\_hist}(\widehat{T}) = \mathsf{get\_hist}(\widehat{T_a})) \wedge (\mathsf{tnum}(\widehat{T}) = \mathsf{tnum}(\widehat{T_a}))\\ &\qquad \wedge (|\widehat{T_a}| = \omega \Longrightarrow (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T_a}) \Longrightarrow \text{e-a-dis}(\mathsf{tid}(e), \widehat{T_a})))\\ &\qquad \wedge (|\widehat{T_a}| \neq \omega \Longrightarrow (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T_a}) \Longrightarrow \mathsf{tid}(e) \in \mathsf{bset}(\mathsf{last}(\widehat{T_a}))))\end{aligned}$$

We only need to prove the following lemmas.

Lemma A.13. $\Pi \sqsubseteq^{\chi}_{\varphi} \Gamma \Longrightarrow \Pi \sqsubseteq^{\mathsf{fin}}_{\varphi} \Gamma$.

Lemma A.14. $\Pi \sqsubseteq^{\mathsf{sfair}}_{\varphi} \Gamma \Longrightarrow \mathsf{PSF\text{-}O}^{\mathsf{sfair}}_{\varphi,\Gamma}(\Pi)$.

Lemma A.15. $\mathsf{PSF\text{-}O}^{\chi}_{\varphi,\Gamma}(\Pi) \iff \mathsf{PSF}^{\chi}_{\varphi,\Gamma}(\Pi)$.

Lemma A.16. $\Pi \sqsubseteq^{\mathsf{fin}}_{\varphi} \Gamma \wedge \mathsf{PSF\text{-}O}^{\mathsf{sfair}}_{\varphi,\Gamma}(\Pi) \Longrightarrow \Pi \sqsubseteq^{\mathsf{sfair}}_{\varphi} \Gamma$.

Proof of Lemma A.13. For any $n, C_1, \ldots, C_n, \sigma_c, \sigma$ and Σ such that $\varphi(\sigma) = \Sigma$, for any $\mathcal{E}$, if

$$\mathcal{E} \in O[\![(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma, \circledcirc)]\!],$$

we know there exists T_1 such that $\mathcal{E} = \text{get\_obsv}(T_1)$ and

$$T_1 \in \mathcal{T}[\![(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma, \circledcirc)]\!].$$

We can construct T_1' and T_1'' such that

$$T_1'' = T_1 :: T_1' \;\;,\;\; \chi(T_1'') \quad \text{and} \quad T_1'' \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma, \circledcirc)]\!].$$

Since $\Pi \sqsubseteq^{\chi}_{\varphi} \Gamma$, we know there exists T_2'' such that

$$T_2'' \in \mathcal{T}_\omega[\![(\textbf{let } \Gamma \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \Sigma, \circledcirc)]\!] \;\;,\;\; \chi(T_2'') \quad \text{and}$$
$$\text{get\_obsv}(T_2'') = \text{get\_obsv}(T_1'') = \mathcal{E} :: \text{get\_obsv}(T_1').$$

Thus there exists T_2 such that

$$T_2 \in \mathcal{T}[\![(\textbf{let } \Gamma \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \Sigma, \circledcirc)]\!] \quad \text{and} \quad \text{get\_obsv}(T_2) = \mathcal{E}.$$

Thus $\mathcal{E} \in O[\![(\textbf{let } \Gamma \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \Sigma, \circledcirc)]\!]$ and we are done. □

Proof of Lemma A.14. For any n, σ, Σ and $\widehat{T_o}$ such that $\widehat{T_o} \in \mathcal{T}_\omega^{\text{o}}[\![\Pi, \sigma]\!]$, sfair-o($\widehat{T_o}$) and $\varphi(\sigma) = \Sigma$, suppose

$$\neg\text{abt}(\widehat{T_o}) \quad \text{and} \quad \neg\text{prog-t}(\widehat{T_o}) \quad \text{and} \quad \neg\text{well-blocked-o}(\widehat{T_o}, (\Gamma, \Sigma)) \,.$$

Thus we know

$$(|\widehat{T_o}| \neq \omega) \implies \forall \mathsf{t} \in [1..\text{tnum}(\widehat{T_o})].\ \text{term-o}(\widehat{T_o}|_{\mathsf{t}}) \vee \mathsf{t} \in \text{bset}(\text{last}(\widehat{T_o})) \,.$$

Since $\widehat{T_o} \in \mathcal{T}_\omega^{\text{o}}[\![\Pi, \sigma]\!]$, we know there exist n and $\widehat{T_c}$ such that

$$\widehat{T_c} \in \mathcal{T}_\omega^{\text{c}}[\![\mathsf{MGCp1}_n, \sigma_{\mathsf{MGC}(n)}]\!], \quad \text{get\_hist}(\widehat{T_c}) = \text{get\_hist}(\widehat{T_o}), \quad n = \text{tnum}(\widehat{T_o}),$$
$$\neg\text{abt}(\widehat{T_c}), \quad \text{sfair-c}(\widehat{T_c}), \quad \text{fin\_coherent}(\widehat{T_c}, \widehat{T_o}) \,.$$

Here

$$\begin{array}{rcl}
\mathsf{MGTp1}_{\mathsf{t}} & \stackrel{\text{def}}{=} & \textbf{while } (\textbf{rand}() > 0)\{ \\
& & \quad x_{\mathsf{t}} := \textbf{rand}();\ y_{\mathsf{t}} := \textbf{rand}(m);\ \textbf{print}(0, y_{\mathsf{t}}, x_{\mathsf{t}}); \\
& & \quad z_{\mathsf{t}} := f_{y_{\mathsf{t}}}(x_{\mathsf{t}});\ \textbf{print}(1, z_{\mathsf{t}}); \\
& & \} \\
& & \textbf{print}(2); \\
\mathsf{MGCp1}_n & \stackrel{\text{def}}{=} & \|_{\mathsf{t} \in [1..n]}\, \mathsf{MGTp1}_{\mathsf{t}} \\
\sigma_{\mathsf{MGC}(n)} & \stackrel{\text{def}}{=} & \{x_{\mathsf{t}} \rightsquigarrow _, y_{\mathsf{t}} \rightsquigarrow _, z_{\mathsf{t}} \rightsquigarrow _ \mid 1 \leq \mathsf{t} \leq n\}
\end{array}$$

By Composition Theorem A.2, we know there exists T such that

$$T \in \mathcal{T}_\omega[\![\textbf{let } \Pi \textbf{ in } \mathsf{MGCp1}_n, (\sigma_{\mathsf{MGC}(n)}, \sigma, \circledcirc)]\!], \quad \widehat{T_c} = \text{get\_clt}(T), \quad \widehat{T_o} = \text{get\_obj}(T).$$

Since sfair-o($\widehat{T_o}$) and sfair-c($\widehat{T_c}$), by Lemma A.7, we know

$$\text{sfair}(T).$$

Since $\varphi(\sigma) = \Sigma$ and $\Pi \sqsubseteq^{\text{sfair}}_{\varphi} \Gamma$, we know:

$$O_{\text{sfair}}[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGCp1}_n), (\sigma_{\mathsf{MGC}(n)}, \sigma)]\!] \subseteq O_{\text{sfair}}[\![(\textbf{let } \Gamma \textbf{ in } \mathsf{MGCp1}_n), (\sigma_{\mathsf{MGC}(n)}, \Sigma)]\!] \,.$$

Thus there exists T' such that

$$T' \in \mathcal{T}_\omega[\![(\textbf{let } \Gamma \textbf{ in } \mathsf{MGCp1}_n), (\sigma_{\mathsf{MGC}(n)}, \Sigma, \circledcirc)]\!] \,, \quad \text{sfair}(T') \quad \text{and} \quad \text{get\_obsv}(T') = \text{get\_obsv}(T) \,.$$

Also, by the definition of $\mathsf{MGCp1}_n$ and the operational semantics, we can construct T_a and an execution such that

$$T_a \in \mathcal{T}_\omega[\![(\textbf{let } \Gamma \textbf{ in } \mathsf{MGCp1}_n), (\sigma_{\mathsf{MGC}(n)}, \Sigma, \circledcirc)]\!],$$
$$\text{sfair}(T_a), \quad \text{get\_obsv}(T_a) = \text{get\_obsv}(T'), \quad \text{get\_hist}(T) = \text{get\_hist}(T_a).$$

By Decomposition Theorem A.1, we know there exists $\widehat{T_a}$ such that

$$\widehat{T}_a \in \mathcal{T}_\omega^{\mathrm{o}}[\![\Gamma, \Sigma]\!], \quad \widehat{T}_a = \mathsf{get\_obj}(T_a), \quad n = \mathsf{tnum}(\widehat{T}_a).$$

Since $\mathsf{sfair}(T_a)$, by Lemma A.6, we know

$$\mathsf{sfair\text{-}o}(\widehat{T}_a).$$

Since $\widehat{T}_a = \mathsf{get\_obj}(T_a)$, $\widehat{T}_o = \mathsf{get\_obj}(T)$ and $\mathsf{get\_hist}(T) = \mathsf{get\_hist}(T_a)$, we know

$$\mathsf{get\_hist}(\widehat{T}_o) = \mathsf{get\_hist}(\widehat{T}_a)\,.$$

Since $n = \mathsf{tnum}(\widehat{T}_o)$ and $n = \mathsf{tnum}(\widehat{T}_a)$, we know

$$\mathsf{tnum}(\widehat{T}_o) = \mathsf{tnum}(\widehat{T}_a)\,.$$

- Suppose $|\widehat{T}_a| \neq \omega$. For any $e \in \mathsf{pend\_inv}(\widehat{T}_a)$ and $\mathrm{t}_0 = \mathsf{tid}(e)$, by the operational semantics and the code of Γ, since $\mathsf{sfair\text{-}o}(\widehat{T}_a)$, we know
$$\mathrm{t}_0 \in \mathsf{bset}(\mathsf{last}(\widehat{T}_a))\,.$$
Thus we know $\mathsf{well\text{-}blocked\text{-}o}(\widehat{T}_o, (\Gamma, \Sigma))$ holds, which contradicts our assumption.
- Suppose $|\widehat{T}_a| = \omega$. For any $e \in \mathsf{pend\_inv}(\widehat{T}_a)$ and $\mathrm{t}_0 = \mathsf{tid}(e)$, by the operational semantics and the code of Γ, we know $|(\widehat{T}_a|_{\mathrm{t}_0})| \neq \omega$. Since $\mathsf{sfair\text{-}o}(\widehat{T}_a)$, we know
$$\mathsf{e\text{-}a\text{-}dis}(\mathrm{t}_0, \widehat{T}_a)\,.$$
Thus we know $\mathsf{well\text{-}blocked\text{-}o}(\widehat{T}_o, (\Gamma, \Sigma))$ holds, which contradicts our assumption.

Thus we are done. □

Proof of Lemma A.15. The "$\Longrightarrow$" direction. For any n, C_1, …, C_n, σ_c, σ, Σ and T, suppose $T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma, \circledcirc)]\!]$, $\varphi(\sigma) = \Sigma$ and $\chi(T)$. If $\mathsf{abt}(T)$, then we are done. Otherwise, $\neg\mathsf{abt}(T)$ holds. By Decomposition Theorem A.1, we know there exist $\widehat{T}_c$ and $\widehat{T}_o$ such that

$$\widehat{T}_c \in \mathcal{T}_\omega^{\mathrm{c}}[\![C_1 \,\|\, \ldots \,\|\, C_n, \sigma_c]\!], \quad \widehat{T}_o \in \mathcal{T}_\omega^{\mathrm{o}}[\![\Pi, \sigma]\!],$$
$$\mathsf{get\_hist}(\widehat{T}_c) = \mathsf{get\_hist}(\widehat{T}_o), \quad \widehat{T}_c = \mathsf{get\_clt}(T), \quad \widehat{T}_o = \mathsf{get\_obj}(T), \quad n = \mathsf{tnum}(\widehat{T}_o).$$

Since $\chi(T)$, by Lemmas A.5 and A.6 (or Lemmas A.8 and A.9), we know

$$\chi\text{-}\mathsf{c}(\widehat{T}_c) \quad \text{and} \quad \chi\text{-}\mathsf{o}(\widehat{T}_o).$$

Since $\neg\mathsf{abt}(T)$ holds, we know $\neg\mathsf{abt}(\widehat{T}_o)$ and $\neg\mathsf{abt}(\widehat{T}_c)$ hold. Thus we know

$$(|\widehat{T}_c| \neq \omega) \implies \forall \mathrm{t} \in [1..\mathsf{tnum}(\widehat{T}_c)].\ \mathsf{term\text{-}c}(\widehat{T}_c|_{\mathrm{t}}) \vee \mathrm{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}_c))\,.$$

Since $\mathsf{PSF\text{-}O}^{\chi}_{\varphi,\Gamma}(\Pi)$, we know $\mathsf{prog\text{-}t}(\widehat{T}_o) \vee \mathsf{well\text{-}blocked\text{-}o}(\widehat{T}_o, (\Gamma, \Sigma))$.

- $\mathsf{prog\text{-}t}(\widehat{T}_o)$ holds. By the following Lemma A.17, we know $\mathsf{prog\text{-}t}(T)$.
- $\mathsf{well\text{-}blocked\text{-}o}(\widehat{T}_o, (\Gamma, \Sigma))$ holds. Thus there exists $\widehat{T}_a$ such that $\widehat{T}_a \in \mathcal{T}_\omega^{\mathrm{o}}[\![\Gamma, \Sigma]\!]$, $\mathsf{get\_hist}(\widehat{T}_o) = \mathsf{get\_hist}(\widehat{T}_a)$, $\mathsf{tnum}(\widehat{T}_o) = \mathsf{tnum}(\widehat{T}_a)$, $|\widehat{T}_a| = \omega \implies (\forall e.\, e \in \mathsf{pend\_inv}(\widehat{T}_a) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e), \widehat{T}_a))$ and $|\widehat{T}_a| \neq \omega \implies (\forall e.\, e \in \mathsf{pend\_inv}(\widehat{T}_a) \implies \mathsf{tid}(e) \in \mathsf{bset}(\mathsf{last}(\widehat{T}_a)))$. Thus we know
$$(|\widehat{T}_a| \neq \omega) \implies \forall \mathrm{t} \in [1..\mathsf{tnum}(\widehat{T}_a)].\ \mathsf{term\text{-}o}(\widehat{T}_a|_{\mathrm{t}}) \vee \mathrm{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}_a))\,.$$
Thus $\mathsf{fin\_coherent}(\widehat{T}_c, \widehat{T}_a)$ holds. Since $\mathsf{get\_hist}(\widehat{T}_c) = \mathsf{get\_hist}(\widehat{T}_o)$ and $n = \mathsf{tnum}(\widehat{T}_o)$, we know
$$\mathsf{get\_hist}(\widehat{T}_c) = \mathsf{get\_hist}(\widehat{T}_a) \quad \text{and} \quad n = \mathsf{tnum}(\widehat{T}_a).$$
Since $\widehat{T}_c \in \mathcal{T}_\omega^{\mathrm{c}}[\![C_1 \,\|\, \ldots \,\|\, C_n, \sigma_c]\!]$, by Composition Theorem A.2, we know there exists T' such that
$$T' \in \mathcal{T}_\omega[\![\mathbf{let}\ \Gamma\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \Sigma, \circledcirc)]\!], \quad \widehat{T}_c = \mathsf{get\_clt}(T'), \quad \widehat{T}_a = \mathsf{get\_obj}(T').$$
Since $\widehat{T}_o = \mathsf{get\_obj}(T)$, we know
$$\mathsf{get\_hist}(T) = \mathsf{get\_hist}(T').$$
Suppose $|T'| = \omega$. For any e such that $e \in \mathsf{pend\_inv}(T')$, we know $e \in \mathsf{pend\_inv}(\widehat{T}_a)$.

- Suppose $|\widehat{T}_a| = \omega$. Thus e-a-dis(tid(e), $\widehat{T}_a$)) holds. Then, by Lemma A.11, we know e-a-dis(tid(e), T')) holds.
- Suppose $|\widehat{T}_a| \neq \omega$. Thus we know
$$\mathsf{tid}(e) \in \mathsf{bset}(\mathsf{last}(\widehat{T}_a)).$$
Since $|T'| = \omega$ and $\widehat{T}_a = \mathsf{get\_obj}(T')$, we know there exists i such that
$$\widehat{T}_a = \mathsf{get\_obj}(T'(1..i)), \quad \mathsf{tid}(e) \in \mathsf{bset}(T'(i)).$$
By the operational semantics, we know
$$\forall j \geq i.\ \mathsf{tid}(e) \in \mathsf{bset}(T'(j)).$$
Thus e-a-dis(tid(e), T')).

Thus well-blocked(T, (**let** Γ **in** $C_1 \parallel \ldots \parallel C_n, (\sigma_c, \Sigma, \circledcirc)$)) holds.

Thus we have proved the "$\Longrightarrow$" direction.

The "$\Longleftarrow$" direction. By unfolding the definitions. □

Lemma A.17. *For any T and $\widehat{T}_o$, if $\widehat{T}_o = \mathsf{get\_obj}(T)$ and* prog-t($\widehat{T}_o$), *then* prog-t(T).

Proof. By unfolding the definitions. □

Proof of Lemma A.16. The key is to show the following (A.1).

For any $n, C_1, \ldots, C_n, \sigma_c, \sigma, \Sigma, T$ and T_a such that $\varphi(\sigma) = \Sigma$, if $T \in \mathcal{T}_\omega[\![\textbf{let } \Pi \textbf{ in } C_1 \parallel \ldots \parallel C_n, (\sigma_c, \sigma, \circledcirc)]\!]$, $\neg\mathsf{abt}(T)$ and $\mathsf{sfair}(T)$, then there exists T_a such that $T_a \in \mathcal{T}_\omega[\![\textbf{let } \Gamma \textbf{ in } C_1 \parallel \ldots \parallel C_n, (\sigma_c, \Sigma, \circledcirc)]\!]$, $\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$ and $\mathsf{sfair}(T_a)$. (A.1)

Since $T \in \mathcal{T}_\omega[\![\textbf{let } \Pi \textbf{ in } C_1 \parallel \ldots \parallel C_n, (\sigma_c, \sigma, \circledcirc)]\!]$ and $\neg\mathsf{abt}(T)$, by Decomposition Theorem A.1, we know there exist $\widehat{T}_c$ and $\widehat{T}_o$ such that

$$\widehat{T}_c \in \mathcal{T}_\omega^{\mathrm{c}}[\![C_1 \parallel \ldots \parallel C_n, \sigma_c]\!], \quad \widehat{T}_o \in \mathcal{T}_\omega^{\mathrm{o}}[\![\Pi, \sigma]\!],$$
$$\mathsf{get\_hist}(\widehat{T}_c) = \mathsf{get\_hist}(\widehat{T}_o), \quad \widehat{T}_c = \mathsf{get\_clt}(T), \quad \widehat{T}_o = \mathsf{get\_obj}(T), \quad n = \mathsf{tnum}(\widehat{T}_o).$$

Since $\mathsf{sfair}(T)$, by Lemmas A.5 and A.6, we know

$$\mathsf{sfair\text{-}c}(\widehat{T}_c) \quad \text{and} \quad \mathsf{sfair\text{-}o}(\widehat{T}_o).$$

Thus we know

$$(|\widehat{T}_c| \neq \omega) \implies \forall \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T}_c)].\ \mathsf{term\text{-}c}(\widehat{T}_c|_{\mathsf{t}}) \vee \mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}_c)) .$$

Since PSF-O$^{\mathsf{sfair}}_{\varphi,\Gamma}(\Pi)$, we know prog-t($\widehat{T}_o$) $\vee$ well-blocked-o($\widehat{T}_o$, (Γ, Σ)) holds.

- prog-t($\widehat{T}_o$) holds. Since $\Pi \sqsubseteq^{\mathsf{fin}}_{\varphi} \Gamma$, we know
$$\mathcal{H}[\![\Pi, \sigma]\!] \subseteq \mathcal{H}[\![\Gamma, \Sigma]\!].$$
By Lemma A.18, we know there exists $\widehat{T}_a$ such that
$$\widehat{T}_a \in \mathcal{T}_\omega^{\mathrm{o}}[\![\Gamma, \Sigma]\!], \quad \mathsf{get\_hist}(\widehat{T}_a) = \mathsf{get\_hist}(\widehat{T}_o), \quad \mathsf{tnum}(\widehat{T}_a) = \mathsf{tnum}(\widehat{T}_o).$$
Thus $\mathsf{get\_hist}(\widehat{T}_c) = \mathsf{get\_hist}(\widehat{T}_a)$, $n = \mathsf{tnum}(\widehat{T}_a)$ and $\neg\mathsf{abt}(\widehat{T}_a)$. Also, by Lemma A.19, we know
$$\text{prog-t}(\widehat{T}_a).$$
Thus
$$(|\widehat{T}_a| \neq \omega) \implies \forall \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T}_a)].\ \mathsf{term\text{-}o}(\widehat{T}_a|_{\mathsf{t}}) .$$
Thus fin_coherent($\widehat{T}_c, \widehat{T}_a$) holds. By Composition Theorem A.2, we know there exists T_a such that
$$T_a \in \mathcal{T}_\omega[\![\textbf{let } \Gamma \textbf{ in } C_1 \parallel \ldots \parallel C_n, (\sigma_c, \Sigma, \circledcirc)]\!], \quad \widehat{T}_c = \mathsf{get\_clt}(T_a), \quad \widehat{T}_a = \mathsf{get\_obj}(T_a).$$

Thus

$$\text{get\_obsv}(T) = \text{get\_obsv}(T_a).$$

By Lemma A.20, we know

$$\text{sfair-o}(\widehat{T}_a).$$

Then, by Lemma A.7, we know

$$\text{sfair}(T_a).$$

- well-blocked-o$(\widehat{T}_o, (\Gamma, \Sigma))$ holds. Thus there exists $\widehat{T}_a$ such that

$$\widehat{T}_a \in \mathcal{T}_\omega^o[\![\Gamma, \Sigma]\!], \quad \text{get\_hist}(\widehat{T}) = \text{get\_hist}(\widehat{T}_a), \quad \text{tnum}(\widehat{T}) = \text{tnum}(\widehat{T}_a),$$
$$|\widehat{T}_a| = \omega \implies (\forall e.\ e \in \text{pend\_inv}(\widehat{T}_a) \implies \text{e-a-dis}(\text{tid}(e), \widehat{T}_a)),$$
$$|\widehat{T}_a| \neq \omega \implies (\forall e.\ e \in \text{pend\_inv}(\widehat{T}_a) \implies \text{tid}(e) \in \text{bset}(\text{last}(\widehat{T}_a))).$$

Thus we know

$$(|\widehat{T}_a| \neq \omega) \implies \forall \text{t} \in [1..\text{tnum}(\widehat{T}_a)].\ \text{term-o}(\widehat{T}_a|_\text{t}) \vee \text{t} \in \text{bset}(\text{last}(\widehat{T}_a))\ .$$

Thus $\text{fin\_coherent}(\widehat{T}_c, \widehat{T}_a)$ holds. Also, $\text{get\_hist}(\widehat{T}_c) = \text{get\_hist}(\widehat{T}_a)$, $n = \text{tnum}(\widehat{T}_a)$ and $\neg\text{abt}(\widehat{T}_a)$. By Composition Theorem A.2, we know there exists T_a such that

$$T_a \in \mathcal{T}_\omega[\![\textbf{let}\ \Gamma\ \textbf{in}\ C_1 \parallel \ldots \parallel C_n, (\sigma_c, \Sigma, \circledcirc)]\!], \quad \widehat{T}_c = \text{get\_clt}(T_a), \quad \widehat{T}_a = \text{get\_obj}(T_a).$$

Thus

$$\text{get\_obsv}(T) = \text{get\_obsv}(T_a).$$

By Lemma A.21, we know

$$\text{sfair-o}(\widehat{T}_a).$$

Then, by Lemma A.7, we know

$$\text{sfair}(T_a).$$

Thus we are done. □

Lemma A.18. *If* $\mathcal{H}[\![\Pi, \sigma]\!] \subseteq \mathcal{H}[\![\Gamma, \Sigma]\!]$, $\widehat{T}_o \in \mathcal{T}_\omega^o[\![\Pi, \sigma]\!]$ *and* $\text{prog-t}(\widehat{T}_o)$, *then there exists* $\widehat{T}_a$ *such that* $\widehat{T}_a \in \mathcal{T}_\omega^o[\![\Gamma, \Sigma]\!]$, $\text{get\_hist}(\widehat{T}_a) = \text{get\_hist}(\widehat{T}_o)$ *and* $\text{tnum}(\widehat{T}_a) = \text{tnum}(\widehat{T}_o)$.

Proof. By constructing simulations. □

Lemma A.19. *If* $\text{get\_hist}(\widehat{T}_1) = \text{get\_hist}(\widehat{T}_2)$ *and* $\text{prog-t}(\widehat{T}_1)$, *then* $\text{prog-t}(\widehat{T}_2)$.

Proof. By unfolding the definitions. □

Lemma A.20. *If* $\text{prog-t}(\widehat{T})$ *and* $\widehat{T} \in \mathcal{T}_\omega^o[\![\Pi, \sigma]\!]$, *then* $\text{sfair-o}(\widehat{T})$.

Proof. For any $\text{t} \in [1..\text{tnum}(\widehat{T})]$, we know either $|(\widehat{T}|_\text{t})| = \omega$ or $|(\widehat{T}|_\text{t})| \neq \omega$. If $|(\widehat{T}|_\text{t})| = \omega$, we are done. Otherwise, since $\widehat{T} \in \mathcal{T}_\omega^o[\![\Pi, \Sigma]\!]$, we know

$$|(\widehat{T}|_\text{t})| = 0 \vee \text{is\_ret}(\text{evt}(\text{last}(\widehat{T}|_\text{t}))) \vee \text{is\_inv}(\text{evt}(\text{last}(\widehat{T}|_\text{t}))) \vee \text{evt}(\text{last}(\widehat{T}|_\text{t})) = (\text{t}, \textbf{obj}).$$

If $|(\widehat{T}|_\text{t})| = 0 \vee \text{is\_ret}(\text{evt}(\text{last}(\widehat{T}|_\text{t})))$ holds, then we are done. Otherwise, we know there exists e such that $e \in \text{pend\_inv}(\widehat{T}|_\text{t})$. Thus $e \in \text{pend\_inv}(\widehat{T})$ holds, which contradicts the premise $\text{prog-t}(\widehat{T})$. Thus we are done. □

Lemma A.21. *If* $|\widehat{T}| = \omega \implies (\forall e.\ e \in \text{pend\_inv}(\widehat{T}) \implies \text{e-a-dis}(\text{tid}(e), \widehat{T}) \vee (|(\widehat{T}|_{\text{tid}(e)})| = \omega))$, $|\widehat{T}| \neq \omega \implies (\forall e.\ e \in \text{pend\_inv}(\widehat{T}) \implies \text{tid}(e) \in \text{bset}(\text{last}(\widehat{T})))$ *and* $\widehat{T} \in \mathcal{T}_\omega^o[\![\Pi, \Sigma]\!]$, *then* $\text{sfair-o}(\widehat{T})$.

Proof. Suppose $\neg\mathsf{abt}(\widehat{T})$. For any $\mathsf{t} \in [1..\mathsf{tnum}(\widehat{T})]$, we know either $|(\widehat{T}|_\mathsf{t})| = \omega$ or $|(\widehat{T}|_\mathsf{t})| \neq \omega$. If $|(\widehat{T}|_\mathsf{t})| = \omega$, we are done. Otherwise, since $\widehat{T} \in \mathcal{T}_\omega^\circ[\![\Pi, \Sigma]\!]$, we know

$$|(\widehat{T}|_\mathsf{t})| = 0 \vee \mathsf{is\_ret}(\mathsf{evt}(\mathsf{last}(\widehat{T}|_\mathsf{t}))) \vee \mathsf{is\_inv}(\mathsf{evt}(\mathsf{last}(\widehat{T}|_\mathsf{t}))) \vee \mathsf{evt}(\mathsf{last}(\widehat{T}|_\mathsf{t})) = (\mathsf{t}, \mathbf{obj}).$$

If $|(\widehat{T}|_\mathsf{t})| = 0 \vee \mathsf{is\_ret}(\mathsf{evt}(\mathsf{last}(\widehat{T}|_\mathsf{t})))$ holds, then we are done. Otherwise, we know there exists e such that $e \in \mathsf{pend\_inv}(\widehat{T}|_\mathsf{t})$. Thus $\mathsf{e\text{-}a\text{-}dis}(\mathsf{t}, \widehat{T})$ or $\mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}))$ holds. Thus we are done. □

A.3 Proofs of Theorem 6.2-2 (PSF under weak fairness)

By Theorem 4.4, the goal is reduced to the following:

$$\Pi \sqsubseteq_\varphi^{\mathsf{fin}} \Gamma \wedge \mathsf{PSF}_{\varphi,\Gamma}^{\mathsf{wfair}}(\Pi) \iff \Pi \sqsubseteq_{\mathsf{wr}_{\mathsf{PSF}}(\varphi)}^{\mathsf{wfair}} \mathsf{wr}_{\mathsf{PSF}}^{\mathsf{wfair}}(\Gamma).$$

By Lemmas A.13 and A.15, we only need to prove the following lemmas.

Lemma A.22. $\mathsf{wr}_{\mathsf{PSF}}^{\mathsf{wfair}}(\Gamma) \sqsubseteq_{\mathsf{wr}_{\mathsf{PSF}}^{-1}(id)}^{\mathsf{wfair}} \Gamma$. *Here id is the identity function, and* $\mathsf{wr}_{\mathsf{PSF}}^{-1}$ *is a state mapping that removes the variable* `listid` *from the state:* $\forall\sigma.\ (\mathsf{wr}_{\mathsf{PSF}}^{-1}(id))(\sigma \uplus \{\mathtt{listid} \rightsquigarrow \epsilon\}) = \sigma$.

Lemma A.23. $\Pi \sqsubseteq_{\mathsf{wr}_{\mathsf{PSF}}(\varphi)}^{\mathsf{wfair}} \mathsf{wr}_{\mathsf{PSF}}^{\mathsf{wfair}}(\Gamma) \implies \mathsf{PSF\text{-}O}_{\varphi,\Gamma}^{\mathsf{wfair}}(\Pi)$.

Lemma A.24. $\Pi \sqsubseteq_\varphi^{\mathsf{fin}} \Gamma \wedge \mathsf{PSF\text{-}O}_{\varphi,\Gamma}^{\mathsf{wfair}}(\Pi) \implies \Pi \sqsubseteq_{\mathsf{wr}_{\mathsf{PSF}}(\varphi)}^{\mathsf{wfair}} \mathsf{wr}_{\mathsf{PSF}}^{\mathsf{wfair}}(\Gamma)$.

Proof of Lemma A.22. For any $n, C_1, \ldots, C_n, \sigma_c$ and Σ, for any $\mathcal{E}$, if

$$\mathcal{E} \in O_{\mathsf{wfair}}[\![(\mathbf{let}\ \mathsf{wr}_{\mathsf{PSF}}^{\mathsf{wfair}}(\Gamma)\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \Sigma \uplus \{\mathtt{listid} \rightsquigarrow \epsilon\}, \circledcirc)]\!],$$

we know there exists T_2 such that

$$T_2 \in \mathcal{T}_\omega[\![(\mathbf{let}\ \mathsf{wr}_{\mathsf{PSF}}^{\mathsf{wfair}}(\Gamma)\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \Sigma \uplus \{\mathtt{listid} \rightsquigarrow \epsilon\}, \circledcirc)]\!],$$
$$\mathsf{wfair}(T_2) \quad \text{and} \quad \mathsf{get\_obsv}(T_2) = \mathcal{E}.$$

By Decomposition Theorem A.1, we know there exist $\widehat{T_c}$ and $\widehat{T_2}$ such that

$$\widehat{T_c} \in \mathcal{T}_\omega^{\mathsf{c}}[\![C_1 \,\|\, \ldots \,\|\, C_n, \sigma_c]\!], \quad \widehat{T_2} \in \mathcal{T}_\omega^\circ[\![\mathsf{wr}_{\mathsf{PSF}}^{\mathsf{wfair}}(\Gamma), \Sigma \uplus \{\mathtt{listid} \rightsquigarrow \epsilon\}]\!],$$
$$\mathsf{get\_hist}(\widehat{T_c}) = \mathsf{get\_hist}(\widehat{T_2}), \quad \widehat{T_c} = \mathsf{get\_clt}(T_2), \quad \widehat{T_2} = \mathsf{get\_obj}(T_2), \quad n = \mathsf{tnum}(\widehat{T_2}).$$

By Lemmas A.8 and A.9, we know

$$\mathsf{wfair\text{-}c}(\widehat{T_c}) \quad \text{and} \quad \mathsf{wfair\text{-}o}(\widehat{T_2}).$$

Thus we know

$$(|\widehat{T_c}| \neq \omega) \wedge (\neg\mathsf{abt}(\widehat{T_3})) \implies \forall \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T_c})].\ \mathsf{term\text{-}c}(\widehat{T_c}|_\mathsf{t}) \vee \mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T_c})).$$

Then we prove there exists $\widehat{T_3}$ such that

$$\widehat{T_3} \in \mathcal{T}_\omega^\circ[\![\Gamma, \Sigma]\!], \quad \mathsf{get\_hist}(\widehat{T_2}) = \mathsf{get\_hist}(\widehat{T_3}), \quad \mathsf{wfair\text{-}o}(\widehat{T_3}).$$

The proof is done by constructing a simulation between the object-local executions of $\mathsf{wr}_{\mathsf{PSF}}^{\mathsf{wfair}}(\Gamma)$ and Γ. We need the simulation $\lesssim$ to satisfy the following (A.2).

For any $\widetilde{C}_1, \Sigma_1, \mathcal{K}_1, \widetilde{C}, \Sigma, \mathcal{K}$ and $\widehat{\iota}_1$, if $(\widetilde{C}_1, (\Sigma_1, \mathcal{K}_1)) \lesssim (\widetilde{C}, (\Sigma, \mathcal{K}))$, then

(1) if $(\widetilde{C}_1, (\Sigma_1, \mathcal{K})) \overset{\widehat{\iota}_1}{\Longmapsto}_{\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\Gamma)}$ **abort**, then

there exists $\widehat{T}$ such that $(\widetilde{C}, (\Sigma, \mathcal{K})) \overset{\widehat{T}}{\Longmapsto}^*_\Gamma$ **abort** and $\mathsf{get\_clt}(\widehat{\iota}_1) = \mathsf{get\_clt}(\widehat{T})$;

(2) if $(\widetilde{C}_1, (\Sigma_1, \mathcal{K})) \overset{\widehat{\iota}_1}{\Longmapsto}_{\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\Gamma)} (\widetilde{C}'_1, (\Sigma'_1, \mathcal{K}'))$, then

there exist $\widehat{T}$, $\widetilde{C}'$, Σ' and $\mathcal{K}'$ such that $(\widetilde{C}, (\Sigma, \mathcal{K})) \overset{\widehat{T}}{\Longmapsto}^*_\Gamma (\widetilde{C}', (\Sigma', \mathcal{K}'))$,
$\mathsf{get\_clt}(\widehat{\iota}_1) = \mathsf{get\_clt}(\widehat{T})$ and $(\widetilde{C}'_1, (\Sigma'_1, \mathcal{K}')) \lesssim (\widetilde{C}', (\Sigma', \mathcal{K}'))$;

(3) if $\neg(\exists \widehat{\iota}_1.\ (\widetilde{C}_1, (\Sigma_1, \mathcal{K})) \overset{\widehat{\iota}_1}{\longmapsto}_{\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\Gamma)} \_)$, then $\neg(\exists \widehat{\iota}.\ (\widetilde{C}, (\Sigma, \mathcal{K})) \overset{\widehat{\iota}}{\longmapsto}_\Gamma \_)$;

(4) if $\neg(\exists \widehat{\iota}.\ (\widetilde{C}, (\Sigma, \mathcal{K})) \overset{\widehat{\iota}}{\longmapsto}_\Gamma \_)$, then

there exists $\widetilde{C}'_1$, Σ'_1, $\mathcal{K}'_1$ and $\widehat{T}_1$ such that $(\widetilde{C}_1, (\Sigma_1, \mathcal{K}_1)) \overset{\widehat{T}_1}{\Longmapsto}^*_{\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\Gamma)} (\widetilde{C}'_1, (\Sigma'_1, \mathcal{K}'_1))$ and

$\neg(\exists \widehat{\iota}_1.\ (\widetilde{C}'_1, (\Sigma'_1, \mathcal{K}'_1)) \overset{\widehat{\iota}_1}{\longmapsto}_{\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\Gamma)} \_)$. (A.2)

The simulation relation $\lesssim$ is constructed as follows.

$$(\widetilde{C}_1, (\Sigma_1, \mathcal{K}_1)) \lesssim (\widetilde{C}, (\Sigma, \mathcal{K})) \text{ iff } (\widetilde{C}_1 \backsim_{\mathcal{K}} \widetilde{C}) \wedge (\Sigma = \Sigma_1 \backslash \{\mathtt{listid}\}) \wedge (\mathcal{K}_1 = \mathcal{K})$$

Here $\widetilde{C}_1 \backsim_{\mathcal{K}} \widetilde{C}$ requires the following hold:

$\forall \mathsf{t}.\ (\mathcal{K}(\mathsf{t}) = \circ) \Longrightarrow (\widetilde{C}_1|_\mathsf{t} = \widetilde{C}|_\mathsf{t})$,

$\forall \mathsf{t}.\ (\mathcal{K}(\mathsf{t}) \neq \circ) \Longrightarrow (\widetilde{C}_1|_\mathsf{t} = \widetilde{C}|_\mathsf{t}) \vee \exists B, C.\ (\widetilde{C}_1|_\mathsf{t} = \mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\mathbf{await}(B)\{C\})) \wedge (\widetilde{C}|_\mathsf{t} = \mathbf{await}(B)\{C\}) \vee$
$(\widetilde{C}_1|_\mathsf{t} = \mathbf{await}(B \wedge \mathtt{cid} = \mathbf{enhd}(\mathtt{listid}))\{C; \mathtt{listid} := \mathtt{listid} \backslash \mathtt{cid};\}) \wedge (\widetilde{C}|_\mathsf{t} = \mathbf{await}(B)\{C\})$.

To prove (A.2), we make a case-split on the derivation of $\widehat{\iota}_1$.

- If $\widehat{\iota}_1$ is an abort event, then we could generate the same abort event at the next step of $(\widetilde{C}, (\Sigma, \mathcal{K}))$.
- If $\widehat{\iota}_1$ is a client event, then we could generate the same client event at the next step of $(\widetilde{C}, (\Sigma, \mathcal{K}))$.
- If $\widehat{\iota}_1$ is an invocation event, then we could generate the same invocation event at the next step of $(\widetilde{C}, (\Sigma, \mathcal{K}))$.
- If $\widehat{\iota}_1$ is an object event and the step is executing $\mathtt{listid} := \mathtt{listid} +\!\!+ (\mathtt{cid}, B)$, then we execute zero step of $(\widetilde{C}, (\Sigma, \mathcal{K}))$.
- If $\widehat{\iota}_1$ is an object event and the step is executing $\mathbf{await}(B \wedge \mathtt{cid} = \mathbf{enhd}(\mathtt{listid}))\{C; \mathtt{listid} := \mathtt{listid} \backslash \mathtt{cid};\}$, then we could generate the same object event at the next step of $(\widetilde{C}, (\Sigma, \mathcal{K}))$ and that step is executing $\mathbf{await}(B)\{C\}$.
- If $\widehat{\iota}_1$ is a return event, then we could generate the same return event at the next step of $(\widetilde{C}, (\Sigma, \mathcal{K}))$.

Also, from the construction of the simulation, we know the cases (3) and (4) of (A.2) hold. Thus, we have proved (A.2). Then, we can find $\widehat{T}_3$ such that $\widehat{T}_3 \in \mathcal{T}^{\mathsf{o}}_\omega[\![\Gamma, \Sigma]\!]$ and $\mathsf{get\_hist}(\widehat{T}_2) = \mathsf{get\_hist}(\widehat{T}_3)$ hold.

Below we prove $\mathsf{wfair\text{-}o}(\widehat{T}_3)$. Since $\mathsf{wfair\text{-}o}(\widehat{T}_2)$, we know either $\mathsf{abt}(\widehat{T}_2)$, or for any t,

$$\mathsf{term\text{-}o}(\widehat{T}_2|_\mathsf{t}), \quad \text{or} \quad \mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}_2)), \quad \text{or} \quad \mathsf{i\text{-}o\text{-}dis}(\mathsf{t}, \widehat{T}_2), \quad \text{or} \quad |(\widehat{T}_2|_\mathsf{t})| = \omega.$$

If $\mathsf{abt}(\widehat{T}_2)$, we know $\mathsf{abt}(\widehat{T}_3)$. Otherwise, for any t,

- If $\mathsf{term\text{-}o}(\widehat{T}_2|_\mathsf{t})$ holds, we know $\mathsf{term\text{-}o}(\widehat{T}_3|_\mathsf{t})$.
- If $|(\widehat{T}_2|_\mathsf{t})| = \omega$ holds, we know $|(\widehat{T}_3|_\mathsf{t})| = \omega$.

- If $\mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}_2))$ holds, we know $\mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}_3))$.
- If $|(\widehat{T}_2|_\mathsf{t})| \neq \omega$ and $\mathsf{i\text{-}o\text{-}dis}(\mathsf{t}, \widehat{T}_2)$, we want to prove $\mathsf{i\text{-}o\text{-}dis}(\mathsf{t}, \widehat{T}_3)$. Since $|\widehat{T}_2| = \omega$, we know $|\widehat{T}_3| = \omega$. Suppose $\mathsf{i\text{-}o\text{-}dis}(\mathsf{t}, \widehat{T}_3)$ does not hold. Then $\mathsf{e\text{-}a\text{-}enabled}(\mathsf{t}, T_3)$ holds. We make a case-split on the last event of $\widehat{T}_2|_\mathsf{t}$:
  - $\mathsf{is\_inv}(\mathsf{evt}(\mathsf{last}(\widehat{T}_2|_\mathsf{t})))$. Then, by the operational semantics and the code of $\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\Gamma)$, we know $\mathsf{e\text{-}a\text{-}enabled}(\mathsf{t}, \widehat{T}_2)$ holds, which contradicts our assumption.
  - $\mathsf{is\_ret}(\mathsf{evt}(\mathsf{last}(\widehat{T}_2|_\mathsf{t})))$. Then, by the operational semantics, we know the code of thread t remains to be executed is **inclt**. Thus $\mathsf{e\text{-}a\text{-}enabled}(\mathsf{t}, \widehat{T}_2)$, which contradicts our assumption.
  - $\mathsf{evt}(\mathsf{last}(\widehat{T}_2|_\mathsf{t})) = (\mathsf{t}, \mathbf{obj})$. Since $\mathsf{i\text{-}o\text{-}dis}(\mathsf{t}, \widehat{T}_2)$, by the operational semantics, we know in $\widehat{T}_2$, the execution of thread t is blocked at an await statement. That is, if we suppose the configurations deriving the trace $\widehat{T}_2$ are $(\widetilde{C}_0, (\sigma_0, \mathcal{K}_0)), (\widetilde{C}_1, (\sigma_1, \mathcal{K}_1)), (\widetilde{C}_2, (\sigma_2, \mathcal{K}_2)), \ldots$, then there exist i, B, C and $\mathbf{E}$ such that
    $$\forall j \geq i.\ \widetilde{C}_j|_\mathsf{t} = \mathbf{E}[\,\mathbf{await}(B \wedge \mathtt{cid} = \mathbf{enhd}(\mathtt{listid}))\{C; \mathtt{listid} := \mathtt{listid}\backslash\mathtt{cid};\}\,],$$
    $$\text{and}\quad \forall j.\ \exists k \geq j.\ \neg((\sigma_k, \mathcal{K}_k(\mathsf{t})) \models (B \wedge \mathtt{cid} = \mathbf{enhd}(\mathtt{listid}))).$$
    Since $\mathsf{e\text{-}a\text{-}enabled}(\mathsf{t}, T_3)$ and $\mathtt{listid} \notin \mathsf{FV}(\Gamma)$, by the construction and the operational semantics, we know there exists $i' \geq i$ such that
    $$\forall j \geq i'.\ (\sigma_j, \mathcal{K}_j(\mathsf{t})) \models B.$$
    By the operational semantics and and the code of $\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\Gamma)$, we know there exist l and $i'' \geq i'$ such that
    $$\forall j \geq i''.\ \exists l'.\ \sigma_j(\mathtt{listid}) = l +\!\!+ (\mathsf{t}, B) +\!\!+ l'.$$
    If l is not empty, consider each (t', B') in l in order. If $\exists j \geq i''.\ (\sigma_j, \mathcal{K}_j(\mathsf{t}')) \models B'$, then $\exists j \geq i''.\ (\sigma_j, \mathcal{K}_j(\mathsf{t}')) \models (B' \wedge \mathtt{cid} = \mathbf{enhd}(\mathtt{listid}))$. By the operational semantics, we know $\exists j \geq i''.\ \forall k \geq j.\ (\sigma_k, \mathcal{K}_k(\mathsf{t}')) \models (B' \wedge \mathtt{cid} = \mathbf{enhd}(\mathtt{listid}))$. Since $\mathsf{wfair\text{-}o}(\widehat{T}_2)$, we know t' will eventually be executed and $\exists k \geq i''.\ \exists l'.\ \sigma_k(\mathtt{listid}) = (l\backslash\mathsf{t}') +\!\!+ (\mathsf{t}, B) +\!\!+ l'$ which contradicts the above result. Thus
    $$\forall (\mathsf{t}', B') \in l.\ \forall j \geq i''.\ \neg((\sigma_j, \mathcal{K}_j(\mathsf{t}')) \models B').$$
    As a result, we know
    $$\forall j \geq i''.\ (\sigma_j, \mathcal{K}_j(\mathsf{t})) \models (B \wedge \mathtt{cid} = \mathbf{enhd}(\mathtt{listid})),$$
    which contradicts the assumption.

  Thus $\mathsf{i\text{-}o\text{-}dis}(\mathsf{t}, \widehat{T}_3)$.

Thus $\mathsf{wfair\text{-}o}(\widehat{T}_3)$. Thus we know

$$(|\widehat{T}_3| \neq \omega) \wedge (\neg\mathsf{abt}(\widehat{T}_3)) \implies \forall \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T}_3)].\ \mathsf{term\text{-}o}(\widehat{T}_3|_\mathsf{t}) \vee \mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}_3))\ .$$

Thus $\mathsf{fin\_coherent}(\widehat{T}_c, \widehat{T}_3)$. Then, by Composition Theorem A.2, we know there exists T_3 such that

$$T_3 \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Gamma\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \Sigma, \circledcirc)]\!],\quad \widehat{T}_c = \mathsf{get\_clt}(T_3),\quad \widehat{T}_3 = \mathsf{get\_obj}(T_3).$$

By Lemma A.10, we know

$$\mathsf{wfair}(T_3).$$

Thus $\mathcal{E} \in \mathcal{O}_{\mathsf{wfair}}[\![(\mathbf{let}\ \Gamma\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \Sigma, \circledcirc)]\!]$ and we are done. □

Proof of Lemma A.23. Similar to the proof of Lemma A.14. For any n, σ, Σ and $\widehat{T}_o$ such that $\widehat{T}_o \in \mathcal{T}^{\mathsf{o}}_\omega[\![\Pi, \sigma]\!]$, $\mathsf{wfair\text{-}o}(\widehat{T}_o)$ and $\varphi(\sigma) = \Sigma$, suppose

$$\neg\mathsf{abt}(\widehat{T}_o) \quad \text{and} \quad \neg\mathsf{prog\text{-}t}(\widehat{T}_o) \quad \text{and} \quad \neg\mathsf{well\text{-}blocked\text{-}o}(\widehat{T}_o, (\Gamma, \Sigma))\ .$$

Thus we know

$$(|\widehat{T}_o| \neq \omega) \implies \forall \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T}_o)].\ \mathsf{term\text{-}o}(\widehat{T}_o|_\mathsf{t}) \vee \mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}_o))\ .$$

Since $\widehat{T}_o \in \mathcal{T}^{\mathsf{o}}_\omega[\![\Pi, \sigma]\!]$, we know there exist n and $\widehat{T}_c$ such that

$$\widehat{T_c} \in \mathcal{T}_\omega^c[\![\mathsf{MGCp1}_n, \sigma_{\mathsf{MGC}(n)}]\!], \quad \mathsf{get\_hist}(\widehat{T_c}) = \mathsf{get\_hist}(\widehat{T_o}), \quad n = \mathsf{tnum}(\widehat{T_o}),$$
$$\neg\mathsf{abt}(\widehat{T_c}), \quad \mathsf{wfair\text{-}c}(\widehat{T_c}), \quad \mathsf{fin\_coherent}(\widehat{T_c}, \widehat{T_o}).$$

By Composition Theorem A.2, we know there exists T such that

$$T \in \mathcal{T}_\omega[\![\textbf{let } \Pi \textbf{ in } \mathsf{MGCp1}_n, (\sigma_{\mathsf{MGC}(n)}, \sigma, \circledcirc)]\!], \quad \widehat{T_c} = \mathsf{get\_clt}(T), \quad \widehat{T_o} = \mathsf{get\_obj}(T).$$

Since $\mathsf{wfair\text{-}o}(\widehat{T_o})$ and $\mathsf{wfair\text{-}c}(\widehat{T_c})$, by Lemma A.10, we know

$$\mathsf{wfair}(T).$$

Since $\varphi(\sigma) = \Sigma$ and $\Pi \sqsubseteq^{\mathsf{wfair}}_{\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\varphi)} \mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\Gamma)$, we know:

$$O_{\mathsf{wfair}}[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGCp1}_n), (\sigma_{\mathsf{MGC}(n)}, \sigma)]\!] \subseteq$$
$$O_{\mathsf{wfair}}[\![(\textbf{let } \mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\Gamma) \textbf{ in } \mathsf{MGCp1}_n), (\sigma_{\mathsf{MGC}(n)}, \Sigma \uplus \{\mathtt{listid} \rightsquigarrow \epsilon\})]\!].$$

Thus there exists T' such that

$$T' \in \mathcal{T}_\omega[\![(\textbf{let } \mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\Gamma) \textbf{ in } \mathsf{MGCp1}_n), (\sigma_{\mathsf{MGC}(n)}, \Sigma \uplus \{\mathtt{listid} \rightsquigarrow \epsilon\}, \circledcirc)]\!],$$
$$\mathsf{wfair}(T') \quad \text{and} \quad \mathsf{get\_obsv}(T') = \mathsf{get\_obsv}(T).$$

Also, by the definition of $\mathsf{MGCp1}_n$ and the operational semantics, we can construct T_1 and an execution such that

$$T_1 \in \mathcal{T}_\omega[\![(\textbf{let } \mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\Gamma) \textbf{ in } \mathsf{MGCp1}_n), (\sigma_{\mathsf{MGC}(n)}, \Sigma \uplus \{\mathtt{listid} \rightsquigarrow \epsilon\}, \circledcirc)]\!],$$
$$\mathsf{wfair}(T_1), \quad \mathsf{get\_obsv}(T_1) = \mathcal{E}, \quad \mathsf{get\_hist}(T) = \mathsf{get\_hist}(T_1).$$

By Decomposition Theorem A.1, we know there exists $\widehat{T_1}$ such that

$$\widehat{T_1} \in \mathcal{T}_\omega^o[\![\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\Gamma), \Sigma \uplus \{\mathtt{listid} \rightsquigarrow \epsilon\}]\!], \quad \widehat{T_1} = \mathsf{get\_obj}(T_1), \quad n = \mathsf{tnum}(\widehat{T_1}).$$

Since $\mathsf{wfair}(T_1)$, by Lemma A.9, we know

$$\mathsf{wfair\text{-}o}(\widehat{T_1}).$$

Since $\widehat{T_1} = \mathsf{get\_obj}(T_1)$, $\widehat{T_o} = \mathsf{get\_obj}(T)$ and $\mathsf{get\_hist}(T) = \mathsf{get\_hist}(T_1)$, we know

$$\mathsf{get\_hist}(\widehat{T_o}) = \mathsf{get\_hist}(\widehat{T_1}).$$

Since $n = \mathsf{tnum}(\widehat{T_o})$ and $n = \mathsf{tnum}(\widehat{T_1})$, we know

$$\mathsf{tnum}(\widehat{T_o}) = \mathsf{tnum}(\widehat{T_1}).$$

Similar to the proof of Lemma A.22, we know there exists $\widehat{T_a}$ such that

$$\widehat{T_a} \in \mathcal{T}_\omega^o[\![\Gamma, \Sigma]\!], \quad \mathsf{get\_hist}(\widehat{T_a}) = \mathsf{get\_hist}(\widehat{T_1}), \quad \mathsf{tnum}(\widehat{T_a}) = \mathsf{tnum}(\widehat{T_1}).$$

- Suppose $|\widehat{T_a}| \neq \omega$. By the construction of $\widehat{T_a}$, we know $|\widehat{T_1}| \neq \omega$. For any $e \in \mathsf{pend\_inv}(\widehat{T_a})$ and $\mathsf{t} = \mathsf{tid}(e)$, since $\mathsf{get\_hist}(\widehat{T_a}) = \mathsf{get\_hist}(\widehat{T_1})$, we know $e \in \mathsf{pend\_inv}(\widehat{T_1})$ and $|(\widehat{T_1}|_\mathsf{t})| \neq \omega$. Since $\mathsf{wfair\text{-}o}(\widehat{T_1})$, we know
$$\mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T_1})).$$
By the construction of $\widehat{T_a}$, we know $\mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T_a}))$. Thus we know $\mathsf{well\text{-}blocked\text{-}o}(\widehat{T_o}, (\Gamma, \Sigma))$ holds, which contradicts our assumption.
- Suppose $|\widehat{T_a}| = \omega$. By the construction of $\widehat{T_a}$, we know $|\widehat{T_1}| = \omega$. For any $e \in \mathsf{pend\_inv}(\widehat{T_a})$ and $\mathsf{t} = \mathsf{tid}(e)$, by the operational semantics and the code of Γ, we know $|(\widehat{T_a}|_\mathsf{t})| \neq \omega$. Since $\mathsf{get\_hist}(\widehat{T_a}) = \mathsf{get\_hist}(\widehat{T_1})$, we know $e \in \mathsf{pend\_inv}(\widehat{T_1})$ and $|(\widehat{T_1}|_\mathsf{t})| \neq \omega$. Since $\mathsf{wfair\text{-}o}(\widehat{T_1})$, we know $\mathsf{i\text{-}o\text{-}dis}(\mathsf{t}, \widehat{T_1})$ holds. Thus
$$\forall i_0.\ \exists j_0 \geq i_0.\ \mathsf{t} \in \mathsf{bidset}(\widehat{T_1}(j_0)).$$
Since $\mathsf{well\text{-}blocked\text{-}o}(\widehat{T_o}, (\Gamma, \Sigma))$ does not hold, we know $\mathsf{e\text{-}a\text{-}dis}(\mathsf{t}, \widehat{T_a})$ does not hold. Thus $\mathsf{i\text{-}o\text{-}enabled}(\mathsf{t}, \widehat{T_a})$ holds. Thus
$$\forall i_1.\ \exists j_1 \geq i_1.\ \mathsf{t} \notin \mathsf{bidset}(\widehat{T_a}(j_1)).$$

Suppose the configurations deriving the trace $\widehat{T}_1$ are $(\widetilde{C}_0,(\sigma_0,\mathcal{K}_0)),(\widetilde{C}_1,(\sigma_1,\mathcal{K}_1)),(\widetilde{C}_2,(\sigma_2,\mathcal{K}_2)),\ldots$, then there exist i, B, C and $\mathbf{E}$ such that

$$\forall j \geq i.\ \widetilde{C}_j|_{\mathsf{t}} = \mathbf{E}[\ \mathbf{await}(B \wedge \mathtt{cid} = \mathbf{enhd}(\mathtt{listid}))\{C; \mathtt{listid} := \mathtt{listid}\backslash\mathtt{cid};\}\ ],$$
$$\text{and}\quad \forall j.\ \exists k \geq j.\ \neg((\sigma_k,\mathcal{K}_k(\mathsf{t})) \models (B \wedge \mathtt{cid} = \mathbf{enhd}(\mathtt{listid}))).$$

By the construction of $\widehat{T}_a$, we know

$$\forall i_1.\ \exists j_1 \geq i_1.\ (\sigma_{j_1},\mathcal{K}_{j_1}(\mathsf{t})) \models B\,.$$

By the operational semantics and and the code of $\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\Gamma)$, we know there exist l and $i'' \geq i'$ such that

$$\forall j \geq i''.\ \exists l'.\ \sigma_j(\mathtt{listid}) = l +\!\!+ (\mathsf{t},B) +\!\!+ l'.$$

If l is not empty, consider each (t',B') in l in order. If $\exists j \geq i''.\ (\sigma_j,\mathcal{K}_j(\mathsf{t}')) \models B'$, then $\exists j \geq i''.\ (\sigma_j,\mathcal{K}_j(\mathsf{t}')) \models (B' \wedge \mathtt{cid} = \mathbf{enhd}(\mathtt{listid}))$. By the operational semantics, we know $\exists j \geq i''.\ \forall k \geq j.\ (\sigma_k,\mathcal{K}_k(\mathsf{t}')) \models (B' \wedge \mathtt{cid} = \mathbf{enhd}(\mathtt{listid}))$. Since wfair-o($\widehat{T}_1$), we know t' will eventually be executed and $\exists k \geq i''.\ \exists l'.\ \sigma_k(\mathtt{listid}) = (l\backslash\mathsf{t}') +\!\!+ (\mathsf{t},B) +\!\!+ l'$ which contradicts the above result. Thus

$$\forall(\mathsf{t}',B') \in l.\ \forall j \geq i''.\ \neg((\sigma_j,\mathcal{K}_j(\mathsf{t}')) \models B').$$

Since $\forall i_1.\ \exists j_1 \geq i_1.\ (\sigma_{j_1},\mathcal{K}_{j_1}(\mathsf{t})) \models B$, we know there exists $j'' \geq i''$ such that $(\sigma_{j''},\mathcal{K}_{j''}(\mathsf{t})) \models B$. As a result, we know

$$\forall j \geq j''.\ (\sigma_j,\mathcal{K}_j(\mathsf{t})) \models (B \wedge \mathtt{cid} = \mathbf{enhd}(\mathtt{listid})),$$

which contradicts the assumption.

Thus we are done. □

Proof of Lemma A.24. The key is to show the following (A.3).

For any n, $C_1,\ldots,C_n$, σ_c, σ, Σ, T and T_a such that $\varphi(\sigma) = \Sigma$,
if $T \in \mathcal{T}_\omega[\![\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n,(\sigma_c,\sigma,\circledcirc)]\!]$, $\neg\mathsf{abt}(T)$ and $\mathsf{wfair}(T)$, then there exists T_a such that $T_a \in \mathcal{T}_\omega[\![\mathbf{let}\ \mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\Gamma)\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n,(\sigma_c,\Sigma \uplus \{\mathtt{listid} \rightsquigarrow \epsilon\},\circledcirc)]\!]$, $\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$ and $\mathsf{wfair}(T_a)$. (A.3)

Since $T \in \mathcal{T}_\omega[\![\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n,(\sigma_c,\sigma,\circledcirc)]\!]$ and $\neg\mathsf{abt}(T)$, by Decomposition Theorem A.1, we know there exist $\widehat{T}_c$ and $\widehat{T}_o$ such that

$$\widehat{T}_c \in \mathcal{T}^{\mathsf{c}}_\omega[\![C_1 \parallel \ldots \parallel C_n,\sigma_c]\!],\quad \widehat{T}_o \in \mathcal{T}^{\mathsf{o}}_\omega[\![\Pi,\sigma]\!],$$
$$\mathsf{get\_hist}(\widehat{T}_c) = \mathsf{get\_hist}(\widehat{T}_o),\quad \widehat{T}_c = \mathsf{get\_clt}(T),\quad \widehat{T}_o = \mathsf{get\_obj}(T),\quad n = \mathsf{tnum}(\widehat{T}_o).$$

Since $\mathsf{wfair}(T)$, by Lemmas A.8 and A.9, we know

$$\mathsf{wfair\text{-}c}(\widehat{T}_c)\quad\text{and}\quad \mathsf{wfair\text{-}o}(\widehat{T}_o).$$

Thus we know

$$(|\widehat{T}_c| \neq \omega) \implies \forall \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T}_c)].\ \mathsf{term\text{-}c}(\widehat{T}_c|_{\mathsf{t}}) \vee \mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}_c))\ .$$

Since $\mathsf{PSF\text{-}O}^{\mathsf{wfair}}_{\varphi,\Gamma}(\Pi)$, we know $\mathsf{prog\text{-}t}(\widehat{T}_o) \vee \mathsf{well\text{-}blocked\text{-}o}(\widehat{T}_o,(\Gamma,\Sigma))$ holds.

- $\mathsf{prog\text{-}t}(\widehat{T}_o)$ holds. Since $\Pi \sqsubseteq^{\mathsf{fin}}_\varphi \Gamma$, we know

  $$\mathcal{H}[\![\Pi,\sigma]\!] \subseteq \mathcal{H}[\![\Gamma,\Sigma]\!].$$

  By Lemma A.18, we know there exists $\widehat{T}_a$ such that

  $$\widehat{T}_a \in \mathcal{T}^{\mathsf{o}}_\omega[\![\Gamma,\Sigma]\!],\quad \mathsf{get\_hist}(\widehat{T}_a) = \mathsf{get\_hist}(\widehat{T}_o),\quad \mathsf{tnum}(\widehat{T}_a) = \mathsf{tnum}(\widehat{T}_o).$$

  By Lemma A.25, we know there exists $\widehat{T}_1$ such that

$$\widehat{T_1} \in \mathcal{T}_\omega^o[\![\mathsf{wr}_{\mathsf{PSF}}^{\mathsf{wfair}}(\Gamma), \Sigma \uplus \{\mathtt{listid} \rightsquigarrow \epsilon\}]\!], \quad \mathsf{get\_hist}(\widehat{T_a}) = \mathsf{get\_hist}(\widehat{T_1}), \\ \mathsf{tnum}(\widehat{T_a}) = \mathsf{tnum}(\widehat{T_1}).$$

Thus $\mathsf{get\_hist}(\widehat{T_c}) = \mathsf{get\_hist}(\widehat{T_1})$, $n = \mathsf{tnum}(\widehat{T_1})$ and $\neg\mathsf{abt}(\widehat{T_1})$. Also, by Lemma A.19, we know

$$\mathsf{prog\text{-}t}(\widehat{T_1}).$$

Thus

$$(|\widehat{T_1}| \neq \omega) \implies \forall \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T_1})].\ \mathsf{term\text{-}o}(\widehat{T_1}|_{\mathsf{t}}) \ .$$

Thus $\mathsf{fin\_coherent}(\widehat{T_c}, \widehat{T_1})$ holds. By Composition Theorem A.2, we know there exists T_1 such that

$$T_1 \in \mathcal{T}_\omega[\![\textbf{let } \mathsf{wr}_{\mathsf{PSF}}^{\mathsf{wfair}}(\Gamma) \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \Sigma \uplus \{\mathtt{listid} \rightsquigarrow \epsilon\}, \circledcirc)]\!], \\ \widehat{T_c} = \mathsf{get\_clt}(T_1), \quad \widehat{T_1} = \mathsf{get\_obj}(T_1).$$

Thus

$$\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_1).$$

By Lemma A.20, we know

$$\mathsf{sfair\text{-}o}(\widehat{T_1}).$$

Thus we know $\mathsf{wfair\text{-}o}(\widehat{T_1})$. Then, by Lemma A.10, we know

$$\mathsf{wfair}(T_1).$$

- $\mathsf{well\text{-}blocked\text{-}o}(\widehat{T_o}, (\Gamma, \Sigma))$ holds. Thus there exists $\widehat{T_a}$ such that

$$\widehat{T_a} \in \mathcal{T}_\omega^o[\![\Gamma, \Sigma]\!], \quad \mathsf{get\_hist}(\widehat{T}) = \mathsf{get\_hist}(\widehat{T_a}), \quad \mathsf{tnum}(\widehat{T}) = \mathsf{tnum}(\widehat{T_a}), \\ |\widehat{T_a}| = \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T_a}) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e), \widehat{T_a})), \\ |\widehat{T_a}| \neq \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T_a}) \implies \mathsf{tid}(e) \in \mathsf{bset}(\mathsf{last}(\widehat{T_a}))).$$

By Lemma A.26, we know there exists $\widehat{T_1}$ such that

$$\widehat{T_1} \in \mathcal{T}_\omega^o[\![\mathsf{wr}_{\mathsf{PSF}}^{\mathsf{wfair}}(\Gamma), \Sigma \uplus \{\mathtt{listid} \rightsquigarrow \epsilon\}]\!], \quad \mathsf{get\_hist}(\widehat{T_a}) = \mathsf{get\_hist}(\widehat{T_1}), \\ \mathsf{tnum}(\widehat{T_a}) = \mathsf{tnum}(\widehat{T_1}), \\ |\widehat{T_1}| = \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T_1}) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e), \widehat{T_1})), \\ |\widehat{T_1}| \neq \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T_1}) \implies \mathsf{tid}(e) \in \mathsf{bset}(\mathsf{last}(\widehat{T_1}))).$$

Thus $\mathsf{get\_hist}(\widehat{T_c}) = \mathsf{get\_hist}(\widehat{T_1})$, $n = \mathsf{tnum}(\widehat{T_1})$ and $\neg\mathsf{abt}(\widehat{T_1})$. Also we know

$$(|\widehat{T_1}| \neq \omega) \implies \forall \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T_1})].\ \mathsf{term\text{-}o}(\widehat{T_1}|_{\mathsf{t}}) \vee \mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T_1})) \ .$$

Thus $\mathsf{fin\_coherent}(\widehat{T_c}, \widehat{T_1})$ holds. By Composition Theorem A.2, we know there exists T_1 such that

$$T_1 \in \mathcal{T}_\omega[\![\textbf{let } \mathsf{wr}_{\mathsf{PSF}}^{\mathsf{wfair}}(\Gamma) \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \Sigma \uplus \{\mathtt{listid} \rightsquigarrow \epsilon\}, \circledcirc)]\!], \\ \widehat{T_c} = \mathsf{get\_clt}(T_1), \quad \widehat{T_1} = \mathsf{get\_obj}(T_1).$$

Thus

$$\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_1).$$

By Lemma A.21, we know

$$\mathsf{sfair\text{-}o}(\widehat{T_1}).$$

Thus we know $\mathsf{wfair\text{-}o}(\widehat{T_1})$. Then, by Lemma A.10, we know

$$\mathsf{wfair}(T_1).$$

Thus we are done. □

Lemma A.25. *If* $\widehat{T}_a \in \mathcal{T}_\omega^o[\![\Gamma,\Sigma]\!]$ *and* prog-t$(\widehat{T}_a)$*, then there exists* $\widehat{T}_1$ *such that* $\widehat{T}_1 \in \mathcal{T}_\omega^o[\![\mathsf{wr}_{\mathsf{PSF}}^{\mathsf{wfair}}(\Gamma), \Sigma \uplus \{\mathtt{listid} \rightsquigarrow \epsilon\}]\!]$, $\mathsf{get\_hist}(\widehat{T}_a) = \mathsf{get\_hist}(\widehat{T}_1)$ *and* $\mathsf{tnum}(\widehat{T}_a) = \mathsf{tnum}(\widehat{T}_1)$.

Proof. Similar to the proof of Lemma A.26. □

Lemma A.26. *If* $\widehat{T}_a \in \mathcal{T}_\omega^o[\![\Gamma,\Sigma]\!]$ *and* $|\widehat{T}_a| = \omega \implies (\forall e.\, e \in \mathsf{pend\_inv}(\widehat{T}_a) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e), \widehat{T}_a))$ *and* $|\widehat{T}_a| \neq \omega \implies (\forall e.\, e \in \mathsf{pend\_inv}(\widehat{T}_a) \implies \mathsf{tid}(e) \in \mathsf{bset}(\mathsf{last}(\widehat{T}_a)))$*, then there exists* $\widehat{T}_1$ *such that*

$$\widehat{T}_1 \in \mathcal{T}_\omega^o[\![\mathsf{wr}_{\mathsf{PSF}}^{\mathsf{wfair}}(\Gamma), \Sigma \uplus \{\mathtt{listid} \rightsquigarrow \epsilon\}]\!], \quad \mathsf{get\_hist}(\widehat{T}_a) = \mathsf{get\_hist}(\widehat{T}_1), \quad \mathsf{tnum}(\widehat{T}_a) = \mathsf{tnum}(\widehat{T}_1),$$
$$|\widehat{T}_1| = \omega \implies (\forall e.\, e \in \mathsf{pend\_inv}(\widehat{T}_1) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e), \widehat{T}_1)),$$
$$|\widehat{T}_1| \neq \omega \implies (\forall e.\, e \in \mathsf{pend\_inv}(\widehat{T}_1) \implies \mathsf{tid}(e) \in \mathsf{bset}(\mathsf{last}(\widehat{T}_1))).$$

Proof. Since $\widehat{T}_a \in \mathcal{T}_\omega^o[\![\Gamma,\Sigma]\!]$, we know there exist n, $\widehat{T}$ and $\widetilde{C}_0$ such that $\widehat{T}_a = ((\mathbf{spawn}, n), \emptyset) :: \mathsf{get\_obj}(\widehat{T})$ and

$$(\widetilde{C}_0, (\Sigma, \circledcirc)) \overset{\widehat{T}}{\mapsto}{}^{\omega}_{\Gamma} \cdot \quad \text{or} \quad (\widetilde{C}_0, (\Sigma, \circledcirc)) \overset{\widehat{T}}{\mapsto}{}^{*}_{\Gamma} \mathbf{abort} \quad \text{or}$$
$$\exists \widetilde{C}', \Sigma', \mathcal{K}'.\ ((\widetilde{C}_0, (\Sigma, \circledcirc)) \overset{\widehat{T}}{\mapsto}{}^{*}_{\Gamma} (\widetilde{C}', (\Sigma', \mathcal{K}'))) \wedge \neg(\exists \widehat{\iota}.\ (\widetilde{C}', (\Sigma', \mathcal{K}')) \overset{\widehat{\iota}}{\mapsto}_{\Gamma} \_).$$

Below we only consider the case when $|\widehat{T}| = \omega$. Proof for the case when $|\widehat{T}| \neq \omega$ is similar. Let

$$S_p = \{(\mathsf{t}, i) \mid \exists e.\, e \in \mathsf{pend\_inv}(\widehat{T}) \wedge \mathsf{t} = \mathsf{tid}(e) \wedge (\forall j \geq i.\, \mathsf{t} \in \mathsf{bidset}(\widehat{T}(j))) \wedge (\mathsf{t} \notin \mathsf{bidset}(\widehat{T}(i-1)))\}.$$

We construct the trace $\widehat{T}_1$ such that $((\mathbf{spawn}, n), \emptyset) :: \mathsf{get\_obj}(\widehat{T}_1)$ satisfies the desired properties. The idea is to construct a simulation between the executions. Informally, our construction of $\widehat{T}_1$ considers every prefix $\widehat{T}(1..i)$ of $\widehat{T}$ and builds traces $\widehat{T}_1^i$ and their derivations for $\widehat{T}_1^i$. The resulting series are such that for $i < j$, the derivation of $\widehat{T}_1^i$ is a prefix of that of $\widehat{T}_1^j$, which also implies that the trace $\widehat{T}_1^i$ is a prefix of $\widehat{T}_1^j$. Because of this, we could get a simulation relation between the executions, and the limit derivation and the limit trace are the desired $\widehat{T}_1$ and the corresponding derivation. The following claim lies at the core of our construction:

Consider a prefix $\widehat{T}(1..i)$ of $\widehat{T}$, the trace $\widehat{T}_1^i$ and $\widetilde{C}_1$, $\widetilde{C}$, Σ_1', Σ', $\mathcal{K}'$ such that

$$(\widetilde{C}_0, (\Sigma, \circledcirc)) \overset{\widehat{T}(1..i)}{\mapsto}{}^{*}_{\Gamma} (\widetilde{C}, (\Sigma', \mathcal{K}')),$$
$$(\widetilde{C}_0, (\Sigma \uplus \{\mathtt{listid} \rightsquigarrow \epsilon\}, \circledcirc)) \overset{\widehat{T}_1^i}{\mapsto}{}^{*}_{\mathsf{wr}_{\mathsf{PSF}}^{\mathsf{wfair}}(\Gamma)} (\widetilde{C}_1, (\Sigma_1', \mathcal{K}')),$$
$$\Sigma' = \Sigma_1' \backslash \{\mathtt{listid}\}, \quad \mathsf{list2set}(\Sigma_1'(\mathtt{listid})) = \{\mathsf{t} \mid S_p(\mathsf{t}) \leq i\},$$
$$\widetilde{C} \smile_{\Sigma_1', \mathcal{K}'} \widetilde{C}_1, \quad \mathsf{get\_hist}(\widehat{T}(1..i)) = \mathsf{get\_hist}(\widehat{T}_1^i).$$

Here $\widetilde{C} \smile_{\Sigma_1', \mathcal{K}'} \widetilde{C}_1$ requires the following hold:

$$\forall \mathsf{t}.\ (\mathcal{K}'(\mathsf{t}) = \circ) \implies (\widetilde{C}_1|_{\mathsf{t}} = \widetilde{C}|_{\mathsf{t}}),$$
$$\forall \mathsf{t}.\ (\mathcal{K}'(\mathsf{t}) \neq \circ) \wedge \mathsf{t} \notin \mathsf{list2set}(\Sigma_1'(\mathtt{listid})) \implies (\widetilde{C}|_{\mathsf{t}} = \widetilde{C}_1|_{\mathsf{t}}) \vee$$
$$\exists B, C.\ (\widetilde{C}|_{\mathsf{t}} = \mathbf{await}(B)\{C\}) \wedge (\widetilde{C}_1|_{\mathsf{t}} = \mathsf{wr}_{\mathsf{PSF}}^{\mathsf{wfair}}(\mathbf{await}(B)\{C\}))$$
$$\forall \mathsf{t}.\ (\mathcal{K}'(\mathsf{t}) \neq \circ) \wedge \mathsf{t} \in \mathsf{list2set}(\Sigma_1'(\mathtt{listid})) \implies$$
$$(\widetilde{C}_1|_{\mathsf{t}} = \mathbf{await}(B \wedge \mathtt{cid} = \mathbf{enhd}(\mathtt{listid}))\{C; \mathtt{listid} := \mathtt{listid} \backslash \mathtt{cid};\}) \wedge (\widetilde{C}|_{\mathsf{t}} = \mathbf{await}(B)\{C\}).$$

If $(\widetilde{C}, (\Sigma', \mathcal{K}')) \overset{\widehat{T}(i+1)}{\mapsto}{}_{\Gamma} (\widetilde{C}', (\Sigma'', \mathcal{K}''))$, then there exist $\widetilde{C}_1'$, Σ_1'' and an extension $\widehat{T}_1^{i+1}$ of $\widehat{T}_1^i$ with the corresponding derivation such that

$$(\widetilde{C}_0, (\Sigma \uplus \{\texttt{listid} \rightsquigarrow \epsilon\}, , \circledcirc)) \xrightarrow{\widehat{T}_1^{i+1}}{}^{*}_{\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\Gamma)} (\widetilde{C}_1', (\Sigma_1'', \mathcal{K}'')),$$
$$\Sigma'' = \Sigma_1''\backslash\{\texttt{listid}\}, \quad \mathsf{list2set}(\Sigma_1''(\texttt{listid})) = \{\mathrm{t} \mid S_p(\mathrm{t}) \le i+1\},$$
$$\widetilde{C}' \frown_{\Sigma_1'', \mathcal{K}''} \widetilde{C}_1', \quad \mathsf{get\_hist}(\widehat{T}(1..i+1)) = \mathsf{get\_hist}(\widehat{T}_1^{i+1}).$$

Also, if $(\widetilde{C}, (\Sigma', \mathcal{K}')) \xrightarrow{\widehat{T}(i+1)}_{\Gamma} \textbf{abort}$, then there exists an extension $\widehat{T}_1^{i+1}$ of $\widehat{T}_1^i$ such that $(\widetilde{C}_0, (\Sigma \uplus \{\texttt{listid} \rightsquigarrow \epsilon\}, , \circledcirc)) \xrightarrow{\widehat{T}_1^{i+1}}{}^{*}_{\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\Gamma)} \textbf{abort}$ and $\mathsf{get\_hist}(\widehat{T}(1..i+1)) = \mathsf{get\_hist}(\widehat{T}_1^{i+1})$.

To prove the claim, we make a case-split on the derivation of $\widehat{T}(i+1)$.

- If $\widehat{T}(i+1)$ is an abort event, then we could generate the same abort event at the next step of $(\widetilde{C}_1, (\Sigma_1', \mathcal{K}'))$.
- If $\widehat{T}(i+1)$ is an invocation event of thread t, then we could generate the same invocation event at the next step of $(\widetilde{C}_1, (\Sigma_1', \mathcal{K}'))$. If $S_p(\mathrm{t}) \le i+1$, then we also execute its first step of the method body $\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\textbf{await}(B)\{C\})$.
- If $\widehat{T}(i+1)$ is an object event and the step is executing $\textbf{await}(B)\{C\}$ of thread t, then we know $\mathrm{t} \notin \mathsf{list2set}(\Sigma_1'(\texttt{listid}))$. Then we execute $\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\textbf{await}(B)\{C\})$ (two steps) from $(\widetilde{C}_1, (\Sigma_1', \mathcal{K}'))$. Since the object state has been changed, we also compute the blocked threads, and for any new thread t′ such that $S_p(\mathrm{t}') \le i+1$, we also execute its first step of the method body.
- If $\widehat{T}(i+1)$ is a return event, then we could generate the same return event at the next step of $(\widetilde{C}_1, (\Sigma_1', \mathcal{K}'))$.

By the construction, we are done. □

A.4 Proofs of Theorem 6.2-4 (PDF under weak fairness)

By Theorem 4.4, the goal is reduced to the following:

$$\Pi \sqsubseteq^{\mathsf{fin}}_{\varphi} \Gamma \wedge \mathsf{PDF}^{\mathsf{wfair}}_{\varphi,\Gamma}(\Pi) \iff \Pi \sqsubseteq^{\mathsf{wfair}}_{\mathsf{wr}_{\mathsf{PDF}}(\varphi)} \mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PDF}}(\Gamma).$$

We first define the object version of partial deadlock-freedom.

Definition A.27. $\mathsf{PDF\text{-}O}^{\chi}_{\varphi,\Gamma}(\Pi)$, iff

$$\forall n, \sigma, \Sigma, T.\ \widehat{T} \in \mathcal{T}^{\mathrm{o}}_{\omega}[\![\Pi, \sigma]\!] \wedge (\varphi(\sigma) = \Sigma) \wedge \chi\text{-o}(\widehat{T})$$
$$\implies \mathsf{abt}(\widehat{T}) \vee \mathsf{prog\text{-}p}(\widehat{T}) \vee \mathsf{well\text{-}blocked\text{-}o}(\widehat{T}, (\Gamma, \Sigma)).$$

We only need to prove the following lemmas.

Lemma A.28. $\Pi \sqsubseteq^{\mathsf{wfair}}_{\mathsf{wr}_{\mathsf{PDF}}(\varphi)} \mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PDF}}(\Gamma) \implies \Pi \sqsubseteq^{\mathsf{fin}}_{\varphi} \Gamma$.

Lemma A.29. $\Pi \sqsubseteq^{\mathsf{wfair}}_{\mathsf{wr}_{\mathsf{PDF}}(\varphi)} \mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PDF}}(\Gamma) \implies \mathsf{PDF\text{-}O}^{\mathsf{wfair}}_{\varphi,\Gamma}(\Pi)$.

Lemma A.30. $\mathsf{PDF\text{-}O}^{\chi}_{\varphi,\Gamma}(\Pi) \iff \mathsf{PDF}^{\chi}_{\varphi,\Gamma}(\Pi)$.

Lemma A.31. $\Pi \sqsubseteq^{\mathsf{fin}}_{\varphi} \Gamma \wedge \mathsf{PDF\text{-}O}^{\mathsf{wfair}}_{\varphi,\Gamma}(\Pi) \implies \Pi \sqsubseteq^{\mathsf{wfair}}_{\mathsf{wr}_{\mathsf{PDF}}(\varphi)} \mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PDF}}(\Gamma)$.

Proof of Lemma A.28. For any $n, C_1, \ldots, C_n, \sigma_c, \sigma$ and Σ such that $\varphi(\sigma) = \Sigma$, for any $\mathcal{E}$, if

$$\mathcal{E} \in \mathcal{O}[\![(\textbf{let}\ \Pi\ \textbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma, \circledcirc)]\!],$$

we know there exists T_1 such that $\mathcal{E} = \mathsf{get\_obsv}(T_1)$ and

$$T_1 \in \mathcal{T}[\![(\textbf{let}\ \Pi\ \textbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma, \circledcirc)]\!].$$

We can construct T_1' and T_1'' such that

$$T_1'' = T_1 :: T_1' \ , \quad \mathsf{wfair}(T_1'') \quad \text{and} \quad T_1'' \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n), (\sigma_c, \sigma, \circledcirc)]\!].$$

Since $\Pi \sqsubseteq^{\mathsf{wfair}}_{\mathsf{wr}_{\mathsf{PDF}}(\varphi)} \mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PDF}}(\Gamma)$, we know there exists T_2'' such that

$$T_2'' \in \mathcal{T}_\omega[\![(\mathbf{let}\ \mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PDF}}(\Gamma)\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n), (\sigma_c, \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathbf{false}\}, \circledcirc)]\!] \ , \quad \mathsf{wfair}(T_2'') \quad \text{and}$$
$$\mathsf{get\_obsv}(T_2'') = \mathsf{get\_obsv}(T_1'') = \mathcal{E} :: \mathsf{get\_obsv}(T_1') \, .$$

Thus there exists T_2 such that

$$T_2 \in \mathcal{T}[\![(\mathbf{let}\ \mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PDF}}(\Gamma)\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n), (\sigma_c, \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathbf{false}\}, \circledcirc)]\!] \quad \text{and} \quad \mathsf{get\_obsv}(T_2) = \mathcal{E}.$$

Then we construct T_3 such that

$$T_3 \in \mathcal{T}[\![(\mathbf{let}\ \Gamma\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n), (\sigma_c, \Sigma, \circledcirc)]\!], \quad \mathsf{get\_clt}(T_2) = \mathsf{get\_clt}(T_3).$$

Our construction considers every prefix $T_2(1..i)$ of T_2 and builds traces T_3^i and their derivations for T_3^i. The resulting series are such that for $i < j$, the derivation of T_3^i is a prefix of that of T_3^j, which also implies that the trace T_3^i is a prefix of T_3^j. Then $T_3^{|T_2|}$ is the desired T_3. The following claim lies at the core of our construction:

Consider a prefix $T_2(1..i)$ of T_2, the trace T_3^i and W_1, W, σ_c', Σ_1', Σ', $\mathcal{K}'$ such that

$$(\mathbf{let}\ \mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PDF}}(\Gamma)\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n, (\sigma_c, \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathbf{false}\}, \circledcirc)) \xmapsto{T_2(1..i)}{}^{*} (W_1, (\sigma_c', \Sigma_1', \mathcal{K}')),$$
$$(\mathbf{let}\ \Gamma\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n, (\sigma_c, \Sigma, \circledcirc)) \xmapsto{T_3^i}{}^{*} (W, (\sigma_c', \Sigma', \mathcal{K}')),$$
$$\Sigma' = \Sigma_1' \backslash \{\mathsf{done}\}, \quad W_1 \smile_{\mathcal{K}'} W, \quad \mathsf{get\_clt}(T_2(1..i)) = \mathsf{get\_clt}(T_3^i).$$

Here $W_1 \smile_{\mathcal{K}'} W$ requires the following hold:

$$\forall \mathsf{t}.\ (\mathcal{K}'(\mathsf{t}) = \circ) \Longrightarrow (W_1|_{\mathsf{t}} = W|_{\mathsf{t}}),$$
$$\forall \mathsf{t}.\ (\mathcal{K}'(\mathsf{t}) \neq \circ) \Longrightarrow (W_1|_{\mathsf{t}} = W|_{\mathsf{t}}) \vee$$
$$\exists B, C, E.\ (W_1|_{\mathsf{t}} = (\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PDF}}(\mathbf{await}(B)\{C\}); \mathbf{return}\ E)) \wedge (W|_{\mathsf{t}} = (\mathbf{await}(B)\{C\}; \mathbf{return}\ E)) \vee$$
$$\exists E.\ (W_1|_{\mathsf{t}} = (\mathsf{done} := \mathsf{false}; \mathbf{await}(\neg\mathsf{done})\{\}; \mathbf{return}\ E)) \wedge (W|_{\mathsf{t}} = \mathbf{return}\ E) \vee$$
$$\exists E.\ (W_1|_{\mathsf{t}} = (\mathbf{await}(\neg\mathsf{done})\{\}; \mathbf{return}\ E)) \wedge (W|_{\mathsf{t}} = \mathbf{return}\ E).$$

If $(W_1, (\sigma_c', \Sigma_1', \mathcal{K}')) \xmapsto{T_2(i+1)} (W_1', (\sigma_c'', \Sigma_1'', \mathcal{K}''))$, then there exist W', Σ'' and an extension T_3^{i+1} of T_3^i with the corresponding derivation such that

$$(\mathbf{let}\ \Gamma\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n, (\sigma_c, \Sigma, \circledcirc)) \xmapsto{T_3^{i+1}}{}^{*} (W', (\sigma_c'', \Sigma'', \mathcal{K}'')),$$
$$\Sigma'' = \Sigma_1'' \backslash \{\mathsf{done}\}, \quad W_1' \smile_{\mathcal{K}''} W', \quad \mathsf{get\_clt}(T_2(1..i+1)) = \mathsf{get\_clt}(T_3^{i+1}).$$

Also, if $(W_1, (\sigma_c', \Sigma_1', \mathcal{K}')) \xmapsto{T_2(i+1)}$ **abort**, then there exists an extension T_3^{i+1} of T_3^i such that $(\mathbf{let}\ \Gamma\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n, (\sigma_c, \Sigma, \circledcirc)) \xmapsto{T_3^{i+1}}{}^{*}$ **abort** and $\mathsf{get\_clt}(T_2(1..i+1)) = \mathsf{get\_clt}(T_3^{i+1})$.

To prove the claim, we make a case-split on the derivation of $T_2(i+1)$.

- If $T_2(i+1)$ is a client event, then we could generate the same client event at the next step of $(W, (\sigma_c', \Sigma', \mathcal{K}'))$.
- If $T_2(i+1)$ is an invocation event, then we could generate the same invocation event at the next step of $(W, (\sigma_c', \Sigma', \mathcal{K}'))$.
- If $T_2(i+1)$ is an object event and the step is executing $\mathbf{await}(B \wedge \neg\mathsf{done})\{C; \mathsf{done} := \mathsf{true};\}$, then we execute one step of $(W, (\sigma_c', \Sigma', \mathcal{K}'))$ and that step is executing $\mathbf{await}(B)\{C\}$.
- If $T_2(i+1)$ is an object event and the step is executing $\mathsf{done} := \mathsf{false}$, then we execute zero step of $(W, (\sigma_c', \Sigma', \mathcal{K}'))$ (that is, $T_3^{i+1} = T_3^i$).
- If $T_2(i+1)$ is an object event and the step is executing $\mathbf{await}(\neg\mathsf{done})\{\}$, then we execute zero step of $(W, (\sigma_c', \Sigma', \mathcal{K}'))$ (that is, $T_3^{i+1} = T_3^i$).

- If $T_2(i+1)$ is a return event, then we could generate the same return event at the next step of $(W, (\sigma_c', \Sigma', \mathcal{K}'))$.

Thus $\mathcal{E} \in O[\![(\textbf{let } \Gamma \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \Sigma, \circledcirc)]\!]$ and we are done. □

Proof of Lemma A.29. Similar to the proof of Lemma A.14. For any n, σ, Σ and $\widehat{T_o}$ such that $\widehat{T_o} \in \mathcal{T}_\omega^o[\![\Pi, \sigma]\!]$, wfair-o($\widehat{T_o}$) and $\varphi(\sigma) = \Sigma$, suppose

$$\neg\text{abt}(\widehat{T_o}) \quad \text{and} \quad \neg\text{prog-p}(\widehat{T_o}) \quad \text{and} \quad \neg\text{well-blocked-o}(\widehat{T_o}, (\Gamma, \Sigma)) \,.$$

Since $\widehat{T_o} \in \mathcal{T}_\omega^o[\![\Pi, \sigma]\!]$, we know there exist n and $\widehat{T_c}$ such that

$$\widehat{T_c} \in \mathcal{T}_\omega^c[\![\text{MGCp1}_n, \sigma_{\text{MGC}(n)}]\!], \quad \text{get\_hist}(\widehat{T_c}) = \text{get\_hist}(\widehat{T_o}), \quad n = \text{tnum}(\widehat{T_o}),$$
$$\neg\text{abt}(\widehat{T_c}), \quad \text{wfair-c}(\widehat{T_c}), \quad \text{fin\_coherent}(\widehat{T_c}, \widehat{T_o}) \,.$$

By Composition Theorem A.2, we know there exists T such that

$$T \in \mathcal{T}_\omega[\![\textbf{let } \Pi \textbf{ in } \text{MGCp1}_n, (\sigma_{\text{MGC}(n)}, \sigma, \circledcirc)]\!], \quad \widehat{T_c} = \text{get\_clt}(T), \quad \widehat{T_o} = \text{get\_obj}(T).$$

Since wfair-o($\widehat{T_o}$) and wfair-c($\widehat{T_c}$), by Lemma A.10, we know

$$\text{wfair}(T).$$

Since $\varphi(\sigma) = \Sigma$ and $\Pi \sqsubseteq^{\text{wfair}}_{\text{wr}^{\text{wfair}}_{\text{PDF}}(\varphi)} \text{wr}^{\text{wfair}}_{\text{PDF}}(\Gamma)$, we know:

$$O_{\text{wfair}}[\![(\textbf{let } \Pi \textbf{ in } \text{MGCp1}_n), (\sigma_{\text{MGC}(n)}, \sigma)]\!] \subseteq$$
$$O_{\text{wfair}}[\![(\textbf{let } \text{wr}^{\text{wfair}}_{\text{PDF}}(\Gamma) \textbf{ in } \text{MGCp1}_n), (\sigma_{\text{MGC}(n)}, \Sigma \uplus \{\text{done} \rightsquigarrow \text{false}\})]\!] \,.$$

Thus there exists T' such that

$$T' \in \mathcal{T}_\omega[\![(\textbf{let } \text{wr}^{\text{wfair}}_{\text{PDF}}(\Gamma) \textbf{ in } \text{MGCp1}_n), (\sigma_{\text{MGC}(n)}, \Sigma \uplus \{\text{done} \rightsquigarrow \text{false}\}, \circledcirc)]\!] \,,$$
$$\text{wfair}(T') \quad \text{and} \quad \text{get\_obsv}(T') = \text{get\_obsv}(T) \,.$$

Also, by the definition of MGCp1_n and the operational semantics, we can construct T_1 and an execution such that

$$T_1 \in \mathcal{T}_\omega[\![(\textbf{let } \text{wr}^{\text{wfair}}_{\text{PDF}}(\Gamma) \textbf{ in } \text{MGCp1}_n), (\sigma_{\text{MGC}(n)}, \Sigma \uplus \{\text{done} \rightsquigarrow \text{false}\}, \circledcirc)]\!],$$
$$\text{wfair}(T_1), \quad \text{get\_obsv}(T_1) = \mathcal{E}, \quad \text{get\_hist}(T) = \text{get\_hist}(T_1).$$

By Decomposition Theorem A.1, we know there exists $\widehat{T_1}$ such that

$$\widehat{T_1} \in \mathcal{T}_\omega^o[\![\text{wr}^{\text{wfair}}_{\text{PDF}}(\Gamma), \Sigma \uplus \{\text{done} \rightsquigarrow \text{false}\}]\!], \quad \widehat{T_1} = \text{get\_obj}(T_1), \quad n = \text{tnum}(\widehat{T_1}).$$

Since wfair(T_1), by Lemma A.9, we know

$$\text{wfair-o}(\widehat{T_1}).$$

Since $\widehat{T_1} = \text{get\_obj}(T_1)$, $\widehat{T_o} = \text{get\_obj}(T)$ and $\text{get\_hist}(T) = \text{get\_hist}(T_1)$, we know

$$\text{get\_hist}(\widehat{T_o}) = \text{get\_hist}(\widehat{T_1}) \,.$$

Since $n = \text{tnum}(\widehat{T_o})$ and $n = \text{tnum}(\widehat{T_1})$, we know

$$\text{tnum}(\widehat{T_o}) = \text{tnum}(\widehat{T_1}) \,.$$

Since $\neg\text{prog-p}(\widehat{T_o})$ and $\text{get\_hist}(\widehat{T_o}) = \text{get\_hist}(\widehat{T_1})$, we know

$$\exists e.\ e \in \text{pend\_inv}(\widehat{T_1}) \wedge ((|\widehat{T_1}| \neq \omega) \vee \exists i.\ \forall j \geq i.\ \neg\text{is\_ret}(\text{evt}(\widehat{T_1}(j)))) \,.$$

Since wfair-o($\widehat{T_1}$), we know

$$\forall \text{t} \in [1..\text{tnum}(\widehat{T_1})].\ \text{term-o}(\widehat{T_1}|_\text{t}) \vee (\text{t} \in \text{bset}(\text{last}(\widehat{T_1}))) \vee \text{i-o-dis}(\text{t}, \widehat{T_1}) \vee (|(\widehat{T_1}|_\text{t})| = \omega) \,.$$

Thus, by the operational semantics, we know

$$|\widehat{T_1}| = \omega \implies (\forall e.\ e \in \text{pend\_inv}(\widehat{T_1}) \implies \text{e-a-dis}(\text{tid}(e), \widehat{T_1})) \wedge (\exists i.\ \forall j \geq i.\ \neg\text{is\_ret}(\text{evt}(\widehat{T_1}(j)))),$$
$$|\widehat{T_1}| \neq \omega \implies (\forall e.\ e \in \text{pend\_inv}(\widehat{T_1}) \implies \text{tid}(e) \in \text{bset}(\text{last}(\widehat{T_1}))).$$

By Lemma A.32, we know there exists $\widehat{T_a}$ such that

$$\widehat{T}_a \in \mathcal{T}_\omega^o[\![\Gamma,\Sigma]\!],\quad \mathsf{get\_hist}(\widehat{T}_a) = \mathsf{get\_hist}(\widehat{T}_1),\quad \mathsf{tnum}(\widehat{T}_a) = \mathsf{tnum}(\widehat{T}_1),$$
$$|\widehat{T}_a| = \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_a) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e), \widehat{T}_a)),$$
$$|\widehat{T}_a| \neq \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_a) \implies \mathsf{tid}(e) \in \mathsf{bset}(\mathsf{last}(\widehat{T}_a))).$$

Thus well-blocked-o$(\widehat{T}_o, (\Gamma,\Sigma))$ holds, which contradicts our assumption. Thus we are done. □

Lemma A.32. *If* $\widehat{T}_1 \in \mathcal{T}_\omega^o[\![\mathsf{wr}_{\mathsf{PDF}}^{\mathsf{wfair}}(\Gamma), \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}]\!]$ *and*
$|\widehat{T}_1| = \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_1) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e), \widehat{T}_1)) \wedge (\exists i.\ \forall j \geq i.\ \neg\mathsf{is\_ret}(\mathsf{evt}(\widehat{T}_1(j))))$
and $|\widehat{T}_1| \neq \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_1) \implies \mathsf{tid}(e) \in \mathsf{bset}(\mathsf{last}(\widehat{T}_1)))$*, then there exists* $\widehat{T}_a$ *such that*

$$\widehat{T}_a \in \mathcal{T}_\omega^o[\![\Gamma,\Sigma]\!],\quad \mathsf{get\_hist}(\widehat{T}_a) = \mathsf{get\_hist}(\widehat{T}_1),\quad \mathsf{tnum}(\widehat{T}_a) = \mathsf{tnum}(\widehat{T}_1),$$
$$|\widehat{T}_a| = \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_a) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e), \widehat{T}_a)),$$
$$|\widehat{T}_a| \neq \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_a) \implies \mathsf{tid}(e) \in \mathsf{bset}(\mathsf{last}(\widehat{T}_a))).$$

Proof. Since $\widehat{T}_1 \in \mathcal{T}_\omega^o[\![\mathsf{wr}_{\mathsf{PDF}}^{\mathsf{wfair}}(\Gamma), \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}]\!]$, we know there exist n, $\widehat{T}$ and $\widetilde{C}_0$ such that $\widehat{T}_1 = ((\mathbf{spawn}, n), \emptyset) :: \mathsf{get\_obj}(\widehat{T})$ and

$$(\widetilde{C}_0, (\Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}, \circledcirc)) \overset{\widehat{T}}{\mapsto}{}^{\omega}_{\mathsf{wr}_{\mathsf{PDF}}^{\mathsf{wfair}}(\Gamma)} \cdot \quad \text{or} \quad (\widetilde{C}_0, (\Sigma, \circledcirc)) \overset{\widehat{T}}{\mapsto}{}^{*}_{\mathsf{wr}_{\mathsf{PDF}}^{\mathsf{wfair}}(\Gamma)} \mathbf{abort} \quad \text{or}$$
$$\exists \widetilde{C}_1', \Sigma_1', \mathcal{K}_1'.\ ((\widetilde{C}_0, (\Sigma, \circledcirc)) \overset{\widehat{T}}{\mapsto}{}^{*}_{\mathsf{wr}_{\mathsf{PDF}}^{\mathsf{wfair}}(\Gamma)} (\widetilde{C}_1', (\Sigma_1', \mathcal{K}_1'))) \wedge \neg(\exists \widehat{\imath}.\ (\widetilde{C}_1', (\Sigma_1', \mathcal{K}_1')) \overset{\widehat{\imath}}{\mapsto}{}_{\mathsf{wr}_{\mathsf{PDF}}^{\mathsf{wfair}}(\Gamma)} \_).$$

Below we only consider the case when $|\widehat{T}| = \omega$. Proof for the case when $|\widehat{T}| \neq \omega$ is similar. We construct the trace $\widehat{T}_a$ such that $((\mathbf{spawn}, n), \emptyset) :: \mathsf{get\_obj}(\widehat{T}_a)$ satisfies the desired properties. The idea is to construct a simulation between the executions. Informally, our construction is similar to the one in the proof of Lemma A.28.

Then, suppose $|\widehat{T}_a| = \omega$. By the construction of $\widehat{T}_a$, we know $|\widehat{T}_1| = \omega$. For any $e \in \mathsf{pend\_inv}(\widehat{T}_a)$ and $\mathsf{t} = \mathsf{tid}(e)$, by the operational semantics and the code of Γ, we know $|(\widehat{T}_a|_{\mathsf{t}})| \neq \omega$. Since $\mathsf{get\_hist}(\widehat{T}_a) = \mathsf{get\_hist}(\widehat{T}_1)$, we know $e \in \mathsf{pend\_inv}(\widehat{T}_1)$ and $|(\widehat{T}_1|_{\mathsf{t}})| \neq \omega$. Thus $\mathsf{e\text{-}a\text{-}dis}(\mathsf{t}, \widehat{T}_1))$ holds. Suppose the configurations deriving the trace $\widehat{T}_1$ are $(\widetilde{C}_0, (\sigma_0, \mathcal{K}_0)), (\widetilde{C}_1, (\sigma_1, \mathcal{K}_1)), (\widetilde{C}_2, (\sigma_2, \mathcal{K}_2)), \ldots$, then one of the following holds:

- There exist i, B, C and $\mathbf{E}$ such that
$$\forall j \geq i.\ \widetilde{C}_j|_{\mathsf{t}} = \mathbf{E}[\,\mathbf{await}(B \wedge \neg\mathsf{done})\{C; \mathsf{done} := \mathsf{true};\}\,],$$
$$\text{and}\quad \forall j \geq i.\ \neg((\sigma_j, \mathcal{K}_j(\mathsf{t})) \models (B \wedge \neg\mathsf{done})).$$
Since $\exists i'.\ \forall j' \geq i'.\ \neg\mathsf{is\_ret}(\mathsf{evt}(\widehat{T}_1(j')))$, we know there exists $i_1 \geq i$ such that
$$\forall j_1 \geq i_1.\ \neg((\sigma_{j_1}, \mathcal{K}_{j_1}(\mathsf{t})) \models B)\,.$$
Thus $\mathsf{e\text{-}a\text{-}disabled}(\mathsf{t}, \widehat{T}_a)$.
- There exist i and $\mathbf{E}$ such that
$$\forall j \geq i.\ \widetilde{C}_j|_{\mathsf{t}} = \mathbf{E}[\,\mathbf{await}(\neg\mathsf{done})\{\}\,],$$
$$\text{and}\quad \forall j \geq i.\ (\sigma_j, \mathcal{K}_j(\mathsf{t})) \models \mathsf{done}.$$
But this is impossible, since $\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_1) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e), \widehat{T}_1)$.

Thus we are done. □

Proof of Lemma A.30. Similar to the proof of Lemma A.15. In the proof, we need to apply the following Lemma A.33.

Lemma A.33. *For any* T *and* $\widehat{T}_o$*, if* $\widehat{T}_o = \mathsf{get\_obj}(T)$ *and* $\mathsf{prog\text{-}p}(\widehat{T}_o)$*, then* $\mathsf{prog\text{-}p}(T)$.

Proof. By unfolding the definitions. □

Proof of Lemma A.31. The key is to show the following (A.4).

For any $n, C_1, \ldots, C_n, \sigma_c, \sigma, \Sigma, T$ and T_a such that $\varphi(\sigma) = \Sigma$, if $T \in \mathcal{T}_\omega[\![\textbf{let } \Pi \textbf{ in } C_1 \parallel \ldots \parallel C_n, (\sigma_c, \sigma, \circledcirc)]\!]$, $\neg\mathsf{abt}(T)$ and $\mathsf{wfair}(T)$, then there exists T_a such that $T_a \in \mathcal{T}_\omega[\![\textbf{let } \mathsf{wr}_{\mathsf{PDF}}^{\mathsf{wfair}}(\Gamma) \textbf{ in } C_1 \parallel \ldots \parallel C_n, (\sigma_c, \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}, \circledcirc)]\!]$, $\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$ and $\mathsf{wfair}(T_a)$. (A.4)

Since $T \in \mathcal{T}_\omega[\![\textbf{let } \Pi \textbf{ in } C_1 \parallel \ldots \parallel C_n, (\sigma_c, \sigma, \circledcirc)]\!]$ and $\neg\mathsf{abt}(T)$, by Decomposition Theorem A.1, we know there exist $\widehat{T}_c$ and $\widehat{T}_o$ such that

$$\widehat{T}_c \in \mathcal{T}_\omega^{\mathsf{c}}[\![C_1 \parallel \ldots \parallel C_n, \sigma_c]\!], \quad \widehat{T}_o \in \mathcal{T}_\omega^{\mathsf{o}}[\![\Pi, \sigma]\!],$$
$$\mathsf{get\_hist}(\widehat{T}_c) = \mathsf{get\_hist}(\widehat{T}_o), \quad \widehat{T}_c = \mathsf{get\_clt}(T), \quad \widehat{T}_o = \mathsf{get\_obj}(T), \quad n = \mathsf{tnum}(\widehat{T}_o).$$

Since $\mathsf{wfair}(T)$, by Lemmas A.8 and A.9, we know

$$\mathsf{wfair\text{-}c}(\widehat{T}_c) \quad \text{and} \quad \mathsf{wfair\text{-}o}(\widehat{T}_o).$$

Thus we know

$$(|\widehat{T}_c| \neq \omega) \implies \forall \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T}_c)].\ \mathsf{term\text{-}c}(\widehat{T}_c|_{\mathsf{t}}) \vee \mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}_c)) .$$

Since $\mathsf{PDF\text{-}O}_{\varphi,\Gamma}^{\mathsf{wfair}}(\Pi)$, we know $\mathsf{prog\text{-}p}(\widehat{T}_o) \vee \mathsf{well\text{-}blocked\text{-}o}(\widehat{T}_o, (\Gamma, \Sigma))$ holds.

- $\mathsf{prog\text{-}p}(\widehat{T}_o)$ holds. Since $\Pi \sqsubseteq_\varphi^{\mathsf{fin}} \Gamma$, we know
$$\mathcal{H}[\![\Pi, \sigma]\!] \subseteq \mathcal{H}[\![\Gamma, \Sigma]\!].$$
By Lemma A.34, we know there exists $\widehat{T}_a$ such that
$$\widehat{T}_a \in \mathcal{T}_\omega^{\mathsf{o}}[\![\Gamma, \Sigma]\!], \quad \mathsf{get\_hist}(\widehat{T}_a) = \mathsf{get\_hist}(\widehat{T}_o), \quad \mathsf{tnum}(\widehat{T}_a) = \mathsf{tnum}(\widehat{T}_o).$$
By Lemma A.35, we know there exists $\widehat{T}_1$ such that
$$\widehat{T}_1 \in \mathcal{T}_\omega^{\mathsf{o}}[\![\mathsf{wr}_{\mathsf{PDF}}^{\mathsf{wfair}}(\Gamma), \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}]\!],$$
$$\mathsf{get\_hist}(\widehat{T}_a) = \mathsf{get\_hist}(\widehat{T}_1), \quad \mathsf{tnum}(\widehat{T}_a) = \mathsf{tnum}(\widehat{T}_1), \quad \mathsf{wfair\text{-}o}(\widehat{T}_1).$$
Thus $\mathsf{get\_hist}(\widehat{T}_c) = \mathsf{get\_hist}(\widehat{T}_1)$, $n = \mathsf{tnum}(\widehat{T}_1)$ and $\neg\mathsf{abt}(\widehat{T}_1)$. Thus
$$(|\widehat{T}_1| \neq \omega) \implies \forall \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T}_1)].\ \mathsf{term\text{-}o}(\widehat{T}_1|_{\mathsf{t}}) \vee (\mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}_1))).$$
Thus $\mathsf{fin\_coherent}(\widehat{T}_c, \widehat{T}_1)$ holds. By Composition Theorem A.2, we know there exists T_1 such that
$$T_1 \in \mathcal{T}_\omega[\![\textbf{let } \mathsf{wr}_{\mathsf{PDF}}^{\mathsf{wfair}}(\Gamma) \textbf{ in } C_1 \parallel \ldots \parallel C_n, (\sigma_c, \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}, \circledcirc)]\!],$$
$$\widehat{T}_c = \mathsf{get\_clt}(T_1), \quad \widehat{T}_1 = \mathsf{get\_obj}(T_1).$$
Thus
$$\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_1).$$
Then, by Lemma A.10, we know
$$\mathsf{wfair}(T_1).$$
- $\mathsf{well\text{-}blocked\text{-}o}(\widehat{T}_o, (\Gamma, \Sigma))$ holds. Thus there exists $\widehat{T}_a$ such that
$$\widehat{T}_a \in \mathcal{T}_\omega^{\mathsf{o}}[\![\Gamma, \Sigma]\!], \quad \mathsf{get\_hist}(\widehat{T}) = \mathsf{get\_hist}(\widehat{T}_a), \quad \mathsf{tnum}(\widehat{T}) = \mathsf{tnum}(\widehat{T}_a),$$
$$|\widehat{T}_a| = \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_a) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e), \widehat{T}_a)),$$
$$|\widehat{T}_a| \neq \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_a) \implies \mathsf{tid}(e) \in \mathsf{bset}(\mathsf{last}(\widehat{T}_a))).$$
By Lemma A.36, we know there exists $\widehat{T}_1$ such that
$$\widehat{T}_1 \in \mathcal{T}_\omega^{\mathsf{o}}[\![\mathsf{wr}_{\mathsf{PDF}}^{\mathsf{wfair}}(\Gamma), \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}]\!], \quad \mathsf{get\_hist}(\widehat{T}_a) = \mathsf{get\_hist}(\widehat{T}_1),$$
$$\mathsf{tnum}(\widehat{T}_a) = \mathsf{tnum}(\widehat{T}_1),$$
$$|\widehat{T}_1| = \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_1) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e), \widehat{T}_1)),$$
$$|\widehat{T}_1| \neq \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_1) \implies \mathsf{tid}(e) \in \mathsf{bset}(\mathsf{last}(\widehat{T}_1))).$$

Thus $\mathsf{get\_hist}(\widehat{T}_c) = \mathsf{get\_hist}(\widehat{T}_1)$, $n = \mathsf{tnum}(\widehat{T}_1)$ and $\neg\mathsf{abt}(\widehat{T}_1)$. Also we know

$$(|\widehat{T}_1| \neq \omega) \implies \forall \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T}_1)].\ \mathsf{term\text{-}o}(\widehat{T}_1|_{\mathsf{t}}) \vee \mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}_1))\ .$$

Thus $\mathsf{fin\_coherent}(\widehat{T}_c, \widehat{T}_1)$ holds. By Composition Theorem A.2, we know there exists T_1 such that

$$T_1 \in \mathcal{T}_\omega[\![\textbf{let}\ \mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PDF}}(\Gamma)\ \textbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}, \circledcirc)]\!],$$
$$\widehat{T}_c = \mathsf{get\_clt}(T_1), \quad \widehat{T}_1 = \mathsf{get\_obj}(T_1).$$

Thus

$$\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_1).$$

By Lemma A.21, we know

$$\mathsf{sfair\text{-}o}(\widehat{T}_1).$$

Thus we know $\mathsf{wfair\text{-}o}(\widehat{T}_1)$. Then, by Lemma A.10, we know

$$\mathsf{wfair}(T_1).$$

Thus we are done. □

Lemma A.34. *If* $\mathcal{H}[\![\Pi, \sigma]\!] \subseteq \mathcal{H}[\![\Gamma, \Sigma]\!]$, $\widehat{T}_o \in \mathcal{T}^o_\omega[\![\Pi, \sigma]\!]$ *and* $\mathsf{prog\text{-}p}(\widehat{T}_o)$, *then there exists* $\widehat{T}_a$ *such that* $\widehat{T}_a \in \mathcal{T}^o_\omega[\![\Gamma, \Sigma]\!]$, $\mathsf{get\_hist}(\widehat{T}_a) = \mathsf{get\_hist}(\widehat{T}_o)$ *and* $\mathsf{tnum}(\widehat{T}_a) = \mathsf{tnum}(\widehat{T}_o)$.

Proof. By constructing simulations. □

Lemma A.35. *If* $\widehat{T}_a \in \mathcal{T}^o_\omega[\![\Gamma, \Sigma]\!]$ *and* $\mathsf{prog\text{-}p}(\widehat{T}_a)$, *then there exists* $\widehat{T}_1$ *such that* $\widehat{T}_1 \in \mathcal{T}^o_\omega[\![\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PDF}}(\Gamma), \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}]\!]$, $\mathsf{get\_hist}(\widehat{T}_a) = \mathsf{get\_hist}(\widehat{T}_1)$, $\mathsf{tnum}(\widehat{T}_a) = \mathsf{tnum}(\widehat{T}_1)$ *and* $\mathsf{wfair\text{-}o}(\widehat{T}_1)$.

Proof. Similar to the proof of Lemma A.36, we can construct the trace $\widehat{T}_1$ such that $\widehat{T}_1 \in \mathcal{T}^o_\omega[\![\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PDF}}(\Gamma), \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}]\!]$, $\mathsf{get\_hist}(\widehat{T}_a) = \mathsf{get\_hist}(\widehat{T}_1)$ and $\mathsf{tnum}(\widehat{T}_a) = \mathsf{tnum}(\widehat{T}_1)$. Since $\mathsf{prog\text{-}p}(\widehat{T}_a)$, we know $\mathsf{prog\text{-}p}(\widehat{T}_1)$.

Suppose $|\widehat{T}_1| \neq \omega$. Since $\mathsf{prog\text{-}p}(\widehat{T}_1)$, we know $\mathsf{pend\_inv}(\widehat{T}_1) = \emptyset$. Thus

$$(|\widehat{T}_1| \neq \omega) \implies \forall \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T}_1)].\ \mathsf{term\text{-}o}(\widehat{T}_1|_{\mathsf{t}}).$$

Suppose $|\widehat{T}_1| = \omega$. For any $\mathsf{t} \in [1..\mathsf{tnum}(\widehat{T}_1)]$, we know either $|(\widehat{T}_1|_{\mathsf{t}})| = \omega$ or $|(\widehat{T}_1|_{\mathsf{t}})| \neq \omega$. Suppose $|(\widehat{T}_1|_{\mathsf{t}})| \neq \omega$. Then, since $\widehat{T}_1 \in \mathcal{T}^o_\omega[\![\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PDF}}(\Gamma), \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}]\!]$, we know

$$|(\widehat{T}_1|_{\mathsf{t}})| = 0 \vee \mathsf{is\_ret}(\mathsf{evt}(\mathsf{last}(\widehat{T}_1|_{\mathsf{t}}))) \vee \mathsf{is\_inv}(\mathsf{evt}(\mathsf{last}(\widehat{T}_1|_{\mathsf{t}}))) \vee \mathsf{evt}(\mathsf{last}(\widehat{T}_1|_{\mathsf{t}})) = (\mathsf{t}, \textbf{obj}).$$

If $|(\widehat{T}_1|_{\mathsf{t}})| = 0 \vee \mathsf{is\_ret}(\mathsf{evt}(\mathsf{last}(\widehat{T}_1|_{\mathsf{t}})))$ holds, then $\mathsf{wfair\text{-}o}(\widehat{T}_1)$ holds. Otherwise, we know there exists e such that $e \in \mathsf{pend\_inv}(\widehat{T}_1|_{\mathsf{t}})$. Since $\mathsf{prog\text{-}p}(\widehat{T}_1)$, we know

$$\forall i.\ \exists j.\ j > i \wedge \mathsf{is\_ret}(\mathsf{evt}(\widehat{T}_1(j))).$$

By the construction of $\widehat{T}_1$, we know $\mathsf{i\text{-}o\text{-}dis}(\mathsf{t}, \widehat{T}_1)$ holds. Thus we are done. □

Lemma A.36. *If* $\widehat{T}_a \in \mathcal{T}^o_\omega[\![\Gamma, \Sigma]\!]$ *and* $|\widehat{T}_a| = \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_a) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e), \widehat{T}_a))$ *and* $|\widehat{T}_a| \neq \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_a) \implies \mathsf{tid}(e) \in \mathsf{bset}(\mathsf{last}(\widehat{T}_a)))$, *then there exists* $\widehat{T}_1$ *such that*

$$\widehat{T}_1 \in \mathcal{T}^o_\omega[\![\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PDF}}(\Gamma), \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}]\!], \quad \mathsf{get\_hist}(\widehat{T}_a) = \mathsf{get\_hist}(\widehat{T}_1), \quad \mathsf{tnum}(\widehat{T}_a) = \mathsf{tnum}(\widehat{T}_1),$$
$$|\widehat{T}_1| = \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_1) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e), \widehat{T}_1)),$$
$$|\widehat{T}_1| \neq \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_1) \implies \mathsf{tid}(e) \in \mathsf{bset}(\mathsf{last}(\widehat{T}_1))).$$

Proof. Since $\widehat{T}_a \in \mathcal{T}_\omega^o[\![\Gamma,\Sigma]\!]$, we know there exist n, $\widehat{T}$ and $\widetilde{C}_0$ such that $\widehat{T}_a = ((\mathbf{spawn},n),\emptyset) :: \mathsf{get\_obj}(\widehat{T})$ and

$$(\widetilde{C}_0,(\Sigma,\circledcirc)) \overset{\widehat{T}}{\hookrightarrow}_\Gamma^\omega \cdot \quad \text{or} \quad (\widetilde{C}_0,(\Sigma,\circledcirc)) \overset{\widehat{T}}{\hookrightarrow}_\Gamma^* \mathbf{abort} \quad \text{or}$$
$$\exists \widetilde{C}',\Sigma',\mathcal{K}'.\ ((\widetilde{C}_0,(\Sigma,\circledcirc)) \overset{\widehat{T}}{\hookrightarrow}_\Gamma^* (\widetilde{C}',(\Sigma',\mathcal{K}'))) \wedge \neg(\exists \widehat{\iota}.\ (\widetilde{C}',(\Sigma',\mathcal{K}')) \overset{\widehat{\iota}}{\hookrightarrow}_\Gamma \_).$$

Below we only consider the case when $|\widehat{T}| = \omega$. Proof for the case when $|\widehat{T}| \neq \omega$ is similar. We construct the trace $\widehat{T}_1$ such that $((\mathbf{spawn},n),\emptyset) :: \mathsf{get\_obj}(\widehat{T}_1)$ satisfies the desired properties. The idea is to construct a simulation between the executions. Informally, our construction of $\widehat{T}_1$ considers every prefix $\widehat{T}(1..i)$ of $\widehat{T}$ and builds traces $\widehat{T}_1^i$ and their derivations for $\widehat{T}_1^i$. The resulting series are such that for $i < j$, the derivation of $\widehat{T}_1^i$ is a prefix of that of $\widehat{T}_1^j$, which also implies that the trace $\widehat{T}_1^i$ is a prefix of $\widehat{T}_1^j$. Because of this, we could get a simulation relation between the executions, and the limit derivation and the limit trace are the desired $\widehat{T}_1$ and the corresponding derivation. The following claim lies at the core of our construction:

Consider a prefix $\widehat{T}(1..i)$ of $\widehat{T}$, the trace $\widehat{T}_1^i$ and $\widetilde{C}_1$, $\widetilde{C}$, Σ_1', Σ', $\mathcal{K}'$ such that

$$(\widetilde{C}_0,(\Sigma,\circledcirc)) \overset{\widehat{T}(1..i)}{\hookrightarrow}_\Gamma^* (\widetilde{C},(\Sigma',\mathcal{K}')),$$
$$(\widetilde{C}_0,(\Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\},\circledcirc)) \overset{\widehat{T}_1^i}{\hookrightarrow}^*_{\mathsf{wr}_{\mathsf{PDF}}^{\mathsf{wfair}}(\Gamma)} (\widetilde{C}_1,(\Sigma_1',\mathcal{K}')),$$
$$\Sigma_1' = \Sigma' \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\},$$
$$\widetilde{C} \backsim_{\mathcal{K}'} \widetilde{C}_1, \quad \mathsf{get\_hist}(\widehat{T}(1..i)) = \mathsf{get\_hist}(\widehat{T}_1^i).$$

Here $\widetilde{C} \backsim_{\mathcal{K}'} \widetilde{C}_1$ requires the following hold:

$\forall \mathsf{t}.\ (\mathcal{K}'(\mathsf{t}) = \circ) \Longrightarrow (\widetilde{C}_1|_\mathsf{t} = \widetilde{C}|_\mathsf{t})$,

$\forall \mathsf{t}.\ (\mathcal{K}'(\mathsf{t}) \neq \circ) \Longrightarrow (\widetilde{C}|_\mathsf{t} = \widetilde{C}_1|_\mathsf{t}) \vee$
$\exists B,C,E.\ (\widetilde{C}|_\mathsf{t} = (\mathbf{await}(B)\{C\};\mathbf{return}\ E)) \wedge (\widetilde{C}_1|_\mathsf{t} = (\mathsf{wr}_{\mathsf{PDF}}^{\mathsf{wfair}}(\mathbf{await}(B)\{C\});\mathbf{return}\ E)) \vee$
$\exists E.\ (\widetilde{C}|_\mathsf{t} = \mathbf{return}\ E) \wedge (\widetilde{C}_1|_\mathsf{t} = (\mathbf{await}(\neg\mathsf{done})\{\};\mathbf{return}\ E))$.

If $(\widetilde{C},(\Sigma',\mathcal{K}')) \overset{\widehat{T}(i+1)}{\hookrightarrow}_\Gamma (\widetilde{C}',(\Sigma'',\mathcal{K}''))$, then there exist $\widetilde{C}_1'$, Σ_1'' and an extension $\widehat{T}_1^{i+1}$ of $\widehat{T}_1^i$ with the corresponding derivation such that

$$(\widetilde{C}_0,(\Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\},,\circledcirc)) \overset{\widehat{T}_1^{i+1}}{\hookrightarrow}^*_{\mathsf{wr}_{\mathsf{PDF}}^{\mathsf{wfair}}(\Gamma)} (\widetilde{C}_1',(\Sigma_1'',\mathcal{K}'')),$$
$$\Sigma_1'' = \Sigma'' \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\},$$
$$\widetilde{C}' \backsim_{\mathcal{K}''} \widetilde{C}_1', \quad \mathsf{get\_hist}(\widehat{T}(1..i+1)) = \mathsf{get\_hist}(\widehat{T}_1^{i+1}).$$

Also, if $(\widetilde{C},(\Sigma',\mathcal{K}')) \overset{\widehat{T}(i+1)}{\hookrightarrow}_\Gamma \mathbf{abort}$, then there exists an extension $\widehat{T}_1^{i+1}$ of $\widehat{T}_1^i$ such that $(\widetilde{C}_0,(\Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\},,\circledcirc)) \overset{\widehat{T}_1^{i+1}}{\hookrightarrow}^*_{\mathsf{wr}_{\mathsf{PDF}}^{\mathsf{wfair}}(\Gamma)} \mathbf{abort}$ and $\mathsf{get\_hist}(\widehat{T}(1..i+1)) = \mathsf{get\_hist}(\widehat{T}_1^{i+1})$.

To prove the claim, we make a case-split on the derivation of $\widehat{T}(i+1)$.

- If $\widehat{T}(i+1)$ is an abort event, then we could generate the same abort event at the next step of $(\widetilde{C}_1,(\Sigma_1',\mathcal{K}'))$.
- If $\widehat{T}(i+1)$ is a client event, then we could generate the same client event at the next step of $(\widetilde{C}_1,(\Sigma_1',\mathcal{K}'))$.

- If $\widehat{T}(i+1)$ is an invocation event of thread t, then we could generate the same invocation event at the next step of $(\widetilde{C}_1, (\Sigma_1', \mathcal{K}'))$.
- If $\widehat{T}(i+1)$ is an object event and the step is executing $\mathbf{await}(B)\{C\}$ of thread t, then we execute the first two steps (i.e., the first **await** and the reset of done) of $\mathsf{wr}_{\mathsf{PDF}}^{\mathsf{wfair}}(\mathbf{await}(B)\{C\})$ from $(\widetilde{C}_1, (\Sigma_1', \mathcal{K}'))$.
- If $\widehat{T}(i+1)$ is a return event, then we execute the last step (i.e., $\mathbf{await}(\neg\mathsf{done})\{\}$) of $\mathsf{wr}_{\mathsf{PDF}}^{\mathsf{wfair}}(\mathbf{await}(B)\{C\})$ and the return command from $(\widetilde{C}_1, (\Sigma_1', \mathcal{K}'))$. We could generate the same return event.

By the construction, we are done. □

A.5 Proofs of Theorem 6.2-3 (PDF under strong fairness)

By Theorem 4.4, the goal is reduced to the following:

$$\Pi \sqsubseteq_{\varphi}^{\mathsf{fin}} \Gamma \wedge \mathsf{PDF}_{\varphi,\Gamma}^{\mathsf{sfair}}(\Pi) \iff \Pi \sqsubseteq_{\mathsf{wr}_{\mathsf{PDF}}(\varphi)}^{\mathsf{sfair}} \mathsf{wr}_{\mathsf{PDF}}^{\mathsf{sfair}}(\Gamma).$$

By Lemma A.30, we only need to prove the following lemmas.

Lemma A.37. $\Pi \sqsubseteq_{\mathsf{wr}_{\mathsf{PDF}}(\varphi)}^{\mathsf{sfair}} \mathsf{wr}_{\mathsf{PDF}}^{\mathsf{sfair}}(\Gamma) \implies \Pi \sqsubseteq_{\varphi}^{\mathsf{fin}} \Gamma$.

Lemma A.38. $\Pi \sqsubseteq_{\mathsf{wr}_{\mathsf{PDF}}(\varphi)}^{\mathsf{sfair}} \mathsf{wr}_{\mathsf{PDF}}^{\mathsf{sfair}}(\Gamma) \implies \mathsf{PDF\text{-}O}_{\varphi,\Gamma}^{\mathsf{sfair}}(\Pi)$.

Lemma A.39. $\Pi \sqsubseteq_{\varphi}^{\mathsf{fin}} \Gamma \wedge \mathsf{PDF\text{-}O}_{\varphi,\Gamma}^{\mathsf{sfair}}(\Pi) \implies \Pi \sqsubseteq_{\mathsf{wr}_{\mathsf{PDF}}(\varphi)}^{\mathsf{sfair}} \mathsf{wr}_{\mathsf{PDF}}^{\mathsf{sfair}}(\Gamma)$.

Proof of Lemma A.37. For any $n, C_1, \ldots, C_n, \sigma_c, \sigma$ and Σ such that $\varphi(\sigma) = \Sigma$, for any $\mathcal{E}$, if

$$\mathcal{E} \in O[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma, \circledcirc)]\!],$$

we know there exists T_1 such that $\mathcal{E} = \mathsf{get\_obsv}(T_1)$ and

$$T_1 \in \mathcal{T}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma, \circledcirc)]\!].$$

We can construct T_1' and T_1'' such that

$$T_1'' = T_1 :: T_1' \ , \quad \mathsf{sfair}(T_1'') \quad \text{and} \quad T_1'' \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma, \circledcirc)]\!].$$

Since $\Pi \sqsubseteq_{\mathsf{wr}_{\mathsf{PDF}}(\varphi)}^{\mathsf{sfair}} \mathsf{wr}_{\mathsf{PDF}}^{\mathsf{sfair}}(\Gamma)$, we know there exists T_2'' such that

$$T_2'' \in \mathcal{T}_\omega[\![(\mathbf{let}\ \mathsf{wr}_{\mathsf{PDF}}^{\mathsf{sfair}}(\Gamma)\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathbf{false}\}, \circledcirc)]\!] \ , \quad \mathsf{sfair}(T_2'') \quad \text{and}$$
$$\mathsf{get\_obsv}(T_2'') = \mathsf{get\_obsv}(T_1'') = \mathcal{E} :: \mathsf{get\_obsv}(T_1').$$

Thus there exists T_2 such that

$$T_2 \in \mathcal{T}[\![(\mathbf{let}\ \mathsf{wr}_{\mathsf{PDF}}^{\mathsf{sfair}}(\Gamma)\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathbf{false}\}, \circledcirc)]\!] \quad \text{and} \quad \mathsf{get\_obsv}(T_2) = \mathcal{E}.$$

Then we construct T_3 such that

$$T_3 \in \mathcal{T}[\![(\mathbf{let}\ \Gamma\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \Sigma, \circledcirc)]\!], \quad \mathsf{get\_clt}(T_2) = \mathsf{get\_clt}(T_3).$$

Our construction considers every prefix $T_2(1..i)$ of T_2 and builds traces T_3^i and their derivations for T_3^i. The resulting series are such that for $i < j$, the derivation of T_3^i is a prefix of that of T_3^j, which also implies that the trace T_3^i is a prefix of T_3^j. Then $T_3^{|T_2|}$ is the desired T_3. The following claim lies at the core of our construction:

Consider a prefix $T_2(1..i)$ of T_2, the trace T_3^i and $W_1, W, \sigma_c', \Sigma_1', \Sigma', \mathcal{K}'$ such that

$$(\mathbf{let}\ \mathsf{wr}_{\mathsf{PDF}}^{\mathsf{sfair}}(\Gamma)\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathbf{false}\}, \circledcirc)) \xmapsto{T_2(1..i)}{}^{*} (W_1, (\sigma_c', \Sigma_1', \mathcal{K}')),$$
$$(\mathbf{let}\ \Gamma\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \Sigma, \circledcirc)) \xmapsto{T_3^i}{}^{*} (W, (\sigma_c', \Sigma', \mathcal{K}')),$$
$$\Sigma' = \Sigma_1' \backslash \{\mathsf{done}\}, \quad W_1 \smile_{\mathcal{K}'} W, \quad \mathsf{get\_clt}(T_2(1..i)) = \mathsf{get\_clt}(T_3^i).$$

Here $W_1 \smile_{\mathcal{K}'} W$ requires the following hold:

$\forall \mathsf{t}.\ (\mathcal{K}'(\mathsf{t}) = \circ) \implies (W_1|_\mathsf{t} = W|_\mathsf{t}),$

$\forall \mathsf{t}.\ (\mathcal{K}'(\mathsf{t}) \neq \circ) \implies (W_1|_\mathsf{t} = W|_\mathsf{t}) \vee$
$\exists B, C, E.\ (W_1|_\mathsf{t} = (\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\mathbf{await}(B)\{C\}); \mathbf{return}\ E)) \wedge (W|_\mathsf{t} = (\mathbf{await}(B)\{C\}; \mathbf{return}\ E)) \vee$
$\exists B, C, E.\ (W_1|_\mathsf{t} = (\mathbf{await}(B \wedge \neg\mathsf{done})\{C; \ldots\}; \ldots; \mathbf{return}\ E)) \wedge (W|_\mathsf{t} = (\mathbf{await}(B)\{C\}; \mathbf{return}\ E)) \vee$
$\exists E.\ (W_1|_\mathsf{t} = (\mathsf{done} := \mathsf{false}; \mathbf{while}\ (\mathsf{done})\{\}; \mathbf{return}\ E)) \wedge (W|_\mathsf{t} = \mathbf{return}\ E) \vee$
$\exists E.\ (W_1|_\mathsf{t} = (\mathbf{while}\ (\mathsf{done})\{\}; \mathbf{return}\ E)) \wedge (W|_\mathsf{t} = \mathbf{return}\ E).$

If $(W_1, (\sigma_c', \Sigma_1', \mathcal{K}')) \xmapsto{T_2(i+1)} (W_1', (\sigma_c'', \Sigma_1'', \mathcal{K}''))$, then there exist W', Σ'' and an extension T_3^{i+1} of T_3^i with the corresponding derivation such that

$$(\mathbf{let}\ \Gamma\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n, (\sigma_c, \Sigma, \circledcirc)) \xmapsto{T_3^{i+1}}{}^{*} (W', (\sigma_c'', \Sigma'', \mathcal{K}'')),$$
$$\Sigma'' = \Sigma_1''\backslash\{\mathsf{done}\}, \quad W_1' \smile_{\mathcal{K}''} W', \quad \mathsf{get\_clt}(T_2(1..i+1)) = \mathsf{get\_clt}(T_3^{i+1}).$$

Also, if $(W_1, (\sigma_c', \Sigma_1', \mathcal{K}')) \xmapsto{T_2(i+1)} \mathbf{abort}$, then there exists an extension T_3^{i+1} of T_3^i such that $(\mathbf{let}\ \Gamma\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n, (\sigma_c, \Sigma, \circledcirc)) \xmapsto{T_3^{i+1}}{}^{*} \mathbf{abort}$ and $\mathsf{get\_clt}(T_2(1..i+1)) = \mathsf{get\_clt}(T_3^{i+1})$.

To prove the claim, we make a case-split on the derivation of $T_2(i+1)$.

- If $T_2(i+1)$ is a client event, then we could generate the same client event at the next step of $(W, (\sigma_c', \Sigma', \mathcal{K}'))$.
- If $T_2(i+1)$ is an invocation event, then we could generate the same invocation event at the next step of $(W, (\sigma_c', \Sigma', \mathcal{K}'))$.
- If $T_2(i+1)$ is an object event and the step is executing **while** (done){}, then we execute zero step of $(W, (\sigma_c', \Sigma', \mathcal{K}'))$ (that is, $T_3^{i+1} = T_3^i$).
- If $T_2(i+1)$ is an object event and the step is executing $\mathbf{await}(B \wedge \neg\mathsf{done})\{C; \mathsf{done} := \mathsf{true};\}$, then we execute one step of $(W, (\sigma_c', \Sigma', \mathcal{K}'))$ and that step is executing $\mathbf{await}(B)\{C\}$.
- If $T_2(i+1)$ is an object event and the step is executing done := false, then we execute zero step of $(W, (\sigma_c', \Sigma', \mathcal{K}'))$ (that is, $T_3^{i+1} = T_3^i$).
- If $T_2(i+1)$ is a return event, then we could generate the same return event at the next step of $(W, (\sigma_c', \Sigma', \mathcal{K}'))$.

Thus $\mathcal{E} \in O[\![(\mathbf{let}\ \Gamma\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n), (\sigma_c, \Sigma, \circledcirc)]\!]$ and we are done. □

Proof of Lemma A.38. Similar to the proof of Lemma A.14. For any n, σ, Σ and $\widehat{T_o}$ such that $\widehat{T_o} \in \mathcal{T}_\omega^{\mathsf{o}}[\![\Pi, \sigma]\!]$, sfair-o($\widehat{T_o}$) and $\varphi(\sigma) = \Sigma$, suppose

$$\neg\mathsf{abt}(\widehat{T_o}) \quad \text{and} \quad \neg\mathsf{prog\text{-}p}(\widehat{T_o}) \quad \text{and} \quad \neg\mathsf{well\text{-}blocked\text{-}o}(\widehat{T_o}, (\Gamma, \Sigma))\,.$$

Since $\widehat{T_o} \in \mathcal{T}_\omega^{\mathsf{o}}[\![\Pi, \sigma]\!]$, we know there exist n and $\widehat{T_c}$ such that

$$\widehat{T_c} \in \mathcal{T}_\omega^{\mathsf{c}}[\![\mathsf{MGCp1}_n, \sigma_{\mathsf{MGC}(n)}]\!], \quad \mathsf{get\_hist}(\widehat{T_c}) = \mathsf{get\_hist}(\widehat{T_o}), \quad n = \mathsf{tnum}(\widehat{T_o}),$$
$$\neg\mathsf{abt}(\widehat{T_c}), \quad \mathsf{sfair\text{-}c}(\widehat{T_c}), \quad \mathsf{fin\_coherent}(\widehat{T_c}, \widehat{T_o})\,.$$

By Composition Theorem A.2, we know there exists T such that

$$T \in \mathcal{T}_\omega[\![\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGCp1}_n, (\sigma_{\mathsf{MGC}(n)}, \sigma, \circledcirc)]\!], \quad \widehat{T_c} = \mathsf{get\_clt}(T), \quad \widehat{T_o} = \mathsf{get\_obj}(T).$$

Since sfair-o($\widehat{T_o}$) and sfair-c($\widehat{T_c}$), by Lemma A.7, we know

$$\mathsf{sfair}(T).$$

Since $\varphi(\sigma) = \Sigma$ and $\Pi \sqsubseteq^{\mathsf{sfair}}_{\mathsf{wr}_{\mathsf{PDF}}(\varphi)} \mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\Gamma)$, we know:

$$O_{\mathsf{sfair}}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGCp1}_n), (\sigma_{\mathsf{MGC}(n)}, \sigma)]\!] \subseteq$$
$$O_{\mathsf{sfair}}[\![(\mathbf{let}\ \mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\Gamma)\ \mathbf{in}\ \mathsf{MGCp1}_n), (\sigma_{\mathsf{MGC}(n)}, \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\})]\!]\,.$$

Thus there exists T' such that

$$T' \in \mathcal{T}_\omega[\![(\textbf{let } \mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\Gamma) \textbf{ in } \mathsf{MGCp1}_n), (\sigma_{\mathsf{MGC}(n)}, \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}, \circledcirc)]\!], \\ \mathsf{sfair}(T') \quad \text{and} \quad \mathsf{get\_obsv}(T') = \mathsf{get\_obsv}(T)\,.$$

Also, by the definition of $\mathsf{MGCp1}_n$ and the operational semantics, we can construct T_1 and an execution such that

$$T_1 \in \mathcal{T}_\omega[\![(\textbf{let } \mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\Gamma) \textbf{ in } \mathsf{MGCp1}_n), (\sigma_{\mathsf{MGC}(n)}, \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}, \circledcirc)]\!], \\ \mathsf{sfair}(T_1), \quad \mathsf{get\_obsv}(T_1) = \mathcal{E}, \quad \mathsf{get\_hist}(T) = \mathsf{get\_hist}(T_1).$$

By Decomposition Theorem A.1, we know there exists $\widehat{T}_1$ such that

$$\widehat{T}_1 \in \mathcal{T}^o_\omega[\![\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\Gamma), \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}]\!], \quad \widehat{T}_1 = \mathsf{get\_obj}(T_1), \quad n = \mathsf{tnum}(\widehat{T}_1).$$

Since $\mathsf{sfair}(T_1)$, by Lemma A.6, we know

$$\mathsf{sfair\text{-}o}(\widehat{T}_1).$$

Since $\widehat{T}_1 = \mathsf{get\_obj}(T_1)$, $\widehat{T}_o = \mathsf{get\_obj}(T)$ and $\mathsf{get\_hist}(T) = \mathsf{get\_hist}(T_1)$, we know

$$\mathsf{get\_hist}(\widehat{T}_o) = \mathsf{get\_hist}(\widehat{T}_1)\,.$$

Since $n = \mathsf{tnum}(\widehat{T}_o)$ and $n = \mathsf{tnum}(\widehat{T}_1)$, we know

$$\mathsf{tnum}(\widehat{T}_o) = \mathsf{tnum}(\widehat{T}_1)\,.$$

Since $\neg\mathsf{prog\text{-}p}(\widehat{T}_o)$ and $\mathsf{get\_hist}(\widehat{T}_o) = \mathsf{get\_hist}(\widehat{T}_1)$, we know

$$\exists e.\ e \in \mathsf{pend\_inv}(\widehat{T}_1) \wedge ((|\widehat{T}_1| \neq \omega) \vee \exists i.\ \forall j \geq i.\ \neg\mathsf{is\_ret}(\mathsf{evt}(\widehat{T}_1(j))))\,.$$

Since $\mathsf{sfair\text{-}o}(\widehat{T}_1)$, we know

$$\forall \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T}_1)].\ \mathsf{term\text{-}o}(\widehat{T}_1|_{\mathsf{t}}) \vee (\mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}_1))) \vee \mathsf{e\text{-}a\text{-}dis}(\mathsf{t}, \widehat{T}_1) \vee (|(\widehat{T}_1|_{\mathsf{t}})| = \omega)\,.$$

Thus, by the operational semantics, we know

$$|\widehat{T}_1| = \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_1) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e), \widehat{T}_1) \vee (|(\widehat{T}_1|_{\mathsf{tid}(e)})| = \omega)) \\ \wedge (\exists i.\ \forall j \geq i.\ \neg\mathsf{is\_ret}(\mathsf{evt}(\widehat{T}_1(j))))\,,$$

$$|\widehat{T}_1| \neq \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_1) \implies \mathsf{tid}(e) \in \mathsf{bset}(\mathsf{last}(\widehat{T}_1)))\,.$$

By Lemma A.40, we know there exists $\widehat{T}_a$ such that

$$\widehat{T}_a \in \mathcal{T}^o_\omega[\![\Gamma, \Sigma]\!], \quad \mathsf{get\_hist}(\widehat{T}_a) = \mathsf{get\_hist}(\widehat{T}_1), \quad \mathsf{tnum}(\widehat{T}_a) = \mathsf{tnum}(\widehat{T}_1), \\ |\widehat{T}_a| = \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_a) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e), \widehat{T}_a)), \\ |\widehat{T}_a| \neq \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_a) \implies \mathsf{tid}(e) \in \mathsf{bset}(\mathsf{last}(\widehat{T}_a))).$$

Thus $\mathsf{well\text{-}blocked\text{-}o}(\widehat{T}_o, (\Gamma, \Sigma))$ holds, which contradicts our assumption. Thus we are done. □

Lemma A.40. *If* $\widehat{T}_1 \in \mathcal{T}^o_\omega[\![\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\Gamma), \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}]\!]$ *and* $|\widehat{T}_1| = \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_1) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e), \widehat{T}_1) \vee (|(\widehat{T}_1|_{\mathsf{tid}(e)})| = \omega)) \wedge (\exists i.\ \forall j \geq i.\ \neg\mathsf{is\_ret}(\mathsf{evt}(\widehat{T}_1(j))))$ *and* $|\widehat{T}_1| \neq \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_1) \implies \mathsf{tid}(e) \in \mathsf{bset}(\mathsf{last}(\widehat{T}_1)))$*, then there exists* $\widehat{T}_a$ *such that*

$$\widehat{T}_a \in \mathcal{T}^o_\omega[\![\Gamma, \Sigma]\!], \quad \mathsf{get\_hist}(\widehat{T}_a) = \mathsf{get\_hist}(\widehat{T}_1), \quad \mathsf{tnum}(\widehat{T}_a) = \mathsf{tnum}(\widehat{T}_1), \\ |\widehat{T}_a| = \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_a) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e), \widehat{T}_a)), \\ |\widehat{T}_a| \neq \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_a) \implies \mathsf{tid}(e) \in \mathsf{bset}(\mathsf{last}(\widehat{T}_a))).$$

Proof. Since $\widehat{T}_1 \in \mathcal{T}^o_\omega[\![\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\Gamma), \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}]\!]$, we know there exist n, $\widehat{T}$ and $\widetilde{C}_0$ such that $\widehat{T}_1 = ((\textbf{spawn}, n), \emptyset) :: \mathsf{get\_obj}(\widehat{T})$ and

$$(\widetilde{C}_0,(\Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\},\circledcirc)) \overset{\widehat{T}}{\hookrightarrow}{}^{\omega}_{\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\Gamma)} \cdot \quad \text{or} \quad (\widetilde{C}_0,(\Sigma,\circledcirc)) \overset{\widehat{T}}{\hookrightarrow}{}^{*}_{\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\Gamma)} \textbf{abort} \quad \text{or}$$
$$\exists \widetilde{C}_1',\Sigma_1',\mathcal{K}_1'.\ ((\widetilde{C}_0,(\Sigma,\circledcirc)) \overset{\widehat{T}}{\hookrightarrow}{}^{*}_{\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\Gamma)} (\widetilde{C}_1',(\Sigma_1',\mathcal{K}_1'))) \wedge \neg(\exists \widehat{\iota}.\ (\widetilde{C}_1',(\Sigma_1',\mathcal{K}_1')) \overset{\widehat{\iota}}{\hookrightarrow}_{\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\Gamma)} \_).$$

Below we only consider the case when $|\widehat{T}| = \omega$. Proof for the case when $|\widehat{T}| \neq \omega$ is similar. We construct the trace $\widehat{T}_a$ such that $((\mathbf{spawn},n),\emptyset)::\mathsf{get\_obj}(\widehat{T}_a)$ satisfies the desired properties. The idea is to construct a simulation between the executions. Informally, our construction is similar to the one in the proof of Lemma A.37.

Then, suppose $|\widehat{T}_a| = \omega$. By the construction of $\widehat{T}_a$, we know $|\widehat{T}_1| = \omega$. For any $e \in \mathsf{pend\_inv}(\widehat{T}_a)$ and $\mathsf{t} = \mathsf{tid}(e)$, by the operational semantics and the code of Γ, we know $|(\widehat{T}_a|_{\mathsf{t}})| \neq \omega$. Since $\mathsf{get\_hist}(\widehat{T}_a) = \mathsf{get\_hist}(\widehat{T}_1)$, we know $e \in \mathsf{pend\_inv}(\widehat{T}_1)$. Thus $\mathsf{e\text{-}a\text{-}dis}(\mathsf{t},\widehat{T}_1))$ or $|(\widehat{T}_1|_{\mathsf{t}})| = \omega$ holds.

- $|(\widehat{T}_1|_{\mathsf{t}})| = \omega$ holds. Suppose the configurations deriving the trace $\widehat{T}_1$ are $(\widetilde{C}_0,(\sigma_0,\mathcal{K}_0)),(\widetilde{C}_1,(\sigma_1,\mathcal{K}_1)),(\widetilde{C}_2,(\sigma_2,\mathcal{K}_2)),\ldots$, then there exist i, B, C and $\mathbf{E}$ such that
$$\forall j \geq i.\ \widetilde{C}_j|_{\mathsf{t}} = \mathbf{E}[\ \textbf{while}\ (\mathsf{done})\{\}\ ].$$
However, since $\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_1) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e),\widehat{T}_1) \vee (|(\widehat{T}_1|_{\mathsf{tid}(e)})| = \omega)$ and $\exists i'.\ \forall j' \geq i'.\ \neg\mathsf{is\_ret}(\mathsf{evt}(\widehat{T}_1(j')))$, we know there exists $i_1 \geq i$ such that
$$\forall j_1 \geq i_1.\ (\sigma_{j_1},\mathcal{K}_{j_1}(\mathsf{t})) \models \neg\mathsf{done}\,.$$
Thus it is impossible to have $\forall j \geq i.\ \widetilde{C}_j|_{\mathsf{t}} = \mathbf{E}[\ \textbf{while}\ (\mathsf{done})\{\}\ ]$.
- $\mathsf{e\text{-}a\text{-}dis}(\mathsf{t},\widehat{T}_1))$ holds. Suppose the configurations deriving the trace $\widehat{T}_1$ are $(\widetilde{C}_0,(\sigma_0,\mathcal{K}_0)),(\widetilde{C}_1,(\sigma_1,\mathcal{K}_1)),(\widetilde{C}_2,(\sigma_2,\mathcal{K}_2)),\ldots$, then there exist i, B, C and $\mathbf{E}$ such that
$$\forall j \geq i.\ \widetilde{C}_j|_{\mathsf{t}} = \mathbf{E}[\ \textbf{await}(B \wedge \neg\mathsf{done})\{C;\mathsf{done} := \mathsf{true};\}\ ],$$
$$\text{and}\quad \forall j \geq i.\ \neg((\sigma_j,\mathcal{K}_j(\mathsf{t})) \models (B \wedge \neg\mathsf{done})).$$
Since $\exists i'.\ \forall j' \geq i'.\ \neg\mathsf{is\_ret}(\mathsf{evt}(\widehat{T}_1(j')))$, we know there exists $i_1 \geq i$ such that
$$\forall j_1 \geq i_1.\ \neg((\sigma_{j_1},\mathcal{K}_{j_1}(\mathsf{t})) \models B)\,.$$
Thus $\mathsf{e\text{-}a\text{-}dis}(\mathsf{t},\widehat{T}_a)$.

Thus we are done. □

Proof of Lemma A.39. The key is to show the following (A.5).

For any $n, C_1, \ldots, C_n, \sigma_c, \sigma, \Sigma, T$ and T_a such that $\varphi(\sigma) = \Sigma$, if $T \in \mathcal{T}_\omega[\![\textbf{let}\ \Pi\ \textbf{in}\ C_1 \parallel \ldots \parallel C_n,(\sigma_c,\sigma,\circledcirc)]\!]$, $\neg\mathsf{abt}(T)$ and $\mathsf{sfair}(T)$, then there exists T_a such that $T_a \in \mathcal{T}_\omega[\![\textbf{let}\ \mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\Gamma)\ \textbf{in}\ C_1 \parallel \ldots \parallel C_n,(\sigma_c,\Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\},\circledcirc)]\!]$, $\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$ and $\mathsf{sfair}(T_a)$. (A.5)

Since $T \in \mathcal{T}_\omega[\![\textbf{let}\ \Pi\ \textbf{in}\ C_1 \parallel \ldots \parallel C_n,(\sigma_c,\sigma,\circledcirc)]\!]$ and $\neg\mathsf{abt}(T)$, by Decomposition Theorem A.1, we know there exist $\widehat{T}_c$ and $\widehat{T}_o$ such that
$$\widehat{T}_c \in \mathcal{T}^{\mathsf{c}}_{\omega}[\![C_1 \parallel \ldots \parallel C_n,\sigma_c]\!],\quad \widehat{T}_o \in \mathcal{T}^{\mathsf{o}}_{\omega}[\![\Pi,\sigma]\!],$$
$$\mathsf{get\_hist}(\widehat{T}_c) = \mathsf{get\_hist}(\widehat{T}_o),\quad \widehat{T}_c = \mathsf{get\_clt}(T),\quad \widehat{T}_o = \mathsf{get\_obj}(T),\quad n = \mathsf{tnum}(\widehat{T}_o).$$
Since $\mathsf{sfair}(T)$, by Lemmas A.5 and A.6, we know
$$\mathsf{sfair\text{-}c}(\widehat{T}_c) \quad \text{and} \quad \mathsf{sfair\text{-}o}(\widehat{T}_o).$$
Thus we know
$$(|\widehat{T}_c| \neq \omega) \implies \forall \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T}_c)].\ \mathsf{term\text{-}c}(\widehat{T}_c|_{\mathsf{t}}) \vee \mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}_c))\,.$$
Since $\mathsf{PDF\text{-}O}^{\mathsf{sfair}}_{\varphi,\Gamma}(\Pi)$, we know $\mathsf{prog\text{-}p}(\widehat{T}_o) \vee \mathsf{well\text{-}blocked\text{-}o}(\widehat{T}_o,(\Gamma,\Sigma))$ holds.

- prog-p($\widehat{T}_o$) holds. Since $\Pi \sqsubseteq^{\mathsf{fin}}_{\varphi} \Gamma$, we know
$$\mathcal{H}[\![\Pi, \sigma]\!] \subseteq \mathcal{H}[\![\Gamma, \Sigma]\!].$$
By Lemma A.34, we know there exists $\widehat{T}_a$ such that
$$\widehat{T}_a \in \mathcal{T}^{\mathsf{o}}_{\omega}[\![\Gamma, \Sigma]\!], \quad \mathsf{get\_hist}(\widehat{T}_a) = \mathsf{get\_hist}(\widehat{T}_o), \quad \mathsf{tnum}(\widehat{T}_a) = \mathsf{tnum}(\widehat{T}_o).$$
By Lemma A.41, we know there exists $\widehat{T}_1$ such that
$$\widehat{T}_1 \in \mathcal{T}^{\mathsf{o}}_{\omega}[\![\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\Gamma), \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}]\!],$$
$$\mathsf{get\_hist}(\widehat{T}_a) = \mathsf{get\_hist}(\widehat{T}_1), \quad \mathsf{tnum}(\widehat{T}_a) = \mathsf{tnum}(\widehat{T}_1), \quad \mathsf{sfair\text{-}o}(\widehat{T}_1).$$
Thus $\mathsf{get\_hist}(\widehat{T}_c) = \mathsf{get\_hist}(\widehat{T}_1)$, $n = \mathsf{tnum}(\widehat{T}_1)$ and $\neg\mathsf{abt}(\widehat{T}_1)$. Thus
$$(|\widehat{T}_1| \neq \omega) \implies \forall \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T}_1)].\ \mathsf{term\text{-}o}(\widehat{T}_1|_{\mathsf{t}}) \vee (\mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}_1))).$$
Thus $\mathsf{fin\_coherent}(\widehat{T}_c, \widehat{T}_1)$ holds. By Composition Theorem A.2, we know there exists T_1 such that
$$T_1 \in \mathcal{T}_{\omega}[\![\mathbf{let}\ \mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\Gamma)\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}, \circledcirc)]\!],$$
$$\widehat{T}_c = \mathsf{get\_clt}(T_1), \quad \widehat{T}_1 = \mathsf{get\_obj}(T_1).$$
Thus
$$\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_1).$$
Then, by Lemma A.7, we know
$$\mathsf{sfair}(T_1).$$
- well-blocked-o($\widehat{T}_o, (\Gamma, \Sigma)$) holds. Thus there exists $\widehat{T}_a$ such that
$$\widehat{T}_a \in \mathcal{T}^{\mathsf{o}}_{\omega}[\![\Gamma, \Sigma]\!], \quad \mathsf{get\_hist}(\widehat{T}) = \mathsf{get\_hist}(\widehat{T}_a), \quad \mathsf{tnum}(\widehat{T}) = \mathsf{tnum}(\widehat{T}_a),$$
$$|\widehat{T}_a| = \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_a) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e), \widehat{T}_a)),$$
$$|\widehat{T}_a| \neq \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_a) \implies \mathsf{tid}(e) \in \mathsf{bset}(\mathsf{last}(\widehat{T}_a))).$$
By Lemma A.42, we know there exists $\widehat{T}_1$ such that
$$\widehat{T}_1 \in \mathcal{T}^{\mathsf{o}}_{\omega}[\![\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\Gamma), \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}]\!], \quad \mathsf{get\_hist}(\widehat{T}_a) = \mathsf{get\_hist}(\widehat{T}_1),$$
$$\mathsf{tnum}(\widehat{T}_a) = \mathsf{tnum}(\widehat{T}_1),$$
$$|\widehat{T}_1| = \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_1) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e), \widehat{T}_1) \vee (|(\widehat{T}_1|_{\mathsf{tid}(e)})| = \omega)),$$
$$|\widehat{T}_1| \neq \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_1) \implies \mathsf{tid}(e) \in \mathsf{bset}(\mathsf{last}(\widehat{T}_1))).$$
Thus $\mathsf{get\_hist}(\widehat{T}_c) = \mathsf{get\_hist}(\widehat{T}_1)$, $n = \mathsf{tnum}(\widehat{T}_1)$ and $\neg\mathsf{abt}(\widehat{T}_1)$. Also we know
$$(|\widehat{T}_1| \neq \omega) \implies \forall \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T}_1)].\ \mathsf{term\text{-}o}(\widehat{T}_1|_{\mathsf{t}}) \vee \mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}_1))\ .$$
Thus $\mathsf{fin\_coherent}(\widehat{T}_c, \widehat{T}_1)$ holds. By Composition Theorem A.2, we know there exists T_1 such that
$$T_1 \in \mathcal{T}_{\omega}[\![\mathbf{let}\ \mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\Gamma)\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}, \circledcirc)]\!],$$
$$\widehat{T}_c = \mathsf{get\_clt}(T_1), \quad \widehat{T}_1 = \mathsf{get\_obj}(T_1).$$
Thus
$$\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_1).$$
By Lemma A.21, we know
$$\mathsf{sfair\text{-}o}(\widehat{T}_1).$$
Then, by Lemma A.7, we know
$$\mathsf{sfair}(T_1).$$

Thus we are done. □

Lemma A.41. *If* $\widehat{T}_a \in \mathcal{T}_\omega^o[\![\Gamma, \Sigma]\!]$ *and* prog-p($\widehat{T}_a$)*, then there exists* $\widehat{T}_1$ *such that* $\widehat{T}_1 \in \mathcal{T}_\omega^o[\![\mathsf{wr}_{\mathsf{PDF}}^{\mathsf{sfair}}(\Gamma), \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}]\!]$, get_hist($\widehat{T}_a$) = get_hist($\widehat{T}_1$), tnum($\widehat{T}_a$) = tnum($\widehat{T}_1$) *and* sfair-o($\widehat{T}_1$).

Proof. Similar to the proof of Lemma A.42, we can construct the trace $\widehat{T}_1$ such that $\widehat{T}_1 \in \mathcal{T}_\omega^o[\![\mathsf{wr}_{\mathsf{PDF}}^{\mathsf{sfair}}(\Gamma), \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}]\!]$, get_hist($\widehat{T}_a$) = get_hist($\widehat{T}_1$) and tnum($\widehat{T}_a$) = tnum($\widehat{T}_1$). Since prog-p($\widehat{T}_a$), we know prog-p($\widehat{T}_1$).

Suppose $|\widehat{T}_1| \neq \omega$. Since prog-p($\widehat{T}_1$), we know pend_inv($\widehat{T}_1$) = $\emptyset$. Thus

$$(|\widehat{T}_1| \neq \omega) \implies \forall \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T}_1)].\ \mathsf{term\text{-}o}(\widehat{T}_1|_\mathsf{t}).$$

Suppose $|\widehat{T}_1| = \omega$. For any $\mathsf{t} \in [1..\mathsf{tnum}(\widehat{T}_1)]$, we know either $|(\widehat{T}_1|_\mathsf{t})| = \omega$ or $|(\widehat{T}_1|_\mathsf{t})| \neq \omega$. Suppose $|(\widehat{T}_1|_\mathsf{t})| \neq \omega$. Then, since $\widehat{T}_1 \in \mathcal{T}_\omega^o[\![\mathsf{wr}_{\mathsf{PDF}}^{\mathsf{sfair}}(\Gamma), \Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\}]\!]$, we know

$$|(\widehat{T}_1|_\mathsf{t})| = 0 \vee \mathsf{is\_ret}(\mathsf{evt}(\mathsf{last}(\widehat{T}_1|_\mathsf{t}))) \vee \mathsf{is\_inv}(\mathsf{evt}(\mathsf{last}(\widehat{T}_1|_\mathsf{t}))) \vee \mathsf{evt}(\mathsf{last}(\widehat{T}_1|_\mathsf{t})) = (\mathsf{t}, \mathbf{obj}).$$

If $|(\widehat{T}_1|_\mathsf{t})| = 0 \vee \mathsf{is\_ret}(\mathsf{evt}(\mathsf{last}(\widehat{T}_1|_\mathsf{t})))$ holds, then sfair-o($\widehat{T}_1$) holds. Otherwise, we know there exists e such that $e \in \mathsf{pend\_inv}(\widehat{T}_1|_\mathsf{t})$. By the construction of $\widehat{T}_1$, we know $|(\widehat{T}_1|_\mathsf{t})| = \omega$ holds, which contradicts the assumption. Thus we are done. □

Lemma A.42. *If* $\widehat{T}_a \in \mathcal{T}_\omega^o[\![\Gamma, \Sigma]\!]$ *and* $|\widehat{T}_a| = \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_a) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e), \widehat{T}_a))$ *and* $|\widehat{T}_a| \neq \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_a) \implies \mathsf{tid}(e) \in \mathsf{bset}(\mathsf{last}(\widehat{T}_a)))$*, then there exists* $\widehat{T}_1$ *such that*

$$\widehat{T}_1 \in \mathcal{T}_\omega^o[\![\mathsf{wr}_{\mathsf{PDF}}^{\mathsf{sfair}}(\Gamma), \Sigma \uplus \{\mathtt{listid} \rightsquigarrow \epsilon\}]\!], \quad \mathsf{get\_hist}(\widehat{T}_a) = \mathsf{get\_hist}(\widehat{T}_1), \quad \mathsf{tnum}(\widehat{T}_a) = \mathsf{tnum}(\widehat{T}_1),$$
$$|\widehat{T}_1| = \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_1) \implies \mathsf{e\text{-}a\text{-}dis}(\mathsf{tid}(e), \widehat{T}_1) \vee (|(\widehat{T}_1|_{\mathsf{tid}(e)})| = \omega)),$$
$$|\widehat{T}_1| \neq \omega \implies (\forall e.\ e \in \mathsf{pend\_inv}(\widehat{T}_1) \implies \mathsf{tid}(e) \in \mathsf{bset}(\mathsf{last}(\widehat{T}_1))).$$

Proof. Since $\widehat{T}_a \in \mathcal{T}_\omega^o[\![\Gamma, \Sigma]\!]$, we know there exist n, $\widehat{T}$ and $\widetilde{C}_0$ such that $\widehat{T}_a = ((\mathbf{spawn}, n), \emptyset) :: \mathsf{get\_obj}(\widehat{T})$ and

$$(\widetilde{C}_0, (\Sigma, \circledcirc)) \overset{\widehat{T}}{\hookrightarrow}_\Gamma^\omega \cdot \quad \text{or} \quad (\widetilde{C}_0, (\Sigma, \circledcirc)) \overset{\widehat{T}}{\hookrightarrow}_\Gamma^* \mathbf{abort} \quad \text{or}$$
$$\exists \widetilde{C}', \Sigma', \mathcal{K}'.\ ((\widetilde{C}_0, (\Sigma, \circledcirc)) \overset{\widehat{T}}{\hookrightarrow}_\Gamma^* (\widetilde{C}', (\Sigma', \mathcal{K}'))) \wedge \neg(\exists \widehat{\iota}.\ (\widetilde{C}', (\Sigma', \mathcal{K}')) \overset{\widehat{\iota}}{\hookrightarrow}_\Gamma \_).$$

Below we only consider the case when $|\widehat{T}| = \omega$. Proof for the case when $|\widehat{T}| \neq \omega$ is similar. Let

$$S_p = \{(i, \mathsf{t}) \mid \exists e.\ e \in \mathsf{pend\_inv}(\widehat{T}) \wedge \mathsf{t} = \mathsf{tid}(e) \wedge (e = \mathsf{evt}(\widehat{T}(i)))\}.$$

We construct the trace $\widehat{T}_1$ such that $((\mathbf{spawn}, n), \emptyset) :: \mathsf{get\_obj}(\widehat{T}_1)$ satisfies the desired properties. The idea is to construct a simulation between the executions. Informally, our construction of $\widehat{T}_1$ considers every prefix $\widehat{T}(1..i)$ of $\widehat{T}$ and builds traces $\widehat{T}_1^i$ and their derivations for $\widehat{T}_1^i$. The resulting series are such that for $i < j$, the derivation of $\widehat{T}_1^i$ is a prefix of that of $\widehat{T}_1^j$, which also implies that the trace $\widehat{T}_1^i$ is a prefix of $\widehat{T}_1^j$. Because of this, we could get a simulation relation between the executions, and the limit derivation and the limit trace are the desired $\widehat{T}_1$ and the corresponding derivation. The following claim lies at the core of our construction:

Consider a prefix $\widehat{T}(1..i)$ of $\widehat{T}$, the trace $\widehat{T}_1^i$ and $\widetilde{C}_1$, $\widetilde{C}$, Σ_1', Σ', $\mathcal{K}'$ such that

$$(\widetilde{C}_0,(\Sigma,\circledcirc)) \overset{\widehat{T}(1..i)}{\Longrightarrow}{}^{*}_{\Gamma} (\widetilde{C},(\Sigma',\mathcal{K}')),$$
$$(\widetilde{C}_0,(\Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\},\circledcirc)) \overset{\widehat{T}_1^i}{\Longrightarrow}{}^{*}_{\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\Gamma)} (\widetilde{C}_1,(\Sigma'_1,\mathcal{K}')),$$
$$\Sigma'_1 = \Sigma' \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\},$$
$$\widetilde{C} \smile_{\mathcal{K}'} \widetilde{C}_1, \quad \mathsf{get\_hist}(\widehat{T}(1..i)) = \mathsf{get\_hist}(\widehat{T}_1^i).$$

Here $\widetilde{C} \smile_{\mathcal{K}'} \widetilde{C}_1$ requires the following hold:

$\forall \mathsf{t}.\ (\mathcal{K}'(\mathsf{t}) = \circ) \Longrightarrow (\widetilde{C}_1|_{\mathsf{t}} = \widetilde{C}|_{\mathsf{t}})$,

$\forall \mathsf{t}.\ (\mathcal{K}'(\mathsf{t}) \neq \circ) \Longrightarrow (\widetilde{C}|_{\mathsf{t}} = \widetilde{C}_1|_{\mathsf{t}}) \vee$
$\exists B,C,E.\ (\widetilde{C}|_{\mathsf{t}} = (\mathbf{await}(B)\{C\};\mathbf{return}\ E)) \wedge (\widetilde{C}_1|_{\mathsf{t}} = (\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\mathbf{await}(B)\{C\});\mathbf{return}\ E)) \vee$
$\exists E.\ (\widetilde{C}|_{\mathsf{t}} = \mathbf{return}\ E) \wedge (\widetilde{C}_1|_{\mathsf{t}} = (\mathbf{while}\ (\mathsf{done})\{\};\mathbf{return}\ E))$.

If $(\widetilde{C},(\Sigma',\mathcal{K}')) \overset{\widehat{T}(i+1)}{\Longrightarrow}_{\Gamma} (\widetilde{C}',(\Sigma'',\mathcal{K}''))$, then there exist $\widetilde{C}'_1$, Σ''_1 and an extension $\widehat{T}_1^{i+1}$ of $\widehat{T}_1^i$ with the corresponding derivation such that

$$(\widetilde{C}_0,(\Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\},,\circledcirc)) \overset{\widehat{T}_1^{i+1}}{\Longrightarrow}{}^{*}_{\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\Gamma)} (\widetilde{C}'_1,(\Sigma''_1,\mathcal{K}'')),$$
$$\Sigma''_1 = \Sigma'' \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\},$$
$$\widetilde{C}' \smile_{\mathcal{K}''} \widetilde{C}'_1, \quad \mathsf{get\_hist}(\widehat{T}(1..i+1)) = \mathsf{get\_hist}(\widehat{T}_1^{i+1}).$$

Also, if $(\widetilde{C},(\Sigma',\mathcal{K}')) \overset{\widehat{T}(i+1)}{\Longrightarrow}_{\Gamma} \mathbf{abort}$, then there exists an extension $\widehat{T}_1^{i+1}$ of $\widehat{T}_1^i$ such that $(\widetilde{C}_0,(\Sigma \uplus \{\mathsf{done} \rightsquigarrow \mathsf{false}\},,\circledcirc)) \overset{\widehat{T}_1^{i+1}}{\Longrightarrow}{}^{*}_{\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\Gamma)} \mathbf{abort}$ and $\mathsf{get\_hist}(\widehat{T}(1..i+1)) = \mathsf{get\_hist}(\widehat{T}_1^{i+1})$.

To prove the claim, we make a case-split on the derivation of $\widehat{T}(i+1)$.

- If $\widehat{T}(i+1)$ is an abort event, then we could generate the same abort event at the next step of $(\widetilde{C}_1,(\Sigma'_1,\mathcal{K}'))$.
- If $\widehat{T}(i+1)$ is a client event, then we could generate the same client event at the next step of $(\widetilde{C}_1,(\Sigma'_1,\mathcal{K}'))$. If $\forall j > i.\ \mathsf{is\_clt}(\widehat{T}(j))$ holds, then for each thread $\mathsf{t}' \in S_p$, we infinitely often execute the while loop (whose boolean condition must be continuously evaluated to true) of $\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\mathbf{await}(B)\{C\})$ of the thread t'.
- If $\widehat{T}(i+1)$ is an invocation event of thread t, then we could generate the same invocation event at the next step of $(\widetilde{C}_1,(\Sigma'_1,\mathcal{K}'))$.
- If $\widehat{T}(i+1)$ is an object event and the step is executing $\mathbf{await}(B)\{C\}$ of thread t, then we first execute the while-loop (whose boolean condition must be evaluated to false) and the await steps of $\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\mathbf{await}(B)\{C\})$ from $(\widetilde{C}_1,(\Sigma'_1,\mathcal{K}'))$. Then, for each thread t' such that $\exists i' \leq i.\ (i',\mathsf{t}) \in S_p$, we execute one iteration of the while loop (whose boolean condition must be evaluated to true) of $\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\mathbf{await}(B)\{C\})$ of the thread t'. Finally we execute the step of resetting done of $\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\mathbf{await}(B)\{C\})$ of thread t.
- If $\widehat{T}(i+1)$ is a return event, then we execute the second while-loop (whose boolean condition must be evaluated to false) of $\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\mathbf{await}(B)\{C\})$ and the following return command of thread t. We could generate the same return event at the next step of $(\widetilde{C}_1,(\Sigma'_1,\mathcal{K}'))$.

By the construction, we are done. □

$$
\begin{array}{rrcl}
(\mathit{RelAssn}) & P,Q,I & ::= & B \mid \mathsf{own}(x) \mid \mathsf{emp} \mid E \mapsto E \mid E \Rightarrow E \mid \lVert p \rVert \mid P * Q \mid P \wedge Q \mid P \vee Q \mid \ldots \\
(\mathit{FullAssn}) & p,q,J & ::= & P \mid \mathsf{arem}(C) \mid \Diamond(E) \mid \blacklozenge(E_k,\ldots,E_1) \mid \lfloor p \rfloor_{\Diamond} \mid \lfloor p \rfloor_{\mathsf{a}} \mid p * q \mid p \wedge q \mid \ldots \\
(\mathit{RelAct}) & R,G & ::= & P \ltimes_k Q \mid [P] \mid \lfloor G \rfloor_0 \mid \mathcal{D} \mid G * G \mid G \wedge G \mid G \vee G \mid \ldots \\
(\mathit{DAct}) & \mathcal{D} & ::= & P \rightsquigarrow Q \mid \forall x.\, \mathcal{D} \mid \mathcal{D} \wedge \mathcal{D}
\end{array}
$$

Fig. 16. Syntax of assertions. (We highlight the constructs which are new here if compared with Fig. 9.)

B FULL LOGIC AND SOUNDNESS PROOFS

B.1 LRG-style program logic

We have explained the basic ideas and key inference rules of our program logic in Sec. 7. In this section we give the full version, which extends the advanced Rely-Guarantee-based logic LRG [Feng 2009] to support dynamic allocation and ownership transfer. The top level judgment is now in the form of $\mathcal{D},R,G,I \vdash \{P\}\Pi : \Gamma$. Here the fence I is used to determine the boundary of the shared memory following LRG [Feng 2009]. Just like P, I is also a relational assertion specifying the consistency relation between the concrete data representation and the abstract value.

B.1.1 Assertions. The full set of the syntax of the assertions is given in Fig. 16. We highlight the constructs which are not shown in Fig. 9. Following LRG [Feng 2009], we treat program variables as resources [Parkinson et al. 2006] and use $\mathsf{own}(x)$ for the ownership of the program variable x. We use $\lfloor p \rfloor_{\Diamond}$ to ignore the descriptions in p about the number of $\Diamond$-tokens. $\lfloor p \rfloor_{\Diamond}$ will be useful when we want to hide the $\Diamond$-tokens that are introduced for the proofs of **while**-loops. Similarly, $\lfloor p \rfloor_{\mathsf{a}}$ ignores the descriptions in p about $\mathsf{arem}(C)$. It will be used in the inference rule for **return** commands.

Fig. 17 shows the semantics of assertions. Fig. 18 defines the transition levels and the ordering over token numbers. Fig. 19 shows the definitions of $\mathsf{wffAct}(R,\mathcal{D})$, stability and "view shifts" (which execute the abstract code). These definitions are the same as in LiLi [Liang and Feng 2016]. The syntactic sugars Id, Emp and True represent arbitrary identity transitions, empty transitions and arbitrary transitions respectively.

Fence. Since we logically split states into local and shared parts as in LRG [Feng 2009], we need a precise invariant I to uniquely determine the boundary between local and shared resources. We define the fence $I \triangleright G$ in Fig. 17(c), which says that the transition G must be made within the boundary specified by I. Here $\mathsf{Precise}(I)$ follows its usual meaning as in separation logic but is now interpreted over relational states. The need of fence $I \triangleright \{R,G\}$ is inherited from LRG. It is orthogonal to the problems studied in this paper, and readers who are unfamiliar with LRG can safely ignore it.

B.1.2 Inference rules. Figure 20 presents the complete set of inference rules. We have explained the OBJ, WHL, AWAIT-W and AWAIT-S rules in Sec. 7. The slight modifications here are only to distinguish shared and local state assertions. For instance, R and G should describe transitions over shared states only, and be fenced by I.

The AWAIT-W and AWAIT-S rules have the premise $\mathcal{D},[I],G,I \vdash \{p \wedge B\}\langle C\rangle\{q\}$, which can be derived using the ATOM rule. The ATOM rule allows us to logically execute the abstract code simultaneously with every concrete step. We use $\vdash [p]C[q']$ to represent the total correctness of C in sequential separation logic. The corresponding rules are standard and elided here. We use $q' \Rrightarrow_k q$ (defined in Fig. 19) for the zero or multiple-step executions from the abstract code specified by q' to the code specified by q. It also requires the number of $\blacklozenge$-tokens at level k to be decreased if $k \geq 1$. That is, the current thread should lose $\blacklozenge$-tokens when it performs a level-k action that may delay other threads. The ATOM rule allows us to execute zero-or-more steps of the abstract code

$$
\begin{array}{lll}
((s,h),(\mathbb{s},\mathbb{h})) \models B & \text{iff} & [\![B]\!]_{s \uplus \mathbb{s}} = \mathbf{true} \\
((s,h),(\mathbb{s},\mathbb{h})) \models \mathsf{own}(x) & \text{iff} & dom(s \uplus \mathbb{s}) = \{x\} \\
((s,h),(\mathbb{s},\mathbb{h})) \models \mathsf{emp} & \text{iff} & dom(h) = dom(\mathbb{h}) = \emptyset \\
((s,h),(\mathbb{s},\mathbb{h})) \models E_1 \mapsto E_2 & \text{iff} & h = \{[\![E_1]\!]_{s \uplus \mathbb{s}} \rightsquigarrow [\![E_2]\!]_{s \uplus \mathbb{s}}\} \\
((s,h),(\mathbb{s},\mathbb{h})) \models E_1 \Mapsto E_2 & \text{iff} & \mathbb{h} = \{[\![E_1]\!]_{s \uplus \mathbb{s}} \rightsquigarrow [\![E_2]\!]_{s \uplus \mathbb{s}}\} \\
\mathfrak{S} \models P * Q & \text{iff} & \exists \mathfrak{S}_1, \mathfrak{S}_2.\, (\mathfrak{S} = \mathfrak{S}_1 \uplus \mathfrak{S}_2) \wedge (\mathfrak{S}_1 \models P) \wedge (\mathfrak{S}_2 \models Q)
\end{array}
$$

$$
(\sigma,\Sigma) \uplus (\sigma',\Sigma') \stackrel{\text{def}}{=} (\sigma \uplus \sigma', \Sigma \uplus \Sigma') \quad \text{where } (s,h) \uplus (s',h') \stackrel{\text{def}}{=} (s \uplus s', h \uplus h')
$$

(a) Semantics of relational state assertions P and Q.

$$
\begin{array}{lll}
(\mathfrak{S},(u,w),C) \models P & \text{iff} & \mathfrak{S} \models P \\
(\mathfrak{S},(u,w),C) \models \mathsf{arem}(C') & \text{iff} & C = C' \\
(\mathfrak{S},(u,w),C) \models \Diamond(E) & \text{iff} & \exists n.\, ([\![E]\!]_{\mathfrak{S}.s} = n) \wedge (n \le w) \\
(\mathfrak{S},(u,w),C) \models \blacklozenge(E_k,\ldots,E_1) & \text{iff} & ([\![E_k]\!]_{\mathfrak{S}.s},\ldots,[\![E_1]\!]_{\mathfrak{S}.s}) \le u \\
(\mathfrak{S},(u,w),C) \models \lfloor p \rfloor_{\Diamond} & \text{iff} & \exists w'.\, (\mathfrak{S},(u,w'),C) \models p \\
(\mathfrak{S},(u,w),C) \models \lfloor p \rfloor_{\mathsf{a}} & \text{iff} & \exists C'.\, (\mathfrak{S},(u,w),C') \models p \\
\mathfrak{S} \models [\![p]\!] & \text{iff} & \exists u,w,C.\, (\mathfrak{S},(u,w),C) \models p
\end{array}
$$

$$
C \uplus C' \stackrel{\text{def}}{=} \begin{cases} C' & \text{if } C = \mathbf{skip} \\ C & \text{if } C' = \mathbf{skip} \end{cases}
\qquad
\begin{array}{l}
(\mathfrak{S},(u,w),C) \uplus (\mathfrak{S}',(u',w'),C') \stackrel{\text{def}}{=} \\
\qquad (\mathfrak{S} \uplus \mathfrak{S}', (u+u', w+w'), C \uplus C')
\end{array}
$$

(b) Semantics of full assertions p and q.

$$
\begin{array}{lll}
(\mathfrak{S},\mathfrak{S}') \models P \ltimes_{k'} Q & \text{iff} & (\mathfrak{S} \models P) \wedge (\mathfrak{S}' \models Q) \\
(\mathfrak{S},\mathfrak{S}') \models [P] & \text{iff} & (\mathfrak{S}' = \mathfrak{S}) \wedge (\mathfrak{S} \models P) \\
(\mathfrak{S},\mathfrak{S}') \models G_1 * G_2 & \text{iff} & \exists \mathfrak{S}_1,\mathfrak{S}_2,\mathfrak{S}'_1,\mathfrak{S}'_2.\, \mathfrak{S} = \mathfrak{S}_1 \uplus \mathfrak{S}_2 \wedge \mathfrak{S}' = \mathfrak{S}'_1 \uplus \mathfrak{S}'_2 \\
& & \qquad \wedge ((\mathfrak{S}_1,\mathfrak{S}'_1) \models G_1) \wedge ((\mathfrak{S}_2,\mathfrak{S}'_2) \models G_2)
\end{array}
$$

$$
\mathsf{Emp} \stackrel{\text{def}}{=} \mathsf{emp} \ltimes \mathsf{emp} \qquad \mathsf{True} \stackrel{\text{def}}{=} \mathsf{true} \ltimes \mathsf{true} \qquad \mathsf{Id} \stackrel{\text{def}}{=} [\mathsf{true}]
$$

$$
I \triangleright G \;\text{ iff }\; ([I] \Rightarrow G) \wedge (G \Rightarrow (I \ltimes I)) \wedge \mathsf{Precise}(I)
$$

(c) Semantics of relational rely/guarantee assertions R and G.

$$
\begin{array}{lll}
(\mathfrak{S},\mathfrak{S}') \models P \rightsquigarrow Q & \text{iff} & (\mathfrak{S} \models P) \implies (\mathfrak{S}' \models Q) \\
(\mathfrak{S},\mathfrak{S}') \models \forall x.\, \mathcal{D} & \text{iff} & \forall n.\, (\mathfrak{S}\{x \rightsquigarrow n\}, \mathfrak{S}'\{x \rightsquigarrow n\}) \models \mathcal{D} \\
(\mathfrak{S},\mathfrak{S}') \models \mathcal{D}_1 \wedge \mathcal{D}_2 & \text{iff} & ((\mathfrak{S},\mathfrak{S}') \models \mathcal{D}_1) \wedge ((\mathfrak{S},\mathfrak{S}') \models \mathcal{D}_2)
\end{array}
$$

$$
\begin{array}{rcl}
\mathsf{Enabled}(P \rightsquigarrow Q) & \stackrel{\text{def}}{=} & P \\
\mathsf{Enabled}(\forall x.\, \mathcal{D}) & \stackrel{\text{def}}{=} & \exists x.\, \mathsf{Enabled}(\mathcal{D}) \\
\mathsf{Enabled}(\mathcal{D}_1 \wedge \mathcal{D}_2) & \stackrel{\text{def}}{=} & \mathsf{Enabled}(\mathcal{D}_1) \vee \mathsf{Enabled}(\mathcal{D}_2) \\
\\
\langle \mathcal{D} \rangle & \stackrel{\text{def}}{=} & \mathcal{D} \wedge (\mathsf{Enabled}(\mathcal{D}) \ltimes \mathsf{true}) \\
[\mathcal{D}] & \stackrel{\text{def}}{=} & \mathsf{Enabled}(\mathcal{D}) \rightsquigarrow \mathsf{Enabled}(\mathcal{D})
\end{array}
$$

$$
\mathcal{D}' \leqslant \mathcal{D} \;\text{ iff }\; (\mathsf{Enabled}(\mathcal{D}') \Rightarrow \mathsf{Enabled}(\mathcal{D})) \wedge (\mathcal{D} \Rightarrow \mathcal{D}')
$$

(d) Semantics of definite actions $\mathcal{D}$.

Fig. 17. Semantics of assertions.

with the execution of C, as long as the overall transition (including the abstract steps, the concrete

$$\mathcal{L}((\mathfrak{S},\mathfrak{S}'),P \ltimes_k Q) \stackrel{\text{def}}{=} \begin{cases} k & \text{if } (\mathfrak{S},\mathfrak{S}') \models P \ltimes_k Q \\ \mathsf{maxL} & \text{otherwise} \end{cases}$$

$$\mathcal{L}((\mathfrak{S},\mathfrak{S}'),[P]) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } (\mathfrak{S},\mathfrak{S}') \models [P] \\ \mathsf{maxL} & \text{otherwise} \end{cases}$$

$$\mathcal{L}((\mathfrak{S},\mathfrak{S}'),\mathcal{D}) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } (\mathfrak{S},\mathfrak{S}') \models \mathcal{D} \\ \mathsf{maxL} & \text{otherwise} \end{cases}$$

$$\mathcal{L}((\mathfrak{S},\mathfrak{S}'),R \wedge R') \stackrel{\text{def}}{=} \max(\mathcal{L}((\mathfrak{S},\mathfrak{S}'),R),\mathcal{L}((\mathfrak{S},\mathfrak{S}'),R'))$$

$$\mathcal{L}((\mathfrak{S},\mathfrak{S}'),R \vee R') \stackrel{\text{def}}{=} \min(\mathcal{L}((\mathfrak{S},\mathfrak{S}'),R),\mathcal{L}((\mathfrak{S},\mathfrak{S}'),R'))$$

$$\mathcal{L}((\mathfrak{S},\mathfrak{S}'),\lfloor R \rfloor_0) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } \mathcal{L}((\mathfrak{S},\mathfrak{S}'),R) = 0 \\ \mathsf{maxL} & \text{otherwise} \end{cases}$$

$$\mathcal{L}((\mathfrak{S},\mathfrak{S}'),G_1 * G_2) \stackrel{\text{def}}{=} min\{k \mid \exists \mathfrak{S}_1,\mathfrak{S}_2,\mathfrak{S}'_1,\mathfrak{S}'_2.\ (\mathfrak{S} = \mathfrak{S}_1 \uplus \mathfrak{S}_2) \wedge (\mathfrak{S}' = \mathfrak{S}'_1 \uplus \mathfrak{S}'_2) \wedge k = max(\mathcal{L}((\mathfrak{S}_1,\mathfrak{S}'_1),G_1),\mathcal{L}((\mathfrak{S}_2,\mathfrak{S}'_2),G_2))\}$$

$$(\mathfrak{S},\mathfrak{S}') \models \lfloor R \rfloor_0 \text{ iff } \mathcal{L}((\mathfrak{S},\mathfrak{S}'),R) = 0$$

$$(\mathfrak{S},\mathfrak{S}',k) \models R \text{ iff } \mathcal{L}((\mathfrak{S},\mathfrak{S}'),R) = k \text{ and } k < \mathsf{maxL}$$

$$R \Rightarrow R' \text{ iff } \forall \mathfrak{S},\mathfrak{S}',k.\ ((\mathfrak{S},\mathfrak{S}',k) \models R) \Longrightarrow \exists k' \le k.\ (\mathfrak{S},\mathfrak{S}',k') \models R'$$

$$u ::= (n_k,\ldots,n_1) \quad (1 \le k < \mathsf{maxL})$$

$$(n'_m,\ldots,n'_1) <_k (n_m,\ldots,n_1) \text{ iff } (\forall i > k.\ (n'_i = n_i)) \wedge (n'_k < n_k)$$

$$(n'_m,\ldots,n'_1) \approx_k (n_m,\ldots,n_1) \text{ iff } (\forall i \ge k.\ (n'_i = n_i))$$

$$u < u' \text{ iff } \exists k.\ u <_k u' \qquad u \le u' \text{ iff } u < u' \vee u = u'$$

$$(u,w) <_k (u',w') \text{ iff } (u <_k u') \vee (k = 0 \wedge u = u' \wedge w = w')$$

$$(u,w) \approx_k (u',w') \text{ iff } u \approx_k u' \wedge (k = 0 \Longrightarrow w = w')$$

Fig. 18. Levels of state transitions and tokens.

$$\mathsf{wffAct}(R,\mathcal{D}) \text{ iff } \forall \mathsf{t}.\ \lfloor R_\mathsf{t} \rfloor_0 \Rightarrow [\mathcal{D}_\mathsf{t}] \wedge (\forall \mathsf{t}' \neq \mathsf{t}.\ [\mathcal{D}_{\mathsf{t}'}] \vee \mathcal{D}_{\mathsf{t}'})$$

$$\begin{aligned} p \Rrightarrow_k q \text{ iff } & \forall \mathsf{t},\sigma,\Sigma,u,w,C,\Sigma_F. \\ & (((\sigma,\Sigma),(u,w),C) \models p) \wedge (\Sigma \perp \Sigma_F) \Longrightarrow \exists u',w',C',\Sigma'. \\ & ((C,\Sigma \uplus \Sigma_F) \longrightarrow^*_\mathsf{t} (C',\Sigma' \uplus \Sigma_F)) \wedge (((\sigma,\Sigma'),(u',w'),C') \models q) \\ & \wedge (u',w') <_k (u,w) \end{aligned}$$

$$\begin{aligned} p \overset{G}{\Rrightarrow} q \text{ iff } & \forall \mathsf{t},\sigma,\Sigma,u,w,C,\Sigma_F. \\ & (((\sigma,\Sigma),(u,w),C) \models p) \wedge (\Sigma \perp \Sigma_F) \Longrightarrow \exists k,u',w',C',\Sigma'. \\ & ((C,\Sigma \uplus \Sigma_F) \longrightarrow^*_\mathsf{t} (C',\Sigma' \uplus \Sigma_F)) \wedge (((\sigma,\Sigma),(\sigma,\Sigma'),k) \models G * \mathsf{True}) \\ & \wedge (((\sigma,\Sigma'),(u',w'),C') \models q) \wedge (u',w') <_k (u,w) \end{aligned}$$

$$\begin{aligned} \mathsf{Sta}(p,R) \text{ iff } & \forall \mathfrak{S},\mathfrak{S}',u,w,C,k. \\ & ((\mathfrak{S},(u,w),C) \models p) \wedge ((\mathfrak{S},\mathfrak{S}',k) \models R) \Longrightarrow \exists u',w'. \\ & ((\mathfrak{S}',(u',w'),C) \models p) \ \wedge \ ((u',w') \approx_k (u,w)) \end{aligned}$$

Fig. 19. Key auxiliary definitions for inference rules.

steps and the level k) satisfies the relational guarantee G. We can lift C's total correctness to the concurrent setting as long as the environment consists of identity transitions only.

$$\frac{\begin{array}{c}\text{for all } f \in dom(\Pi):\quad \Pi(f) = (\mathcal{P}, x, C) \qquad \Gamma(f) = (\mathcal{P}', y, C') \qquad P \Rightarrow (\mathcal{P} \wedge \mathcal{P}') \vee (\neg\mathcal{P} \wedge \neg\mathcal{P}') \\ \mathcal{D}, R, G \vdash \{(P \wedge \mathcal{P}) * \mathsf{own}(x,y) \wedge (x = y) \wedge \mathsf{arem}(C') \wedge \blacklozenge(E_k, \ldots, E_1)\}\, C\, \{P * \mathsf{own}(x,y) \wedge \mathsf{arem}(\mathbf{skip})\} \\ \forall \mathsf{t}, \mathsf{t}'.\ \mathsf{t} \neq \mathsf{t}' \Longrightarrow G_{\mathsf{t}} \Rightarrow R_{\mathsf{t}'} \qquad \mathsf{wffAct}(R, \mathcal{D}) \qquad P \Rightarrow \neg\mathsf{Enabled}(\mathcal{D}) \qquad P \vee \mathsf{Enabled}(\mathcal{D}) \Rightarrow I\end{array}}{\mathcal{D}, R, G, I \vdash \{P\}\Pi : \Gamma}\ (\textsc{obj})$$

$$\frac{\begin{array}{c}p \wedge B \Rightarrow p' \qquad p \wedge B \wedge (\mathsf{Enabled}(\mathcal{D}) \vee Q) * \mathsf{true} \Rightarrow p' * (\Diamond \wedge \mathsf{emp}) \qquad \mathcal{D}, R, G, I \vdash \{p'\} C \{p\} \\ p \Rightarrow (B = B) * I \qquad J \vee Q \Rightarrow I \qquad \mathsf{Sta}(J, R \vee G) \qquad \mathcal{D}' \leqslant \mathcal{D} \qquad \mathsf{wffAct}(R, \mathcal{D}') \\ p \wedge B \Rightarrow J * \mathsf{true} \wedge \mathsf{arem}(\mathbf{await}(B')\{C'\}) \qquad J \Rightarrow (R, G\colon \mathcal{D}' \xrightarrow{f} (Q, B'))\end{array}}{\mathcal{D}, R, G, I \vdash \{p\}\mathbf{while}\ (B)\{C\}\{p \wedge \neg B\}}\ (\textsc{whl})$$

$$\frac{\begin{array}{c}p \wedge \mathsf{Enabled}(\mathcal{D}) * \mathsf{true} \Rightarrow B \qquad \mathcal{D}, [I], G, I \vdash \{p \wedge B\}\langle C\rangle\{q\} \qquad \mathsf{Sta}(\{p,q\}, R * \mathsf{Id}) \\ \mathcal{D}' \leqslant \mathcal{D} \qquad \mathsf{wffAct}(R, \mathcal{D}') \qquad p \Rightarrow \exists B', C'.\ \mathsf{arem}(\mathbf{await}(B')\{C'\}) \wedge (R\colon \mathcal{D}' \circ\!\!\xrightarrow{f} (B, B'))\end{array}}{\mathcal{D}, R, G, I \vdash_{\mathsf{wfair}} \{p\}\mathbf{await}(B)\{C\}\{q\}}\ (\textsc{await-w})$$

$$\frac{\begin{array}{c}p \wedge \mathsf{Enabled}(\mathcal{D}) * \mathsf{true} \Rightarrow B \qquad \mathcal{D}, [I], G, I \vdash \{p \wedge B\}\langle C\rangle\{q\} \qquad \mathsf{Sta}(\{p,q\}, R * \mathsf{Id}) \\ \mathcal{D}' \leqslant \mathcal{D} \qquad \mathsf{wffAct}(R, \mathcal{D}') \qquad p \Rightarrow \exists B', C'.\ \mathsf{arem}(\mathbf{await}(B')\{C'\}) \wedge (R\colon \mathcal{D}' \bullet\!\!\xrightarrow{f} (B, B'))\end{array}}{\mathcal{D}, R, G, I \vdash_{\mathsf{sfair}} \{p\}\mathbf{await}(B)\{C\}\{q\}}\ (\textsc{await-s})$$

$$\frac{\vdash [p]C[q'] \qquad I \triangleright G \qquad p \vee q \Rightarrow I * \mathsf{true} \qquad q' \Rrightarrow_k q \qquad (\llbracket p \rrbracket \ltimes_k \llbracket q \rrbracket) \Rightarrow G * \mathsf{True}}{\mathcal{D}, [I], G, I \vdash \{p\}\langle C\rangle\{q\}}\ (\textsc{atom})$$

$$\frac{\vdash [p]C[q] \qquad \mathsf{Sta}(r, R * \mathsf{Id}) \qquad I \triangleright \{G, R\} \qquad r \Rightarrow I * \mathsf{true}}{\mathcal{D}, R, G, I \vdash \{p * r\}C\{q * r\}}\ (\textsc{prim})$$

$$\frac{p \Rightarrow (E = \mathbb{E}) * I \qquad \mathsf{Sta}(p, R * \mathsf{Id}) \qquad I \triangleright \{R, G\}}{\mathcal{D}, R, G, I \vdash \{\lfloor p \rfloor_{\mathsf{a}} \wedge \mathsf{arem}(\mathbf{return}\ \mathbb{E})\}\mathbf{return}\ E\{\lfloor p \rfloor_{\mathsf{a}} \wedge \mathsf{arem}(\mathbf{skip})\}}\ (\textsc{ret})$$

$$\frac{\begin{array}{c}\mathcal{D}, R, G, I \vdash \{p\}C_1\{r\} \\ \mathcal{D}, R, G, I \vdash \{r\}C_2\{q\}\end{array}}{\mathcal{D}, R, G, I \vdash \{p\}C_1; C_2\{q\}}\ (\textsc{seq}) \qquad \frac{\begin{array}{c}p \Rightarrow (B = B) * I \qquad \mathcal{D}, R, G, I \vdash \{p \wedge B\}C_1\{q\} \\ \mathcal{D}, R, G, I \vdash \{p \wedge \neg B\}C_2\{q\}\end{array}}{\mathcal{D}, R, G, I \vdash \{p\}\mathbf{if}\ (B)\ C_1\ \mathbf{else}\ C_2\{q\}}\ (\textsc{if})$$

$$\frac{\mathcal{D}, R, G, I \vdash \{p\}C\{q\}}{\mathcal{D}, R, G, I \vdash \{\lfloor p \rfloor_\Diamond\}C\{\lfloor q \rfloor_\Diamond\}}\ (\textsc{hide-}\Diamond) \qquad \frac{\mathcal{D}, R, G, I \vdash \{p\}C\{q\}}{\mathcal{D}, R, G, I \vdash \{p * r\}C\{q * r\}}\ (\textsc{frm})$$

$$\frac{\begin{array}{c}p' \overset{G}{\Rightarrow} p \qquad R' \Rightarrow R \qquad q \overset{G}{\Rightarrow} q' \qquad G \Rightarrow G' \\ \mathcal{D}, R, G, I \vdash \{p\}C\{q\} \qquad \mathsf{Enabled}(\mathcal{D}) \Rightarrow I \qquad \mathsf{wffAct}(R, \mathcal{D}) \\ p' \vee q' \Rightarrow I' * \mathsf{true} \qquad I' \triangleright \{G', R'\} \qquad \mathsf{Sta}(\{p', q'\}, R * \mathsf{Id})\end{array}}{\mathcal{D}, R', G', I' \vdash \{p'\}C\{q'\}}\ (\textsc{csq}) \qquad \frac{\begin{array}{c}\mathcal{D}, R, G, I \vdash \{p\}C\{q\} \\ x \notin fv(\mathcal{D}, R, G, I)\end{array}}{\mathcal{D}, R, G, I \vdash \{\exists x.\ p\}C\{\exists x.\ q\}}\ (\textsc{ex})$$

$$\frac{\begin{array}{c}\mathcal{D}, R, G, I \vdash \{p_1\}C\{q_1\} \\ \mathcal{D}, R, G, I \vdash \{p_2\}C\{q_2\}\end{array}}{\mathcal{D}, R, G, I \vdash \{p_1 \wedge p_2\}C\{q_1 \wedge q_2\}}\ (\textsc{conj}) \qquad \frac{\begin{array}{c}\mathcal{D}, R, G, I \vdash \{p_1\}C\{q_1\} \\ \mathcal{D}, R, G, I \vdash \{p_2\}C\{q_2\}\end{array}}{\mathcal{D}, R, G, I \vdash \{p_1 \vee p_2\}C\{q_1 \vee q_2\}}\ (\textsc{disj})$$

Fig. 20. LRG-style inference rules.

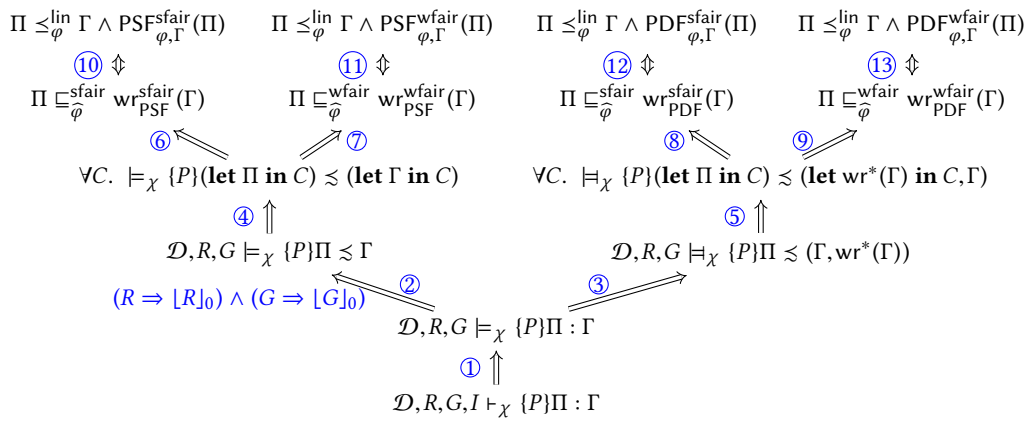

Fig. 21. Structures of logic soundness proofs.

The (csq) rule uses $p' \overset{G}{\Rrightarrow} p$ and $q \overset{G}{\Rrightarrow} q'$. The definition of $p \overset{G}{\Rrightarrow} q$ is similar to $p \Rrightarrow_k q$, as defined in Fig. 19. In addition to executing the abstract code and decreasing the corresponding ♦-tokens, $p \overset{G}{\Rrightarrow} q$ also requires the overall transition to satisfy G.

B.2 Overview of the structures of soundness proofs

In the following sections, we prove Theorem 7.3 about the logic soundness. We first give an overview of the proof structures. Notice that the two await rules actually give us two program logics, for strongly fair and weakly fair scheduling respectively. So we need two definitions of the semantics, $\mathcal{D}, R, G \models_{\mathsf{wfair}} \{P\}\Pi : \Gamma$ and $\mathcal{D}, R, G \models_{\mathsf{sfair}} \{P\}\Pi : \Gamma$, for the two judgments for object verification. On the other hand, our logics need to ensure *four* contextual refinements, each of which gives us one combination of PSF/PDF and strong/weak fairness (by Abstraction Theorem). Thus the soundness proofs go through four paths towards the final goals.

We show the proof structures in Fig. 21. Each proof path follows LiLi. The key in each path is a termination-preserving simulation, which extends the simulation in LiLi to support partial methods of objects, and **await** commands at both concrete and abstract levels. The logic ensures that the concrete implementation Π is simulated by the abstraction generated by the wrapper. Then we prove the simulation ensures the contextual refinement $\sqsubseteq$. In the diagram, we use "$\Longrightarrow$" for implications and "$\Longleftrightarrow$" for equivalences.

The top at Fig. 21 shows our final goals: verifying linearizability $\Pi \preceq^{\mathsf{lin}}_{P} \Gamma$, and the PSF/PDF properties $\mathsf{Prog}^{\chi}_{\varphi,\Gamma}(\Pi)$. By the Abstraction Theorem, we reduce the goals (see ⑩, ⑪, ⑫ and ⑬ in Fig. 21) to verifying contextual refinements $\sqsubseteq$. Next we propose four simulations $\Pi \precsim \Pi'$ as a proof technique for the contextual refinement $\sqsubseteq$.

At the bottom of Fig. 21, we define semantics for our logic judgment $\mathcal{D}, R, G \vdash_{\chi} \{P\}\Pi : \Gamma$ ($\chi \in \{\mathsf{sfair}, \mathsf{wfair}\}$). Note that the logic uses the atomic Γ as specification. The judgment semantics $\mathcal{D}, R, G \models_{\chi} \{P\}\Pi : \Gamma$ is also based on simulations, but it is much closer to the logic rules, which can simplify the task of proving the validity of each rule in Fig. 20. It can be directly (see ② and ③) translated to $\Pi \precsim \Pi'$, where Π' is either Γ (for PSF objects) or $\mathsf{wr}^*(\Gamma)$ (for PDF objects). The special wrapper wr^* is defined similarly as $\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}$.

All the formal definitions and detailed proofs are given in the following subsections.

- Section B.3 defines the judgments' semantics: $\mathcal{D}, R, G \models_{\chi} \{P\}\Pi : \Gamma$ and $\mathcal{D}, R, G, I \models_{\chi} \{p\}C\{q\}$.

- Section B.4 shows the proofs of ① of Fig. 21, i.e., the logic rules are sound with respect to the judgment semantics.
- Section B.5 defines the local simulations $\precsim$ and shows the proofs of ② and ③ of Fig. 21, i.e., the judgment semantics implies the simulations.
- Section B.6 and Section B.7 show the proofs of ④ and ⑤ of Fig. 21. Section B.6 lifts the local simulations $\precsim$ to thread simulations, and Section B.7 defines the whole-program simulations and proves the parallel compositionality.
- Section B.8 and Section B.9 show the proofs of ⑥, ⑦, ⑧ and ⑨ of Fig. 21, i.e., the simulations imply the contextual refinements.

B.3 Judgment semantics

The judgment semantics $\mathcal{D}, R, G \models_{\chi} \{P\}\Pi : \Gamma$ (here $\chi \in \{\mathsf{sfair}, \mathsf{wfair}\}$) is based on a simulation between Π and Γ, parameterized with well-founded metrics: M, ξ, $\mathbb{M}$ and u. The metric M is to ensure that the current thread t must fulfill its definite action $\mathcal{D}_\mathsf{t}$ in a finite number of steps. If thread t is blocked, the metric ξ specifies the set of the environment threads that t is waiting for. It shrinks when an environment thread t' finishes definite action $\mathcal{D}_{\mathsf{t}'}$. The metric $\mathbb{M}$ corresponds to the number of white tokens $\lozenge$, which is to ensure that thread t progresses on its own when ξ becomes empty. The last metric u corresponds to the tuple of black tokens. It bounds the number of actions made by thread t which could delay the progress of its environment threads.

Definition B.1. $\mathcal{D}, R, G \models_{\chi} \{P\}\Pi : \Gamma$ iff, for any $f \in dom(\Pi)$, for any σ and Σ, for any t, if $\Pi(f) = (\mathcal{P}, x, C)$, $\Gamma(f) = (\mathcal{P}', y, \mathbb{C})$ and $(\sigma, \Sigma) \models (P_\mathsf{t} \wedge \mathcal{P}_\mathsf{t}) * \mathsf{own}(x) * \mathsf{own}(y) \wedge (x = y)$, there exist four well-founded metrics u, $\mathbb{M}$, M and aw, a boolean flag wb and two sets $\xi, \xi_a \in \mathscr{P}(\mathit{ThrdID})$ such that $wb = \textbf{false}$ and

$$\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (C, \sigma) \preceq (\mathbb{C}, \Sigma) \diamond (u, \mathbb{M}, M, wb, aw) \Downarrow_{\xi, \xi_a} (P * \mathsf{own}(x) * \mathsf{own}(y)).$$

Here $\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (C, \sigma) \preceq (\mathbb{C}, \Sigma) \diamond (u, \mathbb{M}, M, wb, aw) \Downarrow_{\xi, \xi_a} Q$ is co-inductively defined as follows. Whenever $\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (C, \sigma) \preceq (\mathbb{C}, \Sigma) \diamond (u, \mathbb{M}, M, wb, aw) \Downarrow_{\xi, \xi_a} Q$ holds, then the following hold:

(1)(a) Suppose $\sigma = (s, h)$. Then $\xi \cup \xi_a \subseteq s(\mathtt{TIDS})$ and $\mathsf{t} \notin \xi$ and $\mathsf{t} \notin \xi_a$.
(b) For any $\mathsf{t}' \in \xi \cup \xi_a$, we have $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}'}) * \mathsf{true}$.
(c) If $wb = \textbf{false}$, then $\xi = \emptyset$. If $wb = \textbf{true}$, then $\xi \neq \emptyset \vee \Sigma \models \neg\mathsf{en}(\mathbb{C})$.
(d) If $\sigma \models \neg\mathsf{en}(C)$ and $\Sigma \models \mathsf{en}(\mathbb{C})$, then $\xi_a \neq \emptyset$.
(e) If $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_\mathsf{t}) * \mathsf{true}$, then $\sigma \models \mathsf{en}(C)$.
(2) If $C = \mathbf{E}[\,\textbf{return}\ E\,]$, then there exists $\mathbb{E}$ such that
(a) $\mathbb{C} = (\textbf{return}\ \mathbb{E})$, and
(b) $(\sigma, \Sigma) \models Q_\mathsf{t}$ and $[\![E]\!]_{\sigma.s} = [\![\mathbb{E}]\!]_{\Sigma.s}$, and
(c) $((\sigma, \Sigma), (\sigma, \Sigma), 0) \models G_\mathsf{t} * \mathsf{True}$, and
(d) $wb = \textbf{false}$.
(3) For any σ_F, $(C, \sigma \uplus \sigma_F) \not\longrightarrow_{\mathsf{t}} \textbf{abort}$.
(4) For any C', σ'', σ_F and Σ_F, if $(C, \sigma \uplus \sigma_F) \longrightarrow_{\mathsf{t}} (C', \sigma'')$ and $\Sigma \perp \Sigma_F$, then there exist σ', $\mathbb{C}'$, Σ', k, u', $\mathbb{M}'$, M', wb', aw', ξ' and ξ'_a such that
(a) $\sigma'' = \sigma' \uplus \sigma_F$, and
(b) $(\mathbb{C}, \Sigma \uplus \Sigma_F) \longrightarrow^{*}_{\mathsf{t}} (\mathbb{C}', \Sigma' \uplus \Sigma_F)$, and
(c) $\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (C', \sigma') \preceq (\mathbb{C}', \Sigma') \diamond (u', \mathbb{M}', M', wb', aw') \Downarrow_{\xi', \xi'_a} Q$, and
(d) $((\sigma, \Sigma), (\sigma', \Sigma'), k) \models G_\mathsf{t} * \mathsf{True}$, and
(e) either $u' <_k u$ and $k > 0$,
or $u' = u$ and $k = 0$ and $\mathbb{M}' < \mathbb{M}$,
or $u' = u$ and $k = 0$ and $\mathbb{M}' = \mathbb{M}$ and $wb' = wb = \textbf{true}$ and $\xi \subseteq \xi'$; and

(f) if $((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle[\mathcal{D}_\mathsf{t}]\rangle * \mathsf{True}$ and $k = 0$, then $M' < M$.

(5) For any k, σ' and Σ', if $((\sigma,\Sigma),(\sigma',\Sigma'),k) \models R_\mathsf{t} * \mathsf{Id}$, then there exist u', $\mathbb{M}'$, M', wb', aw', ξ_d, ξ_{ad}, ξ' and ξ'_a such that

(a) $\mathcal{D},R,G \models^{\chi}_{\mathsf{t}} (C,\sigma') \preceq (\mathbb{C},\Sigma') \diamond (u',\mathbb{M}',M',wb',aw') \Downarrow_{\xi',\xi'_a} Q$, and

(b) $u' \approx_k u$, and

(c) $\xi_d = \{\mathsf{t}' \mid (\mathsf{t}' \in \xi) \wedge (((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle\mathcal{D}_{\mathsf{t}'}\rangle * \mathsf{Id})\}$ and
$k = 0 \Longrightarrow \mathbb{M}' < \mathbb{M} \vee (\mathbb{M}' = \mathbb{M} \wedge wb' = wb)$ and
$k = 0 \wedge wb = \mathbf{true} \wedge (\xi_d \neq \emptyset \vee (\Sigma \models \neg\mathsf{en}(\mathbb{C}) \wedge \Sigma' \models \mathsf{en}(\mathbb{C}))) \Longrightarrow \mathbb{M}' < \mathbb{M}$ and
$k = 0 \wedge \mathbb{M}' = \mathbb{M} \wedge wb' = wb = \mathbf{true} \Longrightarrow \xi \backslash \xi_d \subseteq \xi'$, and

(d) if $k = 0$ and $(\sigma,\Sigma) \models \mathsf{Enabled}(\mathcal{D}_\mathsf{t}) * \mathsf{true}$, then $M' \leq M$; and

(e) $\xi_{ad} = \{\mathsf{t}' \mid (\mathsf{t}' \in \xi_a) \wedge (((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle\mathcal{D}_{\mathsf{t}'}\rangle * \mathsf{Id})\}$ and
$k = 0 \wedge \mathsf{is\_await}(C) \Longrightarrow \xi_a \backslash \xi_{ad} \subseteq \xi'_a$ and
$k = 0 \wedge \mathsf{is\_await}(C) \wedge (\xi_{ad} \neq \emptyset \vee (\Sigma \models \neg\mathsf{en}(\mathbb{C}) \wedge \Sigma' \models \mathsf{en}(\mathbb{C}))) \Longrightarrow aw' < aw$ and
$(\chi = \mathsf{sfair}) \wedge k = 0 \wedge \mathsf{is\_await}(C) \wedge (\sigma \models \neg\mathsf{en}(C)) \wedge (\sigma' \models \neg\mathsf{en}(C)) \Longrightarrow aw' \leq aw$ and
$(\chi = \mathsf{wfair}) \wedge k = 0 \wedge \mathsf{is\_await}(C) \Longrightarrow aw' \leq aw$.

Definition B.2. $\mathcal{D},R,G \models_\chi \{p\}C\{q\}$ iff, for any σ,Σ,u,w and $\mathbb{C}$, for any t, if $((\sigma,\Sigma),(u,w),\mathbb{C}) \models p_\mathsf{t}$, then there exist aw and ξ_a such that

$$\mathcal{D},R,G \models^{\chi}_{\mathsf{t}} (C,\sigma) \preceq (\mathbb{C},\Sigma) \diamond (u,((0,0),|C|),(0,|C|),aw,w,\mathsf{height}(C),\mathsf{height}(C)) \Downarrow_{\emptyset,\xi_a} q.$$

Here $\mathcal{D},R,G \models^{\chi}_{\mathsf{t}} (C,\sigma) \preceq (\mathbb{C},\Sigma) \diamond (u,\mathbb{ws},ws,aw,w,wk,\mathcal{H}) \Downarrow_{\xi,\xi_a} q$ is co-inductively defined as follows.

Whenever $\mathcal{D},R,G \models^{\chi}_{\mathsf{t}} (C,\sigma) \preceq (\mathbb{C},\Sigma) \diamond (u,\mathbb{ws},ws,aw,w,wk,\mathcal{H}) \Downarrow_{\xi,\xi_a} q$ holds, then the following hold:

(1)(a) Suppose $\sigma = (s,h)$. Then $\xi \cup \xi_a \subseteq s(\mathtt{TIDS})$ and $\mathsf{t} \notin \xi$ and $\mathsf{t} \notin \xi_a$.

(b) For any $\mathsf{t}' \in \xi \cup \xi_a$, we have $(\sigma,\Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}'}) * \mathsf{true}$.

(c) If $(\sigma,\Sigma) \models \mathsf{Enabled}(\mathcal{D}_\mathsf{t}) * \mathsf{true}$, then $\sigma \models \mathsf{en}(C)$.

(d) If $\sigma \models \neg\mathsf{en}(C)$ and $\Sigma \models \mathsf{en}(\mathbb{C})$, then $\xi_a \neq \emptyset$.

(e) If $wk = \mathcal{H}$, then $\xi = \emptyset$. If $wk < \mathcal{H}$, then $\xi \neq \emptyset \vee \Sigma \models \neg\mathsf{en}(\mathbb{C})$.

(f) $|\mathbb{ws}| \leq \mathcal{H}$ and $(1 \leq wk \leq |\mathbb{ws}| - 1) \vee (wk = \mathcal{H})$.

(2) If $C = \mathbf{skip}$, then for any Σ_F such that $\Sigma \perp \Sigma_F$, there exist $\mathbb{C}'$ and Σ' such that

(a) $(\mathbb{C},\Sigma \uplus \Sigma_F) \longrightarrow^*_\mathsf{t} (\mathbb{C}',\Sigma' \uplus \Sigma_F)$, and

(b) $((\sigma,\Sigma'),(u,w),\mathbb{C}') \models q_\mathsf{t}$, and

(c) $\mathbb{ws} = ((0,0),0)$ and $ws = (0,0)$ and $wk = \mathcal{H}$ and $\xi = \emptyset$, and

(d) $((\sigma,\Sigma),(\sigma,\Sigma'),0) \models G_\mathsf{t} * \mathsf{True}$.

(3) If $C = \mathbf{E}[\,\mathbf{return}\ E\,]$, then there exists $\mathbb{E}$ such that

(a) $\mathbb{C} = (\mathbf{return}\ \mathbb{E})$, and

(b) $((\sigma,\Sigma),(u,w),\mathbf{skip}) \models q_\mathsf{t}$ and $[\![E]\!]_{\sigma.s} = [\![\mathbb{E}]\!]_{\Sigma.s}$, and

(c) $((\sigma,\Sigma),(\sigma,\Sigma),0) \models G_\mathsf{t} * \mathsf{True}$, and

(d) $wk = \mathcal{H}$.

(4) For any σ_F, $(C,\sigma \uplus \sigma_F) \not\longrightarrow_\mathsf{t} \mathbf{abort}$.

(5) For any C', σ'', σ_F and Σ_F, if $(C,\sigma \uplus \sigma_F) \longrightarrow_\mathsf{t} (C',\sigma'')$, then there exist σ', $\mathbb{C}'$, Σ', k, u', $\mathbb{ws}'$, ws', aw', w', wk', ξ' and ξ'_a such that

(a) $\sigma'' = \sigma' \uplus \sigma_F$, and

(b) $(\mathbb{C},\Sigma \uplus \Sigma_F) \longrightarrow^*_\mathsf{t} (\mathbb{C}',\Sigma' \uplus \Sigma_F)$, and

(c) $\mathcal{D},R,G \models^{\chi}_{\mathsf{t}} (C',\sigma') \preceq (\mathbb{C}',\Sigma') \diamond (u',\mathbb{ws}',ws',aw',w',wk',\mathcal{H}) \Downarrow_{\xi',\xi'_a} q$, and

(d) $((\sigma,\Sigma),(\sigma',\Sigma'),k) \models G_\mathsf{t} * \mathsf{True}$, and

(e) either $u' <_k u$ and $k > 0$,
or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}' <^{wk}_{\mathcal{H}} \mathbb{ws}$,

or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}' \approx^{wk}_{\mathcal{H}} \mathbb{ws}$ and $wk' < wk$,
or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}' \approx^{wk}_{\mathcal{H}} \mathbb{ws}$ and $wk' = wk < \mathcal{H}$ and $\xi \subseteq \xi'$; and

(f) if $((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle[\mathcal{D}_t]\rangle * \mathsf{True}$ and $k = 0$, then $ws' <_{\mathcal{H}} ws$.

(6) For any k, σ' and Σ', if $((\sigma,\Sigma),(\sigma',\Sigma'),k) \models R_t * \mathsf{Id}$, then there exist u', $\mathbb{ws}'$, ws', aw', w', wk', ξ_d, ξ_{ad}, ξ' and ξ'_a such that

(a) $\mathcal{D},R,G \models^{\chi}_{t} (C,\sigma') \preceq (\mathbb{C},\Sigma') \diamond (u',\mathbb{ws}',ws',aw',w',wk',\mathcal{H}) \Downarrow_{\xi',\xi'_a} q$, and

(b) $u' \approx_k u$, and
$k = 0 \Longrightarrow w' = w$, and

(c) $\xi_d = \{t' \mid (t' \in \xi) \wedge (((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle\mathcal{D}_{t'}\rangle * \mathsf{Id})\}$ and
$k = 0 \Longrightarrow \mathbb{ws}' <^{wk}_{\mathcal{H}} \mathbb{ws} \vee \mathbb{ws}' \approx^{wk}_{\mathcal{H}} \mathbb{ws}$ and
$k = 0 \wedge wk < \mathcal{H} \wedge (\xi_d \neq \emptyset \vee (\Sigma \models \neg\mathsf{en}(\mathbb{C}) \wedge \Sigma' \models \mathsf{en}(\mathbb{C}))) \Longrightarrow \mathbb{ws}' <^{wk}_{\mathcal{H}} \mathbb{ws}$ and
$k = 0 \wedge (\xi \backslash \xi_d \neq \emptyset \vee \Sigma' \models \neg\mathsf{en}(\mathbb{C})) \Longrightarrow wk' \leq wk$, and
$k = 0 \wedge wk' = wk \Longrightarrow \xi \backslash \xi_d \subseteq \xi'$, and

(d) if $k = 0$ and $(\sigma,\Sigma) \models \mathsf{Enabled}(\mathcal{D}_t) * \mathsf{true}$, then $ws' \leq_{\mathcal{H}} ws$; and

(e) $\xi_{ad} = \{t' \mid (t' \in \xi_a) \wedge (((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle\mathcal{D}_{t'}\rangle * \mathsf{Id})\}$ and
$k = 0 \wedge \mathsf{is\_await}(C) \Longrightarrow \xi_a \backslash \xi_{ad} \subseteq \xi'_a$ and
$k = 0 \wedge \mathsf{is\_await}(C) \wedge (\xi_{ad} \neq \emptyset \vee (\Sigma \models \neg\mathsf{en}(\mathbb{C}) \wedge \Sigma' \models \mathsf{en}(\mathbb{C}))) \Longrightarrow aw' < aw$ and
$(\chi = \mathsf{sfair}) \wedge k = 0 \wedge \mathsf{is\_await}(C) \wedge (\sigma \models \neg\mathsf{en}(C)) \wedge (\sigma' \models \neg\mathsf{en}(C)) \Longrightarrow aw' \leq aw$ and
$(\chi = \mathsf{wfair}) \wedge k = 0 \wedge \mathsf{is\_await}(C) \Longrightarrow aw' \leq aw$.

Below we also define the semantics for the sequential judgment used in the ATOM rule. Note that C only accesses the concrete memory σ, therefore we require the other components in the full state (i.e., u, w, $\mathbb{C}$ and Σ) should remain unchanged during the execution of C.

Definition B.3 (SL judgment semantics, total correctness). $\models [p]C[q]$ iff, for all σ, Σ, u, w and $\mathbb{C}$, for any t, if $((\sigma,\Sigma),(u,w),\mathbb{C}) \models p_t$, the following are true:

(1) for any σ', if $(C,\sigma) \longrightarrow^{*}_{t} (\mathbf{skip},\sigma')$, then $((\sigma',\Sigma),(u,w),\mathbb{C}) \models q_t$;

(2) $(C,\sigma) \not\longrightarrow^{*}_{t} \mathbf{abort}$;

(3) $(C,\sigma) \not\longrightarrow^{\omega}_{t} \cdot$.

LEMMA B.4. *If* $\vdash [p]C[q]$, *then* $\models [p]C[q]$.

Definition B.5 (Locality).
$\mathsf{Locality}(C)$ iff, for any σ_1 and σ_2, let $\sigma = \sigma_1 \uplus \sigma_2$, then the following hold:

(1) (Safety monotonicity) If $(C,\sigma_1) \not\longrightarrow^{*}_{t} \mathbf{abort}$, then $(C,\sigma) \not\longrightarrow^{*}_{t} \mathbf{abort}$.

(2) (Termination monotonicity) If $(C,\sigma_1) \not\longrightarrow^{*}_{t} \mathbf{abort}$ and $(C,\sigma_1) \not\longrightarrow^{\omega}_{t} \cdot$, then $(C,\sigma) \not\longrightarrow^{\omega}_{t} \cdot$.

(3) (Frame property) For any n and σ', if $(C,\sigma_1) \not\longrightarrow^{*}_{t} \mathbf{abort}$ and $(C,\sigma) \longrightarrow^{n}_{t} (C',\sigma')$, then there exists σ'_1 such that $\sigma' = \sigma'_1 \uplus \sigma_2$ and $(C,\sigma_1) \longrightarrow^{n}_{t} (C',\sigma'_1)$.

B.3.1 Instantiating Metrics and Well-Founded Orders. The judgment semantics in Definition B.2 can be viewed as an instantiation of the simulation in Definition B.1. The key is to instantiate the metrics $\mathbb{M}$ and M and the boolean flag wb in $\mathcal{D},R,G \models^{\chi}_{t} (C,\sigma) \preceq (\mathbb{C},\Sigma) \diamond (u,\mathbb{M},M,wb,aw) \Downarrow_{\xi,\xi_a} Q$.

The flag wb is instantiated as the testing $wk \neq \mathcal{H}$. That is, $wb = \mathbf{false}$ if $wk = \mathcal{H}$; and $wb = \mathbf{true}$ if $wk \neq \mathcal{H}$.

The metric M is instantiated as follows.

$$\begin{array}{rrcl} (\mathit{Metric}) & M & ::= & (ws,\mathcal{H}) \\ (\mathit{WfStack}) & ws,\mathbb{ws} & ::= & (wn,n) \mid (wn,n) :: ws \\ (\mathit{StkHeight}) & \mathcal{H} & \in & \mathit{Nat} \end{array}$$

For each single thread, its metric ws is usually a list of (wn, n) pairs, where wn is the while-specific metric (which is left to be instantiated later) and n is a natural number specifying the "code size" as in Liang et al.'s work [Liang et al. 2014]. We let the threaded metric ws be a list (a stack actually) to allow different while-specific metrics for nested loops. That is, when entering a loop, we can push a (wn, n) pair to the ws stack; and when exiting the loop, we pop the pair out of ws.

The threaded metric ws follows the dictionary order. However, the usual dictionary order over lists is not well-founded (consider $B > AB > AAB > AAAB > \ldots$ in a dictionary). To address this issue, we introduce a bound of the list length (stack height), $\mathcal{H}$, and define the well-founded order $\prec_{\mathcal{H}}$ by requiring the length of the lists should be not larger than $\mathcal{H}$. Intuitively, the stack height $\mathcal{H}$ represents the maximal depth of nested loops, so it can be determined for any given program. The well-founded orders $M' < M$ and $ws' \prec_{\mathcal{H}} ws$ are defined as follows.

$$\frac{ws' \prec_{\mathcal{H}} ws \qquad \mathcal{H}' = \mathcal{H}}{(ws', \mathcal{H}') < (ws, \mathcal{H})}$$

$$ws' \prec_{\mathcal{H}} ws \quad \text{iff} \quad (ws' \prec ws) \wedge (|ws'| \le \mathcal{H}) \wedge (|ws| \le \mathcal{H})$$

$$ws' \preceq_{\mathcal{H}} ws \quad \text{iff} \quad (ws' \prec_{\mathcal{H}} ws) \vee (ws' = ws)$$

$$\frac{(wn', n') < (wn, n)}{(wn', n') \prec (wn, n)} \qquad \frac{(wn', n') < (wn, n)}{(wn', n') :: ws_1' \prec (wn, n) :: ws_1}$$

$$\frac{(wn', n') = (wn, n) \qquad ws_1' \prec ws_1}{(wn', n') :: ws_1' \prec (wn, n) :: ws_1}$$

$$\frac{(wn', n') < (wn, n)}{(wn', n') :: ws_1' \prec (wn, n)} \qquad \frac{(wn', n') \le (wn, n)}{(wn', n') \prec (wn, n) :: ws_1}$$

Here $|ws|$ is the length of ws, which is defined as follows:

$$\begin{aligned} |(wn, n)| &= 1 \\ |(wn, n) :: ws| &= 1 + |ws| \end{aligned}$$

The well-founded order over the (wn, n) pairs is a usual dictionary order below, where the order over wn is instantiated later depending on the type of wn.

$$(wn', n') < (wn, n) \quad \text{iff} \quad (wn' < wn) \vee (wn' = wn \wedge n' < n)$$

$$(wn', n') = (wn, n) \quad \text{iff} \quad (wn' = wn) \wedge (n' = n)$$

$$(wn', n') \le (wn, n) \quad \text{iff} \quad (wn', n') < (wn, n) \vee (wn', n') = (wn, n)$$

Lemma B.6 (Well-foundedness). *The relations $M' < M$ and $ws' \prec_{\mathcal{H}} ws$ defined above are both well-founded relations.*

The metric $\mathbb{M}$ is instantiated as follows.

$$(\textit{Metric'}) \quad \mathbb{M} \quad ::= \quad (\mathbb{ws}, wk, \mathcal{H})$$

$$\frac{\mathbb{ws}' \prec^{wk}_{\mathcal{H}} \mathbb{ws} \qquad \mathcal{H}' = \mathcal{H}}{(\mathbb{ws}', wk', \mathcal{H}') < (\mathbb{ws}, wk, \mathcal{H})} \qquad \frac{\mathbb{ws}' \approx^{wk}_{\mathcal{H}} \mathbb{ws} \qquad wk' < wk \qquad \mathcal{H}' = \mathcal{H}}{(\mathbb{ws}', wk', \mathcal{H}') < (\mathbb{ws}, wk, \mathcal{H})}$$

$$\frac{\mathbb{ws}' \approx^{wk}_{\mathcal{H}} \mathbb{ws} \qquad wk' = wk \qquad \mathcal{H}' = \mathcal{H}}{(\mathbb{ws}', wk', \mathcal{H}') = (\mathbb{ws}, wk, \mathcal{H})}$$

$$\mathbb{ws}' \prec^{wk}_{\mathcal{H}} \mathbb{ws} \quad \text{iff} \quad (\mathbb{ws}' \ll^{wk} \mathbb{ws}) \wedge (|\mathbb{ws}'| \le \mathcal{H}) \wedge (|\mathbb{ws}| \le \mathcal{H})$$

$$\mathbb{ws}' \approx^{wk}_{\mathcal{H}} \mathbb{ws} \quad \text{iff} \quad (\mathbb{ws}' \approx^{wk} \mathbb{ws}) \wedge (|\mathbb{ws}'| \le \mathcal{H}) \wedge (|\mathbb{ws}| \le \mathcal{H})$$

Here $\mathbb{ws}' \ll^{wk} \mathbb{ws}$ is defined as follows.

$$\frac{(wn',n') < (wn,n) \qquad wk \geq 1}{(wn',n') \ll^{wk} (wn,n)} \qquad \frac{wn' < wn}{(wn',n') \ll^{0} (wn,n)}$$

$$\frac{(wn',n') < (wn,n) \qquad wk \geq 1}{(wn',n') :: ws_1' \ll^{wk} (wn,n) :: ws_1} \qquad \frac{wn' < wn}{(wn',n') :: ws_1' \ll^{0} (wn,n) :: ws_1}$$

$$\frac{(wn',n') = (wn,n) \qquad ws_1' \ll^{wk} ws_1}{(wn',n') :: ws_1' \ll^{wk+1} (wn,n) :: ws_1}$$

$$\frac{(wn',n') < (wn,n) \qquad wk \geq 1}{(wn',n') :: ws_1' \ll^{wk} (wn,n)} \qquad \frac{(wn',n') = (wn,n) \qquad wk \geq 1}{(wn',n') \ll^{wk} (wn,n) :: ws_1}$$

$$\frac{wn' < wn}{(wn',n') :: ws_1' \ll^{0} (wn,n)} \qquad \frac{wn' < wn}{(wn',n') \ll^{0} (wn,n) :: ws_1}$$

And $\mathbb{ws}' \approx^{wk} \mathbb{ws}$ is defined as follows.

$$\frac{(wn',n') = (wn,n) \qquad wk \geq 1}{(wn',n') \approx^{wk} (wn,n)} \qquad \frac{wn' = wn}{(wn',n') \approx^{0} (wn,n)}$$

$$\frac{(wn',n') = (wn,n) \qquad ws_1' \approx^{wk} ws_1}{(wn',n') :: ws_1' \approx^{wk+1} (wn,n) :: ws_1} \qquad \frac{wn' = wn}{(wn',n') :: ws_1' \approx^{0} (wn,n) :: ws_1}$$

$$\frac{wn' = wn}{(wn',n') :: ws_1' \approx^{wk} (wn,n)}$$

Height $\mathcal{H}$. As we said, the stack height $\mathcal{H}$ represents the maximal depth of nested loops. For any given program C, we can determine the stack height using a function height defined below.

$$\begin{aligned}
\mathsf{height}(\textbf{skip}) &\stackrel{\text{def}}{=} 1 \\
\mathsf{height}(\textbf{return}\ E) &\stackrel{\text{def}}{=} 1 \\
\mathsf{height}(c) &\stackrel{\text{def}}{=} 1 \\
\mathsf{height}(\textbf{await}(B)\{C\}) &\stackrel{\text{def}}{=} 1 \\
\mathsf{height}(C_1; C_2) &\stackrel{\text{def}}{=} max\{\mathsf{height}(C_1), \mathsf{height}(C_2)\} \\
\mathsf{height}(\textbf{if}\ (B)\ C_1\ \textbf{else}\ C_2) &\stackrel{\text{def}}{=} max\{\mathsf{height}(C_1), \mathsf{height}(C_2)\} \\
\mathsf{height}(\textbf{while}\ (B)\{C\}) &\stackrel{\text{def}}{=} \mathsf{height}(C) + 1
\end{aligned}$$

Initial code size. In Definition B.2, the judgment semantics initially takes the static code size $|C|$ defined below as the second dimension of ws and $\mathbb{ws}$.

$$
\begin{aligned}
|\textbf{skip}| &\stackrel{\text{def}}{=} 0\\
|\textbf{return } E| &\stackrel{\text{def}}{=} 1\\
|c| &\stackrel{\text{def}}{=} 1\\
|\textbf{await}(B)\{C\}| &\stackrel{\text{def}}{=} 1\\
|C_1; C_2| &\stackrel{\text{def}}{=} |C_1| + |C_2| + 1\\
|\textbf{if } (B)\ C_1 \textbf{ else } C_2| &\stackrel{\text{def}}{=} max\{|C_1|, |C_2|\} + 1\\
|\textbf{while } (B)\{C\}| &\stackrel{\text{def}}{=} 1
\end{aligned}
$$

Example of ws. Below we use a simple example to show how we assign a proper *ws* to each state during an execution. In the code below, we assign different labels to different layers of a nested while loop. The initial code is `while(i > 0) i--;`. The loop is labeled with 1 and its body code is labeled with 2. After the loop is unfolded, we use the syntax while to be distinguished from the original `while` which has not been unfolded.

In the *ws* below, the first dimension specifies the number of iterations left to unfold, and the second dimension specifies the "code size" at each layer.

	C	σ	ws
1	`while`^{1}`(i > 0) i--`^{2};	`i = 2`	$(0,1)$
2 $\rightarrow$	`i--`^{2}; `while`^{1}`(i > 0) i--`^{2};	`i = 2`	$(0,0)::(1,2)$
3 $\rightarrow$	`skip`^{2}; `while`^{1}`(i > 0) i--`^{2};	`i = 1`	$(0,0)::(1,1)$
4 $\rightarrow$	`while`^{1}`(i > 0) i--`^{2};	`i = 1`	$(0,0)::(1,0)$
5 $\rightarrow$	`i--`^{2}; `while`^{1}`(i > 0) i--`^{2};	`i = 1`	$(0,0)::(0,2)$
6 $\rightarrow$	`skip`^{2}; `while`^{1}`(i > 0) i--`^{2};	`i = 0`	$(0,0)::(0,1)$
7 $\rightarrow$	`while`^{1}`(i > 0) i--`^{2};	`i = 0`	$(0,0)::(0,0)$
8 $\rightarrow$	`skip`^{1};	`i = 0`	$(0,0)$

Initially, *ws* is $(0,1)$: the first dimension is 0 because we have not started to unfold the loop, and the second dimension is 1 because the code size of the whole loop is 1. After one step of the loop, *ws* becomes $(0,0)::(1,2)$. Since we have unfolded the loop, the *ws* stack contains two pairs now. In the second pair, the first dimension is 1 because the loop needs only one more iteration to finish (i.e., we only need to unfold it one more time). Its second dimension is 2 because the size of the loop body code is 2. After the next step, this dimension decreases. At the step of line 5, we unfold the loop again. So the first dimension of the second pair decreases to 0, saying that we do not need to unfold the loop anymore. Finally, at the step of line 8, the loop finishes, thus we pop out the second pair of the *ws* stack.

B.4 Soundness of the inference rules

In this section, we prove Lemma B.7 by induction over the derivation.

Lemma B.7 (① in Fig. 21). *If* $\mathcal{D}, R, G, I \vdash_\chi \{P\}\Pi : \Gamma$, *then* $\mathcal{D}, R, G \models_\chi \{P\}\Pi : \Gamma$.

Proof. By induction over derivation. By Lemma B.8, we only need to prove the following:

> If for all $f \in dom(\Pi)$ such that $\Pi(f) = (\mathcal{P}, x, C)$ and $\Gamma(f) = (\mathcal{P}', y, \mathbb{C})$, we have
> $\mathcal{D}, R, G \models_\chi \{(P \wedge \mathcal{P}) * \mathsf{own}(x) * \mathsf{own}(y) \wedge (x = y) \wedge \mathsf{arem}(\mathbb{C}) \wedge \blacklozenge(E_k, \ldots, E_1)\}\ C\ \{P * \mathsf{own}(x) * \mathsf{own}(y) \wedge \mathsf{arem}(\textbf{skip})\}$,
> then $\mathcal{D}, R, G \models_\chi \{P\}\Pi : \Gamma$. (B.1)

By co-induction. We instantiate the metrics $\mathbb{M}$ and M and the boolean flag wb following the way described in Sec. B.3.1.

The difficult case is to prove the environment step 5(c). From $\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (C,\sigma) \preceq (\mathbb{C},\Sigma) \diamond (u, \mathbb{ws}, ws, aw, w, wk, \mathcal{H}) \Downarrow_{\xi,\xi_a} q$, 6(c), we know

$$\begin{array}{c} k = 0 \Longrightarrow \mathbb{ws}' <^{wk}_{\mathcal{H}} \mathbb{ws} \vee \mathbb{ws}' \approx^{wk}_{\mathcal{H}} \mathbb{ws} \text{ and} \\ k = 0 \wedge wk < \mathcal{H} \wedge (\xi_d \neq \emptyset \vee (\Sigma \models \neg\mathsf{en}(\mathbb{C}) \wedge \Sigma' \models \mathsf{en}(\mathbb{C}))) \Longrightarrow \mathbb{ws}' <^{wk}_{\mathcal{H}} \mathbb{ws} \text{ and} \\ k = 0 \wedge (\xi \backslash \xi_d \neq \emptyset \vee \Sigma' \models \neg\mathsf{en}(\mathbb{C})) \Longrightarrow wk' \leq wk, \text{ and} \\ k = 0 \wedge wk' = wk \Longrightarrow \xi \backslash \xi_d \subseteq \xi', \end{array}$$

Thus we know: if $k = 0$, then

$$\begin{array}{l} \mathbb{ws}' <^{wk}_{\mathcal{H}} \mathbb{ws} \vee \\ (\mathbb{ws}' \approx^{wk}_{\mathcal{H}} \mathbb{ws} \wedge wk = \mathcal{H} \wedge \xi = \emptyset \wedge \xi_d = \emptyset \wedge (wk' < wk \vee wk' = wk \wedge \xi \subseteq \xi')) \vee \\ (\mathbb{ws}' \approx^{wk}_{\mathcal{H}} \mathbb{ws} \wedge \xi_d = \emptyset \wedge (\Sigma \models \mathsf{en}(\mathbb{C}) \vee \Sigma' \models \neg\mathsf{en}(\mathbb{C})) \wedge \\ \qquad (wk' < wk \vee wk' = wk \wedge \xi \subseteq \xi' \vee wk' > wk \wedge \xi \backslash \xi_d = \emptyset \wedge \Sigma' \models \mathsf{en}(\mathbb{C}))) \end{array}$$

For the very last case, we have:

$$\begin{array}{l} \mathbb{ws}' \approx^{wk}_{\mathcal{H}} \mathbb{ws} \wedge \xi_d = \emptyset \wedge (\Sigma \models \mathsf{en}(\mathbb{C}) \vee \Sigma' \models \neg\mathsf{en}(\mathbb{C})) \wedge wk' > wk \wedge \xi \backslash \xi_d = \emptyset \wedge \Sigma' \models \mathsf{en}(\mathbb{C}) \\ \Longrightarrow wk' > wk \wedge \xi = \emptyset \wedge \Sigma \models \mathsf{en}(\mathbb{C}) \\ \Longrightarrow wk' > wk \wedge wk = \mathcal{H} \Longrightarrow \textbf{false} \end{array}$$

As a result, we know

$$\begin{array}{l} k = 0 \Longrightarrow \mathbb{M}' < \mathbb{M} \vee (\mathbb{M}' = \mathbb{M} \wedge wb' = wb) \\ k = 0 \wedge wb = \textbf{true} \wedge (\xi_d \neq \emptyset \vee (\Sigma \models \neg\mathsf{en}(\mathbb{C}) \wedge \Sigma' \models \mathsf{en}(\mathbb{C}))) \Longrightarrow \mathbb{M}' < \mathbb{M} \\ k = 0 \wedge \mathbb{M}' = \mathbb{M} \wedge wb' = wb = \textbf{true} \Longrightarrow \xi \backslash \xi_d \subseteq \xi' \end{array}$$

Thus we are done. □

Lemma B.8. *If*

(1) $\mathcal{D}, R, G, I \vdash_{\chi} \{p\}C\{q\}$;
(2) $\mathsf{Enabled}(\mathcal{D}) \Rightarrow I$;
(3) for any t *and* t' *such that* $\mathsf{t} \neq \mathsf{t}'$, *we have* $G_{\mathsf{t}} \Rightarrow R_{\mathsf{t}'}$;

then $\mathcal{D}, R, G \models_{\chi} \{p\}C\{q\}$.

In the following subsections, we prove Lemma B.8 by induction over the derivation.

B.4.1 The WHL Rule.

Lemma B.9 (WHL-Sound). *If*

(1) $p \wedge B \Rightarrow p'$; $p \wedge B \wedge (\mathsf{Enabled}(\mathcal{D}) \vee Q) * \mathsf{true} \Rightarrow p' * (\Diamond \wedge \mathsf{emp})$;
(2) $\mathcal{D}, R, G \models_{\chi} \{p'\}C\{p\}$;
(3) $p \Rightarrow (B = B) * I$; $J \Rightarrow I$; $\mathsf{Sta}(J, R \vee G)$; $Q \Rightarrow I$; $p \wedge B \Rightarrow J * \mathsf{true} \wedge \mathsf{arem}(\textbf{await}(\mathbb{B}')\{\mathbb{C}'\})$;
(4) $J \Rightarrow (R, G\colon \mathcal{D}' \xrightarrow{f} (Q, \mathbb{B}'))$; $\mathcal{D}' \preccurlyeq \mathcal{D}$; $\mathit{wffAct}(R, \mathcal{D}')$;
(5) $I \triangleright \{R, G\}$; $\mathsf{Sta}(p, R * \mathsf{Id})$; $\mathsf{Enabled}(\mathcal{D}) \Rightarrow I$;
(6) for any t *and* t' *such that* $\mathsf{t} \neq \mathsf{t}'$, *we have* $G_{\mathsf{t}} \Rightarrow R_{\mathsf{t}'}$;

then $\mathcal{D}, R, G \models_{\chi} \{p\}\textbf{while}\ (B)\{C\}\{p \wedge \neg B\}$.

Proof. Let $\mathcal{H} = \mathsf{height}(\textbf{while}\ (B)\{C\}) = \mathsf{height}(C) + 1$. We know $|\textbf{while}\ (B)\{C\}| = 1$. Let

$$\mathbb{ws} = ((0,0),1)\ ,\ ws = (0,1)\ ,\ aw = 0\ ,\ wk = \mathcal{H}\ ,\ \xi = \emptyset\ ,\ \xi_a = \emptyset.$$

Below we prove: for any t, for any σ, Σ, u, w and $\mathbb{C}$,
if $((\sigma,\Sigma),(u,w),\mathbb{C}) \models p_{\mathsf{t}}$, then

$$\mathcal{D},R,G \models^{\chi}_{\mathsf{t}} (\textbf{while}\ (B)\{C\},\sigma) \preceq (\mathbb{C},\Sigma) \diamond (u,ww,ws,aw,w,wk,\mathcal{H}) \Downarrow_{\xi,\xi_a} p \wedge \neg B\ .$$

By co-induction. Suppose $\sigma = (s,h)$. Since $p \Rightarrow (B = B) * I$, we know

$$(\sigma,\Sigma) \models I * \mathsf{true},\ \text{and either } [\![B]\!]_s = \textbf{true} \text{ or } [\![B]\!]_s = \textbf{false}\ .$$

We only need to prove the following (1)(2)(3)(4)(5)(6).

(1)(a) $\xi \cup \xi_a \subseteq s(\mathtt{TIDS})$ and $\mathsf{t} \notin \xi$ and $\mathsf{t} \notin \xi_a$.
(b) For any $\mathsf{t}' \in \xi \cup \xi_a$, we have $(\sigma,\Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}'}) * \mathsf{true}$.
(c) If $(\sigma,\Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}}) * \mathsf{true}$, then $\sigma \models \mathsf{en}(\textbf{while}\ (B)\{C\})$.
(d) If $\sigma \models \neg\mathsf{en}(\textbf{while}\ (B)\{C\})$ and $\Sigma \models \mathsf{en}(\mathbb{C})$, then $\xi_a \neq \emptyset$.
(e) If $wk = \mathcal{H}$, then $\xi = \emptyset$. If $wk < \mathcal{H}$, then $\xi \neq \emptyset \vee \Sigma \models \neg\mathsf{en}(\mathbb{C})$.
(f) $|ww| \leq \mathcal{H}$ and $(1 \leq wk \leq |ww| - 1) \vee (wk = \mathcal{H})$.
Proof: Trivial.

(2) If $\textbf{while}\ (B)\{C\} = \textbf{skip}$, then ....
Proof: It is vacantly true.

(3) If $\textbf{while}\ (B)\{C\} = \mathbf{E}[\ \textbf{return}\ E\ ]$, then ....
Proof: It is vacantly true.

(4) For any σ_F, $(\textbf{while}\ (B)\{C\},\sigma \uplus \sigma_F) \not\longrightarrow_{\mathsf{t}} \textbf{abort}$.
Proof: It holds because $[\![B]\!]_s$ is not undefined.

(5) If $(\textbf{while}\ (B)\{C\},\sigma \uplus \sigma_F) \longrightarrow_{\mathsf{t}} (C',\sigma'')$, then there exist σ', $\mathbb{C}'$, Σ', k, u', ww', ws', aw', w', wk', ξ' and ξ'_a such that
(a) $\sigma'' = \sigma' \uplus \sigma_F$, and
(b) $(\mathbb{C},\Sigma \uplus \Sigma_F) \longrightarrow^*_{\mathsf{t}} (\mathbb{C}',\Sigma' \uplus \Sigma_F)$, and
(c) $\mathcal{D},R,G \models^{\chi}_{\mathsf{t}} (C',\sigma') \preceq (\mathbb{C}',\Sigma') \diamond (u',ww',ws',aw',w',wk',\mathcal{H}) \Downarrow_{\xi',\xi'_a} p \wedge \neg B$, and
(d) $((\sigma,\Sigma),(\sigma',\Sigma'),k) \models G_{\mathsf{t}} * \mathsf{True}$, and
(e) either $u' <_k u$ and $k > 0$,
or $u' = u$ and $k = 0$ and $w' = w$ and $ww' <^{wk}_{\mathcal{H}} ww$,
or $u' = u$ and $k = 0$ and $w' = w$ and $ww' \approx^{wk}_{\mathcal{H}} ww$ and $wk' < wk$,
or $u' = u$ and $k = 0$ and $w' = w$ and $ww' \approx^{wk}_{\mathcal{H}} ww$ and $wk' = wk < \mathcal{H}$ and $\xi \subseteq \xi'$; and
(f) if $((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle[\mathcal{D}_{\mathsf{t}}]\rangle * \mathsf{True}$ and $k = 0$, then $ws' <_{\mathcal{H}} ws$.
Proof: We have two cases depending on whether $[\![B]\!]_s$ is **true** or **false**.

(I) If $[\![B]\!]_s = \textbf{true}$, we know $\sigma'' = \sigma \uplus \sigma_F$ and $(\textbf{while}\ (B)\{C\},\sigma \uplus \sigma_F) \longrightarrow_{\mathsf{t}} (C;\mathsf{while}\ (B)\{C\},\sigma \uplus \sigma_F)$.
Also we know $((\sigma,\Sigma),(u,w),\mathbb{C}) \models p_{\mathsf{t}} \wedge B$. From $p \wedge B \Rightarrow p'$ and $\mathcal{D},R,G \models_{\chi} \{p'\}C\{p\}$, let $ww_1 = ((0,0),|C|)$ and $ws_1 = (0,|C|)$ and $wk_1 = \mathsf{height}(C)$, we get: there exist aw' and ξ'_a such that

$$\mathcal{D},R,G \models^{\chi}_{\mathsf{t}} (C,\sigma) \preceq (\mathbb{C},\Sigma) \diamond (u,ww_1,ws_1,aw',w,wk_1,\mathsf{height}(C)) \Downarrow_{\emptyset,\xi'_a} p\ .$$

Also since $p \wedge B \Rightarrow J * \mathsf{true} \wedge \mathsf{arem}(\textbf{await}(\mathbb{B}')\{\mathbb{C}'\})$, we know

$$(\sigma,\Sigma) \models J * \mathsf{true},\quad \mathbb{C} = \mathsf{arem}(\textbf{await}(\mathbb{B}')\{\mathbb{C}'\})\ .$$

Let

$$ws' = (0,0) :: (w,|C|+1)\ ,$$

we know $ws' <_{\mathcal{H}} ws$. Let

$$ww' = ((0,0),0) :: ((k_s,w),|C|+1)\ .$$

We know $ww' <^{wk}_{\mathcal{H}} ww$. Let

$$k_s = f_{\mathsf{t}}(\sigma,\Sigma)\ \text{ and }\ \xi_0 = \{\mathsf{t}'' \mid (\mathsf{t}'' \neq \mathsf{t}) \wedge ((\sigma,\Sigma) \models \mathsf{Enabled}(\mathcal{D}'_{\mathsf{t}''}) * \mathsf{true})\}\ .$$

Since $J \Rightarrow (R, G\colon \mathcal{D}' \xrightarrow{f} (Q, \mathbb{B}'))$, we have two cases:

- $\xi_0 \neq \emptyset \vee \Sigma \models \neg\mathsf{en}(\mathbb{C})$.
  By Lemma B.10, let
  $$wk' = 1 \text{ and } \xi' = \xi_0 .$$
  we know
  $$\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (C; \mathsf{while}\ (B)\{C\}, \sigma) \preceq (\mathbb{C}, \Sigma) \diamond (u, \mathbb{ws}', ws', aw', w, wk', \mathcal{H}) \Downarrow_{\xi', \xi'_a} p \wedge \neg B .$$
- $\xi_0 = \emptyset \wedge \Sigma \models \mathsf{en}(\mathbb{C})$.
  By Lemma B.10, let
  $$wk' = wk_1 + 1 \text{ and } \xi' = \emptyset .$$
  we know
  $$\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (C; \mathsf{while}\ (B)\{C\}, \sigma) \preceq (\mathbb{C}, \Sigma) \diamond (u, \mathbb{ws}', ws', aw', w, wk', \mathcal{H}) \Downarrow_{\xi', \xi'_a} p \wedge \neg B .$$

(II) If $[\![B]\!]_s = \mathbf{false}$, we know $(\mathbf{while}\ (B)\{C\}, \sigma \uplus \sigma_F) \longrightarrow_{\mathsf{t}} (\mathbf{skip}, \sigma \uplus \sigma_F)$.
By (SKIP) rule, let
$$\mathbb{ws}' = ((0,0),0) \text{ and } ws' = (0,0) ,$$
we know
$$\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (\mathbf{skip}, \sigma) \preceq (\mathbb{C}, \Sigma) \diamond (u, \mathbb{ws}', ws', aw, w, wk, \mathcal{H}) \Downarrow_{\emptyset,\emptyset} p \wedge \neg B .$$
Also we know $\mathbb{ws}' <^{wk}_{\mathcal{H}} \mathbb{ws}$ and $ws' <_{\mathcal{H}} ws$.

Since $(\sigma, \Sigma) \models I * \mathsf{true}$, we know
$$((\sigma, \Sigma), (\sigma, \Sigma), 0) \models [I] * \mathsf{True} .$$
Since $I \triangleright G$, we know
$$((\sigma, \Sigma), (\sigma, \Sigma), 0) \models G_{\mathsf{t}} * \mathsf{True} .$$

(6) If $((\sigma, \Sigma), (\sigma', \Sigma'), k) \models R_{\mathsf{t}} * \mathsf{Id}$, then there exist u', $\mathbb{ws}'$, ws', aw', w', wk', ξ_d, ξ_{ad}, ξ' and ξ'_a such that

(a) $\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (\mathbf{while}\ (B)\{C\}, \sigma') \preceq (\mathbb{C}, \Sigma') \diamond (u', \mathbb{ws}', ws', aw', w', wk', \mathcal{H}) \Downarrow_{\xi', \xi'_a} p \wedge \neg B$, and

(b) $u' \approx_k u$, and
$k = 0 \Longrightarrow w' = w$, and

(c) $\xi_d = \{\mathsf{t}' \mid (\mathsf{t}' \in \xi) \wedge (((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle \mathcal{D}_{\mathsf{t}'} \rangle * \mathsf{Id})\}$ and
$k = 0 \Longrightarrow \mathbb{ws}' <^{wk}_{\mathcal{H}} \mathbb{ws} \vee \mathbb{ws}' \approx^{wk}_{\mathcal{H}} \mathbb{ws}$ and
$k = 0 \wedge wk < \mathcal{H} \wedge (\xi_d \neq \emptyset \vee (\Sigma \models \neg\mathsf{en}(\mathbb{C}) \wedge \Sigma' \models \mathsf{en}(\mathbb{C}))) \Longrightarrow \mathbb{ws}' <^{wk}_{\mathcal{H}} \mathbb{ws}$ and
$k = 0 \wedge (\xi \backslash \xi_d \neq \emptyset \vee \Sigma' \models \neg\mathsf{en}(\mathbb{C})) \Longrightarrow wk' \leq wk$, and
$k = 0 \wedge wk' = wk \Longrightarrow \xi \backslash \xi_d \subseteq \xi'$, and

(d) if $k = 0$ and $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}}) * \mathsf{true}$, then $ws' \leq_{\mathcal{H}} ws$.

Proof: Since $\mathsf{Sta}(p, R * \mathsf{Id})$, we know there exist u' and w' such that
$$((\sigma', \Sigma'), (u', w'), \mathbb{C}) \models p_{\mathsf{t}} \text{ and } u' \approx_k u \text{ and } k = 0 \Longrightarrow w' = w .$$
By the co-induction hypothesis, we get
$$\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (\mathbf{while}\ (B)\{C\}, \sigma') \preceq (\mathbb{C}, \Sigma') \diamond (u', \mathbb{ws}, ws, aw, w', wk, \mathcal{H}) \Downarrow_{\xi, \xi_a} p \wedge \neg B .$$
We know $\xi_d = \emptyset$.

Thus we are done. □

We define:

$$\mathsf{head}(\mathbb{ws}) \stackrel{\text{def}}{=} \begin{cases} ((n_1, n_2), n_3) & \text{if } \mathbb{ws} = ((n_1, n_2), n_3) \\ ((n_1, n_2), n_3) & \text{if } \mathbb{ws} = ((n_1, n_2), n_3) :: \mathbb{ws}' \end{cases}$$

$$\mathsf{inchead}(\mathbb{ws}, ((k_1, k_2), k_3)) \stackrel{\text{def}}{=} \begin{cases} ((n_1 + k_1, n_2 + k_2), n_3 + k_3) & \text{if } \mathbb{ws} = ((n_1, n_2), n_3) \\ ((n_1 + k_1, n_2 + k_2), n_3 + k_3) :: \mathbb{ws}' & \text{if } \mathbb{ws} = ((n_1, n_2), n_3) :: \mathbb{ws}' \end{cases}$$

LEMMA B.10. *If*

(1) $p \wedge B \Rightarrow p'$*;* $p \wedge B \wedge (\mathsf{Enabled}(\mathcal{D}) \vee Q) * \mathsf{true} \Rightarrow p' * (\Diamond \wedge \mathsf{emp})$*;*
(2) $\mathcal{D}, R, G \models_{\chi} \{p'\} C \{p\}$*;*
(3) $p \Rightarrow (B = B) * I$*;* $J \Rightarrow I$*;* $\mathsf{Sta}(J, R \vee G)$*;* $Q \Rightarrow I$*;* $p \wedge B \Rightarrow J * \mathsf{true} \wedge \mathsf{arem}(\mathbf{await}(\mathbb{B}')\{\mathbb{C}'\})$*;*
(4) $J \Rightarrow (R, G\colon \mathcal{D}' \xrightarrow{f} (Q, \mathbb{B}'))$*;* $\mathcal{D}' \Subset \mathcal{D}$*;* $\mathit{wffAct}(R, \mathcal{D}')$*;*
(5) $I \triangleright \{R, G\}$*;* $\mathsf{Sta}(p, R * \mathsf{Id})$*;* $\mathsf{Enabled}(\mathcal{D}) \Rightarrow I$*;*
(6) for any t *and* t′ *such that* $\mathrm{t} \neq \mathrm{t}'$*, we have* $G_{\mathrm{t}} \Rightarrow R_{\mathrm{t}'}$*;*
(7) $\mathcal{D}, R, G \models^{\chi}_{\mathrm{t}} (C_1, \sigma) \preceq (\mathbb{C}, \Sigma) \diamond (u, \mathbb{ws}_1, ws_1, aw, w_1, wk_1, \mathcal{H}) \Downarrow_{\xi_1, \xi_a} p$*;*
(8) $(\sigma, \Sigma) \models J * \mathsf{true}$*;* $\mathsf{height}(C) = \mathcal{H}$*;* $\mathbb{C} = \mathbf{await}(\mathbb{B}')\{\mathbb{C}'\}$*;* $f_{\mathrm{t}}(\sigma, \Sigma) = k_s$*;* $w_1 \leq w$*;*
(9) $\xi_0 = \{\mathrm{t}' \mid (\mathrm{t}' \neq \mathrm{t}) \wedge ((\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}'_{\mathrm{t}'}) * \mathsf{true})\}$*;*
(10) $ws = (0,0) :: \mathsf{inchead}(ws_1, (w_1, 1))$*;*
(11) $\mathbb{ws} = ((0,0),0) :: \mathsf{inchead}(\mathbb{ws}_1, ((k_s, w_1), 1))$*;*
(12) one of the following holds:
- $\xi_0 \neq \emptyset \vee \Sigma \models \neg\mathsf{en}(\mathbb{C})$*:* $wk = 1 \wedge \xi = \xi_0$*;*
- $\xi_0 = \emptyset \wedge \Sigma \models \mathsf{en}(\mathbb{C})$*:* $wk = wk_1 + 1 \wedge \xi = \xi_1$*;*

then $\mathcal{D}, R, G \models^{\chi}_{\mathrm{t}} (C_1; \mathsf{while}\ (B)\{C\}, \sigma) \preceq (\mathbb{C}, \Sigma) \diamond (u, \mathbb{ws}, ws, aw, w, wk, \mathcal{H} + 1) \Downarrow_{\xi, \xi_a} p \wedge \neg B$.

Proof. By co-induction. We only need to prove the following (1)(2)(3)(4)(5).

(1)(a) Suppose $\sigma = (s, h)$. Then $\xi \cup \xi_a \subseteq s(\mathtt{TIDS})$ and $\mathrm{t} \notin \xi$ and $\mathrm{t} \notin \xi_a$.
(b) For any $\mathrm{t}' \in \xi \cup \xi_a$, we have $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathrm{t}'}) * \mathsf{true}$.
(c) If $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathrm{t}}) * \mathsf{true}$, then $\sigma \models \mathsf{en}(C_1; \mathsf{while}\ (B)\{C\})$.
(d) If $\sigma \models \neg\mathsf{en}(C_1; \mathsf{while}\ (B)\{C\})$ and $\Sigma \models \mathsf{en}(\mathbb{C})$, then $\xi_a \neq \emptyset$.
(e) If $wk = \mathcal{H} + 1$, then $\xi = \emptyset$. If $wk < \mathcal{H} + 1$, then $\xi \neq \emptyset \vee \Sigma \models \neg\mathsf{en}(\mathbb{C})$.
(f) $|\mathbb{ws}| \leq \mathcal{H} + 1$ and $(1 \leq wk \leq |\mathbb{ws}| - 1) \vee (wk = \mathcal{H} + 1)$.
Proof: From

$$\mathcal{D}, R, G \models^{\chi}_{\mathrm{t}} (C_1, \sigma) \preceq (\mathbb{C}, \Sigma) \diamond (u, \mathbb{ws}_1, ws_1, aw, w_1, wk_1, \mathcal{H}) \Downarrow_{\xi_1, \xi_a} p\ ,$$

we know for any $\mathrm{t}' \in \xi_1 \cup \xi_a$, we have $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathrm{t}'}) * \mathsf{true}$. Since $\mathcal{D}' \Subset \mathcal{D}$, we know for any $\mathrm{t}' \in \xi_0$, we have $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathrm{t}'}) * \mathsf{true}$. Thus we are done.

(2) If $(C_1; \mathsf{while}\ (B)\{C\}) = \mathbf{skip}$, then ...
Proof: It is vacantly true.

(3) For any σ_F, $(C_1; \mathsf{while}\ (B)\{C\}, \sigma \uplus \sigma_F) \not\longrightarrow_{\mathrm{t}} \mathbf{abort}$.
Proof: From

$$\mathcal{D}, R, G \models^{\chi}_{\mathrm{t}} (C_1, \sigma) \preceq (\mathbb{C}, \Sigma) \diamond (u, \mathbb{ws}_1, ws_1, aw, w_1, wk_1, \mathcal{H}) \Downarrow_{\xi_1, \xi_a} p\ ,$$

we know $(C_1, \sigma \uplus \sigma_F) \not\longrightarrow_{\mathrm{t}} \mathbf{abort}$. By the operational semantics, we are done.

(4) If $(C_1; \mathsf{while}\ (B)\{C\}, \sigma \uplus \sigma_F) \longrightarrow_{\mathrm{t}} (C', \sigma'')$, then there exist σ', $\mathbb{C}''$, Σ', k, u', $\mathbb{ws}'$, ws', aw', w', wk', ξ' and ξ'_a such that
(a) $\sigma'' = \sigma' \uplus \sigma_F$, and
(b) $(\mathbb{C}, \Sigma \uplus \Sigma_F) \longrightarrow^*_{\mathrm{t}} (\mathbb{C}'', \Sigma' \uplus \Sigma_F)$, and
(c) $\mathcal{D}, R, G \models^{\chi}_{\mathrm{t}} (C', \sigma') \preceq (\mathbb{C}'', \Sigma') \diamond (u', \mathbb{ws}', ws', aw', w', wk', \mathcal{H}) \Downarrow_{\xi', \xi'_a} p \wedge \neg B$, and
(d) $((\sigma, \Sigma), (\sigma', \Sigma'), k) \models G_{\mathrm{t}} * \mathrm{True}$, and
(e) either $u' <_k u$ and $k > 0$,
or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}' <^{wk}_{\mathcal{H}+1} \mathbb{ws}$,
or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}' \approx^{wk}_{\mathcal{H}+1} \mathbb{ws}$ and $wk' < wk$,
or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}' \approx^{wk}_{\mathcal{H}+1} \mathbb{ws}$ and $wk' = wk < \mathcal{H} + 1$ and $\xi \subseteq \xi'$; and
(f) if $((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle [\mathcal{D}_{\mathrm{t}}] \rangle * \mathrm{True}$ and $k = 0$, then $ws' <_{\mathcal{H}+1} ws$.
Proof: We have two cases depending on whether C_1 is **skip** or not.

(I) If $(C_1; \text{while } (B)\{C\}, \sigma \uplus \sigma_F) \longrightarrow_{\mathsf{t}} (C_1'; \text{while } (B)\{C\}, \sigma'')$, then $(C_1, \sigma \uplus \sigma_F) \longrightarrow_{\mathsf{t}} (C_1', \sigma'')$.
From $\mathcal{D}, R, G \models_{\mathsf{t}}^{\chi} (C_1, \sigma) \preceq (\mathbb{C}, \Sigma) \diamond (u, \mathbb{ws}_1, ws_1, aw, w_1, wk_1, \mathcal{H}) \Downarrow_{\xi_1, \xi_a} p$, we know there exist σ', $\mathbb{C}''$, Σ', k, u', $\mathbb{ws}_1'$, ws_1', aw', w_1', wk_1', ξ_1' and ξ_a' such that

(A) $\sigma'' = \sigma' \uplus \sigma_F$, and

(B) $(\mathbb{C}, \Sigma \uplus \Sigma_F) \longrightarrow_{\mathsf{t}}^{*} (\mathbb{C}'', \Sigma' \uplus \Sigma_F)$, and

(C) $\mathcal{D}, R, G \models_{\mathsf{t}}^{\chi} (C_1', \sigma') \preceq (\mathbb{C}'', \Sigma') \diamond (u', \mathbb{ws}_1', ws_1', aw', w_1', wk_1', \mathcal{H}) \Downarrow_{\xi_1', \xi_a'} p$, and

(D) $((\sigma, \Sigma), (\sigma', \Sigma'), k) \models G_{\mathsf{t}} * \mathsf{True}$, and

(E) either $u' <_k u$ and $k > 0$,
or $u' = u$ and $k = 0$ and $w_1' = w_1$ and $\mathbb{ws}_1' <_{\mathcal{H}}^{wk_1} \mathbb{ws}_1$,
or $u' = u$ and $k = 0$ and $w_1' = w_1$ and $\mathbb{ws}_1' \approx_{\mathcal{H}}^{wk_1} \mathbb{ws}_1$ and $wk_1' < wk_1$,
or $u' = u$ and $k = 0$ and $w_1' = w_1$ and $\mathbb{ws}_1' \approx_{\mathcal{H}}^{wk_1} \mathbb{ws}_1$ and $wk_1' = wk_1 < \mathcal{H}$ and $\xi_1 \subseteq \xi_1'$; and

(F) if $((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle [\mathcal{D}_{\mathsf{t}}] \rangle * \mathsf{True}$ and $k = 0$, then $ws_1' <_{\mathcal{H}} ws_1$.

Since $((\sigma, \Sigma), (\sigma', \Sigma'), k) \models G_{\mathsf{t}} * \mathsf{True}$, $(\sigma, \Sigma) \models J * \mathsf{true}$, $J \Rightarrow I$, $I \triangleright G$ and $\mathsf{Sta}(J, G \vee R)$, we know

$$(\sigma', \Sigma') \models J * \mathsf{true} .$$

Suppose $k_s' = f_{\mathsf{t}}(\sigma', \Sigma')$. Since $J \Rightarrow (R, G\colon \mathcal{D}' \xrightarrow{f} (Q, \mathbb{B}'))$, we can prove

$$k = 0 \implies k_s' \le k_s .$$

Let

$$\xi_0' = \{\mathsf{t}' \mid (\mathsf{t}' \neq \mathsf{t}) \wedge ((\sigma', \Sigma') \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}'}') * \mathsf{true})\} .$$

Since for any t' such that $\mathsf{t}' \neq \mathsf{t}$ we have $G_{\mathsf{t}} \Rightarrow R_{\mathsf{t}'}$, and since $\mathsf{wffAct}(R, \mathcal{D}')$, $\mathcal{D}' \Subset \mathcal{D}$ and $\mathsf{Enabled}(\mathcal{D}) \Rightarrow I$, we can prove:

$$k = 0 \implies \xi_0 \subseteq \xi_0' .$$

Let

$$ws' = (0,0) :: \mathsf{inchead}(ws_1', (w_1', 1)) \ , \ \mathbb{ws}' = ((0,0), 0) :: \mathsf{inchead}(\mathbb{ws}_1', ((k_s', w_1'), 1)) .$$

If $w_1' = w_1$, let $w' = w$; otherwise let $w' = w_1'$. Thus we know $w_1' \le w'$.
Also we know: if $k = 0$, then $w' = w$, and if $ws_1' <_{\mathcal{H}} ws_1$ then $ws' <_{\mathcal{H}+1} ws$.
If $\xi_0' \neq \emptyset \vee \Sigma' \models \neg\mathsf{en}(\mathbb{C}'')$, let $wk' = 1$ and $\xi' = \xi_0'$; otherwise let $wk' = wk_1' + 1$ and $\xi' = \xi_1'$.
Since $\mathbb{C} = \textbf{await}(\mathbb{B}')\{\mathbb{C}'\}$, we know $\mathbb{C}'' = \textbf{await}(\mathbb{B}')\{\mathbb{C}'\}$ or $\mathbb{C}'' = \textbf{skip}$. Then, by the co-induction hypothesis or by Lemma B.11, we know

$$\mathcal{D}, R, G \models_{\mathsf{t}}^{\chi} (C_1'; \text{while } (B)\{C\}, \sigma') \preceq (\mathbb{C}'', \Sigma') \diamond (u', \mathbb{ws}', ws', aw', w', wk', \mathcal{H} + 1) \Downarrow_{\xi', \xi_a'} p \wedge \neg B .$$

One of the following holds:

- If $\xi_0 \neq \emptyset \vee \Sigma \models \neg\mathsf{en}(\mathbb{C})$, then $wk = 1$ and $\xi = \xi_0$.
  If $k = 0$, we know $\mathbb{ws}_1' <_{\mathcal{H}}^{wk_1} \mathbb{ws}_1$ or $\mathbb{ws}_1' \approx_{\mathcal{H}}^{wk_1} \mathbb{ws}_1$. Since $wk_1 \ge 1$, we know
  $$\mathbb{ws}' <_{\mathcal{H}+1}^{wk} \mathbb{ws} \quad \text{or} \quad \mathbb{ws}' \approx_{\mathcal{H}+1}^{wk} \mathbb{ws}.$$
  If $k = 0$,
  - Suppose $\xi_0 \neq \emptyset$. Since $\xi_0 \subseteq \xi_0'$, we know $\xi_0' \neq \emptyset$. Thus $wk' = 1$ and $\xi' = \xi_0'$.
  - Suppose $\Sigma \models \neg\mathsf{en}(\mathbb{C})$. Since $(\mathbb{C}, \Sigma \uplus \Sigma_F) \longrightarrow_{\mathsf{t}}^{*} (\mathbb{C}'', \Sigma' \uplus \Sigma_F)$, we know $\Sigma' \models \neg\mathsf{en}(\mathbb{C}'')$. Thus $wk' = 1$ and $\xi' = \xi_0'$.
- If $\xi_0 = \emptyset \wedge \Sigma \models \mathsf{en}(\mathbb{C})$, then $wk = wk_1 + 1$ and $\xi = \xi_1$.
  If $k = 0$, we know $\mathbb{ws}_1' <_{\mathcal{H}}^{wk_1} \mathbb{ws}_1$ or $\mathbb{ws}_1' \approx_{\mathcal{H}}^{wk_1} \mathbb{ws}_1$. Thus we know
  $$\mathbb{ws}' <_{\mathcal{H}+1}^{wk} \mathbb{ws} \quad \text{or} \quad \mathbb{ws}' \approx_{\mathcal{H}+1}^{wk} \mathbb{ws}.$$
  Since $\Sigma \models \mathsf{en}(\mathbb{C})$, $\mathbb{C} = \textbf{await}(\mathbb{B}')\{\mathbb{C}'\}$ and $(\mathbb{C}, \Sigma \uplus \Sigma_F) \longrightarrow_{\mathsf{t}}^{*} (\mathbb{C}'', \Sigma' \uplus \Sigma_F)$, we know $\Sigma' \models \mathsf{en}(\mathbb{C}'')$. If $k = 0$,
  - Suppose $\xi_0' \neq \emptyset$. Thus $wk' = 1$ and $\xi' = \xi_0'$. Thus $wk' < wk$.
  - Suppose $\xi_0' = \emptyset$. Thus $wk' = wk_1' + 1$ and $\xi' = \xi_1'$.
    - If $wk_1' < wk_1$, then $wk' < wk$.

- If $wk'_1 = wk_1 < \mathcal{H}$ and $\xi_1 \subseteq \xi'_1$, then $wk' = wk < \mathcal{H} + 1$ and $\xi \subseteq \xi'$.

(II) If $C_1 = \textbf{skip}$ and $(C_1; \mathsf{while}\ (B)\{C\}, \sigma \uplus \sigma_F) \longrightarrow\!\triangleright_{\mathsf{t}} (\mathsf{while}\ (B)\{C\}, \sigma \uplus \sigma_F)$, from $\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (C_1, \sigma) \preceq (\mathbb{C}, \Sigma) \diamond (u, \mathbb{ws}_1, ws_1, aw, w_1, wk_1, \mathcal{H}) \Downarrow_{\xi_1, \xi_a} p$, we know there exist $\mathbb{C}''$ and Σ' such that

(A) $(\mathbb{C}, \Sigma \uplus \Sigma_F) \longrightarrow\!\triangleright^*_{\mathsf{t}} (\mathbb{C}'', \Sigma' \uplus \Sigma_F)$, and

(B) $((\sigma, \Sigma'), (u, w_1), \mathbb{C}'') \models p_{\mathsf{t}}$, and

(C) $\mathbb{ws}_1 = ((0,0),0)$ and $ws_1 = (0,0)$ and $wk_1 = \mathcal{H}$ and $\xi_1 = \emptyset$, and

(D) $((\sigma, \Sigma), (\sigma, \Sigma'), 0) \models G_{\mathsf{t}} * \mathsf{True}$.

Since $(\sigma, \Sigma) \models J * \mathsf{true}$, $J \Rightarrow I$, $I \triangleright G$ and $\mathsf{Sta}(J, G \vee R)$, we know

$$(\sigma, \Sigma') \models J * \mathsf{true}\ .$$

Suppose $k'_s = f_{\mathsf{t}}(\sigma, \Sigma')$. Since $J \Rightarrow (R, G\colon \mathcal{D}' \xrightarrow{f} Q)$, we can prove

$$k = 0 \Longrightarrow k'_s \leq k_s\ .$$

Let

$$\xi'_0 = \{\mathsf{t}' \mid (\mathsf{t}' \neq \mathsf{t}) \wedge ((\sigma, \Sigma') \models \mathsf{Enabled}(\mathcal{D}'_{\mathsf{t}'}) * \mathsf{true})\}\ .$$

Since for any t' such that $\mathsf{t}' \neq \mathsf{t}$ we have $G_{\mathsf{t}} \Rightarrow R_{\mathsf{t}'}$, and since $\mathsf{wffAct}(R, \mathcal{D}')$, $\mathcal{D}' \Subset \mathcal{D}$ and $\mathsf{Enabled}(\mathcal{D}) \Rightarrow I$, we can prove:

$$k = 0 \Longrightarrow \xi_0 \subseteq \xi'_0\ .$$

Let

$$ws' = (0,0) :: \mathsf{inchead}(ws_1, (w_1, 0)) = (0,0) :: (w_1, 0)\ .$$

Thus $ws' <_{\mathcal{H}+1} ws$. Let

$$\mathbb{ws}' = ((0,0),0) :: \mathsf{inchead}(\mathbb{ws}_1, ((k'_s, w_1), 0)) = ((0,0),0) :: ((k'_s, w_1), 0)\ .$$

If $\xi'_0 \neq \emptyset \vee \Sigma' \models \neg\mathsf{en}(\mathbb{C}'')$, let $wk' = 1$ and $\xi' = \xi'_0$; otherwise let $wk' = \mathcal{H} + 1$ and $\xi' = \emptyset$. Then we know in either case $\xi' = \xi'_0$.

One of the following holds:

- If $\xi_0 \neq \emptyset \vee \Sigma \models \neg\mathsf{en}(\mathbb{C})$, then $wk = 1$ and $\xi = \xi_0$.
  If $k = 0$, we know
  $$\mathbb{ws}' <^{wk}_{\mathcal{H}+1} \mathbb{ws} \quad \text{or} \quad \mathbb{ws}' \approx^{wk}_{\mathcal{H}+1} \mathbb{ws}.$$
  If $k = 0$,
  - Suppose $\xi_0 \neq \emptyset$. Since $\xi_0 \subseteq \xi'_0$, we know $\xi'_0 \neq \emptyset$. Thus $wk' = 1$ and $\xi' = \xi'_0$.
  - Suppose $\Sigma \models \neg\mathsf{en}(\mathbb{C})$. Since $(\mathbb{C}, \Sigma \uplus \Sigma_F) \longrightarrow\!\triangleright^*_{\mathsf{t}} (\mathbb{C}'', \Sigma' \uplus \Sigma_F)$, we know $\Sigma' \models \neg\mathsf{en}(\mathbb{C}'')$. Thus $wk' = 1$ and $\xi' = \xi'_0$.
- If $\xi_0 = \emptyset \wedge \Sigma \models \mathsf{en}(\mathbb{C})$, then $wk = wk_1 + 1$ and $\xi = \xi_1$.
  Since $wk_1 \geq 1$, we know
  $$\mathbb{ws}' <^{wk}_{\mathcal{H}+1} \mathbb{ws}.$$

Below we prove:

$$\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (\mathsf{while}\ (B)\{C\}, \sigma) \preceq (\mathbb{C}'', \Sigma') \diamond (u, \mathbb{ws}', ws', aw, w, wk', \mathcal{H} + 1) \Downarrow_{\xi', \xi_a} p \wedge \neg B\ . \tag{B.2}$$

Proof: By co-induction. Suppose $\sigma = (s, h)$. Since $p \Rightarrow (B = B) * I$, we know

$$(\sigma, \Sigma') \models I * \mathsf{true},\ \text{and}\ \text{either}\ [\![B]\!]_s = \textbf{true}\ \text{or}\ [\![B]\!]_s = \textbf{false}\ .$$

We only need to prove the following *(1)*(*2*)(*3*)(*4*)(*5*).

(1)(a) $\xi' \cup \xi_a \subseteq s(\mathtt{TIDS})$ and $\mathsf{t} \notin \xi'$ and $\mathsf{t} \notin \xi_a$.

(b) For any $\mathsf{t}' \in \xi' \cup \xi_a$, we have $(\sigma, \Sigma') \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}'}) * \mathsf{true}$.

(c) If $(\sigma, \Sigma') \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}}) * \mathsf{true}$, then $\sigma \models \mathsf{en}(\mathsf{while}\ (B)\{C\})$.

(d) If $\sigma \models \neg\mathsf{en}(\mathsf{while}\ (B)\{C\})$ and $\Sigma' \models \mathsf{en}(\mathbb{C}'')$, then $\xi_a \neq \emptyset$.

(e) If $wk' = \mathcal{H} + 1$, then $\xi' = \emptyset$. If $wk' < \mathcal{H} + 1$, then $\xi' \neq \emptyset \vee \Sigma' \models \neg\mathsf{en}(\mathbb{C}'')$.

(f) $|\mathbb{ws}'| \leq \mathcal{H} + 1$ and $(1 \leq wk' \leq |\mathbb{ws}'| - 1) \vee (wk' = \mathcal{H} + 1)$.

Proof: Immediate from $\mathcal{D}' \Subset \mathcal{D}$.

(*2*) If while $(B)\{C\} = \textbf{skip}$, then ...
Proof: It is vacantly true.

(*3*) For any σ_F, $(\text{while } (B)\{C\}, \sigma \uplus \sigma_F) \not\longrightarrow_{\mathrm{t}}$ **abort**.
Proof: It holds because $[\![B]\!]_s$ is not undefined.

(*4*) If $(\text{while } (B)\{C\}, \sigma \uplus \sigma_F) \longrightarrow_{\mathrm{t}} (C', \sigma'')$, then there exist σ', $\mathbb{C}'''$, Σ'', k, u', $\mathbb{ws}''$, ws'', aw', w', wk'', ξ'' and ξ'_a such that

(a) $\sigma'' = \sigma' \uplus \sigma_F$, and
(b) $(\mathbb{C}'', \Sigma' \uplus \Sigma_F) \longrightarrow_{\mathrm{t}}^{*} (\mathbb{C}''', \Sigma'' \uplus \Sigma_F)$, and
(c) $\mathcal{D}, R, G \models_{\mathrm{t}}^{\chi} (C', \sigma') \leq (\mathbb{C}''', \Sigma'') \diamond (u', \mathbb{ws}'', ws'', aw', w', wk'', \mathcal{H}+1) \Downarrow_{\xi'', \xi'_a} p \wedge \neg B$, and
(d) $((\sigma, \Sigma'), (\sigma', \Sigma''), k) \models G_{\mathrm{t}} * \mathsf{True}$, and
(e) either $u' <_k u$ and $k > 0$,
or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}'' <_{\mathcal{H}+1}^{wk'} \mathbb{ws}'$,
or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}'' \approx_{\mathcal{H}+1}^{wk'} \mathbb{ws}'$ and $wk'' < wk'$,
or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}'' \approx_{\mathcal{H}+1}^{wk'} \mathbb{ws}'$ and $wk'' = wk' < \mathcal{H}+1$ and $\xi' \subseteq \xi''$; and
(f) if $((\sigma, \Sigma'), (\sigma', \Sigma'')) \models \langle [\mathcal{D}_{\mathrm{t}}] \rangle * \mathsf{True}$ and $k = 0$, then $ws'' <_{\mathcal{H}+1} ws'$.

Proof: We have two cases depending on whether $[\![B]\!]_s$ is **true** or **false**.

- If $[\![B]\!]_s = \textbf{true}$, we know $(\text{while } (B)\{C\}, \sigma \uplus \sigma_F) \longrightarrow_{\mathrm{t}} (C; \text{while } (B)\{C\}, \sigma \uplus \sigma_F)$. Also we know $((\sigma, \Sigma'), (u, w_1), \mathbb{C}'') \models p_{\mathrm{t}} \wedge B$. Since $p \wedge B \Rightarrow J * \mathsf{true} \wedge \mathsf{arem}(\textbf{await}(\mathbb{B}')\{\mathbb{C}'\})$, we know
$$(\sigma, \Sigma') \models J * \mathsf{true} \quad \text{and} \quad \mathbb{C}'' = \textbf{await}(\mathbb{B}')\{\mathbb{C}'\}.$$
  - If $\xi'_0 \neq \emptyset \vee \Sigma' \models \neg\mathsf{en}(\mathbb{C}'')$, then $wk' = 1$ and $\xi' = \xi'_0$. We have two cases below:
    - If $(\sigma, \Sigma') \models \mathsf{Enabled}(\mathcal{D}_{\mathrm{t}}) * \mathsf{true}$, we know
    $$((\sigma, \Sigma'), (u, w_1), \mathbb{C}'') \models p_{\mathrm{t}} \wedge B \wedge \mathsf{Enabled}(\mathcal{D}_{\mathrm{t}}) * \mathsf{true}\ .$$
    Since $p \wedge B \wedge \mathsf{Enabled}(\mathcal{D}_{\mathrm{t}}) * \mathsf{true} \Rightarrow p' * (\Diamond \wedge \mathsf{emp})$, we know there exists w'_1 such that $w'_1 < w_1$ and
    $$((\sigma, \Sigma'), (u, w'_1), \mathbb{C}'') \models p'_{\mathrm{t}}\ .$$
    From $\mathcal{D}, R, G \models_{\chi} \{p'\}C\{p\}$ and $\mathsf{height}(C) = \mathcal{H}$, let $ws'_1 = (0, |C|)$ and $\mathbb{ws}'_1 = ((0,0), |C|)$ and $wk'_1 = \mathcal{H}$, we get: there exists aw' and ξ'_a such that
    $$\mathcal{D}, R, G \models_{\mathrm{t}}^{\chi} (C, \sigma) \leq (\mathbb{C}'', \Sigma') \diamond (u, \mathbb{ws}'_1, ws'_1, aw', w'_1, wk'_1, \mathcal{H}) \Downarrow_{\emptyset, \xi'_a} p\ .$$
    Let
    $$ws'' = (0,0) :: (w'_1, |C|+1) \quad \text{and} \quad \mathbb{ws}'' = ((0,0),0) :: ((k'_s, w'_1), |C|+1)\ .$$
    Then, by the co-induction hypothesis, we know
    $$\mathcal{D}, R, G \models_{\mathrm{t}}^{\chi} (C; \text{while } (B)\{C\}, \sigma) \leq$$
    $$(\mathbb{C}'', \Sigma') \diamond (u, \mathbb{ws}'', ws'', aw', w, wk', \mathcal{H}+1) \Downarrow_{\xi', \xi'_a} p \wedge \neg B\ .$$
    Also we have $ws'' <_{\mathcal{H}+1} ws'$ and $\mathbb{ws}'' <_{\mathcal{H}+1}^{wk'} \mathbb{ws}'$. Since $(\sigma, \Sigma') \models I * \mathsf{true}$, we can prove
    $$((\sigma, \Sigma'), (\sigma, \Sigma'), 0) \models G_{\mathrm{t}} * \mathsf{True}\ .$$
    - Otherwise, from $p \wedge B \Rightarrow p'$, we know
    $$((\sigma, \Sigma'), (u, w_1), \mathbb{C}'') \models p'_{\mathrm{t}}\ .$$
    From $\mathcal{D}, R, G \models_{\chi} \{p'\}C\{p\}$ and $\mathsf{height}(C) = \mathcal{H}$, let $ws'_1 = (0, |C|)$ and $\mathbb{ws}'_1 = ((0,0), |C|)$ and $wk'_1 = \mathcal{H}$, we get: there exists aw' and ξ'_a such that
    $$\mathcal{D}, R, G \models_{\mathrm{t}}^{\chi} (C, \sigma) \leq (\mathbb{C}'', \Sigma') \diamond (u, \mathbb{ws}'_1, ws'_1, aw', w_1, wk'_1, \mathcal{H}) \Downarrow_{\emptyset, \xi'_a} p\ .$$
    Let
    $$ws'' = (0,0) :: (w_1, |C|+1) \quad \text{and} \quad \mathbb{ws}'' = ((0,0),0) :: ((k'_s, w_1), |C|+1)\ .$$
    Then, by the co-induction hypothesis, we know

$$\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (C; \mathsf{while}\ (B)\{C\}, \sigma) \preceq (\mathbb{C}'', \Sigma') \diamond (u, \mathbb{ws}'', ws'', aw', w, wk', \mathcal{H}+1) \Downarrow_{\xi', \xi'_a} p \wedge \neg B \ .$$

Also we know $\mathbb{ws}'' \approx^{wk'}_{\mathcal{H}+1} \mathbb{ws}'$ and $wk' < \mathcal{H}+1$. Since $(\sigma, \Sigma') \models I * \mathsf{true}$, we can prove:

$$((\sigma, \Sigma'), (\sigma, \Sigma'), 0) \models G_{\mathsf{t}} * \mathsf{True} \ .$$

- If $\xi'_0 = \emptyset \wedge \Sigma' \models \mathsf{en}(\mathbb{C}'')$, then $wk' = \mathcal{H}+1$ and $\xi' = \emptyset$. Also from $J \Rightarrow (R, G\colon \mathcal{D}' \xrightarrow{f} (Q, \mathbb{B}'))$, we know $(\sigma, \Sigma') \models Q_{\mathsf{t}} * \mathsf{true}$. Thus we know

$$((\sigma, \Sigma'), (u, w_1), \mathbb{C}'') \models p_{\mathsf{t}} \wedge B \wedge Q_{\mathsf{t}} * \mathsf{true} \ .$$

Since $p \wedge B \wedge Q * \mathsf{true} \Rightarrow p' * (\Diamond \wedge \mathsf{emp})$, we know there exists w'_1 such that $w'_1 < w_1$ and

$$((\sigma, \Sigma'), (u, w'_1), \mathbb{C}'') \models p'_{\mathsf{t}} \ .$$

From $\mathcal{D}, R, G \models_{\chi} \{p'\}C\{p\}$ and $\mathsf{height}(C) = \mathcal{H}$, let $ws'_1 = (0, |C|)$ and $\mathbb{ws}'_1 = ((0,0), |C|)$ and $wk'_1 = \mathcal{H}$, we get: there exists aw' and ξ'_a such that

$$\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (C, \sigma) \preceq (\mathbb{C}'', \Sigma') \diamond (u, \mathbb{ws}'_1, ws'_1, aw', w'_1, wk'_1, \mathcal{H}) \Downarrow_{\emptyset, \xi'_a} p \ .$$

Let

$$ws'' = (0,0) :: (w'_1, |C|+1) \ \text{ and } \ \mathbb{ws}'' = ((0,0),0) :: ((k'_s, w'_1), |C|+1) \ .$$

Then, by the co-induction hypothesis, we know

$$\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (C; \mathsf{while}\ (B)\{C\}, \sigma) \preceq (\mathbb{C}'', \Sigma') \diamond (u, \mathbb{ws}'', ws'', aw', w, wk', \mathcal{H}+1) \Downarrow_{\xi', \xi'_a} p \wedge \neg B \ .$$

Also we have $ws'' <_{\mathcal{H}+1} ws'$ and $\mathbb{ws}'' <^{wk'}_{\mathcal{H}+1} \mathbb{ws}'$. Since $(\sigma, \Sigma') \models I * \mathsf{true}$, we can prove:

$$((\sigma, \Sigma'), (\sigma, \Sigma'), 0) \models G_{\mathsf{t}} * \mathsf{True} \ .$$

- If $[\![B]\!]_s = \mathbf{false}$, we know $(\mathsf{while}\ (B)\{C\}, \sigma \uplus \sigma_F) \longrightarrow_{\mathsf{t}} (\mathbf{skip}, \sigma \uplus \sigma_F)$. By (SKIP) rule, let

$$\mathbb{ws}'' = ((0,0),0) \ \text{ and } \ ws'' = (0,0) \ \text{ and } \ wk'' = \mathcal{H}+1 \ \text{ and } \ \xi'' = \emptyset,$$

we know

$$\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (\mathbf{skip}, \sigma) \preceq (\mathbb{C}'', \Sigma') \diamond (u, \mathbb{ws}'', ws'', aw, w, wk'', \mathcal{H}+1) \Downarrow_{\xi'', \xi_a} p \wedge \neg B \ .$$

Also we have $ws'' <_{\mathcal{H}+1} ws'$ and $\mathbb{ws}'' <^{wk'}_{\mathcal{H}+1} \mathbb{ws}'$. Since $(\sigma, \Sigma') \models I * \mathsf{true}$, we can prove

$$((\sigma, \Sigma'), (\sigma, \Sigma'), 0) \models G_{\mathsf{t}} * \mathsf{True} \ .$$

(5) If $((\sigma, \Sigma'), (\sigma', \Sigma''), k) \models R_{\mathsf{t}} * \mathsf{Id}$, then there exist u', $\mathbb{ws}''$, ws'', aw', w', wk'', ξ_d, ξ'' and ξ'_a such that

(a) $\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (\mathsf{while}\ (B)\{C\}, \sigma') \preceq (\mathbb{C}'', \Sigma'') \diamond (u', \mathbb{ws}'', ws'', aw', w', wk'', \mathcal{H}+1) \Downarrow_{\xi'', \xi'_a} p \wedge \neg B$, and

(b) $u' \approx_k u$, and
$k = 0 \Longrightarrow w' = w$, and

(c) $\xi_d = \{\mathsf{t}' \mid (\mathsf{t}' \in \xi') \wedge (((\sigma, \Sigma'), (\sigma', \Sigma'')) \models \langle \mathcal{D}_{\mathsf{t}'} \rangle * \mathsf{Id})\}$ and
$k = 0 \Longrightarrow \mathbb{ws}'' <^{wk'}_{\mathcal{H}+1} \mathbb{ws}' \vee \mathbb{ws}'' \approx^{wk'}_{\mathcal{H}+1} \mathbb{ws}'$ and
$k = 0 \wedge wk' < \mathcal{H}+1 \wedge (\xi_d \neq \emptyset \vee (\Sigma' \models \neg\mathsf{en}(\mathbb{C}'') \wedge \Sigma'' \models \mathsf{en}(\mathbb{C}'''))) \Longrightarrow \mathbb{ws}'' <^{wk'}_{\mathcal{H}+1} \mathbb{ws}'$ and
$k = 0 \wedge (\xi' \backslash \xi_d \neq \emptyset \vee \Sigma'' \models \neg\mathsf{en}(\mathbb{C}'')) \Longrightarrow wk'' \leq wk'$, and
$k = 0 \wedge wk'' = wk' \Longrightarrow \xi' \backslash \xi_d \subseteq \xi''$, and

(d) if $k = 0$ and $(\sigma, \Sigma') \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}}) * \mathsf{true}$, then $ws'' \preceq_{\mathcal{H}+1} ws'$.

Proof: Since $\mathsf{Sta}(p, R * \mathsf{Id})$, we know there exist u' and w'_1 such that

$$((\sigma', \Sigma''), (u', w'_1), \mathbb{C}'') \models p_{\mathsf{t}} \ \text{ and } \ u' \approx_k u \ \text{ and } \ k = 0 \implies w'_1 = w_1 \ .$$

Also we know

$$(\sigma', \Sigma'') \models J * \mathsf{true} \ .$$

Suppose $k''_s = f_{\mathsf{t}}(\sigma', \Sigma'')$. Since $J \Rightarrow (R, G\colon \mathcal{D}' \xrightarrow{f} (Q, \mathbb{B}'))$, we can prove

$$k = 0 \Longrightarrow k_s'' \leq k_s' .$$

Let

$$\xi_0'' = \{\mathrm{t}'' \mid (\mathrm{t}'' \neq \mathrm{t}) \wedge ((\sigma', \Sigma'') \models \mathsf{Enabled}(\mathcal{D}_{\mathrm{t}''}') * \mathsf{true})\} \text{ and}$$
$$\xi_d = \{\mathrm{t}' \mid (\mathrm{t}' \in \xi') \wedge (((\sigma, \Sigma'), (\sigma', \Sigma'')) \models \langle \mathcal{D}_{\mathrm{t}'} \rangle * \mathsf{Id})\}.$$

Since $\mathsf{Enabled}(\mathcal{D}) \Rightarrow I$, $\mathcal{D}' \Subset \mathcal{D}$ and $\mathsf{wffAct}(R, \mathcal{D}')$, we can prove:

$$k = 0 \implies \xi' \backslash \xi_d \subseteq \xi_0'' .$$

If $w_1' = w_1$, let $w' = w$; otherwise let $w' = w_1'$. Thus we know $w_1' \leq w'$. Also we know: if $k = 0$, then $w' = w$. Let

$$ws'' = (0,0) :: \mathsf{inchead}(ws_1, (w_1', 0)) = (0,0) :: (w_1', 0) \text{ and}$$
$$\mathbb{ws}'' = ((0,0),0) :: \mathsf{inchead}(\mathbb{ws}_1, ((k_s'', w_1'), 0)) = ((0,0),0) :: ((k_s'', w_1'), 0) .$$

Thus if $k = 0$, then $ws'' = ws'$, and

$$\mathbb{ws}'' <^{wk'}_{\mathcal{H}+1} \mathbb{ws}' \quad \text{or} \quad \mathbb{ws}'' \approx^{wk'}_{\mathcal{H}+1} \mathbb{ws}'.$$

If $\xi_0'' \neq \emptyset \vee \Sigma'' \models \neg\mathsf{en}(\mathbb{C}'')$, let $wk'' = 1$ and $\xi'' = \xi_0''$; otherwise let $wk'' = \mathcal{H} + 1$ and $\xi'' = \emptyset$.

By the co-induction hypothesis, we know

$$\mathcal{D}, R, G \models^{\chi}_{\mathrm{t}} (\mathsf{while}\ (B)\{C\}, \sigma') \leq (\mathbb{C}'', \Sigma'') \diamond (u, \mathbb{ws}'', ws'', aw, w', wk'', \mathcal{H}+1) \Downarrow_{\xi'', \xi_a} p \wedge \neg B .$$

Suppose $k = 0$. If $wk' < \mathcal{H} + 1$, then $wk' = 1$. If $\xi_d \neq \emptyset$, then there exists t' such that $\mathrm{t}' \in \xi'$ and $((\sigma, \Sigma'), (\sigma', \Sigma'')) \models \langle \mathcal{D}_{\mathrm{t}'} \rangle * \mathsf{Id}$. Since $\mathcal{D}' \Subset \mathcal{D}$, we can prove

$$((\sigma, \Sigma'), (\sigma', \Sigma'')) \models \langle \mathcal{D}_{\mathrm{t}'}' \rangle * \mathsf{Id} .$$

Since $J \Rightarrow (R, G : \mathcal{D}' \xrightarrow{f} (Q, \mathbb{B}'))$, we know for any $\mathrm{t}' \neq \mathrm{t}$, σ' and Σ'', if $((\sigma, \Sigma'), (\sigma', \Sigma''), 0) \models (\langle \mathcal{D}_{\mathrm{t}'}' \rangle \wedge R_{\mathrm{t}}) * \mathsf{Id}$, then $f_{\mathrm{t}}(\sigma', \Sigma'') < k_s'$. Thus we can prove:

$$k_s'' < k_s' .$$

Also if $\Sigma' \models \neg\mathsf{en}(\mathbb{C}'') \wedge \Sigma'' \models \mathsf{en}(\mathbb{C}'')$, from $J \Rightarrow (R, G : \mathcal{D}' \xrightarrow{f} (Q, \mathbb{B}'))$, we can still prove:

$$k_s'' < k_s' .$$

Thus $\mathbb{ws}'' <^{wk'}_{\mathcal{H}+1} \mathbb{ws}'$ holds.

If $k = 0 \wedge (\xi' \backslash \xi_d \neq \emptyset \vee \Sigma'' \models \neg\mathsf{en}(\mathbb{C}''))$, we know $wk'' = 1$. Thus $wk'' \leq wk'$.

If $k = 0 \wedge wk'' = wk'$, we know $\xi' \backslash \xi_d \subseteq \xi''$.

Thus we have proved (B.2).

(5) If $((\sigma, \Sigma), (\sigma', \Sigma'), k) \models R_{\mathrm{t}} * \mathsf{Id}$, then there exist u', $\mathbb{ws}'$, ws', aw', w', wk', ξ_d, ξ_{ad}, ξ' and ξ_a' such that

(a) $\mathcal{D}, R, G \models^{\chi}_{\mathrm{t}} (C_1; \mathsf{while}\ (B)\{C\}, \sigma') \leq (\mathbb{C}, \Sigma') \diamond (u', \mathbb{ws}', ws', aw', w', wk', \mathcal{H}+1) \Downarrow_{\xi', \xi_a'} p \wedge \neg B$, and

(b) $u' \approx_k u$, and
$k = 0 \Longrightarrow w' = w$, and

(c) $\xi_d = \{\mathrm{t}' \mid (\mathrm{t}' \in \xi) \wedge (((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle \mathcal{D}_{\mathrm{t}'} \rangle * \mathsf{Id})\}$ and
$k = 0 \Longrightarrow \mathbb{ws}' <^{wk}_{\mathcal{H}+1} \mathbb{ws} \vee \mathbb{ws}' \approx^{wk}_{\mathcal{H}+1} \mathbb{ws}$ and
$k = 0 \wedge wk < \mathcal{H} + 1 \wedge (\xi_d \neq \emptyset \vee (\Sigma \models \neg\mathsf{en}(\mathbb{C}) \wedge \Sigma' \models \mathsf{en}(\mathbb{C}))) \Longrightarrow \mathbb{ws}' <^{wk}_{\mathcal{H}+1} \mathbb{ws}$ and
$k = 0 \wedge (\xi \backslash \xi_d \neq \emptyset \vee \Sigma' \models \neg\mathsf{en}(\mathbb{C})) \Longrightarrow wk' \leq wk$, and
$k = 0 \wedge wk' = wk \Longrightarrow \xi \backslash \xi_d \subseteq \xi'$, and

(d) if $k = 0$ and $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathrm{t}}) * \mathsf{true}$, then $ws' \leq_{\mathcal{H}+1} ws$; and

(e) $\xi_{ad} = \{\mathrm{t}' \mid (\mathrm{t}' \in \xi_a) \wedge (((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle \mathcal{D}_{\mathrm{t}'} \rangle * \mathsf{Id})\}$ and
$k = 0 \wedge \mathsf{is\_await}(C_1; \mathsf{while}\ (B)\{C\}) \Longrightarrow \xi_a \backslash \xi_{ad} \subseteq \xi_a'$ and
$k = 0 \wedge \mathsf{is\_await}(C_1; \mathsf{while}\ (B)\{C\}) \wedge (\xi_{ad} \neq \emptyset \vee (\Sigma \models \neg\mathsf{en}(\mathbb{C}) \wedge \Sigma' \models \mathsf{en}(\mathbb{C}))) \Longrightarrow aw' < aw$ and

$(\chi = \text{sfair}) \wedge k = 0 \wedge \text{is\_await}(C_1; \text{while } (B)\{C\}) \wedge (\sigma \models \neg\text{en}(C_1; \text{while } (B)\{C\})) \wedge (\sigma' \models \neg\text{en}(C_1; \text{while } (B)\{C\})) \Longrightarrow aw' \leq aw$ and
$(\chi = \text{wfair}) \wedge k = 0 \wedge \text{is\_await}(C_1; \text{while } (B)\{C\}) \Longrightarrow aw' \leq aw$.

Proof: From $\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (C_1, \sigma) \preceq (\mathbb{C}, \Sigma) \diamond (u, \mathbb{ws}_1, ws_1, aw, w_1, wk_1, \mathcal{H}) \Downarrow_{\xi_1, \xi_a} p$, we know there exist u', $\mathbb{ws}'_1$, ws'_1, aw', w'_1, wk'_1, ξ'_d, ξ_{ad}, ξ'_1 and ξ'_a such that

(A) $\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (C_1, \sigma') \preceq (\mathbb{C}, \Sigma') \diamond (u', \mathbb{ws}'_1, ws'_1, aw', w'_1, wk'_1, \mathcal{H}) \Downarrow_{\xi'_1, \xi'_a} p$, and
(B) $u' \approx_k u$, and
$k = 0 \Longrightarrow w'_1 = w_1$, and
(C) $\xi'_d = \{\mathsf{t}' \mid (\mathsf{t}' \in \xi_1) \wedge (((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle \mathcal{D}_{\mathsf{t}'} \rangle * \mathsf{Id})\}$ and
$k = 0 \Longrightarrow \mathbb{ws}'_1 <^{wk_1}_{\mathcal{H}} \mathbb{ws}_1 \vee \mathbb{ws}'_1 \approx^{wk_1}_{\mathcal{H}} \mathbb{ws}_1$ and
$k = 0 \wedge wk_1 < \mathcal{H} \wedge (\xi'_d \neq \emptyset \vee (\Sigma \models \neg\text{en}(\mathbb{C}) \wedge \Sigma' \models \text{en}(\mathbb{C}))) \Longrightarrow \mathbb{ws}'_1 <^{wk_1}_{\mathcal{H}} \mathbb{ws}_1$ and
$k = 0 \wedge (\xi_1 \backslash \xi'_d \neq \emptyset \vee \Sigma' \models \neg\text{en}(\mathbb{C})) \Longrightarrow wk'_1 \leq wk_1$, and
$k = 0 \wedge wk'_1 = wk_1 \Longrightarrow \xi_1 \backslash \xi'_d \subseteq \xi'_1$, and
(D) if $k = 0$ and $(\sigma, \Sigma) \models \text{Enabled}(\mathcal{D}_{\mathsf{t}}) * \text{true}$, then $ws'_1 \leq_{\mathcal{H}} ws_1$; and
(E) $\xi_{ad} = \{\mathsf{t}' \mid (\mathsf{t}' \in \xi_a) \wedge (((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle \mathcal{D}_{\mathsf{t}'} \rangle * \mathsf{Id})\}$ and
$k = 0 \wedge \text{is\_await}(C_1) \Longrightarrow \xi_a \backslash \xi_{ad} \subseteq \xi'_a$ and
$k = 0 \wedge \text{is\_await}(C_1) \wedge (\xi_{ad} \neq \emptyset \vee (\Sigma \models \neg\text{en}(\mathbb{C}) \wedge \Sigma' \models \text{en}(\mathbb{C}))) \Longrightarrow aw' < aw$ and
$(\chi = \text{sfair}) \wedge k = 0 \wedge \text{is\_await}(C_1) \wedge (\sigma \models \neg\text{en}(C_1)) \wedge (\sigma' \models \neg\text{en}(C_1)) \Longrightarrow aw' \leq aw$ and
$(\chi = \text{wfair}) \wedge k = 0 \wedge \text{is\_await}(C_1) \Longrightarrow aw' \leq aw$.

Since $(\sigma, \Sigma) \models J * \text{true}$ and $\text{Sta}(J, G \vee R)$, we know

$$(\sigma', \Sigma') \models J * \text{true} .$$

Suppose $k'_s = f_{\mathsf{t}}(\sigma', \Sigma')$. Since $J \Rightarrow (R, G\colon \mathcal{D}' \xrightarrow{f} (Q, \mathbb{B}'))$, we can prove

$$k = 0 \Longrightarrow k'_s \leq k_s .$$

Let

$$\xi'_0 = \{\mathsf{t}'' \mid (\mathsf{t}'' \neq \mathsf{t}) \wedge ((\sigma', \Sigma') \models \text{Enabled}(\mathcal{D}'_{\mathsf{t}''}) * \text{true})\},$$
$$\xi_d = \{\mathsf{t}' \mid (\mathsf{t}' \in \xi) \wedge (((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle \mathcal{D}_{\mathsf{t}'} \rangle * \mathsf{Id})\} \text{ and}$$
$$\xi''_d = \{\mathsf{t}' \mid (\mathsf{t}' \in \xi_0) \wedge (((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle \mathcal{D}_{\mathsf{t}'} \rangle * \mathsf{Id})\}.$$

Let

$$ws' = (0,0) :: \text{inchead}(ws'_1, (w_1, 1)) \ , \ \mathbb{ws}' = ((0,0), 0) :: \text{inchead}(\mathbb{ws}'_1, ((k'_s, w'_1), 1)) .$$

If $w'_1 = w_1$, let $w' = w$; otherwise let $w' = w'_1$. Thus we know $w'_1 \leq w'$.
Also we know: if $k = 0$, then $w' = w$, and if $ws'_1 \leq_{\mathcal{H}} ws_1$ then $ws' \leq_{\mathcal{H}+1} ws$.
If $\xi'_0 \neq \emptyset \vee \Sigma' \models \neg\text{en}(\mathbb{C})$, let $wk' = 1$ and $\xi' = \xi'_0$; otherwise let $wk' = wk'_1 + 1$ and $\xi' = \xi'_1$.
Then, by the co-induction hypothesis, we know

$$\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (C'_1; \text{while } (B)\{C\}, \sigma') \preceq (\mathbb{C}, \Sigma') \diamond (u', \mathbb{ws}', ws', aw', w', wk', \mathcal{H} + 1) \Downarrow_{\xi', \xi'_a} p \wedge \neg B .$$

One of the following holds:

- If $\xi_0 \neq \emptyset \vee \Sigma \models \neg\text{en}(\mathbb{C})$, then $wk = 1$ and $\xi = \xi_0$.
  Thus $\xi_d = \xi''_d$. Since $\text{Enabled}(\mathcal{D}) \Rightarrow I$, $\mathcal{D}' \preccurlyeq \mathcal{D}$ and $\text{wffAct}(R, \mathcal{D}')$, we can prove:
  $$k = 0 \implies \xi_0 \backslash \xi''_d \subseteq \xi'_0 .$$
  If $k = 0$, we know $\mathbb{ws}'_1 <^{wk_1}_{\mathcal{H}} \mathbb{ws}_1$ or $\mathbb{ws}'_1 \approx^{wk_1}_{\mathcal{H}} \mathbb{ws}_1$. Since $wk_1 \geq 1$, we know
  $$\mathbb{ws}' <^{wk}_{\mathcal{H}+1} \mathbb{ws} \quad \text{or} \quad \mathbb{ws}' \approx^{wk}_{\mathcal{H}+1} \mathbb{ws}.$$
  Suppose $k = 0$. If $\xi_d \neq \emptyset$, then there exists t' such that $\mathsf{t}' \in \xi$ and $((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle \mathcal{D}_{\mathsf{t}'} \rangle * \mathsf{Id}$. Since $\mathcal{D}' \preccurlyeq \mathcal{D}$, we can prove
  $$((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle \mathcal{D}'_{\mathsf{t}'} \rangle * \mathsf{Id} .$$
  Since $J \Rightarrow (R, G\colon \mathcal{D}' \xrightarrow{f} (Q, \mathbb{B}'))$, we know

$$k_s' < k_s\ .$$

Also if $\Sigma \models \neg\mathsf{en}(\mathbb{C}) \wedge \Sigma' \models \mathsf{en}(\mathbb{C})$, from $J \Rightarrow (R,G\colon \mathcal{D}' \xrightarrow{f} (Q,\mathbb{B}'))$, we can still prove:

$$k_s' < k_s\ .$$

Thus $\mathbb{ws}' <^{wk}_{\mathcal{H}+1} \mathbb{ws}$ holds.

If $k = 0 \wedge (\xi\backslash\xi_d \neq \emptyset \vee \Sigma' \models \neg\mathsf{en}(\mathbb{C}))$, we know $wk' = 1$. Thus $wk' \leq wk$.

If $k = 0 \wedge wk' = wk$, we know $\xi\backslash\xi_d \subseteq \xi'$.

- If $\xi_0 = \emptyset \wedge \Sigma \models \mathsf{en}(\mathbb{C})$, then $wk = wk_1 + 1$ and $\xi = \xi_1$.
  Thus $\xi_d = \xi_d'$. If $k = 0$, we know $\mathbb{ws}_1' <^{wk_1}_{\mathcal{H}} \mathbb{ws}_1$ or $\mathbb{ws}_1' \approx^{wk_1}_{\mathcal{H}} \mathbb{ws}_1$. Thus we know
  $$\mathbb{ws}' <^{wk}_{\mathcal{H}+1} \mathbb{ws} \quad \text{or} \quad \mathbb{ws}' \approx^{wk}_{\mathcal{H}+1} \mathbb{ws}.$$
  If $k = 0 \wedge wk < \mathcal{H} + 1 \wedge (\xi_d \neq \emptyset \vee (\Sigma \models \neg\mathsf{en}(\mathbb{C}) \wedge \Sigma' \models \mathsf{en}(\mathbb{C})))$, we know $k = 0 \wedge wk_1 < \mathcal{H} \wedge (\xi_d' \neq \emptyset \vee (\Sigma \models \neg\mathsf{en}(\mathbb{C}) \wedge \Sigma' \models \mathsf{en}(\mathbb{C})))$. Thus $\mathbb{ws}_1' <^{wk_1}_{\mathcal{H}} \mathbb{ws}_1$. Thus $\mathbb{ws}' <^{wk}_{\mathcal{H}+1} \mathbb{ws}$.
  If $k = 0 \wedge (\xi\backslash\xi_d \neq \emptyset \vee \Sigma' \models \neg\mathsf{en}(\mathbb{C}))$, we know $wk_1' \leq wk_1$. Thus $wk' \leq wk$.
  If $k = 0 \wedge wk' = wk$, we know $wk' = wk_1' + 1$, $wk_1' = wk_1$ and $\xi' = \xi_1'$. Thus $\xi\backslash\xi_d \subseteq \xi'$.

Thus we are done. □

Lemma B.11. *If*

(1) $p \wedge B \Rightarrow p'$; $p \wedge B \wedge (\mathsf{Enabled}(\mathcal{D}) \vee Q) * \mathsf{true} \Rightarrow p' * (\Diamond \wedge \mathsf{emp})$;
(2) $\mathcal{D}, R, G \models_\chi \{p'\}C\{p\}$;
(3) $p \Rightarrow (B = B) * I$; $J \Rightarrow I$; $\mathsf{Sta}(J, R \vee G)$; $Q \Rightarrow I$; $p \wedge B \Rightarrow J * \mathsf{true} \wedge \mathsf{arem}(\mathbf{await}(\mathbb{B}')\{\mathbb{C}'\})$;
(4) $J \Rightarrow (R,G\colon \mathcal{D}' \xrightarrow{f} (Q,\mathbb{B}'))$; $\mathcal{D}' \Subset \mathcal{D}$; $\mathit{wffAct}(R, \mathcal{D}')$;
(5) $I \triangleright \{R,G\}$; $\mathsf{Sta}(p, R * \mathsf{Id})$; $\mathsf{Enabled}(\mathcal{D}) \Rightarrow I$;
(6) for any t *and* t' *such that* $\mathsf{t} \neq \mathsf{t}'$, *we have* $G_\mathsf{t} \Rightarrow R_{\mathsf{t}'}$;
(7) $\mathcal{D}, R, G \models^\chi_\mathsf{t} (C_1, \sigma) \preceq (\mathbf{skip}, \Sigma) \diamond (u, \mathbb{ws}_1, ws_1, aw, w_1, wk_1, \mathcal{H}) \Downarrow_{\xi_1, \xi_a} p$;
(8) $(\sigma, \Sigma) \models J * \mathsf{true}$; $\mathsf{height}(C) = \mathcal{H}$; $\mathbf{await}(\mathbb{B}')\{\mathbb{C}'\} \neq \mathbf{await}(\mathbf{true})\{\mathbf{skip}\}$; $f_\mathsf{t}(\sigma, \Sigma) = k_s$; $w_1 \leq w$;
(9) $\xi_0 = \{\mathsf{t}' \mid (\mathsf{t}' \neq \mathsf{t}) \wedge ((\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}'_{\mathsf{t}'}) * \mathsf{true})\}$;
(10) $ws = (0,0) :: \mathsf{inchead}(ws_1, (w_1, 1))$;
(11) $\mathbb{ws} = ((0,0),0) :: \mathsf{inchead}(\mathbb{ws}_1, ((k_s, w_1), 1))$;
(12) one of the following holds:
- $\xi_0 \neq \emptyset$: $wk = 1 \wedge \xi = \xi_0$;
- $\xi_0 = \emptyset$: $wk = wk_1 + 1 \wedge \xi = \xi_1$;

then $\mathcal{D}, R, G \models^\chi_\mathsf{t} (C_1; \mathsf{while}\ (B)\{C\}, \sigma) \preceq (\mathbf{skip}, \Sigma) \diamond (u, \mathbb{ws}, ws, aw, w, wk, \mathcal{H} + 1) \Downarrow_{\xi, \xi_a} p \wedge \neg B$.

Proof. By co-induction. We only need to prove the following (1)(2)(3)(4)(5).

(1)(a) Suppose $\sigma = (s, h)$. Then $\xi \cup \xi_a \subseteq s(\mathtt{TIDS})$ and $\mathsf{t} \notin \xi$ and $\mathsf{t} \notin \xi_a$.
(b) For any $\mathsf{t}' \in \xi \cup \xi_a$, we have $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}'}) * \mathsf{true}$.
(c) If $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_\mathsf{t}) * \mathsf{true}$, then $\sigma \models \mathsf{en}(C_1; \mathsf{while}\ (B)\{C\})$.
(d) If $\sigma \models \neg\mathsf{en}(C_1; \mathsf{while}\ (B)\{C\})$ and $\Sigma \models \mathsf{en}(\mathbf{skip})$, then $\xi_a \neq \emptyset$.
(e) If $wk = \mathcal{H} + 1$, then $\xi = \emptyset$. If $wk < \mathcal{H} + 1$, then $\xi \neq \emptyset \vee \Sigma \models \neg\mathsf{en}(\mathbf{skip})$.
(f) $|\mathbb{ws}| \leq \mathcal{H} + 1$ and $(1 \leq wk \leq |\mathbb{ws}| - 1) \vee (wk = \mathcal{H} + 1)$.
Proof: From
$$\mathcal{D}, R, G \models^\chi_\mathsf{t} (C_1, \sigma) \preceq (\mathbf{skip}, \Sigma) \diamond (u, \mathbb{ws}_1, ws_1, aw, w_1, wk_1, \mathcal{H}) \Downarrow_{\xi_1, \xi_a} p\ ,$$
we know for any $\mathsf{t}' \in \xi_1 \cup \xi_a$, we have $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}'}) * \mathsf{true}$. Since $\mathcal{D}' \Subset \mathcal{D}$, we know for any $\mathsf{t}' \in \xi_0$, we have $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}'}) * \mathsf{true}$. Thus we are done.

(2) If $(C_1; \mathsf{while}\ (B)\{C\}) = \mathbf{skip}$, then ...
Proof: It is vacantly true.

(3) For any σ_F, $(C_1; \text{while } (B)\{C\}, \sigma \uplus \sigma_F) \not\longrightarrow_{\mathsf{t}}$ **abort**.
Proof: From
$$\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (C_1, \sigma) \preceq (\mathbf{skip}, \Sigma) \diamond (u, \mathbb{ws}_1, ws_1, aw, w_1, wk_1, \mathcal{H}) \Downarrow_{\xi_1, \xi_a} p \,,$$
we know $(C_1, \sigma \uplus \sigma_F) \not\longrightarrow_{\mathsf{t}}$ **abort**. By the operational semantics, we are done.

(4) If $(C_1; \text{while } (B)\{C\}, \sigma \uplus \sigma_F) \longrightarrow_{\mathsf{t}} (C', \sigma'')$, then there exist σ', k, u', $\mathbb{ws}'$, ws', aw', w', wk', ξ' and ξ'_a such that
(a) $\sigma'' = \sigma' \uplus \sigma_F$, and
(b) $\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (C', \sigma') \preceq (\mathbf{skip}, \Sigma) \diamond (u', \mathbb{ws}', ws', aw', w', wk', \mathcal{H}) \Downarrow_{\xi', \xi'_a} p \wedge \neg B$, and
(c) $((\sigma, \Sigma), (\sigma', \Sigma), k) \models G_{\mathsf{t}} * \mathsf{True}$, and
(d) either $u' <_k u$ and $k > 0$,
or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}' <^{wk}_{\mathcal{H}+1} \mathbb{ws}$,
or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}' \approx^{wk}_{\mathcal{H}+1} \mathbb{ws}$ and $wk' < wk$,
or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}' \approx^{wk}_{\mathcal{H}+1} \mathbb{ws}$ and $wk' = wk < \mathcal{H} + 1$ and $\xi \subseteq \xi'$; and
(e) if $((\sigma, \Sigma), (\sigma', \Sigma)) \models \langle [\mathcal{D}_{\mathsf{t}}] \rangle * \mathsf{True}$ and $k = 0$, then $ws' <_{\mathcal{H}+1} ws$.
Proof: We have two cases depending on whether C_1 is **skip** or not.
(I) If $(C_1; \text{while } (B)\{C\}, \sigma \uplus \sigma_F) \longrightarrow_{\mathsf{t}} (C'_1; \text{while } (B)\{C\}, \sigma'')$, then $(C_1, \sigma \uplus \sigma_F) \longrightarrow_{\mathsf{t}} (C'_1, \sigma'')$. From $\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (C_1, \sigma) \preceq (\mathbf{skip}, \Sigma) \diamond (u, \mathbb{ws}_1, ws_1, aw, w_1, wk_1, \mathcal{H}) \Downarrow_{\xi_1, \xi_a} p$, we know there exist σ', k, u', $\mathbb{ws}'_1$, ws'_1, aw', w'_1, wk'_1, ξ'_1 and ξ'_a such that
(A) $\sigma'' = \sigma' \uplus \sigma_F$, and
(B) $\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (C'_1, \sigma') \preceq (\mathbf{skip}, \Sigma) \diamond (u', \mathbb{ws}'_1, ws'_1, aw', w'_1, wk'_1, \mathcal{H}) \Downarrow_{\xi'_1, \xi'_a} p$, and
(C) $((\sigma, \Sigma), (\sigma', \Sigma), k) \models G_{\mathsf{t}} * \mathsf{True}$, and
(D) either $u' <_k u$ and $k > 0$,
or $u' = u$ and $k = 0$ and $w'_1 = w_1$ and $\mathbb{ws}'_1 <^{wk_1}_{\mathcal{H}} \mathbb{ws}_1$,
or $u' = u$ and $k = 0$ and $w'_1 = w_1$ and $\mathbb{ws}'_1 \approx^{wk_1}_{\mathcal{H}} \mathbb{ws}_1$ and $wk'_1 < wk_1$,
or $u' = u$ and $k = 0$ and $w'_1 = w_1$ and $\mathbb{ws}'_1 \approx^{wk_1}_{\mathcal{H}} \mathbb{ws}_1$ and $wk'_1 = wk_1 < \mathcal{H}$ and $\xi_1 \subseteq \xi'_1$; and
(E) if $((\sigma, \Sigma), (\sigma', \Sigma)) \models \langle [\mathcal{D}_{\mathsf{t}}] \rangle * \mathsf{True}$ and $k = 0$, then $ws'_1 <_{\mathcal{H}} ws_1$.
Since $((\sigma, \Sigma), (\sigma', \Sigma), k) \models G_{\mathsf{t}} * \mathsf{True}$, $(\sigma, \Sigma) \models J * \mathsf{true}$, $J \Rightarrow I$, $I \triangleright G$ and $\mathsf{Sta}(J, G \vee R)$, we know
$$(\sigma', \Sigma) \models J * \mathsf{true} \,.$$
Suppose $k'_s = f_{\mathsf{t}}(\sigma', \Sigma)$. Since $J \Rightarrow (R, G\colon \mathcal{D}' \xrightarrow{f} (Q, \mathbb{B}'))$, we can prove
$$k = 0 \implies k'_s \leq k_s \,.$$
Let
$$\xi'_0 = \{\mathsf{t}' \mid (\mathsf{t}' \neq \mathsf{t}) \wedge ((\sigma', \Sigma) \models \mathsf{Enabled}(\mathcal{D}'_{\mathsf{t}'}) * \mathsf{true})\} \,.$$
Since for any t' such that $\mathsf{t}' \neq \mathsf{t}$ we have $G_{\mathsf{t}} \Rightarrow R_{\mathsf{t}'}$, and since $\mathsf{wffAct}(R, \mathcal{D}')$, $\mathcal{D}' \Subset \mathcal{D}$ and $\mathsf{Enabled}(\mathcal{D}) \Rightarrow I$, we can prove:
$$k = 0 \implies \xi_0 \subseteq \xi'_0 \,.$$
Let
$$ws' = (0,0) :: \mathsf{inchead}(ws'_1, (w'_1, 1)) \,,\ \mathbb{ws}' = ((0,0), 0) :: \mathsf{inchead}(\mathbb{ws}'_1, ((k'_s, w'_1), 1)) \,.$$
If $w'_1 = w_1$, let $w' = w$; otherwise let $w' = w'_1$. Thus we know $w'_1 \leq w'$.
Also we know: if $k = 0$, then $w' = w$, and if $ws'_1 <_{\mathcal{H}} ws_1$ then $ws' <_{\mathcal{H}+1} ws$.
If $\xi'_0 \neq \emptyset$, let $wk' = 1$ and $\xi' = \xi'_0$; otherwise let $wk' = wk'_1 + 1$ and $\xi' = \xi'_1$.
Then, by the co-induction hypothesis, we know
$$\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (C'_1; \text{while } (B)\{C\}, \sigma') \preceq (\mathbf{skip}, \Sigma) \diamond (u', \mathbb{ws}', ws', aw', w', wk', \mathcal{H} + 1) \Downarrow_{\xi', \xi'_a} p \wedge \neg B \,.$$
One of the following holds:
- If $\xi_0 \neq \emptyset$, then $wk = 1$ and $\xi = \xi_0$.
If $k = 0$, we know $\mathbb{ws}'_1 <^{wk_1}_{\mathcal{H}} \mathbb{ws}_1$ or $\mathbb{ws}'_1 \approx^{wk_1}_{\mathcal{H}} \mathbb{ws}_1$. Since $wk_1 \geq 1$, we know

$$\mathbb{ws}' <^{wk}_{\mathcal{H}+1} \mathbb{ws} \quad \text{or} \quad \mathbb{ws}' \approx^{wk}_{\mathcal{H}+1} \mathbb{ws}.$$

Since $\xi_0 \subseteq \xi_0'$, we know $\xi_0' \neq \emptyset$. Thus $wk' = 1$ and $\xi' = \xi_0'$.

- If $\xi_0 = \emptyset$, then $wk = wk_1 + 1$ and $\xi = \xi_1$.
  If $k = 0$, we know $\mathbb{ws}_1' <^{wk_1}_{\mathcal{H}} \mathbb{ws}_1$ or $\mathbb{ws}_1' \approx^{wk_1}_{\mathcal{H}} \mathbb{ws}_1$. Thus we know
  $$\mathbb{ws}' <^{wk}_{\mathcal{H}+1} \mathbb{ws} \quad \text{or} \quad \mathbb{ws}' \approx^{wk}_{\mathcal{H}+1} \mathbb{ws}.$$
  If $k = 0$,
  - Suppose $\xi_0' \neq \emptyset$. Thus $wk' = 1$ and $\xi' = \xi_0'$. Thus $wk' < wk$.
  - Suppose $\xi_0' = \emptyset$. Thus $wk' = wk_1' + 1$ and $\xi' = \xi_1'$.
    - If $wk_1' < wk_1$, then $wk' < wk$.
    - If $wk_1' = wk_1 < \mathcal{H}$ and $\xi_1 \subseteq \xi_1'$, then $wk' = wk < \mathcal{H} + 1$ and $\xi \subseteq \xi'$.

(II) If $C_1 = \textbf{skip}$ and $(C_1; \text{while } (B)\{C\}, \sigma \uplus \sigma_F) \longrightarrow_{\mathrm{t}} (\text{while } (B)\{C\}, \sigma \uplus \sigma_F)$, from $\mathcal{D}, R, G \models^{\chi}_{\mathrm{t}} (C_1, \sigma) \preceq (\textbf{skip}, \Sigma) \diamond (u, \mathbb{ws}_1, ws_1, aw, w_1, wk_1, \mathcal{H}) \Downarrow_{\xi_1, \xi_a} p$, we know

(A) $((\sigma, \Sigma), (u, w_1), \textbf{skip}) \models p_{\mathrm{t}}$, and

(B) $\mathbb{ws}_1 = ((0,0),0)$ and $ws_1 = (0,0)$ and $wk_1 = \mathcal{H}$ and $\xi_1 = \emptyset$, and

(C) $((\sigma, \Sigma), (\sigma, \Sigma), 0) \models G_{\mathrm{t}} * \text{True}$.

Let
$$ws' = (0,0) :: \mathsf{inchead}(ws_1, (w_1, 0)) = (0,0) :: (w_1, 0)\ .$$
Thus $ws' <_{\mathcal{H}+1} ws$. Let
$$\mathbb{ws}' = ((0,0),0) :: \mathsf{inchead}(\mathbb{ws}_1, ((k_s, w_1), 0)) = ((0,0),0) :: ((k_s, w_1), 0)\ .$$
If $\xi_0 \neq \emptyset$, let $wk' = 1$ and $\xi' = \xi_0$; otherwise let $wk' = \mathcal{H} + 1$ and $\xi' = \emptyset$. Then we know in either case $\xi' = \xi_0$.

One of the following holds:

- If $\xi_0 \neq \emptyset$, then $wk = 1$ and $\xi = \xi_0$. Also $wk' = 1$ and $\xi' = \xi_0$.
  If $k = 0$, we know
  $$\mathbb{ws}' \approx^{wk}_{\mathcal{H}+1} \mathbb{ws}.$$
- If $\xi_0 = \emptyset$, then $wk = wk_1 + 1$ and $\xi = \xi_1 = \emptyset$. Also $wk' = \mathcal{H} + 1$ and $\xi' = \emptyset$.
  Since $wk_1 \geq 1$, we know
  $$\mathbb{ws}' <^{wk}_{\mathcal{H}+1} \mathbb{ws}.$$

Below we prove:

$$\mathcal{D}, R, G \models^{\chi}_{\mathrm{t}} (\text{while } (B)\{C\}, \sigma) \preceq (\textbf{skip}, \Sigma) \diamond (u, \mathbb{ws}', ws', aw, w, wk', \mathcal{H} + 1) \Downarrow_{\xi', \xi_a} p \wedge \neg B\ . \quad \text{(B.3)}$$

Proof: By co-induction. Suppose $\sigma = (s, h)$. Since $p \Rightarrow (B = B) * I$, we know
$$(\sigma, \Sigma) \models I * \text{true}, \text{ and either } [\![B]\!]_s = \textbf{true} \text{ or } [\![B]\!]_s = \textbf{false}\ .$$
If $[\![B]\!]_s = \textbf{true}$, since $((\sigma, \Sigma), (u, w_1), \textbf{skip}) \models p_{\mathrm{t}}$ and $p \wedge B \Rightarrow J * \text{true} \wedge \mathsf{arem}(\textbf{await}(\mathbb{B}')\{\mathbb{C}'\})$, we know $((\sigma, \Sigma), (u, w_1), \textbf{skip}) \models \mathsf{arem}(\textbf{await}(\mathbb{B}')\{\mathbb{C}'\})$, which is impossible. Thus
$$[\![B]\!]_s = \textbf{false}.$$
We only need to prove the following *(1)(2)(3)(4)(5)*.

(1)(a) $\xi' \cup \xi_a \subseteq s(\mathtt{TIDS})$ and $\mathrm{t} \notin \xi'$ and $\mathrm{t} \notin \xi_a$.

(b) For any $\mathrm{t}' \in \xi' \cup \xi_a$, we have $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathrm{t}'}) * \text{true}$.

(c) If $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathrm{t}}) * \text{true}$, then $\sigma \models \mathsf{en}(\text{while } (B)\{C\})$.

(d) If $\sigma \models \neg\mathsf{en}(\text{while } (B)\{C\})$ and $\Sigma \models \mathsf{en}(\textbf{skip})$, then $\xi_a \neq \emptyset$.

(e) If $wk' = \mathcal{H} + 1$, then $\xi' = \emptyset$. If $wk' < \mathcal{H} + 1$, then $\xi' \neq \emptyset \vee \Sigma' \models \neg\mathsf{en}(\textbf{skip})$.

(f) $|\mathbb{ws}'| \leq \mathcal{H} + 1$ and $(1 \leq wk' \leq |\mathbb{ws}'| - 1) \vee (wk' = \mathcal{H} + 1)$.

Proof: Immediate from $\mathcal{D}' \Subset \mathcal{D}$.

(2) If $\text{while } (B)\{C\} = \textbf{skip}$, then ...

Proof: It is vacantly true.

(3) For any σ_F, $(\text{while } (B)\{C\}, \sigma \uplus \sigma_F) \not\longrightarrow_{\mathrm{t}} \textbf{abort}$.

Proof: It holds because $[\![B]\!]_s$ is not undefined.

(4) If $(\text{while } (B)\{C\}, \sigma \uplus \sigma_F) \longrightarrow_{\mathsf{t}} (C', \sigma'')$, then there exist σ', k, u', $\mathbb{ws}''$, ws'', aw', w', wk'', ξ'' and ξ'_a such that
   (a) $\sigma'' = \sigma' \uplus \sigma_F$, and
   (b) $\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (C', \sigma') \preceq (\mathbf{skip}, \Sigma) \diamond (u', \mathbb{ws}'', ws'', aw', w', wk'', \mathcal{H}+1) \Downarrow_{\xi'', \xi'_a} p \wedge \neg B$, and
   (c) $((\sigma, \Sigma), (\sigma', \Sigma), k) \models G_{\mathsf{t}} * \mathsf{True}$, and
   (d) either $u' <_k u$ and $k > 0$,
   or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}'' <^{wk'}_{\mathcal{H}+1} \mathbb{ws}'$,
   or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}'' \approx^{wk'}_{\mathcal{H}+1} \mathbb{ws}'$ and $wk'' < wk'$,
   or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}'' \approx^{wk'}_{\mathcal{H}+1} \mathbb{ws}'$ and $wk'' = wk' < \mathcal{H}+1$ and $\xi' \subseteq \xi''$; and
   (e) if $((\sigma, \Sigma), (\sigma', \Sigma)) \models \langle [\mathcal{D}_{\mathsf{t}}] \rangle * \mathsf{True}$ and $k = 0$, then $ws'' <_{\mathcal{H}+1} ws'$.

   Proof: Since $[\![B]\!]_s = \mathbf{false}$, we know $(\text{while } (B)\{C\}, \sigma \uplus \sigma_F) \longrightarrow_{\mathsf{t}} (\mathbf{skip}, \sigma \uplus \sigma_F)$. By (SKIP) rule, let

$$\mathbb{ws}'' = ((0,0),0) \;\text{ and }\; ws'' = (0,0) \;\text{ and }\; wk'' = \mathcal{H}+1 \;\text{ and }\; \xi'' = \emptyset,$$

   we know

$$\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (\mathbf{skip}, \sigma) \preceq (\mathbf{skip}, \Sigma) \diamond (u, \mathbb{ws}'', ws'', aw, w, wk'', \mathcal{H}+1) \Downarrow_{\xi'', \xi_a} p \wedge \neg B\,.$$

   Also we have $ws'' <_{\mathcal{H}+1} ws'$ and $\mathbb{ws}'' <^{wk'}_{\mathcal{H}+1} \mathbb{ws}'$. Since $(\sigma, \Sigma) \models I * \mathsf{true}$, we can prove

$$((\sigma, \Sigma), (\sigma, \Sigma), 0) \models G_{\mathsf{t}} * \mathsf{True}\,.$$

(5) If $((\sigma, \Sigma), (\sigma', \Sigma'), k) \models R_{\mathsf{t}} * \mathsf{Id}$, then there exist u', $\mathbb{ws}''$, ws'', aw', w', wk'', ξ_d, ξ'' and ξ'_a such that
   (a) $\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (\text{while } (B)\{C\}, \sigma') \preceq (\mathbf{skip}, \Sigma') \diamond (u', \mathbb{ws}'', ws'', aw', w', wk'', \mathcal{H}+1) \Downarrow_{\xi'', \xi'_a} p \wedge \neg B$, and
   (b) $u' \approx_k u$, and
   $k = 0 \Longrightarrow w' = w$, and
   (c) $\xi_d = \{\mathsf{t}' \mid (\mathsf{t}' \in \xi') \wedge (((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle \mathcal{D}_{\mathsf{t}'} \rangle * \mathsf{Id})\}$ and
   $k = 0 \Longrightarrow \mathbb{ws}'' <^{wk'}_{\mathcal{H}+1} \mathbb{ws}' \vee \mathbb{ws}'' \approx^{wk'}_{\mathcal{H}+1} \mathbb{ws}'$ and
   $k = 0 \wedge wk' < \mathcal{H}+1 \wedge \xi_d \neq \emptyset \Longrightarrow \mathbb{ws}'' <^{wk'}_{\mathcal{H}+1} \mathbb{ws}'$ and
   $k = 0 \wedge \xi' \backslash \xi_d \neq \emptyset \Longrightarrow wk'' \leq wk'$, and
   $k = 0 \wedge wk'' = wk' \Longrightarrow \xi' \backslash \xi_d \subseteq \xi''$, and
   (d) if $k = 0$ and $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}}) * \mathsf{true}$, then $ws'' \leq_{\mathcal{H}+1} ws'$.

   Proof: Since $((\sigma, \Sigma), (u, w_1), \mathbf{skip}) \models p_{\mathsf{t}}$ and $\mathsf{Sta}(p, R * \mathsf{Id})$, we know there exist u' and w'_1 such that

$$((\sigma', \Sigma'), (u', w'_1), \mathbf{skip}) \models p_{\mathsf{t}} \;\text{ and }\; u' \approx_k u \;\text{ and }\; k = 0 \Longrightarrow w'_1 = w_1\,.$$

   Also we know

$$(\sigma', \Sigma') \models J * \mathsf{true}\,.$$

   Suppose $k'_s = f_{\mathsf{t}}(\sigma', \Sigma')$. Since $J \Rightarrow (R, G\colon \mathcal{D}' \xrightarrow{f} (Q, \mathbb{B}'))$, we can prove

$$k = 0 \Longrightarrow k'_s \leq k_s\,.$$

   Let

$$\xi''_0 = \{\mathsf{t}'' \mid (\mathsf{t}'' \neq \mathsf{t}) \wedge ((\sigma', \Sigma') \models \mathsf{Enabled}(\mathcal{D}'_{\mathsf{t}''}) * \mathsf{true})\} \;\text{ and}$$
$$\xi_d = \{\mathsf{t}' \mid (\mathsf{t}' \in \xi') \wedge (((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle \mathcal{D}_{\mathsf{t}'} \rangle * \mathsf{Id})\}.$$

   Since $\mathsf{Enabled}(\mathcal{D}) \Rightarrow I$, $\mathcal{D}' \Subset \mathcal{D}$ and $\mathsf{wffAct}(R, \mathcal{D}')$, we can prove:

$$k = 0 \Longrightarrow \xi' \backslash \xi_d \subseteq \xi''_0\,.$$

   If $w'_1 = w_1$, let $w' = w$; otherwise let $w' = w'_1$. Thus we know $w'_1 \leq w'$. Also we know: if $k = 0$, then $w' = w$. Let

$$ws'' = (0,0)::\mathsf{inchead}(ws_1,(w_1',0)) = (0,0)::(w_1',0) \text{ and}$$
$$\mathbb{ws}'' = ((0,0),0)::\mathsf{inchead}(\mathbb{ws}_1,((k_s',w_1'),0)) = ((0,0),0)::((k_s',w_1'),0)\ .$$

Thus if $k = 0$, then $ws'' = ws'$, and

$$\mathbb{ws}'' <^{wk'}_{\mathcal{H}+1} \mathbb{ws}' \quad \text{or} \quad \mathbb{ws}'' \approx^{wk'}_{\mathcal{H}+1} \mathbb{ws}'.$$

If $\xi_0'' \neq \emptyset$, let $wk'' = 1$ and $\xi'' = \xi_0''$; otherwise let $wk'' = \mathcal{H}+1$ and $\xi'' = \emptyset$.

By the co-induction hypothesis, we know

$$\mathcal{D},R,G \models^{\chi}_{\mathrm{t}} (\text{while } (B)\{C\},\sigma') \preceq (\mathbf{skip},\Sigma') \diamond (u,\mathbb{ws}'',ws'',aw,w',wk'',\mathcal{H}+1) \Downarrow_{\xi'',\xi_a} p \wedge \neg B\ .$$

Suppose $k = 0$. If $wk' < \mathcal{H}+1$, then $wk' = 1$. If $\xi_d \neq \emptyset$, then there exists t' such that $\mathrm{t}' \in \xi'$ and $((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle \mathcal{D}_{\mathrm{t}'} \rangle * \mathsf{Id}$. Since $\mathcal{D}' \Subset \mathcal{D}$, we can prove

$$((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle \mathcal{D}'_{\mathrm{t}'} \rangle * \mathsf{Id}\ .$$

Since $J \Rightarrow (R,G\colon \mathcal{D}' \xrightarrow{f} (Q,\mathbb{B}'))$, we know for any $\mathrm{t}' \neq \mathrm{t}$, σ' and Σ', if $((\sigma,\Sigma),(\sigma',\Sigma'),0) \models (\langle \mathcal{D}'_{\mathrm{t}'} \rangle \wedge R_{\mathrm{t}}) * \mathsf{Id}$, then $f_{\mathrm{t}}(\sigma',\Sigma') < k_s$. Thus we can prove:

$$k_s' < k_s\ .$$

Thus $\mathbb{ws}'' <^{wk'}_{\mathcal{H}+1} \mathbb{ws}'$ holds.

If $k = 0 \wedge \xi' \backslash \xi_d \neq \emptyset$, we know $wk'' = 1$. Thus $wk'' \leq wk'$.

If $k = 0 \wedge wk'' = wk'$, we know $\xi' \backslash \xi_d \subseteq \xi''$.

Thus we have proved (B.3).

(5) If $((\sigma,\Sigma),(\sigma',\Sigma'),k) \models R_{\mathrm{t}} * \mathsf{Id}$, then there exist u', $\mathbb{ws}'$, ws', aw', w', wk', ξ_d, ξ_{ad}, ξ' and ξ_a' such that

(a) $\mathcal{D},R,G \models^{\chi}_{\mathrm{t}} (C_1;\text{while } (B)\{C\},\sigma') \preceq (\mathbf{skip},\Sigma') \diamond (u',\mathbb{ws}',ws',aw',w',wk',\mathcal{H}+1) \Downarrow_{\xi',\xi_a'} p \wedge \neg B$, and

(b) $u' \approx_k u$, and
$k = 0 \Longrightarrow w' = w$, and

(c) $\xi_d = \{\mathrm{t}' \mid (\mathrm{t}' \in \xi) \wedge (((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle \mathcal{D}_{\mathrm{t}'} \rangle * \mathsf{Id})\}$ and
$k = 0 \Longrightarrow \mathbb{ws}' <^{wk}_{\mathcal{H}+1} \mathbb{ws} \vee \mathbb{ws}' \approx^{wk}_{\mathcal{H}+1} \mathbb{ws}$ and
$k = 0 \wedge wk < \mathcal{H}+1 \wedge \xi_d \neq \emptyset \Longrightarrow \mathbb{ws}' <^{wk}_{\mathcal{H}+1} \mathbb{ws}$ and
$k = 0 \wedge \xi \backslash \xi_d \neq \emptyset \Longrightarrow wk' \leq wk$, and
$k = 0 \wedge wk' = wk \Longrightarrow \xi \backslash \xi_d \subseteq \xi'$, and

(d) if $k = 0$ and $(\sigma,\Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathrm{t}}) * \mathsf{true}$, then $ws' \leq_{\mathcal{H}+1} ws$; and

(e) $\xi_{ad} = \{\mathrm{t}' \mid (\mathrm{t}' \in \xi_a) \wedge (((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle \mathcal{D}_{\mathrm{t}'} \rangle * \mathsf{Id})\}$ and
$k = 0 \wedge \mathsf{is\_await}(C_1;\text{while } (B)\{C\}) \Longrightarrow \xi_a \backslash \xi_{ad} \subseteq \xi_a'$ and
$k = 0 \wedge \mathsf{is\_await}(C_1;\text{while } (B)\{C\}) \wedge \xi_{ad} \neq \emptyset \Longrightarrow aw' < aw$ and
$(\chi = \mathsf{sfair}) \wedge k = 0 \wedge \mathsf{is\_await}(C_1;\text{while } (B)\{C\}) \wedge (\sigma \models \neg\mathsf{en}(C_1;\text{while } (B)\{C\})) \wedge (\sigma' \models \neg\mathsf{en}(C_1;\text{while } (B)\{C\})) \Longrightarrow aw' \leq aw$ and
$(\chi = \mathsf{wfair}) \wedge k = 0 \wedge \mathsf{is\_await}(C_1;\text{while } (B)\{C\}) \Longrightarrow aw' \leq aw$.

Proof: From $\mathcal{D},R,G \models^{\chi}_{\mathrm{t}} (C_1,\sigma) \preceq (\mathbf{skip},\Sigma) \diamond (u,\mathbb{ws}_1,ws_1,aw,w_1,wk_1,\mathcal{H}) \Downarrow_{\xi_1,\xi_a} p$, we know there exist u', $\mathbb{ws}_1'$, ws_1', aw', w_1', wk_1', ξ_d', ξ_{ad}, ξ_1' and ξ_a' such that

(A) $\mathcal{D},R,G \models^{\chi}_{\mathrm{t}} (C_1,\sigma') \preceq (\mathbf{skip},\Sigma') \diamond (u',\mathbb{ws}_1',ws_1',aw',w_1',wk_1',\mathcal{H}) \Downarrow_{\xi_1',\xi_a'} p$, and

(B) $u' \approx_k u$, and
$k = 0 \Longrightarrow w_1' = w_1$, and

(C) $\xi_d' = \{\mathrm{t}' \mid (\mathrm{t}' \in \xi_1) \wedge (((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle \mathcal{D}_{\mathrm{t}'} \rangle * \mathsf{Id})\}$ and
$k = 0 \Longrightarrow \mathbb{ws}_1' <^{wk_1}_{\mathcal{H}} \mathbb{ws}_1 \vee \mathbb{ws}_1' \approx^{wk_1}_{\mathcal{H}} \mathbb{ws}_1$ and
$k = 0 \wedge wk_1 < \mathcal{H} \wedge \xi_d' \neq \emptyset \Longrightarrow \mathbb{ws}_1' <^{wk_1}_{\mathcal{H}} \mathbb{ws}_1$ and
$k = 0 \wedge \xi_1 \backslash \xi_d' \neq \emptyset \Longrightarrow wk_1' \leq wk_1$, and
$k = 0 \wedge wk_1' = wk_1 \Longrightarrow \xi_1 \backslash \xi_d' \subseteq \xi_1'$, and

(D) if $k = 0$ and $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_\mathsf{t}) * \mathsf{true}$, then $ws'_1 \leq_{\mathcal{H}} ws_1$; and
(E) $\xi_{ad} = \{\mathsf{t}' \mid (\mathsf{t}' \in \xi_a) \wedge (((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle \mathcal{D}_{\mathsf{t}'} \rangle * \mathsf{Id})\}$ and
$k = 0 \wedge \mathsf{is\_await}(C_1) \Longrightarrow \xi_a \backslash \xi_{ad} \subseteq \xi'_a$ and
$k = 0 \wedge \mathsf{is\_await}(C_1) \wedge \xi_{ad} \neq \emptyset \Longrightarrow aw' < aw$ and
$(\chi = \mathsf{sfair}) \wedge k = 0 \wedge \mathsf{is\_await}(C_1) \wedge (\sigma \models \neg\mathsf{en}(C_1)) \wedge (\sigma' \models \neg\mathsf{en}(C_1)) \Longrightarrow aw' \leq aw$ and
$(\chi = \mathsf{wfair}) \wedge k = 0 \wedge \mathsf{is\_await}(C_1) \Longrightarrow aw' \leq aw$.

Since $(\sigma,\Sigma) \models J * \mathsf{true}$ and $\mathsf{Sta}(J, G \vee R)$, we know

$$(\sigma',\Sigma') \models J * \mathsf{true}\ .$$

Suppose $k'_s = f_\mathsf{t}(\sigma',\Sigma')$. Since $J \Rightarrow (R, G\colon \mathcal{D}' \xrightarrow{f} (Q, \mathbb{B}'))$, we can prove

$$k = 0 \Longrightarrow k'_s \leq k_s\ .$$

Let

$$\xi'_0 = \{\mathsf{t}'' \mid (\mathsf{t}'' \neq \mathsf{t}) \wedge ((\sigma',\Sigma') \models \mathsf{Enabled}(\mathcal{D}'_{\mathsf{t}''}) * \mathsf{true})\},$$
$$\xi_d = \{\mathsf{t}' \mid (\mathsf{t}' \in \xi) \wedge (((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle \mathcal{D}_{\mathsf{t}'} \rangle * \mathsf{Id})\} \text{ and}$$
$$\xi''_d = \{\mathsf{t}' \mid (\mathsf{t}' \in \xi_0) \wedge (((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle \mathcal{D}_{\mathsf{t}'} \rangle * \mathsf{Id})\}.$$

Let

$$ws' = (0,0) :: \mathsf{inchead}(ws'_1, (w_1, 1))\ ,\ \mathbb{ws}' = ((0,0),0) :: \mathsf{inchead}(\mathbb{ws}'_1, ((k'_s, w'_1), 1))\ .$$

If $w'_1 = w_1$, let $w' = w$; otherwise let $w' = w'_1$. Thus we know $w'_1 \leq w'$.
Also we know: if $k = 0$, then $w' = w$, and if $ws'_1 \leq_{\mathcal{H}} ws_1$ then $ws' \leq_{\mathcal{H}+1} ws$.
If $\xi'_0 \neq \emptyset$, let $wk' = 1$ and $\xi' = \xi'_0$; otherwise let $wk' = wk'_1 + 1$ and $\xi' = \xi'_1$. Then, by the co-induction hypothesis, we know

$$\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (C'_1; \mathsf{while}\ (B)\{C\}, \sigma') \leq (\mathbb{C}, \Sigma') \diamond (u', \mathbb{ws}', ws', aw', w', wk', \mathcal{H}+1) \Downarrow_{\xi',\xi'_a} p \wedge \neg B\ .$$

One of the following holds:

- If $\xi_0 \neq \emptyset$, then $wk = 1$ and $\xi = \xi_0$.
  Thus $\xi_d = \xi''_d$. Since $\mathsf{Enabled}(\mathcal{D}) \Rightarrow I$, $\mathcal{D}' \Subset \mathcal{D}$ and $\mathsf{wffAct}(R, \mathcal{D}')$, we can prove:
  $$k = 0 \implies \xi_0 \backslash \xi''_d \subseteq \xi'_0\ .$$
  If $k = 0$, we know $\mathbb{ws}'_1 <^{wk_1}_{\mathcal{H}} \mathbb{ws}_1$ or $\mathbb{ws}'_1 \approx^{wk_1}_{\mathcal{H}} \mathbb{ws}_1$. Since $wk_1 \geq 1$, we know
  $$\mathbb{ws}' <^{wk}_{\mathcal{H}+1} \mathbb{ws} \quad \text{or} \quad \mathbb{ws}' \approx^{wk}_{\mathcal{H}+1} \mathbb{ws}.$$
  Suppose $k = 0$. If $\xi_d \neq \emptyset$, then there exists t' such that $\mathsf{t}' \in \xi$ and $((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle \mathcal{D}_{\mathsf{t}'} \rangle * \mathsf{Id}$. Since $\mathcal{D}' \Subset \mathcal{D}$, we can prove
  $$((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle \mathcal{D}'_{\mathsf{t}'} \rangle * \mathsf{Id}\ .$$
  Since $J \Rightarrow (R, G\colon \mathcal{D}' \xrightarrow{f} (Q, \mathbb{B}'))$, we know
  $$k'_s < k_s\ .$$
  Thus $\mathbb{ws}' <^{wk}_{\mathcal{H}+1} \mathbb{ws}$ holds.
  If $k = 0 \wedge \xi \backslash \xi_d \neq \emptyset$, we know $wk' = 1$. Thus $wk' \leq wk$.
  If $k = 0 \wedge wk' = wk$, we know $\xi \backslash \xi_d \subseteq \xi'$.
- If $\xi_0 = \emptyset$, then $wk = wk_1 + 1$ and $\xi = \xi_1$.
  Thus $\xi_d = \xi'_d$. If $k = 0$, we know $\mathbb{ws}'_1 <^{wk_1}_{\mathcal{H}} \mathbb{ws}_1$ or $\mathbb{ws}'_1 \approx^{wk_1}_{\mathcal{H}} \mathbb{ws}_1$. Thus we know
  $$\mathbb{ws}' <^{wk}_{\mathcal{H}+1} \mathbb{ws} \quad \text{or} \quad \mathbb{ws}' \approx^{wk}_{\mathcal{H}+1} \mathbb{ws}.$$
  If $k = 0 \wedge wk < \mathcal{H} + 1 \wedge \xi_d \neq \emptyset$, we know $k = 0 \wedge wk_1 < \mathcal{H} \wedge \xi'_d \neq \emptyset$. Thus $\mathbb{ws}'_1 <^{wk_1}_{\mathcal{H}} \mathbb{ws}_1$.
  Thus $\mathbb{ws}' <^{wk}_{\mathcal{H}+1} \mathbb{ws}$.
  If $k = 0 \wedge \xi \backslash \xi_d \neq \emptyset$, we know $wk'_1 \leq wk_1$. Thus $wk' \leq wk$.
  If $k = 0 \wedge wk' = wk$, we know $wk' = wk'_1 + 1$, $wk'_1 = wk_1$ and $\xi' = \xi'_1$. Thus $\xi \backslash \xi_d \subseteq \xi'$.

Thus we are done. □

B.4.2 The AWAIT-W rule.

LEMMA B.12 (AWAIT-W-SOUND). *If*

(1) $\mathcal{D}, [I], G \models_{\mathsf{wfair}} \{p \wedge B\}\langle C\rangle\{q\}$*;*
(2) $p \wedge \mathsf{Enabled}(\mathcal{D}) * \mathsf{true} \Rightarrow B$*;* $p \Rightarrow (B = B)$*;*
(3) $\mathsf{Sta}(\{p, q\}, R * \mathsf{Id})$*;*
(4) $\mathcal{D}' \preccurlyeq \mathcal{D}$*;* $\mathit{wffAct}(R, \mathcal{D}')$*;* $p \Rightarrow \exists \mathbb{B}', \mathbb{C}'.\, \mathsf{arem}(\mathbf{await}(\mathbb{B}')\{\mathbb{C}'\}) \wedge (R\colon \mathcal{D}' \overset{f}{\hookrightarrow} (B, \mathbb{B}'))$*;*

then $\mathcal{D}, R, G \models_{\mathsf{wfair}} \{p\}\mathbf{await}(B)\{C\}\{q\}$.

PROOF. Let $\mathcal{H} = \mathsf{height}(\mathbf{await}(B)\{C\}) = 1$. We know $|\mathbf{await}(B)\{C\}| = 1$. Let

$$\mathbb{ws} = ((0,0),1) \;\text{ and }\; ws = (0,1) \;\text{ and }\; wk = \mathcal{H} = 1 \;\text{ and }\; \xi = \emptyset.$$

For any t, for any σ, Σ, u, w and $\mathbb{C}$, if $((\sigma, \Sigma), (u, w), \mathbb{C}) \models p_{\mathsf{t}}$, then let

$$aw = f_{\mathsf{t}}(\sigma, \Sigma) \;\text{ and }\; \xi_a = \{\mathsf{t}' \mid (\mathsf{t}' \neq \mathsf{t}) \wedge ((\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}'_{\mathsf{t}'}) * \mathsf{true})\}\,.$$

Below we prove:

$$\mathcal{D}, R, G \models_{\mathsf{t}}^{\mathsf{wfair}} (\mathbf{await}(B)\{C\}, \sigma) \preceq (\mathbb{C}, \Sigma) \diamond (u, \mathbb{ws}, ws, aw, w, wk, \mathcal{H}) \Downarrow_{\xi, \xi_a} q\,.$$

By co-induction. Since $p \Rightarrow \exists \mathbb{B}', \mathbb{C}'.\, \mathsf{arem}(\mathbf{await}(\mathbb{B}')\{\mathbb{C}'\}) \wedge (R\colon \mathcal{D}' \overset{f}{\hookrightarrow} (B, \mathbb{B}'))$, we know there exist B' and C' such that

$$\mathbb{C} = \mathbf{await}(\mathbb{B}')\{\mathbb{C}'\} \;\text{ and }\; (\sigma, \Sigma) \models (R\colon \mathcal{D}' \overset{f}{\hookrightarrow} (B, \mathbb{B}')).$$

Suppose $\sigma = (s, h)$. We only need to prove the following (1)(2)(3)(4)(5)(6).

(1)(a) $\xi \cup \xi_a \subseteq s(\mathtt{TIDS})$ and $\mathsf{t} \notin \xi$ and $\mathsf{t} \notin \xi_a$.
(b) For any $\mathsf{t}' \in \xi \cup \xi_a$, we have $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}'}) * \mathsf{true}$.
(c) If $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}}) * \mathsf{true}$, then $\sigma \models \mathsf{en}(\mathbf{await}(B)\{C\})$.
(d) If $\sigma \models \neg\mathsf{en}(\mathbf{await}(B)\{C\})$ and $\Sigma \models \mathsf{en}(\mathbb{C})$, then $\xi_a \neq \emptyset$.
(e) If $wk = \mathcal{H}$, then $\xi = \emptyset$. If $wk < \mathcal{H}$, then $\xi \neq \emptyset \vee \Sigma \models \neg\mathsf{en}(\mathbb{C})$.
(f) $|\mathbb{ws}| \leq \mathcal{H}$ and $(1 \leq wk \leq |\mathbb{ws}| - 1) \vee (wk = \mathcal{H})$.
Proof: (a), (e) and (f) are immediate.
(b) Immediate from $\mathcal{D}' \preccurlyeq \mathcal{D}$.
(c) Immediate from $p \wedge \mathsf{Enabled}(\mathcal{D}) * \mathsf{true} \Rightarrow B$.
(d) From $(\sigma, \Sigma) \models (R\colon \mathcal{D}' \overset{f}{\hookrightarrow} (B, \mathbb{B}'))$, we know: either $\sigma \models B$, or $\Sigma \models \neg\mathbb{B}'$, or $\exists \mathsf{t}' \neq \mathsf{t}.\, (\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}'_{\mathsf{t}'}) * \mathsf{true}$. Thus we are done.

(2) If $\mathbf{await}(B)\{C\} = \mathbf{skip}$, then ....
Proof: It is vacantly true.

(3) If $\mathbf{await}(B)\{C\} = \mathbf{E}[\,\mathbf{return}\ E\,]$, then ....
Proof: It is vacantly true.

(4) For any σ_F, $(\mathbf{await}(B)\{C\}, \sigma \uplus \sigma_F) \not\longrightarrow_{\mathsf{t}} \mathbf{abort}$.
Proof: By the operational semantics and $\mathcal{D}, [I], G \models_{\mathsf{wfair}} \{p \wedge B\}\langle C\rangle\{q\}$.

(5) For any C', σ'', σ_F and Σ_F, if $(\mathbf{await}(B)\{C\}, \sigma \uplus \sigma_F) \longrightarrow_{\mathsf{t}} (C', \sigma'')$, then there exist σ', $\mathbb{C}'$, Σ', k, u', $\mathbb{ws}'$, ws', aw', w', wk', ξ' and ξ'_a such that
(a) $\sigma'' = \sigma' \uplus \sigma_F$, and
(b) $(\mathbb{C}, \Sigma \uplus \Sigma_F) \longrightarrow_{\mathsf{t}}^{*} (\mathbb{C}', \Sigma' \uplus \Sigma_F)$, and
(c) $\mathcal{D}, R, G \models_{\mathsf{t}}^{\mathsf{wfair}} (C', \sigma') \preceq (\mathbb{C}', \Sigma') \diamond (u', \mathbb{ws}', ws', aw', w', wk', \mathcal{H}) \Downarrow_{\xi', \xi'_a} q$, and
(d) $((\sigma, \Sigma), (\sigma', \Sigma'), k) \models G_{\mathsf{t}} * \mathsf{True}$, and

(e) either $u' <_k u$ and $k > 0$,
or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}' <^{wk}_{\mathcal{H}} \mathbb{ws}$,
or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}' \approx^{wk}_{\mathcal{H}} \mathbb{ws}$ and $wk' < wk$,
or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}' \approx^{wk}_{\mathcal{H}} \mathbb{ws}$ and $wk' = wk < \mathcal{H}$ and $\xi \subseteq \xi'$; and
(f) if $((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle[\mathcal{D}_t]\rangle * \mathsf{True}$ and $k = 0$, then $ws' <_{\mathcal{H}} ws$.

Proof: By the operational semantics, we know C' must be **skip** and

$$[\![B]\!]_s = \mathbf{true} \;\text{ and }\; (C, \sigma \uplus \sigma_F) \longrightarrow_t^* (\mathbf{skip}, \sigma'') \,.$$

Thus

$$(\langle C\rangle, \sigma \uplus \sigma_F) \longrightarrow_t^* (\mathbf{skip}, \sigma'') \,.$$

From $\mathcal{D}, [I], G \models_{\mathsf{wfair}} \{p \wedge B\}\langle C\rangle\{q\}$, we know there exist aw'' and ξ''_a such that

$$\mathcal{D}, [I], G \models^{\mathsf{wfair}}_t (\langle C\rangle, \sigma) \preceq (\mathbb{C}, \Sigma) \diamond (u, \mathbb{ws}, ws, aw'', w, wk, \mathcal{H}) \Downarrow_{\xi,\xi''_a} q \,.$$

Thus there exist σ', $\mathbb{C}'$, Σ', k, u', $\mathbb{ws}'$, ws', aw', w', wk', ξ' and ξ'_a such that
(A) $\sigma'' = \sigma' \uplus \sigma_F$, and
(B) $(\mathbb{C}, \Sigma \uplus \Sigma_F) \longrightarrow_t^* (\mathbb{C}', \Sigma' \uplus \Sigma_F)$, and
(C) $\mathcal{D}, [I], G \models^{\mathsf{wfair}}_t (\mathbf{skip}, \sigma') \preceq (\mathbb{C}', \Sigma') \diamond (u', \mathbb{ws}', ws', aw', w', wk', \mathcal{H}) \Downarrow_{\xi',\xi'_a} q$, and
(D) $((\sigma,\Sigma),(\sigma',\Sigma'),k) \models G_t * \mathsf{True}$, and
(E) either $u' <_k u$ and $k > 0$,
or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}' <^{wk}_{\mathcal{H}} \mathbb{ws}$,
or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}' \approx^{wk}_{\mathcal{H}} \mathbb{ws}$ and $wk' < wk$,
or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}' \approx^{wk}_{\mathcal{H}} \mathbb{ws}$ and $wk' = wk < \mathcal{H}$ and $\xi \subseteq \xi'$; and
(F) if $((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle[\mathcal{D}_t]\rangle * \mathsf{True}$ and $k = 0$, then $ws' <_{\mathcal{H}} ws$.
From $\mathcal{D}, [I], G \models^{\mathsf{wfair}}_t (\mathbf{skip}, \sigma') \preceq (\mathbb{C}', \Sigma') \diamond (u', \mathbb{ws}', ws', aw', w', wk', \mathcal{H}) \Downarrow_{\xi',\xi'_a} q$, we know there exist $\mathbb{C}''$ and Σ'' such that
(G) $(\mathbb{C}', \Sigma' \uplus \Sigma_F) \longrightarrow_t^* (\mathbb{C}'', \Sigma'' \uplus \Sigma_F)$, and
(H) $((\sigma',\Sigma''),(u',w'),\mathbb{C}'') \models q_t$, and
(I) $\mathbb{ws}' = ((0,0),0)$ and $ws' = (0,0)$ and $wk' = \mathcal{H}$ and $\xi' = \emptyset$, and
(J) $((\sigma',\Sigma''),(\sigma',\Sigma''),0) \models G_t * \mathsf{True}$.
Since $\mathsf{Sta}(q, R * \mathsf{Id})$, by (SKIP) rule, we can prove

$$\mathcal{D}, R, G \models^{\mathsf{wfair}}_t (\mathbf{skip}, \sigma') \preceq (\mathbb{C}'', \Sigma'') \diamond (u', \mathbb{ws}', ws', aw', w', wk', \mathcal{H}) \Downarrow_{\xi',\xi'_a} q \,.$$

Also we know

$$(\mathbb{C}, \Sigma \uplus \Sigma_F) \longrightarrow_t^* (\mathbb{C}'', \Sigma'' \uplus \Sigma_F) \;\text{ and }\; ws' <_{\mathcal{H}} ws \;\text{ and}$$
$$((\sigma,\Sigma),(\sigma',\Sigma''),k) \models G_t * \mathsf{True} \text{ (suppose } G \text{ is transitive-closed)}$$

(6) If $((\sigma,\Sigma),(\sigma',\Sigma'),k) \models R * \mathsf{Id}$, then there exist u', $\mathbb{ws}'$, ws', aw', w', wk', ξ_d, ξ_{ad}, ξ' and ξ'_a such that
(a) $\mathcal{D}, R, G \models^{\mathsf{wfair}}_t (\mathbf{await}(B)\{C\}, \sigma') \preceq (\mathbb{C}, \Sigma') \diamond (u', \mathbb{ws}', ws', aw', w', wk', \mathcal{H}) \Downarrow_{\xi',\xi'_a} q$, and
(b) $u' \approx_k u$, and
$k = 0 \Longrightarrow w' = w$, and
(c) $\xi_d = \{t' \mid (t' \in \xi) \wedge (((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle\mathcal{D}_{t'}\rangle * \mathsf{Id})\}$ and
$k = 0 \Longrightarrow \mathbb{ws}' <^{wk}_{\mathcal{H}} \mathbb{ws} \vee \mathbb{ws}' \approx^{wk}_{\mathcal{H}} \mathbb{ws}$ and
$k = 0 \wedge wk < \mathcal{H} \wedge (\xi_d \neq \emptyset \vee (\Sigma \models \neg\mathsf{en}(\mathbb{C}) \wedge \Sigma' \models \mathsf{en}(\mathbb{C}))) \Longrightarrow \mathbb{ws}' <^{wk}_{\mathcal{H}} \mathbb{ws}$ and
$k = 0 \wedge (\xi \backslash \xi_d \neq \emptyset \vee \Sigma' \models \neg\mathsf{en}(\mathbb{C})) \Longrightarrow wk' \leq wk$, and
$k = 0 \wedge wk' = wk \Longrightarrow \xi \backslash \xi_d \subseteq \xi'$, and
(d) if $k = 0$ and $(\sigma,\Sigma) \models \mathsf{Enabled}(\mathcal{D}_t) * \mathsf{true}$, then $ws' \leq_{\mathcal{H}} ws$; and
(e) $\xi_{ad} = \{t' \mid (t' \in \xi_a) \wedge (((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle\mathcal{D}_{t'}\rangle * \mathsf{Id})\}$ and
$k = 0 \wedge \mathsf{is\_await}(\mathbf{await}(B)\{C\}) \Longrightarrow \xi_a \backslash \xi_{ad} \subseteq \xi'_a$ and

$k = 0 \wedge \mathsf{is\_await}(\mathbf{await}(B)\{C\}) \wedge (\xi_{ad} \neq \emptyset \vee (\Sigma \models \neg\mathsf{en}(\mathbb{C}) \wedge \Sigma' \models \mathsf{en}(\mathbb{C}))) \Longrightarrow aw' < aw$
and
$k = 0 \wedge \mathsf{is\_await}(\mathbf{await}(B)\{C\}) \Longrightarrow aw' \leq aw.$

Proof: Since $\mathsf{Sta}(p, R * \mathsf{Id})$, we know there exist u' and w' such that

$$((\sigma', \Sigma'), (u', w'), \mathbb{C}) \models p_{\mathsf{t}} \;\text{ and }\; u' \approx_k u \;\text{ and }\; k = 0 \implies w' = w\,.$$

Let

$$aw' = f_{\mathsf{t}}(\sigma', \Sigma') \;\text{ and }\; \xi'_a = \{\mathsf{t}' \mid (\mathsf{t}' \neq \mathsf{t}) \wedge ((\sigma', \Sigma') \models \mathsf{Enabled}(\mathcal{D}'_{\mathsf{t}'}) * \mathsf{true})\}\,.$$

By the co-induction hypothesis, we know

$$\mathcal{D}, R, G \models^{\mathsf{wfair}}_{\mathsf{t}} (\mathbf{await}(B)\{C\}, \sigma') \preceq (\mathbb{C}, \Sigma') \diamond (u', \mathbb{ws}, ws, aw', w', wk, \mathcal{H}) \Downarrow_{\xi, \xi'_a} q\,.$$

Let

$$\xi_{ad} = \{\mathsf{t}' \mid (\mathsf{t}' \in \xi_a) \wedge (((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle \mathcal{D}_{\mathsf{t}'} \rangle * \mathsf{Id})\}.$$

Since $\mathsf{Enabled}(\mathcal{D}) \Rightarrow I$, $\mathcal{D}' \Subset \mathcal{D}$ and $\mathsf{wffAct}(R, \mathcal{D}')$, we can prove:

$$k = 0 \implies \xi_a \backslash \xi_{ad} \subseteq \xi'_a\,.$$

From $(\sigma, \Sigma) \models (R\colon \mathcal{D}' \overset{f}{\circ\!\!\rightarrow} (B, \mathbb{B}'))$, we know

$$k = 0 \Longrightarrow aw' \leq aw\,.$$

Suppose $k = 0$. If $\xi_{ad} \neq \emptyset$, then there exists t' such that $\mathsf{t}' \in \xi_a$ and $((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle \mathcal{D}_{\mathsf{t}'} \rangle * \mathsf{Id}$. Since $\mathcal{D}' \Subset \mathcal{D}$, we can prove

$$((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle \mathcal{D}'_{\mathsf{t}'} \rangle * \mathsf{Id}\,.$$

Since $(\sigma, \Sigma) \models (R\colon \mathcal{D}' \overset{f}{\circ\!\!\rightarrow} (B, \mathbb{B}'))$, we know

$$aw' < aw\,.$$

Also if $\Sigma \models \neg\mathsf{en}(\mathbb{C}) \wedge \Sigma' \models \mathsf{en}(\mathbb{C})$, from $(\sigma, \Sigma) \models (R\colon \mathcal{D}' \overset{f}{\circ\!\!\rightarrow} (B, \mathbb{B}'))$, we can still prove:

$$aw' < aw\,.$$

Thus we are done. □

B.4.3 *The* AWAIT-S *rule.*

LEMMA B.13 (AWAIT-S-SOUND). *If*
(1) $\mathcal{D}, [I], G \models_{\mathsf{sfair}} \{p \wedge B\}\langle C\rangle\{q\}$*;*
(2) $p \wedge \mathsf{Enabled}(\mathcal{D}) * \mathsf{true} \Rightarrow B$*;* $p \Rightarrow (B = B)$*;*
(3) $\mathsf{Sta}(\{p, q\}, R * \mathsf{Id})$*;*
(4) $\mathcal{D}' \Subset \mathcal{D}$*;* $\mathit{wffAct}(R, \mathcal{D}')$*;* $p \Rightarrow \exists \mathbb{B}', \mathbb{C}'.\, \mathsf{arem}(\mathbf{await}(\mathbb{B}')\{\mathbb{C}'\}) \wedge (R\colon \mathcal{D}' \overset{f}{\bullet\!\!\rightarrow} (B, \mathbb{B}'))$*;*
then $\mathcal{D}, R, G \models_{\mathsf{sfair}} \{p\}\mathbf{await}(B)\{C\}\{q\}$.

PROOF. Let $\mathcal{H} = \mathsf{height}(\mathbf{await}(B)\{C\}) = 1$. We know $|\mathbf{await}(B)\{C\}| = 1$. Let

$$\mathbb{ws} = ((0,0),1) \;\text{ and }\; ws = (0,1) \;\text{ and }\; wk = \mathcal{H} = 1 \;\text{ and }\; \xi = \emptyset.$$

For any t, for any σ, Σ, u, w and $\mathbb{C}$, if $((\sigma, \Sigma), (u, w), \mathbb{C}) \models p_{\mathsf{t}}$, then let

$$aw = f_{\mathsf{t}}(\sigma, \Sigma) \;\text{ and }\; \xi_a = \{\mathsf{t}' \mid (\mathsf{t}' \neq \mathsf{t}) \wedge ((\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}'_{\mathsf{t}'}) * \mathsf{true})\}\,.$$

Below we prove:

$$\mathcal{D}, R, G \models^{\mathsf{sfair}}_{\mathsf{t}} (\mathbf{await}(B)\{C\}, \sigma) \preceq (\mathbb{C}, \Sigma) \diamond (u, \mathbb{ws}, ws, aw, w, wk, \mathcal{H}) \Downarrow_{\xi, \xi_a} q\,.$$

By co-induction. Since $p \Rightarrow \exists \mathbb{B}', \mathbb{C}'.\, \mathsf{arem}(\mathbf{await}(\mathbb{B}')\{\mathbb{C}'\}) \wedge (R\colon \mathcal{D}' \overset{f}{\bullet\!\!\rightarrow} (B, \mathbb{B}'))$, we know there exist B' and C' such that

$$\mathbb{C} = \mathbf{await}(\mathbb{B}')\{\mathbb{C}'\} \text{ and } (\sigma,\Sigma) \models (R\colon \mathcal{D}' \overset{f}{\bullet\!\!\longrightarrow} (B,\mathbb{B}')).$$

Suppose $\sigma = (s,h)$. We only need to prove the following (1)(2)(3)(4)(5)(6).

(1)(a) $\xi \cup \xi_a \subseteq s(\mathtt{TIDS})$ and $\mathsf{t} \notin \xi$ and $\mathsf{t} \notin \xi_a$.
(b) For any $\mathsf{t}' \in \xi \cup \xi_a$, we have $(\sigma,\Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}'}) * \mathsf{true}$.
(c) If $(\sigma,\Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}}) * \mathsf{true}$, then $\sigma \models \mathsf{en}(\mathbf{await}(B)\{C\})$.
(d) If $\sigma \models \neg\mathsf{en}(\mathbf{await}(B)\{C\})$ and $\Sigma \models \mathsf{en}(\mathbb{C})$, then $\xi_a \neq \emptyset$.
(e) If $wk = \mathcal{H}$, then $\xi = \emptyset$. If $wk < \mathcal{H}$, then $\xi \neq \emptyset \vee \Sigma \models \neg\mathsf{en}(\mathbb{C})$.
(f) $|\mathbb{ws}| \leq \mathcal{H}$ and $(1 \leq wk \leq |\mathbb{ws}| - 1) \vee (wk = \mathcal{H})$.
Proof: (a), (e) and (f) are immediate.
(b) Immediate from $\mathcal{D}' \Subset \mathcal{D}$.
(c) Immediate from $p \wedge \mathsf{Enabled}(\mathcal{D}) * \mathsf{true} \Rightarrow B$.
(d) From $(\sigma,\Sigma) \models (R\colon \mathcal{D}' \overset{f}{\bullet\!\!\longrightarrow} (B,\mathbb{B}'))$, we know: either $\sigma \models B$, or $\Sigma \models \neg\mathbb{B}'$, or $\exists \mathsf{t}' \neq \mathsf{t}.\ (\sigma,\Sigma) \models \mathsf{Enabled}(\mathcal{D}'_{\mathsf{t}'}) * \mathsf{true}$. Thus we are done.
(2) If $\mathbf{await}(B)\{C\} = \mathbf{skip}$, then ....
Proof: It is vacantly true.
(3) If $\mathbf{await}(B)\{C\} = \mathbf{E}[\,\mathbf{return}\ E\,]$, then ....
Proof: It is vacantly true.
(4) For any σ_F, $(\mathbf{await}(B)\{C\}, \sigma \uplus \sigma_F) \not\longrightarrow_{\mathsf{t}} \mathbf{abort}$.
Proof: By the operational semantics and $\mathcal{D},[I],G \models_{\mathsf{sfair}} \{p \wedge B\}\langle C\rangle\{q\}$.
(5) For any C', σ'', σ_F and Σ_F, if $(\mathbf{await}(B)\{C\}, \sigma \uplus \sigma_F) \longrightarrow_{\mathsf{t}} (C', \sigma'')$, then there exist σ', $\mathbb{C}'$, Σ', k, u', $\mathbb{ws}'$, ws', aw', w', wk', ξ' and ξ'_a such that
(a) $\sigma'' = \sigma' \uplus \sigma_F$, and
(b) $(\mathbb{C}, \Sigma \uplus \Sigma_F) \longrightarrow_{\mathsf{t}}^{*} (\mathbb{C}', \Sigma' \uplus \Sigma_F)$, and
(c) $\mathcal{D},R,G \models_{\mathsf{t}}^{\mathsf{sfair}} (C',\sigma') \leq (\mathbb{C}',\Sigma') \diamond (u',\mathbb{ws}',ws',aw',w',wk',\mathcal{H}) \Downarrow_{\xi',\xi'_a} q$, and
(d) $((\sigma,\Sigma),(\sigma',\Sigma'),k) \models G_{\mathsf{t}} * \mathrm{True}$, and
(e) either $u' <_k u$ and $k > 0$,
or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}' <_{\mathcal{H}}^{wk} \mathbb{ws}$,
or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}' \approx_{\mathcal{H}}^{wk} \mathbb{ws}$ and $wk' < wk$,
or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}' \approx_{\mathcal{H}}^{wk} \mathbb{ws}$ and $wk' = wk < \mathcal{H}$ and $\xi \subseteq \xi'$; and
(f) if $((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle[\mathcal{D}_{\mathsf{t}}]\rangle * \mathrm{True}$ and $k = 0$, then $ws' <_{\mathcal{H}} ws$.
Proof: By the operational semantics, we know C' must be **skip** and

$$[\![B]\!]_s = \mathbf{true} \text{ and } (C, \sigma \uplus \sigma_F) \longrightarrow_{\mathsf{t}}^{*} (\mathbf{skip}, \sigma'') .$$

Thus

$$(\langle C\rangle, \sigma \uplus \sigma_F) \longrightarrow_{\mathsf{t}}^{*} (\mathbf{skip}, \sigma'') .$$

From $\mathcal{D},[I],G \models_{\mathsf{sfair}} \{p \wedge B\}\langle C\rangle\{q\}$, we know there exist aw'' and ξ''_a such that

$$\mathcal{D},[I],G \models_{\mathsf{t}}^{\mathsf{sfair}} (\langle C\rangle,\sigma) \leq (\mathbb{C},\Sigma) \diamond (u,\mathbb{ws},ws,aw'',w,wk,\mathcal{H}) \Downarrow_{\xi,\xi''_a} q .$$

Thus there exist σ', $\mathbb{C}'$, Σ', k, u', $\mathbb{ws}'$, ws', aw', w', wk', ξ' and ξ'_a such that
(A) $\sigma'' = \sigma' \uplus \sigma_F$, and
(B) $(\mathbb{C}, \Sigma \uplus \Sigma_F) \longrightarrow_{\mathsf{t}}^{*} (\mathbb{C}', \Sigma' \uplus \Sigma_F)$, and
(C) $\mathcal{D},[I],G \models_{\mathsf{t}}^{\mathsf{sfair}} (\mathbf{skip},\sigma') \leq (\mathbb{C}',\Sigma') \diamond (u',\mathbb{ws}',ws',aw',w',wk',\mathcal{H}) \Downarrow_{\xi',\xi'_a} q$, and
(D) $((\sigma,\Sigma),(\sigma',\Sigma'),k) \models G_{\mathsf{t}} * \mathrm{True}$, and
(E) either $u' <_k u$ and $k > 0$,
or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}' <_{\mathcal{H}}^{wk} \mathbb{ws}$,

or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}' \approx^{wk}_{\mathcal{H}} \mathbb{ws}$ and $wk' < wk$,
or $u' = u$ and $k = 0$ and $w' = w$ and $\mathbb{ws}' \approx^{wk}_{\mathcal{H}} \mathbb{ws}$ and $wk' = wk < \mathcal{H}$ and $\xi \subseteq \xi'$; and

(F) if $((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle[\mathcal{D}_t]\rangle * \mathsf{True}$ and $k = 0$, then $ws' <_{\mathcal{H}} ws$.

From $\mathcal{D},[I],G \models^{\mathsf{sfair}}_t (\mathbf{skip},\sigma') \preceq (\mathbb{C}',\Sigma') \diamond (u',\mathbb{ws}',ws',aw',w',wk',\mathcal{H}) \Downarrow_{\xi',\xi'_a} q$, we know there exist $\mathbb{C}''$ and Σ'' such that

(G) $(\mathbb{C}',\Sigma' \uplus \Sigma_F) \longrightarrow^*_t (\mathbb{C}'',\Sigma'' \uplus \Sigma_F)$, and

(H) $((\sigma',\Sigma''),(u',w'),\mathbb{C}'') \models q_t$, and

(I) $\mathbb{ws}' = ((0,0),0)$ and $ws' = (0,0)$ and $wk' = \mathcal{H}$ and $\xi' = \emptyset$, and

(J) $((\sigma',\Sigma'),(\sigma',\Sigma''),0) \models G_t * \mathsf{True}$.

Since $\mathsf{Sta}(q, R * \mathsf{Id})$, by (SKIP) rule, we can prove

$$\mathcal{D},R,G \models^{\mathsf{sfair}}_t (\mathbf{skip},\sigma') \preceq (\mathbb{C}'',\Sigma'') \diamond (u',\mathbb{ws}',ws',aw',w',wk',\mathcal{H}) \Downarrow_{\xi',\xi'_a} q\,.$$

Also we know

$$(\mathbb{C},\Sigma \uplus \Sigma_F) \longrightarrow^*_t (\mathbb{C}'',\Sigma'' \uplus \Sigma_F) \;\text{ and }\; ws' <_{\mathcal{H}} ws \;\text{ and }$$
$$((\sigma,\Sigma),(\sigma',\Sigma''),k) \models G_t * \mathsf{True} \text{ (suppose } G \text{ is transitive-closed)}$$

(6) If $((\sigma,\Sigma),(\sigma',\Sigma'),k) \models R * \mathsf{Id}$, then there exist u', $\mathbb{ws}'$, ws', aw', w', wk', ξ_d, ξ_{ad}, ξ' and ξ'_a such that

(a) $\mathcal{D},R,G \models^{\mathsf{sfair}}_t (\mathbf{await}(B)\{C\},\sigma') \preceq (\mathbb{C},\Sigma') \diamond (u',\mathbb{ws}',ws',aw',w',wk',\mathcal{H}) \Downarrow_{\xi',\xi'_a} q$, and

(b) $u' \approx_k u$, and
$k = 0 \Longrightarrow w' = w$, and

(c) $\xi_d = \{t' \mid (t' \in \xi) \wedge (((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle\mathcal{D}_{t'}\rangle * \mathsf{Id})\}$ and
$k = 0 \Longrightarrow \mathbb{ws}' <^{wk}_{\mathcal{H}} \mathbb{ws} \vee \mathbb{ws}' \approx^{wk}_{\mathcal{H}} \mathbb{ws}$ and
$k = 0 \wedge wk < \mathcal{H} \wedge (\xi_d \neq \emptyset \vee (\Sigma \models \neg\mathsf{en}(\mathbb{C}) \wedge \Sigma' \models \mathsf{en}(\mathbb{C}))) \Longrightarrow \mathbb{ws}' <^{wk}_{\mathcal{H}} \mathbb{ws}$ and
$k = 0 \wedge (\xi \backslash \xi_d \neq \emptyset \vee \Sigma' \models \neg\mathsf{en}(\mathbb{C})) \Longrightarrow wk' \leq wk$, and
$k = 0 \wedge wk' = wk \Longrightarrow \xi \backslash \xi_d \subseteq \xi'$, and

(d) if $k = 0$ and $(\sigma,\Sigma) \models \mathsf{Enabled}(\mathcal{D}_t) * \mathsf{true}$, then $ws' \leq_{\mathcal{H}} ws$; and

(e) $\xi_{ad} = \{t' \mid (t' \in \xi_a) \wedge (((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle\mathcal{D}_{t'}\rangle * \mathsf{Id})\}$ and
$k = 0 \wedge \mathsf{is\_await}(\mathbf{await}(B)\{C\}) \Longrightarrow \xi_a \backslash \xi_{ad} \subseteq \xi'_a$ and
$k = 0 \wedge \mathsf{is\_await}(\mathbf{await}(B)\{C\}) \wedge (\xi_{ad} \neq \emptyset \vee (\Sigma \models \neg\mathsf{en}(\mathbb{C}) \wedge \Sigma' \models \mathsf{en}(\mathbb{C}))) \Longrightarrow aw' < aw$ and
$k = 0 \wedge \mathsf{is\_await}(\mathbf{await}(B)\{C\}) \wedge (\sigma \models \neg\mathsf{en}(C)) \wedge (\sigma' \models \neg\mathsf{en}(C)) \Longrightarrow aw' \leq aw$.

Proof: Since $\mathsf{Sta}(p, R * \mathsf{Id})$, we know there exist u' and w' such that

$$((\sigma',\Sigma'),(u',w'),\mathbb{C}) \models p_t \;\text{ and }\; u' \approx_k u \;\text{ and }\; k = 0 \Longrightarrow w' = w\,.$$

Let

$$aw' = f_t(\sigma',\Sigma') \;\text{ and }\; \xi'_a = \{t' \mid (t' \neq t) \wedge ((\sigma',\Sigma') \models \mathsf{Enabled}(\mathcal{D}'_{t'}) * \mathsf{true})\}\,.$$

By the co-induction hypothesis, we know

$$\mathcal{D},R,G \models^{\mathsf{sfair}}_t (\mathbf{await}(B)\{C\},\sigma') \preceq (\mathbb{C},\Sigma') \diamond (u',\mathbb{ws},ws,aw',w',wk,\mathcal{H}) \Downarrow_{\xi,\xi'_a} q\,.$$

Let

$$\xi_{ad} = \{t' \mid (t' \in \xi_a) \wedge (((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle\mathcal{D}_{t'}\rangle * \mathsf{Id})\}.$$

Since $\mathsf{Enabled}(\mathcal{D}) \Rightarrow I$, $\mathcal{D}' \Subset \mathcal{D}$ and $\mathsf{wffAct}(R,\mathcal{D}')$, we can prove:

$$k = 0 \Longrightarrow \xi_a \backslash \xi_{ad} \subseteq \xi'_a\,.$$

From $(\sigma,\Sigma) \models (R\colon \mathcal{D}' \bullet\!\!\xrightarrow{f} (B,\mathbb{B}'))$, we know

$$k = 0 \wedge (\sigma \models \neg\mathsf{en}(C)) \wedge (\sigma' \models \neg\mathsf{en}(C)) \Longrightarrow aw' \leq aw\,.$$

Suppose $k = 0$. If $\xi_{ad} \neq \emptyset$, then there exists t' such that $t' \in \xi_a$ and $((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle\mathcal{D}_{t'}\rangle * \mathsf{Id}$. Since $\mathcal{D}' \Subset \mathcal{D}$, we can prove

$$\mathsf{wr}^*(\Gamma)(f) \stackrel{\text{def}}{=} (\mathcal{P}, x, \mathsf{wr}^*(\mathbf{await}(B)\{C\}); \mathbf{return}\ E) \quad \text{if } \Gamma(f) = (\mathcal{P}, x, \mathbf{await}(B)\{C\}; \mathbf{return}\ E)$$

$\mathsf{wr}^*(\mathbf{await}(B)\{C\}) \stackrel{\text{def}}{=}$

```
local u1 := nondet(), u2 := nondet();
while ( u1 >= 0 ) {
  while ( !B || done ) { };
  u1--;
  done := true;
  done := false;
}
await(B ∧ ¬done){C; done := true; };
done := false;
while ( u2 >= 0 ) {
  while ( done ) { };
  u2--;
  done := true;
  done := false;
}
```

$\mathsf{wr}^*(\varphi)(\sigma) \stackrel{\text{def}}{=} \mathsf{wr}^{\chi}_{\mathsf{PDF}}(\varphi)(\sigma)$

Fig. 22. The special SFAIR-PDF wrapper wr^*.

$$((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle \mathcal{D}'_{\mathsf{t}'} \rangle * \mathsf{Id}\,.$$

Since $(\sigma,\Sigma) \models (R\colon \mathcal{D}' \bullet\!\!\xrightarrow{f} (B,\mathbb{B}'))$, we know

$$aw' < aw\,.$$

Also if $\Sigma \models \neg\mathsf{en}(\mathbb{C}) \wedge \Sigma' \models \mathsf{en}(\mathbb{C})$, from $(\sigma,\Sigma) \models (R\colon \mathcal{D}' \bullet\!\!\xrightarrow{f} (B,\mathbb{B}'))$, we can still prove:

$$aw' < aw\,.$$

Thus we are done. □

B.5 Local simulations with respect to abstractions

In this subsection, we define the simulations $\mathcal{D},R,G \models_{\chi} \{P\}\Pi \lesssim \Gamma$ and $\mathcal{D},R,G \models\!\!\models_{\chi} \{P\}\Pi \lesssim (\Gamma,\mathsf{wr}^*(\Gamma))$, and prove ② and ③ in Fig. 21. We define the special wrapper wr^* in Fig. 22. We also give some useful auxiliary definitions in Fig. 24, which describe the executions of the wrapper.

Definition B.14 (Simulations with respect to abstractions). $\mathcal{D},R,G \models_{\chi} \{P\}\Pi \lesssim \Gamma$ iff there exist $\mathcal{D}$, R and G such that $\mathcal{D},R,G \models_{\chi} \{P\}\Pi : \Gamma$, $R \Rightarrow \lfloor R \rfloor_0$ and $G \Rightarrow \lfloor G \rfloor_0$.

$\mathcal{D},R,G \models\!\!\models_{\chi} \{P\}\Pi \lesssim (\Gamma,\mathsf{wr}^*(\Gamma))$ iff, for any $f \in dom(\Pi)$, for any σ, $\mathbb{C}$ and Σ, for any t, if $\Pi(f) = (\mathcal{P},x,C)$, $\Gamma(f) = (\mathcal{P}',y,(\mathbf{await}(\mathbb{B})\{\mathbb{C}_0\}; \mathbf{return}\ \mathbb{E}))$, $\mathbb{C} = (\mathsf{wr}^*(\mathbf{await}(\mathbb{B})\{\mathbb{C}_0\}); \mathbf{return}\ \mathbb{E})$ and $(\sigma,\Sigma) \models (P_{\mathsf{t}} \wedge \mathcal{P}_{\mathsf{t}}) * (\mathsf{done} = \mathsf{false}) * \mathsf{own}(x) * \mathsf{own}(y) \wedge (x = y)$, there exist three well-founded metrics $\mathbb{M}$, M and aw, a boolean flag wb and two sets $\xi, \xi_a \in \mathscr{P}(ThrdID)$ such that $wb = \mathbf{false}$ and

$$\mathcal{D},R,G \models^{\chi}_{\mathsf{t}} (C,\sigma) \lesssim (\mathbb{B},\mathbb{C},\Sigma) \diamond (\mathbb{M},M,wb,aw) \Downarrow_{\xi,\xi_a} (P * \mathsf{own}(x) * \mathsf{own}(y)).$$

Here $\mathcal{D},R,G \models^{\chi}_{\mathsf{t}} (C,\sigma) \lesssim (\mathbb{B},\mathbb{C},\Sigma) \diamond (\mathbb{M},M,wb,aw) \Downarrow_{\xi,\xi_a} Q$ is co-inductively defined as follows. Whenever $\mathcal{D},R,G \models^{\chi}_{\mathsf{t}} (C,\sigma) \lesssim (\mathbb{B},\mathbb{C},\Sigma) \diamond (\mathbb{M},M,wb,aw) \Downarrow_{\xi,\xi_a} Q$ holds, then the following hold:

(1)(a) Suppose $\sigma = (s,h)$. Then $\xi \cup \xi_a \subseteq s(\mathtt{TIDS})$ and $\mathsf{t} \notin \xi$ and $\mathsf{t} \notin \xi_a$.

$\mathsf{wr}_0(\mathbf{await}(B)\{C\}) \stackrel{\text{def}}{=}$

```
while ( u1 >= 0 ) {
  while ( !B || done ) { };
  u1--;
  done := true;
  done := false;
}
await(B ∧ ¬done){C; done := true; };
done := false;
```

$\mathsf{wr}_1(\mathbf{return}\ E) \stackrel{\text{def}}{=}$

```
while ( u2 >= 0 ) {
  while ( done ) { };
  u2--;
  done := true;
  done := false;
}
return E;
```

$\mathsf{wr}'_0(\mathbf{await}(B)\{C\}) \stackrel{\text{def}}{=}$

```
while ( !B || done ) { };
u1--;
done := true;
done := false;
while ( u1 >= 0 ) {
  while ( !B || done ) { };
  u1--;
  done := true;
  done := false;
}
await(B ∧ ¬done){C; done := true; };
done := false;
```

$\mathsf{wr}'_1(\mathbf{return}\ E) \stackrel{\text{def}}{=}$

```
while ( done ) { };
u2--;
done := true;
done := false;
while ( u2 >= 0 ) {
  while ( done ) { };
  u2--;
  done := true;
  done := false;
}
return E;
```

Fig. 23. Useful notations for the special wrapper wr^*.

$\mathsf{wrsteps}^0_{\mathsf{t}}((\mathbb{C},\Sigma),(\mathbb{C}',\Sigma'))$ iff
$\quad \mathbb{C} = \mathbb{C}' \wedge \Sigma = \Sigma'$

$\mathsf{wrsteps}^{n+1}_{\mathsf{t}}((\mathbb{C},\Sigma),(\mathbb{C}',\Sigma'))$ iff
$\quad \exists \mathbb{C}'', \Sigma''.\ (\mathbb{C},\Sigma) \longrightarrow_{\mathsf{t}} (\mathbb{C}'',\Sigma'') \wedge \mathsf{wrsteps}^n_{\mathsf{t}}((\mathbb{C}'',\Sigma''),(\mathbb{C}',\Sigma'))$
$\quad \wedge\ \Sigma'' = \Sigma\{\mathsf{done} \rightsquigarrow \_, \mathsf{u1} \rightsquigarrow \_, \mathsf{u2} \rightsquigarrow \_\}$

$\mathsf{exec0}_{\mathsf{t}}((\mathbb{C},\Sigma),(\mathbb{C}',\Sigma'))$ iff
$\quad \exists \mathbb{C}'', \Sigma'', n_1, n_2.\ \mathsf{wrsteps}^{n_1+1}_{\mathsf{t}}((\mathbb{C},\Sigma),(\mathbb{C}'',\Sigma'')) \wedge \Sigma''(\mathsf{done}) = \mathbf{true}$
$\quad \wedge\ \mathsf{wrsteps}^{n_2+1}_{\mathsf{t}}((\mathbb{C}'',\Sigma''),(\mathbb{C}',\Sigma'))$

$\mathsf{exec1}_{\mathsf{t}}((\mathbb{C},\Sigma),(\mathbb{C}',\Sigma'))$ iff
$\quad \exists \mathbb{C}'', \Sigma'', \mathbb{C}''', \Sigma''', \mathbb{C}'''', \Sigma'''', n_1, n_2, n_3.\ \mathsf{wrsteps}^{n_1+1}_{\mathsf{t}}((\mathbb{C},\Sigma),(\mathbb{C}'',\Sigma'')) \wedge \Sigma''(\mathsf{done}) = \mathbf{true}$
$\quad \wedge\ \mathsf{wrsteps}^{n_2+1}_{\mathsf{t}}((\mathbb{C}'',\Sigma''),(\mathbb{C}''',\Sigma'''))$
$\quad \wedge\ (\mathbb{C}''',\Sigma''') \longrightarrow_{\mathsf{t}} (\mathbb{C}'''',\Sigma'''')$
$\quad \wedge\ \mathsf{wrsteps}^{n_3+1}_{\mathsf{t}}((\mathbb{C}'''',\Sigma''''),(\mathbb{C}',\Sigma'))$

$\mathsf{is\_while0}(\mathbb{C},\mathbb{B})$ iff $\exists \mathbb{E}, \mathbb{C}_0.\ \mathbb{C} = (\mathsf{wr}'_0(\mathbf{await}(\mathbb{B})\{\mathbb{C}_0\}); \mathsf{wr}_1(\mathbf{return}\ \mathbb{B}))$

$\mathsf{is\_while1}(\mathbb{C})$ iff $\exists \mathbb{E}.\ \mathbb{C} = \mathsf{wr}'_1(\mathbf{return}\ \mathbb{B})$

$\mathsf{is\_return}(C)$ iff $\exists \mathbf{E}, E.\ C = \mathbf{E}[\,\mathbf{return}\ E\,]$

Fig. 24. Executions of the special wrapper wr^*.

(b) For any $t' \in \xi \cup \xi_a$, we have $(\sigma,\Sigma) \models \mathsf{Enabled}(\mathcal{D}_{t'}) * \mathsf{true}$.
(c) If $wb = \textbf{false}$, then $\xi = \emptyset$. If $wb = \textbf{true}$, then $\xi \neq \emptyset \vee \Sigma \models \neg\mathbb{B}$.
(d) If $\sigma \models \neg\mathsf{en}(C)$ and $\Sigma \models \mathbb{B}$, then $\xi_a \neq \emptyset$.
(e) If $(\sigma,\Sigma) \models \mathsf{Enabled}(\mathcal{D}_t) * \mathsf{true}$, then $\sigma \models \mathsf{en}(C)$.
(f) $\Sigma(\mathsf{done}) = \textbf{false}$.
(g) If $\Sigma \models \neg\mathbb{B}$, then $\mathsf{is\_while0}(\mathbb{C},\mathbb{B})$. If $\neg\mathsf{is\_return}(C)$, then $\mathsf{is\_while0}(\mathbb{C},\mathbb{B}) \vee \mathsf{is\_while1}(\mathbb{C})$.

(2) If $C = \mathbf{E}[\,\textbf{return}\ E\,]$, then there exists $\mathbb{E}$ such that
(a) $\mathbb{C} = (\textbf{return}\ \mathbb{E})$, and
(b) $(\sigma,\Sigma) \models Q_t$ and $[\![E]\!]_{\sigma.s} = [\![\mathbb{E}]\!]_{\Sigma.s}$, and
(c) $((\sigma,\Sigma),(\sigma,\Sigma),0) \models G_t * \mathsf{True}$, and
(d) $wb = \textbf{false}$.

(3) For any σ_F, $(C, \sigma \uplus \sigma_F) \not\longrightarrow_t \textbf{abort}$.

(4) For any C', σ'', σ_F and Σ_F, if $(C, \sigma \uplus \sigma_F) \longrightarrow_t (C', \sigma'')$ and $\Sigma \perp \Sigma_F$, then there exist σ', $\mathbb{C}'$, Σ', k, $\mathbb{M}'$, M', wb', aw', ξ', ξ'_a, n and $\mathbb{B}'$ such that
(a) $\sigma'' = \sigma' \uplus \sigma_F$, and
(b) $(\mathbb{C}, \Sigma \uplus \Sigma_F) \longrightarrow_t^n (\mathbb{C}', \Sigma' \uplus \Sigma_F)$; and
if $k > 0$, then $\mathsf{exec0}_t((\mathbb{C}, \Sigma \uplus \Sigma_F),(\mathbb{C}', \Sigma' \uplus \Sigma_F)) \vee \mathsf{exec1}_t((\mathbb{C}, \Sigma \uplus \Sigma_F),(\mathbb{C}', \Sigma' \uplus \Sigma_F))$; and
if $\mathsf{is\_while1}(\mathbb{C}')$, then $\mathbb{B}' = \textbf{true}$, otherwise $\mathbb{B}' = \mathbb{B}$; and
(c) $\mathcal{D}, R, G \models_t^{\chi} (C', \sigma') \lesssim (\mathbb{B}', \mathbb{C}', \Sigma') \diamond (\mathbb{M}', M', wb', aw') \Downarrow_{\xi',\xi'_a} Q$, and
(d) $((\sigma,\Sigma),(\sigma',\Sigma'),k) \models G_t * [\mathsf{done} = \mathsf{false}] * \mathsf{True}$, and
(e) either $n > 0$,
or $\mathbb{M}' < \mathbb{M}$,
or $\mathbb{M}' = \mathbb{M}$ and $wb' = wb = \textbf{true}$ and $\xi \subseteq \xi'$; and
(f) if $((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle[\mathcal{D}_t]\rangle * \mathsf{True}$ and $k = 0$, then $M' < M$.

(5) For any k, σ' and Σ', if $((\sigma,\Sigma),(\sigma',\Sigma'),k) \models R_t * [\mathsf{done} = \mathsf{false}] * \mathsf{Id}$, then there exist $\mathbb{M}'$, M', wb', aw', ξ_d, ξ_{ad}, ξ' and ξ'_a such that
(a) $\mathcal{D}, R, G \models_t^{\chi} (C, \sigma') \lesssim (\mathbb{B}, \mathbb{C}, \Sigma') \diamond (\mathbb{M}', M', wb', aw') \Downarrow_{\xi',\xi'_a} Q$, and
(b) if $k > 0$ and $\neg\mathsf{is\_return}(C)$, then for any Σ'' and Σ_F such that $\Sigma'' = \Sigma\{\mathsf{done} \rightsquigarrow \textbf{true}\}$ and $\Sigma_F \perp \Sigma$ we have $(\mathbb{C}, \Sigma'' \uplus \Sigma_F) \longrightarrow_t^+ (\mathbb{C}, \Sigma'' \uplus \Sigma_F)$;
(c) $\xi_d = \{t' \mid (t' \in \xi) \wedge (((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle\mathcal{D}_{t'}\rangle * \mathsf{Id})\}$ and
$k = 0 \Longrightarrow \mathbb{M}' < \mathbb{M} \vee (\mathbb{M}' = \mathbb{M} \wedge wb' = wb)$ and
$k = 0 \wedge wb = \textbf{true} \wedge (\xi_d \neq \emptyset \vee (\Sigma \models \neg\mathbb{B} \wedge \Sigma' \models \mathbb{B})) \Longrightarrow \mathbb{M}' < \mathbb{M}$ and
$k = 0 \wedge \mathbb{M}' = \mathbb{M} \wedge wb' = wb = \textbf{true} \Longrightarrow \xi \backslash \xi_d \subseteq \xi'$, and
(d) if $k = 0$ and $(\sigma,\Sigma) \models \mathsf{Enabled}(\mathcal{D}_t) * \mathsf{true}$, then $M' \leq M$; and
(e) $\xi_{ad} = \{t' \mid (t' \in \xi_a) \wedge (((\sigma,\Sigma),(\sigma',\Sigma')) \models \langle\mathcal{D}_{t'}\rangle * \mathsf{Id})\}$ and
$k = 0 \wedge \mathsf{is\_await}(C) \Longrightarrow \xi_a \backslash \xi_{ad} \subseteq \xi'_a$ and
$k = 0 \wedge \mathsf{is\_await}(C) \wedge (\xi_{ad} \neq \emptyset \vee (\Sigma \models \neg\mathbb{B} \wedge \Sigma' \models \mathbb{B})) \Longrightarrow aw' < aw$ and
$(\chi = \mathsf{sfair}) \wedge k = 0 \wedge \mathsf{is\_await}(C) \wedge (\sigma \models \neg\mathsf{en}(C)) \wedge (\sigma' \models \neg\mathsf{en}(C)) \Longrightarrow aw' \leq aw$ and
$(\chi = \mathsf{wfair}) \wedge k = 0 \wedge \mathsf{is\_await}(C) \Longrightarrow aw' \leq aw$.

B.5.1 Transforming to simulations with respect to abstractions. The key point of $\Pi \lesssim \Pi'$ is that we remove the metric u and use the non-atomic abstract code Π' to describe the effects of delay. Below we prove ② and ③ in Fig. 21.

- By Definition B.14, the proofs of ② are trivial.
- Lemma B.15 shows ③.

Lemma B.15 (③ in Fig. 21). *If* $\mathcal{D}, R, G \models_\chi \{P\}\Pi : \Gamma$, *then* $\mathcal{D}, R, G \models_\chi \{P\}\Pi \lesssim (\Gamma, \mathsf{wr}^*(\Gamma))$.

Proof. For any $f \in dom(\Pi)$, for any σ, $\mathbb{C}$ and Σ, for any t, if $\Pi(f) = (\mathcal{P}, x, C)$, $\Gamma(f) = (\mathcal{P}', y, (\textbf{await}(\mathbb{B})\{\mathbb{C}_0\}; \textbf{return}\ \mathbb{E}))$, $\mathbb{C} = (\mathsf{wr}^*(\textbf{await}(\mathbb{B})\{\mathbb{C}_0\}); \textbf{return}\ \mathbb{E})$ and $(\sigma, \Sigma) \models (P_\mathsf{t} \wedge \mathcal{P}_\mathsf{t}) * (\mathsf{done} = \mathsf{false}) * \mathsf{own}(x) * \mathsf{own}(y) \wedge (x = y)$, we know there exists Σ_1 such that $\Sigma = \Sigma_1 \uplus \{\mathsf{done} \rightsquigarrow \textbf{false}\}$ and

$$(\sigma, \Sigma_1) \models (P_\mathsf{t} \wedge \mathcal{P}_\mathsf{t}) * \mathsf{own}(x) * \mathsf{own}(y) \wedge (x = y)$$

Let $\mathbb{C}_1 = (\textbf{await}(\mathbb{B})\{\mathbb{C}_0\}; \textbf{return}\ \mathbb{E})$. From $\mathcal{D}, R, G \models_\chi \{P\}\Pi : \Gamma$, we know there exist four well-founded metrics u, $\mathbb{M}$, M and aw, a boolean flag wb and two sets $\xi, \xi_a \in \mathscr{P}(ThrdID)$ such that $wb = \textbf{false}$ and

$$\mathcal{D}, R, G \models^\chi_\mathsf{t} (C, \sigma) \preceq (\mathbb{C}_1, \Sigma_1) \diamond (u, \mathbb{M}, M, wb, aw) \Downarrow_{\xi, \xi_a} (P * \mathsf{own}(x) * \mathsf{own}(y)).$$

We want to prove:

$$\mathcal{D}, R, G \models^\chi_\mathsf{t} (C, \sigma) \precsim (\mathbb{B}, \mathbb{C}, \Sigma) \diamond (\mathbb{M}, M, wb, aw) \Downarrow_{\xi, \xi_a} (P * \mathsf{own}(x) * \mathsf{own}(y)).$$

Fig. 23 gives some useful notations for the special wrapper. We only need to prove the following:

(1) If $\mathcal{D}, R, G \models^\chi_\mathsf{t} (C, \sigma) \preceq ((\textbf{await}(\mathbb{B})\{\mathbb{C}_0\}; \textbf{return}\ \mathbb{E}), \Sigma_1) \diamond (u, \mathbb{M}, M, wb, aw) \Downarrow_{\xi, \xi_a} Q$
and $\Sigma = \Sigma_1 \uplus \{\mathsf{done} \rightsquigarrow \textbf{false}, \mathsf{u1} \rightsquigarrow u_1, \mathsf{u2} \rightsquigarrow u_2\}$ and $0 \le u \le u_1$ and $0 \le u \le u_2$,
then $\mathcal{D}, R, G \models^\chi_\mathsf{t} (C, \sigma) \precsim (\mathbb{B}, (\mathsf{wr}'_0(\textbf{await}(\mathbb{B})\{\mathbb{C}_0\}); \mathsf{wr}_1(\textbf{return}\ \mathbb{E})), \Sigma) \diamond (\mathbb{M}, M, wb, aw) \Downarrow_{\xi, \xi_a} Q$.

(2) If $\mathcal{D}, R, G \models^\chi_\mathsf{t} (C, \sigma) \preceq ((\textbf{return}\ \mathbb{E}), \Sigma_1) \diamond (u, \mathbb{M}, M, wb, aw) \Downarrow_{\xi, \xi_a} Q$ and $\neg\mathsf{is\_return}(C)$
and $\Sigma = \Sigma_1 \uplus \{\mathsf{done} \rightsquigarrow \textbf{false}, \mathsf{u1} \rightsquigarrow u_1, \mathsf{u2} \rightsquigarrow u_2\}$ and $0 \le u \le u_2$,
then $\mathcal{D}, R, G \models^\chi_\mathsf{t} (C, \sigma) \precsim (\textbf{true}, \mathsf{wr}'_1(\textbf{return}\ \mathbb{E}), \Sigma) \diamond (\mathbb{M}, M, wb, aw) \Downarrow_{\xi, \xi_a} Q$.

(3) If $\mathcal{D}, R, G \models^\chi_\mathsf{t} (C, \sigma) \preceq ((\textbf{return}\ \mathbb{E}), \Sigma_1) \diamond (u, \mathbb{M}, M, wb, aw) \Downarrow_{\xi, \xi_a} Q$ and $\mathsf{is\_return}(C)$
and $\Sigma = \Sigma_1 \uplus \{\mathsf{done} \rightsquigarrow \textbf{false}, \mathsf{u1} \rightsquigarrow u_1, \mathsf{u2} \rightsquigarrow u_2\}$,
then $\mathcal{D}, R, G \models^\chi_\mathsf{t} (C, \sigma) \precsim (\textbf{true}, (\textbf{return}\ \mathbb{E}), \Sigma) \diamond (\mathbb{M}, M, wb, aw) \Downarrow_{\xi, \xi_a} Q$.

By co-induction. □

B.6 Lifting to Simulations for Client Threads

Since we have two kinds of object-local simulations in Fig. 21, we also need two kinds of simulations for client threads. Below Definition B.16 gives the client simulation with PSF objects (under either strong or weak fairness, corresponding to $\mathcal{D}, R, G \models_\chi \{P\}\Pi \precsim \Gamma$). Definition B.17 gives the client simulation with PDF objects (under either strong or weak fairness, corresponding to $\mathcal{D}, R, G \models_\chi \{P\}\Pi \precsim (\Gamma, \mathsf{wr}^*(\Gamma))$).

In the definitions, we use $\mathsf{is\_clt}(e)$ to say that e is a client event in the form of $(\mathsf{t}, \textbf{out}, n)$, $(\mathsf{t}, \textbf{clt})$ and $(\mathsf{t}, \textbf{term})$. Similarly we use $\mathsf{is\_obj}(e)$ to say that e is an event inside method calls in the form of $(\mathsf{t}, \textbf{obj})$. We also define:

$$\mathsf{getB}(\Pi, f) \stackrel{\text{def}}{=} \mathsf{en}(C) \qquad \text{if } \Pi(f) = (\mathcal{P}, x, C)$$
$$\mathsf{inObj}(C, \kappa) \ \text{ iff } \ (\kappa \neq \circ) \wedge \neg\mathsf{is\_return}(C)$$

Similar to the definitions of $\mathsf{exec0_t}((\mathbb{C}, \Sigma), (\mathbb{C}', \Sigma'))$ and $\mathsf{exec1_t}((\mathbb{C}, \Sigma), (\mathbb{C}', \Sigma'))$ in Fig. 24, we can also define $\mathsf{exec0_t}((\mathbb{C}, \delta), (\mathbb{C}', \delta'))$ and $\mathsf{exec1_t}((\mathbb{C}, \delta), (\mathbb{C}', \delta'))$ for the executions of the thread.

Definition B.16 (Simulation for Thread with PSF objects). $\mathcal{D}, R, G \models_\chi \{P\}(\Pi, C) \precsim (\Gamma, \mathbb{C})$ iff, for any σ_c, σ and Σ, for any t, if $(\sigma, \Sigma) \models P_\mathsf{t}$, there exist three well-founded metrics $\mathbb{M}$, M and aw, a boolean flag wb and two sets $\xi, \xi_a \in \mathscr{P}(ThrdID)$ such that $wb = \textbf{false}$ and

$$\mathcal{D}, R, G \models^\chi_\mathsf{t} (\Pi, C, (\sigma_c, \sigma, \circ)) \precsim (\Gamma, \mathbb{C}, (\Sigma, \circ)) \diamond (\mathbb{M}, M, wb, aw) \Downarrow_{\xi, \xi_a} P.$$

Here $\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (\Pi, C, (\sigma_c, \sigma, \kappa)) \lesssim (\Gamma, \mathbb{C}, (\Sigma, \Bbbk)) \diamond (\mathbb{M}, M, wb, aw) \Downarrow_{\xi, \xi_a} Q$ is co-inductively defined as follows.
Whenever $\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (\Pi, C, (\sigma_c, \sigma, \kappa)) \lesssim (\Gamma, \mathbb{C}, (\Sigma, \Bbbk)) \diamond (\mathbb{M}, M, wb, aw) \Downarrow_{\xi, \xi_a} Q$ holds, then the following hold:

(1)(a) Suppose $\sigma = (s, h)$. Then $\xi \cup \xi_a \subseteq s(\mathsf{TIDS})$ and $\mathsf{t} \notin \xi$ and $\mathsf{t} \notin \xi_a$.
(b) For any $\mathsf{t}' \in \xi \cup \xi_a$, we have $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}'}) * \mathsf{true}$.
(c) If $wb = \mathbf{false}$, then $\xi = \emptyset$. If $wb = \mathbf{true}$, then $\xi \neq \emptyset \vee (\Sigma, \Bbbk) \models \neg\mathsf{en}(\mathbb{C})$.
(d) If $(\sigma, \kappa) \models \neg\mathsf{en}(C)$ and $(\Sigma, \Bbbk) \models \mathsf{en}(\mathbb{C})$ and $\kappa \neq \circ$, then $\xi_a \neq \emptyset$.
(e) If $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}}) * \mathsf{true}$ and $\kappa \neq \circ$, then $(\sigma, \kappa) \models \mathsf{en}(C)$.
(f) If $\neg\mathsf{inObj}(C, \kappa)$, then $wb = \mathbf{false}$.
(g) $\kappa = \circ \Longrightarrow C = \mathbb{C}$, and $\kappa = \circ \Longleftrightarrow \Bbbk = \circ$, and $C = \mathbf{skip} \Longleftrightarrow \mathbb{C} = \mathbf{skip}$.

(2) If $\neg\mathsf{inObj}(C, \kappa)$, then $(\sigma, \Sigma) \models Q_{\mathsf{t}}$.

(3) For any σ_F and Σ_F, if $(C, (\sigma_c, \sigma \uplus \sigma_F, \kappa)) \xrightarrow{e}_{\mathsf{t},\Pi} \mathbf{abort}$ and $\Sigma \perp \Sigma_F$, then $e = (\mathsf{t}, \mathbf{clt}, \mathbf{abort})$ and $(\mathbb{C}, (\sigma_c, \Sigma \uplus \Sigma_F, \Bbbk)) \xrightarrow{e}_{\mathsf{t},\Gamma} \mathbf{abort}$.

(4) For any C', σ_c', σ'', κ', σ_F and Σ_F, if $(C, (\sigma_c, \sigma \uplus \sigma_F, \kappa)) \xrightarrow{e}_{\mathsf{t},\Pi} (C', (\sigma_c', \sigma'', \kappa'))$ and $\Sigma \perp \Sigma_F$, then there exist n, σ', $\mathcal{E}$, $\mathbb{C}'$, Σ', $\Bbbk'$, $\mathbb{M}'$, M', wb', aw', ξ' and ξ_a' such that
(a) $\sigma'' = \sigma' \uplus \sigma_F$, and
(b) $(\mathbb{C}, (\sigma_c, \Sigma \uplus \Sigma_F, \Bbbk)) \xrightarrow{\mathcal{E}}{}^{n}_{\mathsf{t},\Gamma} (\mathbb{C}', (\sigma_c', \Sigma' \uplus \Sigma_F, \Bbbk'))$, and $n = 0 \vee n = 1$, and
(c) $\mathsf{is\_clt}(e) \vee \mathsf{is\_inv}(e) \vee \mathsf{is\_ret}(e) \Longrightarrow \mathcal{E} = e :: \epsilon$, and
$\mathsf{is\_obj}(e) \Longrightarrow (\forall i, e' = \mathcal{E}(i).\ \mathsf{is\_obj}(e'))$, and
(d) $\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (\Pi, C', (\sigma_c', \sigma', \kappa')) \lesssim (\Gamma, \mathbb{C}', (\Sigma', \Bbbk')) \diamond (\mathbb{M}', M', wb', aw') \Downarrow_{\xi', \xi_a'} Q$, and
(e) $((\sigma, \Sigma), (\sigma', \Sigma'), 0) \models G_{\mathsf{t}} * \mathsf{True}$, and
(f) if $\mathsf{inObj}(C, \kappa)$, then $n > 0$, or $\mathbb{M}' < \mathbb{M}$, or $\mathbb{M}' = \mathbb{M}$ and $wb' = wb = \mathbf{true}$ and $\xi \subseteq \xi'$; and
(g) if $((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle [\mathcal{D}_{\mathsf{t}}] \rangle * \mathsf{True}$, then $M' < M$.

(5) For any σ' and Σ', if $((\sigma, \Sigma), (\sigma', \Sigma'), 0) \models R_{\mathsf{t}} * \mathsf{Id}$, then there exist $\mathbb{M}'$, M', wb', aw', ξ_d, ξ_{ad}, ξ' and ξ_a' such that
(a) $\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (\Pi, C, (\sigma_c', \sigma', \kappa)) \lesssim (\Gamma, \mathbb{C}, (\Sigma', \Bbbk)) \diamond (\mathbb{M}', M', wb', aw') \Downarrow_{\xi', \xi_a'} Q$, and
(b) $\xi_d = \{\mathsf{t}' \mid (\mathsf{t}' \in \xi) \wedge (((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle \mathcal{D}_{\mathsf{t}'} \rangle * \mathsf{Id})\}$ and
$\mathsf{inObj}(C, \kappa) \Longrightarrow \mathbb{M}' < \mathbb{M} \vee (\mathbb{M}' = \mathbb{M} \wedge wb' = wb)$ and
$wb = \mathbf{true} \wedge (\xi_d \neq \emptyset \vee ((\Sigma, \Bbbk) \models \neg\mathsf{en}(\mathbb{C}) \wedge (\Sigma', \Bbbk) \models \mathsf{en}(\mathbb{C}))) \Longrightarrow \mathbb{M}' < \mathbb{M}$ and
$\mathbb{M}' = \mathbb{M} \wedge wb' = wb = \mathbf{true} \Longrightarrow \xi \backslash \xi_d \subseteq \xi'$, and
(c) if $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}}) * \mathsf{true}$, then $M' \leq M$; and
(d) $\xi_{ad} = \{\mathsf{t}' \mid (\mathsf{t}' \in \xi_a) \wedge (((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle \mathcal{D}_{\mathsf{t}'} \rangle * \mathsf{Id})\}$ and
$\mathsf{inObj}(C, \kappa) \wedge \mathsf{is\_await}(C) \Longrightarrow \xi_a \backslash \xi_{ad} \subseteq \xi_a'$ and
$\mathsf{inObj}(C, \kappa) \wedge \mathsf{is\_await}(C) \wedge (\xi_{ad} \neq \emptyset \vee ((\Sigma, \Bbbk) \models \neg\mathsf{en}(\mathbb{C}) \wedge (\Sigma', \Bbbk) \models \mathsf{en}(\mathbb{C}))) \Longrightarrow aw' < aw$ and
$(\chi = \mathsf{sfair}) \wedge \mathsf{inObj}(C, \kappa) \wedge \mathsf{is\_await}(C) \wedge ((\sigma, \kappa) \models \neg\mathsf{en}(C)) \wedge ((\sigma', \kappa) \models \neg\mathsf{en}(C)) \Longrightarrow aw' \leq aw$ and
$(\chi = \mathsf{wfair}) \wedge \mathsf{inObj}(C, \kappa) \wedge \mathsf{is\_await}(C) \Longrightarrow aw' \leq aw$.

Definition B.17 (Simulation for Thread with PDF objects). $\mathcal{D}, R, G \models_{\chi} \{P\}(\Pi, C) \lesssim (\Pi', \Gamma, \mathbb{C})$ iff, for any σ_c, σ and Σ, for any t, if $(\sigma, \Sigma) \models P_{\mathsf{t}}$, there exist three well-founded metrics $\mathbb{M}$, M and aw, a boolean flag wb, a boolean expression $\mathbb{B}$ and two sets $\xi, \xi_a \in \mathscr{P}(\mathit{ThrdID})$ such that $wb = \mathbf{false}$ and $\mathbb{B} = \mathbf{true}$ and

$$\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (\Pi, C, (\sigma_c, \sigma, \circ)) \lesssim (\Pi', \Gamma, \mathbb{B}, \mathbb{C}, (\Sigma, \circ)) \diamond (\mathbb{M}, M, wb, aw) \Downarrow_{\xi, \xi_a} P.$$

Here $\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (\Pi, C, (\sigma_c, \sigma, \kappa)) \lesssim (\Pi', \Gamma, \mathbb{B}, \mathbb{C}, (\Sigma, \Bbbk)) \diamond (\mathbb{M}, M, wb, aw) \Downarrow_{\xi, \xi_a} Q$ is co-inductively defined as follows.

Whenever $\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (\Pi, C, (\sigma_c, \sigma, \kappa)) \precsim (\Pi', \Gamma, \mathbb{B}, \mathbb{C}, (\Sigma, \Bbbk)) \diamond (\mathbb{M}, M, wb, aw) \Downarrow_{\xi, \xi_a} Q$ holds, then the following hold:

(1) (a) Suppose $\sigma = (s, h)$. Then $\xi \cup \xi_a \subseteq s(\mathtt{TIDS})$ and $\mathsf{t} \notin \xi$ and $\mathsf{t} \notin \xi_a$.
(b) For any $\mathsf{t}' \in \xi \cup \xi_a$, we have $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}'}) * \mathsf{true}$.
(c) If $wb = \textbf{false}$, then $\xi = \emptyset$. If $wb = \textbf{true}$, then $\xi \neq \emptyset \vee (\Sigma, \Bbbk) \models \neg\mathbb{B}$.
(d) If $(\sigma, \kappa) \models \neg\mathsf{en}(C)$ and $(\Sigma, \Bbbk) \models \mathbb{B}$ and $\kappa \neq \circ$, then $\xi_a \neq \emptyset$.
(e) If $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}}) * \mathsf{true}$ and $\kappa \neq \circ$, then $(\sigma, \kappa) \models \mathsf{en}(C)$.
(f) If $\neg\mathsf{inObj}(C, \kappa)$, then $wb = \textbf{false}$.
(g) $\kappa = \circ \Longrightarrow C = \mathbb{C}$, and $\kappa = \circ \Longleftrightarrow \Bbbk = \circ$, and $C = \textbf{skip} \Longleftrightarrow \mathbb{C} = \textbf{skip}$.
(h) $\Sigma(\mathsf{done}) = \textbf{false}$.
(i) If $(\Sigma, \Bbbk) \models \neg\mathbb{B}$ and $\Bbbk \neq \circ$, then $\mathsf{is\_while0}(\mathbb{C}, \mathbb{B})$.
If $\mathsf{inObj}(C, \kappa)$, then $\mathsf{is\_while0}(\mathbb{C}, \mathbb{B}) \vee \mathsf{is\_while1}(\mathbb{C})$.

(2) If $\neg\mathsf{inObj}(C, \kappa)$, then $(\sigma, \Sigma) \models Q_{\mathsf{t}}$.

(3) For any σ_F and Σ_F, if $(C, (\sigma_c, \sigma \uplus \sigma_F, \kappa)) \xrightarrow{e}_{\mathsf{t},\Pi} \textbf{abort}$ and $\Sigma \perp \Sigma_F$, then $e = (\mathsf{t}, \textbf{clt}, \textbf{abort})$ and $(\mathbb{C}, (\sigma_c, \Sigma \uplus \Sigma_F, \Bbbk)) \xrightarrow{e}_{\mathsf{t},\Pi'} \textbf{abort}$.

(4) For any C', σ_c', σ'', κ', σ_F and Σ_F, if $(C, (\sigma_c, \sigma \uplus \sigma_F, \kappa)) \xrightarrow{e}_{\mathsf{t},\Pi} (C', (\sigma_c', \sigma'', \kappa'))$ and $\Sigma \perp \Sigma_F$, then there exist n, σ', $\mathcal{E}$, $\mathbb{C}'$, Σ', $\Bbbk'$, k, $\mathbb{M}'$, M', wb', aw', ξ', ξ_a' and $\mathbb{B}'$ such that
(a) $\sigma'' = \sigma' \uplus \sigma_F$, and
(b) $(\mathbb{C}, (\sigma_c, \Sigma \uplus \Sigma_F, \Bbbk)) \xrightarrow{\mathcal{E}}{}^{n}_{\mathsf{t},\Pi'} (\mathbb{C}', (\sigma_c', \Sigma' \uplus \Sigma_F, \Bbbk'))$; and
if $k > 0$ and $\mathsf{inObj}(C, \kappa)$, then
$\mathsf{exec0}_{\mathsf{t}}((\mathbb{C}, (\sigma_c, \Sigma \uplus \Sigma_F, \Bbbk)), (\mathbb{C}', (\sigma_c', \Sigma' \uplus \Sigma_F, \Bbbk'))) \vee \mathsf{exec1}_{\mathsf{t}}((\mathbb{C}, (\sigma_c, \Sigma \uplus \Sigma_F, \Bbbk)), (\mathbb{C}', (\sigma_c', \Sigma' \uplus \Sigma_F, \Bbbk')))$; and
if $e = (\mathsf{t}, f, \_)$, then $\mathbb{B}' = \mathsf{getB}(\Gamma, f)$, else if $\mathsf{inObj}(C, \kappa) \wedge \mathsf{is\_while1}(\mathbb{C}')$, then $\mathbb{B}' = \textbf{true}$, else $\mathbb{B}' = \mathbb{B}$; and
(c) $\mathsf{is\_clt}(e) \vee \mathsf{is\_inv}(e) \vee \mathsf{is\_ret}(e) \Longrightarrow \mathcal{E} = e :: \epsilon$, and
$\mathsf{is\_obj}(e) \Longrightarrow (\forall i, e' = \mathcal{E}(i).\ \mathsf{is\_obj}(e'))$, and
(d) $\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (\Pi, C', (\sigma_c', \sigma', \kappa')) \precsim (\Pi', \Gamma, \mathbb{B}', \mathbb{C}', (\Sigma', \Bbbk')) \diamond (\mathbb{M}', M', wb', aw') \Downarrow_{\xi', \xi_a'} Q$, and
(e) $((\sigma, \Sigma), (\sigma', \Sigma'), k) \models G_{\mathsf{t}} * [\mathsf{done} = \mathsf{false}] * \mathsf{True}$, and
(f) if $\mathsf{inObj}(C, \kappa)$, then $n > 0$, or $\mathbb{M}' < \mathbb{M}$, or $\mathbb{M}' = \mathbb{M}$ and $wb' = wb = \textbf{true}$ and $\xi \subseteq \xi'$; and
(g) if $((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle [\mathcal{D}_{\mathsf{t}}] \rangle * \mathsf{True}$ and $k = 0$, then $M' < M$.

(5) For any k, σ' and Σ', if $((\sigma, \Sigma), (\sigma', \Sigma'), k) \models R_{\mathsf{t}} * [\mathsf{done} = \mathsf{false}] * \mathsf{Id}$, then there exist $\mathbb{M}'$, M', wb', aw', ξ_d, ξ_{ad}, ξ' and ξ_a' such that
(a) $\mathcal{D}, R, G \models^{\chi}_{\mathsf{t}} (\Pi, C, (\sigma_c', \sigma', \kappa)) \precsim (\Pi', \Gamma, \mathbb{B}, \mathbb{C}, (\Sigma', \Bbbk)) \diamond (\mathbb{M}', M', wb', aw') \Downarrow_{\xi', \xi_a'} Q$, and
(b) if $k > 0$ and $\mathsf{inObj}(C, \kappa)$, then for any σ_c'', Σ'' and Σ_F such that $\Sigma'' = \Sigma\{\mathsf{done} \rightsquigarrow \textbf{true}\}$ and $\Sigma_F \perp \Sigma$ we have $(\mathbb{C}, (\sigma_c'', \Sigma'' \uplus \Sigma_F, \Bbbk)) \longrightarrow^{+}_{\mathsf{t},\Pi'} (\mathbb{C}, (\sigma_c'', \Sigma'' \uplus \Sigma_F, \Bbbk))$;
(c) $\xi_d = \{\mathsf{t}' \mid (\mathsf{t}' \in \xi) \wedge (((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle \mathcal{D}_{\mathsf{t}'} \rangle * \mathsf{Id})\}$ and
$k = 0 \wedge \mathsf{inObj}(C, \kappa) \Longrightarrow \mathbb{M}' < \mathbb{M} \vee (\mathbb{M}' = \mathbb{M} \wedge wb' = wb)$ and
$k = 0 \wedge wb = \textbf{true} \wedge (\xi_d \neq \emptyset \vee ((\Sigma, \Bbbk) \models \neg\mathbb{B} \wedge (\Sigma', \Bbbk) \models \mathbb{B})) \Longrightarrow \mathbb{M}' < \mathbb{M}$ and
$k = 0 \wedge \mathbb{M}' = \mathbb{M} \wedge wb' = wb = \textbf{true} \Longrightarrow \xi \backslash \xi_d \subseteq \xi'$, and
(d) if $(\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}}) * \mathsf{true}$ and $k = 0$, then $M' \leq M$; and
(e) $\xi_{ad} = \{\mathsf{t}' \mid (\mathsf{t}' \in \xi_a) \wedge (((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle \mathcal{D}_{\mathsf{t}'} \rangle * \mathsf{Id})\}$ and
$k = 0 \wedge \mathsf{inObj}(C, \kappa) \wedge \mathsf{is\_await}(C) \Longrightarrow \xi_a \backslash \xi_{ad} \subseteq \xi_a'$ and
$k = 0 \wedge \mathsf{inObj}(C, \kappa) \wedge \mathsf{is\_await}(C) \wedge (\xi_{ad} \neq \emptyset \vee ((\Sigma, \Bbbk) \models \neg\mathbb{B} \wedge (\Sigma', \Bbbk) \models \mathbb{B})) \Longrightarrow aw' < aw$ and
$(\chi = \mathsf{sfair}) \wedge k = 0 \wedge \mathsf{inObj}(C, \kappa) \wedge \mathsf{is\_await}(C) \wedge ((\sigma, \kappa) \models \neg\mathsf{en}(C)) \wedge ((\sigma', \kappa) \models \neg\mathsf{en}(C)) \Longrightarrow$

$aw' \le aw$ and
$(\chi = \text{wfair}) \wedge k = 0 \wedge \text{inObj}(C,\kappa) \wedge \text{is\_await}(C) \Longrightarrow aw' \le aw$.

Lemma B.18 (Lifting). *Suppose* $dom(\Pi) = dom(\Gamma) = dom(\Pi')$ *and* $P \Rightarrow \neg\text{Enabled}(\mathcal{D})$. *Then both the following hold:*

(1) If $\mathcal{D},R,G \models_\chi \{P\}\Pi \lesssim \Gamma$, *then*
for any C, *we have* $\mathcal{D},R,G \models_\chi \{P\}(\Pi,C) \lesssim (\Gamma,C)$.
(2) If $\mathcal{D},R,G \models\!\!\models_\chi \{P\}\Pi \lesssim (\Gamma,\text{wr}^*(\Gamma))$, *then*
for any C, *we have* $\mathcal{D},R,G \models\!\!\models_\chi \{P\}(\Pi,C) \lesssim (\text{wr}^*(\Gamma),\Gamma,C)$.

Proof. By structural induction over C and by co-induction. □

B.7 Whole-Program Simulations and Parallel Compositionality

Similar to Definitions B.16 and B.17 that give the client simulations with PSF and PDF objects respectively, we also need two definitions (see Definition B.19 and B.20) for whole programs with PSF and PDF objects respectively.

We use $\text{inObjThrds}(W,\mathcal{S})$ to get the set of threads who are executing the object methods. We also use $\text{bset}(\mathbb{W},\mathbb{S})$ to get the set of threads who are blocked. For $\mathscr{B} \in \mathit{ThrdID} \rightharpoonup \mathit{BExp}$, we use $\text{ffset}(\mathscr{B},\mathcal{S})$ to get the set of threads whose boolean condition in $\mathscr{B}$ is false at the state $\mathcal{S}$. They are defined as follows.

$$(\textbf{let } \Pi \textbf{ in } \hat{C}_1 \,\|\, \ldots \hat{C}_\mathsf{t} \ldots \|\, \hat{C}_n)|_\mathsf{t} \stackrel{\text{def}}{=} \hat{C}_\mathsf{t}$$

$$\text{activeThrds}(W) \stackrel{\text{def}}{=} \{\mathsf{t} \mid \exists \hat{C}.\ (W|_\mathsf{t} = \hat{C}) \wedge (\hat{C} \neq \textbf{skip}) \wedge (\hat{C} \neq \textbf{end})\}$$

$$\text{tidset}(\mathcal{E}) \stackrel{\text{def}}{=} \begin{cases} \emptyset & \text{if } \mathcal{E} = \epsilon \\ \{\text{tid}(e)\} \cup \text{tidset}(\mathcal{E}') & \text{if } \mathcal{E} = e :: \mathcal{E}' \end{cases}$$

$$\text{inObjThrds}(W,\mathcal{S}) \stackrel{\text{def}}{=} \{\mathsf{t} \mid \exists C,\sigma_c,\sigma,\mathcal{K}.\ (\mathcal{S} = (\sigma_c,\sigma,\mathcal{K})) \wedge (W|_\mathsf{t} = C) \wedge \text{inObj}(C,\mathcal{K}(\mathsf{t}))\}$$

$$\text{bset}(W,\mathcal{S}) \stackrel{\text{def}}{=} \Delta_c \cup \Delta_o \qquad \text{if } \text{btids}(W,\mathcal{S}) = (\Delta_c,\Delta_o)$$

$$\text{ffset}(\mathscr{B},\mathcal{S}) \stackrel{\text{def}}{=} \{\mathsf{t} \mid \exists \sigma_c,\sigma,\mathcal{K}.\ (\mathcal{S} = (\sigma_c,\sigma,\mathcal{K})) \wedge (\mathcal{K}(\mathsf{t}) \neq \circ) \wedge (\sigma,\mathcal{K}(\mathsf{t})) \models \neg\mathscr{B}(\mathsf{t})\}$$

Definition B.19 (Simulation for whole programs with PSF objects). $\models_\chi \{P\}W \lesssim \mathbb{W}$ iff, for any σ_c, σ and Σ, if $(\sigma,\Sigma) \models P$, there exist $\mathscr{M},\alpha \in \mathit{ThrdID} \rightharpoonup \mathit{Metric}$, $\beta \in \mathit{ThrdID} \rightharpoonup \mathit{Bool}$ and $\zeta,\zeta_a \in \mathit{ThrdID} \rightharpoonup \mathscr{P}(\mathit{ThrdID})$ such that

$$\models_\chi (W,(\sigma_c,\sigma,\circledcirc)) \lesssim (\mathbb{W},(\sigma_c,\Sigma,\circledcirc)) \diamond (\mathscr{M},\zeta,\beta,\alpha,\zeta_a).$$

Here $\models_\chi (W,\mathcal{S}) \lesssim (\mathbb{W},\mathbb{S}) \diamond (\mathscr{M},\zeta,\beta,\alpha,\zeta_a)$ is co-inductively defined as follows. Whenever $\models_\chi (W,\mathcal{S}) \lesssim (\mathbb{W},\mathbb{S}) \diamond (\mathscr{M},\zeta,\beta,\alpha,\zeta_a)$ holds, then the following hold:

(1)(a) $dom(\mathscr{M}) = dom(\zeta) = dom(\beta) = dom(\alpha) = dom(\zeta_a) = \text{inObjThrds}(W,\mathcal{S}) = \text{inObjThrds}(\mathbb{W},\mathbb{S})$.
(b) For any $\mathsf{t} \in \text{inObjThrds}(W,\mathcal{S})$, we have $\zeta(\mathsf{t}) \cup \zeta_a(\mathsf{t}) \subseteq (\text{inObjThrds}(W,\mathcal{S}) \backslash \{\mathsf{t}\})$.
(c) For any $\mathsf{t} \in \text{inObjThrds}(W,\mathcal{S})$, if $\beta(\mathsf{t}) = \textbf{false}$, then $\zeta(\mathsf{t}) = \emptyset$; and if $\beta(\mathsf{t}) = \textbf{true}$, then $\zeta(\mathsf{t}) \neq \emptyset \vee \mathsf{t} \in \text{bset}(\mathbb{W},\mathbb{S})$.
(d) For any $\mathsf{t} \in \text{inObjThrds}(W,\mathcal{S})$, if $\mathsf{t} \in \text{bset}(W,\mathcal{S})$ and $\mathsf{t} \notin \text{bset}(\mathbb{W},\mathbb{S})$, then $\zeta_a(\mathsf{t}) \neq \emptyset$.
(e) For any $\mathsf{t} \in \text{inObjThrds}(W,\mathcal{S})$, for any $\mathsf{t}' \in \zeta(\mathsf{t}) \cup \zeta_a(\mathsf{t})$, we have $\mathsf{t}' \notin \text{bset}(W,\mathcal{S})$.
(2) If $W = (\textbf{let } \Pi \textbf{ in end} \,\|\, \ldots \|\, \textbf{end})$, then $\exists \Pi'.\ \mathbb{W} = (\textbf{let } \Pi' \textbf{ in end} \,\|\, \ldots \|\, \textbf{end})$.
(3) If $(W,\mathcal{S}) \stackrel{\iota}{\longmapsto} \textbf{abort}$, then
there exists t such that $\iota = ((\mathsf{t},\textbf{clt},\textbf{abort}),\emptyset,\emptyset)$ and $(\mathbb{W},\mathbb{S}) \stackrel{\iota}{\longmapsto} \textbf{abort}$.
(4) If $(W,\mathcal{S}) \stackrel{\iota}{\longmapsto} (W',\mathcal{S}')$, then
there exist t, T, $\mathbb{W}'$, $\mathbb{S}'$, $\mathscr{M}'$, ζ', β', α', ζ_a', n, e, Δ_c and Δ_o such that all the following hold:

(a) $(\mathbb{W},\mathbb{S}) \overset{T}{\longmapsto}^{n} (\mathbb{W}',\mathbb{S}')$, and $n = 0 \vee n = 1$;
(b) $\iota = (e,\Delta_c,\Delta_o)$; $\mathsf{t} = \mathsf{tid}(e)$;
$\mathsf{is\_clt}(e) \vee \mathsf{is\_inv}(e) \vee \mathsf{is\_ret}(e) \Longrightarrow T = (e,\Delta_c,\_)::\epsilon$;
$\mathsf{is\_obj}(e) \Longrightarrow (\forall i,\iota' = T(i).\ \mathsf{is\_obj}(\iota'))$;
(c) $\models_\chi (W',\mathcal{S}') \lesssim (\mathbb{W}',\mathbb{S}') \diamond (\mathscr{M}',\zeta',\beta',\alpha',\zeta'_a)$;
(d) if $\mathsf{t} \in \mathsf{inObjThrds}(W,\mathcal{S})$, then
either $\mathsf{t} \in \mathsf{tidset}(T)$,
or $\mathscr{M}'(\mathsf{t}) < \mathscr{M}(\mathsf{t})$,
or $\mathscr{M}'(\mathsf{t}) = \mathscr{M}(\mathsf{t})$ and $\beta'(\mathsf{t}) = \beta(\mathsf{t}) = \textbf{true}$ and $\zeta(\mathsf{t}) \subseteq \zeta'(\mathsf{t}) \subseteq (\mathsf{inObjThrds}(W,\mathcal{S})\backslash\{\mathsf{t}\})$;
(e) for any $\mathsf{t}' \in \mathsf{inObjThrds}(W,\mathcal{S})\backslash\{\mathsf{t}\}$, we have:
either $\mathscr{M}'(\mathsf{t}') < \mathscr{M}(\mathsf{t}')$,
or $\mathscr{M}'(\mathsf{t}') = \mathscr{M}(\mathsf{t}')$ and $\beta'(\mathsf{t}') = \beta(\mathsf{t}') = \textbf{false}$,
or $\mathscr{M}'(\mathsf{t}') = \mathscr{M}(\mathsf{t}')$ and $\beta'(\mathsf{t}') = \beta(\mathsf{t}') = \textbf{true}$ and $\zeta(\mathsf{t}') \subseteq \zeta'(\mathsf{t}') \subseteq (\mathsf{inObjThrds}(W,\mathcal{S})\backslash\{\mathsf{t}'\})$
and $\mathsf{t} \notin \zeta(\mathsf{t}')$;
(f) for any $\mathsf{t}' \in \mathsf{inObjThrds}(W,\mathcal{S})\backslash\{\mathsf{t}\}$ and $\mathsf{is\_await}(W|_{\mathsf{t}'})$, we have:
if $((\chi = \mathsf{sfair}) \wedge (\mathsf{t}' \in \mathsf{bset}(W,\mathcal{S})) \wedge (\mathsf{t}' \in \mathsf{bset}(W',\mathcal{S}'))) \vee (\chi = \mathsf{wfair})$, then
either $\alpha'(\mathsf{t}') < \alpha(\mathsf{t}')$,
or $\alpha'(\mathsf{t}') = \alpha(\mathsf{t}')$ and $\zeta_a(\mathsf{t}') \subseteq \zeta'_a(\mathsf{t}') \subseteq (\mathsf{inObjThrds}(W,\mathcal{S})\backslash\{\mathsf{t}'\})$ and $\mathsf{t} \notin \zeta_a(\mathsf{t}')$ and
$(\zeta_a(\mathsf{t}') \neq \emptyset) \vee (\zeta_a(\mathsf{t}') = \emptyset \wedge \mathsf{t}' \notin \mathsf{bset}(\mathbb{W},\mathbb{S})) \vee (\zeta_a(\mathsf{t}') = \emptyset \wedge \mathsf{t}' \in \mathsf{bset}(\mathbb{W}',\mathbb{S}'))$.

Definition B.20 (Simulation for whole programs with PDF objects). $\models_\chi \{P\}W \lesssim (\mathbb{W},\Gamma)$ iff, for any σ_c, σ and Σ, if $(\sigma,\Sigma) \models P$, there exist $\mathscr{B} \in \mathit{ThrdID} \rightharpoonup \mathit{BExp}$, $\mathscr{M},\alpha \in \mathit{ThrdID} \rightharpoonup \mathit{Metric}$, $\beta \in \mathit{ThrdID} \rightharpoonup \mathit{Bool}$, and $\zeta,\zeta_a \in \mathit{ThrdID} \rightharpoonup \mathscr{P}(\mathit{ThrdID})$ such that $\mathscr{B} = \emptyset$ and

$$\models_\chi (W,(\sigma_c,\sigma,\circledcirc)) \lesssim (\mathbb{W},\Gamma,\mathscr{B},(\sigma_c,\Sigma,\circledcirc)) \diamond (\mathscr{M},\zeta,\beta,\alpha,\zeta_a).$$

Here $\models_\chi (W,\mathcal{S}) \lesssim (\mathbb{W},\Gamma,\mathscr{B},\mathbb{S}) \diamond (\mathscr{M},\zeta,\beta,\alpha,\zeta_a)$ is co-inductively defined as follows. Whenever $\models_\chi (W,\mathcal{S}) \lesssim (\mathbb{W},\Gamma,\mathscr{B},\mathbb{S}) \diamond (\mathscr{M},\zeta,\beta,\alpha,\zeta_a)$ holds, then the following hold:

(1)(a) $\mathit{dom}(\mathscr{M}) = \mathit{dom}(\zeta) = \mathit{dom}(\beta) = \mathit{dom}(\alpha) = \mathit{dom}(\zeta_a) = \mathit{dom}(\mathscr{B}) = \mathsf{inObjThrds}(W,\mathcal{S}) = \mathsf{inObjThrds}(\mathbb{W},\mathbb{S})$.
(b) For any $\mathsf{t} \in \mathsf{inObjThrds}(W,\mathcal{S})$, we have $\zeta(\mathsf{t}) \cup \zeta_a(\mathsf{t}) \subseteq (\mathsf{inObjThrds}(W,\mathcal{S})\backslash\{\mathsf{t}\})$.
(c) For any $\mathsf{t} \in \mathsf{inObjThrds}(W,\mathcal{S})$, if $\beta(\mathsf{t}) = \textbf{false}$, then $\zeta(\mathsf{t}) = \emptyset$; and if $\beta(\mathsf{t}) = \textbf{true}$, then $\zeta(\mathsf{t}) \neq \emptyset \vee \mathsf{t} \in \mathsf{ffset}(\mathscr{B},\mathbb{S})$.
(d) For any $\mathsf{t} \in \mathsf{inObjThrds}(W,\mathcal{S})$, if $\mathsf{t} \in \mathsf{bset}(W,\mathcal{S})$ and $\zeta_a(\mathsf{t}) = \emptyset$, then $\mathsf{t} \in \mathsf{ffset}(\mathscr{B},\mathbb{S})$.
(e) For any $\mathsf{t} \in \mathsf{inObjThrds}(W,\mathcal{S})$, for any $\mathsf{t}' \in \zeta(\mathsf{t}) \cup \zeta_a(\mathsf{t})$, we have $\mathsf{t}' \notin \mathsf{bset}(W,\mathcal{S})$.
(f) For any $\mathsf{t} \in \mathsf{ffset}(\mathscr{B},\mathbb{S})$, we have $\mathsf{is\_while0}(\mathbb{W}|_{\mathsf{t}},\mathscr{B}(\mathsf{t}))$.
For any $\mathsf{t} \in \mathsf{inObjThrds}(W,\mathcal{S})$, we have $\mathsf{is\_while0}(\mathbb{W}|_{\mathsf{t}},\mathscr{B}(\mathsf{t})) \vee \mathsf{is\_while1}(\mathbb{W}|_{\mathsf{t}})$.
(2) If $W = (\textbf{let}\ \Pi\ \textbf{in end} \parallel \ldots \parallel \textbf{end})$, then $\exists \Pi'.\ \mathbb{W} = (\textbf{let}\ \Pi'\ \textbf{in end} \parallel \ldots \parallel \textbf{end})$.
(3) If $(W,\mathcal{S}) \overset{\iota}{\longmapsto} \textbf{abort}$, then
there exists t such that $\iota = ((\mathsf{t},\textbf{clt},\textbf{abort}),\emptyset,\emptyset)$ and $(\mathbb{W},\mathbb{S}) \overset{\iota}{\longmapsto} \textbf{abort}$.
(4) If $(W,\mathcal{S}) \overset{\iota}{\longmapsto} (W',\mathcal{S}')$, then
there exist t, T, $\mathbb{W}'$, $\mathbb{S}'$, $\mathscr{M}'$, ζ', β', α', ζ'_a, $\mathscr{B}'$, e, Δ_c and Δ_o such that all the following hold:
(a) $(\mathbb{W},\mathbb{S}) \overset{T}{\longmapsto}^{*} (\mathbb{W}',\mathbb{S}')$;
(b) $\iota = (e,\Delta_c,\Delta_o)$; $\mathsf{t} = \mathsf{tid}(e)$;
$\mathsf{is\_clt}(e) \vee \mathsf{is\_inv}(e) \vee \mathsf{is\_ret}(e) \Longrightarrow T = (e,\Delta_c,\_)::\epsilon$;
$\mathsf{is\_obj}(e) \Longrightarrow (\forall i,\iota' = T(i).\ \mathsf{is\_obj}(\iota'))$;
if $e = (\mathsf{t},f,\_)$, then $\mathscr{B}' = \mathscr{B}\{\mathsf{t} \rightsquigarrow \mathsf{getB}(\Gamma,f)\}$, else if $\mathsf{is\_obj}(e) \wedge \mathsf{is\_while1}(\mathbb{W}'|_{\mathsf{t}})$, then $\mathscr{B}' = \mathscr{B}\{\mathsf{t} \rightsquigarrow \textbf{true}\}$, else $\mathscr{B}' = \mathscr{B}$;
(c) $\models_\chi (W',\mathcal{S}') \lesssim (\mathbb{W}',\Gamma,\mathscr{B}',\mathbb{S}') \diamond (\mathscr{M}',\zeta',\beta',\alpha',\zeta'_a)$;

(d) if $\mathsf{t} \in \mathsf{inObjThrds}(W, \mathcal{S})$, then
either $\mathsf{t} \in \mathsf{tidset}(T)$,
or $\mathscr{M}'(\mathsf{t}) < \mathscr{M}(\mathsf{t})$,
or $\mathscr{M}'(\mathsf{t}) = \mathscr{M}(\mathsf{t})$ and $\beta'(\mathsf{t}) = \beta(\mathsf{t}) = \textbf{true}$ and $\zeta(\mathsf{t}) \subseteq \zeta'(\mathsf{t}) \subseteq (\mathsf{inObjThrds}(W, \mathcal{S}) \backslash \{\mathsf{t}\})$;
(e) for any $\mathsf{t}' \in \mathsf{inObjThrds}(W, \mathcal{S}) \backslash \{\mathsf{t}\}$, we have:
either $\mathsf{t}' \in \mathsf{tidset}(T)$,
or $\mathscr{M}'(\mathsf{t}') < \mathscr{M}(\mathsf{t}')$,
or $\mathscr{M}'(\mathsf{t}') = \mathscr{M}(\mathsf{t}')$ and $\beta'(\mathsf{t}') = \beta(\mathsf{t}') = \textbf{false}$,
or $\mathscr{M}'(\mathsf{t}') = \mathscr{M}(\mathsf{t}')$ and $\beta'(\mathsf{t}') = \beta(\mathsf{t}') = \textbf{true}$ and $\zeta(\mathsf{t}') \subseteq \zeta'(\mathsf{t}') \subseteq (\mathsf{inObjThrds}(W, \mathcal{S}) \backslash \{\mathsf{t}'\})$
and $\mathsf{t} \notin \zeta(\mathsf{t}')$;
(f) for any $\mathsf{t}' \in \mathsf{inObjThrds}(W, \mathcal{S}) \backslash \{\mathsf{t}\}$ and $\mathsf{is\_await}(W|_{\mathsf{t}'})$, we have:
if $((\chi = \mathsf{sfair}) \wedge (\mathsf{t}' \in \mathsf{bset}(W, \mathcal{S})) \wedge (\mathsf{t}' \in \mathsf{bset}(W', \mathcal{S}'))) \vee (\chi = \mathsf{wfair})$, then
either $\mathsf{t}' \in \mathsf{tidset}(T)$,
or $\alpha'(\mathsf{t}') < \alpha(\mathsf{t}')$,
or $\alpha'(\mathsf{t}') = \alpha(\mathsf{t}')$ and $\zeta_a(\mathsf{t}') \subseteq \zeta'_a(\mathsf{t}') \subseteq (\mathsf{inObjThrds}(W, \mathcal{S}) \backslash \{\mathsf{t}'\})$ and $\mathsf{t} \notin \zeta_a(\mathsf{t}')$ and
$(\zeta_a(\mathsf{t}') \neq \emptyset) \vee (\zeta_a(\mathsf{t}') = \emptyset \wedge \mathsf{t}' \notin \mathsf{ffset}(\mathscr{B}, \mathbb{S})) \vee (\zeta_a(\mathsf{t}') = \emptyset \wedge \mathsf{t}' \in \mathsf{ffset}(\mathscr{B}', \mathbb{S}'))$.

Lemma B.21 (Parallel Compositionality for Simulations with PSF Objects).
If there exist R, G and $\mathcal{D}$ such that the following hold:

(1) for any $\mathsf{t} \in [1..n]$, we have $\mathcal{D}, R, G \models_\chi \{P\}(\Pi, C_\mathsf{t}) \precsim (\Gamma, \mathbb{C}_\mathsf{t})$;
(2) $\forall \mathsf{t}, \mathsf{t}'.\ \mathsf{t} \neq \mathsf{t}' \Longrightarrow G_\mathsf{t} \Rightarrow R_{\mathsf{t}'}$, $\mathit{wffAct}(R, \mathcal{D})$, $P \Rightarrow \neg\mathsf{Enabled}(\mathcal{D})$, $P \vee \mathsf{Enabled}(\mathcal{D}) \Rightarrow I$, $I \triangleright \{R, G\}$,
then $\models_\chi \{\bigwedge_{\mathsf{t} \in [1..n]} P_\mathsf{t}\}(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n) \precsim (\textbf{let } \Gamma \textbf{ in } \mathbb{C}_1 \,\|\, \ldots \,\|\, \mathbb{C}_n)$.

Proof. For any σ_c, σ and Σ, if $(\sigma, \Sigma) \models (\bigwedge_{\mathsf{t} \in [1..n]} P_\mathsf{t})$, from the premises, we know: there exist $\mathbb{M}_1, \ldots, \mathbb{M}_n$, $M_1, \ldots, M_n$, $aw_1, \ldots, aw_n$, $wb_1, \ldots, wb_n$, $\xi_1, \ldots, \xi_n$, $\xi_1^a, \ldots, \xi_n^a$ such that $wb_1 = \ldots = wb_n = \textbf{false}$ and for any $\mathsf{t} \in [1..n]$:

$$\mathcal{D}, R, G \models_\mathsf{t}^\chi (\Pi, C_\mathsf{t}, (\sigma_c, \sigma, \circ)) \precsim (\Gamma, \mathbb{C}_\mathsf{t}, (\Sigma, \circ)) \diamond (\mathbb{M}_\mathsf{t}, M_\mathsf{t}, wb_\mathsf{t}, aw_\mathsf{t}) \Downarrow_{\xi_\mathsf{t}, \xi_\mathsf{t}^a} P.$$

We want to show that there exist $\mathscr{M}$, α, β, ζ and ζ_a such that

$$\models_\chi ((\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma, \circledcirc)) \precsim ((\textbf{let } \Gamma \textbf{ in } \mathbb{C}_1 \,\|\, \ldots \,\|\, \mathbb{C}_n), (\sigma_c, \Sigma, \circledcirc)) \diamond (\mathscr{M}, \zeta, \beta, \alpha, \zeta_a).$$

We generalize the result and prove the following (B.4):

If $(\sigma, \Sigma) \models I$ and the following holds for any $\mathsf{t} \in [1..n]$:

$$\hat{C}_\mathsf{t} \neq \textbf{end} \Longrightarrow \mathcal{D}, R, G \models_\mathsf{t}^\chi (\Pi, \hat{C}_\mathsf{t}, (\sigma_c, \sigma \uplus \sigma_\mathsf{t}, \kappa_\mathsf{t})) \precsim (\Gamma, \hat{\mathbb{C}}_\mathsf{t}, (\Sigma \uplus \Sigma_\mathsf{t}, \mathbb{k}_\mathsf{t})) \diamond (\mathbb{M}_\mathsf{t}, M_\mathsf{t}, wb_\mathsf{t}, aw_\mathsf{t}) \Downarrow_{\xi_\mathsf{t}, \xi_\mathsf{t}^a} P,$$
$$\hat{C}_\mathsf{t} = \textbf{end} \Longrightarrow \hat{\mathbb{C}}_\mathsf{t} = \textbf{end} \wedge \kappa_\mathsf{t} = \circ \wedge \mathbb{k}_\mathsf{t} = \circ,$$

then

$$\models_\chi ((\textbf{let } \Pi \textbf{ in } \hat{C}_1 \,\|\, \ldots \,\|\, \hat{C}_n), (\sigma_c, \sigma \uplus (\biguplus_\mathsf{t} \sigma_\mathsf{t}), \mathcal{K})) \precsim ((\textbf{let } \Gamma \textbf{ in } \hat{\mathbb{C}}_1 \,\|\, \ldots \,\|\, \hat{\mathbb{C}}_n), (\sigma_c, \Sigma \uplus (\biguplus_\mathsf{t} \Sigma_\mathsf{t}), \mathbb{K})) \diamond (\mathscr{M}, \zeta, \beta, \alpha, \zeta_a).$$

Here for any $\mathsf{t} \in [1..n]$, $\mathcal{K}(\mathsf{t}) = \kappa_\mathsf{t}$ and $\mathbb{K}(\mathsf{t}) = \mathbb{k}_\mathsf{t}$, and the functions $\mathscr{M}$, ζ, β, α and ζ_a are defined as follows.

- $dom(\mathscr{M}) = dom(\zeta) = dom(\beta) = dom(\alpha) = dom(\zeta_a) = \mathsf{inObjThrds}((\textbf{let } \Pi \textbf{ in } \hat{C}_1 \,\|\, \ldots \,\|\, \hat{C}_n), (\sigma_c, \sigma \uplus (\biguplus_\mathsf{t} \sigma_\mathsf{t}), \mathcal{K}))$.
- For any $\mathsf{t} \in dom(\mathscr{M})$, we have
  - $\mathscr{M}(\mathsf{t}) = (\mathbb{M}_\mathsf{t}, \{\mathsf{t}' \rightsquigarrow M_{\mathsf{t}'} \mid \mathsf{t}' \in \xi_\mathsf{t}\})$;
  - $\zeta(\mathsf{t}) = \xi_\mathsf{t}$;

- $\beta(\mathrm{t}) = wb_{\mathrm{t}}$;
- $\alpha(\mathrm{t}) = (aw_{\mathrm{t}}, \{\mathrm{t}' \rightsquigarrow M_{\mathrm{t}'} \mid \mathrm{t}' \in \xi^a_{\mathrm{t}}\})$;
- $\zeta_a(\mathrm{t}) = \xi^a_{\mathrm{t}}$.

The order $(\mathbb{M}', \mathcal{M}') < (\mathbb{M}, \mathcal{M})$ is defined as a dictionary order:

$$(\mathbb{M}', \mathcal{M}') < (\mathbb{M}, \mathcal{M}) \text{ iff } (\mathbb{M}' < \mathbb{M}) \vee (\mathbb{M}' = \mathbb{M}) \wedge (\mathcal{M}' < \mathcal{M})$$

$$\mathcal{M}' < \mathcal{M} \text{ iff } \exists \mathrm{t}.\ (\mathcal{M}'(\mathrm{t}) < \mathcal{M}(\mathrm{t})) \wedge (\forall \mathrm{t}' \in dom(\mathcal{M}) \backslash \{\mathrm{t}\}.\ \mathcal{M}'(\mathrm{t}') \leq \mathcal{M}(\mathrm{t}'))$$

$$\mathcal{M}' \leq \mathcal{M} \text{ iff } \forall \mathrm{t} \in dom(\mathcal{M}).\ \mathcal{M}'(\mathrm{t}) \leq \mathcal{M}(\mathrm{t})$$

Clearly $\mathscr{M}'(\mathrm{t}) < \mathscr{M}(\mathrm{t})$ is a well-founded order. (B.4)

By co-induction. Let $W \stackrel{\text{def}}{=} (\textbf{let } \Pi \textbf{ in } \hat{C}_1 \parallel \ldots \parallel \hat{C}_n)$, $\mathbb{W} \stackrel{\text{def}}{=} (\textbf{let } \Gamma \textbf{ in } \hat{\mathbb{C}}_1 \parallel \ldots \parallel \hat{\mathbb{C}}_n)$, $\mathcal{S} \stackrel{\text{def}}{=} (\sigma_c, \sigma \uplus (\biguplus_{\mathrm{t}} \sigma_{\mathrm{t}}), \mathcal{K})$ and $\mathbb{S} \stackrel{\text{def}}{=} (\sigma_c, \Sigma \uplus (\biguplus_{\mathrm{t}} \Sigma_{\mathrm{t}}), \mathbb{K})$. Suppose $\sigma = (s, h)$. Then $s(\mathtt{TIDS}) = [1..n]$.

(1)(a) $dom(\mathscr{M}) = dom(\zeta) = dom(\beta) = dom(\alpha) = dom(\zeta_a) = \mathsf{inObjThrds}(W, \mathcal{S}) = \mathsf{inObjThrds}(\mathbb{W}, \mathbb{S})$.

(b) For any $\mathrm{t} \in \mathsf{inObjThrds}(W, \mathcal{S})$, we have $\zeta(\mathrm{t}) \cup \zeta_a(\mathrm{t}) \subseteq (\mathsf{inObjThrds}(W, \mathcal{S}) \backslash \{\mathrm{t}\})$.

(c) For any $\mathrm{t} \in \mathsf{inObjThrds}(W, \mathcal{S})$, if $\beta(\mathrm{t}) = \textbf{false}$, then $\zeta(\mathrm{t}) = \emptyset$; and if $\beta(\mathrm{t}) = \textbf{true}$, then $\zeta(\mathrm{t}) \neq \emptyset \vee \mathrm{t} \in \mathsf{bset}(\mathbb{W}, \mathbb{S})$.

(d) For any $\mathrm{t} \in \mathsf{inObjThrds}(W, \mathcal{S})$, if $\mathrm{t} \in \mathsf{bset}(W, \mathcal{S})$ and $\mathrm{t} \notin \mathsf{bset}(\mathbb{W}, \mathbb{S})$, then $\zeta_a(\mathrm{t}) \neq \emptyset$.

(e) For any $\mathrm{t} \in \mathsf{inObjThrds}(W, \mathcal{S})$, for any $\mathrm{t}' \in \zeta(\mathrm{t}) \cup \zeta_a(\mathrm{t})$, we have $\mathrm{t}' \notin \mathsf{bset}(W, \mathcal{S})$.

Proof: For any $\mathrm{t} \in [1..n]$, from the premise, we know: $\kappa_{\mathrm{t}} = \circ \iff \Bbbk_{\mathrm{t}} = \circ$. Thus we have $\mathsf{inObjThrds}(W, \mathcal{S}) = \mathsf{inObjThrds}(\mathbb{W}, \mathbb{S})$.

Also, from the premise, we know: for any $\mathrm{t}' \in \zeta(\mathrm{t}) \cup \zeta_a(\mathrm{t})$, we have $\mathrm{t}' \neq \mathrm{t}$ and $(\sigma \uplus \sigma_{\mathrm{t}'}, \Sigma \uplus \Sigma_{\mathrm{t}'}) \models \mathsf{Enabled}(\mathcal{D}_{\mathrm{t}'}) * \mathsf{true}$. Then we only need to prove $\mathsf{inObj}(C_{\mathrm{t}'}, \kappa_{\mathrm{t}'})$. Suppose $\neg\mathsf{inObj}(C_{\mathrm{t}'}, \kappa_{\mathrm{t}'})$, then we have $(\sigma \uplus \sigma_{\mathrm{t}'}, \Sigma \uplus \Sigma_{\mathrm{t}'}) \models P_{\mathrm{t}'}$. Since $P \Rightarrow \neg\mathsf{Enabled}(\mathcal{D})$, we get a contradiction. Thus $\mathsf{inObj}(C_{\mathrm{t}'}, \kappa_{\mathrm{t}'})$ holds. Also, from the premise, we know: $(\sigma \uplus \sigma_{\mathrm{t}'}, \Sigma \uplus \Sigma_{\mathrm{t}'}) \models \mathsf{en}(C_{\mathrm{t}'})$. Thus $\mathrm{t}' \notin \mathsf{bset}(W, \mathcal{S})$ holds.

(2) If $W = (\textbf{let } \Pi \textbf{ in end} \parallel \ldots \parallel \textbf{end})$, then $\exists \Pi'.\ \mathbb{W} = (\textbf{let } \Pi' \textbf{ in end} \parallel \ldots \parallel \textbf{end})$.

Proof: Since $\hat{C}_{\mathrm{t}} = \textbf{end} \Longrightarrow \hat{\mathbb{C}}_{\mathrm{t}} = \textbf{end}$, we are done.

(3) If $(W, \mathcal{S}) \stackrel{\iota}{\longmapsto} \textbf{abort}$, then

there exists t such that $\iota = ((\mathrm{t}, \textbf{clt}, \textbf{abort}), \emptyset, \emptyset)$ and $(\mathbb{W}, \mathbb{S}) \stackrel{\iota}{\longmapsto} \textbf{abort}$.

Proof: By the operational semantics we know: there exist $\mathrm{t} \in [1..n]$ and e such that $\iota = (e, \emptyset, \emptyset)$ and

$$(\hat{C}_{\mathrm{t}}, (\sigma_c, \sigma \uplus (\biguplus_{\mathrm{t}} \sigma_{\mathrm{t}}), \kappa_{\mathrm{t}})) \xrightarrow{e}_{\mathrm{t},\Pi} \textbf{abort}.$$

By $\mathcal{D}, R, G \models^{\chi}_{\mathrm{t}} (\Pi, \hat{C}_{\mathrm{t}}, (\sigma_c, \sigma \uplus \sigma_{\mathrm{t}}, \kappa_{\mathrm{t}})) \precsim (\Gamma, \hat{\mathbb{C}}_{\mathrm{t}}, (\Sigma \uplus \Sigma_{\mathrm{t}}, \Bbbk_{\mathrm{t}})) \diamond (\mathbb{M}_{\mathrm{t}}, M_{\mathrm{t}}, wb_{\mathrm{t}}, aw_{\mathrm{t}}) \Downarrow_{\xi_{\mathrm{t}}, \xi^a_{\mathrm{t}}} P$, we know $e = (\mathrm{t}, \textbf{clt}, \textbf{abort})$ and

$$(\hat{\mathbb{C}}_{\mathrm{t}}, (\sigma_c, \Sigma \uplus (\biguplus_{\mathrm{t}} \Sigma_{\mathrm{t}}), \Bbbk_{\mathrm{t}})) \xrightarrow{e}_{\mathrm{t},\Gamma} \textbf{abort}.$$

Thus $(\mathbb{W}, \mathbb{S}) \stackrel{\iota}{\longmapsto} \textbf{abort}$.

(4) If $(W, \mathcal{S}) \stackrel{\iota}{\longmapsto} (W', \mathcal{S}')$, then

there exist t, T, $\mathbb{W}'$, $\mathbb{S}'$, $\mathscr{M}'$, ζ', β', α', ζ'_a, n, e, Δ_c and Δ_o such that all the following hold:

(a) $(\mathbb{W}, \mathbb{S}) \stackrel{T}{\longmapsto}^n (\mathbb{W}', \mathbb{S}')$, and $n = 0 \vee n = 1$;

(b) $\iota = (e, \Delta_c, \Delta_o)$; $\mathrm{t} = \mathsf{tid}(e)$;
$\mathsf{is\_clt}(e) \vee \mathsf{is\_inv}(e) \vee \mathsf{is\_ret}(e) \Longrightarrow T = (e, \Delta_c, \_) :: \epsilon$;
$\mathsf{is\_obj}(e) \Longrightarrow (\forall i, \iota' = T(i).\ \mathsf{is\_obj}(\iota'))$;

(c) $\models_{\chi} (W', \mathcal{S}') \precsim (\mathbb{W}', \mathbb{S}') \diamond (\mathscr{M}', \zeta', \beta', \alpha', \zeta'_a)$;

(d) if $\mathrm{t} \in \mathsf{inObjThrds}(W, \mathcal{S})$, then
either $\mathrm{t} \in \mathsf{tidset}(T)$,

or $\mathscr{M}'(\mathrm{t}) < \mathscr{M}(\mathrm{t})$,
or $\mathscr{M}'(\mathrm{t}) = \mathscr{M}(\mathrm{t})$ and $\beta'(\mathrm{t}) = \beta(\mathrm{t}) = \mathbf{true}$ and $\zeta(\mathrm{t}) \subseteq \zeta'(\mathrm{t}) \subseteq (\mathsf{inObjThrds}(W,\mathcal{S})\backslash\{\mathrm{t}\})$;

(e) for any $\mathrm{t}' \in \mathsf{inObjThrds}(W,\mathcal{S})\backslash\{\mathrm{t}\}$, we have:
either $\mathscr{M}'(\mathrm{t}') < \mathscr{M}(\mathrm{t}')$,
or $\mathscr{M}'(\mathrm{t}') = \mathscr{M}(\mathrm{t}')$ and $\beta'(\mathrm{t}') = \beta(\mathrm{t}') = \mathbf{false}$,
or $\mathscr{M}'(\mathrm{t}') = \mathscr{M}(\mathrm{t}')$ and $\beta'(\mathrm{t}') = \beta(\mathrm{t}') = \mathbf{true}$ and $\zeta(\mathrm{t}') \subseteq \zeta'(\mathrm{t}') \subseteq (\mathsf{inObjThrds}(W,\mathcal{S})\backslash\{\mathrm{t}'\})$ and $\mathrm{t} \notin \zeta(\mathrm{t}')$;

(f) for any $\mathrm{t}' \in \mathsf{inObjThrds}(W,\mathcal{S})\backslash\{\mathrm{t}\}$ and $\mathsf{is\_await}(W|_{\mathrm{t}'})$, we have:
if $((\chi = \mathrm{sfair}) \wedge (\mathrm{t}' \in \mathsf{bset}(W,\mathcal{S})) \wedge (\mathrm{t}' \in \mathsf{bset}(W',\mathcal{S}'))) \vee (\chi = \mathrm{wfair})$, then
either $\alpha'(\mathrm{t}') < \alpha(\mathrm{t}')$,
or $\alpha'(\mathrm{t}') = \alpha(\mathrm{t}')$ and $\zeta_a(\mathrm{t}') \subseteq \zeta'_a(\mathrm{t}') \subseteq (\mathsf{inObjThrds}(W,\mathcal{S})\backslash\{\mathrm{t}'\})$ and $\mathrm{t} \notin \zeta_a(\mathrm{t}')$ and
$(\zeta_a(\mathrm{t}') \neq \emptyset) \vee (\zeta_a(\mathrm{t}') = \emptyset \wedge \mathrm{t}' \notin \mathsf{bset}(\mathbb{W},\mathbb{S})) \vee (\zeta_a(\mathrm{t}') = \emptyset \wedge \mathrm{t}' \in \mathsf{bset}(\mathbb{W}',\mathbb{S}'))$.

Proof: By the operational semantics, we know one of the two cases holds:

- there exist $\mathrm{t}, \hat{C}'_{\mathrm{t}}, W', \mathcal{S}', e, \Delta_c$ and Δ_o such that

$$\hat{C}_{\mathrm{t}} = \mathbf{skip},\ \mathcal{K}(\mathrm{t}) = \circ,\ \hat{C}'_{\mathrm{t}} = \mathbf{end},\ \iota = (e,\Delta_c,\Delta_o),\ e = (\mathrm{t},\mathbf{term}),$$
$$W' = (\mathbf{let}\ \Pi\ \mathbf{in}\ \hat{C}_1 \parallel \ldots \hat{C}'_{\mathrm{t}} \ldots \parallel \hat{C}_n),\ \mathcal{S}' = \mathcal{S},\ \mathsf{btids}(W,\mathcal{S}) = (\Delta_c,\Delta_o)\ .$$

By $\mathcal{D},R,G \models^{\chi}_{\mathrm{t}} (\Pi,\hat{C}_{\mathrm{t}},(\sigma_c,\sigma \uplus \sigma_{\mathrm{t}},\kappa_{\mathrm{t}})) \lesssim (\Gamma,\hat{\mathbb{C}}_{\mathrm{t}},(\Sigma \uplus \Sigma_{\mathrm{t}},\mathbb{k}_{\mathrm{t}})) \diamond (\mathbb{M}_{\mathrm{t}},M_{\mathrm{t}},wb_{\mathrm{t}},aw_{\mathrm{t}}) \Downarrow_{\xi_{\mathrm{t}},\xi^a_{\mathrm{t}}} P$, we know $\hat{\mathbb{C}}_{\mathrm{t}} = \mathbf{skip}$ and $\mathbb{k}_{\mathrm{t}} = \circ$. Let $\hat{\mathbb{C}}'_{\mathrm{t}} = \mathbf{end}$, $\mathsf{btids}(\mathbb{W},\mathbb{S}) = (\Delta'_c,\Delta'_o)$, $\iota' = (e,\Delta'_c,\Delta'_o)$ and $\mathbb{W}' = (\mathbf{let}\ \Gamma\ \mathbf{in}\ \hat{\mathbb{C}}_1 \parallel \ldots \hat{\mathbb{C}}'_{\mathrm{t}} \ldots \parallel \hat{\mathbb{C}}_n)$. Then $(\mathbb{W},\mathbb{S}) \overset{\iota'}{\longmapsto} (\mathbb{W}',\mathbb{S})$. From the premise, we know $\forall \mathrm{t}'.\ (\kappa_{\mathrm{t}'} = \circ \Longrightarrow \hat{C}_{\mathrm{t}'} = \hat{\mathbb{C}}_{\mathrm{t}'}) \wedge (\kappa_{\mathrm{t}'} = \circ \Longleftrightarrow \mathbb{k}_{\mathrm{t}'} = \circ)$, thus

$$\Delta'_c = \Delta_c.$$

Also we know

$$((\sigma,\Sigma),(\sigma,\Sigma),0) \models G_{\mathrm{t}}.$$

For any $\mathrm{t}' \neq \mathrm{t}$, since $G_{\mathrm{t}} \Rightarrow R_{\mathrm{t}'}$, we have $((\sigma,\Sigma),(\sigma,\Sigma),0) \models R_{\mathrm{t}'}$. Thus

$$((\sigma \uplus \sigma_{\mathrm{t}'},\Sigma \uplus \Sigma_{\mathrm{t}'}),(\sigma \uplus \sigma_{\mathrm{t}'},\Sigma \uplus \Sigma_{\mathrm{t}'}),0) \models R_{\mathrm{t}'} * \mathsf{Id}.$$

By $\mathcal{D},R,G \models^{\chi}_{\mathrm{t}'} (\Pi,\hat{C}_{\mathrm{t}'},(\sigma_c,\sigma\uplus\sigma_{\mathrm{t}'},\kappa_{\mathrm{t}'})) \lesssim (\Gamma,\hat{\mathbb{C}}_{\mathrm{t}'},(\Sigma\uplus\Sigma_{\mathrm{t}'},\mathbb{k}_{\mathrm{t}'}))\diamond(\mathbb{M}_{\mathrm{t}'},M_{\mathrm{t}'},wb_{\mathrm{t}'},aw_{\mathrm{t}'}) \Downarrow_{\xi_{\mathrm{t}'},\xi^a_{\mathrm{t}'}} P$, and by the co-induction hypothesis, we can finish the case.

- there exist $\mathrm{t}, \hat{C}'_{\mathrm{t}}, W', \sigma'_c, \sigma'', \kappa'_{\mathrm{t}}, \mathcal{K}', \mathcal{S}', e, \Delta_c$ and Δ_o such that

$$(\hat{C}_{\mathrm{t}},(\sigma_c,\sigma \uplus (\biguplus_{\mathrm{t}} \sigma_{\mathrm{t}}),\kappa_{\mathrm{t}})) \xrightarrow{e}_{\mathrm{t},\Pi} (\hat{C}'_{\mathrm{t}},(\sigma'_c,\sigma'',\kappa'_{\mathrm{t}}))),\ \mathcal{K}' = \mathcal{K}\{\mathrm{t} \rightsquigarrow \kappa'_{\mathrm{t}}\},\ \mathcal{S}' = (\sigma'_c,\sigma'',\mathcal{K}'),$$
$$W' = (\mathbf{let}\ \Pi\ \mathbf{in}\ \hat{C}_1 \parallel \ldots \hat{C}'_{\mathrm{t}} \ldots \parallel \hat{C}_n),\ \iota = (e,\Delta_c,\Delta_o),\ \mathrm{t} = \mathsf{tid}(e),\ \mathsf{btids}(W',\mathcal{S}') = (\Delta_c,\Delta_o).$$

By $\mathcal{D},R,G \models^{\chi}_{\mathrm{t}} (\Pi,\hat{C}_{\mathrm{t}},(\sigma_c,\sigma \uplus \sigma_{\mathrm{t}},\kappa_{\mathrm{t}})) \lesssim (\Gamma,\hat{\mathbb{C}}_{\mathrm{t}},(\Sigma \uplus \Sigma_{\mathrm{t}},\mathbb{k}_{\mathrm{t}})) \diamond (\mathbb{M}_{\mathrm{t}},M_{\mathrm{t}},wb_{\mathrm{t}},aw_{\mathrm{t}}) \Downarrow_{\xi_{\mathrm{t}},\xi^a_{\mathrm{t}}} P$, we know

(A) For any $\mathrm{t}' \in \xi_{\mathrm{t}} \cup \xi^a_{\mathrm{t}}$, we have $\mathrm{t}' \neq \mathrm{t}$ and $(\sigma \uplus \sigma_{\mathrm{t}},\Sigma \uplus \Sigma_{\mathrm{t}}) \models \mathsf{Enabled}(\mathcal{D}_{\mathrm{t}'}) * \mathsf{true}$.
And there exist $n, \sigma''', \mathcal{E}, \hat{\mathbb{C}}'_{\mathrm{t}}, \Sigma''', \mathbb{k}'_{\mathrm{t}}, \mathbb{M}'_{\mathrm{t}}, M'_{\mathrm{t}}, wb'_{\mathrm{t}}, aw'_{\mathrm{t}}, \xi'_{\mathrm{t}}$ and ξ^{a1}_{t} such that
(B) $\sigma'' = \sigma''' \uplus (\biguplus_{\mathrm{t}' \neq \mathrm{t}} \sigma_{\mathrm{t}'})$, and
(C) $(\hat{\mathbb{C}}_{\mathrm{t}},(\sigma_c,\Sigma \uplus (\biguplus_{\mathrm{t}} \Sigma_{\mathrm{t}}),\mathbb{k}_{\mathrm{t}})) \xrightarrow{\mathcal{E}}{}^{n}_{\mathrm{t},\Gamma} (\hat{\mathbb{C}}'_{\mathrm{t}},(\sigma'_c,\Sigma''' \uplus (\biguplus_{\mathrm{t}' \neq \mathrm{t}} \Sigma_{\mathrm{t}'}),\mathbb{k}'_{\mathrm{t}}))$, and $n = 0 \vee n = 1$, and
(D) $\mathsf{is\_clt}(e) \vee \mathsf{is\_inv}(e) \vee \mathsf{is\_ret}(e) \Longrightarrow \mathcal{E} = e :: \epsilon$, and
$\mathsf{is\_obj}(e) \Longrightarrow (\forall i, e' = \mathcal{E}(i).\ \mathsf{is\_obj}(e'))$, and
(E) $\mathcal{D},R,G \models^{\chi}_{\mathrm{t}} (\Pi,\hat{C}'_{\mathrm{t}},(\sigma'_c,\sigma''',\kappa'_{\mathrm{t}})) \lesssim (\Gamma,\hat{\mathbb{C}}'_{\mathrm{t}},(\Sigma''',\mathbb{k}'_{\mathrm{t}})) \diamond (\mathbb{M}'_{\mathrm{t}},M'_{\mathrm{t}},wb'_{\mathrm{t}},aw'_{\mathrm{t}}) \Downarrow_{\xi'_{\mathrm{t}},\xi^{a1}_{\mathrm{t}}} P$, and
(F) $((\sigma \uplus \sigma_{\mathrm{t}},\Sigma \uplus \Sigma_{\mathrm{t}}),(\sigma''',\Sigma'''),0) \models G_{\mathrm{t}} * \mathsf{True}$, and
(G) if $\mathsf{inObj}(\hat{C}_{\mathrm{t}},\kappa_{\mathrm{t}})$, then $n > 0$, or $\mathbb{M}'_{\mathrm{t}} < \mathbb{M}_{\mathrm{t}}$, or $\mathbb{M}'_{\mathrm{t}} = \mathbb{M}_{\mathrm{t}}$ and $wb'_{\mathrm{t}} = wb_{\mathrm{t}} = \mathbf{true}$ and $\xi_{\mathrm{t}} \subseteq \xi'_{\mathrm{t}}$; and
(H) if $((\sigma \uplus \sigma_{\mathrm{t}},\Sigma \uplus \Sigma_{\mathrm{t}}),(\sigma''',\Sigma''')) \models \langle[\mathcal{D}_{\mathrm{t}}]\rangle * \mathsf{True}$, then $M'_{\mathrm{t}} < M_{\mathrm{t}}$.
Since $(\sigma,\Sigma) \models I$ and $I \triangleright \{R,G\}$, we know there exist σ', σ'_{t}, Σ' and Σ'_{t} such that

$$\sigma''' = \sigma' \uplus \sigma'_{\mathrm{t}},\quad \Sigma''' = \Sigma' \uplus \Sigma'_{\mathrm{t}},\quad (\sigma',\Sigma') \models I \quad \text{and} \quad ((\sigma,\Sigma),(\sigma',\Sigma'),0) \models G_{\mathrm{t}}.$$

For any $\mathrm{t}' \neq \mathrm{t}$, since $G_{\mathrm{t}} \Rightarrow R_{\mathrm{t}'}$, we have $((\sigma,\Sigma),(\sigma',\Sigma'),0) \models R_{\mathrm{t}'}$. Thus

$$((\sigma \uplus \sigma_{\mathrm{t}'},\Sigma \uplus \Sigma_{\mathrm{t}'}),(\sigma' \uplus \sigma_{\mathrm{t}'},\Sigma' \uplus \Sigma_{\mathrm{t}'}),0) \models R_{\mathrm{t}'} * \mathsf{Id}.$$

By $\mathcal{D},R,G \models^{\chi}_{\mathrm{t}'} (\Pi,\hat{C}_{\mathrm{t}'},(\sigma_c,\sigma\uplus\sigma_{\mathrm{t}'},\kappa_{\mathrm{t}'})) \precsim (\Gamma,\hat{\mathbb{C}}_{\mathrm{t}'},(\Sigma\uplus\Sigma_{\mathrm{t}'},\mathbb{k}_{\mathrm{t}'}))\diamond(\mathbb{M}_{\mathrm{t}'},M_{\mathrm{t}'},wb_{\mathrm{t}'},aw_{\mathrm{t}'}) \Downarrow_{\xi_{\mathrm{t}'},\xi^{a}_{\mathrm{t}'}} P$, we know

(I) For any $\mathrm{t}'' \in \xi_{\mathrm{t}'} \cup \xi^{a}_{\mathrm{t}'}$, we have $\mathrm{t}'' \neq \mathrm{t}'$ and $(\sigma \uplus \sigma_{\mathrm{t}'},\Sigma \uplus \Sigma_{\mathrm{t}'}) \models \mathsf{Enabled}(\mathcal{D}_{\mathrm{t}''}) * \mathsf{true}$.

And there exist $\mathbb{M}'_{\mathrm{t}'}$, $M'_{\mathrm{t}'}$, $wb'_{\mathrm{t}'}$, $aw'_{\mathrm{t}'}$, $\xi^{d}_{\mathrm{t}'}$, $\xi^{ad}_{\mathrm{t}'}$, $\xi'_{\mathrm{t}'}$ and $\xi^{a1}_{\mathrm{t}'}$ such that

(J) $\mathcal{D},R,G \models^{\chi}_{\mathrm{t}'} (\Pi,\hat{C}_{\mathrm{t}'},(\sigma'_c,\sigma'\uplus\sigma_{\mathrm{t}'},\kappa_{\mathrm{t}'})) \precsim (\Gamma,\hat{\mathbb{C}}_{\mathrm{t}'},(\Sigma'\uplus\Sigma_{\mathrm{t}'},\mathbb{k}_{\mathrm{t}'}))\diamond(\mathbb{M}'_{\mathrm{t}'},M'_{\mathrm{t}'},wb'_{\mathrm{t}'},aw'_{\mathrm{t}'}) \Downarrow_{\xi'_{\mathrm{t}'},\xi^{a1}_{\mathrm{t}'}} P$, and

(K) $\xi^{d}_{\mathrm{t}'} = \{\mathrm{t}'' \mid (\mathrm{t}'' \in \xi_{\mathrm{t}'}) \wedge (((\sigma \uplus \sigma_{\mathrm{t}'},\Sigma \uplus \Sigma_{\mathrm{t}'}),(\sigma' \uplus \sigma_{\mathrm{t}'},\Sigma' \uplus \Sigma_{\mathrm{t}'})) \models \langle \mathcal{D}_{\mathrm{t}''} \rangle * \mathsf{Id})\}$ and
$\mathsf{inObj}(\hat{C}_{\mathrm{t}'},\kappa_{\mathrm{t}'}) \Longrightarrow \mathbb{M}'_{\mathrm{t}'} < \mathbb{M}_{\mathrm{t}'} \vee (\mathbb{M}'_{\mathrm{t}'} = \mathbb{M}_{\mathrm{t}'} \wedge wb'_{\mathrm{t}'} = wb_{\mathrm{t}'})$ and
$wb_{\mathrm{t}'} = \mathbf{true} \wedge (\xi^{d}_{\mathrm{t}'} \neq \emptyset \vee ((\Sigma\uplus\Sigma_{\mathrm{t}'},\mathbb{k}_{\mathrm{t}'}) \models \neg\mathsf{en}(\hat{\mathbb{C}}_{\mathrm{t}'}) \wedge (\Sigma'\uplus\Sigma_{\mathrm{t}'},\mathbb{k}_{\mathrm{t}'}) \models \mathsf{en}(\hat{\mathbb{C}}_{\mathrm{t}'}))) \Longrightarrow \mathbb{M}'_{\mathrm{t}'} < \mathbb{M}_{\mathrm{t}'}$ and
$\mathbb{M}'_{\mathrm{t}'} = \mathbb{M}_{\mathrm{t}'} \wedge wb'_{\mathrm{t}'} = wb_{\mathrm{t}'} = \mathbf{true} \Longrightarrow \xi_{\mathrm{t}'} \backslash \xi^{d}_{\mathrm{t}'} \subseteq \xi'_{\mathrm{t}'}$, and

(L) if $(\sigma \uplus \sigma_{\mathrm{t}'},\Sigma \uplus \Sigma_{\mathrm{t}'}) \models \mathsf{Enabled}(\mathcal{D}_{\mathrm{t}'}) * \mathsf{true}$, then $M'_{\mathrm{t}'} \leq M_{\mathrm{t}'}$; and

(M) $\xi^{ad}_{\mathrm{t}'} = \{\mathrm{t}'' \mid (\mathrm{t}'' \in \xi^{a}_{\mathrm{t}'}) \wedge (((\sigma \uplus \sigma_{\mathrm{t}'},\Sigma \uplus \Sigma_{\mathrm{t}'}),(\sigma' \uplus \sigma'_{\mathrm{t}'},\Sigma' \uplus \Sigma_{\mathrm{t}'})) \models \langle \mathcal{D}_{\mathrm{t}''} \rangle * \mathsf{Id})\}$ and
$\mathsf{inObj}(\hat{C}_{\mathrm{t}'},\kappa_{\mathrm{t}'}) \wedge \mathsf{is\_await}(\hat{C}_{\mathrm{t}'}) \Longrightarrow \xi^{a}_{\mathrm{t}'} \backslash \xi^{ad}_{\mathrm{t}'} \subseteq \xi^{a1}_{\mathrm{t}'}$ and
$\mathsf{inObj}(\hat{C}_{\mathrm{t}'},\kappa_{\mathrm{t}'}) \wedge \mathsf{is\_await}(\hat{C}_{\mathrm{t}'}) \wedge (\xi^{ad}_{\mathrm{t}'} \neq \emptyset \vee ((\Sigma \uplus \Sigma_{\mathrm{t}'},\mathbb{k}_{\mathrm{t}'}) \models \neg\mathsf{en}(\hat{\mathbb{C}}_{\mathrm{t}'}) \wedge (\Sigma' \uplus \Sigma_{\mathrm{t}'},\mathbb{k}_{\mathrm{t}'}) \models \mathsf{en}(\hat{\mathbb{C}}_{\mathrm{t}'}))) \Longrightarrow aw'_{\mathrm{t}'} < aw_{\mathrm{t}'}$ and
$(\chi = \mathsf{sfair}) \wedge \mathsf{inObj}(\hat{C}_{\mathrm{t}'},\kappa_{\mathrm{t}'}) \wedge \mathsf{is\_await}(\hat{C}_{\mathrm{t}'}) \wedge ((\sigma \uplus \sigma_{\mathrm{t}'},\kappa_{\mathrm{t}'}) \models \neg\mathsf{en}(\hat{C}_{\mathrm{t}'})) \wedge ((\sigma' \uplus \sigma_{\mathrm{t}'},\kappa_{\mathrm{t}'}) \models \neg\mathsf{en}(\hat{C}_{\mathrm{t}'})) \Longrightarrow aw'_{\mathrm{t}'} \leq aw_{\mathrm{t}'}$ and
$(\chi = \mathsf{wfair}) \wedge \mathsf{inObj}(\hat{C}_{\mathrm{t}'},\kappa_{\mathrm{t}'}) \wedge \mathsf{is\_await}(\hat{C}_{\mathrm{t}'}) \Longrightarrow aw'_{\mathrm{t}'} \leq aw_{\mathrm{t}'}$.

Let $\mathbb{W}' = (\mathbf{let}\ \Gamma\ \mathbf{in}\ \hat{\mathbb{C}}_1 \parallel \ldots \hat{\mathbb{C}}'_{\mathrm{t}} \ldots \parallel \hat{\mathbb{C}}_n)$ and $\mathbb{S}' = (\sigma'_c,\Sigma' \uplus \Sigma'_{\mathrm{t}} \uplus (\biguplus_{\mathrm{t}'\neq\mathrm{t}} \Sigma_{\mathrm{t}'}),\mathbb{K}\{\mathrm{t} \rightsquigarrow \mathbb{k}'_{\mathrm{t}}\})$. From (C), we know there exists T such that

$$(\mathbb{W},\mathbb{S})) \xmapsto{T}{}^{n} (\mathbb{W}',\mathbb{S}')\ .$$

From (D), we know

$$\mathsf{is\_clt}(e) \vee \mathsf{is\_inv}(e) \vee \mathsf{is\_ret}(e) \Longrightarrow T = (e,\Delta_c,\_)::\epsilon,$$
$$\mathsf{is\_obj}(e) \Longrightarrow (\forall i,\iota' = T(i).\ \mathsf{is\_obj}(\iota') \wedge \mathrm{t} = \mathsf{tid}(\iota'))\ .$$

Define the functions $\mathscr{M}'$, ζ', β', α' and ζ'_a as follows.

- $dom(\mathscr{M}') = dom(\zeta') = dom(\beta') = dom(\alpha') = dom(\zeta'_a) = \mathsf{inObjThrds}(W',\mathcal{S}')$.
- For any $\mathrm{t} \in dom(\mathscr{M}')$, we have
  - $\mathscr{M}'(\mathrm{t}) = (\mathbb{M}'_{\mathrm{t}},\{\mathrm{t}' \rightsquigarrow M'_{\mathrm{t}'} \mid \mathrm{t}' \in \xi'_{\mathrm{t}}\})$;
  - $\zeta'(\mathrm{t}) = \xi'_{\mathrm{t}}$;
  - $\beta'(\mathrm{t}) = wb'_{\mathrm{t}}$;
  - $\alpha'(\mathrm{t}) = (aw'_{\mathrm{t}},\{\mathrm{t}' \rightsquigarrow M'_{\mathrm{t}'} \mid \mathrm{t}' \in \xi^{a1}_{\mathrm{t}}\})$;
  - $\zeta'_a(\mathrm{t}) = \xi^{a1}_{\mathrm{t}}$.

Then by the co-induction hypothesis, we know

$$\models_{\chi} (W',\mathcal{S}') \precsim (\mathbb{W}',\mathbb{S}') \diamond (\mathscr{M}',\zeta',\beta',\alpha',\zeta'_a).$$

For the thread t, from (G), we know: if $\mathrm{t} \in \mathsf{inObjThrds}(W,\mathcal{S})$, then one of the following holds:

- $n > 0$. Thus $\mathrm{t} \in \mathsf{tidset}(T)$.
- $\mathbb{M}'_{\mathrm{t}} < \mathbb{M}_{\mathrm{t}}$. Thus $\mathscr{M}'(\mathrm{t}) < \mathscr{M}(\mathrm{t})$.
- $\mathbb{M}'_{\mathrm{t}} = \mathbb{M}_{\mathrm{t}}$ and $wb'_{\mathrm{t}} = wb_{\mathrm{t}} = \mathbf{true}$ and $\xi_{\mathrm{t}} \subseteq \xi'_{\mathrm{t}}$.
  Thus $\beta'(\mathrm{t}) = \beta(\mathrm{t}) = \mathbf{true}$ and $\zeta(\mathrm{t}) \subseteq \zeta'(\mathrm{t})$. Also we know $\xi'_{\mathrm{t}} \subseteq (\mathsf{inObjThrds}(W,\mathcal{S})\backslash\{\mathrm{t}\})$.
  Then we only need to show the following (B.5):

$$\{\mathrm{t}' \rightsquigarrow M'_{\mathrm{t}'} \mid \mathrm{t}' \in \xi'_{\mathrm{t}}\} \ \leq\ \{\mathrm{t}' \rightsquigarrow M_{\mathrm{t}'} \mid \mathrm{t}' \in \xi_{\mathrm{t}}\} \tag{B.5}$$

From (A), since $\forall t'.\ \mathsf{Enabled}(\mathcal{D}_{t'}) \Rightarrow I$ and $\mathsf{Precise}(I)$, we know: for any $t' \in \xi_t$,

$$t' \neq t \quad \text{and} \quad (\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{t'}) .$$

Thus $(\sigma \uplus \sigma_{t'}, \Sigma \uplus \Sigma_{t'}) \models \mathsf{Enabled}(\mathcal{D}_{t'}) * \mathsf{true}$. From (M), we have $M'_{t'} \leq M_{t'}$. Thus (B.5) holds.

For any $t' \in \mathsf{inObjThrds}(W, \mathcal{S}) \backslash \{t\}$, from (K), we know one of the following holds:

- $\mathbb{M}'_{t'} < \mathbb{M}_{t'}$. Thus $\mathscr{M}'(t') < \mathscr{M}(t')$.
- $\mathbb{M}'_{t'} = \mathbb{M}_{t'}$ and $wb'_{t'} = wb_{t'} = \textbf{false}$.
  Thus $\beta'(t') = \beta(t') = \textbf{false}$. Then from the premise, we know
  $$\xi_{t'} = \xi'_{t'} = \emptyset.$$
  Thus the following (B.6) holds:
  $$\{t'' \rightsquigarrow M'_{t''} \mid t'' \in \xi'_{t'}\} \ = \ \{t'' \rightsquigarrow M_{t''} \mid t'' \in \xi_{t'}\} \tag{B.6}$$
  Thus $\mathscr{M}'(t') = \mathscr{M}(t')$ holds.
- $\mathbb{M}'_{t'} = \mathbb{M}_{t'}$ and $wb'_{t'} = wb_{t'} = \textbf{true}$.
  Thus $\beta'(t') = \beta(t') = \textbf{true}$. From (K), we know
  $$\xi^d_{t'} = \emptyset \wedge ((\Sigma \uplus \Sigma_{t'}, \mathbb{k}_{t'}) \models \mathsf{en}(\hat{\mathbb{C}}_{t'}) \vee (\Sigma' \uplus \Sigma_{t'}, \mathbb{k}_{t'}) \models \neg\mathsf{en}(\hat{\mathbb{C}}_{t'})).$$
  Also, since $\xi_{t'} \backslash \xi^d_{t'} \subseteq \xi'_{t'}$, we know
  $$\xi_{t'} \subseteq \xi'_{t'}.$$
  One of the following holds:
  - $t \notin \xi_{t'}$. Thus $t \notin \zeta(t')$ and $\zeta(t') \subseteq \zeta'(t') \subseteq (\mathsf{inObjThrds}(W, \mathcal{S}) \backslash \{t'\})$. We only need to show the following (B.7):
    $$\{t'' \rightsquigarrow M'_{t''} \mid t'' \in \xi'_{t'}\} \ \leq \ \{t'' \rightsquigarrow M_{t''} \mid t'' \in \xi_{t'}\} \tag{B.7}$$
    From (I), we know: for any $t'' \in \xi_{t'}$,
    $$t'' \neq t' \quad \text{and} \quad (\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{t''}) .$$
    Thus $(\sigma \uplus \sigma_{t''}, \Sigma \uplus \Sigma_{t''}) \models \mathsf{Enabled}(\mathcal{D}_{t''}) * \mathsf{true}$. From (L), we have $M'_{t''} \leq M_{t''}$. Thus (B.7) holds.
  - $t \in \xi_{t'}$. Since $\mathsf{wffAct}(R, \mathcal{D})$, we know
    $$((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle [\mathcal{D}_t] \rangle .$$
    From (H), we know
    $$M'_t < M_t.$$
    We only need to show the following (B.8):
    $$\{t'' \rightsquigarrow M'_{t''} \mid t'' \in \xi'_{t'}\} \ < \ \{t'' \rightsquigarrow M_{t''} \mid t'' \in \xi_{t'}\} \tag{B.8}$$
    For any $t'' \in \xi_{t'} \backslash \{t\}$, from (I), we know:
    $$t'' \neq t' \quad \text{and} \quad (\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{t''}) .$$
    Thus $(\sigma \uplus \sigma_{t''}, \Sigma \uplus \Sigma_{t''}) \models \mathsf{Enabled}(\mathcal{D}_{t''}) * \mathsf{true}$. From (L), we have $M'_{t''} \leq M_{t''}$. Thus (B.8) holds. Thus $\mathscr{M}'(t') < \mathscr{M}(t')$ holds.

For any $t' \in \mathsf{inObjThrds}(W, \mathcal{S}) \backslash \{t\}$ and $\mathsf{is\_await}(W|_{t'})$, we have:
if $((\chi = \mathsf{sfair}) \wedge (t' \in \mathsf{bset}(W, \mathcal{S})) \wedge (t' \in \mathsf{bset}(W', \mathcal{S}'))) \vee (\chi = \mathsf{wfair})$, then from (M), we know $aw'_{t'} \leq aw_{t'}$. Thus one of the following holds:

- $aw'_{t'} < aw_{t'}$. Thus $\alpha'(t') < \alpha(t')$.
- $aw'_{t'} = aw_{t'}$. From (M), we know
  $$\xi^{ad}_{t'} = \emptyset \wedge ((\Sigma \uplus \Sigma_{t'}, \mathbb{k}_{t'}) \models \mathsf{en}(\hat{\mathbb{C}}_{t'}) \vee (\Sigma' \uplus \Sigma_{t'}, \mathbb{k}_{t'}) \models \neg\mathsf{en}(\hat{\mathbb{C}}_{t'})).$$
  Also since $\xi^a_{t'} \backslash \xi^{ad}_{t'} \subseteq \xi^{a1}_{t'}$, we know
  $$\xi^a_{t'} \subseteq \xi^{a1}_{t'}.$$
  One of the following holds:
  - $t \notin \xi^a_{t'}$. Thus $t \notin \zeta_a(t')$ and $\zeta_a(t') \subseteq \zeta'_a(t') \subseteq (\mathsf{inObjThrds}(W, \mathcal{S}) \backslash \{t'\})$. Also we know

$$(\zeta_a(\mathrm{t}') \neq \emptyset) \vee (\zeta_a(\mathrm{t}') = \emptyset \wedge \mathrm{t}' \notin \mathrm{bset}(\mathbb{W}, \mathbb{S})) \vee (\zeta_a(\mathrm{t}') = \emptyset \wedge \mathrm{t}' \in \mathrm{bset}(\mathbb{W}', \mathbb{S}')).$$

We only need to show the following (B.9):

$$\{\mathrm{t}'' \rightsquigarrow M'_{\mathrm{t}''} \mid \mathrm{t}'' \in \xi^{a1}_{\mathrm{t}'}\} \;\leq\; \{\mathrm{t}'' \rightsquigarrow M_{\mathrm{t}''} \mid \mathrm{t}'' \in \xi^{a}_{\mathrm{t}'}\} \tag{B.9}$$

From (I), we know: for any $\mathrm{t}'' \in \xi^a_{\mathrm{t}'}$,

$$\mathrm{t}'' \neq \mathrm{t}' \quad \text{and} \quad (\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathrm{t}''}) .$$

Thus $(\sigma \uplus \sigma_{\mathrm{t}''}, \Sigma \uplus \Sigma_{\mathrm{t}''}) \models \mathsf{Enabled}(\mathcal{D}_{\mathrm{t}''}) * \mathsf{true}$. From (L), we have $M'_{\mathrm{t}''} \leq M_{\mathrm{t}''}$. Thus (B.9) holds.

- $\mathrm{t} \in \xi^a_{\mathrm{t}'}$. Since $\mathsf{wffAct}(R, \mathcal{D})$, we know

$$((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle [\mathcal{D}_{\mathrm{t}}] \rangle .$$

From (H), we know

$$M'_{\mathrm{t}} < M_{\mathrm{t}}.$$

We only need to show the following (B.10):

$$\{\mathrm{t}'' \rightsquigarrow M'_{\mathrm{t}''} \mid \mathrm{t}'' \in \xi^{a1}_{\mathrm{t}'}\} \;<\; \{\mathrm{t}'' \rightsquigarrow M_{\mathrm{t}''} \mid \mathrm{t}'' \in \xi^{a}_{\mathrm{t}'}\} \tag{B.10}$$

For any $\mathrm{t}'' \in \xi^a_{\mathrm{t}'} \backslash \{\mathrm{t}\}$, from (I), we know:

$$\mathrm{t}'' \neq \mathrm{t}' \quad \text{and} \quad (\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathrm{t}''}) .$$

Thus $(\sigma \uplus \sigma_{\mathrm{t}''}, \Sigma \uplus \Sigma_{\mathrm{t}''}) \models \mathsf{Enabled}(\mathcal{D}_{\mathrm{t}''}) * \mathsf{true}$. From (L), we have $M'_{\mathrm{t}''} \leq M_{\mathrm{t}''}$. Thus (B.10) holds. Thus $\alpha'(\mathrm{t}') < \alpha(\mathrm{t}')$ holds.

Thus we are done. □

Lemma B.22 (Parallel Compositionality for Simulations with PDF Objects). *If there exist R, G and $\mathcal{D}$ such that the following hold:*

(1) for any $\mathrm{t} \in [1..n]$, we have $\mathcal{D}, R, G \models^{\chi}_{\mathrm{t}} \{P\}(\Pi, C_{\mathrm{t}}) \precsim (\Pi', \Gamma, \mathbb{C}_{\mathrm{t}})$;

(2) $\forall \mathrm{t}, \mathrm{t}'.\ \mathrm{t} \neq \mathrm{t}' \Longrightarrow G_{\mathrm{t}} \Rightarrow R_{\mathrm{t}'}$, $\mathit{wffAct}(R, \mathcal{D})$, $P \Rightarrow \neg\mathsf{Enabled}(\mathcal{D})$, $P \vee \mathsf{Enabled}(\mathcal{D}) \Rightarrow I$, $I \triangleright \{R, G\}$,

then $\models_{\chi} \{\bigwedge_{\mathrm{t} \in [1..n]} P_{\mathrm{t}}\}(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n) \precsim (\mathbf{let}\ \Pi'\ \mathbf{in}\ \mathbb{C}_1 \parallel \ldots \parallel \mathbb{C}_n, \Gamma)$.

Proof. For any σ_c, σ and Σ, if $(\sigma, \Sigma) \models (\bigwedge_{\mathrm{t} \in [1..n]} P_{\mathrm{t}})$, from the premises, we know: there exist $\mathbb{M}_1, \ldots, \mathbb{M}_n$, $M_1, \ldots, M_n$, $aw_1, \ldots, aw_n$, $wb_1, \ldots, wb_n$, $\mathbb{B}_1, \ldots, \mathbb{B}_n$, $\xi_1, \ldots, \xi_n$, $\xi^a_1, \ldots, \xi^a_n$ such that $wb_1 = \ldots = wb_n = \mathbf{false}$ and for any $\mathrm{t} \in [1..n]$:

$$\mathcal{D}, R, G \models^{\chi}_{\mathrm{t}} (\Pi, C_{\mathrm{t}}, (\sigma_c, \sigma, \circ)) \precsim (\Pi', \Gamma, \mathbb{B}_{\mathrm{t}}, \mathbb{C}_{\mathrm{t}}, (\Sigma, \circ)) \diamond (\mathbb{M}_{\mathrm{t}}, M_{\mathrm{t}}, wb_{\mathrm{t}}, aw_{\mathrm{t}}) \Downarrow_{\xi_{\mathrm{t}}, \xi^a_{\mathrm{t}}} P.$$

We want to show that there exist $\mathscr{B}$, $\mathscr{M}$, α, β, ζ and ζ_a such that

$$\models_{\chi} ((\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n), (\sigma_c, \sigma, \circledcirc)) \precsim ((\mathbf{let}\ \Gamma\ \mathbf{in}\ \mathbb{C}_1 \parallel \ldots \parallel \mathbb{C}_n), \Gamma, \mathscr{B}, (\sigma_c, \Sigma, \circledcirc)) \diamond (\mathscr{M}, \zeta, \beta, \alpha, \zeta_a).$$

We generalize the result and prove the following (B.11):

If $(\sigma, \Sigma) \models I$ and the following holds for any $\mathrm{t} \in [1..n]$:

$$\hat{C}_{\mathrm{t}} \neq \mathbf{end} \Longrightarrow \mathcal{D}, R, G \models^{\chi}_{\mathrm{t}} (\Pi, \hat{C}_{\mathrm{t}}, (\sigma_c, \sigma \uplus \sigma_{\mathrm{t}}, \kappa_{\mathrm{t}})) \precsim (\Pi', \Gamma, \mathbb{B}_{\mathrm{t}}, \hat{\mathbb{C}}_{\mathrm{t}}, (\Sigma \uplus \Sigma_{\mathrm{t}}, \mathbb{k}_{\mathrm{t}})) \diamond (\mathbb{M}_{\mathrm{t}}, M_{\mathrm{t}}, wb_{\mathrm{t}}, aw_{\mathrm{t}}) \Downarrow_{\xi_{\mathrm{t}}, \xi^a_{\mathrm{t}}} P\ ,$$

$$\hat{C}_{\mathrm{t}} = \mathbf{end} \Longrightarrow \hat{\mathbb{C}}_{\mathrm{t}} = \mathbf{end} \wedge \kappa_{\mathrm{t}} = \circ \wedge \mathbb{k}_{\mathrm{t}} = \circ,$$

then

$$\models_{\chi} ((\mathbf{let}\ \Pi\ \mathbf{in}\ \hat{C}_1 \parallel \ldots \parallel \hat{C}_n), (\sigma_c, \sigma \uplus (\biguplus_{\mathrm{t}} \sigma_{\mathrm{t}}), \mathcal{K})) \precsim ((\mathbf{let}\ \Gamma\ \mathbf{in}\ \hat{\mathbb{C}}_1 \parallel \ldots \parallel \hat{\mathbb{C}}_n), \Gamma, \mathscr{B}, (\sigma_c, \Sigma \uplus (\biguplus_{\mathrm{t}} \Sigma_{\mathrm{t}}), \mathbb{K})) \diamond (\mathscr{M}, \zeta, \beta, \alpha, \zeta_a).$$

Here for any $\mathrm{t} \in [1..n]$, $\mathcal{K}(\mathrm{t}) = \kappa_{\mathrm{t}}$ and $\mathbb{K}(\mathrm{t}) = \mathbb{k}_{\mathrm{t}}$, and the functions $\mathscr{B}$, $\mathscr{M}$, ζ, β, α and ζ_a are defined as follows.

- $dom(\mathscr{B}) = dom(\mathscr{M}) = dom(\zeta) = dom(\beta) = dom(\alpha) = dom(\zeta_a) = \mathsf{inObjThrds}((\mathbf{let}\ \Pi\ \mathbf{in}\ \hat{C}_1 \parallel \ldots \parallel \hat{C}_n), (\sigma_c, \sigma \uplus (\biguplus_{\mathrm{t}} \sigma_{\mathrm{t}}), \mathcal{K}))$.

- For any $\mathsf{t} \in dom(\mathscr{M})$, we have
  - $\mathscr{B}(\mathsf{t}) = \mathbb{B}_\mathsf{t}$;
  - $\mathscr{M}(\mathsf{t}) = (\mathbb{M}_\mathsf{t}, \{\mathsf{t}' \rightsquigarrow M_{\mathsf{t}'} \mid \mathsf{t}' \in \xi_\mathsf{t}\})$;
  - $\zeta(\mathsf{t}) = \xi_\mathsf{t}$;
  - $\beta(\mathsf{t}) = wb_\mathsf{t}$;
  - $\alpha(\mathsf{t}) = (aw_\mathsf{t}, \{\mathsf{t}' \rightsquigarrow M_{\mathsf{t}'} \mid \mathsf{t}' \in \xi^a_\mathsf{t}\})$;
  - $\zeta_a(\mathsf{t}) = \xi^a_\mathsf{t}$.

  The order $(\mathbb{M}', \mathcal{M}') < (\mathbb{M}, \mathcal{M})$ is defined as a dictionary order:

  $$(\mathbb{M}', \mathcal{M}') < (\mathbb{M}, \mathcal{M}) \text{ iff } (\mathbb{M}' < \mathbb{M}) \vee (\mathbb{M}' = \mathbb{M}) \wedge (\mathcal{M}' < \mathcal{M})$$

  $$\mathcal{M}' < \mathcal{M} \text{ iff } \exists \mathsf{t}.\ (\mathcal{M}'(\mathsf{t}) < \mathcal{M}(\mathsf{t})) \wedge (\forall \mathsf{t}' \in dom(\mathcal{M}) \backslash \{\mathsf{t}\}.\ \mathcal{M}'(\mathsf{t}') \leq \mathcal{M}(\mathsf{t}'))$$

  $$\mathcal{M}' \leq \mathcal{M} \text{ iff } \forall \mathsf{t} \in dom(\mathcal{M}).\ \mathcal{M}'(\mathsf{t}) \leq \mathcal{M}(\mathsf{t})$$

  Clearly $\mathscr{M}'(\mathsf{t}) < \mathscr{M}(\mathsf{t})$ is a well-founded order.

(B.11)

By co-induction. Let $W \stackrel{\text{def}}{=} (\textbf{let } \Pi \textbf{ in } \hat{C}_1 \parallel \ldots \parallel \hat{C}_n)$, $\mathbb{W} \stackrel{\text{def}}{=} (\textbf{let } \Gamma \textbf{ in } \hat{\mathbb{C}}_1 \parallel \ldots \parallel \hat{\mathbb{C}}_n)$, $\mathcal{S} \stackrel{\text{def}}{=} (\sigma_c, \sigma \uplus (\biguplus_\mathsf{t} \sigma_\mathsf{t}), \mathcal{K})$ and $\mathbb{S} \stackrel{\text{def}}{=} (\sigma_c, \Sigma \uplus (\biguplus_\mathsf{t} \Sigma_\mathsf{t}), \mathbb{K})$. Suppose $\sigma = (s, h)$. Then $s(\mathtt{TIDS}) = [1..n]$.

(1)(a) $dom(\mathscr{B}) = dom(\mathscr{M}) = dom(\zeta) = dom(\beta) = dom(\alpha) = dom(\zeta_a) = \mathsf{inObjThrds}(W, \mathcal{S}) = \mathsf{inObjThrds}(\mathbb{W}, \mathbb{S})$.

(b) For any $\mathsf{t} \in \mathsf{inObjThrds}(W, \mathcal{S})$, we have $\zeta(\mathsf{t}) \cup \zeta_a(\mathsf{t}) \subseteq (\mathsf{inObjThrds}(W, \mathcal{S}) \backslash \{\mathsf{t}\})$.

(c) For any $\mathsf{t} \in \mathsf{inObjThrds}(W, \mathcal{S})$, if $\beta(\mathsf{t}) = \textbf{false}$, then $\zeta(\mathsf{t}) = \emptyset$; and if $\beta(\mathsf{t}) = \textbf{true}$, then $\zeta(\mathsf{t}) \neq \emptyset \vee \mathsf{t} \in \mathsf{ffset}(\mathscr{B}, \mathbb{S})$.

(d) For any $\mathsf{t} \in \mathsf{inObjThrds}(W, \mathcal{S})$, if $\mathsf{t} \in \mathsf{bset}(W, \mathcal{S})$ and $\mathsf{t} \notin \mathsf{ffset}(\mathscr{B}, \mathbb{S})$, then $\zeta_a(\mathsf{t}) \neq \emptyset$.

(e) For any $\mathsf{t} \in \mathsf{inObjThrds}(W, \mathcal{S})$, for any $\mathsf{t}' \in \zeta(\mathsf{t}) \cup \zeta_a(\mathsf{t})$, we have $\mathsf{t}' \notin \mathsf{bset}(W, \mathcal{S})$.

(f) For any $\mathsf{t} \in \mathsf{ffset}(\mathscr{B}, \mathbb{S})$, we have $\mathsf{is\_while0}(\mathbb{W}|_\mathsf{t}, \mathscr{B}(\mathsf{t}))$.
For any $\mathsf{t} \in \mathsf{inObjThrds}(W, \mathcal{S})$, we have $\mathsf{is\_while0}(\mathbb{W}|_\mathsf{t}, \mathscr{B}(\mathsf{t})) \vee \mathsf{is\_while1}(\mathbb{W}|_\mathsf{t})$.

Proof: For any $\mathsf{t} \in [1..n]$, from the premise, we know: $\kappa_\mathsf{t} = \circ \iff \mathbb{k}_\mathsf{t} = \circ$. Thus we have $\mathsf{inObjThrds}(W, \mathcal{S}) = \mathsf{inObjThrds}(\mathbb{W}, \mathbb{S})$.
Also, from the premise, we know: for any $\mathsf{t}' \in \zeta(\mathsf{t}) \cup \zeta_a(\mathsf{t})$, we have $\mathsf{t}' \neq \mathsf{t}$ and $(\sigma \uplus \sigma_{\mathsf{t}'}, \Sigma \uplus \Sigma_{\mathsf{t}'}) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}'}) * \mathsf{true}$. Then we only need to prove $\mathsf{inObj}(C_{\mathsf{t}'}, \kappa_{\mathsf{t}'})$. Suppose $\neg\mathsf{inObj}(C_{\mathsf{t}'}, \kappa_{\mathsf{t}'})$, then we have $(\sigma \uplus \sigma_{\mathsf{t}'}, \Sigma \uplus \Sigma_{\mathsf{t}'}) \models P_{\mathsf{t}'}$. Since $P \Rightarrow \neg\mathsf{Enabled}(\mathcal{D})$, we get a contradiction. Thus $\mathsf{inObj}(C_{\mathsf{t}'}, \kappa_{\mathsf{t}'})$ holds. Also, from the premise, we know: $(\sigma \uplus \sigma_{\mathsf{t}'}, \Sigma \uplus \Sigma_{\mathsf{t}'}) \models \mathsf{en}(C_{\mathsf{t}'})$. Thus $\mathsf{t}' \notin \mathsf{bset}(W, \mathcal{S})$ holds.

(2) If $W = (\textbf{let } \Pi \textbf{ in end} \parallel \ldots \parallel \textbf{end})$, then $\exists \Pi'.\ \mathbb{W} = (\textbf{let } \Pi' \textbf{ in end} \parallel \ldots \parallel \textbf{end})$.
Proof: Since $\hat{C}_\mathsf{t} = \textbf{end} \Longrightarrow \hat{\mathbb{C}}_\mathsf{t} = \textbf{end}$, we are done.

(3) If $(W, \mathcal{S}) \overset{\iota}{\longmapsto} \textbf{abort}$, then
there exists t such that $\iota = ((\mathsf{t}, \textbf{clt}, \textbf{abort}), \emptyset, \emptyset)$ and $(\mathbb{W}, \mathbb{S}) \overset{\iota}{\longmapsto} \textbf{abort}$.
Proof: By the operational semantics we know: there exist $\mathsf{t} \in [1..n]$ and e such that $\iota = (e, \emptyset, \emptyset)$ and

$$(\hat{C}_\mathsf{t}, (\sigma_c, \sigma \uplus (\biguplus_\mathsf{t} \sigma_\mathsf{t}), \kappa_\mathsf{t})) \xrightarrow{e}_{\mathsf{t}, \Pi} \textbf{abort}.$$

By $\mathcal{D}, R, G \models^\chi_\mathsf{t} (\Pi, \hat{C}_\mathsf{t}, (\sigma_c, \sigma \uplus \sigma_\mathsf{t}, \kappa_\mathsf{t})) \precsim (\Pi', \Gamma, \mathbb{B}_\mathsf{t}, \hat{\mathbb{C}}_\mathsf{t}, (\Sigma \uplus \Sigma_\mathsf{t}, \mathbb{k}_\mathsf{t})) \diamond (\mathbb{M}_\mathsf{t}, M_\mathsf{t}, wb_\mathsf{t}, aw_\mathsf{t}) \Downarrow_{\xi_\mathsf{t}, \xi^a_\mathsf{t}} P$, we know $e = (\mathsf{t}, \textbf{clt}, \textbf{abort})$ and

$$(\hat{\mathbb{C}}_\mathsf{t}, (\sigma_c, \Sigma \uplus (\biguplus_\mathsf{t} \Sigma_\mathsf{t}), \mathbb{k}_\mathsf{t})) \xrightarrow{e}_{\mathsf{t}, \Pi'} \textbf{abort}.$$

Thus $(\mathbb{W}, \mathbb{S}) \overset{\iota}{\longmapsto} \textbf{abort}$.

(4) If $(W, \mathcal{S}) \overset{\iota}{\longmapsto} (W', \mathcal{S}')$, then
there exist t, T, $\mathbb{W}'$, $\mathbb{S}'$, $\mathscr{M}'$, ζ', β', α', ζ'_a, $\mathscr{B}'$, e, Δ_c and Δ_o such that all the following hold:

(a) $(\mathbb{W},\mathbb{S}) \overset{T}{\longmapsto}^{*} (\mathbb{W}',\mathbb{S}')$;
(b) $\iota = (e,\Delta_c,\Delta_o)$; $\mathrm{t} = \mathsf{tid}(e)$;
is_clt$(e) \vee$ is_inv$(e) \vee$ is_ret$(e) \Longrightarrow T = (e,\Delta_c,_)::\epsilon$;
is_obj$(e) \Longrightarrow (\forall i,\iota' = T(i).\ \mathsf{is_obj}(\iota'))$;
if $e = (\mathrm{t},f,\_)$, then $\mathscr{B}' = \mathscr{B}\{\mathrm{t} \rightsquigarrow \mathsf{getB}(\Gamma,f)\}$, else if is_obj$(e) \wedge$ is_while1$(\mathbb{W}'|_{\mathrm{t}})$, then $\mathscr{B}' = \mathscr{B}\{\mathrm{t} \rightsquigarrow \mathbf{true}\}$, else $\mathscr{B}' = \mathscr{B}$;
(c) $\models_{\chi} (W',\mathcal{S}') \lesssim (\mathbb{W}',\Gamma,\mathscr{B}',\mathbb{S}') \diamond (\mathscr{M}',\zeta',\beta',\alpha',\zeta'_a)$;
(d) if $\mathrm{t} \in \mathsf{inObjThrds}(W,\mathcal{S})$, then
either $\mathrm{t} \in \mathsf{tidset}(T)$,
or $\mathscr{M}'(\mathrm{t}) < \mathscr{M}(\mathrm{t})$,
or $\mathscr{M}'(\mathrm{t}) = \mathscr{M}(\mathrm{t})$ and $\beta'(\mathrm{t}) = \beta(\mathrm{t}) = \mathbf{true}$ and $\zeta(\mathrm{t}) \subseteq \zeta'(\mathrm{t}) \subseteq (\mathsf{inObjThrds}(W,\mathcal{S})\backslash\{\mathrm{t}\})$;
(e) for any $\mathrm{t}' \in \mathsf{inObjThrds}(W,\mathcal{S})\backslash\{\mathrm{t}\}$, we have:
either $\mathrm{t}' \in \mathsf{tidset}(T)$,
or $\mathscr{M}'(\mathrm{t}') < \mathscr{M}(\mathrm{t}')$,
or $\mathscr{M}'(\mathrm{t}') = \mathscr{M}(\mathrm{t}')$ and $\beta'(\mathrm{t}') = \beta(\mathrm{t}') = \mathbf{false}$,
or $\mathscr{M}'(\mathrm{t}') = \mathscr{M}(\mathrm{t}')$ and $\beta'(\mathrm{t}') = \beta(\mathrm{t}') = \mathbf{true}$ and $\zeta(\mathrm{t}') \subseteq \zeta'(\mathrm{t}') \subseteq (\mathsf{inObjThrds}(W,\mathcal{S})\backslash\{\mathrm{t}'\})$ and $\mathrm{t} \notin \zeta(\mathrm{t}')$;
(f) for any $\mathrm{t}' \in \mathsf{inObjThrds}(W,\mathcal{S})\backslash\{\mathrm{t}\}$ and is_await$(W|_{\mathrm{t}'})$, we have:
if $((\chi = \mathsf{sfair}) \wedge (\mathrm{t}' \in \mathsf{bset}(W,\mathcal{S})) \wedge (\mathrm{t}' \in \mathsf{bset}(W',\mathcal{S}'))) \vee (\chi = \mathsf{wfair})$, then
either $\mathrm{t}' \in \mathsf{tidset}(T)$,
or $\alpha'(\mathrm{t}') < \alpha(\mathrm{t}')$,
or $\alpha'(\mathrm{t}') = \alpha(\mathrm{t}')$ and $\zeta_a(\mathrm{t}') \subseteq \zeta'_a(\mathrm{t}') \subseteq (\mathsf{inObjThrds}(W,\mathcal{S})\backslash\{\mathrm{t}'\})$ and $\mathrm{t} \notin \zeta_a(\mathrm{t}')$ and $(\zeta_a(\mathrm{t}') \neq \emptyset) \vee (\zeta_a(\mathrm{t}') = \emptyset \wedge \mathrm{t}' \notin \mathsf{ffset}(\mathscr{B},\mathbb{S})) \vee (\zeta_a(\mathrm{t}') = \emptyset \wedge \mathrm{t}' \in \mathsf{ffset}(\mathscr{B}',\mathbb{S}'))$.

Proof: By the operational semantics, we know one of the two cases holds:

- there exist t, $\hat{C}'_{\mathrm{t}}$, W', $\mathcal{S}'$, e, Δ_c and Δ_o such that
$$\hat{C}_{\mathrm{t}} = \mathbf{skip},\ \mathcal{K}(\mathrm{t}) = \circ,\ \hat{C}'_{\mathrm{t}} = \mathbf{end},\ \iota = (e,\Delta_c,\Delta_o),\ e = (\mathrm{t},\mathbf{term}),$$
$$W' = (\mathbf{let}\ \Pi\ \mathbf{in}\ \hat{C}_1 \parallel \ldots \hat{C}'_{\mathrm{t}} \ldots \parallel \hat{C}_n),\ \mathcal{S}' = \mathcal{S},\ \mathsf{btids}(W,\mathcal{S}) = (\Delta_c,\Delta_o)\ .$$
By $\mathcal{D},R,G \models^{\chi}_{\mathrm{t}} (\Pi,\hat{C}_{\mathrm{t}},(\sigma_c,\sigma \uplus \sigma_{\mathrm{t}},\kappa_{\mathrm{t}})) \lesssim (\Pi',\Gamma,\mathbb{B}_{\mathrm{t}},\hat{\mathbb{C}}_{\mathrm{t}},(\Sigma \uplus \Sigma_{\mathrm{t}},\mathbb{k}_{\mathrm{t}})) \diamond (\mathbb{M}_{\mathrm{t}},M_{\mathrm{t}},wb_{\mathrm{t}},aw_{\mathrm{t}}) \Downarrow_{\xi_{\mathrm{t}},\xi^a_{\mathrm{t}}} P$, we know $\hat{\mathbb{C}}_{\mathrm{t}} = \mathbf{skip}$ and $\mathbb{k}_{\mathrm{t}} = \circ$. Let $\hat{\mathbb{C}}'_{\mathrm{t}} = \mathbf{end}$, $\mathsf{btids}(\mathbb{W},\mathbb{S}) = (\Delta'_c,\Delta'_o)$, $\iota' = (e,\Delta'_c,\Delta'_o)$ and $\mathbb{W}' = (\mathbf{let}\ \Gamma\ \mathbf{in}\ \hat{\mathbb{C}}_1 \parallel \ldots \hat{\mathbb{C}}'_{\mathrm{t}} \ldots \parallel \hat{\mathbb{C}}_n)$. Then $(\mathbb{W},\mathbb{S}) \overset{\iota'}{\longmapsto} (\mathbb{W}',\mathbb{S})$. From the premise, we know $\forall \mathrm{t}'.\ (\kappa_{\mathrm{t}'} = \circ \Longrightarrow \hat{C}_{\mathrm{t}'} = \hat{\mathbb{C}}_{\mathrm{t}'}) \wedge (\kappa_{\mathrm{t}'} = \circ \Longleftrightarrow \mathbb{k}_{\mathrm{t}'} = \circ)$, thus
$$\Delta'_c = \Delta_c.$$
Also we know
$$((\sigma,\Sigma),(\sigma,\Sigma),0) \models G_{\mathrm{t}}.$$
For any $\mathrm{t}' \neq \mathrm{t}$, since $G_{\mathrm{t}} \Rightarrow R_{\mathrm{t}'}$, we have $((\sigma,\Sigma),(\sigma,\Sigma),0) \models R_{\mathrm{t}'}$. Thus
$$((\sigma \uplus \sigma_{\mathrm{t}'},\Sigma \uplus \Sigma_{\mathrm{t}'}),(\sigma \uplus \sigma_{\mathrm{t}'},\Sigma \uplus \Sigma_{\mathrm{t}'}),0) \models R_{\mathrm{t}'} * \mathsf{Id}.$$
By $\mathcal{D},R,G \models^{\chi}_{\mathrm{t}'} (\Pi,\hat{C}_{\mathrm{t}'},(\sigma_c,\sigma \uplus \sigma_{\mathrm{t}'},\kappa_{\mathrm{t}'})) \lesssim (\Pi',\Gamma,\mathbb{B}_{\mathrm{t}'},\hat{\mathbb{C}}_{\mathrm{t}'},(\Sigma \uplus \Sigma_{\mathrm{t}'},\mathbb{k}_{\mathrm{t}'})) \diamond (\mathbb{M}_{\mathrm{t}'},M_{\mathrm{t}'},wb_{\mathrm{t}'},aw_{\mathrm{t}'}) \Downarrow_{\xi_{\mathrm{t}'},\xi^a_{\mathrm{t}'}} P$, and by the co-induction hypothesis, we can finish the case.
- there exist t, $\hat{C}'_{\mathrm{t}}$, W', σ'_c, σ'', κ'_{t}, $\mathcal{K}'$, $\mathcal{S}'$, e, Δ_c and Δ_o such that
$(\hat{C}_{\mathrm{t}},(\sigma_c,\sigma \uplus (\biguplus_{\mathrm{t}} \sigma_{\mathrm{t}}),\kappa_{\mathrm{t}})) \xrightarrow{e}_{\mathrm{t},\Pi} (\hat{C}'_{\mathrm{t}},(\sigma'_c,\sigma'',\kappa'_{\mathrm{t}})))$, $\mathcal{K}' = \mathcal{K}\{\mathrm{t} \rightsquigarrow \kappa'_{\mathrm{t}}\}$, $\mathcal{S}' = (\sigma'_c,\sigma'',\mathcal{K}')$,
$W' = (\mathbf{let}\ \Pi\ \mathbf{in}\ \hat{C}_1 \parallel \ldots \hat{C}'_{\mathrm{t}} \ldots \parallel \hat{C}_n)$, $\iota = (e,\Delta_c,\Delta_o)$, $\mathrm{t} = \mathsf{tid}(e)$, $\mathsf{btids}(W',\mathcal{S}') = (\Delta_c,\Delta_o)$.
By $\mathcal{D},R,G \models^{\chi}_{\mathrm{t}} (\Pi,\hat{C}_{\mathrm{t}},(\sigma_c,\sigma \uplus \sigma_{\mathrm{t}},\kappa_{\mathrm{t}})) \lesssim (\Pi',\Gamma,\mathbb{B}_{\mathrm{t}},\hat{\mathbb{C}}_{\mathrm{t}},(\Sigma \uplus \Sigma_{\mathrm{t}},\mathbb{k}_{\mathrm{t}})) \diamond (\mathbb{M}_{\mathrm{t}},M_{\mathrm{t}},wb_{\mathrm{t}},aw_{\mathrm{t}}) \Downarrow_{\xi_{\mathrm{t}},\xi^a_{\mathrm{t}}} P$, we know
(A) For any $\mathrm{t}' \in \xi_{\mathrm{t}} \cup \xi^a_{\mathrm{t}}$, we have $\mathrm{t}' \neq \mathrm{t}$ and $(\sigma \uplus \sigma_{\mathrm{t}},\Sigma \uplus \Sigma_{\mathrm{t}}) \models \mathsf{Enabled}(\mathcal{D}_{\mathrm{t}'}) * \mathsf{true}$.
And there exist n, σ''', $\mathcal{E}$, $\hat{\mathbb{C}}'_{\mathrm{t}}$, Σ''', $\mathbb{k}'_{\mathrm{t}}$, k, $\mathbb{B}'_{\mathrm{t}}$, $\mathbb{M}'_{\mathrm{t}}$, M'_{t}, wb'_{t}, aw'_{t}, ξ'_{t} and ξ^{a1}_{t} such that
(B) $\sigma'' = \sigma''' \uplus (\biguplus_{\mathrm{t}' \neq \mathrm{t}} \sigma_{\mathrm{t}'})$, and

(C) $(\hat{\mathbb{C}}_t, (\sigma_c, \Sigma \uplus (\biguplus_t \Sigma_t), \Bbbk_t)) \xrightarrow{\mathcal{E}}{}^{n}_{t,\Pi'} (\hat{\mathbb{C}}'_t, (\sigma'_c, \Sigma''' \uplus (\biguplus_{t' \neq t} \Sigma_{t'}), \Bbbk'_t))$; and
if $k > 0$ and $\mathsf{inObj}(\hat{C}_t, \kappa_t)$, then
$\mathsf{exec0}_t((\hat{\mathbb{C}}_t, (\sigma_c, \Sigma \uplus (\biguplus_t \Sigma_t), \Bbbk_t)), (\hat{\mathbb{C}}'_t, (\sigma'_c, \Sigma''' \uplus (\biguplus_{t' \neq t} \Sigma_{t'}), \Bbbk'_t)))$ or $\mathsf{exec1}_t((\hat{\mathbb{C}}_t, (\sigma_c, \Sigma \uplus (\biguplus_t \Sigma_t), \Bbbk_t)), (\hat{\mathbb{C}}'_t, (\sigma'_c, \Sigma''' \uplus (\biguplus_{t' \neq t} \Sigma_{t'}), \Bbbk'_t)))$; and
if $e = (t, f, \_)$, then $\mathbb{B}'_t = \mathsf{getB}(\Gamma, f)$, else if $\mathsf{inObj}(\hat{C}_t, \kappa_t) \wedge \mathsf{is\_while1}(\hat{\mathbb{C}}'_t)$, then $\mathbb{B}'_t = \mathbf{true}$, else $\mathbb{B}'_t = \mathbb{B}_t$; and

(D) $\mathsf{is\_clt}(e) \vee \mathsf{is\_inv}(e) \vee \mathsf{is\_ret}(e) \Longrightarrow \mathcal{E} = e :: \epsilon$, and
$\mathsf{is\_obj}(e) \Longrightarrow (\forall i, e'. \ \mathcal{E}(i). \ \mathsf{is\_obj}(e'))$, and

(E) $\mathcal{D}, R, G \models^{\chi}_{t} (\Pi, \hat{C}'_t, (\sigma'_c, \sigma''', \kappa'_t)) \lesssim (\Pi', \Gamma, \mathbb{B}'_t, \hat{\mathbb{C}}'_t, (\Sigma''', \Bbbk'_t)) \diamond (\mathbb{M}'_t, M'_t, wb'_t, aw'_t) \Downarrow_{\xi'_t, \xi^{a1}_t} P$, and

(F) $((\sigma \uplus \sigma_t, \Sigma \uplus \Sigma_t), (\sigma''', \Sigma'''), k) \models G_t * [\mathsf{done} = \mathsf{false}] * \mathsf{True}$, and

(G) if $\mathsf{inObj}(\hat{C}_t, \kappa_t)$, then $n > 0$, or $\mathbb{M}'_t < \mathbb{M}_t$, or $\mathbb{M}'_t = \mathbb{M}_t$ and $wb'_t = wb_t = \mathbf{true}$ and $\xi_t \subseteq \xi'_t$; and

(H) if $((\sigma \uplus \sigma_t, \Sigma \uplus \Sigma_t), (\sigma''', \Sigma''')) \models \langle [\mathcal{D}_t] \rangle * \mathsf{True}$ and $k = 0$, then $M'_t < M_t$.
Since $(\sigma, \Sigma) \models I$ and $I \triangleright \{R, G\}$, we know there exist σ', σ'_t, Σ' and Σ'_t such that
$$\sigma''' = \sigma' \uplus \sigma'_t, \quad \Sigma''' = \Sigma' \uplus \Sigma'_t, \quad (\sigma', \Sigma') \models I \quad \text{and} \quad ((\sigma, \Sigma), (\sigma', \Sigma'), k) \models G_t.$$
For any $t' \neq t$, since $G_t \Rightarrow R_{t'}$, we have $((\sigma, \Sigma), (\sigma', \Sigma'), k) \models R_{t'}$. Thus
$$((\sigma \uplus \sigma_{t'}, \Sigma \uplus \Sigma_{t'}), (\sigma' \uplus \sigma_{t'}, \Sigma' \uplus \Sigma_{t'}), k) \models R_{t'} * \mathsf{Id}.$$
By $\mathcal{D}, R, G \models^{\chi}_{t'} (\Pi, \hat{C}_{t'}, (\sigma_c, \sigma \uplus \sigma_{t'}, \kappa_{t'})) \lesssim (\Pi', \Gamma, \mathbb{B}_{t'}, \hat{\mathbb{C}}_{t'}, (\Sigma \uplus \Sigma_{t'}, \Bbbk_{t'})) \diamond (\mathbb{M}_{t'}, M_{t'}, wb_{t'}, aw_{t'}) \Downarrow_{\xi_{t'}, \xi^{a}_{t'}} P$, we know

(I) For any $t'' \in \xi_{t'} \cup \xi^{a}_{t'}$, we have $t'' \neq t'$ and $(\sigma \uplus \sigma_{t'}, \Sigma \uplus \Sigma_{t'}) \models \mathsf{Enabled}(\mathcal{D}_{t''}) * \mathsf{true}$.
And there exist $\mathbb{B}'_{t'}$, $\mathbb{M}'_{t'}$, $M'_{t'}$, $wb'_{t'}$, $aw'_{t'}$, $\xi^{d}_{t'}$, $\xi^{ad}_{t'}$, $\xi'_{t'}$ and $\xi^{a1}_{t'}$ such that $\mathbb{B}'_{t'} = \mathbb{B}_{t'}$ and

(J) $\mathcal{D}, R, G \models^{\chi}_{t'} (\Pi, \hat{C}_{t'}, (\sigma'_c, \sigma' \uplus \sigma_{t'}, \kappa_{t'})) \lesssim (\Pi', \Gamma, \mathbb{B}_{t'}, \hat{\mathbb{C}}_{t'}, (\Sigma' \uplus \Sigma_{t'}, \Bbbk_{t'})) \diamond (\mathbb{M}'_{t'}, M'_{t'}, wb'_{t'}, aw'_{t'}) \Downarrow_{\xi'_{t'}, \xi^{a1}_{t'}} P$, and

(K) if $k > 0$ and $\mathsf{inObj}(\hat{C}_{t'}, \kappa_{t'})$, then for any σ''_c, Σ'' and Σ_F such that $\Sigma'' = \Sigma\{\mathsf{done} \rightsquigarrow \mathbf{true}\}$ and $\Sigma_F \perp \Sigma$ we have $(\hat{\mathbb{C}}_{t'}, (\sigma''_c, \Sigma'' \uplus \Sigma_{t'} \uplus \Sigma_F, \Bbbk_{t'})) \longrightarrow^{+}_{t',\Pi'} (\hat{\mathbb{C}}_{t'}, (\sigma''_c, \Sigma'' \uplus \Sigma_{t'} \uplus \Sigma_F, \Bbbk_{t'}))$;

(L) $\xi^{d}_{t'} = \{t'' \mid (t'' \in \xi_{t'}) \wedge (((\sigma \uplus \sigma_{t'}, \Sigma \uplus \Sigma_{t'}), (\sigma' \uplus \sigma_{t'}, \Sigma' \uplus \Sigma_{t'})) \models \langle \mathcal{D}_{t''} \rangle * \mathsf{Id})\}$ and
$k = 0 \wedge \mathsf{inObj}(\hat{C}_{t'}, \kappa_{t'}) \Longrightarrow \mathbb{M}'_{t'} < \mathbb{M}_{t'} \vee (\mathbb{M}'_{t'} = \mathbb{M}_{t'} \wedge wb'_{t'} = wb_{t'})$ and
$k = 0 \wedge wb_{t'} = \mathbf{true} \wedge (\xi^{d}_{t'} \neq \emptyset \vee ((\Sigma \uplus \Sigma_{t'}, \Bbbk_{t'}) \models \neg \mathbb{B}_{t'} \wedge (\Sigma' \uplus \Sigma_{t'}, \Bbbk_{t'}) \models \mathbb{B}_{t'})) \Longrightarrow \mathbb{M}'_{t'} < \mathbb{M}_{t'}$ and
$k = 0 \wedge \mathbb{M}'_{t'} = \mathbb{M}_{t'} \wedge wb'_{t'} = wb_{t'} = \mathbf{true} \Longrightarrow \xi_{t'} \backslash \xi^{d}_{t'} \subseteq \xi'_{t'}$, and

(M) if $(\sigma \uplus \sigma_{t'}, \Sigma \uplus \Sigma_{t'}) \models \mathsf{Enabled}(\mathcal{D}_{t'}) * \mathsf{true}$ and $k = 0$, then $M'_{t'} \leq M_{t'}$; and

(N) $\xi^{ad}_{t'} = \{t'' \mid (t'' \in \xi^{a}_{t'}) \wedge (((\sigma \uplus \sigma_{t'}, \Sigma \uplus \Sigma_{t'}), (\sigma' \uplus \sigma'_{t'}, \Sigma' \uplus \Sigma_{t'})) \models \langle \mathcal{D}_{t''} \rangle * \mathsf{Id})\}$ and
$k = 0 \wedge \mathsf{inObj}(\hat{C}_{t'}, \kappa_{t'}) \wedge \mathsf{is\_await}(\hat{C}_{t'}) \Longrightarrow \xi^{a}_{t'} \backslash \xi^{ad}_{t'} \subseteq \xi^{a1}_{t'}$ and
$k = 0 \wedge \mathsf{inObj}(\hat{C}_{t'}, \kappa_{t'}) \wedge \mathsf{is\_await}(\hat{C}_{t'}) \wedge (\xi^{ad}_{t'} \neq \emptyset \vee ((\Sigma \uplus \Sigma_{t'}, \Bbbk_{t'}) \models \neg \mathbb{B}_{t'} \wedge (\Sigma' \uplus \Sigma_{t'}, \Bbbk_{t'}) \models \mathbb{B}_{t'})) \Longrightarrow aw'_{t'} < aw_{t'}$ and
$k = 0 \wedge (\chi = \mathsf{sfair}) \wedge \mathsf{inObj}(\hat{C}_{t'}, \kappa_{t'}) \wedge \mathsf{is\_await}(\hat{C}_{t'}) \wedge ((\sigma \uplus \sigma_{t'}, \kappa_{t'}) \models \neg \mathsf{en}(\hat{C}_{t'})) \wedge ((\sigma' \uplus \sigma_{t'}, \kappa_{t'}) \models \neg \mathsf{en}(\hat{C}_{t'})) \Longrightarrow aw'_{t'} \leq aw_{t'}$ and
$k = 0 \wedge (\chi = \mathsf{wfair}) \wedge \mathsf{inObj}(\hat{C}_{t'}, \kappa_{t'}) \wedge \mathsf{is\_await}(\hat{C}_{t'}) \Longrightarrow aw'_{t'} \leq aw_{t'}$.

Let $\mathbb{W}' = (\mathbf{let}\ \Pi'\ \mathbf{in}\ \hat{\mathbb{C}}_1 \parallel \ldots \hat{\mathbb{C}}'_t \ldots \parallel \hat{\mathbb{C}}_n)$ and $\mathbb{S}' = (\sigma'_c, \Sigma' \uplus \Sigma'_t \uplus (\biguplus_{t' \neq t} \Sigma_{t'}), \mathbb{K}\{t \rightsquigarrow \Bbbk'_t\})$.
From (C) and (K), we know there exists T such that
$$(\mathbb{W}, \mathbb{S})) \xrightarrow{T}{}^{*} (\mathbb{W}', \mathbb{S}'), \quad \forall t' \in \mathsf{inObjThrds}(W, \mathcal{S}) \backslash \{t\}. \ k > 0 \Longrightarrow t' \in \mathsf{tidset}(T).$$
From (D) and (K), we know
$$\mathsf{is\_clt}(e) \vee \mathsf{is\_inv}(e) \vee \mathsf{is\_ret}(e) \Longrightarrow T = (e, \Delta_c, \_) :: \epsilon,$$
$$\mathsf{is\_obj}(e) \Longrightarrow (\forall i, \iota' = T(i). \ \mathsf{is\_obj}(\iota')).$$

Define the functions $\mathscr{B}'$, $\mathscr{M}'$, ζ', β', α' and ζ'_a as follows.

- $dom(\mathscr{B}') = dom(\mathscr{M}') = dom(\zeta') = dom(\beta') = dom(\alpha') = dom(\zeta'_a) = \mathsf{inObjThrds}(W', \mathcal{S}')$.
- For any $\mathsf{t} \in dom(\mathscr{M}')$, we have
  - $\mathscr{B}'(\mathsf{t}) = \mathbb{B}'_{\mathsf{t}}$;
  - $\mathscr{M}'(\mathsf{t}) = (\mathbb{M}'_{\mathsf{t}}, \{\mathsf{t}' \rightsquigarrow M'_{\mathsf{t}'} \mid \mathsf{t}' \in \xi'_{\mathsf{t}}\})$;
  - $\zeta'(\mathsf{t}) = \xi'_{\mathsf{t}}$;
  - $\beta'(\mathsf{t}) = wb'_{\mathsf{t}}$;
  - $\alpha'(\mathsf{t}) = (aw'_{\mathsf{t}}, \{\mathsf{t}' \rightsquigarrow M'_{\mathsf{t}'} \mid \mathsf{t}' \in \xi^{a1}_{\mathsf{t}}\})$;
  - $\zeta'_a(\mathsf{t}) = \xi^{a1}_{\mathsf{t}}$.

Then by the co-induction hypothesis, we know

$$\models_\chi (W', \mathcal{S}') \lesssim (\mathbb{W}', \Gamma, \mathscr{B}', \mathbb{S}') \diamond (\mathscr{M}', \zeta', \beta', \alpha', \zeta'_a).$$

For the thread t, from (G), we know: if $\mathsf{t} \in \mathsf{inObjThrds}(W, \mathcal{S})$, then one of the following holds:

- $n > 0$. Thus $\mathsf{t} \in \mathsf{tidset}(T)$.
- $\mathbb{M}'_{\mathsf{t}} < \mathbb{M}_{\mathsf{t}}$. Thus $\mathscr{M}'(\mathsf{t}) < \mathscr{M}(\mathsf{t})$.
- $\mathbb{M}'_{\mathsf{t}} = \mathbb{M}_{\mathsf{t}}$ and $wb'_{\mathsf{t}} = wb_{\mathsf{t}} = \mathbf{true}$ and $\xi_{\mathsf{t}} \subseteq \xi'_{\mathsf{t}}$.
  Thus $\beta'(\mathsf{t}) = \beta(\mathsf{t}) = \mathbf{true}$ and $\zeta(\mathsf{t}) \subseteq \zeta'(\mathsf{t})$. Also we know $\xi'_{\mathsf{t}} \subseteq (\mathsf{inObjThrds}(W, \mathcal{S}) \backslash \{\mathsf{t}\})$. Then we only need to show the following (B.12):

$$\{\mathsf{t}' \rightsquigarrow M'_{\mathsf{t}'} \mid \mathsf{t}' \in \xi'_{\mathsf{t}}\} \;\leq\; \{\mathsf{t}' \rightsquigarrow M_{\mathsf{t}'} \mid \mathsf{t}' \in \xi_{\mathsf{t}}\} \tag{B.12}$$

  From (A), since $\forall \mathsf{t}'.\ \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}'}) \Rightarrow I$ and $\mathsf{Precise}(I)$, we know: for any $\mathsf{t}' \in \xi_{\mathsf{t}}$,

$$\mathsf{t}' \neq \mathsf{t} \quad \text{and} \quad (\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}'})\ .$$

  Thus $(\sigma \uplus \sigma_{\mathsf{t}'}, \Sigma \uplus \Sigma_{\mathsf{t}'}) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}'}) * \mathsf{true}$. From (M), we have $M'_{\mathsf{t}'} \leq M_{\mathsf{t}'}$. Thus (B.12) holds.

For any $\mathsf{t}' \in \mathsf{inObjThrds}(W, \mathcal{S}) \backslash \{\mathsf{t}\}$, from (K) and (L), we know one of the following holds:

- $k > 0$. Then $\mathsf{t}' \in \mathsf{tidset}(T)$.
- $\mathbb{M}'_{\mathsf{t}'} < \mathbb{M}_{\mathsf{t}'}$. Thus $\mathscr{M}'(\mathsf{t}') < \mathscr{M}(\mathsf{t}')$.
- $\mathbb{M}'_{\mathsf{t}'} = \mathbb{M}_{\mathsf{t}'}$ and $wb'_{\mathsf{t}'} = wb_{\mathsf{t}'} = \mathbf{false}$.
  Thus $\beta'(\mathsf{t}') = \beta(\mathsf{t}') = \mathbf{false}$. Then from the premise, we know

$$\xi_{\mathsf{t}'} = \xi'_{\mathsf{t}'} = \emptyset.$$

  Thus the following (B.13) holds:

$$\{\mathsf{t}'' \rightsquigarrow M'_{\mathsf{t}''} \mid \mathsf{t}'' \in \xi'_{\mathsf{t}'}\} \;=\; \{\mathsf{t}'' \rightsquigarrow M_{\mathsf{t}''} \mid \mathsf{t}'' \in \xi_{\mathsf{t}'}\} \tag{B.13}$$

  Thus $\mathscr{M}'(\mathsf{t}') = \mathscr{M}(\mathsf{t}')$ holds.
- $\mathbb{M}'_{\mathsf{t}'} = \mathbb{M}_{\mathsf{t}'}$ and $wb'_{\mathsf{t}'} = wb_{\mathsf{t}'} = \mathbf{true}$.
  Thus $\beta'(\mathsf{t}') = \beta(\mathsf{t}') = \mathbf{true}$. From (L), we know

$$\xi^d_{\mathsf{t}'} = \emptyset \wedge ((\Sigma \uplus \Sigma_{\mathsf{t}'}, \Bbbk_{\mathsf{t}'}) \models \mathbb{B}_{\mathsf{t}'} \vee (\Sigma' \uplus \Sigma_{\mathsf{t}'}, \Bbbk_{\mathsf{t}'}) \models \neg \mathbb{B}_{\mathsf{t}'}).$$

  Also, since $\xi_{\mathsf{t}'} \backslash \xi^d_{\mathsf{t}'} \subseteq \xi'_{\mathsf{t}'}$, we know

$$\xi_{\mathsf{t}'} \subseteq \xi'_{\mathsf{t}'}.$$

  One of the following holds:
  - $\mathsf{t} \notin \xi_{\mathsf{t}'}$. Thus $\mathsf{t} \notin \zeta(\mathsf{t}')$ and $\zeta(\mathsf{t}') \subseteq \zeta'(\mathsf{t}') \subseteq (\mathsf{inObjThrds}(W, \mathcal{S}) \backslash \{\mathsf{t}'\})$. We only need to show the following (B.14):

$$\{\mathsf{t}'' \rightsquigarrow M'_{\mathsf{t}''} \mid \mathsf{t}'' \in \xi'_{\mathsf{t}'}\} \;\leq\; \{\mathsf{t}'' \rightsquigarrow M_{\mathsf{t}''} \mid \mathsf{t}'' \in \xi_{\mathsf{t}'}\} \tag{B.14}$$

    From (I), we know: for any $\mathsf{t}'' \in \xi_{\mathsf{t}'}$,

$$\mathsf{t}'' \neq \mathsf{t}' \quad \text{and} \quad (\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}''})\ .$$

    Thus $(\sigma \uplus \sigma_{\mathsf{t}''}, \Sigma \uplus \Sigma_{\mathsf{t}''}) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}''}) * \mathsf{true}$. From (M), we have $M'_{\mathsf{t}''} \leq M_{\mathsf{t}''}$. Thus (B.14) holds.

- $\mathsf{t} \in \xi_{\mathsf{t}'}$. Since $\mathsf{wffAct}(R, \mathcal{D})$, we know
$$((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle [\mathcal{D}_{\mathsf{t}}] \rangle .$$
From (H), we know
$$M'_{\mathsf{t}} < M_{\mathsf{t}}.$$
We only need to show the following (B.15):
$$\{\mathsf{t}'' \rightsquigarrow M'_{\mathsf{t}''} \mid \mathsf{t}'' \in \xi'_{\mathsf{t}'}\} \ < \ \{\mathsf{t}'' \rightsquigarrow M_{\mathsf{t}''} \mid \mathsf{t}'' \in \xi_{\mathsf{t}'}\} \tag{B.15}$$
For any $\mathsf{t}'' \in \xi_{\mathsf{t}'} \backslash \{\mathsf{t}\}$, from (I), we know:
$$\mathsf{t}'' \neq \mathsf{t}' \quad \text{and} \quad (\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}''}) .$$
Thus $(\sigma \uplus \sigma_{\mathsf{t}''}, \Sigma \uplus \Sigma_{\mathsf{t}''}) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}''}) * \mathsf{true}$. From (M), we have $M'_{\mathsf{t}''} \leq M_{\mathsf{t}''}$. Thus (B.15) holds. Thus $\mathscr{M}'(\mathsf{t}') < \mathscr{M}(\mathsf{t}')$ holds.

For any $\mathsf{t}' \in \mathsf{inObjThrds}(W, \mathcal{S}) \backslash \{\mathsf{t}\}$ and $\mathsf{is\_await}(W|_{\mathsf{t}'})$, we have:
if $((\chi = \mathsf{sfair}) \wedge (\mathsf{t}' \in \mathsf{bset}(W, \mathcal{S})) \wedge (\mathsf{t}' \in \mathsf{bset}(W', \mathcal{S}'))) \vee (\chi = \mathsf{wfair})$, then from (K) and (N), we know one of the following holds:

- $k > 0$. Then $\mathsf{t}' \in \mathsf{tidset}(T)$.
- $aw'_{\mathsf{t}'} < aw_{\mathsf{t}'}$. Thus $\alpha'(\mathsf{t}') < \alpha(\mathsf{t}')$.
- $aw'_{\mathsf{t}'} = aw_{\mathsf{t}'}$. From (M), we know
$$\xi^{ad}_{\mathsf{t}'} = \emptyset \wedge ((\Sigma \uplus \Sigma_{\mathsf{t}'}, \mathbb{k}_{\mathsf{t}'}) \models \mathbb{B}_{\mathsf{t}'} \vee (\Sigma' \uplus \Sigma_{\mathsf{t}'}, \mathbb{k}_{\mathsf{t}'}) \models \neg \mathbb{B}_{\mathsf{t}'}).$$
Also since $\xi^{a}_{\mathsf{t}'} \backslash \xi^{ad}_{\mathsf{t}'} \subseteq \xi^{a1}_{\mathsf{t}'}$, we know
$$\xi^{a}_{\mathsf{t}'} \subseteq \xi^{a1}_{\mathsf{t}'}.$$
One of the following holds:
  - $\mathsf{t} \notin \xi^{a}_{\mathsf{t}'}$. Thus $\mathsf{t} \notin \zeta_a(\mathsf{t}')$ and $\zeta_a(\mathsf{t}') \subseteq \zeta'_a(\mathsf{t}') \subseteq (\mathsf{inObjThrds}(W, \mathcal{S}) \backslash \{\mathsf{t}'\})$. Also we know $(\zeta_a(\mathsf{t}') \neq \emptyset) \vee (\zeta_a(\mathsf{t}') = \emptyset \wedge \mathsf{t}' \notin \mathsf{ffset}(\mathscr{B}, \mathbb{S})) \vee (\zeta_a(\mathsf{t}') = \emptyset \wedge \mathsf{t}' \in \mathsf{ffset}(\mathscr{B}', \mathbb{S}'))$.
  We only need to show the following (B.16):
  $$\{\mathsf{t}'' \rightsquigarrow M'_{\mathsf{t}''} \mid \mathsf{t}'' \in \xi^{a1}_{\mathsf{t}'}\} \ \leq \ \{\mathsf{t}'' \rightsquigarrow M_{\mathsf{t}''} \mid \mathsf{t}'' \in \xi^{a}_{\mathsf{t}'}\} \tag{B.16}$$
  From (I), we know: for any $\mathsf{t}'' \in \xi^{a}_{\mathsf{t}'}$,
  $$\mathsf{t}'' \neq \mathsf{t}' \quad \text{and} \quad (\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}''}) .$$
  Thus $(\sigma \uplus \sigma_{\mathsf{t}''}, \Sigma \uplus \Sigma_{\mathsf{t}''}) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}''}) * \mathsf{true}$. From (M), we have $M'_{\mathsf{t}''} \leq M_{\mathsf{t}''}$. Thus (B.16) holds.
  - $\mathsf{t} \in \xi^{a}_{\mathsf{t}'}$. Since $\mathsf{wffAct}(R, \mathcal{D})$, we know
  $$((\sigma, \Sigma), (\sigma', \Sigma')) \models \langle [\mathcal{D}_{\mathsf{t}}] \rangle .$$
  From (H), we know
  $$M'_{\mathsf{t}} < M_{\mathsf{t}}.$$
  We only need to show the following (B.17):
  $$\{\mathsf{t}'' \rightsquigarrow M'_{\mathsf{t}''} \mid \mathsf{t}'' \in \xi^{a1}_{\mathsf{t}'}\} \ < \ \{\mathsf{t}'' \rightsquigarrow M_{\mathsf{t}''} \mid \mathsf{t}'' \in \xi^{a}_{\mathsf{t}'}\} \tag{B.17}$$
  For any $\mathsf{t}'' \in \xi^{a}_{\mathsf{t}'} \backslash \{\mathsf{t}\}$, from (I), we know:
  $$\mathsf{t}'' \neq \mathsf{t}' \quad \text{and} \quad (\sigma, \Sigma) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}''}) .$$
  Thus $(\sigma \uplus \sigma_{\mathsf{t}''}, \Sigma \uplus \Sigma_{\mathsf{t}''}) \models \mathsf{Enabled}(\mathcal{D}_{\mathsf{t}''}) * \mathsf{true}$. From (M), we have $M'_{\mathsf{t}''} \leq M_{\mathsf{t}''}$. Thus (B.17) holds. Thus $\alpha'(\mathsf{t}') < \alpha(\mathsf{t}')$ holds.

Thus we are done. □

B.8 Towards Simulations with Fixed Low-Level Traces

As in the proof of LiLi, we extend each event e in a trace to include the information that can uniquely determine a step (for instance, to include the value written to x for the non-deterministic instruction `x := rand()`). We overload the notations e, ι, $\mathcal{E}$ and T.

$$\frac{(W,\mathcal{S}) \overset{T}{\longmapsto}^{+} (W',\mathcal{S}') \quad \text{get\_obsv}(T) = \epsilon \quad T' \models_\Delta O^{\text{co}-\omega}_{\text{sfair}}(W',\mathcal{S},\epsilon) \quad \text{activeThrds}(W) - \Delta \subseteq \text{tidset}(T) \quad \Delta \subseteq \text{bset}(W,\mathcal{S}) \quad \forall i,\iota = T(i).\ \Delta \subseteq \text{bset}(\iota)}{T :: T' \models_\Delta O^{\text{co}-\omega}_{\text{sfair}}(W,\mathcal{S},\epsilon)}$$

$$\frac{(W,\mathcal{S}) \overset{T}{\longmapsto}^{+} (W',\mathcal{S}') \quad \text{get\_obsv}(T) = e :: \mathcal{E} \quad T' \models_\Delta O^{\text{co}-\omega}_{\text{sfair}}(W',\mathcal{S},\mathcal{E}') \quad \text{activeThrds}(W) - \Delta \subseteq \text{tidset}(T) \quad \Delta \subseteq \text{bset}(W,\mathcal{S}) \quad \forall i,\iota = T(i).\ \Delta \subseteq \text{bset}(\iota)}{T :: T' \models_\Delta O^{\text{co}-\omega}_{\text{sfair}}(W,\mathcal{S},e :: \mathcal{E} :: \mathcal{E}')}$$

(a) χ = sfair

$$\frac{(W,\mathcal{S}) \overset{T}{\longmapsto}^{+} (W',\mathcal{S}') \quad \text{get\_obsv}(T) = \epsilon \quad T' \models_\Delta O^{\text{co}-\omega}_{\text{wfair}}(W',\mathcal{S},\epsilon) \quad \text{activeThrds}(W) - \Delta = \text{tidset}(T) \quad \forall \text{t} \in \Delta.\ \exists i.\ \text{t} \in \text{bset}(T(i))}{T :: T' \models_\Delta O^{\text{co}-\omega}_{\text{wfair}}(W,\mathcal{S},\epsilon)}$$

$$\frac{(W,\mathcal{S}) \overset{T}{\longmapsto}^{+} (W',\mathcal{S}') \quad \text{get\_obsv}(T) = e :: \mathcal{E} \quad T' \models_\Delta O^{\text{co}-\omega}_{\text{wfair}}(W',\mathcal{S},\mathcal{E}') \quad \text{activeThrds}(W) - \Delta = \text{tidset}(T) \quad \forall \text{t} \in \Delta.\ \exists i.\ \text{t} \in \text{bset}(T(i))}{T :: T' \models_\Delta O^{\text{co}-\omega}_{\text{wfair}}(W,\mathcal{S},e :: \mathcal{E} :: \mathcal{E}')}$$

(b) χ = wfair

Fig. 25. Co-inductive definitions for generating observable event traces of infinite and χ-fair executions.

We define $O^{\omega}_{\chi}[\![W,\mathcal{S}]\!]$ for the set of observable traces generated from *infinite* and χ-fair executions in $\mathcal{T}_{\omega}[\![W,\mathcal{S}]\!]$.

$$\mathcal{T}^{\omega}_{\omega}[\![W,\mathcal{S}]\!] \overset{\text{def}}{=} \{((\textbf{spawn},|W|),\Delta_c,\Delta_o) :: T \mid \text{btids}(W,\mathcal{S}) = (\Delta_c,\Delta_o) \wedge ((W,\mathcal{S}) \overset{T}{\longmapsto}^{\omega} \cdot)\}$$

$$O^{\omega}_{\chi}[\![W,\mathcal{S}]\!] \overset{\text{def}}{=} \{(\mathcal{E},T) \mid T \in \mathcal{T}^{\omega}_{\omega}[\![W,\mathcal{S}]\!] \wedge \chi(T) \wedge \text{get\_obsv}(T) = \mathcal{E}\} \qquad \chi \in \{\text{sfair},\text{wfair}\}$$

Then $O_{\chi}[\![W,\mathcal{S}]\!]$ is equivalent to the union of $O^{\omega}_{\chi}[\![W,\mathcal{S}]\!]$ and the set of observable traces generated from *finite* executions (note that all finite traces are χ-fair). Actually here we pair each observable trace $\mathcal{E}$ in $O_{\chi}[\![W,\mathcal{S}]\!]$ with the execution trace T generating $\mathcal{E}$.

$$\mathcal{T}^{\text{fin}}_{\omega}[\![W,\mathcal{S}]\!] \overset{\text{def}}{=} \{((\textbf{spawn},|W|),\Delta_c,\Delta_o) :: T \mid \text{btids}(W,\mathcal{S}) = (\Delta_c,\Delta_o) \wedge (((W,\mathcal{S}) \overset{T}{\longmapsto}^{*} \textbf{abort}) \vee \exists W',\mathcal{S}'.\ ((W,\mathcal{S}) \overset{T}{\longmapsto}^{*} (W',\mathcal{S})) \wedge \neg(\exists \iota.\ (W',\mathcal{S}') \overset{\iota}{\longmapsto} \_))\}$$

$$O_{\chi}[\![W,\mathcal{S}]\!] = O^{\omega}_{\chi}[\![W,\mathcal{S}]\!] \cup \{(\mathcal{E},T) \mid T \in \mathcal{T}^{\text{fin}}_{\omega}[\![W,\mathcal{S}]\!] \wedge \text{get\_obsv}(T) = \mathcal{E}\}$$

We observe that for each trace T in $O^{\omega}_{\chi}[\![W,\mathcal{S}]\!]$, from some point on, it can be cut into pieces where each piece (called a round) contains at least one step of every thread (unless the thread is not needed to execute in a χ-fair execution). In Fig. 25 we give the definition of $T \models_\Delta O^{\text{co}-\omega}_{\chi}(W,\mathcal{S},\mathcal{E})$, which says that T can be cut into such kind of pieces. This is useful to help derive the whole-program simulations with fixed low-level traces (see Definitions B.23 and B.24).

Definition B.23 (Simulation with fixed low-level traces (for whole programs with PSF objects)). $T \models (W,\mathcal{S}) \preceq (\mathbb{W},\mathbb{S}) \diamond \mathcal{M}$ is co-inductively defined as follows. Here $\mathcal{M} \in \mathit{ThrdID} \rightharpoonup \mathit{Metric}$. Whenever $T \models (W,\mathcal{S}) \preceq (\mathbb{W},\mathbb{S}) \diamond \mathcal{M}$ holds, then the following hold:

(1) $\mathit{dom}(\mathcal{M}) = \text{inObjThrds}(W,\mathcal{S}) = \text{inObjThrds}(\mathbb{W},\mathbb{S})$.

(2) If $(W,\mathcal{S}) \overset{\iota}{\longmapsto} (W',\mathcal{S}')$ and $T = \iota :: T'$, then
there exist t, e, Δ_c, Δ_o, T'', n, $\mathbb{W}'$, $\mathbb{S}'$ and $\mathcal{M}'$ such that all the following hold:
(a) $(\mathbb{W},\mathbb{S}) \overset{T''}{\longmapsto}^n (\mathbb{W}',\mathbb{S}')$, and $n = 0 \vee n = 1$;
(b) $\iota = (e,\Delta_c,\Delta_o)$; t = tid($e$);
is_clt(e) $\vee$ is_inv(e) $\vee$ is_ret(e) $\Longrightarrow T'' = (e,\Delta_c,\_) :: \epsilon$;
is_obj(e) $\Longrightarrow (\forall i, \iota' = T''(i).$ is_obj(ι'));
(c) $T' \models (W',\mathcal{S}') \preceq (\mathbb{W}',\mathbb{S}') \diamond \mathcal{M}'$;
(d) for any t′ $\in$ inObjThrds($W,\mathcal{S}$), we have:
either t′ $\in$ tidset(T''),
or $\mathcal{M}'$(t′) $<$ $\mathcal{M}$(t′),
or $\mathcal{M}'$(t′) = $\mathcal{M}$(t′) and t′ $\in$ bset($\mathbb{W},\mathbb{S}$) and t′ $\in$ bset($\mathbb{W}',\mathbb{S}'$).

Definition B.24 (Simulation with fixed low-level traces (for whole programs with PDF objects)). $T \models\!\models (W,\mathcal{S}) \preceq (\mathbb{W},\Gamma,\mathscr{B},\mathbb{S}) \diamond \mathcal{M}$ is co-inductively defined as follows. Here $\mathcal{M} \in$ *ThrdID* $\rightharpoonup$ *Metric*. Whenever $T \models\!\models (W,\mathcal{S}) \preceq (\mathbb{W},\Gamma,\mathscr{B},\mathbb{S}) \diamond \mathcal{M}$ holds, then the following hold:

(1) $dom(\mathcal{M})$ = inObjThrds($W,\mathcal{S}$) = inObjThrds($\mathbb{W},\mathbb{S}$);
for any t $\in$ ffset($\mathscr{B},\mathbb{S}$), we have is_while0($\mathbb{W}|_{\mathrm{t}},\mathscr{B}$(t)).
(2) If $(W,\mathcal{S}) \overset{\iota}{\longmapsto} (W',\mathcal{S}')$ and $T = \iota :: T'$, then
there exist t, e, Δ_c, Δ_o, T'', $\mathscr{B}'$, $\mathbb{W}'$, $\mathbb{S}'$ and $\mathcal{M}'$ such that all the following hold:
(a) $(\mathbb{W},\mathbb{S}) \overset{T''}{\longmapsto}^* (\mathbb{W}',\mathbb{S}')$;
(b) $\iota = (e,\Delta_c,\Delta_o)$; t = tid($e$);
is_clt(e) $\vee$ is_inv(e) $\vee$ is_ret(e) $\Longrightarrow T'' = (e,\Delta_c,\_) :: \epsilon$;
is_obj(e) $\Longrightarrow (\forall i, \iota' = T''(i).$ is_obj(ι'));
if $e = (\mathrm{t}, f, \_)$, then $\mathscr{B}' = \mathscr{B}\{\mathrm{t} \rightsquigarrow \mathrm{getB}(\Gamma, f)\}$, else if is_obj($e$) $\wedge$ is_while1($\mathbb{W}'|_{\mathrm{t}}$), then $\mathscr{B}' = \mathscr{B}\{\mathrm{t} \rightsquigarrow \mathbf{true}\}$, else $\mathscr{B}' = \mathscr{B}$;
(c) $T' \models\!\models (W',\mathcal{S}') \preceq (\mathbb{W}',\Gamma,\mathscr{B}',\mathbb{S}') \diamond \mathcal{M}'$;
(d) for any t′ $\in$ inObjThrds($W,\mathcal{S}$), we have:
either t′ $\in$ tidset(T''),
or $\mathcal{M}'$(t′) $<$ $\mathcal{M}$(t′),
or $\mathcal{M}'$(t′) = $\mathcal{M}$(t′) and t′ $\in$ ffset($\mathscr{B},\mathbb{S}$) and t′ $\in$ ffset($\mathscr{B}',\mathbb{S}'$).

Below we prove in Lemmas B.25, B.29, B.32 and B.33 that the whole-program simulations (Definitions B.19 and B.20) imply the simulations with fixed low-level traces (Definitions B.23 and B.23) if the low-level traces are strongly/weakly fair. We first define some useful predicates and ordering in Fig. 26.

Lemma B.25 (Towards Simulations with Fixed Low-Level Traces (for PSF Objects under Strong Fairness)). *If* $T \models_\Delta O^{\mathrm{co}-\omega}_{\mathrm{sfair}}(W,\mathcal{S},\mathcal{E})$ *and* $\models_{\mathrm{sfair}} (W,\mathcal{S}) \precsim (\mathbb{W},\mathbb{S}) \diamond (\mathscr{M},\zeta,\beta,\alpha,\zeta_a)$, *then there exists* $\mathcal{M}$ *such that* $T \models (W,\mathcal{S}) \preceq (\mathbb{W},\mathbb{S}) \diamond \mathcal{M}$.

Proof. We prove the following (B.18) by inversion over $T \models_\Delta O^{\mathrm{co}-\omega}_{\mathrm{sfair}}(W,\mathcal{S},\mathcal{E})$.

If $T \models_\Delta O^{\mathrm{co}-\omega}_{\mathrm{sfair}}(W,\mathcal{S},\mathcal{E})$, then there exists T_x such that roundsub(T, activeThrds(W) $-$ Δ, T_x). (B.18)

Also, by inversion over $\models_{\mathrm{sfair}} (W,\mathcal{S}) \precsim (\mathbb{W},\mathbb{S}) \diamond (\mathscr{M},\zeta,\beta,\alpha,\zeta_a)$, we know

$$dom(\mathscr{M}) = dom(\zeta) = dom(\beta) = dom(\alpha) = dom(\zeta_a) = \mathrm{inObjThrds}(W,\mathcal{S}) = \mathrm{inObjThrds}(\mathbb{W},\mathbb{S}).$$

Choose T_x such that roundsub(T, activeThrds(W) $-$ Δ, T_x). Next we choose $\mathcal{M}$ to be a function such that the following hold:

$$\mathsf{roundsub}(T,\xi,T_x) \text{ iff } (|T|\neq\omega)\wedge(T_x=T)\ \vee\ \exists T'.\ (T=T_x::T')\wedge(|T_x|\neq\omega)\wedge(\xi\subseteq\mathsf{tidset}(T_x))$$

$$\mathsf{minPos}(T,\xi)\stackrel{\text{def}}{=}\begin{cases}0 & \text{if } \xi=\emptyset\\ 1 & \text{if } T=\epsilon\wedge\xi\neq\emptyset\\ 1 & \text{if } T=\iota::T'\wedge|T|\neq\omega\wedge\mathsf{tid}(\iota)\in\xi\\ 1+\mathsf{minPos}(T',\xi) & \text{if } T=\iota::T'\wedge|T|\neq\omega\wedge\mathsf{tid}(\iota)\notin\xi\wedge\xi\neq\emptyset\end{cases}$$

$$\mathsf{minDis}_\Delta(T,\mathsf{t})\stackrel{\text{def}}{=}\begin{cases}0 & \text{if } \mathsf{t}\notin\Delta\\ 1 & \text{if } T=\epsilon\wedge\mathsf{t}\in\Delta\\ 1 & \text{if } T=\iota::T'\wedge\mathsf{t}\in\mathsf{bset}(\iota)\wedge\mathsf{t}\in\Delta\\ 1+\mathsf{minDis}_\Delta(T',\mathsf{t}) & \text{if } T=\iota::T'\wedge\mathsf{t}\notin\mathsf{bset}(\iota)\wedge\mathsf{t}\in\Delta\end{cases}$$

$$\mathsf{minPos}_\Delta(T,\xi)\stackrel{\text{def}}{=}\begin{cases}\mathsf{minPos}(T,\xi-\Delta) & \text{if } \xi-\Delta\neq\emptyset\\ \mathit{min}\{\mathsf{minDis}_\Delta(T,\mathsf{t}')\mid\mathsf{t}'\in\xi\} & \text{if } \emptyset\subset\xi\subseteq\Delta\\ 0 & \text{if } \xi=\emptyset\end{cases}$$

$$\mathsf{minPos}_{\Delta,\beta}(T,\xi,\mathsf{t})\stackrel{\text{def}}{=}\begin{cases}\mathsf{minPos}(T,\xi-\Delta) & \text{if } \xi-\Delta\neq\emptyset\\ \mathit{min}\{\mathsf{minDis}_\Delta(T,\mathsf{t}')\mid\mathsf{t}'\in\xi\} & \text{if } \emptyset\subset\xi\subseteq\Delta\\ -1 & \text{if } \xi=\emptyset\wedge\beta(\mathsf{t})=\textbf{false}\wedge\mathsf{t}\in\Delta\\ \mathsf{minPos}(T,\{\mathsf{t}\}) & \text{if } \xi=\emptyset\wedge\beta(\mathsf{t})=\textbf{false}\wedge\mathsf{t}\notin\Delta\\ 0 & \text{if } \xi=\emptyset\wedge\beta(\mathsf{t})=\textbf{true}\end{cases}$$

Fig. 26. Definitions related to rounds.

(1) $dom(\mathcal{M})=\mathsf{inObjThrds}(W,\mathcal{S})$.
(2) For any $\mathsf{t}\in dom(\mathcal{M})$, we have:

$$\mathcal{M}(\mathsf{t})=(\mathscr{M}(\mathsf{t}),\ (\zeta(\mathsf{t}),dom(\zeta)\backslash\{\mathsf{t}\}),\ \mathsf{minPos}_{\Delta,\beta}(T_x,\zeta(\mathsf{t}),\mathsf{t}),\\ \alpha(\mathsf{t}),\ (\zeta_a(\mathsf{t}),dom(\zeta_a)\backslash\{\mathsf{t}\}),\ \mathsf{minPos}_\Delta(T_x,\zeta_a(\mathsf{t})))\ .$$

We define the order $\mathcal{M}'(\mathsf{t})<\mathcal{M}(\mathsf{t})$ as a dictionary order:

$$\begin{array}{l}(M',(\xi',\xi'_D),k',M'_a,(\xi'_a,\xi'_{aD}),k'_a)<(M,(\xi,\xi_D),k,M_a,(\xi_a,\xi_{aD}),k_a)\ \text{ iff}\\ \quad(M'<M)\\ \quad\vee(M'=M)\wedge((\xi',\xi'_D)<(\xi,\xi_D))\\ \quad\vee(M'=M)\wedge((\xi',\xi'_D)=(\xi,\xi_D))\wedge(k'<k)\\ \quad\vee(M'=M)\wedge((\xi',\xi'_D)=(\xi,\xi_D))\wedge(k'=k=-1)\wedge(M'_a<M_a)\\ \quad\vee(M'=M)\wedge((\xi',\xi'_D)=(\xi,\xi_D))\wedge(k'=k=-1)\wedge(M'_a=M_a)\wedge((\xi'_a,\xi'_{aD})<(\xi_a,\xi_{aD}))\\ \quad\vee(M'=M)\wedge((\xi',\xi'_D)=(\xi,\xi_D))\wedge(k'=k=-1)\wedge(M'_a=M_a)\wedge((\xi'_a,\xi'_{aD})=(\xi_a,\xi_{aD}))\wedge(k'_a<k_a)\end{array}$$

$$(\xi',\xi'_D)<(\xi,\xi_D)\ \text{ iff}\\ \quad(\xi'\supset\xi)\wedge(\xi'\subseteq\xi'_D)\wedge(\xi\subseteq\xi_D)\wedge(\xi'_D\subseteq\xi_D)$$

$$(\xi',\xi'_D)=(\xi,\xi_D)\ \text{ iff}\\ \quad(\xi'=\xi)\wedge(\xi'\subseteq\xi'_D)\wedge(\xi\subseteq\xi_D)\wedge(\xi'_D\subseteq\xi_D)$$

Clearly that $\mathcal{M}'(\mathsf{t})<\mathcal{M}(\mathsf{t})$ is a well-founded order.

Next we prove: for any T, Δ, W, $\mathcal{S}$, $\mathcal{E}$, $\mathbb{W}$, $\mathbb{S}$, $\mathscr{M}$, ζ, β, α, ζ_a, $\mathcal{M}$ and T_x, if

(1) $T\models_\Delta O^{\text{co}-\omega}_{\text{sfair}}(W,\mathcal{S},\mathcal{E})$;
(2) $\models_{\text{sfair}}(W,\mathcal{S})\preccurlyeq(\mathbb{W},\mathbb{S})\diamond(\mathscr{M},\zeta,\beta,\alpha,\zeta_a)$;
(3) $\mathsf{roundsub}(T,\mathsf{activeThrds}(W)-\Delta,T_x)$;
$dom(\mathcal{M})=\mathsf{inObjThrds}(W,\mathcal{S})$; and
for any $\mathsf{t}\in dom(\mathcal{M})$, we have $\mathcal{M}(\mathsf{t})=(\mathscr{M}(\mathsf{t}),\ (\zeta(\mathsf{t}),dom(\zeta)\backslash\{\mathsf{t}\}),\ \mathsf{minPos}_{\Delta,\beta}(T_x,\zeta(\mathsf{t}),\mathsf{t}),$
$\alpha(\mathsf{t}),\ (\zeta_a(\mathsf{t}),dom(\zeta_a)\backslash\{\mathsf{t}\}),\ \mathsf{minPos}_\Delta(T_x,\zeta_a(\mathsf{t})))$,

then $T\models(W,\mathcal{S})\preceq(\mathbb{W},\mathbb{S})\diamond\mathcal{M}$.

By co-induction. We need to prove: if $(W,\mathcal{S}) \overset{\iota}{\longmapsto} (W',\mathcal{S}')$ and $T = \iota :: T'$, then there exist t, e, Δ_c, Δ_o, T'', n, $\mathbb{W}'$, $\mathbb{S}'$ and $\mathcal{M}'$ such that all the following hold:

(a) $(\mathbb{W},\mathbb{S}) \overset{T''}{\longmapsto}^n (\mathbb{W}',\mathbb{S}')$, and $n = 0 \vee n = 1$;
(b) $\iota = (e,\Delta_c,\Delta_o)$; t = tid($e$);
is_clt(e) $\vee$ is_inv(e) $\vee$ is_ret(e) $\Longrightarrow T'' = (e,\Delta_c,\_) :: \epsilon$;
is_obj(e) $\Longrightarrow (\forall i, \iota' = T''(i).\ \text{is\_obj}(\iota'))$;
(c) $T' \models (W',\mathcal{S}') \preceq (\mathbb{W}',\mathbb{S}') \diamond \mathcal{M}'$;
(d) for any t′ $\in$ inObjThrds($W,\mathcal{S}$), we have:
either t′ $\in$ tidset(T''),
or $\mathcal{M}'$(t′) $<$ $\mathcal{M}$(t′),
or $\mathcal{M}'$(t′) = $\mathcal{M}$(t′) and t′ $\in$ bset($\mathbb{W},\mathbb{S}$) and t′ $\in$ bset($\mathbb{W}',\mathbb{S}'$).

From $\models_{\text{sfair}} (W,\mathcal{S}) \precsim (\mathbb{W},\mathbb{S}) \diamond (\mathscr{M},\zeta,\beta,\alpha,\zeta_a)$, we know there exist t, T'', $\mathbb{W}'$, $\mathbb{S}'$, $\mathscr{M}'$, ζ', β', α', ζ_a', n, e, Δ_c and Δ_o such that all the following hold:

(A) $(\mathbb{W},\mathbb{S}) \overset{T''}{\longmapsto}^n (\mathbb{W}',\mathbb{S}')$, and $n = 0 \vee n = 1$;
(B) $\iota = (e,\Delta_c,\Delta_o)$; t = tid($e$);
is_clt(e) $\vee$ is_inv(e) $\vee$ is_ret(e) $\Longrightarrow T'' = (e,\Delta_c,\_) :: \epsilon$;
is_obj(e) $\Longrightarrow (\forall i, \iota' = T''(i).\ \text{is\_obj}(\iota'))$;
(C) $\models_{\text{sfair}} (W',\mathcal{S}') \precsim (\mathbb{W}',\mathbb{S}') \diamond (\mathscr{M}',\zeta',\beta',\alpha',\zeta_a')$;
(D) if t $\in$ inObjThrds($W,\mathcal{S}$), then
either t $\in$ tidset(T''),
or $\mathscr{M}'$(t) $<$ $\mathscr{M}$(t),
or $\mathscr{M}'$(t) = $\mathscr{M}$(t) and β'(t) = β(t) = **true** and ζ(t) $\subseteq$ ζ'(t) $\subseteq$ (inObjThrds($W,\mathcal{S}$)\{t});
(E) for any t′ $\in$ inObjThrds($W,\mathcal{S}$)\{t}, we have:
either $\mathscr{M}'$(t′) $<$ $\mathscr{M}$(t′),
or $\mathscr{M}'$(t′) = $\mathscr{M}$(t′) and β'(t′) = β(t′) = **false**,
or $\mathscr{M}'$(t′) = $\mathscr{M}$(t′) and β'(t′) = β(t′) = **true** and ζ(t′) $\subseteq$ ζ'(t′) $\subseteq$ (inObjThrds($W,\mathcal{S}$)\{t′}) and t $\notin$ ζ(t′);
(F) for any t′ $\in$ inObjThrds($W,\mathcal{S}$)\{t} and is_await($W|_{\text{t}'}$), we have:
if (t′ $\in$ bset($W,\mathcal{S}$)) $\wedge$ (t′ $\in$ bset($W',\mathcal{S}'$)), then
either α'(t′) $<$ α(t′),
or α'(t′) = α(t′) and ζ_a(t′) $\subseteq$ ζ_a'(t′) $\subseteq$ (inObjThrds($W,\mathcal{S}$)\{t′}) and t $\notin$ ζ_a(t′) and
(ζ_a(t′) $\neq \emptyset$) $\vee$ (ζ_a(t′) $= \emptyset \wedge$ t′ $\notin$ bset($\mathbb{W},\mathbb{S}$)) $\vee$ (ζ_a(t′) $= \emptyset \wedge$ t′ $\in$ bset($\mathbb{W}',\mathbb{S}'$)).

Since $T \models_\Delta O^{\text{co}-\omega}_{\text{sfair}}(W,\mathcal{S},\mathcal{E})$, by Lemmas B.26 and B.27, we know there exists $\mathcal{E}'$ such that $\mathcal{E}$ = (get_obsv(ι)) $:: \mathcal{E}'$ and $T' \models_\Delta O^{\text{co}-\omega}_{\text{sfair}}(W',\mathcal{S}',\mathcal{E}')$.

By (B.18), we know there exists T_x' such that roundsub(T',activeThrds(W') $-\ \Delta, T_x'$).

Choose $\mathcal{M}'$ such that $dom(\mathcal{M}')$ = inObjThrds($W',\mathcal{S}'$); and
for any t $\in$ $dom(\mathcal{M}')$, we have $\mathcal{M}'$(t) = ($\mathscr{M}'$(t), (ζ'(t), $dom(\zeta')$\{t}), minPos$_{\Delta,\beta'}(T_x',\zeta'(\text{t}),\text{t})$, α'(t), (ζ_a'(t), $dom(\zeta_a')$\{t}), minPos$_\Delta(T_x',\zeta_a'(\text{t}))$).

By the co-induction hypothesis, we know $T' \models (W',\mathcal{S}') \preceq (\mathbb{W}',\mathbb{S}') \diamond \mathcal{M}'$.

(1) If t $\in$ inObjThrds($W,\mathcal{S}$), below we prove: either t $\in$ tidset(T''), or $\mathcal{M}'$(t) $<$ $\mathcal{M}$(t), or $\mathcal{M}'$(t) = $\mathcal{M}$(t) and t $\in$ bset($\mathbb{W},\mathbb{S}$) and t $\in$ bset($\mathbb{W}',\mathbb{S}'$). From (D), we know one of the following holds:
- t $\in$ tidset(T''). Thus we are done.
- $\mathscr{M}'$(t) $<$ $\mathscr{M}$(t). Then $\mathcal{M}'$(t) $<$ $\mathcal{M}$(t).
- $\mathscr{M}'$(t) = $\mathscr{M}$(t) and β'(t) = β(t) = **true** and ζ(t) $\subseteq$ ζ'(t) $\subseteq$ (inObjThrds($W,\mathcal{S}$)\{t}).
We know one of the following three cases holds:

- $\zeta(\mathsf{t}) \subset \zeta'(\mathsf{t})$.
  We prove $\mathcal{M}'(\mathsf{t}) < \mathcal{M}(\mathsf{t})$ as follows. From $\models_{\mathsf{sfair}} (W,\mathcal{S}) \preccurlyeq (\mathbb{W},\mathbb{S}) \diamond (\mathscr{M},\zeta,\beta,\alpha,\zeta_a)$ and $\models_{\mathsf{sfair}} (W',\mathcal{S}') \preccurlyeq (\mathbb{W}',\mathbb{S}') \diamond (\mathscr{M}',\zeta',\beta',\alpha',\zeta'_a)$, we know
  $$dom(\zeta') = \mathsf{inObjThrds}(W',\mathcal{S}'), \quad dom(\zeta) = \mathsf{inObjThrds}(W,\mathcal{S}),$$
  $$\zeta'(\mathsf{t}) \subseteq (dom(\zeta')\backslash\{\mathsf{t}\}), \quad \zeta(\mathsf{t}) \subseteq (dom(\zeta)\backslash\{\mathsf{t}\}).$$
  By the operational semantics, we know $\mathsf{inObjThrds}(W',\mathcal{S}') \subseteq \mathsf{inObjThrds}(W,\mathcal{S})$. Thus we have
  $$(\zeta'(\mathsf{t}), dom(\zeta')\backslash\{\mathsf{t}\}) < (\zeta(\mathsf{t}), dom(\zeta)\backslash\{\mathsf{t}\}).$$
  Thus $\mathcal{M}'(\mathsf{t}) < \mathcal{M}(\mathsf{t})$.
- $\emptyset \subset \zeta(\mathsf{t}) = \zeta'(\mathsf{t})$.
  We prove $\mathcal{M}'(\mathsf{t}) < \mathcal{M}(\mathsf{t})$ as follows. First we have $(\zeta'(\mathsf{t}), dom(\zeta')\backslash\{\mathsf{t}\}) = (\zeta(\mathsf{t}), dom(\zeta)\backslash\{\mathsf{t}\})$. Also, since $\models_{\mathsf{sfair}} (W,\mathcal{S}) \preccurlyeq (\mathbb{W},\mathbb{S}) \diamond (\mathscr{M},\zeta,\beta,\alpha,\zeta_a)$, we know for any $\mathsf{t}' \in \zeta(\mathsf{t})$, we have $\mathsf{t}' \notin \mathsf{bset}(W,\mathcal{S})$. Then, from $T \models_\Delta O^{\mathsf{co}-\omega}_{\mathsf{sfair}}(W,\mathcal{S},\mathcal{E})$, we know
  $$\zeta(\mathsf{t}) \cap \Delta = \emptyset.$$
  Thus we know
  $$\mathsf{minPos}_{\Delta,\beta'}(T'_x,\zeta'(\mathsf{t}),\mathsf{t}) = \mathsf{minPos}(T'_x,\zeta(\mathsf{t})) \text{ and}$$
  $$\mathsf{minPos}_{\Delta,\beta}(T_x,\zeta(\mathsf{t}),\mathsf{t}) = \mathsf{minPos}(T_x,\zeta(\mathsf{t})).$$
  Since $\mathsf{roundsub}(T,\mathsf{activeThrds}(W)-\Delta,T_x)$, we know there exists T_z such that $T = T_x :: T_z$. Also $T_x \neq \epsilon$. Since $T = \iota :: T'$, we know there exists T_y such that $T_x = \iota :: T_y$ and $T' = T_y :: T_z$. Also since $\mathsf{roundsub}(T,\mathsf{activeThrds}(W)-\Delta,T_x)$, we know $\mathsf{activeThrds}(W)-\Delta \subseteq \mathsf{tidset}(T_x)$. Thus $\zeta(\mathsf{t}) \subseteq \mathsf{tidset}(T_x)$. Since $\mathsf{t} = \mathsf{tid}(\iota)$ and $\mathsf{t} \notin \zeta(\mathsf{t})$, we know
  $$\zeta(\mathsf{t}) \subseteq \mathsf{tidset}(T_y).$$
  Thus
  $$\mathsf{roundsub}(T',\zeta(\mathsf{t}),T_y).$$
  Then by Lemma B.28, we know
  $$\mathsf{minPos}(T'_x,\zeta(\mathsf{t})) = \mathsf{minPos}(T_y,\zeta(\mathsf{t})) < \mathsf{minPos}(T_x,\zeta(\mathsf{t})).$$
  Thus $\mathcal{M}'(\mathsf{t}) < \mathcal{M}(\mathsf{t})$.
- $\emptyset = \zeta(\mathsf{t}) = \zeta'(\mathsf{t})$.
  We have $(\zeta'(\mathsf{t}), dom(\zeta')\backslash\{\mathsf{t}\}) = (\zeta(\mathsf{t}), dom(\zeta)\backslash\{\mathsf{t}\})$. Since $\beta'(\mathsf{t}) = \beta(\mathsf{t}) = \mathbf{true}$, we know
  $$\mathsf{t} \in \mathsf{bset}(\mathbb{W},\mathbb{S}) \text{ and } \mathsf{t} \in \mathsf{bset}(\mathbb{W}',\mathbb{S}').$$
  Also we know
  $$\mathsf{minPos}_{\Delta,\beta'}(T'_x,\zeta'(\mathsf{t}),\mathsf{t}) = 0 \text{ and}$$
  $$\mathsf{minPos}_{\Delta,\beta}(T_x,\zeta(\mathsf{t}),\mathsf{t}) = 0.$$
  Thus $\mathcal{M}'(\mathsf{t}) = \mathcal{M}(\mathsf{t})$.

(2) For any $\mathsf{t}' \in \mathsf{inObjThrds}(W,\mathcal{S})\backslash\{\mathsf{t}\}$, below we prove: either $\mathcal{M}'(\mathsf{t}') < \mathcal{M}(\mathsf{t}')$, or $\mathcal{M}'(\mathsf{t}') = \mathcal{M}(\mathsf{t}')$ and $\mathsf{t}' \in \mathsf{bset}(\mathbb{W},\mathbb{S})$ and $\mathsf{t}' \in \mathsf{bset}(\mathbb{W}',\mathbb{S}')$. From (E), we know one of the following holds:

- $\mathscr{M}'(\mathsf{t}') < \mathscr{M}(\mathsf{t}')$. Then $\mathcal{M}'(\mathsf{t}') < \mathcal{M}(\mathsf{t}')$.
- $\mathscr{M}'(\mathsf{t}') = \mathscr{M}(\mathsf{t}')$ and $\beta'(\mathsf{t}') = \beta(\mathsf{t}') = \mathbf{false}$.
  Since $\beta'(\mathsf{t}') = \beta(\mathsf{t}') = \mathbf{false}$, we know $\zeta(\mathsf{t}') = \zeta'(\mathsf{t}') = \emptyset$. Thus
  $$(\zeta'(\mathsf{t}'), dom(\zeta')\backslash\{\mathsf{t}'\}) = (\zeta(\mathsf{t}'), dom(\zeta)\backslash\{\mathsf{t}'\}).$$
  One of the following two cases holds:
  - $\mathsf{t}' \notin \Delta$. Thus we know
    $$\mathsf{minPos}_{\Delta,\beta'}(T'_x,\zeta'(\mathsf{t}'),\mathsf{t}') = \mathsf{minPos}(T'_x,\{\mathsf{t}'\}) \text{ and}$$
    $$\mathsf{minPos}_{\Delta,\beta}(T_x,\zeta(\mathsf{t}'),\mathsf{t}') = \mathsf{minPos}(T_x,\{\mathsf{t}'\}).$$
    Since $\mathsf{roundsub}(T,\mathsf{activeThrds}(W)-\Delta,T_x)$, we know there exists T_z such that $T = T_x :: T_z$. Also $T_x \neq \epsilon$. Since $T = \iota :: T'$, we know there exists T_y such that $T_x = \iota :: T_y$ and $T' = T_y :: T_z$.

Also since $\mathsf{roundsub}(T, \mathsf{activeThrds}(W)-\Delta, T_x)$, we know $\mathsf{activeThrds}(W)-\Delta \subseteq \mathsf{tidset}(T_x)$. Thus $\mathsf{t}' \in \mathsf{tidset}(T_x)$. Since $\mathsf{t} = \mathsf{tid}(\iota)$ and $\mathsf{t} \neq \mathsf{t}'$, we know

$$\mathsf{t}' \in \mathsf{tidset}(T_y).$$

Thus

$$\mathsf{roundsub}(T', \{\mathsf{t}'\}, T_y).$$

Then by Lemma B.28, we know

$$\mathsf{minPos}(T'_x, \{\mathsf{t}'\}) = \mathsf{minPos}(T_y, \{\mathsf{t}'\})) < \mathsf{minPos}(T_x, \{\mathsf{t}'\}).$$

Thus $\mathcal{M}'(\mathsf{t}') < \mathcal{M}(\mathsf{t}')$.

- $\mathsf{t}' \in \Delta$. Thus we know

$$\mathsf{minPos}_{\Delta,\beta'}(T'_x, \zeta'(\mathsf{t}'), \mathsf{t}') = -1 \text{ and}$$
$$\mathsf{minPos}_{\Delta,\beta}(T_x, \zeta(\mathsf{t}'), \mathsf{t}') = -1.$$

Since $T \models_\Delta O^{\mathsf{co}-\omega}_{\mathsf{sfair}}(W, \mathcal{S}, \mathcal{E})$ and $\mathsf{t}' \in \Delta$, we know

$$\mathsf{t}' \in \mathsf{bset}(W, \mathcal{S}) \text{ and } \mathsf{t}' \in \mathsf{bset}(W', \mathcal{S}').$$

From (F), we know one of the following holds:

- $\alpha'(\mathsf{t}') < \alpha(\mathsf{t}')$. Then $\mathcal{M}'(\mathsf{t}') < \mathcal{M}(\mathsf{t}')$.
- $\alpha'(\mathsf{t}') = \alpha(\mathsf{t}')$ and $\zeta_a(\mathsf{t}') \subseteq \zeta'_a(\mathsf{t}') \subseteq (\mathsf{inObjThrds}(W, \mathcal{S})\backslash\{\mathsf{t}'\})$ and $\mathsf{t} \notin \zeta_a(\mathsf{t}')$ and $(\zeta_a(\mathsf{t}') \neq \emptyset) \vee (\zeta_a(\mathsf{t}') = \emptyset \wedge \mathsf{t}' \notin \mathsf{bset}(\mathbb{W}, \mathbb{S})) \vee (\zeta_a(\mathsf{t}') = \emptyset \wedge \mathsf{t}' \in \mathsf{bset}(\mathbb{W}', \mathbb{S}'))$.
  We know one of the following three cases holds:
  - $\zeta_a(\mathsf{t}') \subset \zeta'_a(\mathsf{t}')$.
    We prove $\mathcal{M}'(\mathsf{t}') < \mathcal{M}(\mathsf{t}')$ as follows. From $\models_{\mathsf{sfair}} (W, \mathcal{S}) \precsim (\mathbb{W}, \mathbb{S}) \diamond (\mathscr{M}, \zeta, \beta, \alpha, \zeta_a)$ and $\models_{\mathsf{sfair}} (W', \mathcal{S}') \precsim (\mathbb{W}', \mathbb{S}') \diamond (\mathscr{M}', \zeta', \beta', \alpha', \zeta'_a)$, we know
    $$dom(\zeta'_a) = \mathsf{inObjThrds}(W', \mathcal{S}'), \quad dom(\zeta_a) = \mathsf{inObjThrds}(W, \mathcal{S}),$$
    $$\zeta'_a(\mathsf{t}') \subseteq (dom(\zeta'_a)\backslash\{\mathsf{t}'\}), \quad \zeta_a(\mathsf{t}') \subseteq (dom(\zeta_a)\backslash\{\mathsf{t}'\}).$$
    By the operational semantics, we know $\mathsf{inObjThrds}(W', \mathcal{S}') \subseteq \mathsf{inObjThrds}(W, \mathcal{S})$. Thus we have
    $$(\zeta'_a(\mathsf{t}'), dom(\zeta'_a)\backslash\{\mathsf{t}'\}) < (\zeta_a(\mathsf{t}'), dom(\zeta_a)\backslash\{\mathsf{t}'\}).$$
    Thus $\mathcal{M}'(\mathsf{t}') < \mathcal{M}(\mathsf{t}')$.
  - $\emptyset \subset \zeta_a(\mathsf{t}') = \zeta'_a(\mathsf{t}')$.
    We prove $\mathcal{M}'(\mathsf{t}') < \mathcal{M}(\mathsf{t}')$ as follows. First $(\zeta'_a(\mathsf{t}'), dom(\zeta'_a)\backslash\{\mathsf{t}'\}) = (\zeta_a(\mathsf{t}'), dom(\zeta_a)\backslash\{\mathsf{t}'\})$. Also, since $\models_{\mathsf{sfair}} (W, \mathcal{S}) \precsim (\mathbb{W}, \mathbb{S}) \diamond (\mathscr{M}, \zeta, \beta, \alpha, \zeta_a)$, we know for any $\mathsf{t}'' \in \zeta_a(\mathsf{t}')$, we have $\mathsf{t}'' \notin \mathsf{bset}(W, \mathcal{S})$. Then, from $T \models_\Delta O^{\mathsf{co}-\omega}_{\mathsf{sfair}}(W, \mathcal{S}, \mathcal{E})$, we know
    $$\zeta_a(\mathsf{t}') \cap \Delta = \emptyset.$$
    Thus we know
    $$\mathsf{minPos}_\Delta(T'_x, \zeta'_a(\mathsf{t}')) = \mathsf{minPos}(T'_x, \zeta_a(\mathsf{t}')) \text{ and}$$
    $$\mathsf{minPos}_\Delta(T_x, \zeta_a(\mathsf{t}')) = \mathsf{minPos}(T_x, \zeta_a(\mathsf{t}')).$$
    Since $\mathsf{roundsub}(T, \mathsf{activeThrds}(W) - \Delta, T_x)$, we know there exists T_z such that $T = T_x :: T_z$. Also $T_x \neq \epsilon$. Since $T = \iota :: T'$, we know there exists T_y such that $T_x = \iota :: T_y$ and $T' = T_y :: T_z$.
    Also since $\mathsf{roundsub}(T, \mathsf{activeThrds}(W) - \Delta, T_x)$, we know $\mathsf{activeThrds}(W) - \Delta \subseteq \mathsf{tidset}(T_x)$. Thus $\zeta_a(\mathsf{t}') \subseteq \mathsf{tidset}(T_x)$. Since $\mathsf{t} = \mathsf{tid}(\iota)$ and $\mathsf{t} \notin \zeta_a(\mathsf{t}')$, we know
    $$\zeta_a(\mathsf{t}') \subseteq \mathsf{tidset}(T_y).$$
    Thus
    $$\mathsf{roundsub}(T', \zeta_a(\mathsf{t}'), T_y).$$
    Then by Lemma B.28, we know
    $$\mathsf{minPos}(T'_x, \zeta_a(\mathsf{t}')) = \mathsf{minPos}(T_y, \zeta_a(\mathsf{t}')) < \mathsf{minPos}(T_x, \zeta_a(\mathsf{t}')).$$
    Thus $\mathcal{M}'(\mathsf{t}') < \mathcal{M}(\mathsf{t}')$.
  - $\emptyset = \zeta_a(\mathsf{t}') = \zeta'_a(\mathsf{t}')$.

We have $(\zeta'_a(\mathrm{t}'), dom(\zeta'_a)\backslash\{\mathrm{t}'\}) = (\zeta_a(\mathrm{t}'), dom(\zeta_a)\backslash\{\mathrm{t}'\})$. From $\models_{\mathsf{sfair}} (W,\mathcal{S}) \precsim (\mathbb{W},\mathbb{S})\diamond (\mathscr{M},\zeta,\beta,\alpha,\zeta_a)$ and $\models_{\mathsf{sfair}} (W',\mathcal{S}') \precsim (\mathbb{W}',\mathbb{S}')\diamond(\mathscr{M}',\zeta',\beta',\alpha',\zeta'_a)$, since $\mathrm{t}' \in \mathsf{bset}(W,\mathcal{S})$ and $\mathrm{t}' \in \mathsf{bset}(W',\mathcal{S}')$, we know

$$\mathrm{t}' \in \mathsf{bset}(\mathbb{W},\mathbb{S}) \text{ and } \mathrm{t}' \in \mathsf{bset}(\mathbb{W}',\mathbb{S}').$$

Also we know

$$\mathsf{minPos}_{\Delta}(T'_x,\zeta'_a(\mathrm{t}')) = 0 \text{ and}$$
$$\mathsf{minPos}_{\Delta}(T_x,\zeta_a(\mathrm{t}')) = 0.$$

Thus $\mathcal{M}'(\mathrm{t}') = \mathcal{M}(\mathrm{t}')$.

- $\mathscr{M}'(\mathrm{t}') = \mathscr{M}(\mathrm{t}')$ and $\beta'(\mathrm{t}') = \beta(\mathrm{t}') = \textbf{true}$ and $\zeta(\mathrm{t}') \subseteq \zeta'(\mathrm{t}') \subseteq (\mathsf{inObjThrds}(W,\mathcal{S})\backslash\{\mathrm{t}'\})$ and $\mathrm{t} \notin \zeta(\mathrm{t}')$.
  We know one of the following three cases holds:
  - $\zeta(\mathrm{t}') \subset \zeta'(\mathrm{t}')$.
    We prove $\mathcal{M}'(\mathrm{t}') < \mathcal{M}(\mathrm{t}')$ as follows. From $\models_{\mathsf{sfair}} (W,\mathcal{S}) \precsim (\mathbb{W},\mathbb{S}) \diamond (\mathscr{M},\zeta,\beta,\alpha,\zeta_a)$ and $\models_{\mathsf{sfair}} (W',\mathcal{S}') \precsim (\mathbb{W}',\mathbb{S}') \diamond (\mathscr{M}',\zeta',\beta',\alpha',\zeta'_a)$, we know
    $$dom(\zeta') = \mathsf{inObjThrds}(W',\mathcal{S}'), \quad dom(\zeta) = \mathsf{inObjThrds}(W,\mathcal{S}),$$
    $$\zeta'(\mathrm{t}') \subseteq (dom(\zeta')\backslash\{\mathrm{t}'\}), \quad \zeta(\mathrm{t}') \subseteq (dom(\zeta)\backslash\{\mathrm{t}'\}).$$
    By the operational semantics, we know $\mathsf{inObjThrds}(W',\mathcal{S}') \subseteq \mathsf{inObjThrds}(W,\mathcal{S})$. Thus we have
    $$(\zeta'(\mathrm{t}'), dom(\zeta')\backslash\{\mathrm{t}'\}) < (\zeta(\mathrm{t}'), dom(\zeta)\backslash\{\mathrm{t}'\}).$$
    Thus $\mathcal{M}'(\mathrm{t}') < \mathcal{M}(\mathrm{t}')$.
  - $\emptyset \subset \zeta(\mathrm{t}') = \zeta'(\mathrm{t}')$.
    We prove $\mathcal{M}'(\mathrm{t}') < \mathcal{M}(\mathrm{t}')$ as follows. First we have $(\zeta'(\mathrm{t}'), dom(\zeta')\backslash\{\mathrm{t}'\}) = (\zeta(\mathrm{t}'), dom(\zeta)\backslash\{\mathrm{t}'\})$. Also, since $\models_{\mathsf{sfair}} (W,\mathcal{S}) \precsim (\mathbb{W},\mathbb{S}) \diamond (\mathscr{M},\zeta,\beta,\alpha,\zeta_a)$, we know for any $\mathrm{t}'' \in \zeta(\mathrm{t}')$, we have $\mathrm{t}'' \notin \mathsf{bset}(W,\mathcal{S})$. Then, from $T \models_{\Delta} O^{\mathsf{co}-\omega}_{\mathsf{sfair}}(W,\mathcal{S},\mathcal{E})$, we know
    $$\zeta(\mathrm{t}') \cap \Delta = \emptyset.$$
    Thus we know
    $$\mathsf{minPos}_{\Delta,\beta'}(T'_x,\zeta'(\mathrm{t}'),\mathrm{t}') = \mathsf{minPos}(T'_x,\zeta(\mathrm{t}')) \text{ and}$$
    $$\mathsf{minPos}_{\Delta,\beta}(T_x,\zeta(\mathrm{t}'),\mathrm{t}') = \mathsf{minPos}(T_x,\zeta(\mathrm{t}')).$$
    Since $\mathsf{roundsub}(T,\mathsf{activeThrds}(W)-\Delta,T_x)$, we know there exists T_z such that $T = T_x :: T_z$. Also $T_x \neq \epsilon$. Since $T = \iota :: T'$, we know there exists T_y such that $T_x = \iota :: T_y$ and $T' = T_y :: T_z$. Also since $\mathsf{roundsub}(T,\mathsf{activeThrds}(W)-\Delta,T_x)$, we know $\mathsf{activeThrds}(W)-\Delta \subseteq \mathsf{tidset}(T_x)$. Thus $\zeta(\mathrm{t}') \subseteq \mathsf{tidset}(T_x)$. Since $\mathrm{t} = \mathsf{tid}(\iota)$ and $\mathrm{t} \notin \zeta(\mathrm{t}')$, we know
    $$\zeta(\mathrm{t}') \subseteq \mathsf{tidset}(T_y).$$
    Thus
    $$\mathsf{roundsub}(T',\zeta(\mathrm{t}'),T_y).$$
    Then by Lemma B.28, we know
    $$\mathsf{minPos}(T'_x,\zeta(\mathrm{t}')) = \mathsf{minPos}(T_y,\zeta(\mathrm{t}')) < \mathsf{minPos}(T_x,\zeta(\mathrm{t}')).$$
    Thus $\mathcal{M}'(\mathrm{t}') < \mathcal{M}(\mathrm{t}')$.
  - $\emptyset = \zeta(\mathrm{t}') = \zeta'(\mathrm{t}')$.
    We have $(\zeta'(\mathrm{t}'), dom(\zeta')\backslash\{\mathrm{t}'\}) = (\zeta(\mathrm{t}'), dom(\zeta)\backslash\{\mathrm{t}'\})$. Since $\beta'(\mathrm{t}') = \beta(\mathrm{t}') = \textbf{true}$, we know
    $$\mathrm{t}' \in \mathsf{bset}(\mathbb{W},\mathbb{S}) \text{ and } \mathrm{t}' \in \mathsf{bset}(\mathbb{W}',\mathbb{S}').$$
    Also we know
    $$\mathsf{minPos}_{\Delta,\beta'}(T'_x,\zeta'(\mathrm{t}'),\mathrm{t}') = 0 \text{ and}$$
    $$\mathsf{minPos}_{\Delta,\beta}(T_x,\zeta(\mathrm{t}'),\mathrm{t}') = 0.$$
    Thus $\mathcal{M}'(\mathrm{t}') = \mathcal{M}(\mathrm{t}')$.

Thus we are done. □

Lemma B.26. *If* $(\iota :: T) \models_\Delta O^{co-\omega}_\chi(W,\mathcal{S},\mathcal{E}_a)$*, then there exists* $\mathcal{E}_b$ *such that* $\mathcal{E}_a = (\text{get\_obsv}(\iota)) :: \mathcal{E}_b$.

Lemma B.27. *If* $(\iota :: T) \models_\Delta O^{co-\omega}_\chi(W,\mathcal{S},\mathcal{E}_a)$, $(W,\mathcal{S}) \xmapsto{\iota} (W_x,\mathcal{S}_x)$ *and* $\mathcal{E}_a = (\text{get\_obsv}(\iota)) :: \mathcal{E}_b$, *then* $T \models_\Delta O^{co-\omega}_\chi(W,\mathcal{S},\mathcal{E}_b)$.

Proof. By co-induction, and then by inversion three times over $(\iota :: T) \models_\Delta O^{co-\omega}_\chi(W,\mathcal{S},\mathcal{E}_a)$. □

Lemma B.28. *If* $|T| = \omega$, $\xi \neq \emptyset$, $\text{roundsub}(T,\xi,T_x)$ *and* $\text{roundsub}(T,\xi,T_y)$, *then* $\text{minPos}(T_x,\xi) = \text{minPos}(T_y,\xi)$.

Proof. From the premises, we know $|T_x| \neq \omega$ and $|T_y| \neq \omega$. Suppose $|T_x| \leq |T_y|$.
From $\text{roundsub}(T,\xi,T_x)$, we know there exists T'_x such that $T = T_x :: T'_x$ and $\xi \subseteq \text{tidset}(T_x)$.
From $\text{roundsub}(T,\xi,T_y)$, we know there exists T'_y such that $T = T_y :: T'_y$.
Thus there exists T_z such that $T_y = T_x :: T_z$ and $T'_x = T_z :: T'_y$. Since $\xi \neq \emptyset$, we know $\text{tidset}(T_x) \neq \emptyset$. Thus there exist ι and T_0 such that $T_x = \iota :: T_0$. Thus $T_y = \iota :: T_0 :: T_z$. By induction over $|T_0|$. We are done. □

Lemma B.29 (Towards Simulations with Fixed Low-Level Traces (for PSF Objects under Weak Fairness)). *If* $T \models_\Delta O^{co-\omega}_{\text{wfair}}(W,\mathcal{S},\mathcal{E})$ *and* $\models_{\text{wfair}} (W,\mathcal{S}) \precsim (\mathbb{W},\mathbb{S}) \diamond (\mathscr{M},\zeta,\beta,\alpha,\zeta_a)$, *then there exists* $\mathcal{M}$ *such that* $T \models (W,\mathcal{S}) \preceq (\mathbb{W},\mathbb{S}) \diamond \mathcal{M}$.

Proof. We prove the following (B.19) by inversion over $T \models_\Delta O^{co-\omega}_{\text{wfair}}(W,\mathcal{S},\mathcal{E})$.

$$\text{If } T \models_\Delta O^{co-\omega}_{\text{wfair}}(W,\mathcal{S},\mathcal{E}), \text{ then there exists } T_x \text{ such that } \text{roundsub}(T,\text{activeThrds}(W) - \Delta, T_x), \text{activeThrds}(W) - \Delta = \text{tidset}(T_x) \text{ and } (\forall \text{t} \in \Delta.\ \exists i.\ \text{t} \in \text{bset}(T_x(i))). \tag{B.19}$$

Also, by inversion over $\models_{\text{wfair}} (W,\mathcal{S}) \precsim (\mathbb{W},\mathbb{S}) \diamond (\mathscr{M},\zeta,\beta,\alpha,\zeta_a)$, we know

$$dom(\mathscr{M}) = dom(\zeta) = dom(\beta) = dom(\alpha) = dom(\zeta_a) = \text{inObjThrds}(W,\mathcal{S}) = \text{inObjThrds}(\mathbb{W},\mathbb{S}).$$

Choose T_x such that $\text{roundsub}(T,\text{activeThrds}(W) - \Delta, T_x)$. Next we choose $\mathcal{M}$ to be a function such that the following hold:

(1) $dom(\mathcal{M}) = \text{inObjThrds}(W,\mathcal{S})$.
(2) For any $\text{t} \in dom(\mathcal{M})$, we have:

$$\begin{aligned}\mathcal{M}(\text{t}) = (&\mathscr{M}(\text{t}),\ (\zeta(\text{t}), dom(\zeta)\backslash\{\text{t}\}),\ \text{minPos}_{\Delta,\beta}(T_x,\zeta(\text{t}),\text{t}),\\ &\alpha(\text{t}),\ (\zeta_a(\text{t}), dom(\zeta_a)\backslash\{\text{t}\}),\ \text{minPos}_\Delta(T_x,\zeta_a(\text{t})),\\ &(\text{t} \in \text{bset}(\mathbb{W},\mathbb{S}), \text{minDis}_\Delta(T_x,\text{t}))) .\end{aligned}$$

We define the order $\mathcal{M}'(\text{t}) < \mathcal{M}(\text{t})$ as a dictionary order:

$$
\begin{array}{l}
(M',(\xi',\xi'_D),k',M'_a,(\xi'_a,\xi'_{aD}),k'_a,(b',k'_d)) < (M,(\xi,\xi_D),k,M_a,(\xi_a,\xi_{aD}),k_a,(b,k_d)) \ \text{iff} \\
\quad (M' < M) \\
\quad \vee\ (M' = M) \wedge ((\xi',\xi'_D) < (\xi,\xi_D)) \\
\quad \vee\ (M' = M) \wedge ((\xi',\xi'_D) = (\xi,\xi_D)) \wedge (k' < k) \\
\quad \vee\ (M' = M) \wedge ((\xi',\xi'_D) = (\xi,\xi_D)) \wedge (k' = k = -1) \wedge (M'_a < M_a) \\
\quad \vee\ (M' = M) \wedge ((\xi',\xi'_D) = (\xi,\xi_D)) \wedge (k' = k = -1) \wedge (M'_a = M_a) \wedge ((\xi'_a,\xi'_{aD}) < (\xi_a,\xi_{aD})) \\
\quad \vee\ (M' = M) \wedge ((\xi',\xi'_D) = (\xi,\xi_D)) \wedge (k' = k = -1) \wedge (M'_a = M_a) \wedge ((\xi'_a,\xi'_{aD}) = (\xi_a,\xi_{aD})) \\
\qquad \wedge (k'_a < k_a) \\
\quad \vee\ (M' = M) \wedge ((\xi',\xi'_D) = (\xi,\xi_D)) \wedge (k' = k = -1) \wedge (M'_a = M_a) \wedge ((\xi'_a,\xi'_{aD}) = (\xi_a,\xi_{aD})) \\
\qquad \wedge (k'_a = k_a = 0) \wedge (b',k'_d) < (b,k_d)
\end{array}
$$

$$
\begin{array}{l}
(M',(\xi',\xi'_D),k',M'_a,(\xi'_a,\xi'_{aD}),k'_a,(b',k'_d)) = (M,(\xi,\xi_D),k,M_a,(\xi_a,\xi_{aD}),k_a,(b,k_d)) \ \text{iff} \\
\quad (M' = M) \wedge ((\xi',\xi'_D) = (\xi,\xi_D)) \wedge (k' = k \neq -1) \\
\quad \vee\ (M' = M) \wedge ((\xi',\xi'_D) = (\xi,\xi_D)) \wedge (k' = k = -1) \wedge (M'_a = M_a) \wedge ((\xi'_a,\xi'_{aD}) = (\xi_a,\xi_{aD})) \\
\qquad \wedge (k'_a = k_a \neq 0) \\
\quad \vee\ (M' = M) \wedge ((\xi',\xi'_D) = (\xi,\xi_D)) \wedge (k' = k = -1) \wedge (M'_a = M_a) \wedge ((\xi'_a,\xi'_{aD}) = (\xi_a,\xi_{aD})) \\
\qquad \wedge (k'_a = k_a = 0) \wedge (b',k'_d) = (b,k_d)
\end{array}
$$

$$
\begin{array}{l}
(b',k'_d) < (b,k_d) \ \text{iff} \\
\quad (b' = b = \textbf{false} \wedge k' < k) \vee (b' = \textbf{true} \wedge b = \textbf{false})
\end{array}
$$

$$
\begin{array}{l}
(b',k'_d) = (b,k_d) \ \text{iff} \\
\quad (b' = b = \textbf{true}) \vee (b' = b = \textbf{false} \wedge k' = k)
\end{array}
$$

Clearly that $\mathcal{M}'(\mathrm{t}) < \mathcal{M}(\mathrm{t})$ is a well-founded order.

Next we prove: for any T, Δ, W, $\mathcal{S}$, $\mathcal{E}$, $\mathbb{W}$, $\mathbb{S}$, $\mathscr{M}$, ζ, β, α, ζ_a, $\mathcal{M}$ and T_x, if

(1) $T \models_\Delta O^{\mathrm{co}-\omega}_{\mathsf{wfair}}(W,\mathcal{S},\mathcal{E})$;
(2) $\models_{\mathsf{wfair}} (W,\mathcal{S}) \precsim (\mathbb{W},\mathbb{S}) \diamond (\mathscr{M},\zeta,\beta,\alpha,\zeta_a)$;
(3) $\mathsf{roundsub}(T,\mathsf{activeThrds}(W)-\Delta,T_x)$; $\mathsf{activeThrds}(W)-\Delta = \mathsf{tidset}(T_x)$;
$(\forall \mathrm{t} \in \Delta.\ \exists i.\ \mathrm{t} \in \mathsf{bset}(T_x(i)))$;
$dom(\mathcal{M}) = \mathsf{inObjThrds}(W,\mathcal{S})$; and
for any $\mathrm{t} \in dom(\mathcal{M})$, we have $\mathcal{M}(\mathrm{t}) = (\mathscr{M}(\mathrm{t}),\ (\zeta(\mathrm{t}), dom(\zeta)\backslash\{\mathrm{t}\}),\ \mathsf{minPos}_{\Delta,\beta}(T_x,\zeta(\mathrm{t}),\mathrm{t}),$
$\alpha(\mathrm{t}),\ (\zeta_a(\mathrm{t}), dom(\zeta_a)\backslash\{\mathrm{t}\}),\ \mathsf{minPos}_\Delta(T_x,\zeta_a(\mathrm{t})),(\mathrm{t} \in \mathsf{bset}(\mathbb{W},\mathbb{S}),\mathsf{minDis}_\Delta(T_x,\mathrm{t})))$,

then $T \models (W,\mathcal{S}) \precsim (\mathbb{W},\mathbb{S}) \diamond \mathcal{M}$.

By co-induction. We need to prove: if $(W,\mathcal{S}) \overset{\iota}{\longmapsto} (W',\mathcal{S}')$ and $T = \iota :: T'$, then there exist t, e, Δ_c, Δ_o, T'', n, $\mathbb{W}'$, $\mathbb{S}'$ and $\mathcal{M}'$ such that all the following hold:

(a) $(\mathbb{W},\mathbb{S}) \overset{T''}{\longmapsto}{}^{n} (\mathbb{W}',\mathbb{S}')$, and $n = 0 \vee n = 1$;
(b) $\iota = (e,\Delta_c,\Delta_o)$; $\mathrm{t} = \mathsf{tid}(e)$;
$\mathsf{is\_clt}(e) \vee \mathsf{is\_inv}(e) \vee \mathsf{is\_ret}(e) \Longrightarrow T'' = (e,\Delta_c,\_) :: \epsilon$;
$\mathsf{is\_obj}(e) \Longrightarrow (\forall i,\iota' = T''(i).\ \mathsf{is\_obj}(\iota'))$;
(c) $T' \models (W',\mathcal{S}') \precsim (\mathbb{W}',\mathbb{S}') \diamond \mathcal{M}'$;
(d) for any $\mathrm{t}' \in \mathsf{inObjThrds}(W,\mathcal{S})$, we have:
either $\mathrm{t}' \in \mathsf{tidset}(T'')$,
or $\mathcal{M}'(\mathrm{t}') < \mathcal{M}(\mathrm{t}')$,
or $\mathcal{M}'(\mathrm{t}') = \mathcal{M}(\mathrm{t}')$ and $\mathrm{t}' \in \mathsf{bset}(\mathbb{W},\mathbb{S})$ and $\mathrm{t}' \in \mathsf{bset}(\mathbb{W}',\mathbb{S}')$.

From $\models_{\mathsf{wfair}} (W,\mathcal{S}) \precsim (\mathbb{W},\mathbb{S}) \diamond (\mathscr{M},\zeta,\beta,\alpha,\zeta_a)$, we know there exist t, T'', $\mathbb{W}'$, $\mathbb{S}'$, $\mathscr{M}'$, ζ', β', α', ζ'_a, n, e, Δ_c and Δ_o such that all the following hold:

(A) $(\mathbb{W},\mathbb{S}) \overset{T''}{\longmapsto}{}^{n} (\mathbb{W}',\mathbb{S}')$, and $n = 0 \vee n = 1$;

(B) $\iota = (e, \Delta_c, \Delta_o)$; t = tid($e$);
is_clt(e) $\vee$ is_inv(e) $\vee$ is_ret(e) $\Longrightarrow T'' = (e, \Delta_c, \_) :: \epsilon$;
is_obj(e) $\Longrightarrow (\forall i, \iota' = T''(i).$ is_obj(ι'));
(C) $\models_{\text{wfair}} (W', \mathcal{S}') \preccurlyeq (\mathbb{W}', \mathbb{S}') \diamond (\mathscr{M}', \zeta', \beta', \alpha', \zeta'_a)$;
(D) if t $\in$ inObjThrds($W, \mathcal{S}$), then
either t $\in$ tidset(T''),
or $\mathscr{M}'$(t) $<$ $\mathscr{M}$(t),
or $\mathscr{M}'$(t) = $\mathscr{M}$(t) and β'(t) = β(t) = **true** and ζ(t) $\subseteq$ ζ'(t) $\subseteq$ (inObjThrds($W, \mathcal{S}$)\{t});
(E) for any t′ $\in$ inObjThrds($W, \mathcal{S}$)\{t}, we have:
either $\mathscr{M}'$(t′) $<$ $\mathscr{M}$(t′),
or $\mathscr{M}'$(t′) = $\mathscr{M}$(t′) and β'(t′) = β(t′) = **false**,
or $\mathscr{M}'$(t′) = $\mathscr{M}$(t′) and β'(t′) = β(t′) = **true** and ζ(t′) $\subseteq$ ζ'(t′) $\subseteq$ (inObjThrds($W, \mathcal{S}$)\{t′}) and t $\notin$ ζ(t′);
(F) for any t′ $\in$ inObjThrds($W, \mathcal{S}$)\{t} and is_await($W|_{\text{t}'}$), we have:
either α'(t′) $<$ α(t′),
or α'(t′) = α(t′) and ζ_a(t′) $\subseteq$ ζ'_a(t′) $\subseteq$ (inObjThrds($W, \mathcal{S}$)\{t′}) and t $\notin$ ζ_a(t′) and
(ζ_a(t′) $\neq \emptyset$) $\vee$ (ζ_a(t′) = $\emptyset \wedge$ t′ $\notin$ bset($\mathbb{W}, \mathbb{S}$)) $\vee$ (ζ_a(t′) = $\emptyset \wedge$ t′ $\in$ bset($\mathbb{W}', \mathbb{S}'$)).

Since $T \models_\Delta O^{\text{co}-\omega}_{\text{wfair}}(W, \mathcal{S}, \mathcal{E})$, by Lemmas B.26 and B.27, we know there exists $\mathcal{E}'$ such that $\mathcal{E}$ = (get_obsv(ι))$::\mathcal{E}'$ and $T' \models_\Delta O^{\text{co}-\omega}_{\text{wfair}}(W', \mathcal{S}', \mathcal{E}')$.

By (B.19), we know there exists T'_x such that roundsub(T', activeThrds(W') $-\ \Delta, T'_x$), activeThrds(W') $-\ \Delta$ = tidset(T'_x) and ($\forall$t $\in \Delta$. $\exists i$. t $\in$ bset($T'_x(i)$)).

Choose $\mathcal{M}'$ such that $dom(\mathcal{M}')$ = inObjThrds($W', \mathcal{S}'$); and
for any t $\in dom(\mathcal{M}')$, we have $\mathcal{M}'$(t) = ($\mathscr{M}'$(t), (ζ'(t), $dom(\zeta')$\{t}), minPos$_{\Delta,\beta'}(T'_x, \zeta'$(t), t), α'(t), (ζ'_a(t), $dom(\zeta'_a)$\{t}), minPos$_\Delta(T'_x, \zeta'_a$(t)), (t $\in$ bset($\mathbb{W}', \mathbb{S}'$), minDis$_\Delta(T'_x$, t))).

By the co-induction hypothesis, we know $T' \models (W', \mathcal{S}') \preceq (\mathbb{W}', \mathbb{S}') \diamond \mathcal{M}'$.

(1) If t $\in$ inObjThrds($W, \mathcal{S}$), below we prove: either t $\in$ tidset(T''), or $\mathcal{M}'$(t) $<$ $\mathcal{M}$(t), or $\mathcal{M}'$(t) = $\mathcal{M}$(t) and t $\in$ bset($\mathbb{W}, \mathbb{S}$) and t $\in$ bset($\mathbb{W}', \mathbb{S}'$). From (D), we know one of the following holds:
- t $\in$ tidset(T''). Thus we are done.
- $\mathscr{M}'$(t) $<$ $\mathscr{M}$(t). Then $\mathcal{M}'$(t) $<$ $\mathcal{M}$(t).
- $\mathscr{M}'$(t) = $\mathscr{M}$(t) and β'(t) = β(t) = **true** and ζ(t) $\subseteq$ ζ'(t) $\subseteq$ (inObjThrds($W, \mathcal{S}$)\{t}).
We know one of the following three cases holds:
  - ζ(t) $\subset$ ζ'(t).
  We prove $\mathcal{M}'$(t) $<$ $\mathcal{M}$(t) as follows. From $\models_{\text{wfair}} (W, \mathcal{S}) \preccurlyeq (\mathbb{W}, \mathbb{S}) \diamond (\mathscr{M}, \zeta, \beta, \alpha, \zeta_a)$ and $\models_{\text{wfair}} (W', \mathcal{S}') \preccurlyeq (\mathbb{W}', \mathbb{S}') \diamond (\mathscr{M}', \zeta', \beta', \alpha', \zeta'_a)$, we know
  $$dom(\zeta') = \text{inObjThrds}(W', \mathcal{S}'), \quad dom(\zeta) = \text{inObjThrds}(W, \mathcal{S}),$$
  $$\zeta'(\text{t}) \subseteq (dom(\zeta')\backslash\{\text{t}\}), \quad \zeta(\text{t}) \subseteq (dom(\zeta)\backslash\{\text{t}\}).$$
  By the operational semantics, we know inObjThrds($W', \mathcal{S}'$) $\subseteq$ inObjThrds($W, \mathcal{S}$). Thus we have
  $$(\zeta'(\text{t}), dom(\zeta')\backslash\{\text{t}\}) < (\zeta(\text{t}), dom(\zeta)\backslash\{\text{t}\}).$$
  Thus $\mathcal{M}'$(t) $<$ $\mathcal{M}$(t).
  - $\emptyset \subset \zeta(\text{t}) = \zeta'(\text{t})$.
  We prove $\mathcal{M}'$(t) $<$ $\mathcal{M}$(t) as follows. First we have (ζ'(t), $dom(\zeta')$\{t}) = (ζ(t), $dom(\zeta)$\{t}). Since roundsub(T, activeThrds(W) $-\ \Delta, T_x$), we know there exists T_z such that $T = T_x :: T_z$. Also $T_x \neq \epsilon$. Since $T = \iota :: T'$, we know there exists T_y such that $T_x = \iota :: T_y$ and $T' = T_y :: T_z$. Since roundsub(T', activeThrds(W') $-\ \Delta, T'_x$), we know there exists T'_z such that $T' = T'_x :: T'_z$.

We know one of the following holds:

- $\zeta(\mathsf{t}) - \Delta \neq \emptyset$.
  Thus we know
  $$\mathsf{minPos}_{\Delta,\beta'}(T'_x, \zeta'(\mathsf{t}), \mathsf{t}) = \mathsf{minPos}(T'_x, \zeta(\mathsf{t}) - \Delta) \text{ and}$$
  $$\mathsf{minPos}_{\Delta,\beta}(T_x, \zeta(\mathsf{t}), \mathsf{t}) = \mathsf{minPos}(T_x, \zeta(\mathsf{t}) - \Delta).$$
  Also since $\mathsf{roundsub}(T, \mathsf{activeThrds}(W) - \Delta, T_x)$, we know $\mathsf{activeThrds}(W) - \Delta \subseteq \mathsf{tidset}(T_x)$. Thus $\zeta(\mathsf{t}) - \Delta \subseteq \mathsf{tidset}(T_x)$. Since $\mathsf{t} = \mathsf{tid}(\iota)$ and $\mathsf{t} \notin \zeta(\mathsf{t})$, we know
  $$\zeta(\mathsf{t}) - \Delta \subseteq \mathsf{tidset}(T_y).$$
  Thus
  $$\mathsf{roundsub}(T', \zeta(\mathsf{t}) - \Delta, T_y).$$
  Then by Lemma B.28, we know
  $$\mathsf{minPos}(T'_x, \zeta(\mathsf{t}) - \Delta) = \mathsf{minPos}(T_y, \zeta(\mathsf{t}) - \Delta) < \mathsf{minPos}(T_x, \zeta(\mathsf{t}) - \Delta).$$
  Thus $\mathcal{M}'(\mathsf{t}) < \mathcal{M}(\mathsf{t})$.
- $\zeta(\mathsf{t}) \subseteq \Delta$.
  Thus we know
  $$\mathsf{minPos}_{\Delta,\beta'}(T'_x, \zeta'(\mathsf{t}), \mathsf{t}) = \mathit{min}\{\mathsf{minDis}_\Delta(T'_x, \mathsf{t}') \mid \mathsf{t}' \in \zeta(\mathsf{t})\} \text{ and}$$
  $$\mathsf{minPos}_{\Delta,\beta}(T_x, \zeta(\mathsf{t}), \mathsf{t}) = \mathit{min}\{\mathsf{minDis}_\Delta(T_x, \mathsf{t}') \mid \mathsf{t}' \in \zeta(\mathsf{t})\}.$$
  Since $\models_{\mathsf{wfair}} (W, \mathcal{S}) \precsim (\mathbb{W}, \mathbb{S}) \diamond (\mathscr{M}, \zeta, \beta, \alpha, \zeta_a)$ and $\models_{\mathsf{wfair}} (W', \mathcal{S}') \precsim (\mathbb{W}', \mathbb{S}') \diamond (\mathscr{M}', \zeta', \beta', \alpha', \zeta'_a)$, we know for any $\mathsf{t}' \in \zeta(\mathsf{t})$, we have $\mathsf{t}' \notin \mathsf{bset}(W, \mathcal{S})$ and $\mathsf{t}' \notin \mathsf{bset}(W', \mathcal{S}')$. Since $(W, \mathcal{S}) \overset{\iota}{\longmapsto} (W', \mathcal{S}')$, we know
  $$\zeta(\mathsf{t}) \cap \mathsf{bset}(\iota) = \emptyset.$$
  Then, since $T_x = \iota :: T_y$, we know
  $$\mathit{min}\{\mathsf{minDis}_\Delta(T_x, \mathsf{t}') \mid \mathsf{t}' \in \zeta(\mathsf{t})\} = \mathit{min}\{\mathsf{minDis}_\Delta(T_y, \mathsf{t}') \mid \mathsf{t}' \in \zeta(\mathsf{t})\} + 1.$$
  Also since $(\forall \mathsf{t}' \in \Delta.\ \exists i.\ \mathsf{t}' \in \mathsf{bset}(T_x(i)))$, we know
  $$(\forall \mathsf{t}' \in \zeta(\mathsf{t}).\ \exists i.\ \mathsf{t}' \in \mathsf{bset}(T_y(i))).$$
  Then, since $(\forall \mathsf{t}' \in \zeta(\mathsf{t}).\ \exists i.\ \mathsf{t}' \in \mathsf{bset}(T'_x(i)))$, by Lemma B.31, we know
  $$\mathit{min}\{\mathsf{minDis}_\Delta(T'_x, \mathsf{t}') \mid \mathsf{t}' \in \zeta(\mathsf{t})\} = \mathit{min}\{\mathsf{minDis}_\Delta(T_y, \mathsf{t}') \mid \mathsf{t}' \in \zeta(\mathsf{t})\}.$$
  Thus $\mathcal{M}'(\mathsf{t}) < \mathcal{M}(\mathsf{t})$.
- $\emptyset = \zeta(\mathsf{t}) = \zeta'(\mathsf{t})$.
  We have $(\zeta'(\mathsf{t}), \mathit{dom}(\zeta') \backslash \{\mathsf{t}\}) = (\zeta(\mathsf{t}), \mathit{dom}(\zeta) \backslash \{\mathsf{t}\})$. Since $\beta'(\mathsf{t}) = \beta(\mathsf{t}) = \mathbf{true}$, we know
  $$\mathsf{t} \in \mathsf{bset}(\mathbb{W}, \mathbb{S}) \text{ and } \mathsf{t} \in \mathsf{bset}(\mathbb{W}', \mathbb{S}').$$
  Also we know
  $$\mathsf{minPos}_{\Delta,\beta'}(T'_x, \zeta'(\mathsf{t}), \mathsf{t}) = 0 \text{ and}$$
  $$\mathsf{minPos}_{\Delta,\beta}(T_x, \zeta(\mathsf{t}), \mathsf{t}) = 0.$$
  Thus $\mathcal{M}'(\mathsf{t}) = \mathcal{M}(\mathsf{t})$.

(2) For any $\mathsf{t}' \in \mathsf{inObjThrds}(W, \mathcal{S}) \backslash \{\mathsf{t}\}$, below we prove: either $\mathcal{M}'(\mathsf{t}') < \mathcal{M}(\mathsf{t}')$, or $\mathcal{M}'(\mathsf{t}') = \mathcal{M}(\mathsf{t}')$ and $\mathsf{t}' \in \mathsf{bset}(\mathbb{W}, \mathbb{S})$ and $\mathsf{t}' \in \mathsf{bset}(\mathbb{W}', \mathbb{S}')$. From (E), we know one of the following holds:

- $\mathscr{M}'(\mathsf{t}') < \mathscr{M}(\mathsf{t}')$. Then $\mathcal{M}'(\mathsf{t}') < \mathcal{M}(\mathsf{t}')$.
- $\mathscr{M}'(\mathsf{t}') = \mathscr{M}(\mathsf{t}')$ and $\beta'(\mathsf{t}') = \beta(\mathsf{t}') = \mathbf{false}$.
  Since $\beta'(\mathsf{t}') = \beta(\mathsf{t}') = \mathbf{false}$, we know $\zeta(\mathsf{t}') = \zeta'(\mathsf{t}') = \emptyset$. Thus
  $$(\zeta'(\mathsf{t}'), \mathit{dom}(\zeta') \backslash \{\mathsf{t}'\}) = (\zeta(\mathsf{t}'), \mathit{dom}(\zeta) \backslash \{\mathsf{t}'\}).$$
  Since $\mathsf{roundsub}(T, \mathsf{activeThrds}(W) - \Delta, T_x)$, we know there exists T_z such that $T = T_x :: T_z$. Also $T_x \neq \epsilon$. Since $T = \iota :: T'$, we know there exists T_y such that $T_x = \iota :: T_y$ and $T' = T_y :: T_z$. Since $\mathsf{roundsub}(T', \mathsf{activeThrds}(W') - \Delta, T'_x)$, we know there exists T'_z such that $T' = T'_x :: T'_z$. One of the following two cases holds:
  - $\mathsf{t}' \notin \Delta$. Thus we know

$$\mathsf{minPos}_{\Delta,\beta'}(T'_x,\zeta'(\mathsf{t}'),\mathsf{t}') = \mathsf{minPos}(T'_x,\{\mathsf{t}'\}) \text{ and}$$
$$\mathsf{minPos}_{\Delta,\beta}(T_x,\zeta(\mathsf{t}'),\mathsf{t}') = \mathsf{minPos}(T_x,\{\mathsf{t}'\}).$$

Also since $\mathsf{roundsub}(T,\mathsf{activeThrds}(W)-\Delta,T_x)$, we know $\mathsf{activeThrds}(W)-\Delta \subseteq \mathsf{tidset}(T_x)$. Thus $\mathsf{t}' \in \mathsf{tidset}(T_x)$. Since $\mathsf{t} = \mathsf{tid}(\iota)$ and $\mathsf{t} \neq \mathsf{t}'$, we know

$$\mathsf{t}' \in \mathsf{tidset}(T_y).$$

Thus

$$\mathsf{roundsub}(T',\{\mathsf{t}'\},T_y).$$

Then by Lemma B.28, we know

$$\mathsf{minPos}(T'_x,\{\mathsf{t}'\}) = \mathsf{minPos}(T_y,\{\mathsf{t}'\})) < \mathsf{minPos}(T_x,\{\mathsf{t}'\}).$$

Thus $\mathcal{M}'(\mathsf{t}') < \mathcal{M}(\mathsf{t}')$.

- $\mathsf{t}' \in \Delta$. Thus we know

$$\mathsf{minPos}_{\Delta,\beta'}(T'_x,\zeta'(\mathsf{t}'),\mathsf{t}') = -1 \text{ and}$$
$$\mathsf{minPos}_{\Delta,\beta}(T_x,\zeta(\mathsf{t}'),\mathsf{t}') = -1.$$

Since $\mathsf{activeThrds}(W) - \Delta = \mathsf{tidset}(T_x)$ and $(\forall \mathsf{t}' \in \Delta.\ \exists i.\ \mathsf{t}' \in \mathsf{bset}(T_x(i)))$ and $\mathsf{t}' \in \Delta$, we know

$$\mathsf{is\_await}(W|_{\mathsf{t}'}).$$

From (F), we know one of the following holds:

- $\alpha'(\mathsf{t}') < \alpha(\mathsf{t}')$. Then $\mathcal{M}'(\mathsf{t}') < \mathcal{M}(\mathsf{t}')$.
- $\alpha'(\mathsf{t}') = \alpha(\mathsf{t}')$ and $\zeta_a(\mathsf{t}') \subseteq \zeta'_a(\mathsf{t}') \subseteq (\mathsf{inObjThrds}(W,\mathcal{S})\backslash\{\mathsf{t}'\})$ and $\mathsf{t} \notin \zeta_a(\mathsf{t}')$ and $(\zeta_a(\mathsf{t}') \neq \emptyset) \vee (\zeta_a(\mathsf{t}') = \emptyset \wedge \mathsf{t}' \notin \mathsf{bset}(\mathbb{W},\mathbb{S})) \vee (\zeta_a(\mathsf{t}') = \emptyset \wedge \mathsf{t}' \in \mathsf{bset}(\mathbb{W}',\mathbb{S}'))$.
  We know one of the following three cases holds:
  - $\zeta_a(\mathsf{t}') \subset \zeta'_a(\mathsf{t}')$.
    We prove $\mathcal{M}'(\mathsf{t}') < \mathcal{M}(\mathsf{t}')$ as follows. From $\models_{\mathsf{wfair}} (W,\mathcal{S}) \preccurlyeq (\mathbb{W},\mathbb{S}) \diamond (\mathscr{M},\zeta,\beta,\alpha,\zeta_a)$ and $\models_{\mathsf{wfair}} (W',\mathcal{S}') \preccurlyeq (\mathbb{W}',\mathbb{S}') \diamond (\mathscr{M}',\zeta',\beta',\alpha',\zeta'_a)$, we know
    $$dom(\zeta'_a) = \mathsf{inObjThrds}(W',\mathcal{S}'),\quad dom(\zeta_a) = \mathsf{inObjThrds}(W,\mathcal{S}),$$
    $$\zeta'_a(\mathsf{t}') \subseteq (dom(\zeta'_a)\backslash\{\mathsf{t}'\}),\quad \zeta_a(\mathsf{t}') \subseteq (dom(\zeta_a)\backslash\{\mathsf{t}'\}).$$
    By the operational semantics, we know $\mathsf{inObjThrds}(W',\mathcal{S}') \subseteq \mathsf{inObjThrds}(W,\mathcal{S})$. Thus we have
    $$(\zeta'_a(\mathsf{t}'), dom(\zeta'_a)\backslash\{\mathsf{t}'\}) < (\zeta_a(\mathsf{t}'), dom(\zeta_a)\backslash\{\mathsf{t}'\}).$$
    Thus $\mathcal{M}'(\mathsf{t}') < \mathcal{M}(\mathsf{t}')$.
  - $\emptyset \subset \zeta_a(\mathsf{t}') = \zeta'_a(\mathsf{t}')$.
    We prove $\mathcal{M}'(\mathsf{t}') < \mathcal{M}(\mathsf{t}')$ as follows. First $(\zeta'_a(\mathsf{t}'), dom(\zeta'_a)\backslash\{\mathsf{t}'\}) = (\zeta_a(\mathsf{t}'), dom(\zeta_a)\backslash\{\mathsf{t}'\})$.
    We know one of the following holds:
    - $\zeta_a(\mathsf{t}') - \Delta \neq \emptyset$.
      Thus we know
      $$\mathsf{minPos}_{\Delta}(T'_x,\zeta'_a(\mathsf{t}')) = \mathsf{minPos}(T'_x,\zeta_a(\mathsf{t}') - \Delta) \text{ and}$$
      $$\mathsf{minPos}_{\Delta}(T_x,\zeta_a(\mathsf{t}')) = \mathsf{minPos}(T_x,\zeta_a(\mathsf{t}') - \Delta).$$
      Also since $\mathsf{roundsub}(T,\mathsf{activeThrds}(W) - \Delta,T_x)$, we know $\mathsf{activeThrds}(W) - \Delta \subseteq \mathsf{tidset}(T_x)$. Thus $\zeta_a(\mathsf{t}') - \Delta \subseteq \mathsf{tidset}(T_x)$. Since $\mathsf{t} = \mathsf{tid}(\iota)$ and $\mathsf{t} \notin \zeta_a(\mathsf{t}')$, we know
      $$\zeta_a(\mathsf{t}') - \Delta \subseteq \mathsf{tidset}(T_y).$$
      Thus
      $$\mathsf{roundsub}(T',\zeta_a(\mathsf{t}') - \Delta,T_y).$$
      Then by Lemma B.28, we know
      $$\mathsf{minPos}(T'_x,\zeta_a(\mathsf{t}') - \Delta) = \mathsf{minPos}(T_y,\zeta_a(\mathsf{t}') - \Delta) < \mathsf{minPos}(T_x,\zeta_a(\mathsf{t}') - \Delta).$$
      Thus $\mathcal{M}'(\mathsf{t}') < \mathcal{M}(\mathsf{t}')$.
    - $\zeta_a(\mathsf{t}') \subseteq \Delta$.
      Thus we know
      $$\mathsf{minPos}_{\Delta}(T'_x,\zeta'_a(\mathsf{t}')) = min\{\mathsf{minDis}_{\Delta}(T'_x,\mathsf{t}'') \mid \mathsf{t}'' \in \zeta_a(\mathsf{t}')\} \text{ and}$$

$$\mathsf{minPos}_\Delta(T_x, \zeta_a(\mathtt{t}')) = \mathit{min}\{\mathsf{minDis}_\Delta(T_x, \mathtt{t}'') \mid \mathtt{t}'' \in \zeta_a(\mathtt{t}')\}.$$

Since $\models_{\mathsf{wfair}} (W, \mathcal{S}) \preceq (\mathbb{W}, \mathbb{S}) \diamond (\mathscr{M}, \zeta, \beta, \alpha, \zeta_a)$ and $\models_{\mathsf{wfair}} (W', \mathcal{S}') \preceq (\mathbb{W}', \mathbb{S}') \diamond (\mathscr{M}', \zeta', \beta', \alpha', \zeta_a')$, we know for any $\mathtt{t}'' \in \zeta_a(\mathtt{t}')$, we have $\mathtt{t}'' \notin \mathsf{bset}(W, \mathcal{S})$ and $\mathtt{t}'' \notin \mathsf{bset}(W', \mathcal{S}')$. Since $(W, \mathcal{S}) \xmapsto{\iota} (W', \mathcal{S}')$, we know

$$\zeta_a(\mathtt{t}') \cap \mathsf{bset}(\iota) = \emptyset.$$

Then, since $T_x = \iota :: T_y$, we know

$$\mathit{min}\{\mathsf{minDis}_\Delta(T_x, \mathtt{t}'') \mid \mathtt{t}'' \in \zeta_a(\mathtt{t}')\} = \mathit{min}\{\mathsf{minDis}_\Delta(T_y, \mathtt{t}'') \mid \mathtt{t}'' \in \zeta_a(\mathtt{t}')\} + 1.$$

Also since $(\forall \mathtt{t}'' \in \Delta.\ \exists i.\ \mathtt{t}'' \in \mathsf{bset}(T_x(i)))$, we know

$$(\forall \mathtt{t}'' \in \zeta_a(\mathtt{t}').\ \exists i.\ \mathtt{t}'' \in \mathsf{bset}(T_y(i))).$$

Then, since $(\forall \mathtt{t}'' \in \zeta_a(\mathtt{t}').\ \exists i.\ \mathtt{t}'' \in \mathsf{bset}(T_x'(i)))$, by Lemma B.31, we know

$$\mathit{min}\{\mathsf{minDis}_\Delta(T_x', \mathtt{t}'') \mid \mathtt{t}'' \in \zeta_a(\mathtt{t}')\} = \mathit{min}\{\mathsf{minDis}_\Delta(T_y, \mathtt{t}'') \mid \mathtt{t}'' \in \zeta_a(\mathtt{t}')\}.$$

Thus $\mathcal{M}'(\mathtt{t}') < \mathcal{M}(\mathtt{t}')$.

- $\emptyset = \zeta_a(\mathtt{t}') = \zeta_a'(\mathtt{t}')$.

  We have $(\zeta_a'(\mathtt{t}'), \mathit{dom}(\zeta_a') \backslash \{\mathtt{t}'\}) = (\zeta_a(\mathtt{t}'), \mathit{dom}(\zeta_a) \backslash \{\mathtt{t}'\})$. Also we know

  $$\mathsf{minPos}_\Delta(T_x', \zeta_a'(\mathtt{t}')) = 0 \text{ and}$$
  $$\mathsf{minPos}_\Delta(T_x, \zeta_a(\mathtt{t}')) = 0.$$

  One of the following holds:

  - $\mathtt{t}' \in \mathsf{bset}(\mathbb{W}, \mathbb{S})$ and $\mathtt{t}' \in \mathsf{bset}(\mathbb{W}', \mathbb{S}')$. Thus $\mathcal{M}'(\mathtt{t}') = \mathcal{M}(\mathtt{t}')$.
  - $\mathtt{t}' \notin \mathsf{bset}(\mathbb{W}, \mathbb{S})$ and $\mathtt{t}' \in \mathsf{bset}(\mathbb{W}', \mathbb{S}')$. Thus $\mathcal{M}'(\mathtt{t}') < \mathcal{M}(\mathtt{t}')$.
  - $\mathtt{t}' \notin \mathsf{bset}(\mathbb{W}, \mathbb{S})$ and $\mathtt{t}' \notin \mathsf{bset}(\mathbb{W}', \mathbb{S}')$.

    Since $\models_{\mathsf{wfair}} (W, \mathcal{S}) \preceq (\mathbb{W}, \mathbb{S}) \diamond (\mathscr{M}, \zeta, \beta, \alpha, \zeta_a)$ and $\models_{\mathsf{wfair}} (W', \mathcal{S}') \preceq (\mathbb{W}', \mathbb{S}') \diamond (\mathscr{M}', \zeta', \beta', \alpha', \zeta_a')$, we know

    $$\mathtt{t}' \notin \mathsf{bset}(W, \mathcal{S}) \text{ and } \mathtt{t}' \notin \mathsf{bset}(W', \mathcal{S}').$$

    Since $(W, \mathcal{S}) \xmapsto{\iota} (W', \mathcal{S}')$, we know

    $$\mathtt{t}' \notin \mathsf{bset}(\iota).$$

    Then, since $T_x = \iota :: T_y$, we know

    $$\mathsf{minDis}_\Delta(T_x, \mathtt{t}') = \mathsf{minDis}_\Delta(T_y, \mathtt{t}') + 1.$$

    Also since $\mathtt{t}' \in \Delta$ and $(\forall \mathtt{t}'' \in \Delta.\ \exists i.\ \mathtt{t}'' \in \mathsf{bset}(T_x(i)))$, we know

    $$\exists i.\ \mathtt{t}' \in \mathsf{bset}(T_y(i))).$$

    Then, since $\exists i.\ \mathtt{t}' \in \mathsf{bset}(T_x'(i)))$, by Lemma B.30, we know

    $$\mathsf{minDis}_\Delta(T_x', \mathtt{t}') = \mathsf{minDis}_\Delta(T_y, \mathtt{t}').$$

    Thus $\mathcal{M}'(\mathtt{t}') < \mathcal{M}(\mathtt{t}')$.

- $\mathscr{M}'(\mathtt{t}') = \mathscr{M}(\mathtt{t}')$ and $\beta'(\mathtt{t}') = \beta(\mathtt{t}') = \mathbf{true}$ and $\zeta(\mathtt{t}') \subseteq \zeta'(\mathtt{t}') \subseteq (\mathsf{inObjThrds}(W, \mathcal{S}) \backslash \{\mathtt{t}'\})$ and $\mathtt{t} \notin \zeta(\mathtt{t}')$.

  We know one of the following three cases holds:

  - $\zeta(\mathtt{t}') \subset \zeta'(\mathtt{t}')$.

    We prove $\mathcal{M}'(\mathtt{t}') < \mathcal{M}(\mathtt{t}')$ as follows. From $\models_{\mathsf{wfair}} (W, \mathcal{S}) \preceq (\mathbb{W}, \mathbb{S}) \diamond (\mathscr{M}, \zeta, \beta, \alpha, \zeta_a)$ and $\models_{\mathsf{wfair}} (W', \mathcal{S}') \preceq (\mathbb{W}', \mathbb{S}') \diamond (\mathscr{M}', \zeta', \beta', \alpha', \zeta_a')$, we know

    $$\mathit{dom}(\zeta') = \mathsf{inObjThrds}(W', \mathcal{S}'), \quad \mathit{dom}(\zeta) = \mathsf{inObjThrds}(W, \mathcal{S}),$$
    $$\zeta'(\mathtt{t}') \subseteq (\mathit{dom}(\zeta') \backslash \{\mathtt{t}'\}), \quad \zeta(\mathtt{t}') \subseteq (\mathit{dom}(\zeta) \backslash \{\mathtt{t}'\}).$$

    By the operational semantics, we know $\mathsf{inObjThrds}(W', \mathcal{S}') \subseteq \mathsf{inObjThrds}(W, \mathcal{S})$. Thus we have

    $$(\zeta'(\mathtt{t}'), \mathit{dom}(\zeta') \backslash \{\mathtt{t}'\}) < (\zeta(\mathtt{t}'), \mathit{dom}(\zeta) \backslash \{\mathtt{t}'\}).$$

    Thus $\mathcal{M}'(\mathtt{t}') < \mathcal{M}(\mathtt{t}')$.
  - $\emptyset \subset \zeta(\mathtt{t}') = \zeta'(\mathtt{t}')$.

    We prove $\mathcal{M}'(\mathtt{t}') < \mathcal{M}(\mathtt{t}')$ as follows. First we have $(\zeta'(\mathtt{t}'), \mathit{dom}(\zeta') \backslash \{\mathtt{t}'\}) = (\zeta(\mathtt{t}'), \mathit{dom}(\zeta) \backslash \{\mathtt{t}'\})$.

Since $\mathsf{roundsub}(T, \mathsf{activeThrds}(W) - \Delta, T_x)$, we know there exists T_z such that $T = T_x :: T_z$. Also $T_x \neq \epsilon$. Since $T = \iota :: T'$, we know there exists T_y such that $T_x = \iota :: T_y$ and $T' = T_y :: T_z$. Since $\mathsf{roundsub}(T', \mathsf{activeThrds}(W') - \Delta, T'_x)$, we know there exists T'_z such that $T' = T'_x :: T'_z$.

We know one of the following holds:

- $\zeta(\mathsf{t}') - \Delta \neq \emptyset$.
  Thus we know
  $$\mathsf{minPos}_{\Delta,\beta'}(T'_x, \zeta'(\mathsf{t}'), \mathsf{t}') = \mathsf{minPos}(T'_x, \zeta(\mathsf{t}') - \Delta) \text{ and}$$
  $$\mathsf{minPos}_{\Delta,\beta}(T_x, \zeta(\mathsf{t}'), \mathsf{t}') = \mathsf{minPos}(T_x, \zeta(\mathsf{t}') - \Delta).$$
  Also since $\mathsf{roundsub}(T, \mathsf{activeThrds}(W) - \Delta, T_x)$, we know $\mathsf{activeThrds}(W) - \Delta \subseteq \mathsf{tidset}(T_x)$. Thus $\zeta(\mathsf{t}') - \Delta \subseteq \mathsf{tidset}(T_x)$. Since $\mathsf{t} = \mathsf{tid}(\iota)$ and $\mathsf{t} \notin \zeta(\mathsf{t}')$, we know
  $$\zeta(\mathsf{t}') - \Delta \subseteq \mathsf{tidset}(T_y).$$
  Thus
  $$\mathsf{roundsub}(T', \zeta(\mathsf{t}') - \Delta, T_y).$$
  Then by Lemma B.28, we know
  $$\mathsf{minPos}(T'_x, \zeta(\mathsf{t}') - \Delta) = \mathsf{minPos}(T_y, \zeta(\mathsf{t}') - \Delta) < \mathsf{minPos}(T_x, \zeta(\mathsf{t}') - \Delta).$$
  Thus $\mathcal{M}'(\mathsf{t}') < \mathcal{M}(\mathsf{t}')$.
- $\zeta(\mathsf{t}') \subseteq \Delta$.
  Thus we know
  $$\mathsf{minPos}_{\Delta,\beta'}(T'_x, \zeta'(\mathsf{t}'), \mathsf{t}') = \mathit{min}\{\mathsf{minDis}_\Delta(T'_x, \mathsf{t}'') \mid \mathsf{t}'' \in \zeta(\mathsf{t}')\} \text{ and}$$
  $$\mathsf{minPos}_{\Delta,\beta}(T_x, \zeta(\mathsf{t}'), \mathsf{t}') = \mathit{min}\{\mathsf{minDis}_\Delta(T_x, \mathsf{t}'') \mid \mathsf{t}'' \in \zeta(\mathsf{t}')\}.$$
  Since $\models_{\mathsf{wfair}} (W, \mathcal{S}) \preceq (\mathbb{W}, \mathbb{S}) \diamond (\mathscr{M}, \zeta, \beta, \alpha, \zeta_a)$ and $\models_{\mathsf{wfair}} (W', \mathcal{S}') \preceq (\mathbb{W}', \mathbb{S}') \diamond (\mathscr{M}', \zeta', \beta', \alpha', \zeta'_a)$, we know for any $\mathsf{t}'' \in \zeta(\mathsf{t}')$, we have $\mathsf{t}'' \notin \mathsf{bset}(W, \mathcal{S})$ and $\mathsf{t}'' \notin \mathsf{bset}(W', \mathcal{S}')$. Since $(W, \mathcal{S}) \overset{\iota}{\longmapsto} (W', \mathcal{S}')$, we know
  $$\zeta(\mathsf{t}') \cap \mathsf{bset}(\iota) = \emptyset.$$
  Then, since $T_x = \iota :: T_y$, we know
  $$\mathit{min}\{\mathsf{minDis}_\Delta(T_x, \mathsf{t}'') \mid \mathsf{t}'' \in \zeta(\mathsf{t}')\} = \mathit{min}\{\mathsf{minDis}_\Delta(T_y, \mathsf{t}'') \mid \mathsf{t}'' \in \zeta(\mathsf{t}')\} + 1.$$
  Also since $(\forall \mathsf{t}'' \in \Delta.\ \exists i.\ \mathsf{t}'' \in \mathsf{bset}(T_x(i)))$, we know
  $$(\forall \mathsf{t}'' \in \zeta(\mathsf{t}').\ \exists i.\ \mathsf{t}'' \in \mathsf{bset}(T_y(i))).$$
  Then, since $(\forall \mathsf{t}'' \in \zeta(\mathsf{t}').\ \exists i.\ \mathsf{t}'' \in \mathsf{bset}(T'_x(i)))$, by Lemma B.31, we know
  $$\mathit{min}\{\mathsf{minDis}_\Delta(T'_x, \mathsf{t}'') \mid \mathsf{t}'' \in \zeta(\mathsf{t}')\} = \mathit{min}\{\mathsf{minDis}_\Delta(T_y, \mathsf{t}'') \mid \mathsf{t}'' \in \zeta(\mathsf{t}')\}.$$
  Thus $\mathcal{M}'(\mathsf{t}') < \mathcal{M}(\mathsf{t}')$.
- $\emptyset = \zeta(\mathsf{t}') = \zeta'(\mathsf{t}')$.
  We have $(\zeta'(\mathsf{t}'), \mathit{dom}(\zeta') \backslash \{\mathsf{t}'\}) = (\zeta(\mathsf{t}'), \mathit{dom}(\zeta) \backslash \{\mathsf{t}'\})$. Since $\beta'(\mathsf{t}') = \beta(\mathsf{t}') = \mathbf{true}$, we know
  $$\mathsf{t}' \in \mathsf{bset}(\mathbb{W}, \mathbb{S}) \text{ and } \mathsf{t}' \in \mathsf{bset}(\mathbb{W}', \mathbb{S}').$$
  Also we know
  $$\mathsf{minPos}_{\Delta,\beta'}(T'_x, \zeta'(\mathsf{t}'), \mathsf{t}') = 0 \text{ and}$$
  $$\mathsf{minPos}_{\Delta,\beta}(T_x, \zeta(\mathsf{t}'), \mathsf{t}') = 0.$$
  Thus $\mathcal{M}'(\mathsf{t}') = \mathcal{M}(\mathsf{t}')$.

Thus we are done. □

Lemma B.30. *If* $\mathsf{t} \in \Delta$, $(\exists i.\ \mathsf{t} \in \mathsf{bset}(T_x(i)))$, $(\exists i.\ \mathsf{t} \in \mathsf{bset}(T_y(i)))$, $T = T_x :: T'_x$ *and* $T = T_y :: T'_y$, *then* $\mathsf{minDis}_\Delta(T_x, \mathsf{t}) = \mathsf{minDis}_\Delta(T_y, \mathsf{t})$.

Proof. By induction over $|T_x|$. □

Lemma B.31. *If* $\xi \subseteq \Delta$, $\xi \neq \emptyset$, $(\forall \mathsf{t} \in \xi.\ \exists i.\ \mathsf{t} \in \mathsf{bset}(T_x(i)))$, $(\forall \mathsf{t} \in \xi.\ \exists i.\ \mathsf{t} \in \mathsf{bset}(T_y(i)))$, $T = T_x :: T'_x$ *and* $T = T_y :: T'_y$, *then* $\mathit{min}\{\mathsf{minDis}_\Delta(T_x, \mathsf{t}) \mid \mathsf{t} \in \xi\} = \mathit{min}\{\mathsf{minDis}_\Delta(T_y, \mathsf{t}) \mid \mathsf{t} \in \xi\}$.

Proof. By induction over the size of ξ and by applying Lemma B.30. □

Lemma B.32 (Towards Simulations with Fixed Low-Level Traces (for PDF Objects under Strong Fairness)). *If* $T \models_\Delta O^{co-\omega}_{\mathsf{sfair}}(W,\mathcal{S},\mathcal{E})$ *and* $\models_{\mathsf{sfair}} (W,\mathcal{S}) \preccurlyeq (\mathbb{W},\Gamma,\mathscr{B},\mathbb{S}) \diamond (\mathscr{M},\zeta,\beta,\alpha,\zeta_a)$, *then there exists* $\mathcal{M}$ *such that* $T \models\!\!\models (W,\mathcal{S}) \preceq (\mathbb{W},\Gamma,\mathscr{B},\mathbb{S}) \diamond \mathcal{M}$.

Proof. Similar to the proof of Lemma B.25. □

Lemma B.33 (Towards Simulations with Fixed Low-Level Traces (for PDF Objects under Weak Fairness)). *If* $T \models_\Delta O^{co-\omega}_{\mathsf{wfair}}(W,\mathcal{S},\mathcal{E})$ *and* $\models_{\mathsf{wfair}} (W,\mathcal{S}) \preccurlyeq (\mathbb{W},\Gamma,\mathscr{B},\mathbb{S}) \diamond (\mathscr{M},\zeta,\beta,\alpha,\zeta_a)$, *then there exists* $\mathcal{M}$ *such that* $T \models\!\!\models (W,\mathcal{S}) \preceq (\mathbb{W},\Gamma,\mathscr{B},\mathbb{S}) \diamond \mathcal{M}$.

Proof. Choose T_x such that $\mathsf{roundsub}(T, \mathsf{activeThrds}(W) - \Delta, T_x)$. Then we choose $\mathcal{M}$ to be a function such that the following hold:

(1) $dom(\mathcal{M}) = \mathsf{inObjThrds}(W,\mathcal{S})$.
(2) For any $\mathsf{t} \in dom(\mathcal{M})$, we have:

$$\begin{aligned}\mathcal{M}(\mathsf{t}) = (&\mathscr{M}(\mathsf{t}),\ (\zeta(\mathsf{t}), dom(\zeta)\backslash\{\mathsf{t}\}),\ \mathsf{minPos}_{\Delta,\beta}(T_x,\zeta(\mathsf{t}),\mathsf{t}),\\ &\alpha(\mathsf{t}),\ (\zeta_a(\mathsf{t}), dom(\zeta_a)\backslash\{\mathsf{t}\}),\ \mathsf{minPos}_{\Delta}(T_x,\zeta_a(\mathsf{t})),\\ &(\mathsf{t} \in \mathsf{ffset}(\mathscr{B},\mathbb{S}), \mathsf{minDis}_\Delta(T_x,\mathsf{t})))\ .\end{aligned}$$

Similar to the proof of Lemma B.29, we are done. □

B.9 From Simulations to Progress-Aware Contextual Refinements

In this section, we finish the proof of the logic soundness Theorem 7.3.

Proof of Theorem 7.3. From $\mathcal{D},R,G,I \vdash_\chi \{P\}\Pi : \Gamma$, by Lemma B.7, we know

$$\mathcal{D},R,G \models_\chi \{P\}\Pi : \Gamma.$$

Also we know

$$\begin{gathered}dom(\Pi) = dom(\Gamma),\ \forall \mathsf{t},\mathsf{t}'.\ \mathsf{t} \neq \mathsf{t}' \Longrightarrow G_\mathsf{t} \Rightarrow R_{\mathsf{t}'},\\ \mathsf{wffAct}(R,\mathcal{D}),\ P \Rightarrow \neg\mathsf{Enabled}(\mathcal{D}),\ P \vee \mathsf{Enabled}(\mathcal{D}) \Rightarrow I,\ I \triangleright \{R,G\}.\end{gathered}$$

By Lemma B.15, we know

$$\mathcal{D},R,G \models\!\!\models_\chi \{P\}\Pi \preccurlyeq (\Gamma, \mathsf{wr}^*(\Gamma)).$$

Then, by Lemma B.18, we know

(1) Suppose $R \Rightarrow \lfloor R \rfloor_0$ and $G \Rightarrow \lfloor G \rfloor_0$. Then for any C, we have $\mathcal{D},R,G \models_\chi \{P\}(\Pi,C) \preccurlyeq (\Gamma,C)$.
(2) For any C, we have $\mathcal{D},R,G \models\!\!\models_\chi \{P\}(\Pi,C) \preccurlyeq (\mathsf{wr}^*(\Gamma),\Gamma,C)$.

For (1), by Lemma B.21, we know:

$$\text{for any } n, C_1, \ldots, C_n, \text{ we have } \models_\chi \{\textstyle\bigwedge_{\mathsf{t}\in[1..n]} P_\mathsf{t}\}(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n) \preccurlyeq (\mathbf{let}\ \Gamma\ \mathbf{in}\ \mathbb{C}_1 \parallel \ldots \parallel \mathbb{C}_n).$$

For (2), by Lemma B.22, we know

$$\text{for any } n, C_1, \ldots, C_n, \models\!\!\models_\chi \{\textstyle\bigwedge_{\mathsf{t}\in[1..n]} P_\mathsf{t}\}(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n) \preccurlyeq (\mathbf{let}\ \mathsf{wr}^*(\Gamma)\ \mathbf{in}\ \mathbb{C}_1 \parallel \ldots \parallel \mathbb{C}_n, \Gamma).$$

We prove four contextual refinements:

(1) Suppose $R \Rightarrow \lfloor R \rfloor_0$ and $G \Rightarrow \lfloor G \rfloor_0$. Then $\Pi \sqsubseteq^{\mathsf{sfair}}_\varphi \mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PSF}}(\Gamma)$.

Proof. For any $n, C_1, \ldots, C_n, \sigma_c, \sigma$ and Σ, if $\varphi(\sigma) = \Sigma$, we know $(\sigma,\Sigma) \models \bigwedge_{\mathsf{t}\in[1..n]} P_\mathsf{t}$. Let $W = (\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n)$, $\mathbb{W} = (\mathbf{let}\ \mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PSF}}(\Gamma)\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n)$, $\mathcal{S} = (\sigma_c,\sigma,\circledcirc)$ and $\mathbb{S} = (\sigma_c,\Sigma,\circledcirc)$. For any $\mathcal{E}$ and T, if $(\mathcal{E},T) \in O_{\mathsf{sfair}}[\![W,\mathcal{S}]\!]$, we know one of the following holds:

(a) $T \in \mathcal{T}_\omega^{\mathsf{fin}}[\![W, \mathcal{S}]\!]$ and $\mathsf{get\_obsv}(T) = \mathcal{E}$.
Since $\models_{\mathsf{sfair}} \{\bigwedge_{\mathsf{t}\in[1..n]} P_\mathsf{t}\} W \precsim \mathbb{W}$, we know there exist $\mathscr{M}$, ζ, β, α and ζ_a such that
$$\models_{\mathsf{sfair}} (W, \mathcal{S}) \precsim (\mathbb{W}, \mathbb{S}) \diamond (\mathscr{M}, \zeta, \beta, \alpha, \zeta_a).$$
By induction over $|T|$, we know there exists T' such that $T' \in \mathcal{T}_\omega^{\mathsf{fin}}[\![\mathbb{W}, \mathbb{S}]\!]$ and $\mathsf{get\_obsv}(T') = \mathcal{E}$. Thus $(\mathcal{E}, T') \in O_{\mathsf{sfair}}[\![\mathbb{W}, \mathbb{S}]\!]$.

(b) $(\mathcal{E}, T) \in O^{\omega}_{\mathsf{sfair}}[\![W, \mathcal{S}]\!]$.
Let $\Delta = \{\mathsf{t} \mid \text{e-a-disabled}(\mathsf{t}, T) \wedge (|(T|_\mathsf{t})| \neq \omega)\}$. We know there must exist i such that $\forall j \geq i.\ \forall \mathsf{t} \in \Delta.\ \mathsf{t} \in \mathsf{bset}(T(j))$. Thus there exist W_1, $\mathcal{S}_1$, T_0, T_1, $\mathcal{E}_0$ and $\mathcal{E}_1$ such that
$$T_0 = T(1..i),\ T = T_0 :: T_1,\ \mathcal{E} = \mathcal{E}_0 :: \mathcal{E}_1,\ \mathcal{E}_0 = \mathsf{get\_obsv}(T_0),\ \mathcal{E}_1 = \mathsf{get\_obsv}(T_1),$$
$$(W, \mathcal{S}) \xrightarrow{T_0}{}^{*} (W_1, \mathcal{S}_1),\ (W_1, \mathcal{S}_1) \xrightarrow{T_1}{}^{\omega} \cdot,\ \text{sfair-c}(\mathsf{get\_clt}(T_1)).$$
Also we know
$$T_1 \models_\Delta O^{\mathsf{co}-\omega}_{\mathsf{sfair}}(W_1, \mathcal{S}_1, \mathcal{E}_1).$$
Since $\models_{\mathsf{sfair}} \{\bigwedge_{\mathsf{t}\in[1..n]} P_\mathsf{t}\} W \precsim \mathbb{W}$, we know there exist $\mathscr{M}$, ζ, β, α and ζ_a such that
$$\models_{\mathsf{sfair}} (W, \mathcal{S}) \precsim (\mathbb{W}, \mathbb{S}) \diamond (\mathscr{M}, \zeta, \beta, \alpha, \zeta_a).$$
Then we know there exist $\mathbb{W}_1$, $\mathbb{S}_1$, T_0', $\mathscr{M}'$, ζ', β', α' and ζ_a' such that
$$(\mathbb{W}, \mathbb{S}) \xrightarrow{T_0'}{}^{*} (\mathbb{W}_1, \mathbb{S}_1),\ \mathcal{E}_0 = \mathsf{get\_obsv}(T_0'),$$
$$\models_{\mathsf{sfair}} (W_1, \mathcal{S}_1) \precsim (\mathbb{W}_1, \mathbb{S}_1) \diamond (\mathscr{M}', \zeta', \beta', \alpha', \zeta_a').$$
By Lemma B.25, we know there exists $\mathcal{M}$ such that
$$T_1 \models (W_1, \mathcal{S}_1) \preceq (\mathbb{W}_1, \mathbb{S}_1) \diamond \mathcal{M}.$$
By Lemma B.34, we know there exists T_1' such that
$$T_1' \in \mathcal{T}_\omega^{\omega}[\![\mathbb{W}_1, \mathbb{S}_1]\!],\ \text{sfair-o}(\mathsf{get\_obj}(T_1')),\ \mathsf{get\_clt}(T_1') = \mathsf{get\_clt}(T_1).$$
Thus we know $\mathsf{sfair}(T_1')$ and $\mathsf{get\_obsv}(T_1') = \mathcal{E}_1$. Thus $(\mathcal{E}, T_0' :: T_1') \in O^{\omega}_{\mathsf{sfair}}[\![\mathbb{W}, \mathbb{S}]\!]$.
Thus we are done. □

(2) Suppose $R \Rightarrow \lfloor R \rfloor_0$ and $G \Rightarrow \lfloor G \rfloor_0$. Then $\Pi \sqsubseteq^{\mathsf{wfair}}_{\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\varphi)} \mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\Gamma)$.

Proof. For any n, $C_1, \ldots, C_n$, σ_c, σ, Σ and Σ', if $\varphi(\sigma) = \Sigma$ and $\Sigma' = \Sigma \uplus \{\mathtt{listid} \rightsquigarrow \epsilon\}$, we know $(\sigma, \Sigma) \models \bigwedge_{\mathsf{t}\in[1..n]} P_\mathsf{t}$. Let $W = (\textbf{let}\ \Pi\ \textbf{in}\ C_1 \parallel \ldots \parallel C_n)$, $\mathbb{W} = (\textbf{let}\ \Gamma\ \textbf{in}\ C_1 \parallel \ldots \parallel C_n)$, $\mathbb{W}' = (\textbf{let}\ \mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\Gamma)\ \textbf{in}\ C_1 \parallel \ldots \parallel C_n)$, $\mathcal{S} = (\sigma_c, \sigma, \circledcirc)$, $\mathbb{S} = (\sigma_c, \Sigma, \circledcirc)$ and $\mathbb{S}' = (\sigma_c, \Sigma', \circledcirc)$. For any $\mathcal{E}$ and T, if $(\mathcal{E}, T) \in O_{\mathsf{wfair}}[\![W, \mathcal{S}]\!]$, we know one of the following holds:

(a) $T \in \mathcal{T}_\omega^{\mathsf{fin}}[\![W, \mathcal{S}]\!]$ and $\mathsf{get\_obsv}(T) = \mathcal{E}$.
Similar to the proof of (1)(a).

(b) $(\mathcal{E}, T) \in O^{\omega}_{\mathsf{wfair}}[\![W, \mathcal{S}]\!]$.
Let $\Delta = \{\mathsf{t} \mid \text{i-o-disabled}(\mathsf{t}, T) \wedge (|(T|_\mathsf{t})| \neq \omega)\}$. We know there must exist i such that $\forall j \geq i.\ \forall \mathsf{t} \in \Delta.\ \mathsf{tid}(T(j)) \neq \mathsf{t}$. Thus there exist W_1, $\mathcal{S}_1$, T_0, T_1, $\mathcal{E}_0$ and $\mathcal{E}_1$ such that
$$T_0 = T(1..i),\ T = T_0 :: T_1,\ \mathcal{E} = \mathcal{E}_0 :: \mathcal{E}_1,\ \mathcal{E}_0 = \mathsf{get\_obsv}(T_0),\ \mathcal{E}_1 = \mathsf{get\_obsv}(T_1),$$
$$(W, \mathcal{S}) \xrightarrow{T_0}{}^{*} (W_1, \mathcal{S}_1),\ (W_1, \mathcal{S}_1) \xrightarrow{T_1}{}^{\omega} \cdot,\ \text{wfair-c}(\mathsf{get\_clt}(T_1)).$$
Also we know
$$T_1 \models_\Delta O^{\mathsf{co}-\omega}_{\mathsf{wfair}}(W_1, \mathcal{S}_1, \mathcal{E}_1).$$
Since $\models_{\mathsf{wfair}} \{\bigwedge_{\mathsf{t}\in[1..n]} P_\mathsf{t}\} W \precsim \mathbb{W}$, we know there exist $\mathscr{M}$, ζ, β, α and ζ_a such that
$$\models_{\mathsf{wfair}} (W, \mathcal{S}) \precsim (\mathbb{W}, \mathbb{S}) \diamond (\mathscr{M}, \zeta, \beta, \alpha, \zeta_a).$$
Then we know there exist $\mathbb{W}_1$, $\mathbb{S}_1$, T_0', $\mathscr{M}'$, ζ', β', α' and ζ_a' such that
$$(\mathbb{W}, \mathbb{S}) \xrightarrow{T_0'}{}^{*} (\mathbb{W}_1, \mathbb{S}_1),\ \mathsf{get\_clt}(T_0') = \mathsf{get\_clt}(T_0),$$
$$\models_{\mathsf{wfair}} (W_1, \mathcal{S}_1) \precsim (\mathbb{W}_1, \mathbb{S}_1) \diamond (\mathscr{M}', \zeta', \beta', \alpha', \zeta_a').$$
By Lemma B.29, we know there exists $\mathcal{M}$ such that
$$T_1 \models (W_1, \mathcal{S}_1) \preceq (\mathbb{W}_1, \mathbb{S}_1) \diamond \mathcal{M}.$$
By Lemma B.34, we know there exists T_1' such that

$$T_1' \in \mathcal{T}_\omega^\omega[\![\mathbb{W}_1, \mathbb{S}_1]\!],\ \text{sfair-o}(\text{get\_obj}(T_1')),\ \text{get\_clt}(T_1') = \text{get\_clt}(T_1).$$

Thus we know

$$T_0' :: T_1' \in \mathcal{T}_\omega^\omega[\![\mathbb{W}, \mathbb{S}]\!],\ \text{sfair-o}(\text{get\_obj}(T_0' :: T_1')),\ \text{get\_clt}(T_0' :: T_1') = \text{get\_clt}(T).$$

By Decomposition Theorem A.1, we know there exist $\widehat{T}_c$ and $\widehat{T}_o$ such that

$$\widehat{T}_c \in \mathcal{T}_\omega^c[\![C_1 \parallel \ldots \parallel C_n, \sigma_c]\!],\quad \widehat{T}_o \in \mathcal{T}_\omega^o[\![\Gamma, \Sigma]\!],\quad n = \text{tnum}(\widehat{T}_o),$$
$$\widehat{T}_c = \text{get\_clt}(T),\quad \widehat{T}_o = \text{get\_obj}(T).$$

By Lemma B.36, we know there exists $\widehat{T}_a$ such that

$$\widehat{T}_a \in \mathcal{T}_\omega^o[\![\text{wr}_{\text{PSF}}^{\text{wfair}}(\Gamma), \Sigma \uplus \{\texttt{listid} \rightsquigarrow \epsilon\}]\!],\quad \text{tnum}(\widehat{T}_o) = \text{tnum}(\widehat{T}_a),$$
$$\text{get\_hist}(\widehat{T}_o) = \text{get\_hist}(\widehat{T}_a),\quad \text{sfair-o}(\widehat{T}_a).$$

Thus we know $|\widehat{T}_a| \neq \omega \Longrightarrow \forall \mathsf{t} \in [1..\text{tnum}(\widehat{T}_a)].\ \text{term-o}(\widehat{T}_a|_\mathsf{t}) \vee (\mathsf{t} \in \text{bset}(\text{last}(\widehat{T}_a)))$. Thus $\text{fin\_coherent}(\widehat{T}_c, \widehat{T}_a)$. By Composition Theorem A.2, we know there exists T'' such that

$$T'' \in \mathcal{T}_\omega[\![\mathbb{W}', \mathbb{S}']\!],\quad \widehat{T}_c = \text{get\_clt}(T''),\quad \widehat{T}_a = \text{get\_obj}(T'').$$

Thus we know $\text{wfair}(T'')$ and $\text{get\_obsv}(T'') = \mathcal{E}$. Thus $(\mathcal{E}, T'') \in \mathcal{O}_{\text{wfair}}^\omega[\![\mathbb{W}', \mathbb{S}']\!]$.

Thus we are done. □

(3) $\Pi \sqsubseteq_{\text{wr}_{\text{PDF}}^{\text{sfair}}(\varphi)}^{\text{sfair}} \text{wr}_{\text{PDF}}^{\text{sfair}}(\Gamma)$.

Proof. For any $n, C_1, \ldots, C_n, \sigma_c, \sigma, \Sigma$ and Σ', if $\varphi(\sigma) = \Sigma$ and $\Sigma' = \Sigma \uplus \{\text{done} \rightsquigarrow \textbf{false}\}$, we know $(\sigma, \Sigma) \models \bigwedge_{\mathsf{t} \in [1..n]} P_\mathsf{t}$. Let $W = (\textbf{let } \Pi \textbf{ in } C_1 \parallel \ldots \parallel C_n)$, $\mathbb{W} = (\textbf{let } \text{wr}^*(\Gamma) \textbf{ in } C_1 \parallel \ldots \parallel C_n)$, $\mathbb{W}' = (\textbf{let } \text{wr}_{\text{PDF}}^{\text{sfair}}(\Gamma) \textbf{ in } C_1 \parallel \ldots \parallel C_n)$, $\mathcal{S} = (\sigma_c, \sigma, \circledcirc)$ and $\mathbb{S} = (\sigma_c, \Sigma', \circledcirc)$. For any $\mathcal{E}$ and T, if $(\mathcal{E}, T) \in \mathcal{O}_{\text{sfair}}[\![W, \mathcal{S}]\!]$, we know one of the following holds:

(a) $T \in \mathcal{T}_\omega^{\text{fin}}[\![W, \mathcal{S}]\!]$ and $\text{get\_obsv}(T) = \mathcal{E}$.

Similar to the proof of (1)(a).

(b) $(\mathcal{E}, T) \in \mathcal{O}_{\text{sfair}}^\omega[\![W, \mathcal{S}]\!]$.

Let $\Delta = \{\mathsf{t} \mid \text{e-a-disabled}(\mathsf{t}, T) \wedge (|(T|_\mathsf{t})| \neq \omega)\}$. We know there must exist i such that $\forall j \geq i.\ \forall \mathsf{t} \in \Delta.\ \mathsf{t} \in \text{bset}(T(j))$. Thus there exist $W_1, \mathcal{S}_1, T_0, T_1, \mathcal{E}_0$ and $\mathcal{E}_1$ such that

$$T_0 = T(1..i),\ T = T_0 :: T_1,\ \mathcal{E} = \mathcal{E}_0 :: \mathcal{E}_1,\ \mathcal{E}_0 = \text{get\_obsv}(T_0),\ \mathcal{E}_1 = \text{get\_obsv}(T_1),$$
$$(W, \mathcal{S}) \overset{T_0}{\longmapsto}{}^* (W_1, \mathcal{S}_1),\ (W_1, \mathcal{S}_1) \overset{T_1}{\longmapsto}{}^\omega \cdot,\ \text{sfair-c}(\text{get\_clt}(T_1)).$$

Also we know

$$T_1 \models_\Delta \mathcal{O}_{\text{sfair}}^{\text{co}-\omega}(W_1, \mathcal{S}_1, \mathcal{E}_1).$$

Since $\models_{\text{sfair}} \{\bigwedge_{\mathsf{t} \in [1..n]} P_\mathsf{t}\} W \precsim (\mathbb{W}, \Gamma)$, we know there exist $\mathscr{B}, \mathscr{M}, \zeta, \beta, \alpha$ and ζ_a such that

$$\models_{\text{sfair}} (W, \mathcal{S}) \precsim (\mathbb{W}, \Gamma, \mathscr{B}, \mathbb{S}) \diamond (\mathscr{M}, \zeta, \beta, \alpha, \zeta_a).$$

Then we know there exist $\mathbb{W}_1, \mathbb{S}_1, T_0', \mathscr{B}', \mathscr{M}', \zeta', \beta', \alpha'$ and ζ_a' such that

$$(\mathbb{W}, \mathbb{S}) \overset{T_0'}{\longmapsto}{}^* (\mathbb{W}_1, \mathbb{S}_1),\ \mathcal{E}_0 = \text{get\_obsv}(T_0'),$$
$$\models_{\text{sfair}} (W_1, \mathcal{S}_1) \precsim (\mathbb{W}_1, \Gamma, \mathscr{B}', \mathbb{S}_1) \diamond (\mathscr{M}', \zeta', \beta', \alpha', \zeta_a').$$

By Lemma B.32, we know there exists $\mathcal{M}$ such that

$$T_1 \models (W_1, \mathcal{S}_1) \preceq (\mathbb{W}_1, \Gamma, \mathscr{B}', \mathbb{S}_1) \diamond \mathcal{M}.$$

By Lemma B.35, we know there exists T_1' such that

$$T_1' \in \mathcal{T}_\omega^\omega[\![\mathbb{W}_1, \mathbb{S}_1]\!],\ \text{sfair-o}(\text{get\_obj}(T_1')),\ \text{get\_clt}(T_1') = \text{get\_clt}(T_1).$$

Thus we know

$$T_0' :: T_1' \in \mathcal{T}_\omega^\omega[\![\mathbb{W}, \mathbb{S}]\!],\ \text{sfair-o}(\text{get\_obj}(T_0' :: T_1')),\ \text{get\_clt}(T_0' :: T_1') = \text{get\_clt}(T).$$

By Decomposition Theorem A.1, we know there exist $\widehat{T}_c$ and $\widehat{T}_o$ such that

$$\widehat{T}_c \in \mathcal{T}_\omega^c[\![C_1 \parallel \ldots \parallel C_n, \sigma_c]\!],\quad \widehat{T}_o \in \mathcal{T}_\omega^o[\![\text{wr}^*(\Gamma), \Sigma']\!],\quad n = \text{tnum}(\widehat{T}_o),$$
$$\widehat{T}_c = \text{get\_clt}(T),\quad \widehat{T}_o = \text{get\_obj}(T).$$

By Lemma B.37, we know there exists $\widehat{T}_a$ such that

$$\widehat{T}_a \in \mathcal{T}_\omega^o[\![\text{wr}_{\text{PDF}}^{\text{sfair}}(\Gamma), \Sigma']\!],\quad \text{tnum}(\widehat{T}_o) = \text{tnum}(\widehat{T}_a),$$

$$\mathsf{get\_hist}(\widehat{T}_o) = \mathsf{get\_hist}(\widehat{T}_a), \quad \mathsf{sfair\text{-}o}(\widehat{T}_a).$$

Thus we know $|\widehat{T}_a| \neq \omega \Longrightarrow \forall \mathsf{t} \in [1..\mathsf{tnum}(\widehat{T}_a)].\ \mathsf{term\text{-}o}(\widehat{T}_a|_\mathsf{t}) \vee (\mathsf{t} \in \mathsf{bset}(\mathsf{last}(\widehat{T}_a)))$. Thus $\mathsf{fin\_coherent}(\widehat{T}_c, \widehat{T}_a)$. By Composition Theorem A.2, we know there exists T'' such that

$$T'' \in \mathcal{T}_\omega[\![\mathbb{W}', \mathbb{S}]\!], \quad \widehat{T}_c = \mathsf{get\_clt}(T''), \quad \widehat{T}_a = \mathsf{get\_obj}(T'').$$

Thus we know $\mathsf{sfair}(T'')$ and $\mathsf{get\_obsv}(T'') = \mathcal{E}_1$. Thus $(\mathcal{E}, T'') \in O^\omega_{\mathsf{sfair}}[\![\mathbb{W}', \mathbb{S}]\!]$.

Thus we are done. □

(4) $\Pi \sqsubseteq^{\mathsf{wfair}}_{\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PDF}}(\varphi)} \mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PDF}}(\Gamma)$.

Similar to the proof of (3), but by applying Lemma B.38.

Thus we are done. □

Lemma B.34. *If* $T \models (W, \mathcal{S}) \preceq (\mathbb{W}, \mathbb{S}) \diamond \mathcal{M}$ *and* $T \in \mathcal{T}^\omega_\omega[\![W, \mathcal{S}]\!]$, *then there exists* T' *such that* $T' \in \mathcal{T}^\omega_\omega[\![\mathbb{W}, \mathbb{S}]\!]$, $\mathsf{sfair\text{-}o}(\mathsf{get\_obj}(T'))$, $\mathsf{get\_clt}(T') = \mathsf{get\_clt}(T)$.

Lemma B.35. *If* $T \models\!\models (W, \mathcal{S}) \preceq (\mathbb{W}, \Gamma, \mathscr{B}, \mathbb{S}) \diamond \mathcal{M}$ *and* $T \in \mathcal{T}^\omega_\omega[\![W, \mathcal{S}]\!]$, *then there exists* T' *such that* $T' \in \mathcal{T}^\omega_\omega[\![\mathbb{W}, \mathbb{S}]\!]$, $\mathsf{sfair\text{-}o}(\mathsf{get\_obj}(T'))$, $\mathsf{get\_clt}(T') = \mathsf{get\_clt}(T)$.

Lemma B.36. *Suppose every method body in* Γ *is in the form of an* ***await*** *block. If* $\widehat{T}_o \in \mathcal{T}^o_\omega[\![\Gamma, \Sigma]\!]$, $\mathsf{sfair\text{-}o}(\widehat{T}_o)$ *and* $\neg\mathsf{abt}(\widehat{T}_o)$, *then there exists* $\widehat{T}_a$ *such that* $\widehat{T}_a \in \mathcal{T}^o_\omega[\![\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PSF}}(\Gamma), \Sigma \uplus \{\mathtt{listid} \rightsquigarrow \epsilon\}]\!]$, $\mathsf{tnum}(\widehat{T}_o) = \mathsf{tnum}(\widehat{T}_a)$, $\mathsf{get\_hist}(\widehat{T}_o) = \mathsf{get\_hist}(\widehat{T}_a)$ *and* $\mathsf{sfair\text{-}o}(\widehat{T}_a)$.

Proof. By constructing simulations. □

Lemma B.37. *Suppose every method body in* Γ *is in the form of an* ***await*** *block. If* $\widehat{T}_o \in \mathcal{T}^o_\omega[\![\mathsf{wr}^*(\Gamma), \Sigma]\!]$, $\Sigma(\mathsf{done}) = \mathbf{false}$, $\mathsf{sfair\text{-}o}(\widehat{T}_o)$ *and* $\neg\mathsf{abt}(\widehat{T}_o)$, *then there exists* $\widehat{T}_a$ *such that* $\widehat{T}_a \in \mathcal{T}^o_\omega[\![\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\Gamma), \Sigma]\!]$, $\mathsf{tnum}(\widehat{T}_o) = \mathsf{tnum}(\widehat{T}_a)$, $\mathsf{get\_hist}(\widehat{T}_o) = \mathsf{get\_hist}(\widehat{T}_a)$ *and* $\mathsf{sfair\text{-}o}(\widehat{T}_a)$.

Proof. By constructing simulations. □

Lemma B.38. *Suppose every method body in* Γ *is in the form of an* ***await*** *block. If* $\widehat{T}_o \in \mathcal{T}^o_\omega[\![\mathsf{wr}^{\mathsf{sfair}}_{\mathsf{PDF}}(\Gamma), \Sigma]\!]$, $\Sigma(\mathsf{done}) = \mathbf{false}$, $\mathsf{sfair\text{-}o}(\widehat{T}_o)$ *and* $\neg\mathsf{abt}(\widehat{T}_o)$, *then there exists* $\widehat{T}_a$ *such that* $\widehat{T}_a \in \mathcal{T}^o_\omega[\![\mathsf{wr}^{\mathsf{wfair}}_{\mathsf{PDF}}(\Gamma), \Sigma]\!]$, $\mathsf{tnum}(\widehat{T}_o) = \mathsf{tnum}(\widehat{T}_a)$, $\mathsf{get\_hist}(\widehat{T}_o) = \mathsf{get\_hist}(\widehat{T}_a)$ *and* $\mathsf{wfair\text{-}o}(\widehat{T}_a)$.

Proof. By constructing simulations. □

C EXAMPLE PROOFS

In this section, we show the proofs of ticket locks, bounded partial queues with two locks [Herlihy and Shavit 2008] and Treiber stacks [Treiber 1986] with blocking pop. We also prove that the various wrappers defined in Sec. 6 (as object implementations) are contextual refinement of themselves (as abstraction) under the designated fairness condition.

C.1 Ticket locks

Sec. 8.2 shows the key ideas of the proofs of ticket locks. Here we give the formal details.

Fig. 27 defines the invariant I, rely/guarantee conditions R and G, and definite actions G of ticket locks. The definitions are the same as in Fig. 12 in Sec. 8.2, so we omit the explanations.

Fig. 28 shows the proof outlines of the methods `tkL_acq` and `tkL_rel`. The `tkL_rel` method have the annotated precondition `l = cid`. For `tkL_acq`, we verify it using the precondition P', which is just the object invariant $\mathsf{lock}(s, tl, n_1, n_2)$. For `tkL_rel`, we verify it using the precondition P'', saying that a thread should have acquired the lock before calling the method. The key part in the proofs is to verify the loop in the method `tkL_acq` using the WHL rule, which we have already explained in Sec. 8.2.

$$tl ::= \epsilon \mid \mathsf{t} :: tl$$

$$\mathsf{list2set}(\epsilon) \stackrel{\text{def}}{=} \emptyset$$

$$\mathsf{list2set}(\mathsf{t} :: tl) \stackrel{\text{def}}{=} \{\mathsf{t}\} \cup \mathsf{list2set}(tl)$$

$$I \stackrel{\text{def}}{=} \exists s, tl, n_1, n_2.\ \mathsf{lock}(s, tl, n_1, n_2)$$

$$\mathsf{lock}(s, tl, n_1, n_2) \stackrel{\text{def}}{=} (\mathtt{l} = \mathsf{L} = s \wedge (s = \mathsf{head}(tl) \vee s = 0)) \\ * ((\mathtt{owner} = n_1) * (\mathtt{next} = n_2) \wedge (n_1 \leq n_2)) * \mathsf{tickets}(tl, n_1, n_2)$$

$$\mathsf{tickets}(tl, n_1, n_2) \stackrel{\text{def}}{=} \mathsf{tickets\_used}(0, n_1) * \mathsf{tickets\_used}(tl, n_1, n_2) * \mathsf{tickets\_new}(n_2)$$

$$\mathsf{tickets\_used}(n_1, n_2) \stackrel{\text{def}}{=} \exists tl.\ \mathsf{tickets\_used}(tl, n_1, n_2)$$

$$\mathsf{tickets\_used}(tl, n_1, n_2) \stackrel{\text{def}}{=} (tl = \epsilon) \wedge (n_1 = n_2) \wedge \mathsf{emp} \\ \vee \exists \mathsf{t}, tl'.\ (tl = \mathsf{t} :: tl') \wedge (\mathtt{ticket}_{n_1} = \mathsf{t}) * \mathsf{tickets\_used}(tl', n_1 + 1, n_2)$$

$$\mathsf{tickets\_new}(n_2) \stackrel{\text{def}}{=} (\circledast_{i \geq n_2} \mathtt{ticket}_i = -1)$$

$$R_\mathsf{t} \stackrel{\text{def}}{=} \bigvee_{\mathsf{t}' \neq \mathsf{t}} G_{\mathsf{t}'}$$

$$G_\mathsf{t} \stackrel{\text{def}}{=} (\mathit{Req}_\mathsf{t} \vee \mathit{Acq}_\mathsf{t} \vee \mathit{Rel}_\mathsf{t} \vee \mathsf{Id}) * \mathsf{Id} \wedge (I \ltimes I)$$

$$\mathit{Req}_\mathsf{t} \stackrel{\text{def}}{=} \exists s, tl, n_1, n_2.\ \mathsf{lock}(s, tl, n_1, n_2) \ltimes \mathsf{lock}(s, tl{+}{+}[\mathsf{t}], n_1, n_2 + 1)$$

$$\mathit{Acq}_\mathsf{t} \stackrel{\text{def}}{=} \exists tl, n_1, n_2.\ \mathsf{lock}(0, \mathsf{t} :: tl, n_1, n_2) \ltimes \mathsf{lock}(\mathsf{t}, \mathsf{t} :: tl, n_1, n_2)$$

$$\mathit{RelLock}_\mathsf{t} \stackrel{\text{def}}{=} \exists tl, n_1, n_2.\ \mathsf{lock}(\mathsf{t}, \mathsf{t} :: tl, n_1, n_2) \ltimes \mathsf{lock}(0, tl, n_1 + 1, n_2)$$

$$\mathcal{D}_\mathsf{t} \stackrel{\text{def}}{=} \forall tl, n_1, n_2.\ \mathsf{lock}(0, \mathsf{t} :: tl, n_1, n_2) \rightsquigarrow \mathsf{lock}(\mathsf{t}, \mathsf{t} :: tl, n_1, n_2)$$

Fig. 27. Auxiliary definitions for verifying ticket locks.

$P' \stackrel{\text{def}}{=} \exists s, tl, n_1, n_2.\ \mathsf{lock}(s, tl, n_1, n_2)$

$P'' \stackrel{\text{def}}{=} \exists tl, n_1, n_2.\ \mathsf{lock}(\mathsf{t}, \mathsf{t} :: tl, n_1, n_2)$

$\mathsf{tlocked}_{tl_1,\mathsf{t},tl_2}(s, n_1, n, n_2) \stackrel{\text{def}}{=}$
$(\mathsf{l} = \mathsf{L} = s \wedge (s = \mathsf{head}(tl_1) \vee s = 0)) * ((\mathsf{owner} = n_1) * (\mathsf{next} = n_2) \wedge (n_1 \le n < n_2))$
$* \mathsf{tickets\_used}(0, n_1) * \mathsf{tickets\_used}(tl_1, n_1, n) * \mathsf{tickets\_used}(\mathsf{t} :: tl_2, n, n_2) * \mathsf{tickets\_new}(n_2)$

$P_0(n_1, n, n_2) \stackrel{\text{def}}{=} \exists tl_1, tl_2, s.\ \mathsf{tlocked}_{tl_1,\mathsf{t},tl_2}(s, n_1, n, n_2)$

$P_1(n_1, n_2) \stackrel{\text{def}}{=} \exists tl.\ \mathsf{lock}(0, \mathsf{t} :: tl, n_1, n_2)$

$P_3(n_1, n, n_2) \stackrel{\text{def}}{=} \exists \mathsf{t}', tl_1, tl_2, s.\ \mathsf{tlocked}_{\mathsf{t}' :: tl_1,\mathsf{t},tl_2}(s, n_1, n, n_2)$

$J \stackrel{\text{def}}{=} \exists n_1, n_2.\ P_0(n_1, n, n_2) \qquad Q \stackrel{\text{def}}{=} \mathsf{Enabled}(\mathcal{D})$

$f(\mathfrak{S}) \stackrel{\text{def}}{=} \begin{cases} 2k+1 & \text{if } \mathfrak{S} \models (n - \mathsf{owner} = k) * (\mathsf{l} = 0) \\ 2k & \text{if } \mathfrak{S} \models (n - \mathsf{owner} = k) * (\mathsf{l} \neq 0) \end{cases}$

```
tkL_acq():
 1  local i, o;
    {P' ∧ arem(L_ACQ')}
 2  < i := getAndInc(&next);  ticket_i := cid; >
    {∃n1,n,n2. P0(n1,n,n2) ∧ (i = n) ∧ arem(L_ACQ')}
    {∃n1,n2. P0(n1,n,n2) ∧ (i = n) ∧ arem(L_ACQ')}
 3  o := owner;
    {∃n1,n2. P0(n1,n,n2) ∧ (i = n) ∧ (o ≤ n1) ∧ arem(L_ACQ')}
    {∃n1,n2. P0(n1,n,n2) ∧ (i = n) ∧ (o ≤ n1) ∧ arem(L_ACQ') ∧ ◊(n − o)}
 4  while (i != o) {
      {∃n1,n2. (P1(n,n2) ∧ ◊(n − (o + 1)) ∨ P3(n1,n,n2) ∧ ◊(n − o))
       ∧ (o ≤ n1 ≤ n = i) ∧ (o ≠ i) ∧ arem(L_ACQ')}
 5    o := owner;
      {∃n1,n2. P0(n1,n,n2) ∧ (i = n) ∧ (o ≤ n1) ∧ arem(L_ACQ') ∧ ◊(n − o)}
 6  }
    {∃n1,n2. P1(n1,n2) ∧ arem(L_ACQ')}
 7   <l := cid>;
    {P'' ∧ arem(skip)}

tkL_rel():
    {P'' ∧ arem(L_REL)}
 8  < owner := owner + 1;  l := 0; >
    {P' ∧ arem(skip)}
```

Fig. 28. Proof outline of ticket locks.

C.2 Bounded partial queues

```
initialize(){  Size := 0; Tail := Head := cons(0, null); Hlock := Tlock := 0;
               e := 0; d := 0;  }

enq(v){
 1  local x;
 2  x := cons(v, null);
 3  await (Tlock = 0) { Tlock := cid; }
 4  await (Size < MAX) {}
 5  <Tail.next := x; e := 1 >;
 6  Tail := x;
 7  <Size := Size + 1; e := 0 >;
 8  Tlock := 0;
}

deq(){
 9  local h, x, v;
10  await (Hlock = 0) { Hlock := cid; }
11  await (Size > 0) {}
12  h := Head;
13  x := h.next;
14  v := x.data;
15  <Head := x; d := 1 >;
16  <Size := Size - 1; d := 0 >;
17  Hlock := 0;
18  dispose(h);
19  return v;
}
```

Fig. 29. Bounded partial queues (with auxiliary code in gray).

In this section, we verify the bounded partial queue object [Herlihy and Shavit 2008]. At the abstract side, the object data is a shared variable Q, whose value is a mathematical list representing the queue. The number of items in the queue is limited. The ENQ method should be blocked if the queue is full, and the DEQ method should be blocked if the queue is empty. We define the atomic partial specifications as follows:

```
ENQ(V) {  await(|Q| < MAX){ Q := Q ++ [V] };  }
DEQ() {  local V; await(|Q| > 0){ V := head(Q); Q := tail(Q) }; return V; }
```

Here we use |Q| to get the length of the list. head(Q) returns the first item of the list, and tail(Q) removes the first item and returns the remaining list.

Fig. 29 shows the concrete implementation code. The queue is implemented as a singly-linked list with the Head and Tail pointers and a sentinel node pointed to by Head. The enq method inserts a new node at the tail of the queue. The deq method replaces the sentinel node by its next node and returns the value in the new sentinel. The Tail and Head pointers are protected by two locks Tlock and Hlock respectively. The locks ensure that at any time at most one enq thread and one deq thread can access the queue, but an enq thread and a deq thread do not need to wait for each other.

Note that the lock-acquire and lock-release code (i.e., lines 3, 8, 10 and 17) is implemented using **await** instructions, as in the lock's atomic partial specifications `L_ACQ'` and `L_REL` (defined in (2.3)). They are abstractions for the ticket locks under strongly fair scheduling, and abstractions for test-and-set locks under weakly fair scheduling (see Sec. 2). Below we will verify that the queue object in Fig. 29 is PDF under weakly fair scheduling. Thus if we use the concrete lock implementations (either ticket locks or test-and-set-locks) to replace the **await** code here (i.e., lines 3, 8, 10 and 17), the resulting queue object is still PDF under weakly fair scheduling.

The queue implementation also uses the shared variable `Size` to record the number of nodes (except the sentinel) in the queue, and to block the methods at appropriate situations. Note that the updates of `Size` and the queue's linked list are not simultaneous. So, in order to help verification, we introduce two auxiliary write-only variables `e` and `d` (highlighted in gray in Fig. 29), to indicate whether the item list has been updated but `Size` has not.

Below we will verify that the queue object in Fig. 29 is PDF under *weakly fair* scheduling. Since strong fairness is stronger than weak fairness, we will also know that it is PDF under *strongly fair* scheduling.

Fig. 30 defines the invariant fence I, the object invariant P (served as the pre- and post-conditions of the object methods), the rely/guarantee conditions and the definite actions. We have lock-acquire actions *LockT* and *LockH*, and lock-release actions *UnlockT* and *UnlockH*. The actions *Enq* and *Deq* add and remove nodes respectively. *Swing* moves the `tail` pointer, and *ESz* and *DSz* update `Size`. Here all the actions are defined as level-1 actions, i.e., completing any action is a step towards progress of the method.

The definite actions say that if the thread acquires the lock `Tlock` and `Size < MAX`, it will eventually release the lock; if the thread acquires the lock `Hlock` and `Size > 0`, it will eventually release the lock; and if the thread has set `e` (or `d`), it will eventually resets `e` (or `d`).

Fig. 31 and 32 show the proof outlines for `enq` and `deq` respectively. Below we only discuss the verification of the **await** statements (lines 3 and 4) of `enq`. The **await** statements in `deq` are verified in similar ways.

For line 3 in `enq`, we apply the AWAIT-W rule and discharge the premises (we use p to denote the pre-condition for the **await** block):

- $p \wedge \mathsf{Enabled}(\mathcal{D}) * \mathsf{true} \Rightarrow$ (`Tlock = 0`) holds, because $p \wedge \mathsf{Enabled}(\mathcal{D}) * \mathsf{true}$ is false.
- $p \Rightarrow (R\colon \mathcal{D} \circ\!\xrightarrow{f} ((\texttt{Tlock = 0}), (\texttt{|Q| < MAX})))$ holds, if the metric f is a constant function. In detail, if p_t for thread t holds, and if (`Tlock ≠ 0`) and (`|Q| < MAX`) hold, we know some other thread t′ must acquire `Tlock`. Also either `Size < MAX` holds, or another thread t″ acquires `Hlock` and `d = 1` holds. Thus $\exists t' \neq t.\ \mathsf{Enabled}(\mathcal{D}_{t'})$ holds.

For line 4 in `enq`, we apply the AWAIT-W rule and discharge the premises (we use p_1 to denote the pre-condition for this **await** block):

- It is easy to see that $p_1 \wedge \mathsf{Enabled}(\mathcal{D}) * \mathsf{true} \Rightarrow$ (`Size < MAX`) holds.
- $p_1 \Rightarrow (R\colon \mathcal{D} \circ\!\xrightarrow{f} ((\texttt{Size < MAX}), (\texttt{|Q| < MAX})))$ holds, if the metric f is a constant function. In detail, if p_1 for thread t holds, and if (`|Q| < MAX`) holds but (`Size < MAX`) does not hold, we know some other thread t′ must acquire `Hlock` and `d = 1` holds. Thus $\exists t' \neq t.\ \mathsf{Enabled}(\mathcal{D}_{t'})$ holds.

$$I \stackrel{\text{def}}{=} \exists h,z,s,s',m,e,d.\ (\mathtt{Head}=h) * (\mathtt{Tail}=z) * \mathsf{queue}_e(m,h,z) * \mathsf{lock}_s(\mathtt{Tlock}) * \mathsf{lock}_{s'}(\mathtt{Hlock}) * \mathsf{sz}_{s,s'}(m,e,d)$$

$$\mathsf{sz}_{s,s'}(m,e,d) \stackrel{\text{def}}{=} \exists m'.\ (\mathtt{e}=e) * (\mathtt{d}=d) * (\mathtt{Size}=m') \wedge 0 \le m' \le \mathtt{MAX} \wedge (s=0 \Rightarrow e=0) \wedge (s'=0 \Rightarrow d=0) \wedge (e=0 \vee e=1) \wedge (d=0 \vee d=1) \wedge (m=m'+e-d)$$

$$\mathsf{queue}_e(m,h,z) \stackrel{\text{def}}{=} \exists v_d, A.\ (\mathtt{Q}=A \wedge |A|=m) * (\mathsf{unlag}(h,z,v_d::A) \vee (\mathsf{lag}(h,z,\_,v_d::A) \wedge (e=1)) \vee (\mathsf{cross}(h,z,v_d::A) \wedge (e=1)))$$

$$\mathsf{unlag}(h,z,A) \stackrel{\text{def}}{=} \exists v, A'.\ (A=A'::v) \wedge \mathsf{ls}(h,A',z) * \mathsf{N}(z,v,\mathtt{null})$$

$$\mathsf{lag}(h,z,x,A) \stackrel{\text{def}}{=} \exists v,v',A'.\ (A=A'::v::v') \wedge \mathsf{ls}(h,A',z) * \mathsf{N2}(z,v,x,v',\mathtt{null})$$

$$\mathsf{cross}(h,z,A) \stackrel{\text{def}}{=} \exists v.\ (A=v::\epsilon) \wedge \mathsf{N}(h,v,\mathtt{null}) \wedge (h \neq z)$$

$$\mathsf{ls}(x,A,y) \stackrel{\text{def}}{=} (x=y \wedge A=\epsilon) \vee (x \neq y \wedge \exists z,v,A'.\ A=v::A' \wedge \mathsf{N}(x,v,z) * \mathsf{ls}(z,A',y))$$

$$\mathsf{N}(p,v,y) \stackrel{\text{def}}{=} (p.\mathtt{data}=v) * (p.\mathtt{next}=y) \qquad \mathsf{N2}(p,v,y,v',z) \stackrel{\text{def}}{=} \mathsf{N}(p,v,y) * \mathsf{N}(y,v',z)$$

$$\mathsf{lock}_s(l) \stackrel{\text{def}}{=} (l=s) \qquad \mathsf{lockIrr}_{\mathsf{t},s}(l) \stackrel{\text{def}}{=} \mathsf{lock}_s(l) \wedge (s \neq \mathsf{t})$$

$$P_\mathsf{t} \stackrel{\text{def}}{=} \exists h,z,s,s',m,e,d.\ (\mathtt{Head}=h) * (\mathtt{Tail}=z) * \mathsf{queue}_e(m,h,z) * \mathsf{lockIrr}_{\mathsf{t},s}(\mathtt{Tlock}) * \mathsf{lockIrr}_{\mathsf{t},s'}(\mathtt{Hlock}) * \mathsf{sz}_{s,s'}(m,e,d)$$

$$R_\mathsf{t} \stackrel{\text{def}}{=} \bigvee_{\mathsf{t}' \neq \mathsf{t}} G_{\mathsf{t}'}$$

$$G_\mathsf{t} \stackrel{\text{def}}{=} (\mathit{Enq}_\mathsf{t} \vee \mathit{Swing}_\mathsf{t} \vee \mathit{ESz}_\mathsf{t} \vee \mathit{LockT}_\mathsf{t} \vee \mathit{UnlockT}_\mathsf{t} \vee \mathit{Deq}_\mathsf{t} \vee \mathit{DSz}_\mathsf{t} \vee \mathit{LockH}_\mathsf{t} \vee \mathit{UnlockH}_\mathsf{t} \vee \mathsf{Id}) * \mathsf{Id} \wedge (I \ltimes I)$$

$$\mathit{Enq}_\mathsf{t} \stackrel{\text{def}}{=} \exists x,y,A,v,v',s'.\ [\mathsf{lock}_\mathsf{t}(\mathtt{Tlock}) * (\mathtt{Tail}=x) \wedge |A| < \mathtt{MAX}] * (\mathsf{N}(x,v,\mathtt{null}) * (\mathtt{e}=0) * (\mathtt{Q}=A) \ltimes_1 \mathsf{N2}(x,v,y,v',\mathtt{null}) * (\mathtt{e}=1) * (\mathtt{Q}=A::v'))$$

$$\mathit{Swing}_\mathsf{t} \stackrel{\text{def}}{=} \exists x,v.\ [\mathsf{lock}_\mathsf{t}(\mathtt{Tlock}) * \mathsf{N}(x,v,\mathtt{null})] * ((\mathtt{Tail}=\_) \ltimes_1 (\mathtt{Tail}=x))$$

$$\mathit{ESz}_\mathsf{t} \stackrel{\text{def}}{=} \exists h,z,v_d,A,m'.\ [\mathsf{lock}_\mathsf{t}(\mathtt{Tlock}) * (\mathtt{Head}=h) * (\mathtt{Tail}=z) * \mathsf{unlag}(h,z,v_d::A)] * ((\mathtt{Size}=m') * (\mathtt{e}=1) \ltimes_1 (\mathtt{Size}=m'+1) * (\mathtt{e}=0))$$

$$\mathit{LockT}_\mathsf{t} \stackrel{\text{def}}{=} \exists s'.\ [\mathsf{lockIrr}_{\mathsf{t},s'}(\mathtt{Hlock})] * (\mathsf{lock}_0(\mathtt{Tlock}) \ltimes_1 \mathsf{lock}_\mathsf{t}(\mathtt{Tlock}))$$

$$\mathit{UnlockT}_\mathsf{t} \stackrel{\text{def}}{=} [(\mathtt{e}=0)] * (\mathsf{lock}_\mathsf{t}(\mathtt{Tlock}) \ltimes_1 \mathsf{lock}_0(\mathtt{Tlock}))$$

$$\mathit{Deq}_\mathsf{t} \stackrel{\text{def}}{=} \exists x,y,z,v,v',A.\ [\mathsf{lock}_\mathsf{t}(\mathtt{Hlock})] * ((\mathtt{Head}=x) * \mathsf{N2}(x,v,y,v',z) * (\mathtt{d}=0) * (\mathtt{Q}=v'::A) \ltimes_1 (\mathtt{Head}=y) * \mathsf{N}(y,v',z) * (\mathtt{d}=1) * (\mathtt{Q}=A))$$

$$\mathit{DSz}_\mathsf{t} \stackrel{\text{def}}{=} \exists m'.\ [\mathsf{lock}_\mathsf{t}(\mathtt{Hlock})] * ((\mathtt{Size}=m') * (\mathtt{d}=1) \ltimes_1 (\mathtt{Size}=m'-1) * (\mathtt{d}=0))$$

$$\mathit{LockH}_\mathsf{t} \stackrel{\text{def}}{=} \exists s.\ [\mathsf{lockIrr}_{\mathsf{t},s}(\mathtt{Tlock})] * (\mathsf{lock}_0(\mathtt{Hlock}) \ltimes_1 \mathsf{lock}_\mathsf{t}(\mathtt{Hlock}))$$

$$\mathit{UnlockH}_\mathsf{t} \stackrel{\text{def}}{=} [(\mathtt{d}=0)] * (\mathsf{lock}_\mathsf{t}(\mathtt{Hlock}) \ltimes_1 \mathsf{lock}_0(\mathtt{Hlock}))$$

$$\mathcal{D}_\mathsf{t} \stackrel{\text{def}}{=} (pT_\mathsf{t} \rightsquigarrow qT_\mathsf{t}) \wedge (pH_\mathsf{t} \rightsquigarrow qH_\mathsf{t}) \wedge (pE_\mathsf{t} \rightsquigarrow qE_\mathsf{t}) \wedge (pD_\mathsf{t} \rightsquigarrow qD_\mathsf{t})$$

$$pT_\mathsf{t} \stackrel{\text{def}}{=} \mathsf{lock}_\mathsf{t}(\mathtt{Tlock}) * (\mathtt{Size} < \mathtt{MAX}) * \mathsf{true} \wedge I \qquad qT_\mathsf{t} \stackrel{\text{def}}{=} \mathsf{lock}_0(\mathtt{Tlock}) * \mathsf{true} \wedge I$$

$$pH_\mathsf{t} \stackrel{\text{def}}{=} \mathsf{lock}_\mathsf{t}(\mathtt{Hlock}) * (\mathtt{Size} > 0) * \mathsf{true} \wedge I \qquad qH_\mathsf{t} \stackrel{\text{def}}{=} \mathsf{lock}_0(\mathtt{Hlock}) * \mathsf{true} \wedge I$$

$$pE_\mathsf{t} \stackrel{\text{def}}{=} \mathsf{lock}_\mathsf{t}(\mathtt{Tlock}) * (\mathtt{e}=1) * \mathsf{true} \wedge I \qquad qE_\mathsf{t} \stackrel{\text{def}}{=} \mathsf{lock}_\mathsf{t}(\mathtt{Tlock}) * (\mathtt{e}=0) * \mathsf{true} \wedge I$$

$$pD_\mathsf{t} \stackrel{\text{def}}{=} \mathsf{lock}_\mathsf{t}(\mathtt{Hlock}) * (\mathtt{d}=1) * \mathsf{true} \wedge I \qquad qD_\mathsf{t} \stackrel{\text{def}}{=} \mathsf{lock}_\mathsf{t}(\mathtt{Hlock}) * (\mathtt{d}=0) * \mathsf{true} \wedge I$$

Fig. 30. Auxiliary definitions for verifying bounded partial queues.

$P_1 \stackrel{\text{def}}{=} \exists A.\ P_1(A,0)$

$P_1(A,e) \stackrel{\text{def}}{=} \exists h,z,v_d,s',d.\ (\mathsf{Head} = h) * (\mathtt{Tail} = z) * (\mathtt{Q} = A)$
$\quad * \mathsf{unlag}(h,z,v_d :: A) * \mathsf{lock_t}(\mathtt{Tlock}) * \mathsf{lockIrr_{t,s'}}(\mathtt{Hlock}) * \mathsf{sz_{1,s'}}(|A|,e,d)$

$P_2 \stackrel{\text{def}}{=} \exists A.\ P_1(A,0) \wedge (|A| < \mathtt{MAX})$

$P_3(x) \stackrel{\text{def}}{=} \exists h,z,v_d,A,s',d.\ (\mathsf{Head} = h) * (\mathtt{Tail} = z) * (\mathtt{Q} = A) * \mathsf{sz_{1,s'}}(|A|,1,d)$
$\quad * (\mathsf{lag}(h,z,x,v_d :: A) \vee (\mathsf{cross}(h,z,v_d :: A) \wedge (h = x))) * \mathsf{lock_t}(\mathtt{Tlock}) * \mathsf{lockIrr_{t,s'}}(\mathtt{Hlock})$

$P_4 \stackrel{\text{def}}{=} \exists A.\ P_1(A,1)$

```
enq(v) {
 1  local x;
    {P ∧ arem(ENQ) ∧ (v = V) ∧ ♦(5)}
 2  x := cons(v, null);
    {P * N(x,v,null) ∧ arem(ENQ) ∧ (v = V) ∧ ♦(5)}
 3  await (Tlock = 0) { Tlock := cid; }
    {P1 * N(x,v,null) ∧ arem(ENQ) ∧ (v = V) ∧ ♦(4)}
 4  await (Size < MAX) {}
    {P2 * N(x,v,null) ∧ arem(ENQ) ∧ (v = V) ∧ ♦(4)}
 5  <Tail.next := x;  e := 1 >;
    {P3(x) ∧ arem(skip) ∧ ♦(3)}
 6  Tail := x;
    {P4 ∧ arem(skip) ∧ ♦(2)}
 7  <Size := Size + 1;  e := 0 >;
    {P1 ∧ arem(skip) ∧ ♦(1)}
 8  Tlock := 0;
    {P ∧ arem(skip)}
}
```

Fig. 31. Proof outline for enq.

$P'_1 \stackrel{\text{def}}{=} \exists h, m.\ P'_1(h, m, 0)$

$P'_1(h, m, d) \stackrel{\text{def}}{=} \exists z, s, e.\ (\mathsf{Head} = h) * (\mathtt{Tail} = z) * \mathsf{queue}_e(m, h, z) * \mathsf{lockIrr}_{\mathsf{t},s}(\mathtt{Tlock}) * \mathsf{lock}_{\mathsf{t}}(\mathtt{Hlock}) * \mathsf{sz}_{s,1}(m, e, d)$

$P'_2(h) \stackrel{\text{def}}{=} \exists m.\ P'_1(h, m, 0) \wedge (m > 0)$

$P'_3(h, x) \stackrel{\text{def}}{=} \exists z, A, s, m, e.\ (\mathsf{Head} = h) * (\mathtt{Tail} = z) * (\mathtt{Q} = A \wedge |A| = m > 0) * \mathsf{N}(h, \_, x)$
$* (\mathsf{unlag}(x, z, A) \vee \mathsf{lag}(x, z, \_, A) \wedge (e = 1)) * \mathsf{lockIrr}_{\mathsf{t},s}(\mathtt{Tlock}) * \mathsf{lock}_{\mathsf{t}}(\mathtt{Hlock}) * \mathsf{sz}_{s,1}(m, e, 0)$

$P'_4(h, x, v) \stackrel{\text{def}}{=} \exists s.\ (\mathsf{Head} = h) * (\mathtt{Tail} = h) * (\mathtt{Q} = v :: \epsilon) * \mathsf{N2}(h, \_, x, v, \mathtt{null})$
$* \mathsf{lockIrr}_{\mathsf{t},s}(\mathtt{Tlock}) * \mathsf{lock}_{\mathsf{t}}(\mathtt{Hlock}) * \mathsf{sz}_{s,1}(1, 1, 0)$

$P'_5(h, x, v) \stackrel{\text{def}}{=} \exists y, z, v, v', A, s, e.\ (\mathsf{Head} = h) * (\mathtt{Tail} = z) * (\mathtt{Q} = v :: A) * \mathsf{N2}(h, \_, x, v, y)$
$* ((x = z) \wedge (y = \mathtt{null}) \wedge (A = \epsilon) \vee \mathsf{unlag}(y, z, A) \vee (x = z) \wedge \mathsf{N}(y, v', \mathtt{null}) \wedge (A = v' :: \epsilon) \wedge (e = 1)$
$\vee\ \mathsf{lag}(y, z, \_, A) \wedge (e = 1)) * \mathsf{lockIrr}_{\mathsf{t},s}(\mathtt{Tlock}) * \mathsf{lock}_{\mathsf{t}}(\mathtt{Hlock}) * \mathsf{sz}_{s,1}(|A| + 1, e, 0)$

$P'_6(h) \stackrel{\text{def}}{=} \exists m.\ P'_1(h, m, 1)$

```
int deq() {
 9  local h, x, v;
    {P ∧ arem(DEQ) ∧ ♦(4)}
10  await (Hlock = 0) { Hlock := cid; }
    {P'1 ∧ arem(DEQ) ∧ ♦(3)}
11  await (Size > 0) {}
    {∃h. P'2(h) ∧ arem(DEQ) ∧ ♦(3)}
12  h := Head;
    {P'2(h) ∧ arem(DEQ) ∧ ♦(3)}
13  x := h.next;
    {(P'3(h,x) ∨ P'4(h,s,_)) ∧ arem(DEQ) ∧ ♦(3)}
14  v := x.data;
    {(P'5(h,x,v) ∨ P'4(h,x,v)) ∧ arem(DEQ) ∧ ♦(3)}
15  <Head := x;  d := 1 >;
    {P'6(x) * N(h,_,x) ∧ arem(skip) ∧ (v = V) ∧ ♦(2)}
16  <Size := Size - 1;  d := 0 >;
    {P'1 * N(h,_,x) ∧ arem(skip) ∧ (v = V) ∧ ♦(1)}
17  Hlock := 0;
    {P * N(h,_,x) ∧ arem(skip) ∧ (v = V)}
18  dispose(h);
    {P ∧ arem(skip) ∧ (v = V)}
19  return v;
}
```

Fig. 32. Proof outline for deq.

$$I \stackrel{\text{def}}{=} \exists A.\ \mathsf{stack}(A) \qquad \mathsf{stack}(A) \stackrel{\text{def}}{=} \exists x.\ \mathsf{stack}(x,A)$$

$$\mathsf{stack}(x,A) \stackrel{\text{def}}{=} (\mathtt{Stk} = A) * (\mathtt{Top} = x) * \mathsf{ls}(x,A,\mathtt{null}) * \mathsf{garb}$$

$$\mathsf{ls}(x,A,y) \stackrel{\text{def}}{=} (x = y \wedge A = \epsilon \wedge \mathit{emp}) \vee (x \neq y \wedge \exists z,v,A'.\ A = v{::}A' \wedge \mathsf{node}(x,v,z) * \mathsf{ls}(z,A',y))$$

$$\mathsf{garb} \stackrel{\text{def}}{=} \exists S_g.\ (\mathtt{GN} = S_g) * (\circledast_{x \in S_g} . \mathsf{node}(x))$$

$$\mathsf{node}(x,v,y) \stackrel{\text{def}}{=} x \mapsto (v,y) \qquad \mathsf{node}(x) \stackrel{\text{def}}{=} \mathsf{node}(x,\_,\_)$$

$$R = G \stackrel{\text{def}}{=} (\mathit{Push} \vee \mathit{Pop} \vee \mathsf{Id}) * \mathsf{Id} \wedge (I \ltimes I)$$

$$\mathit{Push} \stackrel{\text{def}}{=} \exists x,y,v,A.\ (\mathtt{Stk} = A) * (\mathtt{Top} = y) \ltimes_1 (\mathtt{Stk} = v{::}A) * (\mathtt{Top} = x) * \mathsf{node}(x,v,y)$$

$$\mathit{Pop} \stackrel{\text{def}}{=} \exists x,y,v,A,S_g.\ (\mathtt{Stk} = v{::}A) * (\mathtt{Top} = x) * \mathsf{node}(x,v,y) * (\mathtt{GN} = S_g) \ltimes_1 (\mathtt{Stk} = A) * (\mathtt{Top} = y) * \mathsf{node}(x,v,y) * (\mathtt{GN} = S_g \cup \{x\})$$

$$\mathcal{D} \stackrel{\text{def}}{=} \mathsf{false} \rightsquigarrow \mathsf{true}$$

Fig. 33. Auxiliary definitions for verifying Treiber stacks with partial pops.

C.3 Treiber stacks with partial pops

We have given the code of Treiber stacks with partial pops at the top of Fig. 8. At the abstract side, the object data is a shared variable Stk, whose value is a mathematical list representing the stack. The PUSH method is total, and the POP method is blocked if the stack is empty. We define the atomic partial specifications as follows:

```
PUSH(V) {  Stk := V :: Stk;  }
POP() {  local V; await(|Stk| > 0){ V:=head(Stk); Stk:=tail(Stk) }; return V; }
```

Below we verify that the stack object in Fig. 8 satisfies PDF under *weakly fair* scheduling. Since strong fairness is stronger than weak fairness, we will also know that it is PDF under *strongly fair* scheduling.

We define the precise invariant I, the rely/guarantee conditions R and G, and the definite actions $\mathcal{D}$ in Figure 33. The invariant I in Figure 33 maps the value sequence A of the concrete list pointed to by Top (denoted by $(\mathtt{Top} = x) * \mathsf{ls}(x,A,\mathtt{null})$) to the abstract stack Stk. To ensure there is no "ABA" problem [Herlihy and Shavit 2008], we follow Turon and Wand [Turon and Wand 2011] and introduce a write-only auxiliary variable GN to remember the nodes which used to be on the stack but no longer are. The precise invariant for shared states should include those garbage nodes (garb). GN does not affect the behaviors of the implementation and is introduced for verification only.

The guarantee condition G includes the *Push* and *Pop* actions. At the concrete side, the steps at line 8 for push and line 12 for pop in Figure 8 are the linearization points, i.e., they correspond to executions of the abstract atomic PUSH and POP operations. Note that when popping a node, we also add the node to GN. Both the *Push* and *Pop* actions are defined as level-1 actions. They may delay the progress of other threads. The definite actions $\mathcal{D}$ could be defined as false $\rightsquigarrow$ true.

We show the proof outline in Figure 34. Below we only explain the verification of the **while**-loops.

To verify the loop in the push method, we apply the WHL rule and prove:

$$I \Rightarrow (R,G\colon \mathcal{D} \xrightarrow{f_1} (Q_1, \mathsf{true})).$$

Here we define Q_1 as true, and f_1 as a constant function.

To verify the loop in the pop method, we apply the WHL rule and prove:

```
push(v){
 1  local x, b, t;
    {I ∧ arem(PUSH) ∧ v = V ∧ ♦}
 2  b := false;
 3  x := cons(v, null);
    {(¬b ∧ I ∗ node(x,v,_) ∧ arem(PUSH) ∧ (v = V) ∧ ♦ ∧ ◊)  ∨  (b ∧ I ∧ arem(skip))}
 4  while (!b) {
      {¬b ∧ I ∗ node(x,v,_) ∧ arem(PUSH) ∧ (v = V) ∧ ♦}
 5    t := Top;
 6    x.next := t;
      {∃a. ¬b ∧ stack(a,_) ∗ node(x,v,t) ∧ (t = a ∨ t ≠ a ∧ ◊) ∧ arem(PUSH) ∧ (v = V) ∧ ♦}
 7    b := cas(&Top, t, x);
      {(b ∧ I ∧ arem(skip))  ∨  (¬b ∧ I ∗ node(x,v,_) ∧ arem(PUSH) ∧ (v = V) ∧ ♦ ∧ ◊)}
 8  }
    {I ∧ arem(skip)}
}

IntSet GN;     //Auxiliary global variable for verification: popped garbage nodes

pop(){
 9  local x, b, t, v;
    {I ∧ arem(POP) ∧ ♦}
10  b := false;
    {(¬b ∧ I ∧ arem(POP) ∧ ♦ ∧ ◊)  ∨  (b ∧ I ∧ arem(skip) ∧ (v = V))}
11  while (!b) {
      {∃A. ¬b ∧ stack(A) ∧ (A = ϵ ∧ ◊ ∨ A ≠ ϵ) ∧ arem(POP) ∧ ♦}
12    t := Top;
      {((t = null) ∧ ¬b ∧ I ∧ arem(POP) ∧ ♦ ∧ ◊)
       ∨ (∃a. node(t) ∗ true ∧ ¬b ∧ stack(a,_) ∧ (t = a ∨ t ≠ a ∧ ◊) ∧ arem(POP) ∧ ♦)}
13    if (t != null) {
        {∃a. node(t) ∗ true ∧ stack(a,_) ∧ (t = a ∨ t ≠ a ∧ ◊) ∧ arem(POP) ∧ ♦}
14      v := t.data;
15      x := t.next;
        {∃a. node(t,v,x) ∗ true ∧ stack(a,_) ∧ (t = a ∨ t ≠ a ∧ ◊) ∧ arem(POP) ∧ ♦}
16      < b := cas(&Top, t, x); if (b) { GN := GN ∪ t; } >;
        {(b ∧ I ∧ arem(skip) ∧ (v = V))  ∨  (¬b ∧ I ∧ arem(POP) ∧ ♦ ∧ ◊)}
17    }
18  }
    {I ∧ arem(skip) ∧ (v = V)}
19  return v;
}
```

Fig. 34. Proof outline of Treiber stacks with partial pops.

$$I \Rightarrow (R, G\colon \mathcal{D} \xrightarrow{f_2} (Q_2, (|\mathsf{Stk}| > 0))).$$

Here we define f_2 as a constant function, and Q_2 as follows:

$$Q_2 \stackrel{\text{def}}{=} \exists A.\ \mathsf{stack}(A) \wedge (A \neq \epsilon).$$

C.4 Our wrappers

Below we use locks as examples and show that the various wrappers defined in Sec. 6 (as object implementations) are contextual refinement of themselves (as abstraction) under the certain fairness condition.

The atomic partial specifications for locks are `L_ACQ'` and `L_REL` defined in (2.3).

```
initialize(){  l := 0; }

lock(){
1  await (l = 0) {
2    l := cid;
3  }
}

unlock(){
4  l := 0;
}
```

Fig. 35. Simple PSF locks under strong fairness.

C.4.1 Simple PSF locks under strong fairness. We first verify that the code shown in Fig. 35 (which is the same as (2.3)) as the lock's implementation is PSF under strong fairness. The `lock` method has the annotated precondition `true`, and the `unlock` method has the annotated precondition (`l=cid`).

One may think that the proof for this example would be trivial, because the implementation code and the atomic partial specification code are exactly the same, so they could always be executed simultaneously. Our proof is indeed simple. In our proof, we indeed let the implementation and the specification be executed simultaneously. But there is another key point in our proof: we also need to prove that the implementation code ensures PSF (not only PDF) under strong fairness.

We first define the invariant fence I, the rely/guarantee conditions R and G, and the definite actions $\mathcal{D}$ in Fig. 36. Note that since we are verifying PSF, all the actions should be defined as level-0 actions. This is very different from the proof for test-and-set locks (see Sec. 8.1).

Fig. 37 gives the proof outline. When verifying the **await** statement at line 1, we use a similar metric f (defined at the bottom of Fig. 37) as in the proof of test-and-set locks. The difference is, for test-and-set locks, the lock-acquire actions from the environment threads are level-1 actions, allowing the metric f to increase. But here, the lock-acquire actions are level-0 actions. The metric f is still allowed to increase, because the definite progress condition for **await** under strong fairness (see the *second* bullet of Definition 7.2) requires the metric to not increase only when the environment transitions satisfy $((\mathtt{l} \neq 0) \ltimes (\mathtt{l} \neq 0))$ (lock-acquire actions are not this case). As we explained in Sec. 7, this is the key to make use of strong fairness.

$$I \stackrel{\text{def}}{=} \mathsf{unlocked} \vee \mathsf{locked}$$
$$\mathsf{unlocked} \stackrel{\text{def}}{=} (\mathtt{l} = \mathtt{L} = 0)$$
$$\mathsf{locked} \stackrel{\text{def}}{=} \exists \mathtt{t}.\ \mathsf{lockedBy}_{\mathtt{t}}$$
$$\mathsf{lockedBy}_{\mathtt{t}} \stackrel{\text{def}}{=} (\mathtt{l} = \mathtt{L} = \mathtt{t}) \wedge (\mathtt{t} \in \mathtt{TIDS})$$

$$G_{\mathtt{t}} \stackrel{\text{def}}{=} (\mathit{Lock}_{\mathtt{t}} \vee \mathit{Unlock}_{\mathtt{t}} \vee \mathsf{Id}) * \mathsf{Id} \wedge (I \ltimes I)$$
$$\mathit{Lock}_{\mathtt{t}} \stackrel{\text{def}}{=} \mathsf{unlocked} \ltimes_0 \mathsf{lockedBy}_{\mathtt{t}}$$
$$\mathit{Unlock}_{\mathtt{t}} \stackrel{\text{def}}{=} \mathsf{lockedBy}_{\mathtt{t}} \ltimes_0 \mathsf{unlocked}$$

$$\mathcal{D}_{\mathtt{t}} \stackrel{\text{def}}{=} \mathsf{false} \rightsquigarrow \mathsf{true}$$

Fig. 36. Auxiliary definitions for verifying the simple PSF locks under strong fairness.

```
lock():
   {I ∧ arem(await(L = 0){L := cid})}
1  await (l = 0) {
     {unlocked ∧ arem(await(L = 0){L := cid})}
2    l := cid;
     {lockedBy_cid ∧ arem(skip)}
3  }
   {lockedBy_cid ∧ arem(skip)}

unlock():
   {lockedBy_cid ∧ arem(L := 0)}
4  l := 0;
   {I ∧ arem(skip)}
```

Here the await command at lines 1–3 is verified as follows. Let

$$f_{\mathtt{t}}(\mathfrak{S}) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } \mathfrak{S} \models \mathsf{locked} \\ 0 & \text{if } \mathfrak{S} \models \mathsf{unlocked} \end{cases}$$

We can prove: $I \Rightarrow (R\colon \mathcal{D} \bullet\!\!\xrightarrow{f} (\mathtt{l} = 0, \mathtt{L} = 0))$.

Fig. 37. Proof outline of the simple PSF locks under strong fairness.

C.4.2 Simple locks implemented using the wrapper for PSF under weak fairness. The code in Fig. 38 results from applying the wrapper $\text{wr}_{\text{PSF}}^{\text{wfair}}$ to the atomic partial specification (2.3) for locks. Note that both `L_ACQ'` and `L_REL` are wrapped with the updates on `listid`. As we have discussed in Sec. 6, we do not intend to claim that the methods in Fig. 38 give the simplest PSF lock under weak fairness. In fact, we have verified that the example in Sec. 8.3 (which has a simpler lock-release method) is also PSF under weak fairness.

```
initialize(){  l := 0; listid := ϵ; }

lock(){
 1  listid := listid ++ [(cid, 'l=0')];
 2  await (l = 0 /\ cid = enhd(listid)) {
 3    l := cid;
 4    listid := listid \ cid;
 5  }
}

unlock(){
 6  listid := listid ++ [(cid, 'true')];
 7  await (cid = enhd(listid)) {
 8    l := 0;
 9    listid := listid \ cid;
10  }
}
```

Fig. 38. Simple locks implemented using the wrapper for PSF under weak fairness.

Below we verify the lock implementation in Fig. 38 using our logic with respect to the atomic partial specification (2.3). By the logic soundness theorem, we know the code in Fig. 38 is PSF under weak fairness. By the Abstraction Theorem, we also know that the code is a contextual refinement of itself under weak fairness.

Fig. 39 defines the invariant I, the rely/guarantee conditions R and G, and the definite actions $\mathcal{D}$. The definitions are very similar to the ones in Fig. 13 for the example in Sec. 8.3. One of the differences is that the items in the list `listid` may be either (t, 'l = 0') or (t, 'true'), depending on whether the thread t is calling the `lock` or `unlock` method. And since there could be at most one thread calling the `unlock` method, there could be at most one (t, 'true') in `listid`. The `unlock` method performs two actions *ReqUnlock* and *Unlock*. The actions *AcqLock* and *Unlock* are definite (see the definition of $\mathcal{D}$ in Fig. 39). Note that since we are verifying PSF, all the actions are level-0 actions.

Fig. 40 gives the proof outline. The **await** statement in the `lock` method is verified in a similar way as we explained in Sec. 8.3. The verification of the **await** statement in the `unlock` method is trivial, because when $\text{inUnlock}_{\texttt{cid}}$ holds, the **await** condition (`cid = enhd(listid)`) always holds for the current thread.

$tb ::= \epsilon \mid (\mathsf{t}, \text{`}B\text{'}) :: tb$

$\mathsf{list2set}(\epsilon) \stackrel{\text{def}}{=} \emptyset$

$\mathsf{list2set}((\mathsf{t}, \text{`}B\text{'}) :: tb) \stackrel{\text{def}}{=} \{(\mathsf{t}, \text{`}B\text{'})\} \cup \mathsf{list2set}(tb)$

$\mathsf{t} \notin tb \text{ iff } \neg\exists B.\ (\mathsf{t}, \text{`}B\text{'}) \in \mathsf{list2set}(tb)$

$\mathsf{diff}(tb) \text{ iff } \forall tb', tb'', tb''', \mathsf{t}_1, \mathsf{t}_2, B_1, B_2.\ (tb = tb' +\!\!+ [(\mathsf{t}_1, \text{`}B_1\text{'})] +\!\!+ tb'' +\!\!+ [(\mathsf{t}_2, \text{`}B_2\text{'})] +\!\!+ tb''') \implies (\mathsf{t}_1 \neq \mathsf{t}_2)$

$\mathsf{all\_reqlock}(tb) \text{ iff } \forall \mathsf{t}, B.\ (\mathsf{t}, \text{`}B\text{'}) \in \mathsf{list2set}(tb) \implies (B = (\mathtt{l} = 0))$

$\mathsf{one\_requnlock_t}(tb) \text{ iff } \exists tb', tb''.\ (tb = tb' +\!\!+ [(\mathsf{t}, \text{`true'})] +\!\!+ tb'') \wedge (\forall \mathsf{t}', B.\ (\mathsf{t}', \text{`}B\text{'}) \in \mathsf{list2set}(tb' +\!\!+ tb'') \implies (B = (\mathtt{l} = 0)))$

$I \stackrel{\text{def}}{=} \exists tb.\ \mathsf{lock}(tb) \qquad \mathsf{lock}(tb) \stackrel{\text{def}}{=} \mathsf{unlocked}(tb) \vee \mathsf{notUnlocked}(tb)$

$\mathsf{notUnlocked}(tb) \stackrel{\text{def}}{=} \exists \mathsf{t}.\ \mathsf{locked_t}(tb) \vee \mathsf{inUnlock_t}(tb)$

$\mathsf{unlocked}(tb) \stackrel{\text{def}}{=} (\mathtt{l} = \mathsf{L} = 0) * (\mathtt{listid} = tb) \wedge \mathsf{all\_reqlock}(tb) \wedge \mathsf{diff}(tb)$

$\mathsf{locked_t}(tb) \stackrel{\text{def}}{=} (\mathtt{l} = \mathsf{L} = \mathsf{t}) * (\mathtt{listid} = tb) \wedge \mathsf{all\_reqlock}(tb) \wedge \mathsf{diff}(tb)$

$\mathsf{inUnlock_t}(tb) \stackrel{\text{def}}{=} (\mathtt{l} = \mathsf{L} = \mathsf{t}) * (\mathtt{listid} = tb) \wedge \mathsf{one\_requnlock_t}(tb) \wedge \mathsf{diff}(tb)$

$\mathsf{locked_t} \stackrel{\text{def}}{=} \exists tb.\ \mathsf{locked_t}(tb) \qquad \mathsf{inUnlock_t} \stackrel{\text{def}}{=} \exists tb.\ \mathsf{inUnlock_t}(tb)$

$P_\mathsf{t} \stackrel{\text{def}}{=} \exists tb.\ \mathsf{lock}(tb) \wedge (\mathsf{t} \notin tb)$

$\mathsf{lockedBy_t}(tb) \stackrel{\text{def}}{=} \mathsf{locked_t}(tb) \wedge (\mathsf{t} \notin tb) \qquad \mathsf{lockedBy_t} \stackrel{\text{def}}{=} \exists tb.\ \mathsf{lockedBy_t}(tb)$

$\mathsf{lockReq_t}(tb) \stackrel{\text{def}}{=} \mathsf{lock}(tb) \wedge ((\mathsf{t}, \text{`}\mathtt{l} = 0\text{'}) \in \mathsf{list2set}(tb)) \qquad \mathsf{lockReq_t} \stackrel{\text{def}}{=} \exists tb.\ \mathsf{lockReq_t}(tb)$

$G_\mathsf{t} \stackrel{\text{def}}{=} (\mathit{ReqLock}_\mathsf{t} \vee \mathit{AcqLock}_\mathsf{t} \vee \mathit{ReqUnlock}_\mathsf{t} \vee \mathit{Unlock}_\mathsf{t} \vee \mathsf{Id}) * \mathsf{Id} \wedge (I \ltimes I)$

$\mathit{ReqLock}_\mathsf{t} \stackrel{\text{def}}{=} \exists tb.\ ((\mathtt{listid} = tb) \wedge (\mathsf{t} \notin tb)) \ltimes (\mathtt{listid} = tb +\!\!+ [(\mathsf{t}, \text{`}\mathtt{l} = 0\text{'})])$

$\mathit{AcqLock}_\mathsf{t} \stackrel{\text{def}}{=} \exists tb.\ \mathsf{unlocked}((\mathsf{t}, \text{`}\mathtt{l} = 0\text{'}) :: tb) \ltimes \mathsf{lockedBy_t}(tb)$

$\mathit{ReqUnlock}_\mathsf{t} \stackrel{\text{def}}{=} \exists tb.\ \mathsf{lockedBy_t}(tb) \ltimes \mathsf{inUnlock_t}(tb +\!\!+ [(\mathsf{t}, \text{`true'})])$

$\mathit{Unlock}_\mathsf{t} \stackrel{\text{def}}{=} \exists tb, tb'.\ \mathsf{inUnlock_t}(tb +\!\!+ [(\mathsf{t}, \text{`true'})] +\!\!+ tb') \ltimes \mathsf{unlocked}(tb +\!\!+ tb')$

$\mathcal{D}_\mathsf{t} \stackrel{\text{def}}{=} (\forall tb.\ \mathsf{unlocked}((\mathsf{t}, \text{`}\mathtt{l} = 0\text{'}) :: tb) \rightsquigarrow \mathsf{lockedBy_t}(tb)) \wedge (\forall tb, tb'.\ \mathsf{inUnlock_t}(tb +\!\!+ [(\mathsf{t}, \text{`true'})] +\!\!+ tb') \rightsquigarrow \mathsf{unlocked}(tb +\!\!+ tb'))$

Fig. 39. Auxiliary definitions for verifying the locks with the wrapper for PSF under weak fairness.

```
lock():
    {P_cid ∧ arem(await(L = 0){L := cid})}
 1  listid := listid ++ [(cid, 'l=0')];
    {lockReq_cid ∧ arem(await(L = 0){L := cid})}
 2  await (l = 0 /\ cid = enhd(listid)) {
      {∃tb. unlocked((cid,'l = 0') :: tb) ∧ arem(await(L = 0){L := cid})}
 3    l := cid;
 4    listid := listid \ cid;
      {∃tb. lockedBy_cid(tb) ∧ arem(skip)}
 5  }
    {lockedBy_cid ∧ arem(skip)}


unlock():
    {lockedBy_cid ∧ arem(L := 0)}
 6  listid := listid ++ [(cid, 'l=0')];
    {inUnlock_cid ∧ arem(L := 0)}
 7  await (cid = enhd(listid)) {
      {inUnlock_cid ∧ arem(L := 0)}
 8    l := 0;
 9    listid := listid \ cid;
      {P_cid ∧ arem(skip)}
10  }
    {P_cid ∧ arem(skip)}
```

Here the await command at lines 2-5 is verified as follows. Let

$$f_t(\Im) \stackrel{\text{def}}{=} \begin{cases} 2k+1 & \text{if } \exists tb, tb'.\ (\Im \models \text{notUnlocked}(tb \mathbin{++} [(t, \text{'l = 0'})] \mathbin{++} tb')) \wedge |tb| = k \\ 2k & \text{if } \exists tb, tb'.\ (\Im \models \text{unlocked}(tb \mathbin{++} [(t, \text{'l = 0'})] \mathbin{++} tb')) \wedge |tb| = k \end{cases}$$

We can prove: lockReq $\Rightarrow (R\colon \mathcal{D} \circ\!\!\xrightarrow{f} (\texttt{l = 0} \wedge \texttt{cid = enhd(listid)}, \texttt{L = 0}))$.

Here the await command at lines 7-10 is verified as follows. Let f be a constant function. We can prove: inUnlock $\Rightarrow (R\colon \mathcal{D} \circ\!\!\xrightarrow{f} (\texttt{cid = enhd(listid)}, \text{true}))$.

Fig. 40. Proof outline of the locks implemented using the wrapper for PSF under weak fairness.

C.4.3 Simple locks implemented using the wrapper for PDF under weak fairness. The code in Fig. 41 results from applying the wrapper $\mathsf{wr}_{\mathsf{PDF}}^{\mathsf{wfair}}$ to the atomic partial specification (2.3) for locks. Both `L_ACQ'` and `L_REL` are wrapped with the updates on the shared variable `done`. Again, we do not intend to claim that the code in Fig. 41 is the simplest PDF lock under weak fairness. In fact, the atomic partial specification (2.3) itself is already PDF under weak fairness.

```
initialize(){  l := 0; done := false; }

lock(){
 1  await (l = 0 /\ !done) {
 2    l := cid;
 3    done := true;
 4  }
 5  done := false;
 6  await (!done) {};
}

unlock(){
 7  await (!done) {
 8    l := 0;
 9    done := true;
10  }
11  done := false;
12  await (!done) {};
}
```

Fig. 41. Simple locks implemented using the wrapper for PDF under weak fairness.

Below we verify the lock implementation in Fig. 41 using our logic with respect to the atomic partial specification (2.3). By the logic soundness theorem, we know the code in Fig. 41 is PDF under weak fairness. By the Abstraction Theorem, we also know that the code is a contextual refinement of itself under weak fairness.

Fig. 42 defines the invariant fence I, the object invariant P (see the OBJ rule in Fig. 10), the rely/guarantee conditions R and G, and the definite actions $\mathcal{D}$. The `lock` method performs two actions *Lock* and *LockDone*, and the `unlock` method performs two actions *Unlock* and *UnlockDone*. Here we play a trick and let the reset of `done` (i.e., line `10` in the `unlock` method in Fig. 41) correspond to the lock-release step at the abstract side, as shown in the definitions of *Unlock* and *UnlockDone*. Then we could know from the abstract lock `L` which thread needs to perform the reset of `done`.

The actions *LockDone* and *UnlockDone* are definite (see the definition of $\mathcal{D}$ in Fig. 42). We define *Lock* and *Unlock* as level-1 actions, and define other actions as non-delaying actions. In fact, for this example, *Unlock* could also be defined as non-delaying actions, and the proofs could be done without any change. (The reason why we label *Unlock* with level 1 here is to make the definitions usable in the proofs for the next example in Fig. 44. We will explain this in Sec. C.4.4.)

Fig. 43 gives the proof outline. The verification of the four **await** statements are given at the bottom of the figure. For the second **await** statement in `lock` (line `6`) and the first **await** statement in `unlock` (lines `7-10`), the proofs are trivial, because when $\mathsf{locked}_{\mathtt{cid}}(\mathsf{false})$ holds, the **await** condition ¬`done` always holds.

$$I \stackrel{\text{def}}{=} \exists t.\ \mathsf{locked}_t \vee \mathsf{toUnlocked}_t \vee \mathsf{unlocked}$$

$$\mathsf{unlocked} \stackrel{\text{def}}{=} (\mathsf{l} = \mathsf{L} = 0) * (\mathsf{done} = \mathsf{false})$$

$$\mathsf{toUnlocked}_t \stackrel{\text{def}}{=} (\mathsf{l} = 0) * (\mathsf{L} = t) * (\mathsf{done} = \mathsf{true})$$

$$\mathsf{locked}_t(b) \stackrel{\text{def}}{=} (\mathsf{l} = \mathsf{L} = t) * (\mathsf{done} = b) \qquad \mathsf{locked}_t \stackrel{\text{def}}{=} \exists b.\ \mathsf{locked}_t(b)$$

$$\mathsf{envLocked}_t(b) \stackrel{\text{def}}{=} \exists t'.\ \mathsf{locked}_{t'}(b) \wedge (t' \neq t) \qquad \mathsf{envLocked}_t \stackrel{\text{def}}{=} \exists b.\ \mathsf{envLocked}_t(b)$$

$$\mathsf{envToUnlocked}_t \stackrel{\text{def}}{=} \exists t'.\ \mathsf{toUnlocked}_{t'} \wedge (t' \neq t)$$

$$\mathsf{notOwn}_t \stackrel{\text{def}}{=} \mathsf{unlocked} \vee \mathsf{envLocked}_t \vee \mathsf{envToUnlocked}_t$$

$$P_t \stackrel{\text{def}}{=} \mathsf{locked}_t(\mathsf{false}) \vee \mathsf{notOwn}_t$$

$$G_t \stackrel{\text{def}}{=} (\mathit{Lock}_t \vee \mathit{LockDone}_t \vee \mathit{Unlock}_t \vee \mathit{UnlockDone}_t \vee \mathsf{Id}) * \mathsf{Id} \wedge (I \ltimes I)$$

$$\mathit{Lock}_t \stackrel{\text{def}}{=} \mathsf{unlocked} \ltimes_1 \mathsf{locked}_t(\mathsf{true})$$

$$\mathit{LockDone}_t \stackrel{\text{def}}{=} \mathsf{locked}_t(\mathsf{true}) \ltimes_0 \mathsf{locked}_t(\mathsf{false})$$

$$\mathit{Unlock}_t \stackrel{\text{def}}{=} \mathsf{locked}_t(\mathsf{false}) \ltimes_1 \mathsf{toUnlocked}_t$$

$$\mathit{UnlockDone}_t \stackrel{\text{def}}{=} \mathsf{toUnlocked}_t \ltimes_0 \mathsf{unlocked}$$

$$\mathcal{D}_t \stackrel{\text{def}}{=} (\mathsf{locked}_t(\mathsf{true}) \rightsquigarrow \mathsf{locked}_t(\mathsf{false})) \wedge (\mathsf{toUnlocked}_t \rightsquigarrow \mathsf{unlocked})$$

Fig. 42. Auxiliary definitions for verifying the locks with the wrapper for PDF under weak fairness.

```
lock():
    {P_cid ∧ ♦ ∧ arem(await(L = 0){L := cid})}
 1  await (l = 0 /\ !done) {
      {unlocked ∧ ♦ ∧ arem(await(L = 0){L := cid})}
 2    l := cid;
 3    done := true;
      {locked_cid(true) ∧ arem(skip)}
 4  }
    {locked_cid(true) ∧ arem(skip)}
 5  done := false;
    {locked_cid(false) ∧ arem(skip)}
 6  await (!done) {};
    {locked_cid(false) ∧ arem(skip)}


unlock():
    {locked_cid(false) ∧ ♦ ∧ arem(L := 0)}
 7  await (!done) {
      {locked_cid(false) ∧ ♦ ∧ arem(L := 0)}
 8    l := 0;
 9    done := true;
      {toUnlocked_cid ∧ arem(L := 0)}
10  }
    {toUnlocked_cid ∧ arem(L := 0)}
11  done := false;
    {notOwn_cid ∧ arem(skip)}
12  await (!done) {};
    {notOwn_cid ∧ arem(skip)}
```

Here the await command at lines 1–4 is verified as follows. Let

$$f_t(\mathfrak{S}) \stackrel{\text{def}}{=} \begin{cases} 3 & \text{if } \mathfrak{S} \models \mathsf{envLocked}_t(\mathsf{true}) \\ 2 & \text{if } \mathfrak{S} \models \mathsf{envLocked}_t(\mathsf{false}) \\ 1 & \text{if } \mathfrak{S} \models \mathsf{envToUnlocked}_t \\ 0 & \text{if } \mathfrak{S} \models \mathsf{unlocked} \end{cases}$$

We can prove: $P \Rightarrow (R\colon \mathcal{D} \overset{f}{\circ\!\!\rightarrow} (1 = 0 \wedge \neg\mathsf{done}, \mathsf{L} = 0))$.

The await command at line 12 can be verified using the same metric f as the above one for lines 1–4. We can prove: $\mathsf{notOwn} \Rightarrow (R\colon \mathcal{D} \overset{f}{\circ\!\!\rightarrow} (\neg\mathsf{done}, \mathsf{true}))$.

The await commands at line 6 and at lines 7–10 are verified in the same way. Let f be a constant function. We can prove: $\mathsf{locked}(\mathsf{false}) \Rightarrow (R\colon \mathcal{D} \overset{f}{\circ\!\!\rightarrow} (\neg\mathsf{done}, \mathsf{true}))$.

Fig. 43. Proof outline of the locks implemented using the wrapper for PDF under weak fairness.

C.4.4 Simple locks implemented using the wrapper for PDF under strong fairness. The code in Fig. 44 results from applying the wrapper $\mathrm{wr}^{\mathrm{sfair}}_{\mathrm{PDF}}$ to the atomic partial specification (2.3) for locks. Again, we do not intend to claim that Fig. 44 is the simplest PDF lock under strong fairness.

```
initialize(){  l := 0; done := false; }

lock(){
 1  local b;
 2  b := done;
 3  while (b) {
 4    b := done;
 5  }
 6  await (l = 0 /\ !done) {
 7    l := cid;
 8    done := true;
 9  }
10  done := false;
11  b := done;
12  while (b) { b := done; }
}

unlock(){
13  local b;
14  b := done;
15  while (b) { b := done; }
16  await (!done) {
17    l := 0;
18    done := true;
19  }
20  done := false;
21  b := done;
22  while (b) {
23    b := done;
24  }
}
```

Fig. 44. Simple locks implemented using the wrapper for PDF under strong fairness.

Below we verify the lock implementation in Fig. 44 using our logic with respect to the atomic partial specification (2.3). By the logic soundness theorem, we know the code in Fig. 44 is PDF under strong fairness. By the Abstraction Theorem, we also know that the code is a contextual refinement of itself under strong fairness.

We give the proof outlines in Fig. 45 and Fig. 46. The proofs use the same invariant fence I, the same object invariant P, the same rely/guarantee conditions R and G, and the same definite actions $\mathcal{D}$, as for verifying the previous example of Fig. 41. The definitions have been given in Fig. 42. As we have mentioned, the action *Unlock* has to be labeled with level 1 for this example. The reason is that the *Unlock* action updates the variable done, which may delay a thread that is executing the loops of lines 3–5 and lines 22–24. In particular, in the proof of lines 22–24 in Fig. 46, the current thread could increase $\Diamond$-tokens when its environment does *Unlock* (that is why the assertion before line 23 is stable).

```
lock():
 1  local b;
    {P_cid ∧ ♦ ∧ arem(await(L = 0){L := cid})}
 2  b := done;
    {P_cid ∧ ♦ ∧ arem(await(L = 0){L := cid}) ∧ (b ∧ ◊ ∨ ¬b)}
 3  while (b) {
      {(unlocked ∨ locked_cid(false) ∨ (envLocked_cid ∨ envToUnlocked_cid) ∧ ◊)
       ∧ ♦ ∧ arem(await(L = 0){L := cid})}
 4    b := done;
      {P_cid ∧ (b ∧ ◊ ∨ ¬b) ∧ ♦ ∧ arem(await(L = 0){L := cid})}
 5  }
    {P_cid ∧ ♦ ∧ arem(await(L = 0){L := cid})}
 6  await (l = 0 /\ !done) {
      {unlocked ∧ ♦ ∧ arem(await(L = 0){L := cid})}
 7    l := cid;
 8    done := true;
      {locked_cid(true) ∧ arem(skip)}
 9  }
    {locked_cid(true) ∧ arem(skip)}
10  done := false;
    {locked_cid(false) ∧ arem(skip)}
11  b := done;
    {¬b ∧ locked_cid(false) ∧ arem(skip)}
12  while (b) { b := done; }
    {locked_cid(false) ∧ arem(skip)}
```

Here the while loop at lines 3–5 is verified as follows. Let

$$Q_t \stackrel{\text{def}}{=} \text{unlocked} \vee \text{locked}_t(\text{false}) \qquad J_t \stackrel{\text{def}}{=} P_t \qquad f_t(\Im) \stackrel{\text{def}}{=} \begin{cases} 3 & \text{if } \Im \models \text{envLocked}_t(\text{true}) \\ 2 & \text{if } \Im \models \text{envLocked}_t(\text{false}) \\ 1 & \text{if } \Im \models \text{envToUnlocked}_t \\ 0 & \text{if } \Im \models \text{unlocked} \end{cases}$$

We can prove: $J \Rightarrow (R, \mathsf{Id} : \mathcal{D} \xrightarrow{f} (Q, (\mathsf{L} = 0)))$.

The await command at lines 6–9 is verified using the same metric f as the above one for the while loop at lines 3–5. We can prove: $P \Rightarrow (R : \mathcal{D} \bullet\!\!\xrightarrow{f} (\mathsf{l} = 0, \mathsf{L} = 0))$.

Proof of the while loop at line 12 is trivial, since the loop condition b does not hold.

Fig. 45. Proof outline for the lock method of the PDF lock under strong fairness (using the same I, G, R and $\mathcal{D}$ as Fig. 42).

```
unlock():
13  local b;
    {locked_cid(false) ∧ ♦ ∧ arem(L := 0)}
14  b := done;
    {locked_cid(false) ∧ ♦ ∧ arem(L := 0) ∧ ¬b}
15  while (b) { b := done; }
    {locked_cid(false) ∧ ♦ ∧ arem(L := 0)}
16  await (!done) {
      {locked_cid(false) ∧ ♦ ∧ arem(L := 0)}
17    l := 0;
18    done := true;
      {toUnlocked_cid ∧ arem(L := 0)}
19  }
    {toUnlocked_cid ∧ arem(L := 0)}
20  done := false;
    {notOwn_cid ∧ arem(skip)}
21  b := done;
    {notOwn_cid ∧ arem(skip) ∧ (b ∧ ◊ ∨ ¬b)}
22  while (b) {
      {(unlocked ∨ envLocked_cid(false) ∨ (envLocked_cid(true) ∨ envToUnlocked_cid) ∧ ◊) ∧ arem(skip)}
23    b := done;
      {notOwn_cid ∧ (b ∧ ◊ ∨ ¬b) ∧ arem(skip)}
24  }
    {notOwn_cid ∧ arem(skip)}
```

Here the proof of the while loop at line 15 is trivial, since the loop condition b does not hold.

The await command at lines 16–19 is verified as follows. Let f be a constant function. We can prove: $\mathsf{locked}(\mathsf{false}) \Rightarrow (R\colon \mathcal{D} \bullet\!\!\xrightarrow{f} (\neg\mathsf{done}, \mathsf{true}))$.

The while loop at lines 22–24 is verified as follows. Let

$$Q_t \stackrel{\text{def}}{=} \mathsf{envLocked}_t(\mathsf{false}) \vee \mathsf{unlocked} \qquad J_t \stackrel{\text{def}}{=} \mathsf{notOwn}_t \qquad f_t(\Im) \stackrel{\text{def}}{=} \begin{cases} 3 & \text{if } \Im \models \mathsf{envLocked}_t(\mathsf{true}) \\ 2 & \text{if } \Im \models \mathsf{envLocked}_t(\mathsf{false}) \\ 1 & \text{if } \Im \models \mathsf{envToUnlocked}_t \\ 0 & \text{if } \Im \models \mathsf{unlocked} \end{cases}$$

We can prove: $J \Rightarrow (R, \mathsf{Id}\colon \mathcal{D} \xrightarrow{f} (Q, \mathsf{true}))$.

Fig. 46. Proof outline for the unlock method of the PDF lock under strong fairness (using the same I, G, R and $\mathcal{D}$ as Fig. 42).