Observing Progress Properties via Contextual Refinements (Extended Version)

Hongjin Liang[1], Jan Hoffmann[2], Xinyu Feng[1], and Zhong Shao[2]

[1] University of Science and Technology of China
[2] Yale University

NOTE: Compared to the submitted version, we include the full semantics and the formal definitions of linearizability in Section 3; the full definitions of progress properties in Section 4; and proofs of all the theorems in the appendix. Besides, we give alternative formulations of these progress properties in Section 4, which are simpler and more constructive (although may not be close to the natural language interpretations). From them, we could clearly see the relationships between progress properties. To support the new formulations, we add a new event to represent thread spawning and slightly change the generation of event traces. The changes are independent from our contextual refinement framework.

Abstract. Implementations of concurrent objects should guarantee linearizability and a progress property such as wait-freedom, lock-freedom, obstruction-freedom, starvation-freedom, or deadlock-freedom. Conventional informal or semi-formal definitions of these progress properties describe conditions under which a method call is guaranteed to complete, but it is unclear how these definitions can be utilized to formally verify system software in a layered and modular way.
In this paper, we propose a unified framework based on contextual refinements to show exactly how progress properties affect the behaviors of client programs. We give formal operational definitions of all common progress properties and prove that for linearizable objects, each progress property is equivalent to a specific type of contextual refinement that preserves termination. The equivalence ensures that verification of such a contextual refinement for a concurrent object guarantees both linearizability and the corresponding progress property. Contextual refinement also enables us to verify safety and liveness properties of client programs at a high abstraction level by soundly replacing concrete method implementations with abstract atomic operations.

1 Introduction

A concurrent object consists of shared data and a set of methods that provide an interface for client threads to manipulate and access the shared data. The synchronization of simultaneous data access within the object affects the progress of the execution of the client threads in the system.

Various progress properties have been proposed for concurrent objects. The most important ones are wait-freedom, lock-freedom and obstruction-freedom for non-blocking implementations, and starvation-freedom and deadlock-freedom for lock-based implementations. These properties describe conditions under which method calls are guaranteed to successfully complete in an execution. For example, lock-freedom guarantees that "infinitely often some method call finishes in a finite number of steps" [10].

Nevertheless, the common informal or semi-formal definitions of the progress properties are difficult to use in a modular and layered program verification because they fail to describe how the progress properties affect clients. In a modular verification of client threads, the concrete implementation Π of the object methods should be replaced by an abstraction (or specification) Π_A that consists of equivalent atomic methods. The progress properties should then characterize whether and how the behaviors of a client program will be affected if a client uses Π instead of Π_A. In particular, we are interested in systematically studying whether the termination of a client using the abstract methods Π_A will be preserved when using an implementation Π with some progress guarantee.

To prove the soundness of such a layered and modular verification of concurrent systems, previous work on verifying the *safety* of concurrent objects (*e.g.*, [4, 5]) has shown that linearizability—a standard safety criterion for concurrent objects—and contextual refinement are equivalent. Informally, an implementation Π is a contextual refinement of a (more abstract) implementation Π_A, if every observable behavior of any client program using Π can also be observed when the client uses Π_A instead. To obtain equivalence to linearizability, the observable behaviors include I/O events but not divergence. Recently, Gotsman and Yang [6] showed that a client program that diverges using a linearizable and *lock-free* object implementation must also diverge when using the abstract atomic operations instead. Their work reveals a connection between lock-freedom and a form of contextual refinement which preserves termination as well as safety properties. It is unclear how other progress guarantees affect termination of client programs, how they are related to contextual refinements, and how we can prove liveness properties of client programs in a modular way.

This paper studies all five commonly-used progress properties and their relationships to contextual refinements. We propose a unified framework in which a certain type of termination-sensitive contextual refinement is equivalent to linearizability together with one of the progress properties. The idea is to identify different observable behaviors for different progress properties. For example, for the contextual refinement for lock-freedom we observe the divergence of the whole program, while for wait-freedom we also need to observe which threads in the program diverge. For the lock-based progress properties starvation-freedom and deadlock-freedom, we have to take fair schedulers into account.

To formally develop the framework, we first have to formalize the progress properties. However, most existing definitions of these progress properties are informal and sometimes ambiguous (*e.g.*, [8, 10]). Some formal or semi-formal definitions are difficult to apply in Hoare-style verification (*e.g.*, [3]) or even

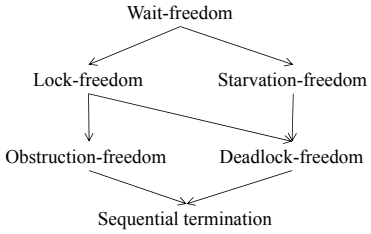

Fig. 1: Relationships between Progress Properties

flawed. (See Appendix A for an example program demonstrating that the recent formulation of obstruction-freedom given by Herlihy and Shavit [9] is inconsistent with the common intuition.)

In this paper, we give formal operational definitions of these progress properties which follow their intuitions, and unify them in our contextual refinement framework. In summary, we make the following contributions:

- We formalize the definitions of the most common progress properties wait-freedom, lock-freedom, obstruction freedom, starvation-freedom, and deadlock-freedom. Our formulation is based on possibly infinite event traces that are operationally generated by any client using the object implementation.
- Based on our formalization, we prove relationships between the progress properties. For example, wait-freedom implies lock-freedom and starvation-freedom implies deadlock-freedom. These relationships form a lattice shown in Figure 1 (where the arrows represent implications). We close the lattice with a bottom element that we call *sequential termination*, a progress property in the sequential setting. It is weaker than any other progress property.
- We develop a unified framework to observe progress properties via contextual refinements. With linearizability, each progress property is proved equivalent to a contextual refinement which takes into account divergence of programs. The formal proofs of our results can be found in Appendix B.

Our contextual refinement framework provides another point of view to understand progress properties. The contextual refinement implied by linearizability and a progress guarantee precisely characterizes the properties at the abstract level that are preserved by the object implementation. When proving these properties of a client of the object, we can soundly replace the concrete method implementations by its abstract operations. On the other hand, since the contextual refinement also implies linearizability and the progress property, we can borrow ideas from existing proof methods for contextual refinements, such as simulations (*e.g.*, [13]) and logical relations (*e.g.*, [2]), to verify linearizability and the progress guarantee together.

In the remainder of this paper, we first informally explain the progress properties and our framework in Section 2. We then introduce the formal setting in Section 3; including the definition of linearizability as the safety criterion of objects. We formulate the progress properties and the contextual refinement framework in Section 4. We discuss related work and conclude in Section 5.

2 Informal Account

In this section, we informally describe our results. We first give an overview of linearizability and its equivalence to the basic contextual refinement. Then we explain the progress properties and summarize our new equivalence results.

Linearizability and Contextual Refinement *Linearizability* is the standard correctness criterion for concurrent objects [10]. Intuitively, linearizability describes atomic behaviors of object implementations. It requires that each method call should appear to take effect instantaneously at some moment between its invocation and return.

Linearizability intuitively establishes a correspondence between the object implementation Π and the intended atomic operations Π_A. This correspondence can also be understood as a *contextual refinement*. Informally, we say that Π is a contextual refinement of Π_A, $\Pi \sqsubseteq \Pi_A$, if substituting Π_A for Π in any context (*i.e.*, in a client program) does not add observable behaviors. Human beings as external observers cannot tell that Π_A has been replaced by Π from monitoring the behaviors of the client program.

It has been proved [4, 5, 12] that linearizability is equivalent to a contextual refinement in which the observable behaviors are finite traces of I/O events. Thus this basic contextual refinement can be used to distinguish linearizable objects from non–linearizable ones. But it cannot identify progress properties of objects.

Progress Properties Figure 2 shows several implementations of a counter with different progress guarantees that we study in this paper. A counter object provides the two methods `inc` and `dec` for incrementing and decrementing a shared variable `x`. The implementations given here are not intended to be practical but merely to demonstrate the meanings of the progress properties.

Informally, an object implementation is *wait-free*, if it guarantees that every thread can complete any started operation of the data structure in a finite number of steps [7]. Figure 2(a) shows an ideal wait-free implementation in which the increment and the decrement are done atomically. This unrealistic implementation is obviously wait-free since it guarantees termination of every method call regardless of interference from other threads. Note that realistic implementations of wait-free counters are more complex and involve arrays and atomic snapshots [1].

Lock-freedom is similar to wait-freedom but only guarantees that *some* thread will complete an operation in a finite number of steps [7]. Typical lock-free implementations (such as the well-known Treiber stack, HSY elimination-backoff stack and Harris-Michael lock-free list) use the atomic compare-and-swap instruction `cas` in a loop to repeatedly attempt an update until it succeeds. Figure 2(b) shows such an implementation of the counter object. It is lock-free, because whenever `inc` and `dec` operations are executed concurrently, there always exists some successful update. Note that this object is not wait-free. For the following

```
1 inc() { x := x + 1; }
2 dec() { x := x - 1; }
```

(a) Wait-Free (Ideal) Impl.

```
1 inc() {
2  local t, b;
3  do {
4    t := x;
5    b := cas(&x,t,t+1);
6  } while(!b);
7 }
```

(b) Lock-Free Impl.

```
 1 inc() {
 2   while (i < 10) {
 3     i := i + 1;
 4   }
 5   x := x + 1;
 6 }
 7 dec() {
 8   while (i > 0) {
 9     i := i - 1;
10   }
11   x := x - 1;
12 }
```

(c) Obstruction-Free Impl.

```
1 inc() {
2   TestAndSet_lock();
3   x := x + 1;
4   TestAndSet_unlock();
5 }
```

(d) Deadlock-Free Impl.

```
 1  inc() {
 2    Bakery_lock();
 3    x := x + 1;
 4    Bakery_unlock();
 5  }
```

(e) Starvation-Free Impl.

Fig. 2: Counter Objects with Methods `inc` and `dec`

program (2.1), the `cas` instruction in the method called by the left thread may continuously fail due to the continuous updates of `x` made by the right thread.

```
inc();   ||   while(true) inc();                    (2.1)
```

Herlihy *et al.* [8] propose *obstruction-freedom* which "guarantees progress for any thread that eventually executes in isolation" (*i.e.*, without other active threads in the system). They present two double-ended queues as examples. In Figure 2(c) we show an obstruction-free counter that may look contrived but nevertheless illustrates the idea of the progress property.

The implementation introduces a variable `i`, and lets `inc` perform the atomic increment after increasing `i` to 10 and `dec` do the atomic decrement after decreasing `i` to 0. Whenever a method is executed in isolation (*i.e.*, without interference from other threads), it will complete. Thus the object is obstruction-free. It is not lock-free, because for the client

```
inc();   ||   dec();                                (2.2)
```

which executes an increment and a decrement concurrently, it is possible that neither of the method calls returns. For instance, under a specific schedule, every increment over `i` made by the left thread is immediately followed by a decrement from the right thread.

Wait-freedom, lock-freedom, and starvation-freedom are progress properties for non-blocking implementations, where a delay of a thread cannot prevent other threads from making progress. In contrast, deadlock-freedom and starvation-freedom are progress properties for lock-based implementations. A delay of a thread holding a lock will block other threads which request the lock.

Deadlock-freedom and starvation-freedom are often defined in terms of locks and critical sections. Deadlock-freedom guarantees that some thread will succeed in acquiring the lock, and starvation-freedom states that every thread attempting to acquire the lock will eventually succeed [10]. For example, a test-and-set spin lock is deadlock-free but not starvation-free. In a concurrent access, some thread

	Wait-Free	Lock-Free	Obstruction-Free	Deadlock-Free	Starvation-Free
Π_A	(t, Div.)	Div.	Div.	Div.	Div. if Fair
Π	(t, Div.)	Div.	Div. if Isolating	Div. if Fair	Div. if Fair

Table 1: Observing Progress Properties via Contextual Refinements $\Pi \sqsubseteq \Pi_A$

will successfully set the bit and get the lock, but there might be a thread that is continuously failing to get the lock. Lamport's bakery lock is starvation-free. It ensures that threads can acquire locks in the order of their requests.

However, as noted by Herlihy and Shavit [9], the above definitions based on locks are unsatisfactory, because it is often difficult to identify a particular field in the object as a lock. Instead, they suggest to define them in terms of method calls. They also notice that the above definitions implicitly assume that every thread acquiring the lock will eventually release it. This assumption requires *fair* scheduling, *i.e.*, every thread gets eventually executed.

Following Herlihy and Shavit [9], we say an object is *deadlock-free*, if in each *fair* execution there always exists some method call that can finish. As an example in Figure 2(d), we use a test-and-set lock to synchronize the increments of the counter. Since some thread is guaranteed to acquire the test-and-set lock, the method call of that thread is guaranteed to finish. Thus the object is deadlock-free. Similarly, a *starvation-free* object guarantees that every method call can finish in fair executions. Figure 2(e) shows a counter implemented with Lamport's bakery lock. It is starvation-free since the bakery lock ensures that every thread can acquire the lock and hence every method call can eventually return.

Our Results None of the above definitions of the five progress properties describes their guarantees regarding the behaviors of client code. In this paper, we define a series of contextual refinements to characterize the effects over client behaviors when the client uses objects with some progress properties. We show that linearizability together with a progress property is equivalent to a certain termination-sensitive contextual refinement. Table 1 summarizes our results.

For each progress property, the new contextual refinement $\Pi \sqsubseteq \Pi_A$ cares about various divergence behaviors and/or fairness at the implementation level (the third row in Table 1) and at the abstract side (the second row), in addition to the I/O events in the basic contextual refinement for linearizability.

- For wait-freedom, we need to observe the divergence of each individual thread t, represented by "(t, Div.)" in Table 1, at both the concrete and the abstract levels. We show that, if the thread t of a client program diverges when the client uses a linearizable and wait-free object Π, then thread t must also diverge when using Π_A instead.
- The case for lock-freedom is similar, except that we now consider the divergence behaviors of the *whole* client program rather than individual threads (denoted by "Div." in Table 1). If a client diverges when using a linearizable and lock-free object Π, it must also diverge when it uses Π_A instead.

- For obstruction-freedom, we focus on the behaviors of *isolating* executions at the concrete side (denoted by "Div. if Isolating" in Table 1). In those executions, eventually only one thread is running. We show that, if a client diverges in isolating executions when it uses a linearizable and obstruction-free object Π, it must also diverge in some abstract executions.
- For deadlock-freedom, we only care about *fair* executions at the concrete level (denoted by "Div. if Fair" in Table 1).
- Starvation-freedom restricts our considerations to fair executions at both the concrete and the abstract levels. A client using Π could diverge in fair executions, only if it can also diverge in fair executions when using Π_A instead.

These new contextual refinements allow us to identify linearizable objects with progress properties. We will formalize the results and give examples in Section 4.

3 Formal Setting and Linearizability

In this section, we formalize linearizability and show its equivalence to a contextual refinement that preserves safety properties. This equivalence is the basis of our new results that relate contextual refinement and progress properties.

Language and Semantics We use a similar language as in previous work of Liang and Feng [12]. As shown in Figure 3, a program W consists of several client threads that run in parallel. Each thread could call the methods declared in the object Π. A method f is defined as a pair (x, C), where x is the formal argument and C is the method body. We write $f \rightsquigarrow (x, C)$. The object Π could be either concrete with fine-grained code that we want to verify, or abstract (usually denoted as Π_A in the following) that we consider as the specification. For the latter case, each method body should be an atomic operation of the form $\langle C \rangle$ and it should be always safe to execute it. For simplicity, we assume there is only one object in the program W and each method takes one argument only. However, it is easy to extend our work to multiple objects and arguments.

We use the command **noret** at the end of methods that terminate but do not execute **return** E. It is automatically appended to the method code and is not supposed to be used by programmers. The command **return** E will first calculate the return value n and reduce to **fret**(n), another runtime command automatically generated during executions. We separate the evaluation of E from returning its value n to the client, to allow interference between the two steps. Note that the atomic block $\langle C \rangle$ may contain the command **return** E. In that case, $\langle C \rangle$ would also reduce to **fret**(n).

To discuss progress properties later, we introduce an auxiliary command **end**. It is a special marker that can be added at the end of a thread, but should not be used directly by programmers. Other commands are mostly standard. Clients can use **print**(E) to produce observable external events. We do not allow the

$$
\begin{array}{rrcl}
(\mathit{Expr}) & E & ::= & x \mid n \mid E + E \mid \ldots \\
(\mathit{BExp}) & B & ::= & \mathbf{true} \mid \mathbf{false} \mid E = E \mid !B \mid \ldots \\
(\mathit{Instr}) & c & ::= & x := E \mid x := [E] \mid [E] := E \mid \mathbf{print}(E) \\
 & & \mid & x := \mathbf{cons}(E, \ldots, E) \mid \mathbf{dispose}(E) \mid \ldots \\
(\mathit{Stmt}) & C & ::= & \mathbf{skip} \mid c \mid x := f(E) \mid \mathbf{return}\ E \mid \mathbf{fret}(n) \mid \mathbf{noret} \\
 & & \mid & \mathbf{end} \mid \langle C \rangle \mid C; C \mid \mathbf{if}\ (B)\ C\ \mathbf{else}\ C \mid \mathbf{while}\ (B)\{C\} \\
(\mathit{Prog}) & W & ::= & \mathbf{skip} \mid \mathbf{let}\ \Pi\ \mathbf{in}\ C \,\|\, \ldots \,\|\, C \\
(\mathit{ODecl}) & \Pi & ::= & \{f_1 \rightsquigarrow (x_1, C_1), \ldots, f_n \rightsquigarrow (x_n, C_n)\}
\end{array}
$$

Fig. 3: Syntax of the Programming Language

$$
\begin{array}{rrcl}
(\mathit{ThrdID}) & \mathtt{t} & \in & \mathit{Nat} \\
(\mathit{Mem}) & \sigma & \in & (\mathit{PVar} \cup \mathit{Nat}) \rightharpoonup \mathit{Int} \\
(\mathit{CallStk}) & \kappa & ::= & (\sigma_l, x, C) \mid \circ \\
(\mathit{ThrdPool}) & \mathcal{K} & ::= & \{\mathtt{t}_1 \rightsquigarrow \kappa_1, \ldots, \mathtt{t}_n \rightsquigarrow \kappa_n\} \\
(\mathit{PState}) & \mathcal{S} & ::= & (\sigma_c, \sigma_o, \mathcal{K}) \\
(\mathit{LState}) & s & ::= & (\sigma_c, \sigma_o, \kappa)
\end{array}
\qquad
\begin{array}{rrcl}
(\mathit{Evt}) & e & ::= & (\mathtt{t}, f, n) \mid (\mathtt{t}, \mathbf{ret}, n) \\
 & & \mid & (\mathtt{t}, \mathbf{obj}) \mid (\mathtt{t}, \mathbf{obj}, \mathbf{abort}) \\
 & & \mid & (\mathtt{t}, \mathbf{out}, n) \mid (\mathtt{t}, \mathbf{clt}) \\
 & & \mid & (\mathtt{t}, \mathbf{clt}, \mathbf{abort}) \mid (\mathtt{t}, \mathbf{term}) \\
 & & \mid & (\mathbf{spawn}, n) \\
(\mathit{ETrace}) & T & ::= & \epsilon \mid e :: T \quad \text{(co-inductive)}
\end{array}
$$

Fig. 4: States and Event Traces

object's methods to produce external events. To simplify the semantics, we also assume there are no nested method calls.

Figure 4 defines program states and event traces. We partition a global state $\mathcal{S}$ into the client memory σ_c, the object σ_o, and a thread pool $\mathcal{K}$. A client can only access the client memory σ_c, unless it calls object methods. The thread pool maps each thread ID $\mathtt{t}$ to its local call stack frame. A call stack κ could be either empty ($\circ$) when the thread is not executing a method, or a triple (σ_l, x, C), where σ_l maps the method's formal argument and local variables to their values, x is the caller's variable to receive the return value, and C is the caller's remaining code to be executed after the method returns. To give a thread-local semantics, we also define the thread local view s of the state that only includes one call stack.

Figure 5 contains selected rules of the operational semantics. To describe the operational semantics for threads, we use an execution context $\mathbf{E}$, where $\mathbf{E} ::= [\,] \mid \mathbf{E}; C$. The execution of code occurs in the hole $[\,]$. The context $\mathbf{E}[\,C\,]$ results from placing C into the hole.

We have three kinds of transitions. We write $(W, \mathcal{S}) \overset{e}{\longmapsto} (W', \mathcal{S}')$ for the top-level program transitions and $(C, s) \overset{e}{\longrightarrow}_{\mathtt{t},\Pi} (C', s')$ for the transitions of thread $\mathtt{t}$ with the object Π. We also introduce the local transition $(C, \sigma) \longrightarrow\!\!\triangleright_{\mathtt{t}} (C', \sigma')$ to describe a step inside or outside method calls of concurrent objects. It accesses only object memory and method local variables (for the case inside method calls), or only client memory (for the other case). We then lift a local transition to a thread transition that produces an event $(\mathtt{t}, \mathbf{obj})$ or $(\mathtt{t}, \mathbf{clt})$. All three transitions also include steps that lead to the error state **abort**.

We define all the generated events e in Figure 4. A method invocation event $(\mathtt{t}, f, n)$ is produced when thread $\mathtt{t}$ executes $x := f(E)$, where the argument E's value is n. A return $(\mathtt{t}, \mathbf{ret}, n)$ is produced with the return value

$$\frac{(C_i, (\sigma_c, \sigma_o, \mathcal{K}(i))) \xrightarrow{e}_{i,\Pi} (C_i', (\sigma_c', \sigma_o', \kappa'))}{(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots C_i \ldots \|\, C_n, (\sigma_c, \sigma_o, \mathcal{K})) \overset{e}{\longmapsto} (\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots C_i' \ldots \|\, C_n, (\sigma_c', \sigma_o', \mathcal{K}\{i \rightsquigarrow \kappa'\}))}$$

(a) Program Transitions

$$\frac{\Pi(f) = (y, C) \qquad [\![E]\!]_{\sigma_c} = n \qquad x \in dom(\sigma_c) \qquad \kappa = (\{y \rightsquigarrow n\}, x, \mathbf{E}[\,\mathbf{skip}\,])}{(\mathbf{E}[\,x := f(E)\,], (\sigma_c, \sigma_o, \circ)) \xrightarrow{(\mathsf{t}, f, n)}_{\mathsf{t},\Pi} (C; \mathbf{noret}, (\sigma_c, \sigma_o, \kappa))}$$

$$\frac{f \notin dom(\Pi) \quad \text{or} \quad [\![E]\!]_{\sigma_c}\ \textit{undefined} \quad \text{or} \quad x \notin dom(\sigma_c)}{(\mathbf{E}[\,x := f(E)\,], (\sigma_c, \sigma_o, \circ)) \xrightarrow{(\mathsf{t}, \mathbf{clt}, \mathbf{abort})}_{\mathsf{t},\Pi} \mathbf{abort}}$$

$$\frac{\kappa = (\sigma_l, x, C) \qquad \sigma_c' = \sigma_c\{x \rightsquigarrow n\}}{(\mathbf{fret}(n), (\sigma_c, \sigma_o, \kappa)) \xrightarrow{(\mathsf{t}, \mathbf{ret}, n)}_{\mathsf{t},\Pi} (C, (\sigma_c', \sigma_o, \circ))} \qquad \frac{}{(\mathbf{end}, s) \xrightarrow{(\mathsf{t}, \mathbf{term})}_{\mathsf{t},\Pi} (\mathbf{skip}, s)}$$

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

$$\frac{(C, \sigma_o \uplus \sigma_l) \longrightarrow_{\mathsf{t}} (C', \sigma_o' \uplus \sigma_l') \qquad dom(\sigma_l) = dom(\sigma_l')}{(C, (\sigma_c, \sigma_o, (\sigma_l, x, C_c))) \xrightarrow{(\mathsf{t}, \mathbf{obj})}_{\mathsf{t},\Pi} (C', (\sigma_c, \sigma_o', (\sigma_l', x, C_c)))}$$

$$\frac{(C, \sigma_o \uplus \sigma_l) \longrightarrow_{\mathsf{t}} \mathbf{abort}}{(C, (\sigma_c, \sigma_o, (\sigma_l, x, C_c))) \xrightarrow{(\mathsf{t}, \mathbf{obj}, \mathbf{abort})}_{\mathsf{t},\Pi} \mathbf{abort}} \qquad \frac{(C, \sigma_c) \longrightarrow_{\mathsf{t}} (C', \sigma_c')}{(C, (\sigma_c, \sigma_o, \circ)) \xrightarrow{(\mathsf{t}, \mathbf{clt})}_{\mathsf{t},\Pi} (C', (\sigma_c', \sigma_o, \circ))}$$

(b) Thread Transitions

$$\frac{[\![E]\!]_{\sigma} = n}{(\mathbf{E}[\,\mathbf{return}\ E\,], \sigma) \longrightarrow_{\mathsf{t}} (\mathbf{fret}(n), \sigma)} \qquad \frac{}{(\mathbf{noret}, \sigma) \longrightarrow_{\mathsf{t}} \mathbf{abort}}$$

$$\frac{(C, \sigma) \longrightarrow_{\mathsf{t}}^{*} (\mathbf{skip}, \sigma')}{(\mathbf{E}[\,\langle C \rangle\,], \sigma) \longrightarrow_{\mathsf{t}} (\mathbf{E}[\,\mathbf{skip}\,], \sigma')} \qquad \frac{(C, \sigma) \longrightarrow_{\mathsf{t}}^{*} (\mathbf{fret}(n), \sigma')}{(\mathbf{E}[\,\langle C \rangle\,], \sigma) \longrightarrow_{\mathsf{t}} (\mathbf{fret}(n), \sigma')} \qquad \frac{(C, \sigma) \longrightarrow_{\mathsf{t}}^{*} \mathbf{abort}}{(\mathbf{E}[\,\langle C \rangle\,], \sigma) \longrightarrow_{\mathsf{t}} \mathbf{abort}}$$

(c) Local Thread Transitions

Fig. 5: Selected Rules of Operational Semantics

n. **print**(E) generates an output $(\mathsf{t}, \mathbf{out}, n)$, and **end** generates a termination marker $(\mathsf{t}, \mathbf{term})$. Other steps generate either normal object actions $(\mathsf{t}, \mathbf{obj})$ (for steps inside method calls) or silent client actions $(\mathsf{t}, \mathbf{clt})$ (for client steps other than **print**(E)). For transitions leading to the error state **abort**, fault events are produced: $(\mathsf{t}, \mathbf{obj}, \mathbf{abort})$ by the object method code and $(\mathsf{t}, \mathbf{clt}, \mathbf{abort})$ by the client code. We also introduce an auxiliary event $(\mathbf{spawn}, n)$ to represent spawning n threads. It is automatically inserted at the beginning of a generated event trace, according to the total number of threads in the program.[3] Note that in this paper, we follow Herlihy and Wing [11] and model dynamic thread creation by simply treating each child thread as an additional thread that executes no operations before being created. Outputs and faults are observable events. We write $\mathsf{tid}(e)$ for the thread ID in the event e. The predicate $\mathsf{is\_clt}(e)$ states that the event e is either a silent client action, an output, or a client fault. We write $\mathsf{is\_inv}(e)$ and $\mathsf{is\_ret}(e)$ to denote that the event e is a method invocation and a return, respectively. The predicate $\mathsf{is\_res}(e)$ denotes a return or an object fault, and $\mathsf{is\_abt}(e)$ denotes a fault of the object or the client. Other predicates are similar and summarized below.

- $\mathsf{is\_inv}(e)$ iff there exist t, f and n such that $e = (\mathsf{t}, f, n)$;
- $\mathsf{is\_ret}(e)$ iff there exist t and n' such that $e = (\mathsf{t}, \mathbf{ret}, n')$;
- $\mathsf{is\_obj\_abt}(e)$ iff there exists t such that $e = (\mathsf{t}, \mathbf{obj}, \mathbf{abort})$;
- $\mathsf{is\_res}(e)$ iff either $\mathsf{is\_ret}(e)$ or $\mathsf{is\_obj\_abt}(e)$ holds;
- $\mathsf{is\_obj}(e)$ iff either $e = (\_, \mathbf{obj})$ or $\mathsf{is\_inv}(e)$ or $\mathsf{is\_res}(e)$ holds;
- $\mathsf{is\_clt\_abt}(e)$ iff there exists t such that $e = (\mathsf{t}, \mathbf{clt}, \mathbf{abort})$;
- $\mathsf{is\_abt}(e)$ iff either $\mathsf{is\_obj\_abt}(e)$ or $\mathsf{is\_clt\_abt}(e)$ holds;
- $\mathsf{is\_clt}(e)$ iff there exists t and n such that either $e = (\mathsf{t}, \mathbf{clt})$ or $e = (\mathsf{t}, \mathbf{out}, n)$ or $e = (\mathsf{t}, \mathbf{clt}, \mathbf{abort})$ holds.

An event trace T is a finite or infinite sequence of events. We write $T(i)$ for the i-th event of T. $\mathsf{last}(T)$ is the last event in a finite T. The trace $T(1..i)$ is the sub-trace $T(1), \ldots, T(i)$ of T, and $|T|$ is the length of T ($|T| = \omega$ if T is infinite). The trace $T|_{\mathsf{t}}$ represents the sub-trace of T consisting of all events whose thread ID is t. We generate event traces from executions in Figure 6. We write $\mathcal{T}[\![W, (\sigma_c, \sigma_o)]\!]$ for the prefix-closed set of finite traces produced by the executions of W with the initial client memory σ_c, the object σ_o, and empty call stacks for all threads. Similarly, we write $\mathcal{T}_\omega[\![W, (\sigma_c, \sigma_o)]\!]$ for the finite or infinite event traces produced by complete executions. In the definitions, we use the notation $\_ \overset{T}{\longmapsto}{}^* \_$ for zero or multiple-step program transitions the generate the trace T. Similarly, $\_ \overset{T}{\longmapsto}{}^\omega \cdot$ denotes the existence of an infinite T-labelled execution. Note that by using $\lfloor W \rfloor$, **end** is automatically appended at the end of each thread in W to explicitly mark the termination of a thread. Using $\lfloor T \rfloor_W$, we insert the spawning event $(\mathbf{spawn}, n)$ at the beginning of T, where n is the total number of threads in W. Then we could use $\mathsf{tnum}(T)$ to get the number

[3] The spawning event $(\mathbf{spawn}, n)$ is newly introduced in this TR. It helps to hide the parameter of the total number of threads in the fairness definition in the submitted version, and to formulate the alternative definitions of progress properties.

$$\mathcal{T}[\![W,(\sigma_c,\sigma_o)]\!] \stackrel{\text{def}}{=} \{\lfloor T\rfloor_W \mid \exists W',\mathcal{S}'.\ (\lfloor W\rfloor,(\sigma_c,\sigma_o,\circledcirc)) \xmapsto{T}{}^{*} (W',\mathcal{S}') \vee (\lfloor W\rfloor,(\sigma_c,\sigma_o,\circledcirc)) \xmapsto{T}{}^{*} \mathbf{abort}\}$$

$$\mathcal{T}_\omega[\![W,(\sigma_c,\sigma_o)]\!] \stackrel{\text{def}}{=} \{\lfloor T\rfloor_W \mid (\lfloor W\rfloor,(\sigma_c,\sigma_o,\circledcirc)) \xmapsto{T}{}^{\omega}\cdot \vee (\lfloor W\rfloor,(\sigma_c,\sigma_o,\circledcirc)) \xmapsto{T}{}^{*} (\mathbf{skip},\_) \vee (\lfloor W\rfloor,(\sigma_c,\sigma_o,\circledcirc)) \xmapsto{T}{}^{*} \mathbf{abort}\}$$

$$\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \| \ldots \| C_n \rfloor \stackrel{\text{def}}{=} \mathbf{let}\ \Pi\ \mathbf{in}\ (C_1;\mathbf{end}) \| \ldots \| (C_n;\mathbf{end})$$

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

$$\circledcirc \stackrel{\text{def}}{=} \{\mathsf{t}_1 \rightsquigarrow \circ,\ldots,\mathsf{t}_n \rightsquigarrow \circ\} \qquad \mathsf{div\_tids}(T) \stackrel{\text{def}}{=} \{\mathsf{t} \mid (\,|(T|_\mathsf{t})| = \omega\,)\}$$

$$\mathsf{iso}(T) \ \text{ iff } \ |T| = \omega \implies \exists \mathsf{t}, i.\ (\forall j.\ j \geq i \implies \mathsf{tid}(T(j)) = \mathsf{t})$$

$$\mathsf{fair}(T) \ \text{ iff } \ |T| = \omega \implies \forall \mathsf{t} \in [1..\mathsf{tnum}(T)].\ |(T|_\mathsf{t})| = \omega \vee \mathsf{last}(T|_\mathsf{t}) = (\mathsf{t},\mathbf{term})$$

$$\mathcal{H}[\![W,(\sigma_c,\sigma_o)]\!] \stackrel{\text{def}}{=} \{\mathsf{get\_hist}(T) \mid T \in \mathcal{T}[\![W,(\sigma_c,\sigma_o)]\!]\}$$

$$\mathcal{O}[\![W,(\sigma_c,\sigma_o)]\!] \stackrel{\text{def}}{=} \{\mathsf{get\_obsv}(T) \mid T \in \mathcal{T}[\![W,(\sigma_c,\sigma_o)]\!]\}$$

$$\mathcal{O}_{t\omega}[\![W,(\sigma_c,\sigma_o)]\!] \stackrel{\text{def}}{=} \{(\mathsf{get\_obsv}(T),\mathsf{div\_tids}(T)) \mid T \in \mathcal{T}_\omega[\![W,(\sigma_c,\sigma_o)]\!]\}$$

$$\mathcal{O}_{\omega}[\![W,(\sigma_c,\sigma_o)]\!] \stackrel{\text{def}}{=} \{\mathsf{get\_obsv}(T) \mid T \in \mathcal{T}_\omega[\![W,(\sigma_c,\sigma_o)]\!]\}$$

$$\mathcal{O}_{i\omega}[\![W,(\sigma_c,\sigma_o)]\!] \stackrel{\text{def}}{=} \{\mathsf{get\_obsv}(T) \mid T \in \mathcal{T}_\omega[\![W,(\sigma_c,\sigma_o)]\!] \wedge \mathsf{iso}(T)\}$$

$$\mathcal{O}_{f\omega}[\![W,(\sigma_c,\sigma_o)]\!] \stackrel{\text{def}}{=} \{\mathsf{get\_obsv}(T) \mid \exists n.\ T \in \mathcal{T}_\omega[\![W,(\sigma_c,\sigma_o)]\!] \wedge \mathsf{fair}(T)\}$$

Fig. 6: Generation of Event Traces

of threads in the program that generates T. Figure 6 also shows various ways to get histories and observable behaviors of a program, which we will explain later.

Linearizability and Basic Contextual Refinement Linearizability [11] is defined using histories. Histories are special event traces only consisting of method invocation, method return, and object faults.

We say a response e_2 *matches* an invocation e_1, denoted as $\mathsf{match}(e_1,e_2)$, iff they have the same thread ID.

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

A history T is *sequential*, *i.e.*, $\mathsf{seq}(T)$, iff the first event of T is an invocation, and each invocation, except possibly the last, is immediately followed by a matching response. It is inductively defined as follows.

$$\frac{}{\mathsf{seq}(\epsilon)} \qquad \frac{\mathsf{is\_inv}(e)}{\mathsf{seq}(e::\epsilon)} \qquad \frac{\mathsf{match}(e_1,e_2) \qquad \mathsf{seq}(T)}{\mathsf{seq}(e_1::e_2::T)}$$

Then T is *well-formed* iff, for all t, $T|_\mathsf{t}$ is sequential.

$$\mathsf{well\_formed}(T) \stackrel{\text{def}}{=} \forall \mathsf{t}.\ \mathsf{seq}(T|_\mathsf{t})\,.$$

T is *complete* iff it is well-formed and every invocation has a matching response. An invocation is *pending* if no matching response follows it. We write $\mathsf{pend\_inv}(T)$ for the set of pending invocations in T.

$$\mathsf{pend\_inv}(T) \stackrel{\text{def}}{=} \{e \mid \exists i.\ e{=}T(i) \wedge \mathsf{is\_inv}(e) \wedge (\forall j.\ i{<}j{\leq}|T| \Rightarrow \neg\mathsf{match}(e, T(j)))\}$$

We handle pending invocations in an incomplete history T following the standard linearizability definition [11]: We append zero or more return events to T, and drop the remaining pending invocations. Then we get a set of complete histories, which is denoted by $\mathsf{completions}(T)$. Formally, we define $\mathsf{completions}(T)$ as follows.

Definition 1 (Extensions of a history). $\mathsf{extensions}(T)$ *is a set of well-formed histories where we extend* T *by appending successful return events:*

$$\frac{\mathsf{well\_formed}(T)}{T \in \mathsf{extensions}(T)} \qquad \frac{T' \in \mathsf{extensions}(T) \qquad \mathsf{is\_ret}(e) \qquad \mathsf{well\_formed}(T'::e)}{T'::e \in \mathsf{extensions}(T)}$$

Or equivalently,

$$\mathsf{extensions}(T) \stackrel{\text{def}}{=} \{T' \mid \mathsf{well\_formed}(T') \wedge \exists T_{ok}.\ T'{=}T{::}T_{ok} \wedge \forall i.\ \mathsf{is\_ret}(T_{ok}(i))\}.$$

Definition 2 (Completions of a history). $\mathsf{truncate}(T)$ *is the maximal complete sub-history of* T*, which is inductively defined by dropping the pending invocations in* T*:*

$$\mathsf{truncate}(\epsilon) \stackrel{\text{def}}{=} \epsilon$$

$$\mathsf{truncate}(e::T) \stackrel{\text{def}}{=} \begin{cases} e::\mathsf{truncate}(T) & \textit{if } \mathsf{is\_res}(e) \textit{ or } \exists i.\ \mathsf{match}(e, T(i)) \\ \mathsf{truncate}(T) & \textit{otherwise} \end{cases}$$

Then $\mathsf{completions}(T) \stackrel{\text{def}}{=} \{\mathsf{truncate}(T') \mid T' \in \mathsf{extensions}(T)\}$*. It's a set of histories without pending invocations.*

Then we can formulate the linearizability relation between well-formed histories, which is a core notion used in the linearizability definition of an object.

Definition 3 (Linearizable Histories). $T \preceq_{\mathsf{lin}} T'$ *iff*

1. $\forall \mathsf{t}.\ T|_{\mathsf{t}} = T'|_{\mathsf{t}}$*;*
2. *there exists a bijection* $\pi : \{1, \ldots, |T|\} \to \{1, \ldots, |T'|\}$ *such that* $\forall i.\ T(i) = T'(\pi(i))$ *and* $\forall i, j.\ i < j \wedge \mathsf{is\_ret}(T(i)) \wedge \mathsf{is\_inv}(T(j)) \implies \pi(i) < \pi(j)$.

That is, T is linearizable *w.r.t.* T' if the latter is a permutation of the former, preserving the order of events in the same threads and the order of the non-overlapping method calls. Then an *object* is linearizable iff all its concurrent histories after completions are linearizable *w.r.t.* some *legal sequential* histories. We use $\Pi_A \rhd (\sigma_a, T')$ to mean that T' is a legal sequential history generated by any client using the specification Π_A with an initial abstract object σ_a.

$$\Pi_A \rhd (\sigma_a, T) \stackrel{\text{def}}{=}$$
$$\exists n, C_1, \ldots, C_n, \sigma_c.\ T \in \mathcal{H}[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a)]\!] \wedge \mathsf{seq}(T)$$

As defined in Figure 6, we use $\mathcal{H}[\![W, (\sigma_c, \sigma_a)]\!]$ to generate histories from W, where $\mathsf{get\_hist}(T)$ projects the event trace T to the sub-history.

Definition 4 (Linearizability of Objects). *The object's implementation Π is linearizable w.r.t. Π_A under a refinement mapping φ, denoted by $\Pi \preceq_\varphi \Pi_A$ iff* $\forall n, C_1, \ldots, C_n, \sigma_c, \sigma_o, \sigma_a, T.\ T \in \mathcal{H}[\![(\textbf{let}\ \Pi\ \textbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o)]\!] \wedge (\varphi(\sigma_o) = \sigma_a)$ $\Longrightarrow\ \exists T_c, T'.\ T_c \in \mathsf{completions}(T) \wedge \Pi_A \rhd (\sigma_a, T') \wedge T_c \preceq_{\mathsf{lin}} T'$

Here the mapping φ relates concrete objects to abstract ones:

$$(\mathit{RefMap})\quad \varphi \ \in\ \mathit{Mem} \rightharpoonup \mathit{AbsObj}$$

The side condition $\varphi(\sigma_o) = \theta$ in the above definition requires the initial concrete object σ_o to be a well-formed data structure representing a valid object θ.

Next we define a contextual refinement between the concrete object and its specification, which is equivalent to linearizability. Informally, this contextual refinement states that for any set of client threads, the program W has no more observable behaviors than the corresponding abstract program. Below we use $\mathcal{O}[\![W, (\sigma_c, \sigma_o)]\!]$ to represent the set of observable event traces generated during the executions of W with the initial state (σ_c, σ_o) (and empty stacks). It is defined similarly as $\mathcal{H}[\![W, (\sigma_c, \sigma_o)]\!]$ in Figure 6, but now the traces consist of observable events only (outputs, client faults or object faults).

Definition 5 (Basic Contextual Refinement). *$\Pi \sqsubseteq_\varphi \Pi_A$ iff*

$\forall n, C_1, \ldots, C_n, \sigma_c, \sigma_o, \sigma_a.\ (\varphi(\sigma_o) = \sigma_a)$
$\Longrightarrow\ \mathcal{O}[\![(\textbf{let}\ \Pi\ \textbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o)]\!] \subseteq \mathcal{O}[\![(\textbf{let}\ \Pi_A\ \textbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a)]\!]\,.$

Following Filipović *et al.* [4], we can prove that linearizability is equivalent to this contextual refinement. We give the proofs in Appendix B.1.

Theorem 6 (Basic Equivalence). $\Pi \preceq_\varphi \Pi_A \iff \Pi \sqsubseteq_\varphi \Pi_A$.

Theorem 6 allows us to use $\Pi \sqsubseteq_\varphi \Pi_A$ to identify linearizable objects. However, we cannot use it to observe progress properties of objects. For the following example, $\Pi \sqsubseteq_\varphi \Pi_A$ holds although no concrete method call of `f` could finish (we assume this object contains a method `f` only).

$\Pi(\texttt{f})$: `while(true) skip;` $\Pi_A(\texttt{f})$: `skip;` C : `print(1); f(); print(1);`

The reason is that $\Pi \sqsubseteq_\varphi \Pi_A$ considers a *prefix-closed* set of event traces at the abstract side. For the above client C, the observable behaviors of **let** Π **in** C can all be found in the prefix-closed set of behaviors produced by **let** Π_A **in** C.

4 Progress Properties and Contextual Refinements

Now we define each progress property and discuss the corresponding termination-sensitive contextual refinement. We assume that the object specification Π_A is *total*, *i.e.*, the abstract operations never block. We provide the full proofs of our equivalence results in Appendix B.

4.1 Wait-Freedom and Observing Threaded Divergence

A wait-free object guarantees that every method call can finish in a finite number of steps [7]. Below we first define wait-freedom over an event trace T.

Definition 7 (Wait-Free Event Trace).
$\mathsf{wait\text{-}free}(T)$ *iff one of the following holds:*

1. *for any i and e, if $e \in \mathsf{pend\_inv}(T(1..i))$, then one of the following holds:*
   (a) *there exists $j > i$ such that $\mathsf{match}(e, T(j))$; or*
   (b) *there exists $j > i$ such that $\forall k \geq j.\ \mathsf{tid}(T(k)) \neq \mathsf{tid}(e)$;*
2. *there exists i such that $\mathsf{is\_abt}(T(i))$ holds.*

We say an event trace T is wait-free, if every pending method call in it eventually returns (condition 1(a)) unless the thread is no longer scheduled after some program point (condition 1(b)). Besides, we assume an event trace ending with a fault is wait-free (condition 2). Remember that $\mathsf{pend\_inv}(T(1..i))$ is the set of pending invocations among the first i events of T.

An object is wait-free iff all its event traces are wait-free. Here we use $\mathcal{T}_\omega[\![W, (\sigma_c, \sigma_o)]\!]$, the set of finite or infinite traces produced by complete executions, which is defined in Figure 6.

Definition 8 (Wait-Free Object). *The object's implementation Π is wait-free under the refinement mapping φ, denoted by $\mathsf{wait\text{-}free}_\varphi(\Pi)$, iff*
$\forall n, C_1, \ldots, C_n, \sigma_c, \sigma_o, T.\ \ T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o)]\!] \wedge (\sigma_o \in \mathit{dom}(\varphi))$
$\implies \mathsf{wait\text{-}free}(T)$

We extend the basic contextual refinement to observe wait-freedom as well as linearizability. The divergence of individual threads as well as I/O events are treated as observable behaviors. In Figure 6, we define $\mathsf{div\_tids}(T)$ to collect the set of threads that diverge in the trace T. We write $\mathcal{O}_{t\omega}[\![W, (\sigma_c, \sigma_o)]\!]$ for the corresponding set of observable behaviors generated by complete executions.

Definition 9 (Contextual Refinement for Wait-Freedom).
$\Pi \sqsubseteq^{t\omega}_\varphi \Pi_A$ iff $(\forall n, C_1, \ldots, C_n, \sigma_c, \sigma_o, \sigma_a.\ (\varphi(\sigma_o) = \sigma_a) \implies$
$\mathcal{O}_{t\omega}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o)]\!] \subseteq \mathcal{O}_{t\omega}[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a)]\!]\,)$.

We can prove that a linearizable and wait-free object preserves the new contextual refinement *w.r.t.* its specification, as shown below.

Theorem 10. $\Pi \preceq_\varphi \Pi_A \wedge \mathsf{wait\text{-}free}_\varphi(\Pi) \iff \Pi \sqsubseteq^{t\omega}_\varphi \Pi_A$.

To see the intuition of this equivalence, consider the client program (2.1). Intuitively, for any execution in which the client uses the abstract operations, only the right thread diverges. The client does not produce more observable behaviors when it uses the wait-free implementation in Figure 2(a) instead. But if it uses a non–wait-free implementation (such as the one in Figure 2(b)), it is possible that both the left and right threads diverge. Therefore, a client with such an implementation produces more observable behaviors than the abstract client, breaking the contextual refinement.

4.2 Lock-Freedom and Divergence of Whole Programs

Lock-freedom is similar to wait-freedom but only guarantees that *some* thread can complete an operation in a finite number of steps [7]. We first define lock-freedom over an event trace T.

Definition 11 (Lock-Free Event Trace).
$\mathsf{lock\text{-}free}(T)$ *iff one of the following holds:*
1. *for any i, if* $\mathsf{pend\_inv}(T(1..i)) \neq \emptyset$*, then one of the following holds:*
   (a) *there exists $j > i$ such that* $\mathsf{is\_ret}(T(j))$*; or*
   (b) *there exists $j > i$ such that* $\forall k \geq j.\ \mathsf{is\_clt}(T(k))$*;*
2. *there exists i such that* $\mathsf{is\_abt}(T(i))$ *holds.*

A lock-free event trace T ensures that there always exists some pending method call that returns (condition 1(a)). When none of the threads with pending calls are scheduled, T may eventually contain client actions only (condition 1(b)). Then a lock-free object Π ensures that all its event traces are lock-free.

Definition 12 (Lock-Free Object). $\mathsf{lock\text{-}free}_\varphi(\Pi)$ *iff*
$\forall n, C_1, \ldots, C_n, \sigma_c, \sigma_o, T.\ \ T \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o)]\!] \wedge (\sigma_o \in dom(\varphi))$
$\implies \mathsf{lock\text{-}free}(T)$

In the corresponding contextual refinement, we observe the divergence of the whole client program. We use $\mathcal{O}_\omega[\![W, (\sigma_c, \sigma_o)]\!]$, as defined in Figure 6, to get the observable traces from complete executions.

Definition 13 (Contextual Refinement for Lock-Freedom).
$\Pi \sqsubseteq^\omega_\varphi \Pi_A$ iff $(\forall n, C_1, \ldots, C_n, \sigma_c, \sigma_o, \sigma_a.\ (\varphi(\sigma_o) = \sigma_a) \implies$
$\mathcal{O}_\omega[\![(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o)]\!] \subseteq \mathcal{O}_\omega[\![(\textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a)]\!]\,).$

Theorem 14. $\Pi \preceq_\varphi \Pi_A \wedge \mathsf{lock\text{-}free}_\varphi(\Pi) \iff \Pi \sqsubseteq^\omega_\varphi \Pi_A$.

To understand this equivalence, consider the client (2.1). The whole client program diverges in every execution both when it uses the lock-free object in Figure 2(b) and when it uses the abstract one. In comparison, the following client must terminate and print out both 1 and 2 in every execution at both levels.

```
inc(); print(1);    ||    dec(); print(2);
```
(4.1)

4.3 Obstruction-Freedom and Isolating Executions

Obstruction-freedom guarantees progress for any thread that eventually executes in isolation [8]. We define obstruction-free event traces as follows.

Definition 15 (Obstruction-Free Event Trace).
$\mathsf{obstruction\text{-}free}(T)$ *iff one of the following holds:*
1. *for any i and e, if* $e \in \mathsf{pend\_inv}(T(1..i))$*, then one of the following holds:*
   (a) *there exists $j > i$ such that* $\mathsf{match}(e, T(j))$*; or*
   (b) $\forall j > i.\ \exists k.\ k \geq j \wedge \mathsf{tid}(T(k)) \neq \mathsf{tid}(e)$*;*
2. *there exists i such that* $\mathsf{is\_abt}(T(i))$ *holds.*

An obstruction-free event trace guarantees that every pending method call eventually finishes unless the thread is not running in isolation for a sufficiently long time. In other words, either the thread returns (condition 1(a)), or it is infinitely often interrupted by other threads (condition 1(b)). An object Π is obstruction-free, denoted as $\mathsf{obstruction\text{-}free}_\varphi(\Pi)$, iff all its event traces are obstruction-free.

Definition 16 (Obstruction-Free Object). $\mathsf{obstruction\text{-}free}_\varphi(\Pi)$ *iff*
$\forall n, C_1, \ldots, C_n, \sigma_c, \sigma_o, T.\ \ T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o)]\!] \wedge (\sigma_o \in dom(\varphi))$
$\Longrightarrow \mathsf{obstruction\text{-}free}(T)$

Obstruction-freedom ensures progress for those *isolating* executions where eventually only one thread is running. In the corresponding termination-sensitive contextual refinement, we are also interested in isolating executions only. We define $\mathsf{iso}(T)$ in Figure 6 to mean T is isolating, and use $\mathcal{O}_{i\omega}[\![W, (\sigma_c, \sigma_o)]\!]$ to get the observable behaviors for the generated event traces T which satisfy $\mathsf{iso}(T)$.

Definition 17 (Contextual Refinement for Obstruction-Freedom).
$\Pi \sqsubseteq^{i\omega}_\varphi \Pi_A$ iff $(\forall n, C_1, \ldots, C_n, \sigma_c, \sigma_o, \sigma_a.\ (\varphi(\sigma_o) = \sigma_a) \Longrightarrow$
$\mathcal{O}_{i\omega}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o)]\!] \subseteq \mathcal{O}_{\omega}[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a)]\!]\ ).$

Theorem 18. $\Pi \preceq_\varphi \Pi_A \wedge \mathsf{obstruction\text{-}free}_\varphi(\Pi) \iff \Pi \sqsubseteq^{i\omega}_\varphi \Pi_A$.

To understand the above equivalence, consider the client (4.1). When using the obstruction-free object in Figure 2(c), the client may diverge and nothing is printed out. But it always terminates and prints out both 1 and 2 for isolating executions. Then the observable behaviors taken by $\mathcal{O}_{i\omega}$ at the concrete side do not include divergence and are thus a subset of those at the abstract level.

4.4 Deadlock-Freedom and Fairness

As we explained in Section 2, deadlock-freedom and starvation-freedom assume fair scheduling. We define $\mathsf{fair}(T)$ in Figure 6. It requires that if T is infinite then every unterminated thread's execution is infinite.[4] Note that a trace ending with a fault is fair, too. An object Π is deadlock-free, denoted as $\mathsf{deadlock\text{-}free}_\varphi(\Pi)$, if in every fair execution, the system as a whole always makes progress (*i.e.*, there always exists some pending method call that finishes). Its formulation is very similar to the lock-freedom definitions (Definitions 11 and 12).[5]

[4] In the submitted version, we use $\mathsf{fair}_n(T)$ which takes the number of threads n as a parameter. Here we could know n from the spawning event at the beginning of T, as defined by $\mathsf{tnum}(T)$ in Figure 6.

[5] The definitions are slightly different from the submitted version, where we rule out unfair event traces at the generation so that we only need to consider fair ones in the definition of deadlock-free traces. Here we think an unfair event trace is deadlock-free, and an object is deadlock-free iff *all* its event traces are deadlock-free. We can make this change because we hide the number of threads from $\mathsf{fair}(T)$ here. Based on this change, we can unify the formulations of progress properties (see Section 4.6).

Definition 19 (Deadlock-Free Event Trace).
$\mathsf{deadlock\text{-}free}(T)$ *iff one of the following holds:*

1. *for any* i*, if* $\mathsf{pend\_inv}(T(1..i)) \neq \emptyset$*, there exists* $j > i$ *such that* $\mathsf{is\_ret}(T(j))$*; or*
2. $\mathsf{fair}(T)$ *does not hold; or*
3. *there exists* i *such that* $\mathsf{is\_abt}(T(i))$ *holds.*

Definition 20 (Deadlock-Free Object). $\mathsf{deadlock\text{-}free}_\varphi(\Pi)$ *iff*
$\forall n, C_1, \ldots, C_n, \sigma_c, \sigma_o, T.\ \ T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o)]\!] \wedge (\sigma_o \in dom(\varphi))$
$\Longrightarrow \mathsf{deadlock\text{-}free}(T)$

The corresponding contextual refinement considers only fair executions at the concrete side. As defined in Figure 6, $\mathcal{O}_{f\omega}[\![W, (\sigma_c, \sigma_o)]\!]$ picks observable behaviors for fair event traces only.

Definition 21 (Contextual Refinement for Deadlock-Freedom).
$\Pi \sqsubseteq_\varphi^{f\omega} \Pi_A$ iff $(\forall n, C_1, \ldots, C_n, \sigma_c, \sigma_o, \sigma_a.\ (\varphi(\sigma_o) = \sigma_a) \Longrightarrow$
$\mathcal{O}_{f\omega}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o)]\!] \subseteq \mathcal{O}_\omega[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a)]\!]\,).$

Theorem 22. $\Pi \preceq_\varphi \Pi_A \wedge \mathsf{deadlock\text{-}free}_\varphi(\Pi) \iff \Pi \sqsubseteq_\varphi^{f\omega} \Pi_A$.

Consider the client (4.1) to understand the above equivalence. When the client uses the deadlock-free object in Figure 2(d), it may diverge and print out nothing. For instance, the left thread acquires the test-and-set lock and gets suspended, then the right thread would keep requesting the lock forever. This undesirable behavior is caused by unfair scheduling. Thus we rule out unfair executions at the implementation level in the contextual refinement (Definition 21).

4.5 Starvation-Freedom and Fairness at Both Levels

Starvation-freedom guarantees that in fair executions, every method call eventually finishes. We formulate $\mathsf{starvation\text{-}free}_\varphi(\Pi)$ similarly as the wait-freedom definitions (Definitions 7 and 8).[6] The related contextual refinement must require both the concrete and abstract executions to be fair.

Definition 23 (Starvation-Free Event Trace).
$\mathsf{starvation\text{-}free}(T)$ *iff one of the following holds:*

1. *for any* i *and* e*, if* $e \in \mathsf{pend\_inv}(T(1..i))$*, then there exists* $j > i$ *such that* $\mathsf{match}(e, T(j))$*; or*
2. $\mathsf{fair}(T)$ *does not hold; or*
3. *there exists* i *such that* $\mathsf{is\_abt}(T(i))$ *holds.*

Definition 24 (Starvation-Free Object). $\mathsf{starvation\text{-}free}_\varphi(\Pi)$ *iff*
$\forall n, C_1, \ldots, C_n, \sigma_c, \sigma_o, T.\ \ T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o)]\!] \wedge (\sigma_o \in dom(\varphi))$
$\Longrightarrow \mathsf{starvation\text{-}free}(T)$

[6] Like the deadlock-freedom definitions, the starvation-freedom definitions are slightly different from the submitted version. Here an unfair event trace is starvation-free, and an object is starvation-free iff *all* its event traces are starvation-free.

$$\mathsf{prog\text{-}t}(T) \;\text{ iff }\; \forall i, e.\ e \in \mathsf{pend\_inv}(T(1..i)) \implies \exists j.\ j > i \wedge \mathsf{match}(e, T(j))$$
$$\mathsf{prog\text{-}s}(T) \;\text{ iff }\; \forall i, e.\ e \in \mathsf{pend\_inv}(T(1..i)) \implies \exists j.\ j > i \wedge \mathsf{is\_ret}(T(j))$$
$$\mathsf{non\text{-}sched}(T) \;\text{ iff }\; \forall e.\ e \in \mathsf{pend\_inv}(T) \implies \exists i.\ \forall j \geq i.\ \mathsf{tid}(T(j)) \neq \mathsf{tid}(e)$$
$$\mathsf{abt}(T) \;\text{ iff }\; \exists i.\ \mathsf{is\_abt}(T(i))$$

$$\mathsf{wait\text{-}free} \iff \mathsf{prog\text{-}t} \vee \mathsf{non\text{-}sched} \vee \mathsf{abt} \iff \mathsf{non\text{-}sched} \vee \mathsf{abt}$$
$$\mathsf{lock\text{-}free} \iff \mathsf{prog\text{-}s} \vee \mathsf{non\text{-}sched} \vee \mathsf{abt} \iff \mathsf{wait\text{-}free} \vee \mathsf{prog\text{-}s}$$
$$\mathsf{obstruction\text{-}free} \iff \mathsf{prog\text{-}t} \vee \mathsf{non\text{-}sched} \vee \neg\mathsf{iso} \vee \mathsf{abt} \iff \mathsf{lock\text{-}free} \vee \neg\mathsf{iso}$$
$$\mathsf{deadlock\text{-}free} \iff \mathsf{prog\text{-}s} \vee \neg\mathsf{fair} \vee \mathsf{abt} \iff \mathsf{lock\text{-}free} \vee \neg\mathsf{fair}$$
$$\mathsf{starvation\text{-}free} \iff \mathsf{prog\text{-}t} \vee \neg\mathsf{fair} \vee \mathsf{abt} \iff \mathsf{wait\text{-}free} \vee \neg\mathsf{fair}$$

Fig. 7: Alternative Formulations of Progress Properties

Definition 25 (Contextual Refinement for Starvation-Freedom).
$\Pi \sqsubseteq_{\varphi}^{ff\omega} \Pi_A$ iff $(\forall n, C_1, \ldots, C_n, \sigma_c, \sigma_o, \sigma_a.\ (\varphi(\sigma_o) = \sigma_a) \implies$
$\mathcal{O}_{f\omega}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o)]\!] \subseteq \mathcal{O}_{f\omega}[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a)]\!]\ ).$

Theorem 26. $\Pi \preceq_{\varphi} \Pi_A \wedge \mathsf{starvation\text{-}free}_{\varphi}(\Pi) \iff \Pi \sqsubseteq_{\varphi}^{ff\omega} \Pi_A$.

The fairness restriction at the abstract level of the contextual refinement allows us to distinguish starvation-free objects from deadlock-free objects. Consider the following client program.

```
inc(); print(1);    ||    while(true) inc();          (4.2)
```

Under fair scheduling, we know it *must* print out 1 when using the starvation-free object in Figure 2(e); and it may or may not print 1 when using the deadlock-free object of Figure 2(d). Correspondingly at the abstract level, the client program may not print out 1 in unfair executions.

4.6 Alternative Formulations of Progress Properties

We give simpler and more structural definitions of the five progress properties in Figure 7.

We first define some properties over event traces T. $\mathsf{prog\text{-}t}(T)$ means that every method call in T eventually finishes. $\mathsf{prog\text{-}s}(T)$ means that there always exist a pending method call that completes. $\mathsf{non\text{-}sched}(T)$ represents the case caused by a "bad" scheduler. It says, every pending thread in T must be no longer scheduled after some program point. The boundary case $\mathsf{abt}(T)$ says that T ends with a fault.

The bottom half of Figure 7 gives alternative definitions of the five progress properties over event traces T. For example, an event trace T is wait-free, iff it satisfies prog-t, non-sched or abt. Since $\mathsf{prog\text{-}t}(T)$ also means there is no pending method call in the whole T, it actually implies $\mathsf{non\text{-}sched}(T)$. We can further reduce $\mathsf{wait\text{-}free}(T)$ to $\mathsf{non\text{-}sched}(T) \vee \mathsf{abt}(T)$. To simplify the presentation, we omit the parameter T in the formulas at the bottom half of the figure. We give their proofs in Appendix B.2.

Then for a progress property P, we can define that an object Π satisfies P iff all its event traces satisfy P.

$P_\varphi(\Pi)$ iff
$\forall n, C_1, \ldots, C_n, \sigma_c, \sigma_o, T.\ \ T \in \mathcal{T}_\omega[\![(\textbf{let}\ \Pi\ \textbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o)]\!] \wedge (\sigma_o \in dom(\varphi))$
$\Longrightarrow\ P(T)$

The alternative formulations in Figure 7 shows a constructive way to define progress properties. We pick a main progress condition prog-t or prog-s, and choose an appropriate scheduler (fair, iso or ¬non-sched, but there might be more). We take the disjunction of these conditions, with the boundary case abt. Using this approach, we could define more progress properties.

Moreover, we could clearly see the relationships between progress properties from Figure 7. The relationships form the lattice in Figure 1. Below we define sequential termination as the bottom element to close the lattice.

4.7 Sequential Termination and the Relationship Lattice

Besides the above five progress properties for concurrent objects, we also define a progress property in the sequential setting, as shown below.

Definition 27 (Sequentially Terminating Object). $\mathsf{seq\text{-}term}_\varphi(\Pi)$ *iff*

$$\forall C_1, \sigma_c, \sigma_o, T.\ \ T \in \mathcal{T}_\omega[\![(\textbf{let}\ \Pi\ \textbf{in}\ C_1), (\sigma_c, \sigma_o)]\!] \wedge (\sigma_o \in dom(\varphi))$$
$$\Longrightarrow \mathsf{abt}(T) \vee \mathsf{prog\text{-}t}(T).$$

It says, every method call must finish when it is executed sequentially.[7] We can show that $\mathsf{seq\text{-}term}_\varphi(\Pi)$ is implied by a termination-sensitive contextual refinement with sequential contexts, and a linearizable and sequentially terminating object preserves this contextual refinement.

Definition 28 (Sequential Contextual Refinement). $\Pi \sqsubseteq^{1\omega}_\varphi \Pi_A$ *iff*

$$\forall C_1, \sigma_c, \sigma_o, \sigma_a.\ (\varphi(\sigma_o) = \sigma_a)$$
$$\Longrightarrow \mathcal{O}_\omega[\![(\textbf{let}\ \Pi\ \textbf{in}\ C_1), (\sigma_c, \sigma_o)]\!] \subseteq \mathcal{O}_\omega[\![(\textbf{let}\ \Pi_A\ \textbf{in}\ C_1), (\sigma_c, \sigma_a)]\!]\,.$$

Theorem 29. *1.* $\Pi \sqsubseteq^{1\omega}_\varphi \Pi_A \Longrightarrow \mathsf{seq\text{-}term}_\varphi(\Pi)$ *;*
2. $\Pi \preceq_\varphi \Pi_A \wedge \mathsf{seq\text{-}term}_\varphi(\Pi) \Longrightarrow \Pi \sqsubseteq^{1\omega}_\varphi \Pi_A$ *.*

Intuitively, $\mathsf{seq\text{-}term}_\varphi(\Pi)$ is weaker than all the five concurrent progress properties. We can formally prove all the implications in the lattice of Figure 1, showing that the widely believed presumptions about the progress properties are all true (see Appendix B.2).

[7] The submitted version uses "starvation-free(T)" to formulate sequential termination, which is equivalent to $\mathsf{abt}(T) \vee \mathsf{prog\text{-}t}(T)$ in this TR.

5 Related Work and Conclusion

There is a large body of work discussing the five progress properties and the contextual refinements individually. Our work in contrast studies their relationships, which have not been considered much before.

As we mentioned in Section 1, Gotsman and Yang [6] show that lock-freedom with linearizability implies termination-sensitive contextual refinement. Here we also prove the other direction that allows us to verify lock-freedom using proof methods for contextual refinement. Besides, we generalize this equivalence for lock-freedom to other progress properties and propose a unified framework to systematically study the relationships with various contextual refinements.

Herlihy and Shavit [9] informally discuss all five progress properties. Our formulation mostly follows their explanation but makes two important contributions. First, we turn their natural language explanation into formal definitions and close the gap between program semantics and their history-based interpretations. Second, we noticed that their obstruction-freedom is inappropriate for some examples (see Appendix A), and propose a different definition that is closer to the usual informal formulations [10]. Besides, for the unified framework, they focus on the scheduling assumptions made by the progress properties, while we mainly consider the effects on client behaviors and relate the progress properties to contextual refinements.

Other formalizations of progress properties usually rely on temporal logics. For example, Petrank *et al.* [14] use linear temporal logic to formalize the three non-blocking properties. Dongol [3] formalizes all the five progress properties using a logic capable of proving temporal properties. Those formulations may make it easier to perform model checking on whole programs (*e.g.*, Petrank *et al.* [14] also build a tool to model check lock-freedom with a bound on the time for progress). Here we follow Herlihy and Shavit [9] and choose more operational formulations to fit into the framework with contextual refinements.

Conclusion. We have introduced a contextual refinement framework to unify various progress properties. For linearizable objects, each progress property is equivalent to a specific termination-sensitive contextual refinement, as summarized in Table 1. The framework allows us to verify safety and liveness properties of client programs at a high abstraction level by replacing concrete method implementations with abstract operations. It also enables us to borrow ideas from existing proof methods for contextual refinements to verify linearizability and a progress property together.

References

1. Aspnes, J., Herlihy, M.: Wait-free data structures in the asynchronous PRAM model. In: SPAA'90
2. Birkedal, L., Sieczkowski, F., Thamsborg, J.: A concurrent logical relation. In: CSL'12

3. Dongol, B.: Formalising progress properties of non-blocking programs. In: ICFEM'06
4. Filipovic, I., O'Hearn, P., Rinetzky, N., Yang, H.: Abstraction for concurrent objects. Theor. Comput. Sci. (2010)
5. Gotsman, A., Yang, H.: Linearizability with ownership transfer. In: CONCUR'12
6. Gotsman, A., Yang, H.: Liveness-preserving atomicity abstraction. In: ICALP'11
7. Herlihy, M.: Wait-free synchronization. ACM Trans. Program. Lang. Syst. (1991)
8. Herlihy, M., Luchangco, V., Moir, M.: Obstruction-free synchronization: Double-ended queues as an example. In: ICDCS'03
9. Herlihy, M., Shavit, N.: On the nature of progress. In: OPODIS'11
10. Herlihy, M., Shavit, N.: The Art of Multiprocessor Programming. Morgan Kaufmann (Apr 2008)
11. Herlihy, M., Wing, J.: Linearizability: a correctness condition for concurrent objects. ACM Trans. Program. Lang. Syst. 12(3), 463–492 (1990)
12. Liang, H., Feng, X.: Modular verification of linearizability with non-fixed linearization points. In: PLDI'13
13. Liang, H., Feng, X., Fu, M.: A rely-guarantee-based simulation for verifying concurrent program transformations. In: POPL'12
14. Petrank, E., Musuvathi, M., Steensgaard, B.: Progress guarantee for parallel programs via bounded lock-freedom. In: PLDI'11

A Comparisons with Herlihy and Shavit's Obstruction-Freedom

Herlihy and Shavit [9] define obstruction-freedom using the notion of *uniformly isolating* executions. A trace is uniformly isolating, if "for every $k > 0$, any thread that takes an infinite number of steps has an interval where it takes at least k concrete contiguous steps" [9]. Then, their obstruction-free object guarantees wait-freedom for every uniformly isolating execution. They also propose a new progress property, clash-freedom, which guarantees lock-freedom for uniformly-isolating executions.

Below we give an example showing that their definition is inconsistent with the common intuition of obstruction-freedom.

Example 30. The object implementation uses three shared variables: `x`, `a` and `b`. It provides two methods `f` and `g`:

```
f() {                          g() {
  while (a <= x <= b) {          while (a <= x <= b) {
    x++;                           x--;
    a--;                           b++;
  }                              }
}                              }
```

We can see that, if `f()` or `g()` is eventually executed in isolation (*i.e.*, we suspend all but one threads), it must returns. Thus intuitively this object should be obstruction-free. It also satisfies our formulation (Definitions 15 and 16).

However, we could construct an execution which is uniformly isolating but is not lock-free or wait-free. Consider the client program `f()` ‖ `g()`. It has an

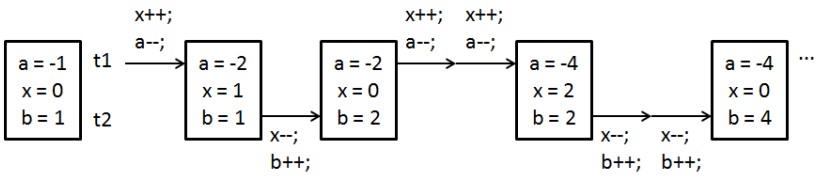

Fig. 8: Execution of `f()` ‖ `g()` in Example 30

execution shown in Figure 8. Starting from `x` = 0, `a` = −1 and `b` = 1, we alternatively let each thread execute more and more iterations. Then for any k, we could always find an interval of k iterations for each thread in this execution. Thus the execution is uniformly isolating. But neither method call finishes. This execution is not lock-free nor wait-free. Thus the object does not satisfy Herlihy and Shavit's obstruction-freedom or clash-freedom definitions.

B Proofs

In the following proofs, we make the call stacks explicit in the generation of event traces. For example, we use $\mathcal{H}[\![W, (\sigma_c, \sigma_o, \circledcirc)]\!]$ instead of $\mathcal{H}[\![W, (\sigma_c, \sigma_o)]\!]$. We generalize the definitions to allow nonempty call stacks in the initial state, *e.g.*, we can use $\mathcal{H}[\![W, (\sigma_c, \sigma_o, \mathcal{K})]\!]$.

B.1 Proofs of Theorem 6

To prove the theorem, we utilize the most general client (MGC). Let's assume $dom(\Pi) = \{f_1, \ldots, f_m\}$. We could use the expression **rand**() to get a random (nondeterministic) integer, and **rand**(m) to get a random integer $r \in [1..m]$. Then, for any n, MGC_n is defined as follows:

$$\mathsf{MGT} \stackrel{\text{def}}{=} \textbf{while } (\textbf{true})\{\ f_{\textbf{rand}(m)}(\textbf{rand}());\ \}$$
$$\mathsf{MGC}_n \stackrel{\text{def}}{=} \|_{i\in[1..n]}\, \mathsf{MGT}$$

Here each thread keeps calling a random method with a random argument. We also define another kind of "most general clients" which print out arguments and return values for method calls:

$$\begin{aligned}\mathsf{MGTp}_{\mathtt{t}} \stackrel{\text{def}}{=}\ & \textbf{while } (\textbf{true})\{\\ & \quad x_{\mathtt{t}} := \textbf{rand}();\ y_{\mathtt{t}} := \textbf{rand}(m);\ \textbf{print}(y_{\mathtt{t}}, x_{\mathtt{t}});\\ & \quad z_{\mathtt{t}} := f_{y_{\mathtt{t}}}(x_{\mathtt{t}});\ \textbf{print}(z_{\mathtt{t}});\\ & \}\end{aligned}$$
$$\mathsf{MGCp}_n \stackrel{\text{def}}{=} \|_{i\in[1..n]}\, \mathsf{MGTp}_i$$

Here $x_{\mathtt{t}}$, $y_{\mathtt{t}}$ and $z_{\mathtt{t}}$ are all local variables for thread `t`. Below we define the MGC versions of "linearizability" and refinements, and prove they are related to the standard definitions of linearizability and contextual refinement.

Definition 31. $\Pi \preceq_{\varphi}^{\mathsf{MGC}} \Pi_A$ *iff*

$$\begin{array}{l}\forall n, \sigma_o, \sigma_a, T.\ \ T \in \mathcal{H}[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \wedge (\varphi(\sigma_o) = \sigma_a) \\ \Longrightarrow\ \exists T_c, T_a.\ T_c \in \mathsf{completions}(T) \wedge \Pi_A \triangleright_n^{\mathsf{MGC}} (\sigma_a, T_a) \wedge T_c \preceq_{\mathsf{lin}} T_a\end{array}$$

where

$$\Pi_A \triangleright_n^{\mathsf{MGC}} (\sigma_a, T) \ \stackrel{\text{def}}{=}\ T \in \mathcal{H}[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!] \wedge \mathsf{seq}(T)\,.$$

$\Pi \subseteq_{\varphi} \Pi_A$ *iff*

$$\begin{array}{l}\forall n, \sigma_o, \sigma_a.\ (\varphi(\sigma_o) = \sigma_a) \\ \Longrightarrow\ \mathcal{H}[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{H}[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!]\,.\end{array}$$

The following lemma shows that every history of an object Π could be generated by the MGC.

Lemma 32 (MGC is the Most General). *For any* n, $C_1, \ldots, C_n$, σ_c, σ_o *and* σ_a, $\mathcal{H}[\![(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{H}[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!]$.

Proof. We define the simulation relation $\precsim_{\mathsf{MGC}}$ between a program and a MGC in Figure 9(a), and prove the following (B.1) by case analysis and the operational semantics:

For any W_1, $\mathcal{S}_1$, W_2, $\mathcal{S}_2$ and e_1, if $(W_1, \mathcal{S}_1) \precsim_{\mathsf{MGC}} (W_2, \mathcal{S}_2)$, then

(1) if $(W_1, \mathcal{S}_1) \stackrel{e_1}{\longmapsto} \textbf{abort}$ and $\mathsf{is\_obj\_abt}(e_1)$, then
there exists T_2 such that $(W_2, \mathcal{S}_2) \stackrel{T_2}{\longmapsto}{}^* \textbf{abort}$ and
$e_1 = \mathsf{get\_hist}(T_2)$;
(2) if $(W_1, \mathcal{S}_1) \stackrel{e_1}{\longmapsto} (W_1', \mathcal{S}_1')$, then
there exist T_2, W_2' and $\mathcal{S}_2'$ such that $(W_2, \mathcal{S}_2) \stackrel{T_2}{\longmapsto}{}^* (W_2', \mathcal{S}_2')$,
$\mathsf{get\_hist}(e_1) = \mathsf{get\_hist}(T_2)$ and $(W_1', \mathcal{S}_1') \precsim_{\mathsf{MGC}} (W_2', \mathcal{S}_2')$.

(B.1)

With (B.1), we can prove the following by induction over the number of steps generating the event trace of $\mathcal{H}[\![W_1, \mathcal{S}_1]\!]$.

$$\text{If } (\lfloor W_1 \rfloor, \mathcal{S}_1) \precsim_{\mathsf{MGC}} (\lfloor W_2 \rfloor, \mathcal{S}_2), \text{ then } \mathcal{H}[\![W_1, \mathcal{S}_1]\!] \subseteq \mathcal{H}[\![W_2, \mathcal{S}_2]\!].$$

Then, since

$$(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \precsim_{\mathsf{MGC}} (\lfloor \textbf{let } \Pi \textbf{ in } \mathsf{MGC}_n \rfloor, (\emptyset, \sigma_o, \circledcirc))\,,$$

we are done. □

For linearizability, the MGC-version is equivalent to the original definition.

Lemma 33. $\Pi \preceq_{\varphi} \Pi_A \iff \Pi \preceq_{\varphi}^{\mathsf{MGC}} \Pi_A$.

Proof. 1. $\Pi \preceq_{\varphi} \Pi_A \implies \Pi \preceq_{\varphi}^{\mathsf{MGC}} \Pi_A$:
For any n, σ_o, σ_a and T such that $T \in \mathcal{H}[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!]$ and $\varphi(\sigma_o) = \sigma_a$, from $\Pi \preceq_{\varphi} \Pi_A$, we know there exist T_c and T_a such that

$$(\textbf{let } \Pi \textbf{ in } C_1 \parallel \ldots \parallel C_n, (\sigma_c, \sigma_o, \{1 \rightsquigarrow \kappa_1, \ldots, n \rightsquigarrow \kappa_n\}))$$
$$\precsim_{\mathsf{MGC}} (\textbf{let } \Pi \textbf{ in } C_1' \parallel \ldots \parallel C_n', (\emptyset, \sigma_o, \{1 \rightsquigarrow \kappa_1', \ldots, n \rightsquigarrow \kappa_n'\}))$$
$$\text{where } \forall i.\ (C_i, \kappa_i) \precsim_{\mathsf{MGC}} (C_i', \kappa_i')$$

$$(C, \circ) \precsim_{\mathsf{MGC}} (\mathsf{MGT}; \textbf{end}, \circ) \qquad (C, (\sigma_l, x, C')) \precsim_{\mathsf{MGC}} (C, (\sigma_l, \cdot, (\textbf{skip}; \mathsf{MGT}; \textbf{end})))$$

(a) Program is Simulated by MGC

$$(\textbf{let } \Pi \textbf{ in } C_1 \parallel \ldots \parallel C_n, (\sigma_c, \sigma_o, \{1 \rightsquigarrow \kappa_1, \ldots, n \rightsquigarrow \kappa_n\}))$$
$$\precsim_{\mathsf{MGCp}} (\textbf{let } \Pi \textbf{ in } C_1' \parallel \ldots \parallel C_n', (\sigma_c', \sigma_o, \{1 \rightsquigarrow \kappa_1', \ldots, n \rightsquigarrow \kappa_n'\}))$$
$$\text{where } \forall i.\ (C_i, \kappa_i) \precsim^{i}_{\mathsf{MGCp}} (C_i', \kappa_i') \text{ and } \sigma_c' = \{x_\mathsf{t} \rightsquigarrow \_, y_\mathsf{t} \rightsquigarrow \_, z_\mathsf{t} \rightsquigarrow \_ \mid 1 \le \mathsf{t} \le n\}$$

$$(C, \circ) \precsim^{\mathsf{t}}_{\mathsf{MGCp}} (\mathsf{MGTp}_\mathsf{t}; \textbf{end}, \circ)$$

$$(C, (\sigma_l, \cdot, C')) \precsim^{\mathsf{t}}_{\mathsf{MGCp}} (C, (\sigma_l, z_\mathsf{t}, (\textbf{skip}; \textbf{print}(z_\mathsf{t}); \mathsf{MGTp}_\mathsf{t}; \textbf{end})))$$

(b) Program is Simulated by MGCp

$$(\textbf{let } \Pi \textbf{ in } C_1 \parallel \ldots \parallel C_n, (\sigma_c, \sigma_o, \{1 \rightsquigarrow \kappa_1, \ldots, n \rightsquigarrow \kappa_n\}))$$
$$\precsim_{\mathsf{MGCp}^-} (\textbf{let } \Pi \textbf{ in } C_1' \parallel \ldots \parallel C_n', (\emptyset, \sigma_o, \{1 \rightsquigarrow \kappa_1', \ldots, n \rightsquigarrow \kappa_n'\}))$$
$$\text{where } \forall i.\ (C_i, \sigma_c, \kappa_i) \precsim^{i,\Pi}_{\mathsf{MGCp}} (C_i', \kappa_i') \text{ and } \sigma_c = \{x_\mathsf{t} \rightsquigarrow \_, y_\mathsf{t} \rightsquigarrow \_, z_\mathsf{t} \rightsquigarrow \_ \mid 1 \le \mathsf{t} \le n\}$$

$$(C, \sigma_c, \circ) \precsim^{\mathsf{t},\Pi}_{\mathsf{MGCp}^-} \begin{cases} (C_o, (\{x \rightsquigarrow n\}, \cdot, (\textbf{skip}; \mathsf{MGT}; \textbf{end})) \\ \qquad \text{if } (C = \mathbf{E}[\, z_\mathsf{t} := f_{y_\mathsf{t}}(x_\mathsf{t})\,] \vee C = \mathbf{E}[\,\textbf{skip}; z_\mathsf{t} := f_{y_\mathsf{t}}(x_\mathsf{t})\,]) \\ \qquad\quad \wedge \sigma_c(x_\mathsf{t}) = n \wedge \sigma_c(y_\mathsf{t}) = i \wedge \Pi(f_i) = (x, C_o) \\ (\textbf{fret}(n'), (\_, \cdot, (\textbf{skip}; \mathsf{MGT}; \textbf{end})) \\ \qquad \text{if } (C = \mathbf{E}[\,\textbf{print}(z_\mathsf{t})\,] \vee C = \mathbf{E}[\,\textbf{skip}; \textbf{print}(z_\mathsf{t})\,]) \\ \qquad\quad \wedge \sigma_c(z_\mathsf{t}) = n' \\ (\mathsf{MGT}; \textbf{end}, \circ) \quad \text{otherwise} \end{cases}$$

$$(C, \sigma_c, (\sigma_l, z_\mathsf{t}, C')) \precsim^{\mathsf{t},\Pi}_{\mathsf{MGCp}^-} (C, (\sigma_l, \cdot, (\textbf{skip}; \mathsf{MGT}; \textbf{end})))$$

(c) MGCp is Simulated by MGC

$$(\textbf{let } \Pi \textbf{ in } C_1 \parallel \ldots \parallel C_n, (\sigma_c, \sigma_o, \{1 \rightsquigarrow \kappa_1, \ldots, n \rightsquigarrow \kappa_n\}))$$
$$\precsim (\textbf{let } \Pi_A \textbf{ in } C_1' \parallel \ldots \parallel C_n', (\emptyset, \sigma_o', \{1 \rightsquigarrow \kappa_1', \ldots, n \rightsquigarrow \kappa_n'\});$$
$$\quad \textbf{let } \Pi_A \textbf{ in } C_1'' \parallel \ldots \parallel C_n'', (\sigma_c, \sigma_o', \{1 \rightsquigarrow \kappa_1'', \ldots, n \rightsquigarrow \kappa_n''\}))$$
$$\text{where } \forall i.\ (C_i, \kappa_i) \precsim (C_i', \kappa_i'; C_i'', \kappa_i'')$$
$$\text{and } \mathcal{H}[\![\textbf{let } \Pi \textbf{ in } C_1 \parallel \ldots \parallel C_n, (\sigma_c, \sigma_o, \{1 \rightsquigarrow \kappa_1, \ldots, n \rightsquigarrow \kappa_n\})]\!]$$
$$\subseteq \mathcal{H}[\![\textbf{let } \Pi_A \textbf{ in } C_1' \parallel \ldots \parallel C_n', (\emptyset, \sigma_o', \{1 \rightsquigarrow \kappa_1', \ldots, n \rightsquigarrow \kappa_n'\})]\!]$$

$$(C, \circ) \precsim (C', \circ; C, \circ) \qquad (C, (\sigma_l, x, C_c)) \precsim (C', (\sigma_l', x', C_c'); C', (\sigma_l', x, C_c))$$

(d) Concrete Program is Simulated by Abstract MGC and Abstract Program

Fig. 9: Simulations between Programs and MGC

$$T_c \in \mathsf{completions}(T) \wedge \mathit{\Pi}_A \rhd (\sigma_a, T_a) \wedge T_c \preceq_{\mathsf{lin}} T_a \,.$$

We only need to show that

$$\mathit{\Pi}_A \rhd (\sigma_a, T_a) \implies \mathit{\Pi}_A \rhd_n^{\mathsf{MGC}} (\sigma_a, T_a) \,.$$

First we know $\forall i.\ \mathsf{tid}(T_a(i)) \in [1..n]$. Second, from $\mathit{\Pi}_A \rhd (\sigma_a, T_a)$, we know there exist n', C_1, …, $C_{n'}$ and σ_c such that $\mathsf{seq}(T_a)$ and

$$T_a \in \mathcal{H}[\![(\mathbf{let}\ \mathit{\Pi}_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_{n'}), (\sigma_c, \sigma_a, \circledcirc)]\!] \,.$$

If $n' \leq n$, then we know

$$T_a \in \mathcal{H}[\![(\mathbf{let}\ \mathit{\Pi}_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_{n'} \,\|\, \mathbf{skip} \,\|\, \ldots \,\|\, \mathbf{skip}), (\sigma_c, \sigma_a, \circledcirc)]\!] \,.$$

From Lemma 32, we are done. Otherwise, since T_a only contains events of threads $1, \ldots, n$, we know the threads $n+1, \ldots, n'$ do not access the object. Similar to the proof of Lemma 32, we can construct simulations and prove $T_a \in \mathcal{H}[\![(\mathbf{let}\ \mathit{\Pi}_A\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!]$. Thus we are done.

2. $\mathit{\Pi} \preceq_\varphi^{\mathsf{MGC}} \mathit{\Pi}_A \implies \mathit{\Pi} \preceq_\varphi \mathit{\Pi}_A$:
For any n, C_1, …, C_n, σ_c, σ_o, σ_a and T such that $\varphi(\sigma_o) = \sigma_a$ and $T \in \mathcal{H}[\![(\mathbf{let}\ \mathit{\Pi}\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!]$, from Lemma 32, we know

$$T \in \mathcal{H}[\![(\mathbf{let}\ \mathit{\Pi}\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \,.$$

From $\mathit{\Pi} \preceq_\varphi^{\mathsf{MGC}} \mathit{\Pi}_A$, we know there exist T_c and T_a such that

$$T_c \in \mathsf{completions}(T) \wedge \mathit{\Pi}_A \rhd_n^{\mathsf{MGC}} (\sigma_a, T_a) \wedge T_c \preceq_{\mathsf{lin}} T_a \,.$$

By definitions, we see

$$\mathit{\Pi}_A \rhd_n^{\mathsf{MGC}} (\sigma_a, T_a) \implies \mathit{\Pi}_A \rhd (\sigma_a, T_a) \,.$$

Thus we are done. □

Below we prove an important lemma which relates the basic contextual refinement to a refinement over MGC which considers histories instead of observable behaviors. The idea behind this lemma will be useful in proving various equivalence results, including those for progress properties.

Lemma 34. $\mathit{\Pi} \sqsubseteq_\varphi \mathit{\Pi}_A \iff \mathit{\Pi} \subseteq_\varphi \mathit{\Pi}_A$.

Proof. 1. $\mathit{\Pi} \sqsubseteq_\varphi \mathit{\Pi}_A \implies \mathit{\Pi} \subseteq_\varphi \mathit{\Pi}_A$:
We first prove the following (a) and (b):
(a) For any n, σ_o, σ_c, T,
if $\sigma_c = \{x_\mathsf{t} \rightsquigarrow \_, y_\mathsf{t} \rightsquigarrow \_, z_\mathsf{t} \rightsquigarrow \_ \mid 1 \leq \mathsf{t} \leq n\}$ and
$T \in \mathcal{H}[\![(\mathbf{let}\ \mathit{\Pi}\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!]$,
then there exists $\mathcal{B}$ such that $T \approx \mathcal{B}$ and
$\mathcal{B} \in \mathcal{O}[\![(\mathbf{let}\ \mathit{\Pi}\ \mathbf{in}\ \mathsf{MGCp}_n), (\sigma_c, \sigma_o, \circledcirc)]\!]$,
where

$$\frac{}{\epsilon \approx \epsilon} \qquad \frac{\lambda \approx e \qquad T \approx \mathcal{B}}{\lambda :: T \approx e :: \mathcal{B}}$$

$$\frac{}{(\mathsf{t}, f_i, n) \approx (\mathsf{t}, \mathbf{out}, (i, n))} \qquad \frac{}{(\mathsf{t}, \mathbf{ret}, n) \approx (\mathsf{t}, \mathbf{out}, n)}$$

$$\frac{}{(\mathsf{t}, \mathbf{obj}, \mathbf{abort}) \approx (\mathsf{t}, \mathbf{obj}, \mathbf{abort})}$$

Proof. We define the simulation relation $\precsim_{\mathsf{MGCp}}$ in Figure 9(b), and prove the following (B.2) by case analysis and the operational semantics. This simulation ensures that at the right side (MGCp), each output of the method argument is immediately followed by invoking the method, and each method return is immediately followed by printing out the return value.

For any W_1, $\mathcal{S}_1$, W_2, $\mathcal{S}_2$ and e_1, if $(W_1, \mathcal{S}_1) \precsim_{\mathsf{MGCp}} (W_2, \mathcal{S}_2)$, then
(1) if $(W_1, \mathcal{S}_1) \overset{e_1}{\longmapsto}$ **abort** and $\mathsf{is\_obj\_abt}(e_1)$, then
there exists T_2 such that $(W_2, \mathcal{S}_2) \overset{T_2}{\longmapsto}{}^*$ **abort** and
$e_1 \approx \mathsf{get\_obsv}(T_2)$;
(2) if $(W_1, \mathcal{S}_1) \overset{e_1}{\longmapsto} (W_1', \mathcal{S}_1')$, then
there exist T_2, W_2' and $\mathcal{S}_2'$ such that $(W_2, \mathcal{S}_2) \overset{T_2}{\longmapsto}{}^* (W_2', \mathcal{S}_2')$,
$\mathsf{get\_hist}(e_1) \approx \mathsf{get\_obsv}(T_2)$ and $(W_1', \mathcal{S}_1') \precsim_{\mathsf{MGCp}} (W_2', \mathcal{S}_2')$. (B.2)

With (B.2), we can prove the following by induction over the number of steps generating the event trace of $\mathcal{H}[\![W_1, \mathcal{S}_1]\!]$.

If $(\lfloor W_1 \rfloor, \mathcal{S}_1) \precsim_{\mathsf{MGCp}} (\lfloor W_2 \rfloor, \mathcal{S}_2)$ and $T \in \mathcal{H}[\![W_1, \mathcal{S}_1]\!]$, then
there exists $\mathcal{B}$ such that $T \approx \mathcal{B}$ and $\mathcal{B} \in \mathcal{O}[\![W_2, \mathcal{S}_2]\!]$.

Then, since

$$(\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n \rfloor, (\emptyset, \sigma_o, \circledcirc)) \precsim_{\mathsf{MGCp}} (\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGCp}_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)),$$

we are done.

(b) For any n, σ_a, σ_c, $\mathcal{B}$,
if $\sigma_c = \{x_\mathsf{t} \rightsquigarrow \_, y_\mathsf{t} \rightsquigarrow \_, z_\mathsf{t} \rightsquigarrow \_ \mid 1 \le \mathsf{t} \le n\}$ and
$\mathcal{B} \in \mathcal{O}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGCp}_n), (\sigma_c, \sigma_a, \circledcirc)]\!]$,
then there exists T such that $T \approx \mathcal{B}$ and
$T \in \mathcal{H}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!]$.

Proof. We define the simulation relation $\precsim_{\mathsf{MGCp}^-}$ in Figure 9(c), and prove the following (B.3) by case analysis and the operational semantics. This simulation ensures two things. (i) Whenever the left side (MGCp) prints out a method argument, the right side (MGC) invokes the method using that argument. (ii) Whenever the left side prints out a return value, the right side must return the same value. We can ensure (i) and (ii) because x_t, y_t and z_t are all thread-local variables.

For any W_1, $\mathcal{S}_1$, W_2, $\mathcal{S}_2$ and e_1, if $(W_1, \mathcal{S}_1) \precsim_{\mathsf{MGCp}^-} (W_2, \mathcal{S}_2)$, then
(1) if $(W_1, \mathcal{S}_1) \overset{e_1}{\longmapsto}$ **abort**, then
there exists T_2 such that $(W_2, \mathcal{S}_2) \overset{T_2}{\longmapsto}{}^*$ **abort** and
$\mathsf{get\_hist}(T_2) \approx e_1$;
(2) if $(W_1, \mathcal{S}_1) \overset{e_1}{\longmapsto} (W_1', \mathcal{S}_1')$, then
there exist T_2, W_2' and $\mathcal{S}_2'$ such that $(W_2, \mathcal{S}_2) \overset{T_2}{\longmapsto}{}^* (W_2', \mathcal{S}_2')$,
$\mathsf{get\_hist}(T_2) \approx \mathsf{get\_obsv}(e_1)$ and $(W_1', \mathcal{S}_1') \precsim_{\mathsf{MGCp}^-} (W_2', \mathcal{S}_2')$. (B.3)

With (B.3), we can prove the following by induction over the number of steps generating the event trace of $\mathcal{O}[\![W_1, \mathcal{S}_1]\!]$.
If $(\lfloor W_1 \rfloor, \mathcal{S}_1) \precsim_{\mathsf{MGCp}^-} (\lfloor W_2 \rfloor, \mathcal{S}_2)$ and $\mathcal{B} \in \mathcal{O}[\![W_1, \mathcal{S}_1]\!]$, then there exists T such that $T \approx \mathcal{B}$ and $T \in \mathcal{H}[\![W_2, \mathcal{S}_2]\!]$.
Then, since

$$(\lfloor \textbf{let } \Pi \textbf{ in } \mathsf{MGCp}_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \precsim_{\mathsf{MGCp}^-} (\lfloor \textbf{let } \Pi \textbf{ in } \mathsf{MGC}_n \rfloor, (\emptyset, \sigma_a, \circledcirc)),$$

we are done.

Then, since $\Pi \sqsubseteq_\varphi \Pi_A$, we know

$$\forall n, \sigma_c, \sigma_o, \sigma_a.\ (\varphi(\sigma_o) = \sigma_a) \implies \mathcal{O}[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGCp}_n), (\sigma_c, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{O}[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGCp}_n), (\sigma_c, \sigma_a, \circledcirc)]\!].$$

Thus from (a) and (b), we get

$$\forall n, \sigma_o, \sigma_a.\ (\varphi(\sigma_o) = \sigma_a) \implies \mathcal{H}[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{H}[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!].$$

Then we are done.

2. $\Pi \subsetneq_\varphi \Pi_A \implies \Pi \sqsubseteq_\varphi \Pi_A$:
We define the simulation relation $\precsim$ in Figure 9(d), and prove the following (B.4) by case analysis and the operational semantics. This simulation relates one program to two programs. We use the MGC at the abstract level to help determine the abstract program that corresponds to the concrete one. Specifically, we require the histories generated by the concrete program can also be generated by the abstract MGC. Then, when an abstract thread is in a method call, its code should be the same as the MGC thread. Otherwise, its code is the same as the concrete thread.

For any W_1, $\mathcal{S}_1$, W_2, $\mathcal{S}_2$, W_3, $\mathcal{S}_3$ and e_1,
if $(W_1, \mathcal{S}_1) \precsim (W_2, \mathcal{S}_2; W_3, \mathcal{S}_3)$, then
(1) if $(W_1, \mathcal{S}_1) \overset{e_1}{\longmapsto}$ **abort**, then
there exists T_3 such that $(W_3, \mathcal{S}_3) \overset{T_3}{\longmapsto}{}^*$ **abort** and
$e_1 = \mathsf{get\_obsv}(T_3)$;
(2) if $(W_1, \mathcal{S}_1) \overset{e_1}{\longmapsto} (W_1', \mathcal{S}_1')$, then
there exist T_2, W_2', $\mathcal{S}_2'$, T_3, W_3' and $\mathcal{S}_3'$ such that
$(W_2, \mathcal{S}_2) \overset{T_2}{\longmapsto}{}^* (W_2', \mathcal{S}_2')$, $(W_3, \mathcal{S}_3) \overset{T_3}{\longmapsto}{}^* (W_3', \mathcal{S}_3')$,
$\mathsf{get\_obsv}(e_1) = \mathsf{get\_obsv}(T_3)$ and $(W_1', \mathcal{S}_1') \precsim (W_2', \mathcal{S}_2'; W_3', \mathcal{S}_3')$. (B.4)

With (B.4), we can prove the following by induction over the number of steps generating the event trace of $\mathcal{O}[\![W_1, \mathcal{S}_1]\!]$.

If $(W_1, \mathcal{S}_1) \precsim (W_2, \mathcal{S}_2; W_3, \mathcal{S}_3)$, then $\mathcal{O}[\![W_1, \mathcal{S}_1]\!] \subseteq \mathcal{O}[\![W_3, \mathcal{S}_3]\!]$.

For any n, C_1, ..., C_n, σ_c, σ_o and σ_a, by Lemma 32, we know

$$\mathcal{H}[\![(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{H}[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] .$$

Since $\Pi \subseteq_\varphi \Pi_A$, we know if $\varphi(\sigma_o) = \sigma_a$, then

$$\mathcal{H}[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{H}[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!] .$$

Then we know

$$\begin{array}{l}(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \sigma_o, \circledcirc)) \\ \precsim (\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n, (\emptyset, \sigma_a, \circledcirc); \\ \quad \textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \sigma_a, \circledcirc)) .\end{array}$$

Thus, we get

$$\begin{array}{l}\mathcal{O}[\![(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!] \\ \subseteq \mathcal{O}[\![(\textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a, \circledcirc)]\!] .\end{array}$$

Thus we are done. □

Then, we prove the following (B.5) and can get Theorem 6.

$$\Pi \subseteq_\varphi \Pi_A \iff \Pi \preceq_\varphi^{\mathsf{MGC}} \Pi_A \tag{B.5}$$

1. $\Pi \subseteq_\varphi \Pi_A \implies \Pi \preceq_\varphi^{\mathsf{MGC}} \Pi_A$:
We only need to prove the following lemma (remember we assume that each C_i in Π_A is of the form $\langle C \rangle$ and it is always safe to execute Π_A).

Lemma 35 (Π_A is Linearizable). *For any n, σ_a and T,*
if $T \in \mathcal{H}[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!]$,
then there exist T_c and T_a such that $T_c \in \mathsf{completions}(T)$, $T_c \preceq_{\mathsf{lin}} T_a$,
$T_a \in \mathcal{H}[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!]$ and $\mathsf{seq}(T_a)$.

Proof. We define a new operational semantics, in which we additionally generate two events at the single step of the method body. We know the method body in the execution can only be $\langle C \rangle; \textbf{noret}$, and hence the resulting code after one step (if not block) must be $\textbf{fret}(n')$ for some n'.

$$\frac{\begin{array}{c}(\langle C \rangle; \textbf{noret}, \sigma_o \uplus \sigma_l) \longrightarrow_{\mathsf{t}} (\textbf{fret}(n'), \sigma_o' \uplus \sigma_l') \\ dom(\sigma_l) = dom(\sigma_l') \qquad \sigma_l = \{y \rightsquigarrow n\} \qquad \Pi(f) = (y, \langle C \rangle)\end{array}}{(\langle C \rangle; \textbf{noret}, (\sigma_c, \sigma_o, (\sigma_l, x, C_c))) \xrightarrow{[\mathsf{t}, f, n] :: [\mathsf{t}, \textbf{ret}, n']}_{\mathsf{t}, \Pi} (\textbf{fret}(n'), (\sigma_c, \sigma_o', (\sigma_l', x, C_c)))}$$

Here $[\mathsf{t}, f, n]$ and $[\mathsf{t}, \textbf{ret}, n']$ are two new events (called *atom-invocation event* and *atom-return event* respectively) generated for the new semantics. We use $T|_{[]}$ to project the event trace T to the new events, and use $\lfloor e \rfloor$ (and $\lfloor T \rfloor$) to transform the new event (and the event trace) to an old event (and a trace of old events), where $[\mathsf{t}, f, n]$ is transformed to (t, f, n) and $[\mathsf{t}, \textbf{ret}, n']$ is transformed to $(\mathsf{t}, \textbf{ret}, n')$. Other parts of the semantics are the same as the operational semantics in Figure 5. We can define $\mathcal{T}_{[]}[\![W, \mathcal{S}]\!]$ in a similar way as $\mathcal{T}[\![W, \mathcal{S}]\!]$, which uses the new semantics instead of the original one and keeps all the events including the new events.

(1) We can prove that there is a lock-step simulation between the original semantics in Figure 5 and the new semantics. Then, for any T such that
$$T \in \mathcal{H}[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!],$$
we have an corresponding execution under the new semantics to generate T_T such that
$$T_T \in \mathcal{T}_{[]}[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!],$$
and $\mathsf{get\_hist}(T_T) = T$.

(2) Below we show:
If $T_T \in \mathcal{T}_{[]}[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!]$, $T = \mathsf{get\_hist}(T_T)$ and $T_a = \lfloor T_T|_{[]} \rfloor$,
then $\mathsf{seq}(T_a)$ and there exists T_c such that $T_c \in \mathsf{completions}(T)$ and $T_c \preceq_{\mathsf{lin}} T_a$.

Proof. By the new operational semantics, we know $\mathsf{seq}(T_a)$ holds.

Construct T_c and Prove Linearizability Condition 1: By the new operational semantics, we know that for any t, $T|_{\mathsf{t}}$ and $T_a|_{\mathsf{t}}$ must satisfy one of the following:
(i) $T|_{\mathsf{t}} = T_a|_{\mathsf{t}}$; or
(ii) $\exists n.\ T|_{\mathsf{t}} :: (\mathsf{t}, \mathbf{ret}, n) = T_a|_{\mathsf{t}}$; or
(iii) $\exists f, n.\ T|_{\mathsf{t}} = T_a|_{\mathsf{t}} :: (\mathsf{t}, f, n)$.
We construct T_e as follows. For any t, if it is the above case (ii), we append the corresponding return event at the end of T. Since $\mathsf{well\_formed}(T)$ and $\mathsf{well\_formed}(T_a)$, we could prove $\mathsf{well\_formed}(T_e)$. Thus $T_e \in \mathsf{extensions}(T)$.
Also T_e satisfies: for any t, one of the following holds:
(i) $T_e|_{\mathsf{t}} = T_a|_{\mathsf{t}}$; or
(ii) $\exists f, n.\ T_e|_{\mathsf{t}} = T_a|_{\mathsf{t}} :: (\mathsf{t}, f, n)$.
Let $T_c = \mathsf{truncate}(T_e)$. Thus $T_c \in \mathsf{completions}(T)$.
Since $\forall \mathsf{t}.\ \mathsf{is\_res}(\mathsf{last}(T_a|_{\mathsf{t}})) \wedge \mathsf{seq}(T_a|_{\mathsf{t}})$, we could prove that for any t,
(i) if $T_e|_{\mathsf{t}} = T_a|_{\mathsf{t}}$, then $T_c|_{\mathsf{t}} = T_e|_{\mathsf{t}}$;
(ii) if $T_e|_{\mathsf{t}} = T_a|_{\mathsf{t}} :: (\mathsf{t}, f, n)$, then $T_c|_{\mathsf{t}} = T_a|_{\mathsf{t}}$.
Thus $\forall \mathsf{t}.\ T_c|_{\mathsf{t}} = T_a|_{\mathsf{t}}$.

Prove Linearizability Condition 2: We informally show that the bijection π implicit in $\forall \mathsf{t}.\ T_c|_{\mathsf{t}} = T_a|_{\mathsf{t}}$ preserves the response-invocation order.
Let $T_c(i)$ be a response event in T_c and let $T_c(j)$ be an invocation event. Then $\pi(i)$ and $\pi(j)$ are the indices of $T_c(i)$ and $T_c(j)$ in T_a respectively. Suppose $i < j$. By the construction of T_c from T, we know the same response and invocation events are in T, and the response happens before the invocation. Let i' and j' be the indices of these events in T. Then $i' < j'$. By the new operational semantics, we know in T_T, the atom-return event is before the atom-invocation event since the history return event is before the history invocation event. Thus $\pi(i) < \pi(j)$.

(3) Finally, we show the following and finish the proof of the lemma:
If $T_T \in \mathcal{T}_{[]}[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!]$ and $T_a = \lfloor T_T|_{[]} \rfloor$,
then $T_a \in \mathcal{H}[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!]$.

This is proved by constructing the following simulation $\precsim_{\mathsf{new}}$. This simulation ensures that the right side invokes and returns from a method at the time when the left side generates the new atomic events.

$(\mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n, (\emptyset, \sigma_a, \{1 \rightsquigarrow \kappa_1, \ldots, n \rightsquigarrow \kappa_n\}))$
$\precsim_{\mathsf{new}} (\mathbf{let}\ \Pi_A\ \mathbf{in}\ C'_1 \,\|\, \ldots \,\|\, C'_n, (\emptyset, \sigma_a, \{1 \rightsquigarrow \kappa'_1, \ldots, n \rightsquigarrow \kappa'_n\}))$
where $\forall i.\ (C_i, \kappa_i) \precsim_{\mathsf{new}} (C'_i, \kappa'_i)$

$(C, \circ) \precsim_{\mathsf{new}} (C, \circ)$

$((\langle C \rangle; \mathbf{noret}),\ (\sigma_l, \cdot, (\mathbf{skip}; \mathsf{MGT}))) \precsim_{\mathsf{new}} ((f_{\mathbf{rand}(m)}(\mathbf{rand}()); \mathsf{MGT}),\ \circ)$

$(\mathbf{fret}(n'), (\sigma_l, \cdot, (\mathbf{skip}; \mathsf{MGT}))) \precsim_{\mathsf{new}} ((\mathbf{skip}; \mathsf{MGT}), \circ)$

We prove the following by case analysis and the operational semantics.

For any W_1, $\mathcal{S}_1$, W_2, $\mathcal{S}_2$ and T_1,
if $(W_1, \mathcal{S}_1) \precsim_{\mathsf{new}} (W_2, \mathcal{S}_2)$ and $(W_1, \mathcal{S}_1) \overset{T_1}{\longmapsto} (W'_1, \mathcal{S}'_1)$ in the new semantics,
then there exist T_2, W'_2 and $\mathcal{S}'_2$ such that $(W_2, \mathcal{S}_2) \overset{T_2}{\longmapsto}{}^* (W'_2, \mathcal{S}'_2)$, $\mathsf{get\_hist}(T_2) = \lfloor T_1|_{[]} \rfloor$ and $(W'_1, \mathcal{S}'_1) \precsim_{\mathsf{new}} (W'_2, \mathcal{S}'_2)$.

Then we can prove the following by induction over the number of steps generating the event trace of $\mathcal{T}_{[]}[\![W_1, \mathcal{S}_1]\!]$.

If $(W_1, \mathcal{S}_1) \precsim_{\mathsf{new}} (W_2, \mathcal{S}_2)$, $T_T \in \mathcal{T}_{[]}[\![W_1, \mathcal{S}_1]\!]$ and $T_a = \lfloor T_T|_{[]} \rfloor$, then $T_a \in \mathcal{H}[\![W_2, \mathcal{S}_2]\!]$.

Since we know
$(\mathbf{let}\ \Pi_A\ \mathbf{in}\ \mathsf{MGC}_n, (\emptyset, \sigma_a, \circledcirc)) \precsim_{\mathsf{new}} (\mathbf{let}\ \Pi_A\ \mathbf{in}\ \mathsf{MGC}_n, (\emptyset, \sigma_a, \circledcirc))$,
we are done.

The lemma is immediate from the above (1), (2) and (3). □

2. $\Pi \preceq_{\varphi}^{\mathsf{MGC}} \Pi_A \implies \Pi \subseteq_{\varphi} \Pi_A$:
We only need to prove the following lemma (similar to the Rearrangement Lemma in [5]):

Lemma 36 (Rearrangement). *For any n, σ_a, T and T_a,*
if $T \preceq_{\mathsf{lin}} T_a$, $T_a \in \mathcal{H}[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!]$ and $\mathsf{seq}(T_a)$,
then $T \in \mathcal{H}[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!]$.

Proof. Suppose $|T| = n$. We know T must not contain the abort event. From $T \preceq_{\mathsf{lin}} T_a$, we know
(i) $\forall \mathsf{t}.\ T|_{\mathsf{t}} = T_a|_{\mathsf{t}}$;
(ii) there exists a bijection $\pi : \{1, \ldots, |T|\} \to \{1, \ldots, |T_a|\}$ such that $\forall i.\ T(i) = T_a(\pi(i))$ and $\forall i, j.\ i < j \wedge \mathsf{is\_res}(T(i)) \wedge \mathsf{is\_inv}(T(j)) \implies \pi(i) < \pi(j)$.
We construct the execution under the new semantics (defined in the proof of Lemma 35) which generates T, and the new events constitute T_a, *i.e.*, we want to show the following holds:

$$\exists T_T.\ T_T \in \mathcal{T}_{[]}[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!] \wedge T = \mathsf{get\_hist}(T_T). \quad \text{(B.6)}$$

Then we prove that there is a lock-step simulation between the new semantics and the original semantics in Figure 5, and we can get

$$T \in \mathcal{H}[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!].$$

Below we prove (B.6). We prove that for any k, there exist T_T, W', $\mathcal{S}'$ and k' such that

$$\begin{array}{l}(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n, (\emptyset, \sigma_a, \circledcirc)) \xmapsto{T_T}{}^* (W', \mathcal{S}') \\ \wedge\, \mathsf{get\_hist}(T_T) = T(1..k) \wedge \lfloor T_T|_{[]} \rfloor = T_a(1..k') \\ \wedge\, (\forall \mathcal{S}''.\ (\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n, (\emptyset, \sigma_a, \circledcirc)) \xmapsto{T_a(1..k')}{}^* (\_, \mathcal{S}'') \\ \quad \Longrightarrow\ \mathcal{S}''|_{\mathbf{obj}} = \mathcal{S}'|_{\mathbf{obj}}),\end{array}$$

where $\mathcal{S}'|_{\mathbf{obj}}$ get the object memory in $\mathcal{S}'$.
By induction over k.
Base Case: If $k = 0$, trivial.
Inductive Step: Suppose there exist T_1, W_1, $\mathcal{S}_1$ and k_1 such that

$$\begin{array}{l}(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n, (\emptyset, \sigma_a, \circledcirc)) \xmapsto{T_1}{}^* (W_1, \mathcal{S}_1) \\ \wedge\, \mathsf{get\_hist}(T_1) = T(1..k) \wedge \lfloor T_1|_{[]} \rfloor = T_a(1..k_1) \\ \wedge\, (\forall \mathcal{S}_1'.\ (\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n, (\emptyset, \sigma_a, \circledcirc)) \xmapsto{T_a(1..k_1)}{}^* (\_, \mathcal{S}_1') \\ \quad \Longrightarrow\ \mathcal{S}_1'|_{\mathbf{obj}} = \mathcal{S}_1|_{\mathbf{obj}}),\end{array}$$

we want to show there exist T_2, W_2, $\mathcal{S}_2$ and k_2 such that

$$\begin{array}{l}(W_1, \mathcal{S}_1) \xmapsto{T_2}{}^* (W_2, \mathcal{S}_2) \\ \wedge\, \mathsf{get\_hist}(T_2) = T(k+1) \wedge \lfloor T_2|_{[]} \rfloor = T_a(k_1 + 1..k_2) \\ \wedge\, (\forall \mathcal{S}_2'.\ (\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n, (\emptyset, \sigma_a, \circledcirc)) \xmapsto{T_a(1..k_2)}{}^* (\_, \mathcal{S}_2') \\ \quad \Longrightarrow\ \mathcal{S}_2'|_{\mathbf{obj}} = \mathcal{S}_2|_{\mathbf{obj}}),\end{array}$$

By case analysis.
(a) $T(k+1) = (\mathsf{t}, f, n')$.
Suppose $T(k+1) = (T|_{\mathsf{t}})(i)$.
From $T|_{\mathsf{t}} = T_a|_{\mathsf{t}}$ and $T_a \in \mathcal{H}[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!]$, we know $i = 1$ or $\mathsf{is\_ret}((T|_{\mathsf{t}})(i-1))$ holds.
i. If $i = 1$, we just let the code MGT of the thread t executes to calling the method f using the argument n, and generates the event (t, f, n').
ii. If $\mathsf{is\_ret}((T|_{\mathsf{t}})(i-1))$ holds, we know the code of the thread t is in the client code. Still we can let it execute to the method call of f using the argument n, generating the event (t, f, n').
(b) $T(k+1) = (\mathsf{t}, \mathbf{ret}, n')$.
Suppose $T(k+1) = (T|_{\mathsf{t}})(i)$. Similar to the previous case, we know $\mathsf{is\_inv}((T|_{\mathsf{t}})(i-1))$ holds. Suppose $(T|_{\mathsf{t}})(i-1) = e = (\mathsf{t}, f, n)$ and $\Pi_A(f) = (x, \langle C \rangle)$. Thus the code of the thread t is either $\langle C \rangle; \mathbf{noret}$ or $\mathbf{fret}(n'')$ (for some n'').
i. The code of t is $\langle C \rangle; \mathbf{noret}$.
Thus $\mathsf{last}(T_1|_{\mathsf{t}}) = e$. Suppose $|T(1..k)|_{\mathsf{t}}| = n_1$. From the operational semantics and the generation of T_1, we know $|T_a(1..k_1)|_{\mathsf{t}}| = n_1 - 1$. For the bijection π in (ii) which maps events in T to events of T_a, we let $k_2 = \pi(k+1)$. Since $T|_{\mathsf{t}} = T_a|_{\mathsf{t}}$, we know $k_2 > k_1$. Let $k' = k_2 - k_1$. Suppose $T_a(k_1 + 1..k_2) = e_1 :: \ldots :: e_{k'}$. Since $\lfloor T_1|_{[]} \rfloor = T_a(1..k_1)$,

by the operational semantics and the generation of T_1, we know $\mathsf{is\_ret}(T_a(k_1))$. Since $\mathsf{seq}(T_a)$, we know $\mathsf{seq}(e_1 :: \ldots :: e_{k'})$ and $k' = 2j$. Suppose the threads of the events e_1, ..., $e_{k'}$ are $\mathtt{t}_1, \ldots, \mathtt{t}_j$ respectively where $\mathtt{t}_j = \mathtt{t}$. Below we prove that for any i such that $1 \leq i \leq j$, the current code of the thread $\mathtt{t}_i$ is $\langle C_i \rangle; \mathbf{noret}$ (for some method body $\langle C_i \rangle$), and $e_{2i-1} = \mathsf{last}(T(1..k)|_{\mathtt{t}_i})$. The proof is by contradiction. Suppose $e_{2i-1} = T(i')$ and $i' > k$. Since $T(k+1) = (\mathtt{t}, \mathbf{ret}, n')$ and $\mathsf{is\_inv}(e_{2i-1})$, we know $i' > k+1$. By (ii), we know $\pi(i') > \pi(k+1) = k_2$, which contradicts the fact that e_{2i-1} is an event in $T_a(k_1 + 1..k_2)$. Thus, $i' \leq k$, and since $\lfloor T_1|_{[]} \rfloor = T_a(1..k_1)$, by the operational semantics and the generation of T_1, we know $e_{2i-1} = \mathsf{last}(T_1|_{\mathtt{t}_i})$. Thus we are done.

We let the threads $\mathtt{t}_1, \ldots, \mathtt{t}_j$ execute one step in order, generating the event trace T_2' which only contains the atom-invocation and atom-return events, and then the thread $\mathtt{t}_j$ execute one more step generating $e_{k'} = T_a(k_2) = T_a(\pi(k+1)) = T(k+1)$. Since $T_a \in \mathcal{H}[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!]$, we know this execution is possible, and moreover we have $\lfloor T_2|_{[]} \rfloor = \lfloor T_2' \rfloor = T_a(k_1 + 1..k_2)$.

ii. The code of $\mathtt{t}$ is $\mathbf{fret}(n'')$.

Thus $\mathsf{last}(T_1|_{\mathtt{t}}) = [\mathtt{t}, \mathbf{ret}, n'']$. Since $\lfloor T_1|_{[]} \rfloor = T_a(1..k_1)$, we know $\mathsf{last}(T_a(1..k_1)|_{\mathtt{t}}) = (\mathtt{t}, \mathbf{ret}, n'')$. Suppose $|T_a(1..k_1)|_{\mathtt{t}}| = n_1$.

From the operational semantics and the generation of T_1, we know $|\mathsf{get\_hist}(T_1|_{\mathtt{t}})| = |T(1..k)|_{\mathtt{t}}| = n_1 - 1$. Since $T|_{\mathtt{t}} = T_a|_{\mathtt{t}}$, we know $n' = n''$. The code of $\mathtt{t}$ is $\mathbf{fret}(n')$. We let it execute one step and generate the event $(\mathtt{t}, \mathbf{ret}, n')$.

Thus (B.6) holds and we are done. □

From $\Pi \preceq_{\varphi}^{\mathsf{MGC}} \Pi_A$, we know

$$\begin{array}{l} \forall n, \sigma_o, \sigma_a, T.\ \ T \in \mathcal{H}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \wedge (\varphi(\sigma_o) = \sigma_a) \\ \Longrightarrow\ \exists T_c, T_a.\ T_c \in \mathsf{completions}(T) \wedge T_a \in \mathcal{H}[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!] \\ \qquad\qquad \wedge\, \mathsf{seq}(T_a) \wedge T_c \preceq_{\mathsf{lin}} T_a \end{array}$$

From Lemma 36, we know

$$\begin{array}{l} \forall n, \sigma_o, \sigma_a, T.\ \ T \in \mathcal{H}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \wedge (\varphi(\sigma_o) = \sigma_a) \\ \Longrightarrow\ \exists T_c.\ T_c \in \mathsf{completions}(T) \wedge T_c \in \mathcal{H}[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!] \end{array}$$

Since $T_c \in \mathsf{completions}(T)$, we know there exists T_e such that $T_c = \mathsf{truncate}(T_e)$ and $T_e \in \mathsf{extensions}(T)$. By the definition of $\mathsf{truncate}(T_e)$, we can prove:

$$T_e \in \mathcal{H}[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!]$$

Then, by the definition of $T_e \in \mathsf{extensions}(T)$, we can prove:

$$T \in \mathcal{H}[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!]$$

Thus we get $\Pi \subseteq_{\varphi} \Pi_A$.

B.2 Proofs of Figures 1 and 7

Lemma 37 (Figure 7). *Assume* $T \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!]$.

1. $\mathsf{wait\text{-}free}(T) \iff \mathsf{prog\text{-}t}(T) \vee \mathsf{non\text{-}sched}(T) \vee \mathsf{abt}(T) \iff \mathsf{non\text{-}sched}(T) \vee \mathsf{abt}(T)$;
2. $\mathsf{lock\text{-}free}(T) \iff \mathsf{prog\text{-}s}(T) \vee \mathsf{non\text{-}sched}(T) \vee \mathsf{abt}(T) \iff \mathsf{wait\text{-}free}(T) \vee \mathsf{prog\text{-}s}(T)$;
3. $\mathsf{obstruction\text{-}free}(T) \iff \mathsf{prog\text{-}t}(T) \vee \mathsf{non\text{-}sched}(T) \vee \neg\mathsf{iso}(T) \vee \mathsf{abt}(T) \iff \mathsf{lock\text{-}free}(T) \vee \neg\mathsf{iso}(T)$;
4. $\mathsf{deadlock\text{-}free}(T) \iff \mathsf{prog\text{-}s}(T) \vee \neg\mathsf{fair}(T) \vee \mathsf{abt}(T) \iff \mathsf{lock\text{-}free}(T) \vee \neg\mathsf{fair}(T)$;
5. $\mathsf{starvation\text{-}free}(T) \iff \mathsf{prog\text{-}t}(T) \vee \neg\mathsf{fair}(T) \vee \mathsf{abt}(T) \iff \mathsf{wait\text{-}free}(T) \vee \neg\mathsf{fair}(T)$.

Proof. 1. By definition.

$$\begin{aligned}
\mathsf{wait\text{-}free}(T) \iff & (\,\forall i, e.\ e \in \mathsf{pend_inv}(T(1..i)) \\
& \quad \implies (\exists j.\ j > i \wedge \mathsf{match}(e, T(j))) \\
& \qquad \vee (\exists j.\ j > i \wedge (\forall k \geq j.\ \mathsf{tid}(T(k)) \neq \mathsf{tid}(e)))\,) \\
& \vee \mathsf{abt}(T) \\
\iff & (\,\forall i, e.\ e \in \mathsf{pend_inv}(T(1..i)) \wedge \neg(\exists j.\ j > i \wedge \mathsf{match}(e, T(j))) \\
& \quad \implies (\exists j.\ j > i \wedge (\forall k \geq j.\ \mathsf{tid}(T(k)) \neq \mathsf{tid}(e)))\,) \\
& \vee \mathsf{abt}(T) \\
\iff & (\,\forall e.\ e \in \mathsf{pend_inv}(T) \implies (\exists j.\ \forall k \geq j.\ \mathsf{tid}(T(k)) \neq \mathsf{tid}(e))\,) \\
& \vee \mathsf{abt}(T) \\
\iff & \mathsf{non\text{-}sched}(T) \vee \mathsf{abt}(T)
\end{aligned}$$

Also, we can prove $\mathsf{prog\text{-}t}(T) \implies \mathsf{non\text{-}sched}(T)$ as follows.

$$\begin{aligned}
\mathsf{prog\text{-}t}(T) & \iff (\,\forall i, e.\ e \in \mathsf{pend_inv}(T(1..i)) \implies \exists j.\ j > i \wedge \mathsf{match}(e, T(j))\,) \\
& \iff (\,\forall i, e.\ e \in \mathsf{pend_inv}(T(1..i)) \implies e \notin \mathsf{pend_inv}(T)\,) \\
& \iff (\,\mathsf{pend_inv}(T) = \emptyset\,) \\
& \implies \mathsf{non\text{-}sched}(T)
\end{aligned}$$

2. We only need to prove the first equivalence. The second is trivial from the first one.

$$\begin{aligned}
\mathsf{lock\text{-}free}(T) \iff & (\,\forall i, e.\ e \in \mathsf{pend_inv}(T(1..i)) \\
& \quad \implies (\exists j.\ j > i \wedge \mathsf{is_ret}(T(j))) \\
& \qquad \vee (\exists j.\ j > i \wedge (\forall k \geq j.\ \mathsf{is_clt}(T(k))))\,) \\
& \vee \mathsf{abt}(T) \\
\iff & \mathsf{prog\text{-}s}(T) \\
& \vee (\,\exists j.\ \forall k \geq j.\ \mathsf{is_clt}(T(k))\,) \\
& \vee \mathsf{abt}(T)
\end{aligned}$$

From $\exists j.\ \forall k \geq j.\ \mathsf{is\_clt}(T(k))$ and the operational semantics generating T, we know $\mathsf{non\text{-}sched}(T)$ holds.
If $\mathsf{non\text{-}sched}(T)$ holds, we know there exists j such that $\forall k \geq j.\ \mathsf{tid}(T(k)) \notin \mathsf{tid}(\mathsf{pend\_inv}(T))$, where $\mathsf{tid}(\mathsf{pend\_inv}(T))$ gets the set of thread IDs of the pending invocations in T. Then by the operational semantics and the generation of T, we know either $\exists j.\ \forall k \geq j.\ \mathsf{is\_clt}(T(k))$ or $\mathsf{prog\text{-}s}(T)$ holds.

3. For obstruction-freedom, we only need to prove the following:
   (1) $\forall T.\ \mathsf{iso}(T) \wedge \mathsf{obstruction\text{-}free}(T) \implies \mathsf{wait\text{-}free}(T)$;
   (2) $\forall T.\ \mathsf{wait\text{-}free}(T) \implies \mathsf{obstruction\text{-}free}(T)$;
   (3) $\forall T.\ \neg\mathsf{iso}(T) \implies \mathsf{obstruction\text{-}free}(T)$;
   (4) $\forall n, C_1, \ldots, C_n, \sigma_c, \sigma_o, T.$
   $T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n), (\sigma_c, \sigma_o, \circledcirc)]\!] \wedge \mathsf{prog\text{-}s}(T)$
   $\implies \mathsf{obstruction\text{-}free}(T)$.

   For (1) $\forall T.\ \mathsf{iso}(T) \wedge \mathsf{obstruction\text{-}free}(T) \implies \mathsf{wait\text{-}free}(T)$:

   Proof. By $\mathsf{obstruction\text{-}free}(T)$, we know one of the following holds:
   (a) there exists i such that $\mathsf{is\_abt}(T(i))$ holds; or
   (b) for any i and e, if $e \in \mathsf{pend\_inv}(T(1..i))$, then one of the following holds:
      (i) there exists $j > i$ such that $\mathsf{match}(e, T(j))$; or
      (ii) $\forall j > i.\ \exists k.\ k \geq j \wedge \mathsf{tid}(T(k)) \neq \mathsf{tid}(e)$.

   For (a), we know $\mathsf{wait\text{-}free}(T)$.
   For (b), for any i and e, where $e \in \mathsf{pend\_inv}(T(1..i))$, we let $\mathsf{t} = \mathsf{tid}(e)$. Since $\mathsf{iso}(T)$, we know

   $$|T| \neq \omega \ \vee\ \exists \mathsf{t}, i.\ (\forall j.\ j \geq i \implies \mathsf{tid}(T(j)) = \mathsf{t})\,.$$

   If $|T| \neq \omega$, we know (ii) cannot hold. Thus (i) must hold.
   Otherwise, we know there exists t_0 and i_0 such that

   $$\forall j.\ j \geq i_0 \implies \mathsf{tid}(T(j)) = \mathsf{t}_0\,.$$

   If $\mathsf{t}_0 = \mathsf{t}$, we know (ii) does not hold, and hence (i) holds. Otherwise, if $\mathsf{t}_0 \neq \mathsf{t}$, we know

   $$\forall k.\ k \geq i_0 \implies \mathsf{tid}(T(k)) \neq \mathsf{tid}(e)\,.$$

   Thus we know $\mathsf{wait\text{-}free}(T)$.

   For (2) $\forall T.\ \mathsf{wait\text{-}free}(T) \implies \mathsf{obstruction\text{-}free}(T)$:

   Proof. From $\mathsf{wait\text{-}free}(T)$, we know one of the following holds:
   (i) there exists i such that $\mathsf{is\_abt}(T(i))$ holds; or
   (ii) for any i and e, if $e \in \mathsf{pend\_inv}(T(1..i))$, then one of the following holds:
      (1) there exists $j > i$ such that $\forall k \geq j.\ \mathsf{tid}(T(k)) \neq \mathsf{tid}(e)$; or
      (2) there exists $j > i$ such that $\mathsf{match}(e, T(j))$.

   For (i), we know $\mathsf{obstruction\text{-}free}(T)$ holds.
   For (ii), for any i and e, where $e \in \mathsf{pend\_inv}(T(1..i))$, if (1) holds, we know

   $$\forall j > i.\ \exists k.\ k \geq j \wedge \mathsf{tid}(T(k)) \neq \mathsf{tid}(e).$$

   Thus we know $\mathsf{obstruction\text{-}free}(T)$.

   For (3) $\forall T.\ \neg\mathsf{iso}(T) \implies \mathsf{obstruction\text{-}free}(T)$:

   Proof. From $\neg\mathsf{iso}(T)$, we know

   $$|T| = \omega \wedge \forall \mathsf{t}, i.\ \exists j.\ j \geq i \wedge \mathsf{tid}(T(j)) \neq \mathsf{t}\,.$$

Thus, for any i and e, where $e \in \mathsf{pend\_inv}(T(1..i))$, we know

$$\forall j.\ \exists k.\ k \geq j \wedge \mathsf{tid}(T(k)) \neq \mathsf{tid}(e)\,.$$

Thus we have proved $\mathsf{obstruction\text{-}free}(T)$.

For (4) $\forall n, C_1, \ldots, C_n, \sigma_c, \sigma_o, T.\ T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!] \wedge \mathsf{prog\text{-}s}(T) \implies \mathsf{obstruction\text{-}free}(T)$:

Proof. From $\mathsf{prog\text{-}s}(T)$, we know: for any i, if $\mathsf{pend\_inv}(T(1..i)) \neq \emptyset$, then there exists $j > i$ such that $\mathsf{is\_ret}(T(j))$.
If $|T| \neq \omega$, by Lemma 51, we know $\mathsf{obstruction\text{-}free}(T)$ hold. Otherwise, $|T| = \omega$. For any i and e such that $e \in \mathsf{pend\_inv}(T(1..i))$, we know one of the following must hold:
(1) there exists $j > i$ such that $\mathsf{match}(e, T(j))$; or
(2) $\forall j.\ j > i \Rightarrow \neg\mathsf{match}(e, T(j))$.
For (2), we know

$$\forall j.\ j > i\ \Rightarrow\ e \in \mathsf{pend\_inv}(T(1..j))\,.$$

Thus we have

$$\forall j.\ j > i\ \Rightarrow\ \exists k.\ k > j \wedge \mathsf{is\_ret}(T(k))\,.$$

Then we know

$$\forall j > i.\ \exists k.\ k > j \wedge \mathsf{is\_ret}(T(k)) \wedge \mathsf{tid}(T(k)) \neq \mathsf{tid}(e)\,.$$

Thus we know $\mathsf{obstruction\text{-}free}(T)$.

4. The first equivalence is trivial from definition. For the second equivalence, we only need to prove the following:

$$\mathsf{non\text{-}sched}(T) \wedge \neg\mathsf{prog\text{-}s}(T)\ \implies\ \neg\mathsf{fair}(T)\,.$$

From the proof of the equivalences for wait-freedom, we know

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

Thus we only need to prove the following.
(1) $\mathsf{non\text{-}sched}(T) \wedge (\mathsf{pend\_inv}(T) \neq \emptyset)\ \implies\ \neg\mathsf{fair}(T)$;
(2) $\mathsf{prog\text{-}t}(T)\ \implies\ \mathsf{prog\text{-}s}(T)$.
For (1), from the premises, we know

$$\exists e, i.\ e \in \mathsf{pend\_inv}(T) \wedge \forall j \geq i.\ \mathsf{tid}(T(j)) \neq \mathsf{tid}(e)\,.$$

Thus from the operational semantics and the generation of T, we know

$$|T| = \omega\ \wedge\ \exists \mathsf{t} \in [1..\mathsf{tnum}(T)].\ |(T|_{\mathsf{t}})| \neq \omega \wedge \mathsf{last}(T|_{\mathsf{t}}) \neq (\mathsf{t}, \mathbf{term})\,.$$

Thus $\neg\mathsf{fair}(T)$ holds.
(2) is trivial from definition.
5. The first equivalence is trivial from definition. For the second equivalence, we only need to prove the following:

$$\mathsf{non\text{-}sched}(T) \wedge \neg \mathsf{prog\text{-}t}(T) \implies \neg \mathsf{fair}(T)\,.$$

It has been proved in the proofs for the equivalences for deadlock-freedom. □

From Lemma 37, we can get most of the implications in the lattice of Figure 1. To prove the remaining implications on sequential termination, we first prove some equivalences in the sequential setting below.

Lemma 38 (Equivalences in Sequential Setting). *For any C_1, σ_c, σ_o and T, if $T \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } C_1), (\sigma_c, \sigma_o, \circledcirc)]\!]$, then*

1. $\mathsf{fair}(T)$ *and* $\mathsf{iso}(T)$ *holds;*
2. $\mathsf{lock\text{-}free}(T) \iff \mathsf{wait\text{-}free}(T) \iff \mathsf{obstruction\text{-}free}(T) \iff \mathsf{deadlock\text{-}free}(T) \iff \mathsf{starvation\text{-}free}(T)$.

Proof. 1. Since $T \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } C_1), (\sigma_c, \sigma_o, \circledcirc)]\!]$, by the operational semantics we know $T(1) = (\textbf{spawn}, 1)$ and

$$\forall i.\ 2 \le i \le |T| \implies \mathsf{tid}(T(i)) = 1\,.$$

If $|T| = \omega$, we know $|(T|_1)| = |T| = \omega$. Thus $\mathsf{fair}(T)$ and $\mathsf{iso}(T)$.

2. By Lemma 37 and the above case. □

From Lemmas 37 and 38, we get the following theorem.

Theorem 39 (Figure 1).

1. $\mathsf{wait\text{-}free}_\varphi(\Pi) \implies \mathsf{lock\text{-}free}_\varphi(\Pi)$;
2. $\mathsf{wait\text{-}free}_\varphi(\Pi) \implies \mathsf{starvation\text{-}free}_\varphi(\Pi)$;
3. $\mathsf{lock\text{-}free}_\varphi(\Pi) \implies \mathsf{obstruction\text{-}free}_\varphi(\Pi)$;
4. $\mathsf{lock\text{-}free}_\varphi(\Pi) \implies \mathsf{deadlock\text{-}free}_\varphi(\Pi)$;
5. $\mathsf{starvation\text{-}free}_\varphi(\Pi) \implies \mathsf{deadlock\text{-}free}_\varphi(\Pi)$;
6. $\mathsf{obstruction\text{-}free}_\varphi(\Pi) \implies \mathsf{seq\text{-}term}_\varphi(\Pi)$;
7. $\mathsf{deadlock\text{-}free}_\varphi(\Pi) \implies \mathsf{seq\text{-}term}_\varphi(\Pi)$.

B.3 Proofs of Theorem 14

Lemma 40 (Finite trace must be lock-free). *For any T, if*

$$T \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!]$$

and $|T| \neq \omega$, then $\mathsf{lock\text{-}free}(T)$ *must hold.*

Proof. Suppose $T = (\textbf{spawn}, n) :: T'$. We know one of the following holds:

(i) $(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T'}{\longmapsto}{}^* \textbf{abort}$; or
(ii) $(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T'}{\longmapsto}{}^* (\textbf{skip}, \_)$.

For either case, we can prove $\mathsf{lock\text{-}free}(T)$ by the operational semantics. □

We define the MGC version of lock-freedom.

Definition 41. $\mathsf{lock\text{-}free}_{\varphi}^{\mathsf{MGC}}(\Pi)$, *iff*

$$\begin{array}{l}\forall n, \sigma_o, T.\ \ T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \wedge (\sigma_o \in dom(\varphi)) \\ \Longrightarrow\ (\exists i.\ \mathsf{is\_obj\_abt}(T(i))) \vee (\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_ret}(T(j)))\end{array}$$

We use $\mathsf{get\_objevt}(T)$ to project T to the sub-trace of object events (including method invocation, return, object fault, and normal object actions). Thus we know:

$$\forall T, T'.\ (\mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T')) \implies (\mathsf{get\_hist}(T) = \mathsf{get\_hist}(T'))\,.$$

The following lemma is similar to Lemma 32 (MGC is the most general). But here we take into account infinite traces generated by complete executions.

Lemma 42. *For any T, if*

$$T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!],$$

then one of the following holds:

(1) $|T| \neq \omega$; or
(2) there exists i such that $\forall j \geq i.\ \mathsf{is\_clt}(T(j))$; or
(3) there exists T_m such that

$$T_m \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!]\,,$$

and $\mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T_m)$.

Proof. By co-induction over $T \in \mathcal{T}_\omega[\![W, \mathcal{S}]\!]$, where

$$(\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\_, \_, \circledcirc)) \stackrel{\tau}{\longmapsto}{}^* (W, \mathcal{S}) \wedge (W \neq \mathbf{skip})\,.$$

In other words, $(W, \mathcal{S})$ is a "well-formed" configuration. We only need to prove the following (B.7):

for any T, W, $\mathcal{S}$, W_m and $\mathcal{S}_m$, if

(a) $(W, \mathcal{S}) \precsim_{\mathsf{MGC}} (W_m, \mathcal{S}_m)$,
(b) $(W, \mathcal{S}) \stackrel{T}{\longmapsto}{}^{\omega} \cdot$, and
(c) $\forall i.\ \exists j.\ j \geq i \wedge \neg \mathsf{is\_clt}(T(j)) \wedge T(j) \neq (\_, \mathbf{term})$,

then there exists T_m such that $(W_m, \mathcal{S}_m) \stackrel{T_m}{\longmapsto}{}^{\omega} \cdot$ and $\mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T_m)$. (B.7)

Here $\precsim_{\mathsf{MGC}}$ is defined in Figure 9(a). We first prove $\precsim_{\mathsf{MGC}}$ is a simulation:

If $(W, \mathcal{S}) \precsim_{\mathsf{MGC}} (W_m, \mathcal{S}_m)$ and $(W, \mathcal{S}) \stackrel{e}{\longmapsto} (W', \mathcal{S}')$, then
there exist $T, W'_m, \mathcal{S}'_m$ such that $(W_m, \mathcal{S}_m) \stackrel{T}{\longmapsto}{}^* (W'_m, \mathcal{S}'_m)$, (B.8)
$\mathsf{get\_objevt}(e) = \mathsf{get\_objevt}(T)$ and
$(W', \mathcal{S}') \precsim_{\mathsf{MGC}} (W'_m, \mathcal{S}'_m)$.

This is proved by case analysis of e.

- If $e = (\mathsf{t}, \mathbf{out}, n)$ or $e = (\mathsf{t}, \mathbf{clt})$ or $e = (\mathsf{t}, \mathbf{term})$, we know the call stack of the current thread t (which makes the step) is $\circ$, before and after the step. Then we simply let $(W_m, \mathcal{S}_m)$ go zero step, and hence $T = \epsilon$. Thus $\mathsf{get\_objevt}(e) = \mathsf{get\_objevt}(T)$ and we can prove $(W', \mathcal{S}') \precsim_{\mathsf{MGC}} (W_m, \mathcal{S}_m)$.
- If $e = (\mathsf{t}, f_i, n)$, we know the call stack of the thread t is $\circ$ before the step and is (σ_l, x, C') after the step. Then we know the code of t in W_m must be MGT. We let it go two steps. After the first step, the code of t becomes $f_{\mathbf{rand}(m)}(\mathbf{rand}()); \mathsf{MGT}$. We evaluate $\mathbf{rand}(m)$ to i and $\mathbf{rand}()$ to n, and make the second step. Thus the resulting configuration satisfies $(W', \mathcal{S}') \precsim_{\mathsf{MGC}} (W'_m, \mathcal{S}'_m)$, and $T = e$.
- If $e = (\mathsf{t}, \mathbf{ret}, n)$, we know the call stack of the thread t is (σ_l, x, C') before the step and is $\circ$ after the step. Then we let the code of t in W_m go two steps. After the first step, the code of t becomes $\mathbf{skip}; \mathsf{MGT}$. After the second step, we have $(W', \mathcal{S}') \precsim_{\mathsf{MGC}} (W'_m, \mathcal{S}'_m)$. Also we know the first step generates the event e, and thus $\mathsf{get\_objevt}(e) = \mathsf{get\_objevt}(T)$.
- If $e = (\mathsf{t}, \mathbf{obj})$, we know the call stack of the thread t is not $\circ$ before or after the step. We let the code of t in W_m go one step, and hence $T = (\mathsf{t}, \mathbf{obj})$ and $(W', \mathcal{S}') \precsim_{\mathsf{MGC}} (W'_m, \mathcal{S}'_m)$.

Thus we have proved (B.8).

From (B.8), we can prove the following by induction over the steps of T:

$$\begin{array}{l}\text{If } (W, \mathcal{S}) \precsim_{\mathsf{MGC}} (W_m, \mathcal{S}_m), (W, \mathcal{S}) \overset{T}{\longmapsto}{}^{+} (W', \mathcal{S}') \text{ and} \\ (\exists i.\ \neg \mathsf{is\_clt}(T(i)) \wedge T(i) \neq (\_, \mathbf{term})), \text{ then} \\ \text{there exist } T_m, W'_m, \mathcal{S}'_m \text{ such that } (W_m, \mathcal{S}_m) \overset{T_m}{\longmapsto}{}^{+} (W'_m, \mathcal{S}'_m), \\ \mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T_m) \text{ and } (W', \mathcal{S}') \precsim_{\mathsf{MGC}} (W'_m, \mathcal{S}'_m).\end{array}$$

Then we can get (B.7) by co-induction.

When $(W, \mathcal{S}) = (\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc))$, we know $(W, \mathcal{S}) \precsim_{\mathsf{MGC}} (\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n \rfloor, (\emptyset, \sigma_o, \circledcirc))$. Thus we are done. □

We prove that the MGC version is equivalent to the original version of lock-freedom.

Lemma 43. $\mathsf{lock\text{-}free}_{\varphi}(\Pi) \iff \mathsf{lock\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi)$.

Proof. 1. $\mathsf{lock\text{-}free}_{\varphi}(\Pi) \implies \mathsf{lock\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi)$:

We prove the following:

$$\begin{array}{l}\forall n, \sigma_o, T.\ \ T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \wedge (\sigma_o \in dom(\varphi)) \wedge \mathsf{lock\text{-}free}(T) \\ \implies (\exists i.\ \mathsf{is\_obj\_abt}(T(i))) \vee (\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_ret}(T(j)))\end{array} \tag{B.9}$$

We unfold $\mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!]$, then we have three cases:

(1) $(\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n \rfloor, (\emptyset, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^{\omega} \_$

(2) $(\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n \rfloor, (\emptyset, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^{*} (\mathbf{skip}, \_)$

(3) $(\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n \rfloor, (\emptyset, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^{*} \mathbf{abort}$

We know from the operational semantics that (2) is impossible.
For (3), we know from the operational semantics that $\mathsf{last}(T) = (\_, \mathbf{obj}, \mathbf{abort})$.
Thus $\exists i.\ \mathsf{is\_obj\_abt}(T(i))$.
For (1), we prove the following by contradiction:

$$\begin{array}{l}\forall n, \sigma_o, T.\ (\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n \rfloor, (\emptyset, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^{\omega}\ \_ \\ \implies \forall i.\ \exists j.\ j \geq i \wedge (\mathsf{is\_inv}(T(j)) \vee \mathsf{is\_ret}(T(j)) \vee T(j) = (\_, \mathbf{obj}))\end{array} \tag{B.10}$$

Then, $\forall i.\ \exists j.\ j \geq i \wedge (\mathsf{is\_ret}(T(j)) \vee \mathsf{pend\_inv}(T(1..j)) \neq \emptyset)$. Thus by $\mathsf{lock\text{-}free}(T)$, we are done.

2. $\mathsf{lock\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi) \implies \mathsf{lock\text{-}free}_{\varphi}(\Pi)$:
For any $T \in \mathcal{T}_{\omega}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n), (\sigma_c, \sigma_o, \circledcirc)]\!]$, by Lemma 42, we know one of the following holds:
(1) $|T| \neq \omega$; or
(2) there exists i such that $\forall j \geq i.\ \mathsf{is\_clt}(T(j))$; or
(3) there exists T_m such that
$$T_m \in \mathcal{T}_{\omega}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!],$$
and $\mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T_m)$.
For (1), by Lemma 40, we know $\mathsf{lock\text{-}free}(T)$.
For (2), we know $\mathsf{lock\text{-}free}(T)$ holds immediately by definition.
For (3), from $\mathsf{lock\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi)$, we know

$$(\exists i.\ \mathsf{is\_obj\_abt}(T_m(i))) \vee (\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_ret}(T_m(j))).$$

Thus we have:

$$(\exists i.\ \mathsf{is\_obj\_abt}(T(i))) \vee (\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_ret}(T(j))).$$

If $\exists i.\ \mathsf{is\_obj\_abt}(T(i))$, we know $\mathsf{lock\text{-}free}(T)$. Otherwise, we know

$$\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_ret}(T(j)).$$

Thus, for any i, if $\mathsf{pend\_inv}(T(1..i)) \neq \emptyset$, then there exists $j > i$ such that $\mathsf{is\_ret}(T(j))$. Therefore $\mathsf{lock\text{-}free}(T)$ and we are done.

□

Then, we only need to prove the following (B.11), (B.12) and (B.13):

$$\Pi \sqsubseteq^{\omega}_{\varphi} \Pi_A \implies \Pi \sqsubseteq_{\varphi} \Pi_A \tag{B.11}$$

$$\Pi \sqsubseteq^{\omega}_{\varphi} \Pi_A \implies \mathsf{lock\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi) \tag{B.12}$$

$$\Pi \sqsubseteq_{\varphi} \Pi_A \wedge \mathsf{lock\text{-}free}_{\varphi}(\Pi) \implies \Pi \sqsubseteq^{\omega}_{\varphi} \Pi_A \tag{B.13}$$

Proofs of (B.11) For any n, C_1, ..., C_n, σ_c, σ_o and σ_a such that $\varphi(\sigma_o) = \sigma_a$, for any T if

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

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

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

Thus there exists T_1' and T_1'' such that $T_1'' = T_1 :: T_1'$ and one of the following holds:

(i) $(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T_1''}{\longmapsto}{}^{\omega} \cdot$; or
(ii) $(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T_1''}{\longmapsto}{}^{*} (\textbf{skip}, \_)$; or
(iii) $(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T_1''}{\longmapsto}{}^{*} \textbf{abort}$.

That is,

$$T_1'' \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!]\,.$$

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

$$T_2'' \in \mathcal{T}_\omega[\![(\textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a, \circledcirc)]\!]\,,$$

and

$$\mathsf{get\_obsv}(T_2'') = \mathsf{get\_obsv}(T_1'') = T :: \mathsf{get\_obsv}(T_1')\,.$$

Thus there exists T_2 such that

$$T_2 \in \mathcal{T}[\![(\textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a, \circledcirc)]\!]\,,$$

and $\mathsf{get\_obsv}(T_2) = T$. Thus

$$T \in \mathcal{O}[\![(\textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a, \circledcirc)]\!]\,,$$

and we are done.

Proofs of (B.12) We construct another most general client as follows:

$$\mathsf{MGTp1} \overset{\text{def}}{=} \textbf{while } (\textbf{true})\{\ f_{\textbf{rand}(m)}(\textbf{rand}());\ \textbf{print}(1);\ \}$$
$$\mathsf{MGCp1}_n \overset{\text{def}}{=} \|_{i \in [1..n]}\, \mathsf{MGTp1}$$

The following lemma describes the relationship between MGCp1 and MGC:

Lemma 44. *(1) For any T, if*

$$T \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!],$$

then there exists T_p such that

$$T_p \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGCp1}_n), (\emptyset, \sigma_o, \circledcirc)]\!],$$

$T_p\backslash(\_, \mathbf{out}, 1) = T$ *and*

$$\forall i, \mathsf{t}.\ T_p(i) = (\mathsf{t}, \mathbf{ret}, \_) \ \Leftrightarrow\ T_p(i+1) = (\mathsf{t}, \mathbf{out}, 1)\,.$$

(2) For any T_p, if

$$T_p \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGCp1}_n), (\emptyset, \sigma_o, \circledcirc)]\!],$$

then there exists T such that

$$T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!]$$

and $T_p\backslash(\_, \mathbf{out}, 1) = T$.

Here we use $T_p\backslash(\_, \mathbf{out}, 1)$ to mean a sub-trace of T_p which removes all the events of the form $(\_, \mathbf{out}, 1)$.

Proof. By constructing simulations between executions of **let** Π **in** MGC_n and **let** Π **in** $\mathsf{MGCp1}_n$. □

Lemma 45. *Suppose Π_A is total.*
For any n, σ_a and T, if $T \in \mathcal{O}_\omega[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ \mathsf{MGCp1}_n), (\emptyset, \sigma_a, \circledcirc)]\!]$, then T is an infinite trace of $(\_, \mathbf{out}, 1)$.

Proof. We need to prove: for any T such that
$T \in \mathcal{O}_\omega[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ \mathsf{MGCp1}_n), (\emptyset, \sigma_a, \circledcirc)]\!]$, the following hold:

(1) $|T| = \omega$;
(2) for any i, $T(i) = (\_, \mathbf{out}, 1)$.

For (1): we can prove for any T' such that

$$T' \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ \mathsf{MGCp1}_n), (\emptyset, \sigma_a, \circledcirc)]\!],$$

we have $|T'| = \omega$. If $|T| \neq \omega$, we know there exists i such that

$$\forall j \geq i.\ \mathsf{is\_inv}(T'(j)) \vee \mathsf{is\_ret}(T'(j)) \vee T'(j) = (\_, \mathbf{obj}) \vee T'(j) = (\_, \mathbf{clt}).$$

Since Π_A is total, from the code and the operational semantics, we know this is impossible.

(2) is easily proved from $|T| = \omega$ and that the code can only produce $(\_, \mathbf{out}, 1)$ as observable events. □

To prove $\mathsf{lock\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi)$, we want to show: for any n, σ_o, σ_a and T, if $T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!]$ and $\varphi(\sigma_o) = \sigma_a$, then

$$(\exists i.\ \mathsf{is\_obj\_abt}(T(i))) \vee (\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_ret}(T(j))) \tag{B.14}$$

First, if $T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!]$, by Lemma 44(1), there exists T_p such that $T_p \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGCp1}_n), (\emptyset, \sigma_o, \circledcirc)]\!]$ and $T_p\backslash(\_, \mathbf{out}, 1) = T$.

Since $\Pi \sqsubseteq^{\omega}_{\varphi} \Pi_A$, we know

$$\mathcal{O}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGCp1}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{O}_\omega[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ \mathsf{MGCp1}_n), (\emptyset, \sigma_a, \circledcirc)]\!]\,.$$

From Lemma 45, we know for any T, if $T \in \mathcal{O}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGCp1}_n), (\emptyset, \sigma_o, \circledcirc)]\!]$, then T is an infinite trace of $(\_, \mathbf{out}, 1)$.

Then we know: $\mathsf{get\_obsv}(T_p)$ is an infinite trace of $(\_, \mathbf{out}, 1)$.
Thus $|T_p| = \omega$ and

$$\forall i.\ \exists j.\ j \geq i \wedge T_p(j) = (\_, \mathbf{out}, 1)\,. \tag{B.15}$$

We prove the following:

$$\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_ret}(T_p(j))\,. \tag{B.16}$$

This is proved as follows. From $|T_p| = \omega$ and (B.15), we know for any i, there exist $j_1, \ldots, j_{n+1}$ such that $i \leq j_1 < \ldots < j_{n+1}$ and $\forall k \in [1..n+1]$. $T_p(j_k) = (\_, \mathbf{out}, 1)$. Then, by the pigeonhole principle, we know there exists a thread t producing two $(\mathsf{t}, \mathbf{out}, 1)$-s. Suppose j_k and j_l are the indexes of the two events produced by t and $j_k < j_l$. By the operational semantics, we know there exists j' such that $i \leq j_k < j' < j_l$ and $\mathsf{is\_ret}(T_p(j'))$. Thus we have proved (B.16).

Since $T_p \backslash (\_, \mathbf{out}, 1) = T$, from (B.16), we know (B.14) holds and we are done.

Proofs of (B.13) We need to prove that if $\Pi \sqsubseteq_\varphi \Pi_A$ and $\mathsf{lock\text{-}free}_\varphi(\Pi)$, then for any n, C_1, ..., C_n, σ_c, σ_o and σ_a such that $\varphi(\sigma_o) = \sigma_a$, we have

$$\begin{array}{l}\mathcal{O}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!] \\ \subseteq \mathcal{O}_\omega[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a, \circledcirc)]\!]\,.\end{array}$$

Thus we only need to prove: for any T,

(1) If $(\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^* \mathbf{abort}$,
then there exists T_a such that
$(\lfloor \mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \overset{T_a}{\longmapsto}{}^* \mathbf{abort}$ and
$\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$.

(2) If $(\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^* (\mathbf{skip}, \_)$,
then there exists T_a such that
$(\lfloor \mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \overset{T_a}{\longmapsto}{}^* (\mathbf{skip}, \_)$ and
$\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$.

(3) If $(\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^\omega \cdot$,
then there exists T_a such that
$(\lfloor \mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \overset{T_a}{\longmapsto}{}^\omega \cdot$ and
$\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$.

Actually neither (1) or (2) depends on progress properties. We can prove the following lemma.

Lemma 46. *If $\Pi \sqsubseteq_\varphi \Pi_A$, then for any n, C_1, ..., C_n, σ_c, σ_o, σ_a and T such that $\varphi(\sigma_o) = \sigma_a$, we have*

1. *If* $(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^* \textbf{abort}$, *then there exists* T_a *such that* $(\lfloor \textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \overset{T_a}{\longmapsto}{}^* \textbf{abort}$ *and* $\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$.
2. *If* $(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^* (\textbf{skip}, \_)$, *then there exists* T_a *such that* $(\lfloor \textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \overset{T_a}{\longmapsto}{}^* (\textbf{skip}, \_)$ *and* $\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$.

Proof. 1. We know $\mathsf{is\_abt}(\mathsf{last}(T))$. By $\Pi \sqsubseteq_\varphi \Pi_A$, we know there exists T_a such that

$$T_a \in \mathcal{T}[\![(\textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a, \circledcirc)]\!]$$

and $\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$. Thus $\mathsf{is\_abt}(\mathsf{last}(T_a))$, and by the operational semantics, we know

$$(\lfloor \textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \overset{T_a}{\longmapsto}{}^* \textbf{abort},$$

and we are done.

2. (a) If $n = 1$, we know

$$(\textbf{let } \Pi \textbf{ in } \{C; \textbf{end}\}, (\sigma_c, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^* (\textbf{skip}, \_).$$

Thus there exists T'' such that $T = T'' :: (1, \textbf{term})$. Let

$$T' = T'' :: (1, \textbf{clt}) :: (1, \textbf{out}, \text{“done”}) :: (1, \textbf{clt}) :: (1, \textbf{term}),$$

where we assume $(1, \textbf{out}, \text{“done”})$ is different from all the events in T, then

$$(\textbf{let } \Pi \textbf{ in } \{C; \textbf{print}(\text{“done”}); \textbf{end}\}, (\sigma_c, \sigma_o, \circledcirc)) \overset{T'}{\longmapsto}{}^* (\textbf{skip}, \_).$$

Since $\Pi \sqsubseteq_\varphi \Pi_A$, we know there exists T'_a such that

$$T'_a \in \mathcal{T}[\![(\textbf{let } \Pi_A \textbf{ in } \{C; \textbf{print}(\text{“done”}); \textbf{end}\}), (\sigma_c, \sigma_a, \circledcirc)]\!]$$

and $\mathsf{get\_obsv}(T') = \mathsf{get\_obsv}(T'_a)$. Thus we know there exists T''_a such that

$$T'_a = T''_a :: (1, \textbf{out}, \text{“done”}) :: (1, \textbf{clt}) :: (1, \textbf{term}),$$

and by the operational semantics, we know there exists T_a such that $T''_a = T_a :: (1, \textbf{clt})$ and

$$(\textbf{let } \Pi_A \textbf{ in } \{C; \textbf{end}\}, (\sigma_c, \sigma_a, \circledcirc)) \overset{T_a :: (1, \textbf{term})}{\longmapsto}{}^* (\textbf{skip}, \_).$$

Also we have $\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$.

(b) If $n > 1$, we construct another program $\textbf{let } \Pi \textbf{ in } C'_1 \,\|\, \ldots \,\|\, C'_n$ as follows: we pick $n - 1$ fresh variables: $\mathtt{d}_2, \ldots, \mathtt{d}_n$,

$$C'_1 = (C_1; \textbf{if } (\mathtt{d}_2 \&\& \ldots \&\& \mathtt{d}_n) \textbf{ print}(\text{“done”});)$$
$$C'_i = (C_i; \mathtt{d}_i := \textbf{true}) \qquad \forall i \in [2..n]$$

and also let

$$\sigma'_c = \sigma_c \uplus \{\mathtt{d}_2 \rightsquigarrow \textbf{false}, \ldots, \mathtt{d}_n \rightsquigarrow \textbf{false}\}.$$

Then, if

$$(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^* (\textbf{skip}, \_),$$

let T'' be the result after removing all the termination markers in T, and

$$T' = T'' :: (2, \mathbf{clt}) :: (2, \mathbf{clt}) :: \ldots :: (n, \mathbf{clt}) :: (n, \mathbf{clt}) \\ :: (1, \mathbf{clt}) :: (1, \mathbf{clt}) :: (1, \mathbf{out}, \text{“done”}) \\ :: (1, \mathbf{clt}) :: (1, \mathbf{term}) :: \ldots :: (n, \mathbf{clt}) :: (n, \mathbf{term})$$

where we still assume $(1, \mathbf{out}, \text{“done”})$ is different from all the events in T, we can prove:

$$(\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ C'_1 \parallel \ldots \parallel C'_n \rfloor, (\sigma'_c, \sigma_o, \circledcirc)) \overset{T'}{\longmapsto}{}^* (\mathbf{skip}, \_).$$

Since $\Pi \sqsubseteq_\varphi \Pi_A$, we know there exists T'_a such that

$$T'_a \in \mathcal{T}[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ C'_1 \parallel \ldots \parallel C'_n), (\sigma'_c, \sigma_a, \circledcirc)]\!]$$

and $\mathsf{get\_obsv}(T') = \mathsf{get\_obsv}(T'_a)$. Thus we know there exists i such that $T'_a(i) = (1, \mathbf{out}, \text{“done”})$. Then we know

$$(\lfloor \mathbf{let}\ \Pi_A\ \mathbf{in}\ C'_1 \parallel \ldots \parallel C'_n \rfloor, (\sigma'_c, \sigma_a, \circledcirc)) \overset{T'_a}{\longmapsto}{}^* (\mathbf{skip}, \_).$$

We can remove all the actions of the newly added commands, construct a simulation between the two executions, and prove: there exists T_a such that

$$(\lfloor \mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \overset{T_a}{\longmapsto}{}^* (\mathbf{skip}, \_),$$

and $\mathsf{get\_obsv}(T_a) = \mathsf{get\_obsv}(T''_a) = \mathsf{get\_obsv}(T)$.

Thus we are done. □

For (3), we define the simulation relation $\precsim$ in Figure 9(d), and prove the following (B.17) by case analysis and the operational semantics:

For any W_1, $\mathcal{S}_1$, W_2, $\mathcal{S}_2$, W_3, $\mathcal{S}_3$ and e_1,
if $(W_1, \mathcal{S}_1) \precsim (W_2, \mathcal{S}_2; W_3, \mathcal{S}_3)$ and $(W_1, \mathcal{S}_1) \overset{e_1}{\longmapsto} (W'_1, \mathcal{S}'_1)$,
then there exist T_2, W'_2, $\mathcal{S}'_2$, T_3, W'_3 and $\mathcal{S}'_3$ such that
$(W_2, \mathcal{S}_2) \overset{T_2}{\longmapsto}{}^* (W'_2, \mathcal{S}'_2)$, $(W_3, \mathcal{S}_3) \overset{T_3}{\longmapsto}{}^* (W'_3, \mathcal{S}'_3)$,
$T_3 \backslash (\_, \mathbf{obj}) = e_1 \backslash (\_, \mathbf{obj})$ and $(W'_1, \mathcal{S}'_1) \precsim (W'_2, \mathcal{S}'_2; W'_3, \mathcal{S}'_3)$. (B.17)

With (B.17), we can prove the following (B.18) by induction over the length of T_1:

For any W_1, $\mathcal{S}_1$, W_2, $\mathcal{S}_2$, W_3, $\mathcal{S}_3$ and T_1,
if $(W_1, \mathcal{S}_1) \precsim (W_2, \mathcal{S}_2; W_3, \mathcal{S}_3)$, $(W_1, \mathcal{S}_1) \overset{T_1}{\longmapsto}{}^+ (W'_1, \mathcal{S}'_1)$ and
$\mathsf{last}(T_1) \neq (\_, \mathbf{obj})$,
then there exist T_2, W'_2, $\mathcal{S}'_2$, T_3, W'_3 and $\mathcal{S}'_3$ such that
$(W_2, \mathcal{S}_2) \overset{T_2}{\longmapsto}{}^* (W'_2, \mathcal{S}'_2)$, $(W_3, \mathcal{S}_3) \overset{T_3}{\longmapsto}{}^+ (W'_3, \mathcal{S}'_3)$,
$T_1 \backslash (\_, \mathbf{obj}) = T_3 \backslash (\_, \mathbf{obj})$ and $(W'_1, \mathcal{S}'_1) \precsim (W'_2, \mathcal{S}'_2; W'_3, \mathcal{S}'_3)$. (B.18)

With (B.18), we can prove the following (B.19):

For any W, $\mathcal{S}$, W_1, $\mathcal{S}_1$, W_2, $\mathcal{S}_2$, W_3, $\mathcal{S}_3$, T_0 and T_1,
if $(W, \mathcal{S})$ is well-formed and out of method calls, $(W, \mathcal{S}) \overset{T_0}{\longmapsto}{}^* (W_1, \mathcal{S}_1)$,
$(W_1, \mathcal{S}_1) \precsim (W_2, \mathcal{S}_2; W_3, \mathcal{S}_3)$, $(W_1, \mathcal{S}_1) \overset{T_1}{\longmapsto}{}^\omega \cdot$ and $\mathsf{lock\text{-}free}(T_0 :: T_1)$,
then there exists T_3 such that $(W_3, \mathcal{S}_3) \overset{T_3}{\longmapsto}{}^\omega \cdot$ and
$T_1 \backslash (\_, \mathbf{obj}) = T_3 \backslash (\_, \mathbf{obj})$. (B.19)

We prove (B.19) as follows. Let $T = T_0 :: T_1$. Since $\mathsf{lock\text{-}free}(T)$, we know one of the following holds:

(i) there exists i such that $\forall j \geq i.\ \mathsf{is\_clt}(T(j))$; or
(ii) for any i, if $\mathsf{pend\_inv}(T(1..i)) \neq \emptyset$, then there exists $j > i$ such that $\mathsf{is\_ret}(T(j))$.

For (i), we know there exist W_1', $\mathcal{S}_1'$, T_1' and T_1'' such that

$$(W_1, \mathcal{S}_1) \xmapsto{T_1'}{}^{+} (W_1', \mathcal{S}_1'), \quad (W_1', \mathcal{S}_1') \xmapsto{T_1''}{}^{\omega} \cdot,$$
$$T_1 = T_1' :: T_1'', \quad T_1' = T_1(1..i), \quad \mathsf{is\_clt}(\mathsf{last}(T_1')), \quad \forall j.\ \mathsf{is\_clt}(T_1''(j)).$$

By (B.18), we know: there exist T_2, W_2', $\mathcal{S}_2'$, T_3', W_3' and $\mathcal{S}_3'$ such that $(W_2, \mathcal{S}_2) \xmapsto{T_2}{}^{*} (W_2', \mathcal{S}_2')$, $(W_3, \mathcal{S}_3) \xmapsto{T_3'}{}^{+} (W_3', \mathcal{S}_3')$, $T_1' \backslash (\_, \mathbf{obj}) = T_3' \backslash (\_, \mathbf{obj})$ and $(W_1', \mathcal{S}_1') \precsim (W_2', \mathcal{S}_2'; W_3', \mathcal{S}_3')$. Then by coinduction over T_1 and from (B.18), we get: there exists T_3'' such that

$$(W_3', \mathcal{S}_3') \xmapsto{T_3''}{}^{\omega} \cdot \text{ and } T_1'' \backslash (\_, \mathbf{obj}) = T_3'' \backslash (\_, \mathbf{obj}).$$

Let $T_3 = T_3' :: T_3''$, and we know

$$(W_3, \mathcal{S}_3) \xmapsto{T_3}{}^{\omega} \cdot \text{ and } T_1 \backslash (\_, \mathbf{obj}) = T_3 \backslash (\_, \mathbf{obj}).$$

Suppose (i) does not hold. Thus we know

$$\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_obj}(T(j)).$$

By the operational semantics, we know

$$\forall i.\ \exists j.\ j \geq i \wedge \mathsf{pend\_inv}(T(1..j)) \neq \emptyset.$$

Since (ii) holds, we know

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

Then by coinduction and from (B.18), we know there exists T_3 such that

$$(W_3, \mathcal{S}_3) \xmapsto{T_3}{}^{\omega} \cdot \text{ and } T_1 \backslash (\_, \mathbf{obj}) = T_3 \backslash (\_, \mathbf{obj}).$$

Thus we have proved (B.19). On the other hand, for any n, C_1, ..., C_n, σ_c, σ_o and σ_a, by Lemma 32, we know

$$\mathcal{H}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{H}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!].$$

From $\Pi \sqsubseteq_\varphi \Pi_A$, by Lemma 34, we know $\Pi \subseteq_\varphi \Pi_A$. Thus, if $\varphi(\sigma_o) = \sigma_a$, then

$$\mathcal{H}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{H}[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!].$$

Then we know

$$\begin{array}{l}(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \sigma_o, \circledcirc)) \\ \precsim (\mathbf{let}\ \Pi_A\ \mathbf{in}\ \mathsf{MGC}_n, (\emptyset, \sigma_a, \circledcirc); \\ \quad \mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \sigma_a, \circledcirc)),\end{array}$$

Thus, if $(\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \xmapsto{T}{}^{\omega} \cdot$, by $\mathsf{lock\text{-}free}_\varphi(\Pi)$, we know $\mathsf{lock\text{-}free}(T)$. Then from (B.19) we get: there exists T_a such that

$$(\lfloor \mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \xmapsto{T_a}{}^{\omega} \cdot$$

and $T \backslash (\_, \mathbf{obj}) = T_a \backslash (\_, \mathbf{obj})$. Thus $\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$ and we are done.

B.4 Proofs of Theorem 10

Similar to Lemma 40, we can prove the following lemma.

Lemma 47 (Finite trace must be wait-free). *For any T, if*

$$T \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } C_1 \parallel \ldots \parallel C_n), (\sigma_c, \sigma_o, \circledcirc)]\!]$$

and $|T| \neq \omega$, then $\mathsf{wait\text{-}free}(T)$ *must hold.*

We define the MGC version of wait-freedom, and prove it is equivalent to the original version.

Definition 48. $\mathsf{wait\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi)$, *iff*

$$\begin{array}{l}\forall n, \sigma_o, T.\ \ T \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \wedge (\sigma_o \in dom(\varphi)) \\ \Longrightarrow \mathsf{wait\text{-}free}(T)\end{array}$$

Lemma 49. $\mathsf{wait\text{-}free}_{\varphi}(\Pi) \iff \mathsf{wait\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi)$.

Proof. 1. $\mathsf{wait\text{-}free}_{\varphi}(\Pi) \implies \mathsf{wait\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi)$:
Trivial.

2. $\mathsf{wait\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi) \implies \mathsf{wait\text{-}free}_{\varphi}(\Pi)$:
For any $T \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } C_1 \parallel \ldots \parallel C_n), (\sigma_c, \sigma_o, \circledcirc)]\!]$, by Lemma 42, we know one of the following holds:
(1) $|T| \neq \omega$; or
(2) there exists i such that $\forall j \geq i.\ \mathsf{is\_clt}(T(j))$; or
(3) there exists T_m such that
$$T_m \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!]\,,$$
and $\mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T_m)$.
For (1), by Lemma 47, we know $\mathsf{wait\text{-}free}(T)$ holds.
For (2), we know $|T| = \omega$.
For any k and e, if $e \in \mathsf{pend\_inv}(T(1..k))$, we know one of the following must hold:
(i) $\exists j.\ j > k \wedge \mathsf{match}(e, T(j))$.
(ii) $\forall j.\ j > k \Rightarrow \neg\mathsf{match}(e, T(j))$. Thus we can prove:
$$\forall j \geq k.\ e \in \mathsf{pend\_inv}(T(1..j)).$$
Let $l = \mathsf{max}(i, k)$. Then we know:
$$\forall j \geq l.\ \mathsf{is\_clt}(T(j)) \wedge e \in \mathsf{pend\_inv}(T(1..j)).$$
Thus by the operational semantics, we can prove:
$$\forall j > l.\ \mathsf{tid}(T(j)) \neq \mathsf{tid}(e).$$
Thus we know $\mathsf{wait\text{-}free}(T)$.
For (3), suppose (1) does not hold for T, and we only need to prove the following:
for any i and e, if $e \in \mathsf{pend\_inv}(T(1..i))$, then there exists $j > i$ such that either $\forall k \geq j.\ \mathsf{tid}(T(k)) \neq \mathsf{tid}(e)$ or $\mathsf{match}(e, T(j))$.
From $\mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T_m)$, we know

$$\neg\exists i.\ \mathsf{is\_obj\_abt}(T_m(i)).$$

Then by the operational semantics and the generation of T_m, we know

$$\neg\exists i.\ \mathsf{is\_abt}(T_m(i)).$$

From $\mathsf{wait\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi)$, we know $\mathsf{wait\text{-}free}(T_m)$, then we have
for any i and e, if $e \in \mathsf{pend\_inv}(T_m(1..i))$, then there exists $j > i$ such that either $\forall k \geq j.\ \mathsf{tid}(T_m(k)) \neq \mathsf{tid}(e)$ or $\mathsf{match}(e, T_m(j))$.
For any i and e, if $e \in \mathsf{pend\_inv}(T(1..i))$, since $\mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T_m)$, we know there exists i_m such that

$$e \in \mathsf{pend\_inv}(T_m(1..i_m)) \text{ and } \mathsf{get\_objevt}(T(1..i)) = \mathsf{get\_objevt}(T_m(1..i_m)).$$

We know there exists $j_m > i_m$ such that one of the following holds:
(i) $\mathsf{match}(e, T_m(j_m))$; or
(ii) $\forall k \geq j_m.\ \mathsf{tid}(T_m(k)) \neq \mathsf{tid}(e)$.
For (i), since $\mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T_m)$, we know there exists $j > i$ such that $\mathsf{match}(e, T(j))$.
For (ii), suppose

$$\forall j > i.\ \neg\mathsf{match}(e, T(j)) \text{ and } \forall j > i.\ \exists k \geq j.\ \mathsf{tid}(T(k)) = \mathsf{tid}(e)\,.$$

Since $e \in \mathsf{pend\_inv}(T(1..i))$, by the operational semantics, we know

$$\forall j > i.\ \exists k \geq j.\ T(k) = (\mathsf{tid}(e), \mathbf{obj})\,.$$

Since $\mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T_m)$, we know

$$\forall j > i_m.\ \exists k \geq j.\ T_m(k) = (\mathsf{tid}(e), \mathbf{obj})\,,$$

which contradicts (ii). Thus we get $\mathsf{wait\text{-}free}(T)$ and we are done.

□

Then, we only need to prove the following (B.20), (B.21) and (B.22):

$$\Pi \sqsubseteq^{t\omega}_{\varphi} \Pi_A \implies \Pi \sqsubseteq^{\omega}_{\varphi} \Pi_A \tag{B.20}$$

$$\Pi \sqsubseteq^{t\omega}_{\varphi} \Pi_A \implies \mathsf{wait\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi) \tag{B.21}$$

$$\Pi \sqsubseteq_{\varphi} \Pi_A \wedge \mathsf{wait\text{-}free}_{\varphi}(\Pi) \implies \Pi \sqsubseteq^{t\omega}_{\varphi} \Pi_A \tag{B.22}$$

Proofs of (B.20) For any n, C_1, …, C_n, σ_c, σ_o and σ_a such that $\varphi(\sigma_o) = \sigma_a$, for any T, suppose

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

Since $\Pi \sqsubseteq^{t\omega}_{\varphi} \Pi_A$, we know there exists T_a such that

$$T_a \in \mathcal{T}_{\omega}[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a, \circledcirc)]\!]\,,$$
$$\mathsf{get\_obsv}(T_a) = \mathsf{get\_obsv}(T) \text{ and } \mathsf{div\_tids}(T_a) = \mathsf{div\_tids}(T)\,.$$

Thus we are done.

Proofs of (B.21) Just like the proofs of (B.12), we use the most general client MGCp1. We first prove the following lemma:

Lemma 50. *Suppose Π_A is total.*
For any n, σ_a, T and S, if $(T, S) \in \mathcal{O}_{t\omega}[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGCp1}_n), (\emptyset, \sigma_a, \circledcirc)]\!]$, then $\mathsf{div\_tids}(T) = S$.

Proof. We know there exists T_1 such that

$$T_1 \in \mathcal{T}_\omega[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGCp1}_n), (\emptyset, \sigma_a, \circledcirc)]\!],$$
$$T = \mathsf{get\_obsv}(T_1) \text{ and } S = \mathsf{div\_tids}(T_1).$$

It's easy to see that $\mathsf{div\_tids}(T) \subseteq S$.
On the other hand, for all $\mathsf{t} \in S$, we know:

$$\forall i.\ \exists j.\ j \geq i \wedge \mathsf{tid}(T_1(j)) = \mathsf{t}\,.$$

By the operational semantics and the generation of T_1, we know

$$\forall i.\ \exists j.\ j \geq i \wedge T_1(j) = (\mathsf{t}, \textbf{out}, 1)\,.$$

Thus we can prove:

$$\forall i.\ \exists j.\ j \geq i \wedge \mathsf{tid}(T(j)) = \mathsf{t}\,.$$

Thus $\mathsf{t} \in \mathsf{div\_tids}(T)$, and we are done. □

For any n, σ_o, σ_a and T such that $\varphi(\sigma_o) = \sigma_a$, if

$$T \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!],$$

by Lemma 44(1), there exists T_p such that

$$T_p \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGCp1}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \text{ and } T_p \backslash (\_, \textbf{out}, 1) = T.$$

Suppose $\neg\exists i.\ \mathsf{is\_abt}(T(i))$.
Then for any i and e, if $e \in \mathsf{pend\_inv}(T(1..i))$, we know there exists i_p such that

$$e \in \mathsf{pend\_inv}(T_p(1..i_p)) \text{ and } (T_p(1..i_p)) \backslash (\_, \textbf{out}, 1) = T(1..i).$$

Let $\mathsf{t} = \mathsf{tid}(e)$, we suppose

$$\forall j > i.\ \exists k \geq j.\ \mathsf{tid}(T(k)) = \mathsf{tid}(e) = \mathsf{t}\,.$$

Since $T_p \backslash (\_, \textbf{out}, 1) = T$, we know:

$$\forall j > i_p.\ \exists k \geq j.\ \mathsf{tid}(T_p(k)) = \mathsf{t}\,.$$

Thus we know

$$\mathsf{t} \in \mathsf{div\_tids}(T_p)\,.$$

On the other hand, since $\Pi \sqsubseteq^{t\omega}_{\varphi} \Pi_A$, we know:

$$\mathcal{O}_{t\omega}[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGCp1}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{O}_{t\omega}[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGCp1}_n), (\emptyset, \sigma_a, \circledcirc)]\!]\,.$$

Then from Lemma 50, we know

$$\mathsf{div\_tids}(T_p) = \mathsf{div\_tids}(\mathsf{get\_obsv}(T_p)).$$

Thus

$$\mathsf{t} \in \mathsf{div\_tids}(\mathsf{get\_obsv}(T_p)),$$

and then we can prove:

$$\forall j.\ \exists k \geq j.\ T_p(k) = (\mathsf{t}, \mathbf{out}, 1)\,.$$

Then since $e \in \mathsf{pend\_inv}(T_p(1..i_p))$ and by the operational semantics, we know

$$\text{there must exist } j \text{ such that } j > i_p \text{ and } \mathsf{match}(e, T_p(j)).$$

Since $T_p \backslash (\_, \mathbf{out}, 1) = T$, we know:

$$\text{there exists } j \text{ such that } j > i \text{ and } \mathsf{match}(e, T(j)).$$

Thus $\mathsf{wait\text{-}free}(T)$ and we are done.

Proofs of (B.22) We need to prove that if $\Pi \sqsubseteq_\varphi \Pi_A$ and $\mathsf{wait\text{-}free}_\varphi(\Pi)$, then for any n, C_1, ..., C_n, σ_c, σ_o and σ_a such that $\varphi(\sigma_o) = \sigma_a$, we have

$$\begin{array}{l}\mathcal{O}_{t\omega}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!] \\ \subseteq \mathcal{O}_{t\omega}[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a, \circledcirc)]\!]\,.\end{array}$$

Thus we only need to prove: for any T,

(1) If $(\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^* \mathbf{abort}$,
then there exists T_a such that
$(\lfloor \mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \overset{T_a}{\longmapsto}{}^* \mathbf{abort}$ and
$\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$.

(2) If $(\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^* (\mathbf{skip}, \_)$,
then there exists T_a such that
$(\lfloor \mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \overset{T_a}{\longmapsto}{}^* (\mathbf{skip}, \_)$ and
$\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$.

(3) If $(\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^\omega \cdot$,
then there exists T_a such that
$(\lfloor \mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \overset{T_a}{\longmapsto}{}^\omega \cdot$,
$\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$ and $\mathsf{div\_tids}(T) = \mathsf{div\_tids}(T_a)$.

(1) and (2) are proved in Lemma 46.

For (3), we define the simulation relation $\precsim$ in Figure 9(d), and as in the proof for (B.13), we can get the following (B.23) from (B.19) and the fact that $\mathsf{wait\text{-}free}(T_0 :: T_1)$ implies $\mathsf{lock\text{-}free}(T_0 :: T_1)$:

For any W, $\mathcal{S}$, W_1, $\mathcal{S}_1$, W_2, $\mathcal{S}_2$, W_3, $\mathcal{S}_3$, T_0 and T_1,
if $(W, \mathcal{S})$ is well-formed and out of method calls, $(W, \mathcal{S}) \overset{T_0}{\longmapsto}{}^* (W_1, \mathcal{S}_1)$,
$(W_1, \mathcal{S}_1) \precsim (W_2, \mathcal{S}_2; W_3, \mathcal{S}_3)$, $(W_1, \mathcal{S}_1) \overset{T_1}{\longmapsto}{}^\omega \cdot$ and $\mathsf{wait\text{-}free}(T_0 :: T_1)$,
then there exists T_3 such that $(W_3, \mathcal{S}_3) \overset{T_3}{\longmapsto}{}^\omega \cdot$ and
$T_1 \backslash (\_, \mathbf{obj}) = T_3 \backslash (\_, \mathbf{obj})$.
(B.23)

On the other hand, for any n, C_1, ..., C_n, σ_c, σ_o and σ_a, by Lemma 32, we know

$$\mathcal{H}[\![(\mathbf{let}\ \mathit{\Pi}\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{H}[\![(\mathbf{let}\ \mathit{\Pi}\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] .$$

From $\mathit{\Pi} \sqsubseteq_\varphi \mathit{\Pi}_A$, by Lemma 34, we know $\mathit{\Pi} \subseteq_\varphi \mathit{\Pi}_A$. Thus, if $\varphi(\sigma_o) = \sigma_a$, then

$$\mathcal{H}[\![(\mathbf{let}\ \mathit{\Pi}\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{H}[\![(\mathbf{let}\ \mathit{\Pi}_A\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!] .$$

Then we know

$$\begin{array}{l}(\mathbf{let}\ \mathit{\Pi}\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \sigma_o, \circledcirc)) \\ \precsim (\mathbf{let}\ \mathit{\Pi}_A\ \mathbf{in}\ \mathsf{MGC}_n, (\emptyset, \sigma_a, \circledcirc); \\ \quad \mathbf{let}\ \mathit{\Pi}_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \sigma_a, \circledcirc)),\end{array}$$

Thus, if $(\lfloor \mathbf{let}\ \mathit{\Pi}\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^\omega \cdot$, by $\mathsf{wait\text{-}free}_\varphi(\mathit{\Pi})$, we know $\mathsf{wait\text{-}free}(T)$. Then from (B.23) we get: there exists T_a such that

$$(\lfloor \mathbf{let}\ \mathit{\Pi}_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \overset{T_a}{\longmapsto}{}^\omega \cdot \text{ and } T \backslash (\_, \mathbf{obj}) = T_a \backslash (\_, \mathbf{obj}).$$

Thus we know $\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$.
Below we prove: $\mathsf{div\_tids}(T) = \mathsf{div\_tids}(T_a)$.

(a) $\mathsf{div\_tids}(T) \subseteq \mathsf{div\_tids}(T_a)$:
For any i, since $T \backslash (\_, \mathbf{obj}) = T_a \backslash (\_, \mathbf{obj})$, we know there exists i' such that $T(1..i') \backslash (\_, \mathbf{obj}) = T_a(1..i) \backslash (\_, \mathbf{obj})$. For any $\mathsf{t} \in \mathsf{div\_tids}(T)$, we know

$$\exists j'.\ j' \geq i' \wedge \mathsf{tid}(T(j')) = \mathsf{t} .$$

If $T(j') \neq (\mathsf{t}, \mathbf{obj})$, since $T \backslash (\_, \mathbf{obj}) = T_a \backslash (\_, \mathbf{obj})$, we know there exists $j \geq i$ such that $T_a(j) = T(j')$.
Otherwise, $T(j') = (\mathsf{t}, \mathbf{obj})$. By the operational semantics and the generation of T, we know there exists e such that

$$e \in \mathsf{pend\_inv}(T(1..j'-1)) \text{ and } \mathsf{tid}(e) = \mathsf{t} .$$

Since $\mathsf{wait\text{-}free}(T)$, we know one of the following holds:
(i) there exists $l \geq j'$ such that $\forall k \geq l.\ \mathsf{tid}(T(k)) \neq \mathsf{t}$; or
(ii) there exists $j'' \geq j'$ such that $\mathsf{match}(e, T(j''))$.
Suppose (i) holds. Since $\mathsf{t} \in \mathsf{div\_tids}(T)$, we know

$$\exists j''.\ j'' \geq l \wedge \mathsf{tid}(T(j'')) = \mathsf{t} ,$$

which is a contradiction.
Thus (ii) must hold. Thus $T(j'') = (\mathsf{t}, \mathbf{ret}, \_)$ and $j'' \geq i'$. Since $T\backslash(\_, \mathbf{obj}) = T_a\backslash(\_, \mathbf{obj})$, we know there exists $j \geq i$ such that $T_a(j) = T(j'')$.
Thus we have proved

$$\exists j.\ j \geq i \wedge \mathsf{tid}(T_a(j)) = \mathsf{t}\,.$$

Therefore $\mathsf{t} \in \mathsf{div\_tids}(T_a)$.

(b) $\mathsf{div\_tids}(T_a) \subseteq \mathsf{div\_tids}(T_a)$:
For any i', since $T\backslash(\_, \mathbf{obj}) = T_a\backslash(\_, \mathbf{obj})$, we know there exists i such that $T(1..i')\backslash(\_, \mathbf{obj}) = T_a(1..i)\backslash(\_, \mathbf{obj})$. For any $\mathsf{t} \in \mathsf{div\_tids}(T_a)$, we know

$$\exists j.\ j \geq i \wedge \mathsf{tid}(T_a(j)) = \mathsf{t}\,.$$

If $T_a(j) \neq (\mathsf{t}, \mathbf{obj})$, since $T\backslash(\_, \mathbf{obj}) = T_a\backslash(\_, \mathbf{obj})$, we know there exists $j' \geq i'$ such that $T_a(j) = T(j')$.
Otherwise, $T_a(j) = (\mathsf{t}, \mathbf{obj})$. By the operational semantics and the generation of T_a, we know one of the following holds:
(i) $\forall k > j.\ \mathsf{tid}(T_a(k)) \neq \mathsf{t}$; or
(ii) there exists $j'' \geq j$ such that $\mathsf{match}(e, T_a(j''))$.
Suppose (i) holds. Since $\mathsf{t} \in \mathsf{div\_tids}(T_a)$, we know

$$\exists j''.\ j'' > j \wedge \mathsf{tid}(T_a(j'')) = \mathsf{t}\,,$$

which is a contradiction.
Thus (ii) must hold. Thus $T_a(j'') = (\mathsf{t}, \mathbf{ret}, \_)$ and $j'' \geq i$. Since $T\backslash(\_, \mathbf{obj}) = T_a\backslash(\_, \mathbf{obj})$, we know there exists $j' \geq i'$ such that $T_a(j'') = T(j')$.
Thus we have proved

$$\exists j'.\ j' \geq i' \wedge \mathsf{tid}(T(j')) = \mathsf{t}\,.$$

Therefore $\mathsf{t} \in \mathsf{div\_tids}(T)$.

Thus we are done.

B.5 Proofs of Theorem 18

Lemma 51 (Finite trace must be obstruction-free). *For any T, if*

$$T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!]$$

and $|T| \neq \omega$, then $\mathsf{obstruction\text{-}free}(T)$ *must hold.*

We define the MGC version of obstruction-freedom, and prove it is equivalent to the original version.

Definition 52. $\mathsf{obstruction\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi)$, *iff*

$$\forall n, \sigma_o, T.\ \ T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \wedge \mathsf{iso}(T) \wedge (\sigma_o \in dom(\varphi))$$
$$\Longrightarrow\ (\exists i.\ \mathsf{is\_obj\_abt}(T(i))) \vee (\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_ret}(T(j)))\,.$$

Lemma 53. $\mathsf{obstruction\text{-}free}_{\varphi}(\Pi) \iff \mathsf{obstruction\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi)$.

Proof. From Figure 7, we know $\mathsf{obstruction\text{-}free}_{\varphi}(\Pi)$ is equivalent to the following:

$$\begin{array}{l}\forall n, C_1, \ldots, C_n, \sigma_c, \sigma_o, T.\\ T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o)]\!] \wedge \mathsf{iso}(T) \wedge (\sigma_o \in dom(\varphi))\\ \implies \mathsf{lock\text{-}free}(T)\end{array}$$

By Lemma 43, we know it is equivalent to the following:

$$\begin{array}{l}\forall n, \sigma_o, T.\ \ T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \wedge \mathsf{iso}(T) \wedge (\sigma_o \in dom(\varphi))\\ \implies (\exists i.\ \mathsf{is\_obj\_abt}(T(i))) \vee (\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_ret}(T(j)))\,.\end{array}$$

Thus we are done. □

Then, we only need to prove the following (B.24), (B.25) and (B.26):

$$\Pi \sqsubseteq^{i\omega}_{\varphi} \Pi_A \implies \Pi \sqsubseteq_{\varphi} \Pi_A \tag{B.24}$$

$$\Pi \sqsubseteq^{i\omega}_{\varphi} \Pi_A \implies \mathsf{obstruction\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi) \tag{B.25}$$

$$\Pi \sqsubseteq_{\varphi} \Pi_A \wedge \mathsf{obstruction\text{-}free}_{\varphi}(\Pi) \implies \Pi \sqsubseteq^{i\omega}_{\varphi} \Pi_A \tag{B.26}$$

Proofs of (B.24) For any n, C_1, …, C_n, σ_c, σ_o and σ_a such that $\varphi(\sigma_o) = \sigma_a$, for any T if

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

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

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

Thus there exists T_1' and T_1'' such that $T_1'' = T_1 :: T_1'$, where $\mathsf{iso}(T_1')$ holds, and one of the following holds:

(i) $(\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \xmapsto{T_1''}{}^{\omega} \cdot$; or

(ii) $(\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \xmapsto{T_1''}{}^{*} (\mathbf{skip}, \_)$; or

(iii) $(\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \xmapsto{T_1''}{}^{*} \mathbf{abort}$.

Thus,

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

Since $\Pi \sqsubseteq^{i\omega}_{\varphi} \Pi_A$, we know there exists T_2'' such that

$$T_2'' \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a, \circledcirc)]\!]\,,$$

and

$$\mathsf{get\_obsv}(T_2'') = \mathsf{get\_obsv}(T_1'') = T :: \mathsf{get\_obsv}(T_1')\,.$$

Thus there exists T_2 such that

$$T_2 \in \mathcal{T}[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a, \circledcirc)]\!]\,,$$

and $\mathsf{get\_obsv}(T_2) = T$. Thus

$$T \in \mathcal{O}[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a, \circledcirc)]\!]\,,$$

and we are done.

Proofs of (B.25) The proof is similar to the proof of (B.12).

To prove $\mathsf{obstruction\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi)$, we want to show: for any n, σ_o, σ_a and T, if $T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!]$, $\mathsf{iso}(T)$ and $\varphi(\sigma_o) = \sigma_a$, then the following (B.14) holds:

$$(\exists i.\ \mathsf{is\_obj\_abt}(T(i))) \vee (\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_ret}(T(j)))\,. \tag{B.14}$$

First, if $T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!]$ and $\mathsf{iso}(T)$, by Lemma 44(1), there exists T_p such that

$$\begin{array}{c} T_p \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGCp1}_n), (\emptyset, \sigma_o, \circledcirc)]\!], \quad T_p \backslash (\_, \mathbf{out}, 1) = T \\ \text{and} \quad \forall i, \mathsf{t}.\ T_p(i) = (\mathsf{t}, \mathbf{ret}, \_) \ \Leftrightarrow\ T_p(i+1) = (\mathsf{t}, \mathbf{out}, 1). \end{array}$$

Since $\mathsf{iso}(T)$, we know

$$|T| = \omega \implies \exists \mathsf{t}, i.\ (\forall j.\ j \geq i \implies \mathsf{tid}(T(j)) = \mathsf{t})\,.$$

If $|T_p| = \omega$, by the generation of T_p and $T_p \backslash (\_, \mathbf{out}, 1) = T$, we know $|T| = \omega$. Thus there exist t_0 and i such that

$$\forall j.\ j \geq i \implies \mathsf{tid}(T(j)) = \mathsf{t}_0\,.$$

Since $T_p \backslash (\_, \mathbf{out}, 1) = T$, we know there exists i_p such that

$$\forall j.\ j \geq i_p \implies \mathsf{tid}(T_p(j)) = \mathsf{t}_0 \vee T_p(j) = (\_, \mathbf{out}, 1)\,.$$

By the generation of T_p, we know there exists i' such that

$$\forall j.\ j \geq i' \implies \mathsf{tid}(T_p(j)) = \mathsf{t}_0\,.$$

Thus $\mathsf{iso}(T_p)$ holds.

Since $\Pi \sqsubseteq^{i\omega}_{\varphi} \Pi_A$, we know

$$\mathcal{O}_{i\omega}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGCp1}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{O}_\omega[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ \mathsf{MGCp1}_n), (\emptyset, \sigma_a, \circledcirc)]\!]\,.$$

From Lemma 45, we know for any T, if $T \in \mathcal{O}_{i\omega}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGCp1}_n), (\emptyset, \sigma_o, \circledcirc)]\!]$, then T is an infinite trace of $(\_, \mathbf{out}, 1)$.

Then we know: $\mathsf{get\_obsv}(T_p)$ is an infinite trace of $(\_, \mathbf{out}, 1)$. Thus $|T_p| = \omega$ and the following (B.15) holds:

$$\forall i.\ \exists j.\ j \geq i \wedge T_p(j) = (\_, \mathbf{out}, 1)\,. \tag{B.15}$$

As in the proof of (B.12), we prove the following (B.16) from (B.15):

$$\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_ret}(T_p(j))\,. \tag{B.16}$$

Since $T_p \backslash (\_, \mathbf{out}, 1) = T$, from (B.16), we know (B.14) holds and we are done.

Proofs of (B.26) We need to prove that if $\Pi \sqsubseteq_\varphi \Pi_A$ and $\mathsf{obstruction\text{-}free}_\varphi(\Pi)$, then for any n, C_1, ..., C_n, σ_c, σ_o and σ_a such that $\varphi(\sigma_o) = \sigma_a$, we have

$$\begin{array}{l}\mathcal{O}_{i\omega}[\![(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!] \\ \subseteq \mathcal{O}_{\omega}[\![(\textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a, \circledcirc)]\!] \,.\end{array}$$

Thus we only need to prove: for any T,

(1) If $(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \stackrel{T}{\longmapsto}{}^* \textbf{abort}$,
then there exists T_a such that
$(\lfloor \textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \stackrel{T_a}{\longmapsto}{}^* \textbf{abort}$ and
$\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$.

(2) If $(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \stackrel{T}{\longmapsto}{}^* (\textbf{skip}, \_)$,
then there exists T_a such that
$(\lfloor \textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \stackrel{T_a}{\longmapsto}{}^* (\textbf{skip}, \_)$ and
$\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$.

(3) If $(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \stackrel{T}{\longmapsto}{}^\omega \cdot$ and $\mathsf{iso}(T)$,
then there exists T_a such that
$(\lfloor \textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \stackrel{T_a}{\longmapsto}{}^\omega \cdot$ and
$\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$.

(1) and (2) are proved in Lemma 46.

For (3), as in the proofs for (B.13), we define the simulation relation $\precsim$ in Figure 9(d), and prove the following (B.19):

For any W, $\mathcal{S}$, W_1, $\mathcal{S}_1$, W_2, $\mathcal{S}_2$, W_3, $\mathcal{S}_3$, T_0 and T_1,
if $(W, \mathcal{S})$ is well-formed and out of method calls, $(W, \mathcal{S}) \stackrel{T_0}{\longmapsto}{}^* (W_1, \mathcal{S}_1)$,
$(W_1, \mathcal{S}_1) \precsim (W_2, \mathcal{S}_2; W_3, \mathcal{S}_3)$, $(W_1, \mathcal{S}_1) \stackrel{T_1}{\longmapsto}{}^\omega \cdot$ and $\mathsf{lock\text{-}free}(T_0 :: T_1)$,
then there exists T_3 such that $(W_3, \mathcal{S}_3) \stackrel{T_3}{\longmapsto}{}^\omega \cdot$ and
$T_1 \backslash (\_, \textbf{obj}) = T_3 \backslash (\_, \textbf{obj})$.

On the other hand, for any n, C_1, ..., C_n, σ_c, σ_o and σ_a, by Lemma 32, we know

$$\mathcal{H}[\![(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{H}[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \,.$$

From $\Pi \sqsubseteq_\varphi \Pi_A$, by Lemma 34, we know $\Pi \subseteq_\varphi \Pi_A$. Thus, if $\varphi(\sigma_o) = \sigma_a$, then

$$\mathcal{H}[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{H}[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!] \,.$$

Then we know

$$\begin{array}{l}(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \sigma_o, \circledcirc)) \\ \precsim (\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n, (\emptyset, \sigma_a, \circledcirc); \\ \quad \textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \sigma_a, \circledcirc)),\end{array}$$

Thus, if $(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \stackrel{T}{\longmapsto}{}^\omega \cdot$ and $\mathsf{iso}(T)$,
by $\mathsf{obstruction\text{-}free}_\varphi(\Pi)$, we know $\mathsf{lock\text{-}free}(T)$. Then from (B.19) we get: there exists T_a such that

$$(\lfloor \textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \stackrel{T_a}{\longmapsto}{}^\omega \,.$$

and $T \backslash (\_, \textbf{obj}) = T_a \backslash (\_, \textbf{obj})$. Thus $\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$ and we are done.

B.6 Proofs of Theorem 22

We define the MGC version of deadlock-freedom, and prove it is equivalent to the original version.

Definition 54. $\mathsf{deadlock\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi)$, *iff*

$$\forall n, \sigma_o, T.\ \ T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \wedge \mathsf{objfair}(T) \wedge (\sigma_o \in dom(\varphi)) \\ \implies (\exists i.\ \mathsf{is\_obj\_abt}(T(i))) \vee (\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_ret}(T(j)))\,,$$

where $\mathsf{objfair}(T)$ *says object steps are fairly scheduled:*

$$\begin{aligned}\mathsf{objfair}(T) \stackrel{\text{def}}{=}\ & |T| = \omega \\ & \implies (\forall \mathsf{t} \in [1..\mathsf{tnum}(T)].\ \forall n.\ |(T|_{\mathsf{t}})| = n \\ & \quad \implies \mathsf{is\_ret}((T|_{\mathsf{t}})(n)) \vee \mathsf{is\_clt}((T|_{\mathsf{t}})(n)) \vee (T|_{\mathsf{t}})(n) = (\mathsf{t}, \mathbf{term}))\,.\end{aligned}$$

It's easy to see:

$$\forall T.\ \mathsf{fair}(T) \implies \mathsf{objfair}(T)\,.$$

Lemma 55. *For any* T *and* T_m, *if* $\mathsf{fair}(T)$, $\mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T_m)$, $|T| = \omega$ *and*

$$T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!]\,,$$

then $\mathsf{objfair}(T_m)$.

Proof. Suppose $|(T_m|_{\mathsf{t}})| = n$ and the index of $(T_m|_{\mathsf{t}})(n)$ in T_m is l. If $\mathsf{is\_ret}(T_m(l))$ or $\mathsf{is\_clt}(T_m(l))$ or $T_m(l) = (\mathsf{t}, \mathbf{term})$, we are done. Otherwise, we know

$$\mathsf{is\_inv}(T_m(l)) \text{ or } T_m(l) = (\mathsf{t}, \mathbf{obj})\,.$$

Since $\mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T_m)$, we know there exists i such that

$$T(i) = T_m(l) \quad \text{and} \quad \mathsf{get\_objevt}(T(1..i)) = \mathsf{get\_objevt}(T_m(1..l))\,.$$

Thus $\mathsf{tid}(T(i)) = \mathsf{t}$ and

$$\mathsf{is\_inv}(T(i)) \text{ or } T(i) = (\mathsf{t}, \mathbf{obj})\,.$$

From $\mathsf{fair}(T)$, we know

$$\exists j.\ j > i \wedge \mathsf{tid}(T(j)) = \mathsf{t}\,.$$

By the generation of T and the operational semantics, we know

$$\exists j.\ j > i \wedge \mathsf{tid}(T(j)) = \mathsf{t} \wedge \mathsf{is\_obj}(T(j))\,.$$

Since $\mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T_m)$, we know

$$\exists j.\ j > l \wedge \mathsf{tid}(T_m(j)) = \mathsf{t} \wedge \mathsf{is\_obj}(T_m(j))\,,$$

which contradicts the assumption that $|(T_m|_{\mathsf{t}})| = n$ and the index of $(T_m|_{\mathsf{t}})(n)$ in T_m is l. Thus neither $\mathsf{is\_inv}(T_m(l))$ nor $T_m(l) = (\mathsf{t}, \mathbf{obj})$ holds, and we are done. □

Lemma 56. $\mathsf{deadlock\text{-}free}_{\varphi}(\Pi) \iff \mathsf{deadlock\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi)$.

Proof. 1. $\mathsf{deadlock\text{-}free}_{\varphi}(\Pi) \implies \mathsf{deadlock\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi)$:
As in the proof for Lemma 43, we can prove the following (B.9):

$$\begin{array}{l}\forall n, \sigma_o, T.\ \ T \in \mathcal{T}_\omega[\![(\textbf{let}\ \Pi\ \textbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \wedge (\sigma_o \in dom(\varphi)) \wedge \mathsf{lock\text{-}free}(T) \\ \implies (\exists i.\ \mathsf{is\_obj\_abt}(T(i))) \vee (\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_ret}(T(j)))\end{array}$$

Then we only need to prove the following (B.27):

$$\begin{array}{l}\forall n, \sigma_o, T.\ \ T \in \mathcal{T}_\omega[\![(\textbf{let}\ \Pi\ \textbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \\ \wedge\, \mathsf{objfair}(T) \wedge (\sigma_o \in dom(\varphi)) \wedge \mathsf{deadlock\text{-}free}_{\varphi}(\Pi) \\ \implies \mathsf{lock\text{-}free}(T)\end{array} \tag{B.27}$$

For T such that $T \in \mathcal{T}_\omega[\![(\textbf{let}\ \Pi\ \textbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!]$ and $\mathsf{objfair}(T)$, if $|T| \neq \omega$, then we know $\mathsf{fair}(T)$. By the definition of $\mathsf{deadlock\text{-}free}_{\varphi}(\Pi)$, we know $\mathsf{lock\text{-}free}(T)$. Otherwise, we know $|T| = \omega$, and let

$$\begin{array}{rcl} S & \stackrel{\text{def}}{=} & \{\mathsf{t} \mid \exists n.\ |(T|_{\mathsf{t}})| = n \wedge (T|_{\mathsf{t}})(n) \neq (\mathsf{t}, \textbf{term})\} \\ & = & \{\mathsf{t} \mid |(T|_{\mathsf{t}})| \neq \omega\}\,. \end{array}$$

Then we construct another program $W = \textbf{let}\ \Pi\ \textbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n$ as follows: for any $\mathsf{t} \in [1..n]$,

$$\begin{array}{l} \mathsf{t} \notin S \ \Rightarrow\ C_{\mathsf{t}} = \mathsf{MGT} \\ \mathsf{t} \in S \\ \Rightarrow\ C_{\mathsf{t}} = \ \text{local}\ i_{\mathsf{t}};\ i_{\mathsf{t}} := 0; \\ \qquad\qquad \textbf{while}\ (i_{\mathsf{t}} < n_{\mathsf{t}})\{\ f_{\textbf{rand}(m)}(\textbf{rand}()); i_{\mathsf{t}} := i_{\mathsf{t}} + 1\ \} \\ \qquad\qquad\qquad\qquad \text{where}\ n_{\mathsf{t}} = |\mathsf{get\_hist}(T|_{\mathsf{t}})|/2 \end{array}$$

Let $\sigma_c = \{i_{\mathsf{t}} \rightsquigarrow 0 \mid \mathsf{t} \in S\}$.
We can construct a simulation between $\textbf{let}\ \Pi\ \textbf{in}\ \mathsf{MGC}_n$ and W, and show that there exists T' such that

$$T' \in \mathcal{T}_\omega[\![W, (\sigma_c, \sigma_o, \circledcirc)]\!]\,, \quad \mathsf{fair}(T') \quad \text{and} \quad \mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T')\,.$$

From $\mathsf{deadlock\text{-}free}_{\varphi}(\Pi)$, we know $\mathsf{lock\text{-}free}(T')$. We can prove the following (B.28):

$$\begin{array}{l}\text{If}\ |T| = \omega,\ \mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T')\ \text{and}\ \mathsf{lock\text{-}free}(T')\,, \\ \text{then}\ \mathsf{lock\text{-}free}(T)\,.\end{array} \tag{B.28}$$

Then we know $\mathsf{lock\text{-}free}(T)$ and hence (B.27) holds.
We prove (B.28) as follows. Since $|T| = \omega$, we know one of the following must hold:
(i) there exists i such that $\forall j \geq i.\ \mathsf{is\_clt}(T(j))$;
(ii) $\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_obj}(T(j))$.
For (i), we know $\mathsf{lock\text{-}free}(T)$.
For (ii), since $\mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T')$, we know

$$\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_obj}(T'(j)).$$

Since $\mathsf{lock\text{-}free}(T')$, we know
for any i', if $\mathsf{pend\_inv}(T'(1..i')) \neq \emptyset$, then there exists $j' > i'$ such that $\mathsf{is\_ret}(T'(j'))$.
For T, for any i, we know there exists i' such that

$$\mathsf{get\_objevt}(T(1..i)) = \mathsf{get\_objevt}(T'(1..i'))\,.$$

If $\mathsf{pend\_inv}(T(1..i)) \neq \emptyset$, we know

$$\mathsf{pend\_inv}(T'(1..i')) \neq \emptyset\,.$$

Then we get:

$$\text{there exists } j' > i' \text{ such that } \mathsf{is\_ret}(T'(j')).$$

Thus we know:

$$\text{there exists } j > i \text{ such that } \mathsf{is\_ret}(T(j)).$$

Therefore $\mathsf{lock\text{-}free}(T)$ and we have proved (B.28).

2. $\mathsf{deadlock\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi) \implies \mathsf{deadlock\text{-}free}_{\varphi}(\Pi)$:
For any T such that

$$T \in \mathcal{T}_{\omega}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!],$$

by Lemma 42, we know one of the following holds:
(1) $|T| \neq \omega$; or
(2) there exists i such that $\forall j \geq i.\ \mathsf{is\_clt}(T(j))$; or
(3) there exists T_m such that
$$T_m \in \mathcal{T}_{\omega}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!]\,,$$
and $\mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T_m)$.
For (1), by Lemma 40, we know $\mathsf{lock\text{-}free}(T)$.
For (2), we know $\mathsf{lock\text{-}free}(T)$ holds immediately by definition.
For (3), suppose (1) does not hold. If $\mathsf{fair}(T)$, by Lemma 55, we know $\mathsf{objfair}(T_m)$ holds. Then from $\mathsf{deadlock\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi)$, we know

$$(\exists i.\ \mathsf{is\_obj\_abt}(T_m(i))) \vee (\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_ret}(T_m(j))).$$

Thus we have:

$$(\exists i.\ \mathsf{is\_obj\_abt}(T(i))) \vee (\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_ret}(T(j))).$$

If $\exists i.\ \mathsf{is\_obj\_abt}(T(i))$, we know $\mathsf{lock\text{-}free}(T)$. Otherwise, we know

$$\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_ret}(T(j)).$$

Thus, for any i, if $\mathsf{pend\_inv}(T(1..i)) \neq \emptyset$, then there exists $j > i$ such that $\mathsf{is\_ret}(T(j))$. Therefore $\mathsf{lock\text{-}free}(T)$ and we are done.

□

Then, we only need to prove the following (B.29), (B.30) and (B.31):

$$\Pi \sqsubseteq^{f\omega}_{\varphi} \Pi_A \implies \Pi \sqsubseteq_{\varphi} \Pi_A \tag{B.29}$$

$$\Pi \sqsubseteq^{f\omega}_{\varphi} \Pi_A \implies \mathsf{deadlock\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi) \tag{B.30}$$

$$\Pi \sqsubseteq_{\varphi} \Pi_A \wedge \mathsf{deadlock\text{-}free}_{\varphi}(\Pi) \implies \Pi \sqsubseteq^{f\omega}_{\varphi} \Pi_A \tag{B.31}$$

Proofs of (B.29) For any n, C_1, ..., C_n, σ_c, σ_o and σ_a such that $\varphi(\sigma_o) = \sigma_a$, for any T if

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

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

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

Thus there exists T_1' and T_1'' such that $T_1'' = T_1 :: T_1'$, where $\mathsf{fair}(T_1')$ holds, and one of the following holds:

(i) $(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \xmapsto{T_1''}{}^{\omega} \cdot$; or
(ii) $(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \xmapsto{T_1''}{}^{*} (\textbf{skip}, \_)$; or
(iii) $(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \xmapsto{T_1''}{}^{*} \textbf{abort}$.

Thus,

$$T_1'' \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!] \quad \text{and} \quad \mathsf{fair}(T_1'')\,.$$

Since $\Pi \sqsubseteq_{=\varphi}^{f\omega} \Pi_A$, we know there exists T_2'' such that

$$T_2'' \in \mathcal{T}_\omega[\![(\textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a, \circledcirc)]\!]\,,$$

and

$$\mathsf{get\_obsv}(T_2'') = \mathsf{get\_obsv}(T_1'') = T :: \mathsf{get\_obsv}(T_1')\,.$$

Thus there exists T_2 such that

$$T_2 \in \mathcal{T}[\![(\textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a, \circledcirc)]\!]\,,$$

and $\mathsf{get\_obsv}(T_2) = T$. Thus

$$T \in \mathcal{O}[\![(\textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a, \circledcirc)]\!]\,,$$

and we are done.

Proofs of (B.30) The proof is similar to the proof of (B.12), except that we need to first prove the following lemma:

Lemma 57. *Suppose Π_A is total. If $\Pi \sqsubseteq_{=\varphi}^{f\omega} \Pi_A$, then*

$$\mathcal{O}_{of\omega}[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGCp1}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{O}_\omega[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGCp1}_n), (\emptyset, \sigma_a, \circledcirc)]\!]\,,$$

where

$$\mathcal{O}_{of\omega}[\![W, \mathcal{S}]\!] \stackrel{\text{def}}{=} \{\mathsf{get\_obsv}(T) \mid T \in \mathcal{T}_\omega[\![W, \mathcal{S}]\!] \wedge \mathsf{objfair}(T) \wedge \forall i, \mathsf{t}.\ T(i) = (\mathsf{t}, \textbf{ret}, \_) \Leftrightarrow T(i+1) = (\mathsf{t}, \textbf{out}, 1)\}\,.$$

Proof. For any T and T_o such that

$$T \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGCp1}_n), (\emptyset, \sigma_o, \circledcirc)]\!], \quad \mathsf{objfair}(T),$$
$$\forall i, \mathsf{t}.\ T(i) = (\mathsf{t}, \textbf{ret}, \_) \ \Leftrightarrow\ T(i+1) = (\mathsf{t}, \textbf{out}, 1),$$

and $T_o = \mathsf{get\_obsv}(T)$, if $|T| \neq \omega$, we know $\mathsf{fair}(T)$ holds, thus

$$T_o \in \mathcal{O}_{f\omega}[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGCp1}_n), (\emptyset, \sigma_o, \circledcirc)]\!].$$

From $\Pi \sqsubseteq^{f\omega}_{\varphi} \Pi_A$, we know

$$T_o \in \mathcal{O}_{\omega}[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGCp1}_n), (\emptyset, \sigma_a, \circledcirc)]\!].$$

Otherwise, we know $|T| = \omega$, and let

$$\begin{aligned} S &\stackrel{\text{def}}{=} \{\mathsf{t} \mid \exists n.\ |(T|_{\mathsf{t}})| = n \wedge (T|_{\mathsf{t}})(n) \neq (\mathsf{t}, \textbf{term})\} \\ &= \{\mathsf{t} \mid |(T|_{\mathsf{t}})| \neq \omega\}. \end{aligned}$$

Since $|T| = \omega$, we know there exists t such that $|(T|_{\mathsf{t}})| = \omega$ and hence $\mathsf{t} \notin S$. Then we construct another program $W = \textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n$ as follows: for any $\mathsf{t} \in [1..n]$,

$$\begin{array}{l} \mathsf{t} \notin S \ \Rightarrow\ C_{\mathsf{t}} = \mathsf{MGTp1} \\ \mathsf{t} \in S \\ \Rightarrow\ C_{\mathsf{t}} = \ \text{local } i_{\mathsf{t}};\ i_{\mathsf{t}} := 0; \\ \qquad\quad \textbf{while } (i_{\mathsf{t}} < n_{\mathsf{t}})\{ \\ \qquad\quad\ f_{\textbf{rand}(m)}(\textbf{rand}()); \textbf{print}(1); i_{\mathsf{t}} := i_{\mathsf{t}} + 1; \\ \qquad\quad \} \\ \qquad\qquad\qquad\qquad \text{where } n_{\mathsf{t}} = |\mathsf{get\_hist}(T|_{\mathsf{t}})|/2 \end{array}$$

Let $\sigma_c = \{i_{\mathsf{t}} \rightsquigarrow 0 \mid \mathsf{t} \in S\}$.
We can construct a simulation between $\textbf{let } \Pi \textbf{ in } \mathsf{MGCp1}_n$ and W, and show that there exists T' such that

$$T' \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!],$$
$$\mathsf{fair}(T') \quad \text{and} \quad \mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T') = T_o.$$

Since $\Pi \sqsubseteq^{f\omega}_{\varphi} \Pi_A$, we know

$$T_o \in \mathcal{O}_{\omega}[\![(\textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a, \circledcirc)]\!].$$

Thus there exists T'' such that

$$T'' \in \mathcal{T}_\omega[\![(\textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a, \circledcirc)]\!], \quad \text{and} \quad \mathsf{get\_obsv}(T'') = T_o.$$

Since there exists t such that $C_{\mathsf{t}} = \mathsf{MGCp1}$, we can construct a simulation and show that there exists T''' such that

$$T''' \in \mathcal{T}_\omega[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGCp1}_n), (\emptyset, \sigma_a, \circledcirc)]\!],$$
$$\text{and} \quad \mathsf{get\_obsv}(T'') = \mathsf{get\_obsv}(T''') = T_o.$$

Thus we are done. □

To prove $\mathsf{deadlock\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi)$, we want to show: for any n, σ_o, σ_a and T, if $T \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!]$, $\mathsf{objfair}(T)$ and $\varphi(\sigma_o) = \sigma_a$, then the following (B.14) holds:

$$(\exists i.\ \mathsf{is\_obj\_abt}(T(i))) \vee (\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_ret}(T(j)))\,.$$

First, if $T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!]$ and $\mathsf{objfair}(T)$, by Lemma 44(1), there exists T_p such that

$$T_p \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGCp1}_n), (\emptyset, \sigma_o, \circledcirc)]\!], \quad T_p \backslash (\_, \mathbf{out}, 1) = T$$
$$\text{and} \quad \forall i, \mathsf{t}.\ T_p(i) = (\mathsf{t}, \mathbf{ret}, \_) \ \Leftrightarrow\ T_p(i+1) = (\mathsf{t}, \mathbf{out}, 1).$$

Since $\mathsf{objfair}(T)$, we know $\mathsf{objfair}(T_p)$ also holds.

Since $\Pi \sqsubseteq_\varphi^{f\omega} \Pi_A$, by Lemma 57, we know

$$\mathcal{O}_{of\omega}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGCp1}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{O}_\omega[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ \mathsf{MGCp1}_n), (\emptyset, \sigma_a, \circledcirc)]\!]\,.$$

From Lemma 45, we know for any T, if $T \in \mathcal{O}_{of\omega}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGCp1}_n), (\emptyset, \sigma_o, \circledcirc)]\!]$, then T is an infinite trace of $(\_, \mathbf{out}, 1)$.

Then we know: $\mathsf{get\_obsv}(T_p)$ is an infinite trace of $(\_, \mathbf{out}, 1)$. Thus $|T_p| = \omega$ and the following (B.15) holds:

$$\forall i.\ \exists j.\ j \geq i \wedge T_p(j) = (\_, \mathbf{out}, 1)\,.$$

As in the proof of (B.12), we prove the following (B.16) from (B.15):

$$\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_ret}(T_p(j))\,.$$

Since $T_p \backslash (\_, \mathbf{out}, 1) = T$, from (B.16), we get (B.14) and thus we are done.

Proofs of (B.31) We need to prove that if $\Pi \sqsubseteq_\varphi \Pi_A$ and $\mathsf{deadlock\text{-}free}_\varphi(\Pi)$, then for any n, C_1, ..., C_n, σ_c, σ_o and σ_a such that $\varphi(\sigma_o) = \sigma_a$, we have

$$\mathcal{O}_{f\omega}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!]$$
$$\subseteq \mathcal{O}_\omega[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a, \circledcirc)]\!]\,.$$

Thus we only need to prove: for any T,

(1) If $(\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \xmapsto{T}{}^* \mathbf{abort}$,
then there exists T_a such that
$(\lfloor \mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \xmapsto{T_a}{}^* \mathbf{abort}$ and
$\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$.

(2) If $(\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \xmapsto{T}{}^* (\mathbf{skip}, \_)$,
then there exists T_a such that
$(\lfloor \mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \xmapsto{T_a}{}^* (\mathbf{skip}, \_)$ and
$\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$.

(3) If $(\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \xmapsto{T}{}^\omega \cdot$ and $\mathsf{fair}(T)$,
then there exists T_a such that
$(\lfloor \mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \xmapsto{T_a}{}^\omega \cdot$ and
$\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$.

(1) and (2) are proved in Lemma 46.

For (3), as in the proofs for (B.13), we define the simulation relation $\precsim$ in Figure 9(d), and prove the following (B.19):

For any W, $\mathcal{S}$, W_1, $\mathcal{S}_1$, W_2, $\mathcal{S}_2$, W_3, $\mathcal{S}_3$, T_0 and T_1,
if $(W, \mathcal{S})$ is well-formed and out of method calls, $(W, \mathcal{S}) \overset{T_0}{\longmapsto}{}^* (W_1, \mathcal{S}_1)$,
$(W_1, \mathcal{S}_1) \precsim (W_2, \mathcal{S}_2; W_3, \mathcal{S}_3)$, $(W_1, \mathcal{S}_1) \overset{T_1}{\longmapsto}{}^\omega \cdot$ and $\mathsf{lock\text{-}free}(T_0 :: T_1)$,
then there exists T_3 such that $(W_3, \mathcal{S}_3) \overset{T_3}{\longmapsto}{}^\omega \cdot$ and
$T_1 \backslash (\_, \mathbf{obj}) = T_3 \backslash (\_, \mathbf{obj})$.

On the other hand, for any n, C_1, ..., C_n, σ_c, σ_o and σ_a, by Lemma 32, we know

$$\mathcal{H}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{H}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] .$$

From $\Pi \sqsubseteq_\varphi \Pi_A$, by Lemma 34, we know $\Pi \subseteq_\varphi \Pi_A$. Thus, if $\varphi(\sigma_o) = \sigma_a$, then

$$\mathcal{H}[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{H}[\![(\mathbf{let}\ \Pi_A\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!] .$$

Then we know

$$\begin{array}{l}(\mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \sigma_o, \circledcirc)) \\ \precsim (\mathbf{let}\ \Pi_A\ \mathbf{in}\ \mathsf{MGC}_n, (\emptyset, \sigma_a, \circledcirc); \\ \quad \mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \sigma_a, \circledcirc)),\end{array}$$

Thus, if $(\lfloor \mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^\omega \cdot$ and $\mathsf{fair}(T)$,
by $\mathsf{deadlock\text{-}free}_\varphi(\Pi)$, we know $\mathsf{lock\text{-}free}(T)$. Then from (B.19) we get: there exists T_a such that

$$(\lfloor \mathbf{let}\ \Pi_A\ \mathbf{in}\ C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \overset{T_a}{\longmapsto}{}^\omega .$$

and $T \backslash (\_, \mathbf{obj}) = T_a \backslash (\_, \mathbf{obj})$. Thus $\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$ and we are done.

B.7 Proofs of Theorem 26

We define the MGC version of starvation-freedom, and prove it is equivalent to the original version.

Definition 58. $\mathsf{starvation\text{-}free}_\varphi^{\mathsf{MGC}}(\Pi)$, *iff*

$$\forall n, \sigma_o, T.\ \ T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \wedge \mathsf{objfair}(T) \wedge (\sigma_o \in dom(\varphi)) \implies \mathsf{wait\text{-}free}(T)$$

Lemma 59. $\mathsf{starvation\text{-}free}_\varphi(\Pi) \iff \mathsf{starvation\text{-}free}_\varphi^{\mathsf{MGC}}(\Pi)$.

Proof. 1. $\mathsf{starvation\text{-}free}_\varphi(\Pi) \implies \mathsf{starvation\text{-}free}_\varphi^{\mathsf{MGC}}(\Pi)$:
We only need to prove the following (B.32):

$$\begin{array}{l}\forall n, \sigma_o, T.\ \ T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \\ \wedge\, \mathsf{objfair}(T) \wedge (\sigma_o \in dom(\varphi)) \wedge \mathsf{starvation\text{-}free}_\varphi(\Pi) \\ \implies \mathsf{wait\text{-}free}(T)\end{array} \tag{B.32}$$

For T such that $T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!]$ and $\mathsf{objfair}(T)$, if $|T| \neq \omega$, then we know $\mathsf{fair}(T)$. By the definition of $\mathsf{starvation\text{-}free}_\varphi(\Pi)$, we know $\mathsf{wait\text{-}free}(T)$. Otherwise, we know $|T| = \omega$, and let

$$\begin{aligned} S &\stackrel{\text{def}}{=} \{\mathsf{t} \mid \exists n.\ |(T|_{\mathsf{t}})| = n \wedge (T|_{\mathsf{t}})(n) \neq (\mathsf{t}, \mathbf{term})\} \\ &= \{\mathsf{t} \mid |(T|_{\mathsf{t}})| \neq \omega\}\,. \end{aligned}$$

Then we construct another program $W = \mathbf{let}\ \Pi\ \mathbf{in}\ C_1 \parallel \ldots \parallel C_n$ as follows: for any $\mathsf{t} \in [1..n]$,

$$\begin{array}{l} \mathsf{t} \notin S \ \Rightarrow\ C_{\mathsf{t}} = \mathsf{MGT} \\ \mathsf{t} \in S \\ \Rightarrow\ C_{\mathsf{t}} = \ \text{local } i_{\mathsf{t}};\ i_{\mathsf{t}} := 0; \\ \qquad\qquad \mathbf{while}\ (i_{\mathsf{t}} < n_{\mathsf{t}})\{\ f_{\mathbf{rand}(m)}(\mathbf{rand}()); i_{\mathsf{t}} := i_{\mathsf{t}} + 1\ \} \\ \qquad\qquad\qquad\qquad \text{where } n_{\mathsf{t}} = |\mathsf{get\_hist}(T|_{\mathsf{t}})|/2 \end{array}$$

Let $\sigma_c = \{i_{\mathsf{t}} \rightsquigarrow 0 \mid \mathsf{t} \in S\}$.
We can construct a simulation between $\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n$ and W, and show that there exists T' such that

$$T' \in \mathcal{T}_\omega[\![W, (\sigma_c, \sigma_o, \circledcirc)]\!]\,, \quad \mathsf{fair}(T') \quad \text{and} \quad \mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T')\,.$$

From $\mathsf{starvation\text{-}free}_\varphi(\Pi)$, we know $\mathsf{wait\text{-}free}(T')$. We can prove the following (B.33):

$$\begin{array}{l} \text{If } |T| = \omega,\ \mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T') \text{ and } \mathsf{wait\text{-}free}(T')\,, \\ \text{then } \mathsf{wait\text{-}free}(T)\,. \end{array} \tag{B.33}$$

Then we know $\mathsf{wait\text{-}free}(T)$ and hence (B.32) holds.
We prove (B.33) as follows. Since $\mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T')$, for any i, we know there exists i' such that

$$\mathsf{get\_objevt}(T(1..i)) = \mathsf{get\_objevt}(T'(1..i'))\,.$$

For any e, if $e \in \mathsf{pend\_inv}(T(1..i))$, we know

$$e \in \mathsf{pend\_inv}(T'(1..i'))\,.$$

From $\mathsf{wait\text{-}free}(T')$, we know one of the following holds:
(i) there exists $j' > i'$ such that $\mathsf{match}(e, T'(j'))$.
(ii) there exists $j' > i'$ such that $\forall k' \geq j'.\ \mathsf{tid}(T'(k')) \neq \mathsf{tid}(e)$.
For (i), since $\mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T')$, we know

$$\text{there exists } j > i \text{ such that } \mathsf{match}(e, T(j)).$$

For (ii), assume (i) does not hold. Then we know $e \in \mathsf{pend\_inv}(T')$. Since $\mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T')$, we can prove

$$e \in \mathsf{pend\_inv}(T)\,.$$

Let $\mathsf{t} = \mathsf{tid}(e)$. Suppose

$$\forall j > i.\ \exists k \geq j.\ \mathsf{tid}(T(k)) = \mathsf{t}\,.$$

Then, by the operational semantics and the generation of T, we know

$$\forall j > i.\ \exists k \geq j.\ T(k) = (\mathsf{t}, \mathbf{obj})\,.$$

Since $\mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T')$, we know

$$\forall j' > i'.\ \exists k' \geq j'.\ T'(k') = (\mathsf{t}, \mathbf{obj})\,,$$

which contradicts (ii). Thus we know

$$\exists j > i.\ \forall k \geq j.\ \mathsf{tid}(T(k)) \neq \mathsf{t}\,.$$

Therefore $\mathsf{wait\text{-}free}(T)$ and we have proved (B.33).

2. $\mathsf{starvation\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi) \implies \mathsf{starvation\text{-}free}_{\varphi}(\Pi)$:
Almost the same as the proof for Lemma 49, except that we need to apply Lemma 55.

□

Then, we only need to prove the following (B.34), (B.35) and (B.36), where (B.34) is trivial from definitions:

$$\Pi \sqsubseteq^{\mathit{ff}\omega}_{\varphi} \Pi_A \implies \Pi \sqsubseteq^{\mathit{f}\omega}_{\varphi} \Pi_A \tag{B.34}$$

$$\Pi \sqsubseteq^{\mathit{ff}\omega}_{\varphi} \Pi_A \implies \mathsf{starvation\text{-}free}^{\mathsf{MGC}}_{\varphi}(\Pi) \tag{B.35}$$

$$\Pi \sqsubseteq_{\varphi} \Pi_A \wedge \mathsf{starvation\text{-}free}_{\varphi}(\Pi) \implies \Pi \sqsubseteq^{\mathit{ff}\omega}_{\varphi} \Pi_A \tag{B.36}$$

Proofs of (B.35) For any n, σ_o, σ_a and T such that $\varphi(\sigma_o) = \sigma_a$, if $T \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!]$ and $\mathsf{objfair}(T)$, suppose

$$\neg\exists i.\ \mathsf{is\_abt}(T(i)),$$

then by the operational semantics, we only need to prove:

> for any i and e, if $e \in \mathsf{pend\_inv}(T(1..i))$, then there exists $j > i$ such that $\mathsf{match}(e, T(j))$.

Suppose it does not hold. Then we know there exists t_0 such that

$$\exists i.\ \forall j.\ j \geq i \Rightarrow (T|_{\mathsf{t}_0})(j) = (\mathsf{t}_0, \mathbf{obj})\,.$$

By Lemma 44(1), there exists T_p such that

$$T_p \in \mathcal{T}_\omega[\![(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{MGCp1}_n), (\emptyset, \sigma_o, \circledcirc)]\!]\,, \quad T_p \backslash (\_, \mathbf{out}, 1) = T$$
$$\text{and} \quad \forall i, \mathsf{t}.\ T_p(i) = (\mathsf{t}, \mathbf{ret}, \_) \ \Leftrightarrow\ T_p(i+1) = (\mathsf{t}, \mathbf{out}, 1).$$

By the operational semantics, we know

$$\exists i.\ \forall j.\ j \geq i \Rightarrow (T_p|_{\mathsf{t}_0})(j) = (\mathsf{t}_0, \mathbf{obj})\,.$$

Let

$$\begin{aligned} S &\stackrel{\text{def}}{=} \{\mathsf{t} \mid \exists n.\ |(T_p|_{\mathsf{t}})| = n \wedge (T_p|_{\mathsf{t}})(n) \neq (\mathsf{t}, \mathbf{term})\} \\ &= \{\mathsf{t} \mid |(T_p|_{\mathsf{t}})| \neq \omega\}\,. \end{aligned}$$

Thus we know

$$\mathsf{t}_0 \notin S.$$

We construct another program $W = \textbf{let } \Pi \textbf{ in } C_1 \parallel \ldots \parallel C_n$ as follows: for any $\mathsf{t} \in [1..n]$,

$$\begin{array}{l}
\mathsf{t} \notin S \;\Rightarrow\; C_\mathsf{t} = \mathsf{MGTp1} \\
\mathsf{t} \in S \\
\Rightarrow\; C_\mathsf{t} = \;\; \text{local } i_\mathsf{t};\; i_\mathsf{t} := 0; \\
\qquad\qquad \textbf{while } (i_\mathsf{t} < n_\mathsf{t})\{ \\
\qquad\qquad\quad f_{\textbf{rand}(m)}(\textbf{rand}()); \textbf{print}(1); i_\mathsf{t} := i_\mathsf{t} + 1; \\
\qquad\qquad \} \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{where } n_\mathsf{t} = |\mathsf{get_hist}(T|_\mathsf{t})|/2
\end{array}$$

Let $\sigma_c = \{i_\mathsf{t} \rightsquigarrow 0 \mid \mathsf{t} \in S\}$. We can construct a simulation between $\textbf{let } \Pi \textbf{ in } \mathsf{MGCp1}_n$ and W, and show that there exists T'_p such that

$$\begin{array}{c}
T'_p \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } C_1 \parallel \ldots \parallel C_n), (\sigma_c, \sigma_o, \circledcirc)]\!]\,, \quad \mathsf{fair}(T'_p)\,, \\
\mathsf{get_objevt}(T_p) = \mathsf{get_objevt}(T'_p) \quad \text{and} \quad \mathsf{get_obsv}(T_p) = \mathsf{get_obsv}(T'_p)\,.
\end{array}$$

Thus we know there exists i such that

$$\forall j.\; j \geq i \Rightarrow (T'_p|_{\mathsf{t}_0})(j) = (\mathsf{t}_0, \textbf{obj})\,.$$

Thus we have

$$|(\mathsf{get\_obsv}(T'_p)|_{\mathsf{t}_0})| < i\,.$$

On the other hand, since $\Pi \sqsubseteq^{ff\omega}_{\varphi} \Pi_A$, we know:

$$\begin{array}{l}
\mathcal{O}_{f\omega}[\![(\textbf{let } \Pi \textbf{ in } C_1 \parallel \ldots \parallel C_n), (\sigma_c, \sigma_o, \circledcirc)]\!] \\
\subseteq \mathcal{O}_{f\omega}[\![(\textbf{let } \Pi_A \textbf{ in } C_1 \parallel \ldots \parallel C_n), (\sigma_c, \sigma_a, \circledcirc)]\!]\,.
\end{array}$$

Thus there exists T''_p such that

$$\begin{array}{c}
T''_p \in \mathcal{T}_\omega[\![(\textbf{let } \Pi_A \textbf{ in } C_1 \parallel \ldots \parallel C_n), (\sigma_c, \sigma_a, \circledcirc)]\!]\,, \\
\mathsf{fair}(T''_p) \quad \text{and} \quad \mathsf{get_obsv}(T''_p) = \mathsf{get_obsv}(T'_p)\,.
\end{array}$$

Since $C_{\mathsf{t}_0} = \mathsf{MGTp1}$ and $\mathsf{fair}(T''_p)$, we know

$$|(T''_p|_{\mathsf{t}_0})| = \omega\,,$$

and also

$$|(\mathsf{get\_obsv}(T'_p)|_{\mathsf{t}_0})| = |(\mathsf{get\_obsv}(T''_p)|_{\mathsf{t}_0})| = \omega\,,$$

which contradicts the fact that $|(\mathsf{get\_obsv}(T'_p)|_{\mathsf{t}_0})| < i$. Thus we know $\mathsf{wait\text{-}free}(T)$ and we are done.

Proofs of (B.36) We need to prove that if $\Pi \sqsubseteq_\varphi \Pi_A$ and $\mathsf{starvation\text{-}free}_\varphi(\Pi)$, then for any n, C_1, …, C_n, σ_c, σ_o and σ_a such that $\varphi(\sigma_o) = \sigma_a$, we have

$$\begin{array}{l}\mathcal{O}_{f\omega}[\![(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!] \\ \subseteq \mathcal{O}_{f\omega}[\![(\textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_a, \circledcirc)]\!]\,.\end{array}$$

Thus we only need to prove: for any T,

(1) If $(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^* \textbf{abort}$, then there exists T_a such that $(\lfloor \textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \overset{T_a}{\longmapsto}{}^* \textbf{abort}$ and $\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$.

(2) If $(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^* (\textbf{skip}, \_)$, then there exists T_a such that $(\lfloor \textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \overset{T_a}{\longmapsto}{}^* (\textbf{skip}, \_)$ and $\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$.

(3) If $(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^\omega \cdot$ and $\mathsf{fair}(T)$, then there exists T_a such that $(\lfloor \textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \overset{T_a}{\longmapsto}{}^\omega \cdot$, $\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$ and $\mathsf{fair}(T_a)$.

(1) and (2) are proved in Lemma 46.

For (3), as in the proofs for (B.22), we define the simulation relation $\precsim$ in Figure 9(d), and prove the following (B.23):

> For any W, $\mathcal{S}$, W_1, $\mathcal{S}_1$, W_2, $\mathcal{S}_2$, W_3, $\mathcal{S}_3$, T_0 and T_1,
> if $(W, \mathcal{S})$ is well-formed and out of method calls, $(W, \mathcal{S}) \overset{T_0}{\longmapsto}{}^* (W_1, \mathcal{S}_1)$,
> $(W_1, \mathcal{S}_1) \precsim (W_2, \mathcal{S}_2; W_3, \mathcal{S}_3)$, $(W_1, \mathcal{S}_1) \overset{T_1}{\longmapsto}{}^\omega \cdot$ and $\mathsf{wait\text{-}free}(T_0 :: T_1)$,
> then there exists T_3 such that $(W_3, \mathcal{S}_3) \overset{T_3}{\longmapsto}{}^\omega \cdot$ and
> $T_1 \backslash (\_, \textbf{obj}) = T_3 \backslash (\_, \textbf{obj})$.

On the other hand, for any n, C_1, …, C_n, σ_c, σ_o and σ_a, by Lemma 32, we know

$$\mathcal{H}[\![(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n), (\sigma_c, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{H}[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!]\,.$$

From $\Pi \sqsubseteq_\varphi \Pi_A$, by Lemma 34, we know $\Pi \subseteq_\varphi \Pi_A$. Thus, if $\varphi(\sigma_o) = \sigma_a$, then

$$\mathcal{H}[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{H}[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n), (\emptyset, \sigma_a, \circledcirc)]\!]\,.$$

Then we know

$$\begin{array}{l}(\textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \sigma_o, \circledcirc)) \\ \precsim (\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_n, (\emptyset, \sigma_a, \circledcirc); \\ \quad \textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n, (\sigma_c, \sigma_a, \circledcirc)),\end{array}$$

Thus, if $(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^\omega \cdot$ and $\mathsf{fair}(T)$, by $\mathsf{starvation\text{-}free}_\varphi(\Pi)$, we know $\mathsf{wait\text{-}free}(T)$. Then from (B.23) we get: there exists T_a such that

$$(\lfloor \textbf{let } \Pi_A \textbf{ in } C_1 \,\|\, \ldots \,\|\, C_n \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \xmapsto{T_a}{}^{\omega} \cdot \text{, and } T\backslash(\_, \textbf{obj}) = T_a\backslash(\_, \textbf{obj}).$$

Thus we know $\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$.

Below we prove: $\mathsf{fair}(T_a)$. Since $\mathsf{fair}(T)$ and $|T| = \omega$, we know for any t,

$$\text{either } |(T|_{\mathsf{t}})| = \omega, \quad \text{or } \mathsf{last}(T|_{\mathsf{t}}) = (\mathsf{t}, \textbf{term})\,.$$

(a) $\mathsf{last}(T|_{\mathsf{t}}) = (\mathsf{t}, \textbf{term})$:
Since $T\backslash(\_, \textbf{obj}) = T_a\backslash(\_, \textbf{obj})$ and by the operational semantics, we know $\mathsf{last}(T_a|_{\mathsf{t}}) = (\mathsf{t}, \textbf{term})$.

(b) $|(T|_{\mathsf{t}})| = \omega$:
Since $T\backslash(\_, \textbf{obj}) = T_a\backslash(\_, \textbf{obj})$, we know

$$(T|_{\mathsf{t}})\backslash(\mathsf{t}, \textbf{obj}) = (T_a|_{\mathsf{t}})\backslash(\mathsf{t}, \textbf{obj})\,.$$

Suppose $|(T_a|_{\mathsf{t}})| \neq \omega$. Then we know $|(T_a|_{\mathsf{t}})\backslash(\mathsf{t}, \textbf{obj})| \neq \omega$. Thus

$$\exists i.\ \forall j.\ j \geq i \ \Rightarrow\ (T|_{\mathsf{t}})(j) = (\mathsf{t}, \textbf{obj})\,.$$

By the operational semantics, we know there exists i such that

$$\mathsf{tid}(T(i)) = \mathsf{t}\,, \quad \mathsf{is\_inv}(T(i))\,, \quad \text{and} \quad \forall j.\ j \geq i \ \Rightarrow\ \neg\mathsf{match}(T(i), T(j))\,.$$

By $\mathsf{wait\text{-}free}(T)$, we know

$$\exists j.\ \forall k \geq j.\ \mathsf{tid}(T(k)) \neq \mathsf{t}\,,$$

which contradicts the assumption that $|(T|_{\mathsf{t}})| = \omega$.
Thus we know $|(T_a|_{\mathsf{t}})| = \omega$.

Thus $\mathsf{fair}(T_a)$ holds and we are done.

B.8 Proofs of Theorem 29

Proofs of Theorem 29(1) For any σ_o, σ_a and T such that $\varphi(\sigma_o) = \sigma_a$, if

$$T \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } C_1), (\sigma_c, \sigma_o, \circledcirc)]\!],$$

by Lemma 42, we know one of the following holds:

(1) $|T| \neq \omega$; or
(2) there exists i such that $\forall j \geq i.\ \mathsf{is\_clt}(T(j))$; or
(3) there exists T_m such that

$$T_m \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGT}), (\emptyset, \sigma_o, \circledcirc)]\!]\,,$$

and $\mathsf{get\_objevt}(T) = \mathsf{get\_objevt}(T_m)$.

For (1), by the operational semantics, we can prove $\mathsf{prog\text{-}t}(T)$ or $\mathsf{abt}(T)$ holds.
For (2), for any k and e, if $e \in \mathsf{pend\_inv}(T(1..k))$, since there exists $i > k$ such that $\mathsf{is\_clt}(T(i))$, by the operational semantics we know there exists j such that $k < j < i$ and $\mathsf{match}(e, T(j))$. Thus $\mathsf{prog\text{-}t}(T)$ holds.
For (3), by Lemma 44(1), there exists T_p such that

$$T_p \in \mathcal{T}_\omega[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGTp1}), (\emptyset, \sigma_o, \circledcirc)]\!] \text{ and } T_p \backslash (\_, \textbf{out}, 1) = T.$$

Since $\Pi \sqsubseteq_\varphi^{1\omega} \Pi_A$, we know

$$\mathcal{O}_\omega[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGTp1}), (\emptyset, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{O}_\omega[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGTp1}), (\emptyset, \sigma_a, \circledcirc)]\!] .$$

From Lemma 45, we know $\mathsf{get\_obsv}(T_p)$ is an infinite trace of $(\_, \textbf{out}, 1)$. Thus $|T_p| = \omega$ and the following (B.15) holds:

$$\forall i.\ \exists j.\ j \geq i \wedge T_p(j) = (\_, \textbf{out}, 1) .$$

As in the proof of (B.12), we prove the following (B.16) from (B.15):

$$\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_ret}(T_p(j)) .$$

Since $T_p \backslash (\_, \textbf{out}, 1) = T$, we know

$$\forall i.\ \exists j.\ j \geq i \wedge \mathsf{is\_ret}(T(j)) .$$

Thus for any i and e, if $e \in \mathsf{pend\_inv}(T(1..i))$, then there exists $j > i$ such that $\mathsf{is\_ret}(T(j))$ holds. By the operational semantics and the generation of T, we know $\mathsf{match}(e, T(j))$ holds. Thus $\mathsf{prog\text{-}t}(T)$ holds. Then we are done.

Proofs of Theorem 29(2) We need to prove that if $\Pi \sqsubseteq_\varphi \Pi_A$ and $\mathsf{seq\text{-}term}_\varphi(\Pi)$, then for any C_1, σ_c, σ_o and σ_a such that $\varphi(\sigma_o) = \sigma_a$, we have

$$\mathcal{O}_\omega[\![(\textbf{let } \Pi \textbf{ in } C_1), (\sigma_c, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{O}_\omega[\![(\textbf{let } \Pi_A \textbf{ in } C_1), (\sigma_c, \sigma_a, \circledcirc)]\!] .$$

Thus we only need to prove: for any T,

(1) If $(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^* \textbf{abort}$,
then there exists T_a such that
$(\lfloor \textbf{let } \Pi_A \textbf{ in } C_1 \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \overset{T_a}{\longmapsto}{}^* \textbf{abort}$ and $\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$.

(2) If $(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^* (\textbf{skip}, \_)$,
then there exists T_a such that
$(\lfloor \textbf{let } \Pi_A \textbf{ in } C_1 \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \overset{T_a}{\longmapsto}{}^* (\textbf{skip}, \_)$ and $\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$.

(3) If $(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^\omega \cdot$,
then there exists T_a such that
$(\lfloor \textbf{let } \Pi_A \textbf{ in } C_1 \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \overset{T_a}{\longmapsto}{}^\omega \cdot$ and $\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$.

(1) and (2) are proved in Lemma 46.

For (3), as in the proofs for (B.13), we define the simulation relation $\precsim$ in Figure 9(d), and prove the following (B.19):

> For any W, $\mathcal{S}$, W_1, $\mathcal{S}_1$, W_2, $\mathcal{S}_2$, W_3, $\mathcal{S}_3$, T_0 and T_1,
> if $(W, \mathcal{S})$ is well-formed and out of method calls, $(W, \mathcal{S}) \overset{T_0}{\longmapsto}{}^* (W_1, \mathcal{S}_1)$,
> $(W_1, \mathcal{S}_1) \precsim (W_2, \mathcal{S}_2; W_3, \mathcal{S}_3)$, $(W_1, \mathcal{S}_1) \overset{T_1}{\longmapsto}{}^\omega \cdot$ and $\mathsf{lock\text{-}free}(T_0 :: T_1)$,
> then there exists T_3 such that $(W_3, \mathcal{S}_3) \overset{T_3}{\longmapsto}{}^\omega \cdot$ and
> $T_1 \backslash (\_, \textbf{obj}) = T_3 \backslash (\_, \textbf{obj})$.

On the other hand, for any n, C_1, $\ldots$, C_n, σ_c, σ_o and σ_a, by Lemma 32, we know

$$\mathcal{H}[\![(\textbf{let } \Pi \textbf{ in } C_1), (\sigma_c, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{H}[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGC}_1), (\emptyset, \sigma_o, \circledcirc)]\!] .$$

From $\Pi \sqsubseteq_\varphi \Pi_A$, by Lemma 34, we know $\Pi \subseteqq_\varphi \Pi_A$. Thus, if $\varphi(\sigma_o) = \sigma_a$, then

$$\mathcal{H}[\![(\textbf{let } \Pi \textbf{ in } \mathsf{MGC}_1), (\emptyset, \sigma_o, \circledcirc)]\!] \subseteq \mathcal{H}[\![(\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_1), (\emptyset, \sigma_a, \circledcirc)]\!] .$$

Then we know

$$\begin{array}{l}(\textbf{let } \Pi \textbf{ in } C_1, (\sigma_c, \sigma_o, \circledcirc)) \\ \precsim (\textbf{let } \Pi_A \textbf{ in } \mathsf{MGC}_1, (\emptyset, \sigma_a, \circledcirc); \\ \quad \textbf{let } \Pi_A \textbf{ in } C_1, (\sigma_c, \sigma_a, \circledcirc)),\end{array}$$

Thus, if $(\lfloor \textbf{let } \Pi \textbf{ in } C_1 \rfloor, (\sigma_c, \sigma_o, \circledcirc)) \overset{T}{\longmapsto}{}^{\omega} \cdot$, by $\mathsf{seq\text{-}term}_\varphi(\Pi)$, we know $\mathsf{lock\text{-}free}(T)$. Then from (B.19) we get: there exists T_a such that

$$(\lfloor \textbf{let } \Pi_A \textbf{ in } C_1 \rfloor, (\sigma_c, \sigma_a, \circledcirc)) \overset{T_a}{\longmapsto}{}^{\omega} .$$

and $\mathsf{get\_obsv}(T) = \mathsf{get\_obsv}(T_a)$, thus we are done.