Weak Progressive Forward Simulation Is Necessary and Sufficient for Strong Observational Refinement

Brijesh Dongol
University of Surrey, UK

Gerhard Schellhorn
Universität Augsburg, Germany

Heike Wehrheim
Universität Oldenburg, Germany

Abstract

Hyperproperties are correctness conditions for labelled transition systems that are more expressive than traditional trace properties, with particular relevance to security. Recently, Attiya and Enea studied a notion of strong observational refinement that preserves all hyperproperties. They analyse the correspondence between forward simulation and strong observational refinement in a setting with only finite traces. We study this correspondence in a setting with both finite and infinite traces. In particular, we show that forward simulation does not preserve hyperliveness properties in this setting. We extend the forward simulation proof obligation with a (weak) progress condition, and prove that this *weak progressive forward simulation* is equivalent to strong observational refinement.

2012 ACM Subject Classification Theory of computation → Semantics and reasoning; Theory of computation → Concurrency; Security and privacy → Formal methods and theory of security

Keywords and phrases Strong Observational Refinement, Hyperproperties, Forward Simulation, Weak Progressiveness

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.31

Funding *Brijesh Dongol*: EPSRC grants EP/V038915/1 and EP/R032556/1, VeTSS and ARC Discovery Grant DP190102142.
Heike Wehrheim: DFG Grant WE2290/12-1.

Acknowledgements We thank John Derrick, Simon Doherty, Constantin Enea and our anonymous CONCUR reviewers for their comments on earlier versions of this paper.

1 Introduction

Linearizability [19] has become a standard safety condition for concurrent objects that access shared state. Golab, Higham and Woelfel [13] however showed that linearizability does *not* preserve probability distributions in randomised algorithms. They therefore proposed a notion called *strong linearizability*, which unlike linearizability, must use the same linearization order for *every* prefix of a linearizable history. Strong linearizability allows consideration of concurrent objects in the presence of *adversaries* and can – amongst others – be used to show the preservation of security properties. Here, the adversary is modelled by an *adversarial scheduler*, which plays the role of a *strong adversary* [1].

Our security properties of interest are *hyperproperties* [5], which are properties over *sets of sets of traces* (analogous to trace properties, which are over sets of traces). Hyperproperties allow characterisation, for instance, of information flow properties such as non-interference and observational determinism. Like trace properties, which can be characterised by a conjunction of a safety and a liveness property, every hyperproperty can be characterised as the conjunction of a hypersafety and hyperliveness property. For instance, as observed

33rd International Conference on Concurrency Theory (CONCUR 2022).
Editors: Bartek Klin, Sławomir Lasota, and Anca Muscholl; Article No. 31; pp. 31:1–31:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

by Clarkson and Schneider [5], *observational determinism* [33] is a hypersafety property, *possibilistic information flow* [23] is a hyperliveness property, and *Goguen and Meseguer's noninterference* property [12] is a conjunction of a hypersafety and hyperliveness property.

Attiya and Enea [2] revisited preservation of hyperproperties in the context of concurrent objects and proposed a generalisation of strong linearizability called *strong observational refinement*. They showed that strong observational refinement preserves *all* hyperproperties, when replacing an abstract library specification, A, by a concrete library implementation, C, in a client program, P. Here, C strongly observationally refines A iff the executions of *any* client program P using C as scheduled by *some* scheduler cannot be observationally distinguished from those of P using A under another scheduler[1].

A second claim in [2] is that forward simulation [22] is *equivalent* to strong observational refinement, i.e., it is both necessary and sufficient. The claim is motivated with examples using (hyper)safety properties, however it raises questions for (hyper)liveness. It turns out, that the study of strong observational refinement and forward simulation by Attiya and Enea is in the restricted setting of finite traces[2], though this restriction is unclear in their paper [2]. Thus, all hyperproperties considered by Attiya and Enea are hypersafety properties, which leaves out a large class of hyperproperties. We described the problem, namely that forward simulation does *not* preserve hyperliveness properties in out recent brief announcement [8]. There, we also proposed a new condition called *progressive forward simulation* that strengthens forward simulation so that it preserves all hyperproperties through refinement (i.e., progressive forward simulation is a *sufficient* condition).

Our point of departure for this paper is the question in the other direction: "Is progressive forward simulation necessary for strong observational refinement?" The answer, it turns out, is no! As we shall see in §4.1, it is possible for a concrete object to be a strong observational refinement of some abstract object, yet for there to be no progressive forward simulation between them.

Contributions

In this paper, we present a relaxation of progressive forward simulation that is both necessary and sufficient. Our main contribution therefore is a new result that closes the gap between strong observational refinement and a corresponding proof technique between concurrent objects. In particular, we provide, for the first time, a stepwise technique that coincides with a notion of refinement that preserves all client-object hyperproperties.

Overview

In §2 we present our main example to demonstrate the inadequacy of forward simulation for hyperliveness properties. §3 presents the formal background and recaps the key definitions and prior results. §4 motivates and defines weak progressive forward simulation, which we prove to be both sufficient (§5) and necessary (§6) for strong observational refinement.

2 Motivating Example

We start by giving an example of an abstract atomic object A and a non-atomic implementation C such that there *is* a forward simulation from C to A, but hyperliveness properties are not preserved for all schedules.

[1] Both of these schedulers are additionally required to be *admissible* and *deterministic* (see §3.2).
[2] Private communication

```
int* current_val initially 0

int fetch_and_inc():
F1. do
F2.   n = LL(&current_val)
F3. while (!SC(&current_val, n + 1))
F4. return n
```

Figure 1 A fetch-and-inc implementation with a nonterminating schedule when `LL` and `SC` are implemented using the algorithm of [20].

As the atomic abstract object A we choose a *fetch-and-inc* object with just one operation, `fetch_and_inc()`, which increments the value of a shared integer variable and returns the value of that variable before the increment. Let P be a program with two threads t_1 and t_2, each of which executes one `fetch_and_inc` operation and assigns the return value to a local variable of the thread. Clearly, for any scheduler S, the variable assignment of both threads will eventually occur. This "eventually" property can be expressed as a hyperproperty.

Now, consider the *fetch-and-inc* implementation presented in Figure 1. This implementation uses the `load-linked/store-conditional` (LL/SC) instruction pair. The `LL(ptr)` operation loads the value at the location pointed to by the pointer `ptr`. The `SC(ptr,v)` conditionally stores the value `v` at the location pointed to by `ptr` if the location has not been modified by another `SC` since the executing thread's most recent `LL(ptr)` operation. If the update actually occurs, `SC` returns `true`, otherwise the location is not modified and `SC` returns `false`. In the first case, we say that the `SC` *succeeds*. Otherwise, we say that it *fails*.

Critically, we stipulate that the LL and SC operations are implemented using the algorithm of [20]. This algorithm has the following property. If thread t_1 executes an `LL` operation, and then thread t_2 executes an `LL` operation *before* t_1 has executed its subsequent `SC` operation, then that `SC` is guaranteed to fail. This happens even though there is no intervening modification of the location.

Now, let C be a *labelled transition systems* (LTS) representing a multithreaded version of this `fetch_and_inc` implementation, using the specified `LL`/`SC` algorithm[3]. Figure 2 gives a sketch of this LTS, detailing just the most important actions. Consider furthermore the program P (above) running against the object O_1. A scheduler can continually alternate the `LL` at line `F2` of t_1 and that of t_2 (with some executions of `F3` in between), such that neither `fetch_and_inc` operation ever completes (see the blue arrows in the LTS). Therefore, unlike when using the A object, the variable assignments of P will never occur, so the C system does not satisfy the hyperproperty for all schedulers.

There is, however, a forward simulation (see Definition 3.2) from C to A. Therefore, standard forward simulation is insufficient to show that all hyperproperties are preserved.

3 Background

Notation. Let ξ and ξ' be sequences. The *empty sequence* is denoted ε and the *length* of ξ denoted $\#\xi$. We write $\xi \sqsubseteq \xi'$ (similarly, $\xi \sqsubset \xi'$) iff ξ is a *prefix* (similarly, *proper prefix*) of ξ'. Assuming $m < n \leq \#\xi$, we write $\xi^{<n}$ for the prefix of ξ of length n and $\xi[m]$ for the element of ξ at index m. Thus, $\xi^{<0} = \varepsilon$, and if $n > 0$, $\xi^{<n} = \xi[0] \cdot \xi[1] \cdot \xi[2] \cdots \xi[n-1]$. If ξ

[3] There are several ways to represent a multithreaded program or object as an LTS, e.g., [21,28].

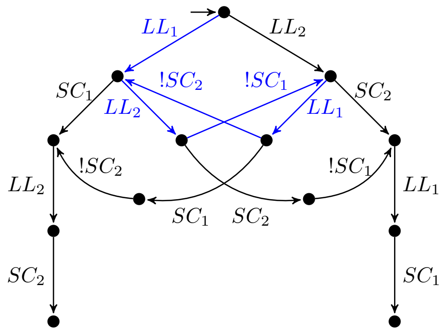

Figure 2 Sketch of labelled transition system for the example in Fig. 1 for threads t_1 and t_2 calling `fetch_and_inc` once (LL_i action for execution of F2 by thread t_i, SC_i for F3 when the `SC` returns true, $!SC_i$ when it returns false, $i \in \{1, 2\}$).

is finite, we let $\mathit{last}(\xi)$ be the last element of ξ, i.e., $\mathit{last}(\xi) = \xi[\#\xi - 1]$. For a set S, let $\xi|_S$ be the sequence ξ restricted to elements in S. We lift this to sets of sequences and define $T|_S = \{\xi|_S \mid \xi \in T\}$. Let $x^\omega = x \cdot x \cdot x \cdots$ be the infinite sequence comprising the element x.

3.1 LTSs, refinement and forward simulation

We describe (concurrent) systems by *labelled transition systems* (LTSs). An LTS $L = (Q, Q^{ini}, \Sigma, \delta)$ consists of a (possibly infinite) set of *states* Q, an alphabet Σ of *actions*, initial states $Q^{ini} \subseteq Q$ and a transition relation $\delta \subseteq Q \times \Sigma \times Q$. We say that an action a is enabled in state q iff there exists a state q' such that $(q, a, q') \in \delta$. Labelled transition systems give rise to (finite or infinite) *runs* which are alternating sequences $q_0 \cdot a_1 \cdot q_1 \cdot a_2 \cdot \ldots$ of states and actions with $(q_i, a_{i+1}, q_{i+1}) \in \delta$. We also write $q_0 \xrightarrow{a_1 \ldots a_n} q_n$ if there is a finite run $q_0 \cdot a_1 \cdot q_1 \cdots a_n \cdot q_n$. In particular, $q \xrightarrow{\varepsilon} q$. A run is an *execution* of an LTS L if $q_0 \in Q^{ini}$.

A *trace* is the sequence of actions of an execution and the set of traces of an LTS L is denoted $T(L)$, which may be partitioned into *finite* traces, denoted Σ^*, and *infinite* traces, denoted Σ^ω. We use $\sigma \in \Sigma^*$ and $\pi \in \Sigma^\omega$ when refering to finite and infinite traces, respectively, and $\rho \in T(L)$ to refer to a trace that may be finite or infinite. Note that for any L, $T(L)$ is prefix closed.

An LTS is *step-deterministic* if it has a single initial state (i.e., $Q^{ini} = \{q^{ini}\}$), and for every state q and action a, if $q \xrightarrow{a} q'$ and $q \xrightarrow{a} q''$ then $q' = q''$. Step-determinism implies that each trace corresponds to a unique run; if the trace is finite then there is at most one state q' such that $q^{ini} \xrightarrow{\sigma} q'$. In this case, we let $\mathit{state}(\sigma)$ denote q'.[4]

Like [2], we use step-deterministic LTSs to describe *objects* and the *programs* that use these objects. The terms "object" and "program" are taken from [2], the object describing a library (e.g. of a data structure) and the program using this library by calling operations of it. To define interfaces between objects and programs, we partition the actions of an LTS into *internal* and *external* actions. Objects offer operations to their environment which can be invoked (by programs) using an external invocation action from a set I with a corresponding external response action from a set R. For this paper, the exact form of invocation and response actions is unimportant. Besides invocations and responses, an object may have further internal actions used to implement the operations.

[4] Note that a step-deterministic LTS differs from the notion of a deterministic automaton of Lynch and Vaandrager [22]. Attiya and Enea simply refer to step-deterministic LTSs as deterministic LTSs [2].

A program *uses* objects by invoking their operations and waiting for the corresponding responses. Thus, if P is an LTS corresponding to a program its actions can be partitioned as follows: $\Sigma_P = I \mathbin{\dot\cup} R \mathbin{\dot\cup} \Gamma_P$, where $\dot\cup$ is a disjoint union and Γ_P is the set of program actions. The composition of a program with an object is formally defined as the *product* of two LTSs.

▶ **Definition 3.1** (Program-object composition). *Suppose that* $P = (Q_P, Q_P^{ini}, \Sigma_P, \delta_P)$ *and* $O = (Q_O, Q_O^{ini}, \Sigma_O, \delta_O)$ *are LTSs. The* product *of* P *with* O*, denoted* $P \times O$*, is the LTS* $(Q, Q^{ini}, \Sigma, \delta)$ *with*

- $Q = Q_P \times Q_O$, $Q^{ini} = Q_P^{ini} \times Q_O^{ini}$, $\Sigma = \Sigma_P \cup \Sigma_O$, *and*
- $\delta = \bigcup_{a \in \Sigma_P \cap \Sigma_O} \{(q_P, q_O) \xrightarrow{a} (q'_P, q'_O) \mid q_P \xrightarrow{a}_P q'_P \wedge q_O \xrightarrow{a}_O q'_O\} \cup$
  $\bigcup_{a \in \Sigma_P \setminus \Sigma_O} \{(q_P, q_O) \xrightarrow{a} (q'_P, q_O) \mid q_P \xrightarrow{a}_P q'_P\} \cup$
  $\bigcup_{a \in \Sigma_O \setminus \Sigma_P} \{(q_P, q_O) \xrightarrow{a} (q_P, q'_O) \mid q_O \xrightarrow{a}_O q'_O\}$

Note that $\Sigma_P \cap \Sigma_O$ in our case will typically be $I \cup R$.

An object can either be an abstract (often sequential) specification (denoted L_A or simply A) or a concrete implementation (denoted L_C or C). A *history* of an LTS is a sequence $\rho|_\Delta$, where ρ is a trace of the LTS and $\Delta \subseteq \Sigma$ is the set of external actions. We formally relate the behaviours of A and C by comparing their histories. We say C is a *Δ-refinement* of A iff $T(C)|_\Delta \subseteq T(A)|_\Delta$. One can establish Δ-refinement between C and A by proving forward simulation between the systems.[5]

▶ **Definition 3.2** (Forward simulation). *Let* C *and* A *be two LTSs with sets of actions* Σ_C *and* Σ_A*, respectively, and let* $\Delta \subseteq \Sigma_C \cap \Sigma_A$*. A relation* $F \subseteq Q_C \times Q_A$ *is a* Δ-forward simulation *from* C *to* A *iff both of the following hold:*

Initialisation. $(q_C^{ini}, q_A^{ini}) \in F$,

Simulation step. *For all* $(q_C, q_A) \in F$*, if* $q_C \xrightarrow{a}_C q'_C$ *then there exist* $\sigma \in \Sigma_A^*$ *and* $q'_A \in Q_A$ *such that* $a|_\Delta = \sigma|_\Delta$*,* $q_A \xrightarrow{\sigma}_A q'_A$ *and* $(q'_C, q'_A) \in F$*.*

Note that σ in the above definition may be ε in which case the condition $a|_\Delta = \sigma|_\Delta$ reduces to $a|_\Delta = \varepsilon$. In this case, the proof obligation for the simulation step forms a *triangular diagram*. For instance, in Figure 4, the step executing τ_3 forms such as diagram.

▶ **Lemma 3.3** (Lynch [21]). *If there is a Δ-forward simulation from C to A, then* $T(C)|_\Delta \subseteq T(A)|_\Delta$.

3.2 Strong Observational Refinement

Attiya and Enea [2] have proposed the notion of *strong observational refinement*, which is a strengthening of refinement (and generalisation of strong linearizability [13]) that preserves all hyperproperties. Strong observational refinement is defined in terms of an adversary and is modelled by a scheduler that is assumed to have full control over a step-deterministic LTS's execution.

Formally, a *scheduler* for an LTS is a function $S : \Sigma^* \to 2^\Sigma$ that determines the next action to be executed based on the sequence of actions that have been executed thus far. A trace ρ is *consistent* with a scheduler S if $\rho[n] \in S(\rho^{<n})$ for all $n < \#\rho$. We write $T(L, S)$ for the set of traces of L that are consistent with S. A scheduler is *admitted* by an LTS L if for all finite traces σ of L consistent with S, the scheduler satisfies

[5] It is well known that forward simulation is sound for proving refinement, but completeness requires both forward and backward simulation [7, 22].

1. $S(\sigma)$ is non-empty and
2. all actions in $S(\sigma)$ are enabled in $state(\sigma)$.

The scheduled traces in $T(L, S)$ can alternatively be viewed as the traces of a product $L \times LTS(S)$, where $LTS(S)$ is an LTS generated from S with set of states Σ^*; initial state ε; and transitions $\sigma \xrightarrow{a} \sigma \cdot a$, where $a \in S(\sigma)$.

In addition to being admissible, the schedulers we consider (for the combination of program with object, $P \times O$) must be *deterministic*: they must deterministically choose one of the enabled actions of the object. A scheduler S for $P \times O$ is *deterministic* if either (i) $S(\sigma) \subseteq \Gamma_P$ (i.e., it can choose several program actions, excluding invocations and responses of object operations) or (ii) $|S(\sigma)| = 1$ (i.e., if S chooses an action of O, including invocations and responses, then it chooses exactly one).

Now we are ready to define strong observational refinement.

▶ **Definition 3.4** (Strong observational refinement). *An object C* strongly observationally refines *an object A, written $C \leq_s A$, iff for every program P and every deterministic scheduler S_C admitted by $P \times C$, there exists a deterministic scheduler S_A admitted by $P \times A$ such that $T(P \times C, S_C)|_{\Gamma_P} = T(P \times A, S_A)|_{\Gamma_P}$.*

Note that unlike Attiya and Enea [2], this definition of strong observational refinement considers infinite traces, which is necessary for preservation of all hyperproperties. In this setting, as discussed in §2, forward simulation is no longer necessary and sufficient for establishing strong observational refinement (contrasting the results of Attiya and Enea [2]).

4 A necessary and sufficient condition

We now motivate and develop the notion of *weak progressive forward simulation*, providing a proof method for strong observational refinement. We first recap progressive forward simulation [8] and show that it is *not* a necessary condition (§4.1). Our new relaxed definition is given in §4.2.

4.1 Progressive forward simulation is too strong

In [8], we developed a condition called *progressive forward simulation* that enhances forward simulation with a well-founded order that rules out infinite stuttering. It is guaranteed by an implementation, e.g., when the underlying implementation is lock-free [6, 18]. First we provide the formal definition of progressive forward simulation.

▶ **Definition 4.1** (Progressive Forward Simulation [8]). *Let C and A be two deterministic LTSs and $\Delta \subseteq \Sigma_C \cup \Sigma_A$. A relation $F \subseteq Q_C \times Q_A$ together with a well-founded order $\gg \subseteq Q_C \times Q_C$ is called a* progressive Δ-forward simulation *from C to A iff*

Initialisation. $(q_C^{ini}, q_A^{ini}) \in F$,

Step. *For all $(q_C, q_A) \in F$, if $q_C \xrightarrow{a}_C q'_C$ then there exist $\sigma \in \Sigma_A^*$ and $q'_A \in Q_A$ such that*

- **Simulation.** $a|_\Delta = \sigma|_\Delta$, $q_A \xrightarrow{\sigma}_A q'_A$ *and* $(q'_C, q'_A) \in F$, *and*
- **Progressiveness.** *if* $\sigma = \varepsilon$ *then* $q_C \gg q'_C$.

In [8], we have additionally shown that progressive forward simulation is *sufficient* for strong observational refinement.

▶ **Theorem 4.2** (Sufficiency [8]). *If there exists a progressive forward simulation between C and A, then $C \leq_s A$.*

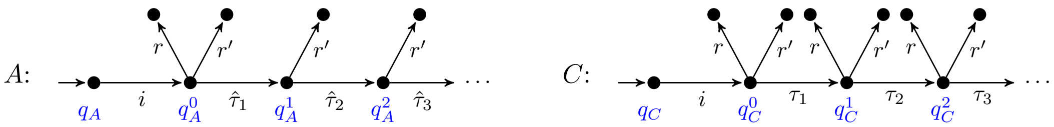

Figure 3 Two objects A and C such that $C \leq_s A$, but there does not exist a progressive forward simulation between C and A.

The main motivation for this paper has been the pursuit of a proof in the other direction, i.e., that progressive forward simulation is also *necessary* for strong observational refinement. However, it turns out that strong observational refinement does *not* imply the existence of a progressive forward simulation.

To see this consider the labelled transition systems depicted in Figure 3. We assume an abstract object A and concrete implementation C with a single operation, external actions $i, r, r' \in I \cup R$, abstract internal actions $\hat\tau_k \in \Sigma_A \setminus (I \cup R)$, and concrete internal actions $\tau_k \in \Sigma_C \setminus (I \cup R)$. The objects A and C differ in that C continually allows both r and r' after c, whereas A only allows both r and r' immediately after i; it stops offering r after $\hat\tau_1$.

It is straightforward to show that $C \leq_s A$:

1. if C generates a run $q_C \cdot i \cdot q'_C \cdot r$ or $q_C \cdot i \cdot q'_C \cdot r'$ without executing any internal actions after i, a corresponding run can clearly be generated by A;
2. if C generates a run $q_C \cdot i \cdot q_C^0 \cdot \tau_1 \cdots \tau_n \cdot q_C^n \cdot r$ or $q_C \cdot i \cdot q_C^0 \cdot \tau_1 \cdots \tau_n \cdot q_C^n \cdot r'$ executing the internal actions $\tau_1,\ldots, \tau_n$, then a corresponding run can be executed in A by not executing any of the internal actions of A;
3. if C generates a (diverging) run $q_C \cdot i \cdot q_C^0 \cdot \tau_1 \cdot q_C^1 \cdot \tau_2 \cdots$ that never responds, a corresponding run $q_A \cdot i \cdot q_A^0 \cdot \hat\tau_1 \cdot q_A^1 \cdot \hat\tau_2 \cdots$ can be generated in A.

Thus, for any program P and scheduler S_C, there exists a scheduler S_A such that $T(P \times C, S_C)|_{\Gamma_P} = T(P \times A, S_A)|_{\Gamma_P}$.

Now we show that there does not exist a progressive forward simulation. First, there exists a forward simulation, F, that allows each τ_k to behave as the corresponding $\hat\tau_k$ and as a stuttering step, i.e., $(q_C^j, q_A^0) \in F$ for each j. However, there is no well-founded ordering over the states since stuttering is unbounded, thus progressiveness cannot be guaranteed. The problem is that progressiveness enforces **2** above, but does not account for the possibility of **3**.

The main result of this work is a weaker form of progressive forward simulation that we prove to *coincide* with strong observational refinement. *Weak progressiveness* does not necessitate a well-founded order when the concrete implementation executes an infinite number of consecutive internal actions provided that the abstraction can also execute an infinite number of consecutive internal actions. This relaxation accounts for the scenario highlighted by **3** above.

4.2 Weak progressive forward simulation

In our LTSs, distinguishing between internal and external actions allows us to define *divergent* states. State q is Δ-divergent (written $q \xrightarrow{\infty}_{\setminus\Delta}$) if there exists an infinite run $q \cdot a_1 \cdot q_1 \cdot a_2 \cdots$ such that $a_i \in \Sigma \setminus \Delta$ for all $i \geq 1$.

We therefore obtain the following definition of weak progressive forward simulation, which relaxes the progressiveness condition from Definition 4.1.

▶ **Definition 4.3** (Weak Progressive Forward Simulation). *Let C and A be two deterministic LTSs and $\Delta \subseteq \Sigma_C \cup \Sigma_A$. A relation $F \subseteq Q_C \times Q_A$ together with a well-founded order $\gg \subseteq Q_C \times Q_C$ is called a* weak progressive Δ-forward simulation *from C to A iff*

Initialisation. $(q_C^{ini}, q_A^{ini}) \in F$,

Step. *For all $(q_C, q_A) \in F$, if $q_C \xrightarrow{a}_C q'_C$ then there exist $\sigma \in \Sigma_A^*$ and $q'_A \in Q_A$ such that*

- **Simulation.** $a|_\Delta = \sigma|_\Delta$*,* $q_A \xrightarrow{\sigma}_A q'_A$ *and* $(q'_C, q'_A) \in F$*, and*
- **Weak progressiveness.** *if $\sigma = \varepsilon$ then either $q_C \gg q'_C$ or $q_A \xrightarrow{\infty}_{\setminus\Delta}$.*

Note that we do not require that any triangular diagram with $q_C \xrightarrow{a}_C q'_C$, $(q_C, q_A) \in F$, and $(q'_C, q_A) \in F$ with no diverging trace from q_A to have $q_C \gg q'_C$. We only require this if there is no other $(q'_C, q'_A) \in F$ with $q_A \xrightarrow{\sigma}_A q'_A$ and $\sigma \neq \varepsilon$.

Also note that for concrete LTSs without any divergence, all three notions (forward simulation, strong observational refinement and weak progressive forward simulation) coincide (because without divergence $s \gg s'$ iff $s \xrightarrow{\tau} s'$ for some internal action τ is a well-founded order). For example, progress conditions such as *lock-freedom* [18] would be sufficient to ensure absence of divergence in the concurrent object (see also [11, 14, 17]).

This definition weakens the progressiveness condition: Either the concrete state must decrease in the well-founded order, or q_C corresponds to a q_A that diverges. A standard forward simulation would allow one to relate all concrete states of the diverging run to q_A, "hiding" the divergence. Weak progressiveness ensures that divergence on the concrete level is not possible without a corresponding diverging run from q_A. Our earlier definition of a *progressive forward simulation* always required the well-founded ordering to decrease for a stuttering concrete transition. With this change in place, we can now show equality of strong observational refinement and this form of forward simulation.

▶ **Theorem 4.4.** *$C \leq_s A$ iff there exists a weak progressive $(I \cup R)$-forward simulation from C to A.*

The rest of the paper is now devoted to proving this theorem. We prove sufficiency in §5 and necessity in §6.

5 Weak Progressive Forward Simulation implies Strong Observational Refinement

We start with the sufficiency of weak progressive forward simulation for strong observational refinement. This proof is an adaptation of the proof for progressive forward simulation [8, 9], so we relegate the details to the appendix.

▶ **Theorem 5.1.** *If there exists a weak progressive $I \cup R$-forward simulation from C to A, then $C \leq_S A$.*

Given two LTLs C and A for which a weak progressive forward simulation $(F, \gg)$ exists and given an arbitrary program P together with a scheduler S_C for traces over $P \times C$, our proof has to construct a scheduler S_A such that $T(P \times C, S_C)|_{\Gamma_P} = T(P \times A, S_A)|_{\Gamma_P}$. The construction is in two steps: First a function f is constructed that maps traces $\rho_C \in T(P \times C, S_C)$ to traces $f(\rho_C) \in T(P \times A)$, such that the executed program actions in Σ_P are the same for both traces.

The construction is shown in Fig. 4. Steps of C are mapped to fixed steps of A using a mapping m (a formal definition is in the appendix), such that the forward simulation is preserved. Program steps in Γ_P are mapped by identity, such that the program states of both traces are always equal. For finite traces $\rho_C \in T(P \times C, S_C)$ this results in a finite trace with the same program steps.

$$
\begin{tikzcd}
 & & & m(q_C^1, \tau_3, q_A^1) = \epsilon & & & \\
f(\pi_C): & (q_P^{ini}, q_A^{ini}) \arrow[r, "{m(q_C^{ini}, a_1, q_A^{ini}) = a_1}"] \arrow[d, dash, "F"] & (q_P^1, q_A^{ini}) \arrow[r, "{m(q_C^{ini}, i_2, q_A^{ini}) = \alpha_1 i_2 \alpha_2}"] \arrow[d, dash, "F"] & (q_P^2, q_A^1) \arrow[rr, "{m(q_C^2, \tau_4, q_A^1) = \alpha}"] \arrow[d, dash, "F"] \arrow[dr, dash, "F"] & & (q_P^2, q_A^3) \arrow[r] \arrow[d, dash, "F"] & \cdots \\
\pi_C: & (q_P^{ini}, q_C^{ini}) \arrow[r, "a_1 \in \Gamma_P"'] & (q_P^1, q_C^{ini}) \arrow[r, "i_2 \in I"'] & (q_P^2, q_C^1) \arrow[r, "\tau_3"'] & (q_P^2, q_C^2) \arrow[r, "\tau_4"'] & (q_P^2, q_C^3) \arrow[r] & \cdots
\end{tikzcd}
$$

Figure 4 Constructing $f(\pi_C) \in T(P \times A)$ from $\pi_C \in T(P \times C, S_C)$ with $\tau_3, \tau_4 \in \Sigma_C \setminus (I \cup R)$, $\alpha, \alpha_1, \alpha_2 \in (\Sigma_A \setminus (I \cup R))^*$ and $q_C^1 \gg q_C^2$.

Infinite traces $\pi_C \in T(P \times C, S_C)$ are mapped either to infinite traces directly, or by exploiting that the forward simulation is weakly progressive: if the abstract trace is finite, ending with $(q_P,\ q_A)$ while the concrete trace ends with an infinite sequence of stuttering steps, then a diverging run of A from q_A is guaranteed to exist. This run (where all A-states are combined with q_P) can then be attached at the end to give an infinite trace which is defined to be $f(\pi_C)$. Again, π_C and $f(\pi_C)$ will have the same program steps.

A formal definition of f will be given in the appendix. Given f, an abstract scheduler S_A can be defined that schedules exactly all the steps of $f(\rho_C)$. For this definition to be well-defined it is crucial that the traces in the image of f form a tree-shaped structure, where branching points are at program actions only. We have the following theorem.

▶ **Theorem 5.2.** $T(P \times A, S_A) = \{\sigma_A \mid \exists \pi_C \in T(P \times C, S_C).\ \sigma_A \sqsubseteq f(\pi_C)\}$.

Theorem 5.1 then is a simple consequence, since both π_C and $f(\pi_C)$ have the same program actions, and each $\pi_A \in T(P \times A, S_A)$ is some $f(\pi_C)$ as stated by Theorem 5.2.

6 Strong Observational Refinement implies Weak Progressive Forward Simulation

We now prove the necessity theorem for weak progressive forward simulation.

▶ **Theorem 6.1.** *If $C \leq_s A$, then there exists a weak progressive $(I \cup R)$-forward simulation from C to A.*

Given that C is a strong observational refinement of A, we must show that a weak progressive forward simulation exists between A and C. Since we are tasked with finding a forward simulation, we must also instantiate a client program and concrete scheduler that act as witnesses to the forward simulation. Our proof proceeds in stages (see Figure 5) for a client program P that invokes the operations of the object in question and concrete scheduler S_C. This method is similar to that of Attiya and Enea [2], but the underlying formal mechanisms have been completely reworked.

Client Program and Concrete Scheduler

The program for concrete object $C = (Q_C, Q_C^{ini}, \Sigma_C, \delta_C)$ that we use is given by the LTS $P = (Q_P, Q_P^{ini}, \Sigma_P, \delta_P)$, where

- $Q_P = \{q_P^{ini}, q_{div}\} \cup \{q_e, q_{rec(e)} \mid e \in I \cup R\}$,
- $Q_P^{ini} = \{q_P^{ini}\}$,
- $\Gamma_P = \{div\} \cup \{g(q), g(e, q), rec(e) \mid q \in Q_C \wedge e \in I \cup R\}$,

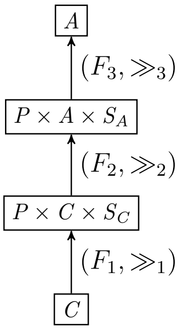

Figure 5 Proof overview.

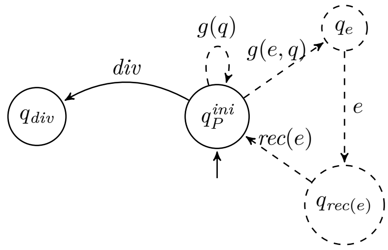

Figure 6 Representation of client P used as a witness to the forward simulation, where $q \in Q_C$ and $e \in I \cup R$.

- $\Sigma_P = \Gamma_P \cup I \cup R$,
- $\delta_P = \{q_P^{ini} \xrightarrow{div} q_{div}\} \cup$
  $\bigcup_{e \in I \cup R, q \in Q_C} \{q_P^{ini} \xrightarrow{g(e,q)} q_e, q_e \xrightarrow{e} q_{rec(e)}, q_{rec(e)} \xrightarrow{rec(e)} q_P^{ini}, q_P^{ini} \xrightarrow{g(q)} q_P^{ini}\}$

This program is depicted in Figure 6, where dashed states and transitions are used to denote *families* of states and transitions. We refer to $g(q)$ and $g(e,q)$ as *guess actions* and $rec(e)$ as *record actions*. These are used to make the (internal) choices made by the client program, concrete object and scheduler visible in the traces of $T(P \times C \times S_C)|_{\Gamma_P}$. Additionally, we use an external action *div* that is enabled whenever the underlying object can diverge, i.e., *div* is enabled in q_C iff $q_C \xrightarrow{\infty}_{\setminus(I \cup R)}$. The program P can only synchronise with *div* when it is in state q_P^{ini}. Once executed, the program transitions to state q_{div} and from this point, it is only possible to schedule *internal actions* of C in $P \times C$.

We now define a particular admissible scheduler. The concrete scheduler S_C must schedule the actions of $P \times C$, i.e., define the next action for a given trace $\sigma \in (\Sigma_P \cup \Sigma_C)^*$.

$$S_C(\sigma) = \begin{cases} \{\tau\} & \text{if } div \in \sigma,\ \tau \in \Sigma_C \setminus (I \cup R), \text{ and} \\ & \quad \exists q_C.\ state(\sigma|_{\Sigma_C}) \xrightarrow{\tau_C} q_C \wedge q_C \xrightarrow{\infty}_{\setminus I \cup R} \\ \{\tau\} & \text{else if } last(\sigma) = g(q_C) \text{ and} \\ & \quad \tau \in \Sigma_C \setminus (I \cup R) \wedge state(\sigma|_{\Sigma_C}) \xrightarrow{\tau} q_C \\ \{e\} & \text{else if } last(\sigma) = g(e, q_C) \\ \{rec(e)\} & \text{else if } last(\sigma) = e \\ \{div \mid state(\sigma|_{\Sigma_C}) \xrightarrow{\infty}_{\setminus I \cup R}\} \cup & \text{otherwise} \\ \{g(q_C) \mid \exists \tau \in \Sigma_C \setminus (I \cup R).\ state(\sigma|_{\Sigma_C}) \xrightarrow{\tau} q_C\} & \\ \cup \{g(e, q_C) \mid \exists e \in I \cup R.\ state(\sigma|_{\Sigma_C}) \xrightarrow{e} q_C\} & \end{cases}$$

Note that $state(\sigma|_{\Sigma_C})$ is calculated wrt the LTS C as opposed to the composition $P \times C$. Furthermore, the scheduler decides on the next action of $P \times C$ based on the last action in the given trace σ. The first two cases determine the next action of C depending on whether the program has executed *div*. In the first two cases, there may be a choice of τ, the scheduler chooses one such that the resulting set is a singleton. Clearly the third case results in a singleton set since the action $g(e, q_C)$ fixes the only external action e allowed by the scheduler. Therefore, the scheduler S_C defined above is deterministic.

The last two cases describe the scheduler's behaviour wrt to a program action. As per Figure 6, these are the record and guess actions as well as *div*. In the fourth case, action $rec(e)$ must be scheduled if the last action executed in σ is e. In the final case, the scheduler may choose to diverge (if the program diverges), perform a guess action, $g(q_C)$ (corresponding to an internal transition of C) or a guess action, $g(e, q_C)$ (corresponding to an external transition of C). Note that for both $g(q_C)$ and $g(e, q_C)$, state q_C is the post state of the given action after executing from $state(\sigma)$.

By design, we therefore have the following proposition for $P \times C$.

▶ **Proposition 6.2.** *Let $\sigma \in T(P \times C, S_C)$ for the scheduler S_C. If $div \in \sigma$ holds then $state(\sigma|_{\Sigma_C}) \xrightarrow{\infty}_{\setminus(I\cup R)}$ and $(\exists\, q'_C.\ state(\sigma|_{\Sigma_C}) \xrightarrow{S_C(\sigma)} q'_C \wedge q'_C \xrightarrow{\infty}_{\setminus(I\cup R)})$.*

Simulation $(F_1, \gg_1)$

Our first step is the construction of a weak progressive forward simulation $(F_1, \gg_1)$ between C and $P \times C \times S_C$ (see Figure 5) with $F_1 \subseteq Q_C \times (Q_P \times Q_C \times \Sigma^*_{P\times C})$. We define F_1 such that $(q_C, (q_P, q_C, \sigma)) \in F_1$ iff

- $q_C^{ini} \xrightarrow{\sigma|_{\Sigma_C}} q_C$,
- $div \notin \sigma$ and $last(\sigma) \notin \{g(q), g(e, q), e \mid q \in Q_C \wedge e \in I \cup R\}$ (so that σ is either empty or ends with $rec(e)$ or an internal action; thus the next step is a guessing step),
- $q_P = q_P^{ini}$, and
- $state(\sigma) = q_C$.

▶ **Lemma 6.3.** *$(F_1, \emptyset)$ is a weak progressive forward simulation from C to $P \times C \times S_C$.*

Simulation $(F_2, \gg_2)$

Our next step is to show that a weak progressive forward simulation $(F_2, \gg_2)$ from $P \times C \times S_C$ to $P \times A \times S_A$ exists, when S_A is any scheduler with $T(P \times C, S_C)|_{\Gamma_P} = T(P \times A, S_A)|_{\Gamma_P}$ that exists due to the assumption $C \leq_s A$. The proof follows from a general completeness result for refinements for Δ-deterministic systems.

▶ **Lemma 6.4.** *If C Δ-refines A and A is Δ-deterministic, then there exists a Δ-forward simulation.*

A Δ-deterministic LTS is one, where every history $h \in \Delta^*$ has a unique state q that can be reached with shortest executions that have history h. (A formal definition of Δ-deterministic LTSs is given in Definition B.1.) Such a shortest execution is empty, if $h = \epsilon$, and otherwise has the last element of h as the action of its last step (note, that this definition of Δ-deterministic is weaker than the ones given in [2] and [22]). Clearly, $P \times A \times S_A$ is Γ_P-deterministic, so the theorem applies. The proof of 6.4 shown in the appendix constructs a forward simulation F_2 that relates all states of $P \times C \times S_C$ that are reached with history h to the unique minimally reachable state of $P \times A \times S_A$ with the same history. F_2 is weak progressive, since S_C never schedules more than one internal action in a row. Our definition of the program P guarantees that F_2 also preserves the actions $e \in I \cup R$, since each of these is followed by the corresponding $rec(e)$ action, that is already preserved.

▶ **Lemma 6.5.** *There exists a weak progressive Σ_P-forward simulation $(F_2, \gg_2)$ between $P \times C \times S_C$ and $P \times A \times S_A$.*

Simulation $(F_3, \gg_3)$

We now define the weak progressive $(I \cup R)$-forward simulation F_3 between $P \times A \times S_A$ and A. The states of $P \times A \times S_A$ includes those of A. Thus we keep the two LTSs synchronised, i.e., the forward simulation is over pairs of the form $((q_P, q_A, \sigma), q_A)$, where $(q_P, q_A, \sigma) \in Q_{P\times A\times S_A}$. For $(q_P, q_A, \sigma) \in Q_{P\times A\times S_A}$, let $F_3 = \{((q_P, q_A, \sigma), q_A) \mid (q_P, q_A, \sigma) \in Q_{P\times A}\}$.

The well-founded ordering that we use is the relation $\gg_3$, where:

1. $(q_P^{ini}, \_, \_) \gg_3 (q_e, \_, \_)$
2. $(q_{rec(e)}, \_, \_) \gg_3 (q_P^{ini}, \_, \_)$
3. $(q_P^{ini}, \_, \_) \gg_3 (q_{div}, \_, \_)$

▶ **Lemma 6.6.** *$(F_3, \gg_3)$ is a weak progressive forward $(I \cup R)$-simulation between $P \times A \times S_A$ and A.*

It is trivial to prove that F_3 is a forward simulation. We therefore focus on a proof of weak progressiveness, which provides further insight into our choice of P and the inclusion of the *div* action in our model.

Note that from $(q_{div}, \_, \_)$, the only possible transition is an $\xrightarrow{a}$ step, where $a \in \Sigma_A \setminus (I \cup R)$, which is non-stuttering. Similarly, from $(q_e, \_, \_)$, the only possible transition is $\xrightarrow{e}$, where $e \in I \cup R$. Thus, when we reach a state that is minimal wrt $\gg_3$, no more stuttering is possible. Any transition from state $(q_{rec(e)}, \_, \_)$ is guaranteed to reduce w.r.t. $\gg_3$, as are transitions corresponding to $g(e, q)$ and *div* from $(q_P^{ini}, \_, \_)$.

This leaves us with transitions corresponding to $g(q)$ from (q_P^{ini}, q_A, σ), which may stutter infinitely often. We we can show that such stuttering only exists if A contains a diverging run from q_A, i.e., *div* is enabled in $(q_P^{ini}, q_A, \sigma) \in Q_{P \times A \times S_A}$.

Suppose there exists an infinite run

$$(q_P^{ini}, q_A, \sigma) \xrightarrow{g(q_1)} (q_P^{ini}, q_A, \sigma \cdot g(q_1)) \xrightarrow{g(q_2)} (q_P^{ini}, q_A, \sigma \cdot g(q_1) \cdot g(q_2)) \xrightarrow{g(q_3)} \ldots .$$

By construction, $P \times A \times S_A$ is an "abstraction" of $P \times C \times S_C$ such that $T(P \times A, S_A)|_\Gamma = T(P \times C, S_C)|_\Gamma$, thus, $(\sigma|_\Gamma) \cdot g(q_1) \cdot g(q_2) \cdot g(q_3) \cdot \cdots \in T(P \times C, S_C)|_\Gamma$. Thus, there exists a q_C such that $last(\sigma) = g(q_C)$ and

$$(q_P^{ini}, q_C, \sigma) \xrightarrow{g(q_1) \cdot \tau_1} (q_P^{ini}, q_1, \sigma \cdot g(q_1) \cdot \tau_1) \xrightarrow{g(q_2) \cdot \tau_2} (q_P^{ini}, q_2, \sigma \cdot g(q_1) \cdot \tau_1 \cdot g(q_2) \cdot \tau_2) \xrightarrow{g(q_3) \cdot \tau_3} \ldots$$

where $\tau_k \in \Sigma_C \setminus (I \cup R)$ for all k. Note that the definition of S_C enforces a $\xrightarrow{g(q_k) \cdot \tau_k}$ transition for each $\xrightarrow{g(q_k)}$ transition in $P \times A \times S_A$. This execution, when restricted to the actions of C corresponds to a diverging run of C:

$$q_C \xrightarrow{\tau_1} q_1 \xrightarrow{\tau_2} q_2 \xrightarrow{\tau_3} \ldots$$

Since this is an infinite run of internal actions, by definition, the action *div* must be offered by $P \times C \times S_C$, and enabled in (q_P^{ini}, q_C, σ). Moreover, since $T(P \times A, S_A)|_\Gamma = T(P \times C, S_C)|_\Gamma$, *div* must also be possible in $P \times A \times S_A$. In particular, *div* must be enabled in (q_P^{ini}, q_A, σ). Now, $P \times A \times S_A$ contains a run with final state σ. Therefore, $P \times A \times S_A$ also contains a run

$$(q_P^{ini}, q_A, \sigma) \xrightarrow{div} (q_{div}, q_A, \sigma) \xrightarrow{\tau_1'} (q_{div}, q_1', \sigma) \xrightarrow{\tau_2'} (q_{div}, q_2', \sigma) \xrightarrow{\tau_3'} \ldots$$

where $\tau_k' \in \Sigma_A \setminus (I \cup R)$ for all k since $P \times A \times S_A$ can no longer schedule any further external actions after executing *div*, i.e., must schedule an internal action. Thus, we must have a diverging run in A as well.

Combined simulation. Finally, to derive at a weak progressive simulation from C to A, we show that the relation of weak progressive forward simulation is transitive.

▶ **Theorem 6.7.** *Let $(F_1, \gg_1)$ be a weak progressive Δ-forward simulation from C to B, and $(F_2, \gg_2)$ one from B to A. Then there exists a weak progressive Δ-forward simulation $(F, \gg)$ from C to A.*

The proof of this theorem uses $F = F_1 \circ F_2$ and $\gg$ as defined by $q_C \gg q'_C$ if $q_C \xrightarrow{a} q'_C$ for an internal action $a \notin \Delta$ such that one of the following two conditions holds:

(1) $\exists q_B.\ (q_C, q_B) \in F_1 \wedge (q'_C, q_B) \in F_1 \wedge q_C \gg_1 q'_C \wedge \neg q_B \xrightarrow{\infty}_{\setminus\Delta}$,

(2) $\exists q_B, \alpha, q'_B, q_A.\ (q_C, q_B) \in F_1 \wedge (q'_C, q'_B) \in F_1 \wedge q_B \xrightarrow{\alpha} q'_B \wedge q_B \gg_2 q'_B$
$\wedge\ (q_B, q_A) \in F_2 \wedge (q'_B, q_A) \in F_2 \wedge \neg q_A \xrightarrow{\infty}_{\setminus\Delta}$

where α is a finite sequence of internal actions.

Case (1) requires a triangular diagram for the lower simulation from B to C with a non-diverging state q_B, where $\gg_1$ decreases. Case (2) requires an arbitrary commuting diagram for the lower simulation, and a triangular diagram for the upper simulation from A to B, where the abstract state q_A does not have a diverging run, and $\gg_2$ decreases.

7 Progressive and weak progressive examples

We now present two example programs to demonstrate the implications of progressive and weak progressive forward simulation on program design. The first satisfies progressive forward simulation (and hence weak progressive forward simulation) w.r.t. its abstract specification, while the second satisfies weak progressive simulation only.

7.1 FAI with Lock-free LL/SC

Consider the FAI implementation from Figure 1, but where the LL/SC is assumed to be lock-free. We refer to this implementation as FAI-LF. Unlike the example in §2, we assume that an `LL` operation executed by one thread does not interfere with an `LL` in another thread. If two concurrent threads have loaded the same `LL` value, then only one `SC` will succeed. Forward simulation for FAI-LF holds for the same reason as FAI. We now define a global well-founded order over states using the technique described in [6], which implies a weak progressive forward simulation. This in turn guarantees that *all* hyperproperties (including hyperliveness) of the abstract specification are preserved by FAI-LF.

The well-founded order is straightforward to define: it is a lexicographic ordering that captures how "close" a thread is to successfully executing a successful `SC` operation. The base of the well-founded ordering guarantees that some thread will successfully execute its `SC` operation. The generic lexicographic scheme is the following, where b, b' are booleans and pc, pc' are program counter values:

$$b \gg_B b' \mathrel{\hat{=}} b \wedge \neg b'$$
$$(b, pc) \gg_L (b', pc') \mathrel{\hat{=}} b \gg_B b' \vee (b' = b \wedge (b \wedge pc \gg_{\checkmark} pc') \wedge (\neg b \wedge pc \gg_{\text{\sffamily X}} pc'))$$

where $\gg_B$ orders *true* before *false* and $\gg_L$ is a lexicographic order with a different orderings on pc depending on whether or not b holds. We instantiate this generic scheme over states as follows, where *current_val*, n_t and pc_t are the variables of the algorithm in Figure 1 for a thread t. In particular, *current_val* corresponds to the shared variable `current_val`, n_t corresponds to the local variable `n` of thread t, and pc_t is the program counter for thread t taking values from the set {`F1`, `F2`, `F3`, `F4`, `idle`}.

$$q \gg q' \mathrel{\hat{=}} \exists t.\ (q(current\_val) = q(n_t), q(pc_t)) \gg_L (q'(current\_val) = q'(n_t), q'(pc_t))$$

All that remains is the instantiation of $\gg_{\checkmark}$ and $\gg_{\text{\sffamily X}}$. The order $\gg_{\checkmark}$ is empty, since $q(current\_val) = q(n_t)$ implies that $q(pc_t) =$ `F3`, and execution of thread t corresponds to a successful `SC` operation. We define `F2` $\gg_{\text{\sffamily X}}$ `F3`, representing a retry since this order allows t to make progress towards making $q(current\_val) = q(n_t)$ true.

7.2 FAI with backoff

We now consider a load-balancing FAI specification, which we refer to as FAI-LB. In this example, we weaken the specification to either perform the FAI or perform an operation $\mathit{backoff}_t$ for a thread t that causes the FAI executed by thread t to be delayed. Abstractly, $\mathit{backoff}_t$ is equivalent to a skip action. Note that although the resulting specification is non-deterministic, the LTS is still step-deterministic since the execution of each action from any state results in exactly one next state.

For FAI-LB, we can prove weak progressive forward simulation even for the obstruction-free FAI implementation from §2. Informally, the interference caused by an `LL` by thread t on another thread's `SC` can be mapped to a $\mathit{backoff}_t$ operation executed by thread t. Thus, although the obstruction-free implementation in §2 has a divergent execution, this execution can be matched by the specification FAI-LB.

8 Related work

The study of refinement, and in particular linearizability [19], in the context of adversaries was initiated by the work of Golab, Higham and Woelfel [13] who observed that replacing atomic objects by linearizable implementations in randomized algorithms [1] does not guarantee the expected substitutability result of linearizability. Instead, the probability distribution of results may differ when using a linearizable implementation instead of an abstract atomic object. The difference is due to the abilities of adversaries scheduling process steps depending on the current system state. To alleviate this problem, Golab, Higham and Woelfel suggested *strong linearizability*, requiring a "prefix preservation" property in addition to the conditions of linearizability.

Following this proposal, Attiya and Enea studied the preservation of *hyperproperties* by linearizability. They proposed the definition of *strong observational refinement* and showed it to (a) preserve all hyperproperties, and (b) to coincide with strong linearizability for atomic abstract specifications. They also proved strong observational refinement to be equivalent to forward simulation. In a brief announcement [8], Derrick et al. gave a counter example to this proof, and provided an alternative result for one direction of the equivalence, proposing progressive forward simulation and proving it to imply strong observational refinement. Our work in this paper closes the missing gap of the relationship between progressive forward simulation and strong observational refinement by proving strong observational refinement to imply (yet another) version of forward simulation, weak progressive forward simulation. In addition, we strengthen the result of Derrick et al. [8] and show that weak progressive forward simulation also implies strong observational refinement, thereby arriving at an equivalence once again.

The relationship between (the standard definition of) linearizability and forward and backward simulation has already been investigated before, with Schellhorn, Wehrheim and Derrick [27, 28] showing linearizability proofs in general to require *both* forward and backward simulations, and Bouajjani et al. [3] studying under what circumstances and how forward simulation alone can be employed. The relationship between observational refinement, safety (linearizability) and progress in the context of atomic objects has been studied in prior works [11, 14]. The use of well-founded orderings to enforce progress for a forward simulation has already been used in the context of ASM refinement [25, 26] and non-atomic refinement [10]. Another form of simulations employing well-founded orders are the normed (forward and backward) simulations of Griffioen and Vaandrager [15, 16]. They require every matching internal (τ) step to decrease a norm defined on a well-founded set. It has been shown that normed forward simulations do not agree with ordinary forward simulations, even on divergence-free LTSs. As weak progressive forward simulations does coincide with forward simulation on divergence-free LTSs, we thus get inequality of normed forward simulation and weak progressive forward simulation.

The study of notions of refinement and equivalence taking internal actions into account has been actively pursued in the field of process algebras, with weak bisimulation [24] for CCS and failures-divergences refinement [4] for CSP being the two most prominent examples. Failures-divergences refinement explicitly considers divergences (i.e., infinite sequences of internal actions) during the comparison. For bisimulation, there are also extensions for divergence, e.g. [31, 32]. A comparison of various such semantic equivalences and preorders for systems with internal actions has been given by van Glabbeek [30]. Finally, the game-theoretic characterisation of bisimulation [29] is in spirit similar to the idea of adversaries in strong observational refinement which try to bring the concrete object into an execution that can or cannot be mimicked by the abstract object.

9 Conclusion

In this paper, we have proposed a new type of forward simulation which is both necessary and sufficient for strong observational refinement, thereby closing an existing gap. The importance of strong observational refinement lays in the fact that it preserves safety and liveness hyperproperties which are themselves of fundamental significance for the area of security. As future work, we plan to look at concrete case studies, and to this end will develop a formalization of weak progressive forward simulation within a theorem prover. We furthermore plan to re-investigate the third contribution of Attiya and Enea [2], namely the fact that strong linearizability coincides with strong observational refinement for atomic abstract objects. Since the proof of this property assumes equality of forward simulation and strong observational refinement, this–in the light of our result–also requires a fresh investigation.

References

1. J. Aspnes. Randomized protocols for asynchronous consensus. *Distributed Comput.*, 16(2-3):165–175, 2003. doi:10.1007/s00446-002-0081-5.
2. H. Attiya and C. Enea. Putting strong linearizability in context: Preserving hyperproperties in programs that use concurrent objects. In J. Suomela, editor, *DISC*, volume 146 of *LIPIcs*, pages 2:1–2:17. Schloss Dagstuhl, 2019. doi:10.4230/LIPIcs.DISC.2019.2.
3. A. Bouajjani, M. Emmi, C. Enea, and S. O. Mutluergil. Proving linearizability using forward simulations. In R. Majumdar and V. Kuncak, editors, *CAV*, volume 10427 of *Lecture Notes in Computer Science*, pages 542–563. Springer, 2017. doi:10.1007/978-3-319-63390-9_28.
4. S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984. doi:10.1145/828.833.
5. M. R. Clarkson and F. B. Schneider. Hyperproperties. *J. Comput. Secur.*, 18(6):1157–1210, 2010. doi:10.3233/JCS-2009-0393.
6. R. Colvin and B. Dongol. A general technique for proving lock-freedom. *Sci. Comput. Program.*, 74(3):143–165, 2009. doi:10.1016/j.scico.2008.09.013.
7. J. Derrick and E. A. Boiten. *Refinement in Z and Object-Z - Foundations and Advanced Applications (2. ed.)*. Springer, 2014. doi:10.1007/978-1-4471-5355-9.
8. J. Derrick, S. Doherty, B. Dongol, G. Schellhorn, and H. Wehrheim. Brief Announcement: On Strong Observational Refinement and Forward Simulation. In S. Gilbert, editor, *DISC*, volume 209 of *LIPIcs*, pages 55:1–55:4, Dagstuhl, Germany, 2021. doi:10.4230/LIPIcs.DISC.2021.55.

9 J. Derrick, S. Doherty, B. Dongol, G. Schellhorn, and H. Wehrheim. On strong observational refinement and forward simulation. *CoRR*, 2021. arXiv:2107.14509.

10 J. Derrick, G. Schellhorn, and H. Wehrheim. Proving linearizability via non-atomic refinement. In J. Davies and J. Gibbons, editors, *iFM*, volume 4591 of *Lecture Notes in Computer Science*, pages 195–214. Springer, 2007. doi:10.1007/978-3-540-73210-5_11.

11 B. Dongol and L. Groves. Contextual trace refinement for concurrent objects: Safety and progress. In K. Ogata, M. Lawford, and S. Liu, editors, *ICFEM*, volume 10009 of *Lecture Notes in Computer Science*, pages 261–278, 2016. doi:10.1007/978-3-319-47846-3_17.

12 J. A. Goguen and J. Meseguer. Security policies and security models. In *S&P*, pages 11–20. IEEE Computer Society, 1982. doi:10.1109/SP.1982.10014.

13 W. M. Golab, L. Higham, and P. Woelfel. Linearizable implementations do not suffice for randomized distributed computation. In L. Fortnow and S. P. Vadhan, editors, *STOC*, pages 373–382. ACM, 2011. doi:10.1145/1993636.1993687.

14 A. Gotsman and H. Yang. Liveness-preserving atomicity abstraction. In L. Aceto, M. Henzinger, and J. Sgall, editors, *ICALP*, volume 6756 of *Lecture Notes in Computer Science*, pages 453–465. Springer, 2011. doi:10.1007/978-3-642-22012-8_36.

15 W. O. D. Griffioen and F. W. Vaandrager. Normed simulations. In A. J. Hu and M. Y. Vardi, editors, *CAV*, volume 1427 of *LNCS*, pages 332–344, Vancouver, BC, Canada, 1998.

16 W. O. D. Griffioen and F. W. Vaandrager. A theory of normed simulations. *ACM Trans. Comput. Log.*, 5(4):577–610, 2004. doi:10.1145/1024922.1024923.

17 M. Helmi, L. Higham, and P. Woelfel. Strongly linearizable implementations: possibilities and impossibilities. In D. Kowalski and A. Panconesi, editors, *PODC*, pages 385–394. ACM, 2012. doi:10.1145/2332432.2332508.

18 M. Herlihy and N. Shavit. *The art of multiprocessor programming.* Morgan Kaufmann, 2008.

19 M. Herlihy and J. M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, 1990. doi:10.1145/78969.78972.

20 V. Luchangco, M. Moir, and N. Shavit. Nonblocking k-compare-single-swap. In A. L. Rosenberg and F. M. auf der Heide, editors, *SPAA*, pages 314–323. ACM, 2003. doi:10.1145/777412.777468.

21 N. A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.

22 N. A. Lynch and F. W. Vaandrager. Forward and backward simulations: I. untimed systems. *Inf. Comput.*, 121(2):214–233, 1995. doi:10.1006/inco.1995.1134.

23 J. McLean. A general theory of composition for a class of "possibilistic" properties. *IEEE Trans. Software Eng.*, 22(1):53–67, 1996. doi:10.1109/32.481534.

24 R. Milner. *Communication and concurrency.* PHI Series in computer science. Prentice Hall, 1989.

25 G. Schellhorn. Verification of ASM Refinements Using Generalized Forward Simulation. *Journal of Universal Computer Science (J.UCS)*, 7(11):952–979, 2001.

26 G. Schellhorn. Completeness of Fair ASM Refinement. *Science of Computer Programming, Elsevier*, 76, issue 9:756–773, 2009.

27 G. Schellhorn, J. Derrick, and H. Wehrheim. A sound and complete proof technique for linearizability of concurrent data structures. *ACM Trans. Comput. Log.*, 15(4):31:1–31:37, 2014. doi:10.1145/2629496.

28 G. Schellhorn, H. Wehrheim, and J. Derrick. How to prove algorithms linearisable. In P. Madhusudan and S. A. Seshia, editors, *CAV*, volume 7358 of *Lecture Notes in Computer Science*, pages 243–259. Springer, 2012. doi:10.1007/978-3-642-31424-7_21.

29 P. Stevens. Abstract games for infinite state processes. In D. Sangiorgi and R. de Simone, editors, *CONCUR*, volume 1466 of *Lecture Notes in Computer Science*, pages 147–162. Springer, 1998. doi:10.1007/BFb0055621.

30 R. J. van Glabbeek. The linear time - branching time spectrum II. In E. Best, editor, *CONCUR*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 1993. doi:10.1007/3-540-57208-2_6.

31 R. J. van Glabbeek, B. Luttik, and N. Trcka. Branching bisimilarity with explicit divergence. *Fundam. Informaticae*, 93(4):371–392, 2009. doi:10.3233/FI-2009-109.
32 D. Walker. Bisimulation and divergence. *Information and Computation*, 85:202–241, 1990.
33 S. Zdancewic and A. C. Myers. Observational determinism for concurrent program security. In *CSFW-16*, page 29. IEEE Computer Society, 2003. doi:10.1109/CSFW.2003.1212703.

A Proofs for Section 5

The proof of theorem 5.1 assumes that two LTLs C and A are given, for which a weak progressive simulation $(F, \gg)$ exists. Given an arbitrary program P together with a scheduler S_C for traces over $P \times C$, the proof has to construct a scheduler S_A such that $T(P \times C, S_C)|_{\Gamma_P} = T(P \times A, S_A)|_{\Gamma_P}$. The construction is in two steps: First a function f is constructed that maps traces $\rho_C \in T(P \times C, S_C)$ to traces $f(\rho_C) \in T(P \times A)$. This function has to be carefully defined to then allow the definition of a scheduler S_A that schedules exactly all the steps of $f(\rho_C)$. Weak progressiveness is key to ensure that for an infinite trace ρ_C the trace $f(\rho_C)$ is infinite as well. This then allows to schedule actions for any prefix.

The construction of f shown in Fig. 4 first has to fix a unique sequence of abstract actions in $f(\rho_C)$ that correspond to a single step of ρ_C. To this end, a mapping m is defined. For two states $q_C \in Q_C$ and $q_A \in Q_A$ with $(q_C, q_A) \in F$ and an action $a \in \Sigma_C$, m returns a fixed sequence $\sigma \in \Sigma_A^*$ such that $(q'_C, q'_A) \in F$ holds again for the (unique) states with $q_C \xrightarrow{a}_C q'_C$ and $q_A \xrightarrow{\sigma}_A q'_A$. Mapping m chooses a triangular diagram with $\sigma = \epsilon$ only, when there is no nonempty choice, so $q_C \gg q'_C$ is implied. The existence of σ is guaranteed by the main proof obligation for a weak progressive forward simulation. To be useful for constructing traces over $P \times A$ when a step of a trace over $P \times C$ is given, we extend the definition to allow a program action $a \in \Gamma_P$ as well. In this case m just returns the one element sequence of a. Intuitively, in addition to the commuting diagrams of the forward simulation this defines commuting diagrams that map program steps one-to-one. Formally,

$$m : Q_C \times (\Sigma_{P\times C}) \times Q_A \to (\Sigma_{P\times A})^*$$

is defined to return $m(q_C, a, q_A) := a$ when $a \in \Gamma_P$, and to return the fixed sequence σ as described above when $a \in \Sigma_C$.

It is then possible to define partial functions $f_0, f_1, \ldots$ (viewed as sets of pairs) with $dom(f_n) = \{\sigma_C \in T(P \times C, S_C) : \#\sigma_C \leq n\}$, $cod(f_n) \subseteq T(P \times A)$, such that $f_0 \subseteq f_1 \subseteq \ldots$ inductively as follows:

$$\begin{aligned} f_0 &= \{(\varepsilon, \varepsilon)\} \\ f_{n+1} &= f_n \cup \{(\sigma_C \cdot a, f(\sigma_C) \cdot \alpha) \mid \sigma_C \cdot a \in T(P \times C, S_C), \#\sigma_C = n, \\ &\qquad\qquad \alpha = m(state(\sigma_C).obj, a, state(f(\sigma_C)).obj)\} \end{aligned}$$

The inductive definition maps the new action $a \in S_C(\sigma_C)$ to the corresponding sequence α that is chosen by m. In the definition $(q_P, q_C).obj := q_C$ and the final state of σ_C is $state(\sigma_C) = (q_P, q_C)$. Analogously $(q_P, q_A).obj = q_A$.

The states $(q_P, q_C) = state(\sigma_C)$ and $(q'_P, q_A) = state(f_n(\sigma_C))$ reached at the end of two corresponding traces always satisfy $q_P = q'_P$ and $(q_C, q_A) \in F$. The use of m in the construction guarantees that all the f_n are prefix-monotone: if f_n is defined on σ and $\sigma' \sqsubseteq \sigma$, then $f_n(\sigma') \sqsubseteq f_n(\sigma)$.

Now, define $f := \bigcup_n f_n$. Function f is obviously prefix-monotone as well. Intuitively, it maps all finite traces of $T(P \times C, S_C)$ to a corresponding abstract trace, where m is used in each commuting diagram to choose the abstract action sequence.

If π_C is an infinite trace from $T(P \times C, S_C)$, and $\sigma_A^n := f({\pi_C}^{<n})$, then $\sigma_A^0 \sqsubseteq \sigma_A^1 \sqsubseteq \sigma_A^2 \sqsubseteq \dots$. Therefore a natural choice for extending f to infinite traces is to use the limit of this ascending chain. We define $f^{lim}(\pi_C)$ to be the limit and will use function f^{lim} in several of the lemmas below. There are two cases in this definition. Either the length of σ_A^n always eventually increases. Then the sequences converges to an infinite sequence $f^{lim}(\pi_C) = \pi_A \in T(P \times A)$. Otherwise, the σ_A^n eventually become a constant finite trace σ_A and we can set $f^{lim}(\pi_C) = \sigma_A$. In this case the final state $\mathit{state}(\sigma_A)$ must have a diverging run, since the forward simulation is weak progressive (otherwise the well-founded relation would have to decrease infinitely often, which is impossible). In this case a diverging run from $\mathit{state}(\sigma_A)$ can be fixed with an infinite sequence π_A of internal actions. Adding this infinite sequence to the trace is necessary for defining the abstract scheduler, so different from setting $f^{lim}(\pi_C) := \sigma_A$ we set $f(\pi_C) := \sigma_A \cdot \pi_A$.

We will now define a scheduler S_A, that will schedule exactly those traces in $\sigma_A \in T(P \times A)$ where σ_A is a prefix of some $f(\pi_C)$ such that π_C is an infinite trace in $T(P \times C, S_C)$. Before we can do this properly, a number of lemmas is needed.

► **Lemma A.1.** *$f(\sigma_C)|\Gamma_P = \sigma_C|\Gamma_P$ for all $\sigma_C \in T(P \times C, S_C)$.*

Proof. This should be obvious from the construction, since the forward simulation guarantees that $m(q_C, a, q_A)|\Gamma_P = a|\Gamma_P$ for all $a \in I \cup R$, while $a \in \Gamma_P$ is mapped by identity. ◄

► **Lemma A.2.** *For two finite traces $\sigma_C, \sigma'_C \in T(P \times C, S_C)$: if $f(\sigma_C)$ and $f(\sigma'_C)$ have the same program actions in Γ_P, then σ_C is a prefix of σ'_C or vice versa, and the longer one just adds internal actions of C.*

Proof. Lemma A.1 implies $\sigma_C|\Gamma_P = \sigma'_C|\Gamma_P$. If the lemma would be wrong, then there would be a maximal common prefix σ_0 and two actions $a \neq a'$ such that $\sigma_0 \cdot a \sqsubseteq \sigma_C$ and $\sigma_0 \cdot a' \sqsubseteq \sigma'_C$. The case where both a and a' are external actions is impossible, because otherwise the external actions in σ_C and σ'_C would not be the same. If however one of them is internal, then $S_C(\sigma_0)$ is a one-element set, and both a and a' must be in the set, contradicting $a \neq a'$. ◄

► **Lemma A.3.** *For all finite prefixes σ_A of $f^{lim}(\pi_C)$, there is a unique n, such that $f({\pi_C}^{<n}) \sqsubseteq \sigma_A \sqsubset f({\pi_C}^{<n}) \cdot \alpha)$, where $\alpha := m(\mathit{state}({\pi_C}^{<n}).obj, \pi[n], \mathit{state}(f({\pi_C}^{<n})).obj) \neq \varepsilon$.*

Intuitively, each element of $f^{lim}(\pi_C)$ is added by a uniquely defined commuting diagram.

Proof. First, note that $f({\pi_C}^{<n+1}) = f({\pi_C}^{<n}) \cdot \alpha$. Since the lengths of $f({\pi_C}^{<n})$ are increasing with n to the length of $f^{lim}(\pi_C)$ (and $f({\pi_C}^{<0}) = f(\varepsilon) = \varepsilon$) n is the biggest index where the length of $f({\pi_C}^{<n})$ is still less or equal to $\#\sigma$. ◄

► **Lemma A.4.** *Assume $\pi_C, \pi'_C \in T(P \times C, S_C)$. if σ_A is a prefix of both $f(\pi_C)$ and $f(\pi'_C)$, then there is m such that ${\pi_C}^{<m} = {\pi'_C}^{<m}$ and $\sigma_A \sqsubseteq f({\pi_C}^{<m})$.*

The lemma says, that a common prefix of two traces in the image of f is possible only as the result of a common prefix in the domain of f.

Proof. Since $\sigma_A \sqsubseteq f(\pi_C)$ and each step from $f({\pi_C}^{<n})$ to $f({\pi_C}^{<n+1})$ adds at most one program action, a minimal index n can be found such that σ_A has the same program actions as $f({\pi_C}^{<n})$, while $f({\pi_C}^{<n-1})$ has fewer when $n \neq 0$. Note that when $f^{lim}(\pi_C)$ is finite, n is less than its length, since the diverging run attached at the end has no program actions at all. Similarly, a minimal index n' can be found such that $\sigma_A|\Gamma_P = f({\pi'_C}^{<n'})$. By Lemma A.2 above, it follows that ${\pi_C}^{<n}$ is a prefix of ${\pi'_C}^{<n'}$ or vice versa, with only internal C-actions

added to the longer one. When both are equal, then $n = n'$ and m can be set to be n. However, when the two are not equal, the longer one, say ${\pi'}^{<n'}$ ends with an internal C-action. But then, since this action is mapped to a sequence of internal A-actions $f({\pi'}^{<n'-1})$ also has the same program actions than σ_A, contradicting the minimality of n'. ◀

Equipped with these lemmas, it is now possible to define the scheduler S_A and to prove it is well-defined. We define $S_A(\sigma_A)$ for any finite prefix σ_A of any $f(\pi_C)$, where $\pi_C \in T(P \times C, S_C)$. There are two cases. Either σ_A is not a prefix of $f^{lim}(\pi_C)$. Then it has the form $f^{lim}(\pi_C)\sigma'_A$ where σ'_A is a prefix of the infinite sequence of internal actions that is used in the definition of f in this case that schedules a diverging run. The next action to be sceduled then is the next action of this sequence. Otherwise the definition uses Lemma A.3 to find unique index n, such that $f({\pi_C}^{<n}) \sqsubseteq \sigma_A \sqsubset f({\pi_C}^{<n} \cdot \alpha$. where $\alpha = m(\mathit{state}({\pi_C}^{<n}).\mathit{obj}, \pi[n], \mathit{state}(f({\pi_C}^{<n})).\mathit{obj}) \neq \epsilon$. Since σ_A is a proper prefix, there is an event a, such that $\sigma_A \cdot a \sqsubseteq f^{lim}({\pi_C}^{<n}) \cdot \alpha$, and a is an element of α. If a is an external action in Γ_P, then a must be equal to $\pi_C[n]$ (α contains either $\pi_C[n]$ if it is an external action, or no external action at all). In this case, we set $S_A(\sigma_A) := S_C({\pi_C}^{<n})$. Note that a is enabled and in $S_C({\pi_C}^{<n})$ in this case. Otherwise, when $a \notin \Gamma_P$, we set $S_A(\sigma_A) := \{a\}$.

▶ **Theorem A.5.** *S_A is well-defined.*

Proof. Assume that σ_A is a prefix of two traces $f(\pi_C)$ and $f(\pi'_C)$. We prove that this never leads to two different definitions of $S_A(\sigma_A)$. First, Lemma A.4 gives an index m with ${\pi_C}^{<m} = {\pi'_C}^{<m}$ and $\sigma_A \sqsubseteq f({\pi_C}^{<m})$ If σ_A is a proper prefix of $f({\pi_C}^{<m})$, then the n used in the construction of S_A must satisfy $n + 1 \geq m$, and the prefix $f({\pi_C}^{<n+1}) = f({\pi_C}^{<n}) \cdot \alpha$ on which the definition of S_A is based, is the same for both traces. The remaining case is $m = n+1$ and $\sigma_A = f({\pi_C}^{<n+1})$. In this case the next elements $\pi_C[n+1]$ and $\pi'_C[n+1]$ in the two traces π_C and π'_C could be different. If one of them is internal (i.e. not in Γ_P), then this is not possible, since then $S_C({\pi_C}^{<n+1})$ is a one-element set that contains both of them. However, it is possible that $\pi[n+1]$ and $\pi'[n+1]$ are two different program events $a \neq a'$, both in Γ_P, but in $S_C({\pi_C}^{<m})$. However, in this case $S_A(f({\pi_C}^{<m}))$ is defined in both cases cases to be $S_C({\pi_C}^{<n+1})$. ◀

The following lemma is the inductive step of the theorem below, that shows that S_A allows exactly all $f(\pi_C)$ as scheduled traces.

▶ **Lemma A.6.** *Given $\sigma_A \in T(P \times A, S_A)$, for which a $\pi_C \in T(P \times C, S_C)$ exists with $\sigma_A \sqsubseteq f(\pi_C)$, then $\sigma_A \cdot a \in T(P \times A, S_A)$ (or equivalently $a \in S_A(\sigma_A)$) is equivalent to the existence of some $\pi'_C \in T(P \times C, S_C)$ such that $\sigma_A \cdot a \sqsubseteq f(\pi'_C)$.*

Proof. The case, where $\sigma_A \not\sqsubseteq f^{lim}(\pi_C)$ is simple, since after $f^{lim}(\pi_C)$ a unique diverging trace is attached that is the scheduled one. Otherwise, Lemma A.3 asserts that there is a unique n such that $f^{lim}({\pi_C}^{<n}) \sqsubseteq \sigma_A \sqsubset f^{lim}({\pi_C}^{<n+1})$. Let ${\pi_C}^{<n+1} = ({\pi_C}^{<n}) \cdot a$ and $\alpha = m(\mathit{state}({\pi_C}^{<n}).\mathit{obj}, a, \mathit{state}(f({\pi_C}^{<n})).\mathit{obj})$.

Case 1: $a \notin \Sigma_C$. Then $\alpha = a$, $a_1 = a$ by definition, implying $\sigma_A = f({\pi_C}^{<n})$.

"⇒": If $\sigma_A \cdot a \in T(P \times A, S_A)$, then $a \in S_A(\sigma_A)$ is equivalent to $a_1 \in S_C(\sigma_A)$, since $a_1 = a$ and $S_A(\sigma)$ is defined to be equal to $S_C({\pi_C}^{<n})$. Since actions in $S_C({\pi_C}^{<n})$ are enabled, and every finite trace can be extended to an infinite one, there is an infinite trace π'_1 with $({\pi_C}^{<n}) \cdot a_1 \sqsubseteq \pi'_1$. π'_1 has the required prefix $\pi_C \cdot a_1$ such that $\sigma_A \cdot a_1 = f({\pi_C}^{<n}) \cdot a$

"⇐": if π'_C exists with $\sigma_A \cdot a \sqsubseteq f(\pi'_1)$, then like in the well-definedness proof ${\pi_C}^{<n}$ and ${\pi'_1}^{<n}$ must be the same (both have the same program actions as σ_A). Therefore $a_1 = a$ is scheduled after ${\pi_C}^{<n}$ as required.

Case 2: $a \notin \Sigma_C$. Then α is a nonempty sequence of internal actions and α is the only continuation of $f({\pi_C}^{<n})$ compatible with S_A. σ_A is $f({\pi_C}^{<n})$ concatenated with some proper prefix of α.

"$\Rightarrow$": If $\sigma_A \cdot a \in T(P \times A, S_A)$, then a must be the next element in α. Then, setting $\pi'_1 := \pi_C$ we get the required prefix $f({\pi_C}^{<n+1}) = f({\pi_C}^{<n}) \cdot \alpha$ of which $\sigma_A \cdot a$ is still a prefix.

"$\Leftarrow$": Assume $\sigma_A \cdot a \sqsubseteq f(\pi'_1)$. Then $\sigma_A \cdot a$ is a prefix of both $f(\pi_C)$ and $f(\pi'_1)$, so Lemma A.3 implies that there is some m, such that $\sigma_A \cdot a \sqsubseteq {\pi'_1}^{<m} = {\pi_C}^{<m}$. Obviously, $m \geq n+1$, so the next element after σ_A in π'_1 is the scheduled a too. ◀

With this, we are now ready to prove Theorem 5.2, which implies the main Theorem 5.1.

Proof. The proof is by contradiction. If the theorem does not hold, then there is a trace $\sigma_A \sqsubseteq T(P \times A, S_A)$ of minimal length and some action a, such that $a \in S_A(\sigma_A)$ is not equivalent to the existence of some $\pi'_C \in T(P \times C, S_C)$ such that $\sigma_A \cdot a \sqsubseteq f(\pi'_C)$. However, this equivalence is asserted by Lemma A.6. ◀

B Proofs for Section 6

▶ **Lemma 6.3.** *$(F_1, \emptyset)$ is a weak progressive forward simulation from C to $P \times C \times S_C$.*

Proof. First of all, observe that $(q_C^{ini}, (q_P^{ini}, q_C^{ini}, \varepsilon)) \in F_1$. Now let $(q_C, (q_P, q_C, \sigma)) \in F_1$ and let $q_C \xrightarrow{a}_C q'_C$ be a step of C. There are two cases to consider.

Internal steps: $a \in \Sigma_C \setminus I \cup R$.

The diagram illustrates the simulation.

$$\begin{tikzcd} \boxed{(q_P^{ini}, q_C, \sigma)} \arrow[r, "g(q'_C) \quad a"] \arrow[d, dashed, no head, "F_1"] & \boxed{\left(q_P^{ini}, q'_C, \sigma \cdot g(q'_C) \cdot a\right)} \arrow[d, dashed, no head, "F_1"] \\ \boxed{q_C} \arrow[r, "a"] & \boxed{q'_C} \end{tikzcd}$$

Since $last(\sigma) = q_C$, we get (by definition of S_C) that $g(q'_C) \in S_C(\sigma)$ and $a \in S_C(\sigma \cdot g(q'_C))$. Hence, the following transitions are possible:

$$(q_P^{ini}, q_C, \sigma) \xrightarrow{g(q'_C)} (q_P^{ini}, q_C, \sigma \cdot g(q'_C)) \xrightarrow{a} (q_P^{ini}, q'_C, \sigma \cdot g(q'_C) \cdot a)$$

We furthermore get $(q_C, (q_P^{ini}, q'_C, \sigma \cdot g(q'_C) \cdot a)) \in F_1$.

Invokes and returns: $a \in I \cup R$.

The diagram illustrates the simulation.

$$\begin{tikzcd} \boxed{(q_P^{ini}, q_C, \sigma)} \arrow[r, "g(a, q'_C) \quad a \quad rec(a)"] \arrow[d, dashed, no head, "F_1"] & \boxed{\left(q_P^{ini}, q'_C, \sigma \cdot g(a, q'_C) \cdot a \cdot rec(a)\right)} \arrow[d, dashed, no head, "F_1"] \\ \boxed{q_C} \arrow[r, "a"] & \boxed{q'_C} \end{tikzcd}$$

Since $last(\sigma) = q_C$, we get $g(a, q'_C) \in S_C(\sigma)$. By definition of S_C we hence get

$$(q_P^{ini}, q_C, \sigma) \xrightarrow{g(a, q'_C)} (q_a, q_C, \sigma \cdot g(q'_C)) \xrightarrow{a} (q_{rec(a)}, q'_C, \sigma \cdot g(q'_C) \cdot a) \xrightarrow{rec(a)} (q_P^{ini}, q'_C, \sigma \cdot g(q'_C) \cdot a \cdot rec(a))$$

We furthermore get $(q_C, (q_P^{ini}, q'_C, \sigma \cdot g(q'_C) \cdot a \cdot rec(a))) \in F_1$.

This is furthermore a weak progressive forward simulation as the matching steps are never empty (no triangular diagrams), so that we can take $\gg_1$ to be empty. ◀

▶ **Definition B.1.** *Given an LTS L, and a subset $\Delta \subseteq \Sigma$ a state q is*

- reachable *with $h \in \Delta^*$, written $reach_L(h, q)$ if $q = last(\xi)$ for an execution ξ that has history h.*
- minimally reachable*, written if additionally, the execution ξ is shortest: its trace σ is either empty when $h = \epsilon$ (then the state is initial), or the last action of σ is the last of h.*

*We then say that an LTS is Δ-*deterministic *if the set of minimally reachable states for every history h consists of a single element which we write $minstate_L(h)$.*

▶ **Lemma B.2.** *The two LTSs $P \times C \times S_C$ and $P \times A \times S_A$ are Σ_P-deterministic.*

Proof. We first prove that $P \times C \times S_C$ and $P \times A \times S_A$ are Γ_P-deterministic. To do so we show that given two finite traces σ_1 and σ_2 with $h = \sigma_1|\Gamma_P = \sigma_2|\Gamma_P$ the corresponding executions must be prefixes of each other by induction over the length of the shorter one. The initial states are the same since the initial state of deterministic systems is unique. Given that the executions agree up to step n, the next action is the same: either it is the single scheduled internal one, or the next action is the common one of the history. Since the steps are deterministic the next state is equal as well. It follows that there is only one minimally reachable state for every history, since in particular the final states of two minimal executions must agree, as their traces must be the same. For our specific program a minimal execution with history h also executes the same actions from $I \cup R$, since each such action e is followed by the corresponding record-action $rec(e)$, otherwise it would not be minimal. Therefore the two LTS are Σ_P-deterministic too. ◀

▶ **Lemma 6.4.** *If C Δ-refines A and A is Δ-deterministic, then there exists a Δ-forward simulation.*

Proof. Define the forward simulation F as

$$F = \{(q_C, q_A) \mid \exists\, h \in \Delta^*.\ reach_C(h, q_C) \wedge q_A = minstate_A(h)\}$$

where $reach_C(h, q_C)$ and $minstate_A(h)$ are from Def. B.1 . The proof obligations of a forward simulation are satisfied: The initial concrete state is related to the initial abstract one by choosing $h = \epsilon$. Given $(q_C, q_A) \in F$, and $q_C \xrightarrow{a} q'_C$ there are two cases: If a is internal, then doing a stuttering step by choosing $q'_A = q_A$ is sufficient to show $(q'_C, q_A) \in F$. If $a \in \Delta$, then $(q_C, q_A) \in F$ implies that there is a history h and a minimal execution of A with this history and final state $q_A = minstate_A(h)$. Also ha is a history of C that has final state q'_C, so by refinement $h \cdot a$ is a history of A too. Therefore, there exists a minimal execution with this history $h \cdot a$ of A ending in some state q'_A. This state fits our correctness proof obligation: since the execution has a minimal prefix with history h (the prefix removes a and all internal actions after the last one of h), uniqueness of the minimal reachable state implies that it must pass through q_A. The trace σ of the remaining steps then has $a|_\Delta = \sigma|_\Delta$ and $q_A \xrightarrow{\sigma}_A q'_A$ as required, and $(q'_C, q'_A) \in F$ holds by definition. ◀

▶ **Lemma 6.5.** *There exists a weak progressive Σ_P-forward simulation $(F_2, \gg_2)$ between $P \times C \times S_C$ and $P \times A \times S_A$.*

Proof. Since strong observational refinement implies Γ_P-refinement, and the abstract system is Σ_P-deterministic (Lemma B.2) Lemma 6.4 ensures that a Σ_P-forward simulation F exists. This forward simulation is weak progressive, since the concrete system never executes more than one internal action in a row ($S_C(\sigma' \cdot a)$ is external, when a is internal). Thus the well-founded order can be chosen to have $(q'_P, q'_C, \sigma')) \gg (q_P, q_C, \sigma)$ iff $q_P = q'_P = q_P^{ini}$ and there is an internal action a with $q_C \xrightarrow{a} q'_C$ and $\sigma' = \sigma \cdot a$. ◀

▶ **Lemma 6.6.** *$(F_3, \gg_3)$ is a weak progressive forward $(I \cup R)$-simulation between $P \times A \times S_A$ and A.*

Proof. First we prove that F_3 is a forward simulation.

Stuttering steps The stuttering steps are $a \in \{g(q), g(e, q), \mathit{div}, \mathit{rec}(e)\}$. The proofs for each of these are trivial, but for completeness, we consider each of these in turn. We have the following transitions:

$\boldsymbol{a = g(q)}$. We have $(q_P^{ini}, q_A, \sigma) \xrightarrow{g(q)} (q_P^{ini}, q_A, \sigma \cdot g(q))$.
$\boldsymbol{a = g(e, q)}$. We have $(q_P^{ini}, q_A, \sigma) \xrightarrow{g(e,q)} (q_e, q_A, \sigma \cdot g(e, q))$.
$\boldsymbol{a = \mathit{div}}$. We have $(q_P^{ini}, q_A, \sigma) \xrightarrow{div} (q_{div}, q_A, \sigma)$.
$\boldsymbol{a = \mathit{rec}(e)}$. We have $(q_{rec(e)}, q_A, \sigma) \xrightarrow{rec(e)} (q_P^{ini}, q_A, \sigma \cdot \mathit{rec}(e))$.

In each of these, if F_3 holds in the pre-state, it holds again in the post state since q_A is unchanged, and the abstract system, i.e., A does not take a step.

Non-stuttering steps All internal and external steps of A are non-stuttering. Since F_3 ensures that the states of A at the concrete and abstract states coincide, these can be trivially discharged too. In particular, the possible transitions are:

$\boldsymbol{a = e \in (I \cup R)}$. We have $(q_e, q_A, \sigma) \xrightarrow{e} (q_{rec(e)}, q'_A, \sigma \cdot e)$.
$\boldsymbol{a \in \Sigma_A \setminus (I \cup R)}$. We have
$(q, q_A, \sigma) \xrightarrow{a} (q, q'_A, \sigma)$.

In both cases, in A, we can take the corresponding transition $q_A \xrightarrow{a} q'_A$, preserving F_3.

We now prove weak progressiveness of F_3. Note that for each stuttering transition, except $g(q)$, the program state changes. We define the well-founded order to be the relation $\gg_3$ such that:

1. $(q_P^{ini}, \_, \_) \gg_3 (q_e, \_, \_)$
2. $(q_{rec(e)}, \_, \_) \gg_3 (q_P^{ini}, \_, \_)$
3. $(q_P^{ini}, \_, \_) \gg_3 (q_{div}, \_, \_)$

Note that from $(q_{div}, \_, \_)$, the only possible transition is an $\xrightarrow{a}$ step, where $a \in \Sigma_A \setminus (I \cup R)$, which is non-stuttering. Similarly, from $(q_e, \_, \_)$, the only possible transition is $\xrightarrow{e}$, where $e \in I \cup R$. Thus, when we reach a state that is minimal wrt $\gg_3$, no more stuttering is possible. Any transition from state $(q_{rec(e)}, \_, \_)$ is guaranteed to reduce w.r.t. $\gg_3$, as are transitions corresponding to $g(e, q)$ and div from $(q_P^{ini}, \_, \_)$.

This leaves us with transitions corresponding to $g(q)$ from (q_P^{ini}, q_A, σ), which may stutter infinitely often. We we can show that such stuttering only exists if A contains a diverging run from q_A, i.e., div is enabled in $(q_P^{ini}, q_A, \sigma) \in Q_{P \times A \times S_A}$.

Suppose there exists an infinite run

$$(q_P^{ini}, q_A, \sigma) \xrightarrow{g(q_1)} (q_P^{ini}, q_A, \sigma \cdot g(q_1)) \xrightarrow{g(q_2)} (q_P^{ini}, q_A, \sigma \cdot g(q_1) \cdot g(q_2)) \xrightarrow{g(q_3)} \dots .$$

By construction, $P \times A \times S_A$ is an "abstraction" of $P \times C \times S_C$ such that $T(P \times A, S_A)|_\Gamma = T(P \times C, S_C)|_\Gamma$, thus, $(\sigma|_\Gamma) \cdot g(q_1) \cdot g(q_2) \cdot g(q_3) \cdot \dots \in T(P \times C, S_C)|_\Gamma$. Thus, there exists a q_C such that $\mathit{last}(\sigma) = g(q_C)$ and

$$(q_P^{ini}, q_C, \sigma) \xrightarrow{g(q_1)\cdot\tau_1} (q_P^{ini}, q_1, \sigma \cdot g(q_1) \cdot \tau_1) \xrightarrow{g(q_2)\cdot\tau_2} (q_P^{ini}, q_2, \sigma \cdot g(q_1) \cdot \tau_1 \cdot g(q_2) \cdot \tau_2) \xrightarrow{g(q_3)\cdot\tau_3} \dots$$

where $\tau_k \in \Sigma_C \setminus (I \cup R)$ for all k. Note that the definition of S_C enforces a $\xrightarrow{g(q_k)\cdot\tau_k}$ transition for each $\xrightarrow{g(q_k)}$ transition in $P \times A \times S_A$. This execution, when restricted to the actions of C corresponds to a diverging run of C:

$$q_C \xrightarrow{\tau_1} q_1 \xrightarrow{\tau_2} q_2 \xrightarrow{\tau_3} \ldots$$

Since this is an infinite run of internal actions, by definition, the action *div* must be offered by $P \times C \times S_C$, and enabled in (q_P^{ini}, q_C, σ). Moreover, since $T(P \times A, S_A)|_\Gamma = T(P \times C, S_C)|_\Gamma$, *div* must also be possible in $P \times A \times S_A$. In particular, *div* must be enabled in (q_P^{ini}, q_A, σ). Now, $P \times A \times S_A$ contains a run with final state σ. Therefore, $P \times A \times S_A$ also contains a run

$$(q_P^{ini}, q_A, \sigma) \xrightarrow{div} (q_{div}, q_A, \sigma) \xrightarrow{\tau_1'} (q_{div}, q_1', \sigma) \xrightarrow{\tau_2'} (q_{div}, q_2', \sigma) \xrightarrow{\tau_3'} \ldots$$

where $\tau_k' \in \Sigma_A \setminus (I \cup R)$ for all k since $P \times A \times S_A$ can no longer schedule any further external actions after executing *div*, i.e., must schedule an internal action. Thus, we must have a diverging run in A as well. ◀

▶ **Theorem 6.7.** *Let $(F_1, \gg_1)$ be a weak progressive Δ-forward simulation from C to B, and $(F_2, \gg_2)$ one from B to A. Then there exists a weak progressive Δ-forward simulation $(F, \gg)$ from C to A.*

Proof. We show that $(F, \gg)$ is a weak progressive forward from C to A. Standard refinement results (e.g. Proposition 4.9 in [22]) imply, that refinement by forward simulation is transitive, i.e. $F_1 \circ F_2$ is a Δ-forward simulation. The definition of $\gg$ also clearly implies that the order $\gg$ decreases on a triangular diagram, when its abstract state has no diverging run. It remains to be shown that $\gg$ is well-founded. An infinite descending chain $q_C^0 \gg q_C^1 \gg \ldots$ leads to a contradiction as follows: by definition of $\gg$ the states are the ones of a diverging run of C that starts with q_C^0. Since F_1 is a forward simulation, there is a corresponding run of B with states $q_B^0, q_B^1, \ldots$. For all k both $(q_C^k, q_B^k) \in F_1$ and $q_B^k \xrightarrow{\alpha^k} q_B^{k+1}$ hold, where each α^k is a sequence of internal actions. Some α^k may be empty, so states may occur several times. It is, however, not possible that there is a final state q_B^k such that $q_B^k = q_B^{k+1} = \ldots$, since then $q_C^k \gg_1 q_C^{k+1} \gg_1 \ldots$ would be implied, contradicting well-foundedness of $\gg_1$. Note that q_B^k cannot start a diverging run in this case, otherwise $q_C^k \gg q_C^{k+1}$ would not hold by definition. Therefore, the states $q_B^0, q_B^1, \ldots$ are some states of an infinite diverging run too.

By applying the same argument for the upper simulation a sequence $q_A^0, q_A^1, \ldots$ of states of A can be found, such that for all k $(q_B^k, q_A^k) \in F_2$ and $q_A^k \xrightarrow{\beta^k} q_A^{k+1}$ holds. Again the β^k are sequences of internal actions, and the existence of a final state with $q_A^k = q_A^{k+1} = \ldots$ would contradict well-foundedness of $\gg_2$. Therefore the construction results in a diverging run from q_A^0, contradicting the definition of $q_C^0 \gg q_C^1$ which required that no corresponding abstract state q_A^0 with a diverging run exists. ◀