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
Weak Progressive Forward Simulation Is Necessary and Sufficient

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 our 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, \texttt{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 \texttt{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 \texttt{LL(ptr)} operation loads the value at the location pointed to by the pointer \(ptr\). The \texttt{SC(ptr,v)} conditionally stores the value \(v\) at the location pointed to by \(ptr\) if the location has not been modified by another \texttt{SC} since the executing thread’s most recent \texttt{LL(ptr)} operation. If the update actually occurs, \texttt{SC} returns \texttt{true}, otherwise the location is not modified and \texttt{SC} returns \texttt{false}. In the first case, we say that the \texttt{SC} succeeds. Otherwise, we say that it fails.

Critically, we stipulate that the \texttt{LL} and \texttt{SC} operations are implemented using the algorithm of [20]. This algorithm has the following property. If thread \(t_1\) executes an \texttt{LL} operation, and then thread \(t_2\) executes an \texttt{LL} operation before \(t_1\) has executed its subsequent \texttt{SC} operation, then that \texttt{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 \texttt{fetch\_and\_inc} implementation, using the specified \texttt{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 \texttt{LL} at line F2 of \(t_1\) and that of \(t_2\) (with some executions of F3 in between), such that neither \texttt{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 \(\xi\) and \(\xi'\) be sequences. The empty sequence is denoted \(\varepsilon\) and the length of \(\xi\) denoted \(#\xi\). We write \(\xi \subseteq \xi'\) (similarly, \(\xi \subset \xi'\)) iff \(\xi\) is a prefix (similarly, proper prefix) of \(\xi'\). Assuming \(m < n \leq \#\xi\), we write \(\xi^{<n}\) for the prefix of \(\xi\) of length \(n\) and \(\xi[m]\) for the element of \(\xi\) 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 \(\xi\)

\(^3\) There are several ways to represent a multithreaded program or object as an LTS, e.g., [21, 28].
31:4 Weak Progressive Forward Simulation Is Necessary and Sufficient

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 $\Sigma$ 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 $q$ if 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 $q_0 \cdot a_1 \cdot q_1 \cdot a_2 \cdot \ldots \cdot a_n \cdot q_n$. 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 $\Sigma^*$, and infinite traces, denoted $\Sigma^\omega$. We use $\sigma \in \Sigma^*$ and $\pi \in \Sigma^\omega$ when referring 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 $q^{ini} \xrightarrow{\varepsilon} q'$. In this case, we let $\text{state}(\sigma)$ denote $q'$.

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.

\[\text{Note that a step-deterministic LTS differs from the notion of a deterministic automaton of Lynch and Vaandrager [22]. Attiya and Enea simply refer to step-deterministic LTSs as deterministic LTSs [2].}\]
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: $\Sigma_P = I \cup R \cup \Gamma_P$, where $\cup$ is a disjoint union and $\Gamma_P$ is the set of program actions. The composition of a program with another 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
\[
\begin{align*}
Q &= Q_P \times Q_O, \\
Q^{ini} &= Q_P^{ini} \times Q_O^{ini}, \\
\Sigma &= \Sigma_P \cup \Sigma_O, \\
\delta &= \bigcup_{a \in \Sigma_P \cap \Sigma_O} \{(q_P, q_O) \xrightarrow{\delta} (q'_P, q'_O) \mid q_P \xrightarrow{\delta_P} q'_P \land q_O \xrightarrow{\delta_O} q'_O\} \cup \\
&\quad \bigcup_{a \in \Sigma_P \setminus \Sigma_O} \{(q_P, q_O) \xrightarrow{\delta} (q'_P, q_O) \mid q_P \xrightarrow{\delta_P} q'_P\} \cup \\
&\quad \bigcup_{a \in \Sigma_O \setminus \Sigma_P} \{(q_P, q_O) \xrightarrow{\delta} (q_P, q'_O) \mid q_O \xrightarrow{\delta_O} q'_O\}.
\end{align*}
\]

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 $\rho$ 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 $\Delta$-refinement of $A$ iff $T(C)|\Delta \subseteq T(A)|\Delta$. One can establish $\Delta$-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 $\Sigma_C$ and $\Sigma_A$, respectively, and let $\Delta \subseteq \Sigma_C \cap \Sigma_A$. A relation $F \subseteq Q_C \times Q_A$ is a $\Delta$-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 $q_C \xrightarrow{\rho_C} q'_C$, then there exist $\sigma \in \Sigma_A' \text{ and } q'_A \in Q_A$ such that $a|\Delta = \sigma|\Delta, q_A \xrightarrow{\sigma} q'_A \text{ and } (q'_C, q'_A) \in F$.

Note that $\sigma$ in the above definition may be $\varepsilon$ 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 $\tau_3$ forms such as diagram.

**Lemma 3.3 (Lynch [21]).** If there is a $\Delta$-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 $\rho$ 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 $\sigma$ of $L$ consistent with $S$, the scheduler satisfies $S$.

\(^5\) It is well known that forward simulation is sound for proving refinement, but completeness requires both forward and backward simulation [7, 22].
Weak Progressive Forward Simulation Is Necessary and Sufficient

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

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 $\Sigma^*$; initial state $\varepsilon$; and transitions $\sigma \trianglelefteq \sigma \cdot a$, 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 $C \trianglelefteq_s A$, 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 $T(P \times C, S_C)|_{r_P} \equiv T(P \times A, S_A)|_{r_P}$.

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 recapture 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 $\Delta$-forward simulation from $C$ to $A$ iff

1. **Initialisation.** $(q_C^n, q_A^m) \in F$,
2. **Step.** For all $(q_C, q_A) \in F$, if $q_C \trianglelefteq_C q_C'$ then there exist $\sigma \in \Sigma_A$ and $q_A' \in Q_A$ such that $a|_{\Delta} = \sigma|_{\Delta}$, $q_A \trianglelefteq_A q_A'$ and $(q_C', q_A') \in F$, and
3. **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 $C \trianglelefteq_s A$. 

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 $\dot{r}_k \in \Sigma_A \setminus (I \cup R)$, 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 $\dot{r}_1$.

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 $q_C \cdot i \cdot q_C \cdot \tau_1 \cdots \tau_n \cdot q_C \cdot r$ or $q_C \cdot i \cdot q_C \cdot \tau_1 \cdots \tau_n \cdot q_C \cdot r'$ 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 $q_A \cdot i \cdot q_A^0 \cdot \dot{r}_1 \cdot q_A^1 \cdot \dot{r}_2 \cdots$ can be generated in $A$.

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

Now we show that there does not exist a progressive forward simulation. First, there exists a forward simulation, $F$, that allows each $\tau_k$ to behave as the corresponding $\dot{r}_k$ and as a stuttering step, i.e., $(q_C^j, q_A^j) \in F$ 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 $\Delta$-divergent (written $q \overset{\Delta}{\rightarrow}$) if there exists an infinite run $q \cdot a_1 \cdot a_2 \cdots$ 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 $\Delta$-forward simulation from $C$ to $A$ iff

- Initialisation. $(q^m_C, q^m_A) \in F$.

Step. For all $(q_C, q_A) \in F$, if $q_C \xrightarrow{\alpha} q'_C$, then there exists $\sigma \in \Sigma_A$ 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 $q_A \gg A q'_A$.

Note that we do not require that any triangular diagram with $q_C \xrightarrow{\alpha} q'_C$, $(q_C, q_A) \in F$, 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 $s \downarrow s'$ for some internal action $\tau$ 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 LTSs $C$ 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 $T(P \times A, S_A) \mid_{C} T(P \times C, S_C) \mid_{P}$. 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 $\Sigma_P$ 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 $F_P$ 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.
Theorem 5.2. Branching points are at program actions only. We have the following theorem.

\[
\text{If } C \subseteq A, \text{ then there exists a weak progressive } (I \cup R)\text{-forward simulation from } C \text{ to } A.
\]

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 \( C = (Q_C, Q_C^{\text{ini}}, \Sigma_C, \delta_C) \) that we use is given by the LTS \( P = (Q_P, Q_P^{\text{ini}},\Sigma_P, \delta_P) \), where

\[
\begin{align*}
Q_P &= \{ q_P^{\text{ini}}, q_\text{div} \} \cup \{ q_e, q_{\text{rec}(e)} \mid e \in I \cup R \}, \\
Q_P^{\text{ini}} &= \{ q_P^{\text{ini}} \}, \\
\Gamma_P &= \{ \text{div} \} \cup \{ g(q), g(e,q), \text{rec}(e) \mid q \in Q_C \land e \in I \cup R \},
\end{align*}
\]
Weak Progressive Forward Simulation Is Necessary and Sufficient

Additionally, we use an external action whether the program has executed must be scheduled if the last action executed in Figure 6, these are the record and guess actions as well as scheduler. Therefore, the scheduler chooses one such that the resulting set is a singleton. Clearly the third case results in the given trace. Furthermore, the scheduler decides on the next action of client program, concrete object and scheduler visible in the traces of and . We now define a particular admissible scheduler. The concrete scheduler is calculated wrt the LTS when it is in state , i.e., define the next action for a given trace . The program can only synchronise with div when it is in state . Once executed, the program transitions to state and from this point, it is only possible to schedule internal actions of in .

We now define a particular admissible scheduler. The concrete scheduler must schedule the actions of in , i.e., define the next action for a given trace .

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

Note that state(\sigma|_{\Sigma_C}) is calculated wrt the LTS as opposed to the composition . Furthermore, the scheduler decides on the next action of based on the last action in the given trace . The first two cases determine the next action of depending on whether the program has executed . In the first two cases, there may be a choice of , the scheduler chooses one such that the resulting set is a singleton. Clearly the third case results in a singleton set since the action fixes the only external action allowed by the scheduler. Therefore, the scheduler 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 . In the fourth case, action must be scheduled if the last action executed in is . In the final case, the scheduler may choose to diverge (if the program diverges), perform a guess action, g(q_C) (corresponding

![Figure 5](image5.png)  
![Figure 6](image6.png)
to an internal transition of \( C \) or a guess action, \( g(\epsilon, q_C) \) (corresponding to an external transition of \( C \)). Note that for both \( g(q_C) \) and \( g(\epsilon, q_C) \), state \( q_C \) is the post state of the given action after executing from \( \text{state}(\sigma) \).

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 \( \text{div} \in \sigma \) holds then \( \text{state}(\sigma|_{S_C}) \not\sim_{\{I \cup R\}} \) and \( (\exists \ q'_C, \ \text{state}(\sigma|_{S_C}) \xrightarrow{S_C(\sigma)} q'_C \land q'_C \not\sim_{\{I \cup R\}}) \).

**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

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

- **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 \( T(P \times C, S_C)|_{I_P} = T(P \times A, S_A)|_{I_P} \) that exists due to the assumption \( C \leq_s A \). The proof follows from a general completeness result for refinements for \( \Delta \)-deterministic systems.

- **Lemma 6.4.** If \( C \Delta \text{-refines } A \) and \( A \) is \( \Delta \)-deterministic, then there exists a \( \Delta \)-forward simulation.

A \( \Delta \)-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 \( \Delta \)-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 \( \Delta \)-deterministic is weaker than the ones given in [2] and [22]). Clearly, \( P \times A \times S_A \) is \( I_P \)-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 \( \epsilon \in I \cup R \), since each of these is followed by the corresponding \( \text{rec}(\epsilon) \) action, that is already preserved.

- **Lemma 6.5.** There exists a weak progressive \( \Sigma_P \)-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. $(q_{\text{div}}^i, \_ \_ \_ \_)$ $\gg_3$ $(q_c, \_ \_ \_ \_)$
2. $(q_{\text{rec}(e)}^i, \_ \_ \_ \_)$ $\gg_3$ $(q_{P}^i, \_ \_ \_ \_)$
3. $(q_{P}^i, \_ \_ \_ \_)$ $\gg_3$ $(q_{\text{div}}^i, \_ \_ \_ \_)$

**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 $\text{div}$ action in our model.

Note that from $(q_{\text{div}}^i, \_ \_ \_ \_)$, the only possible transition is an $\xrightarrow{\_ \_ \_ \_}$ step, where $a \in S_A \setminus (I \cup R)$, which is non-stuttering. Similarly, from $(q_c, \_ \_ \_ \_)$, the only possible transition is $\xrightarrow{\_ \_ \_ \_}$, 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 $(q_{\text{rec}(e)}^i, \_ \_ \_ \_)$ is guaranteed to reduce w.r.t. $\gg_3$, as are transitions corresponding to $g(e, q)$ and $\text{div}$ from $(q_{P}^i, \_ \_ \_ \_)$.

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

Suppose there exists an infinite run

$$(q_{P}^i, q_A, \sigma) \xrightarrow{g(q_1)} (q_{P}^i, q_A, \sigma \cdot g(q_1)) \xrightarrow{g(q_2)} (q_{P}^i, q_A, \sigma \cdot g(q_1) \cdot g(q_2)) \xrightarrow{g(q_3)} \ldots$$

By construction, $P \times A \times S_A$ is an "abstraction" of $P \times C \times S_C$ such that $T(P \times A, S_A)|_R = T(P \times C, S_C)|_R$. Thus, there exists a $q_C$ such that $\text{last}(\sigma) = g(q_C)$ and

$$(q_{P}^i, q_C, \sigma) \xrightarrow{g(q_1)} (q_{P}^i, q_1, \sigma \cdot g(q_1)) \xrightarrow{g(q_2)} (q_{P}^i, q_2, \sigma \cdot g(q_1) \cdot g(q_2)) \xrightarrow{g(q_3)} \ldots$$

where $q_k \in S_C \setminus (I \cup R)$ for all $k$. Note that the definition of $S_C$ enforces a $g(q_k)$ transition for each $g(q_k)$ 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 $\text{div}$ must be offered by $P \times C \times S_C$, and enabled in $(q_{P}^i, q_C, \sigma)$. Moreover, since $T(P \times A, S_A)|_R = T(P \times C, S_C)|_R$, $\text{div}$ must also be possible in $P \times A \times S_A$. In particular, $\text{div}$ must be enabled in $(q_{P}^i, q_A, \sigma)$.

Now, $P \times A \times S_A$ contains a run with final state $\sigma$. Therefore, $P \times A \times S_A$ also contains a run

$$(q_{P}^i, q_A, \sigma) \xrightarrow{\text{div}} (q_{\text{div}}, q_A, \sigma) \xrightarrow{\tau'_1} (q_{\text{div}}, q'_1, \sigma) \xrightarrow{\tau'_2} \ldots$$

where $\tau'_k \in S_A \setminus (I \cup R)$ for all $k$ since $P \times A \times S_A$ can no longer schedule any further external actions after executing $\text{div}$, 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 $\Delta$-forward simulation from $C$ to $B$, and $(F_2, \gg_2)$ one from $B$ to $A$. Then there exists a weak progressive $\Delta$-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 \( q_C \xrightarrow{\alpha} q_C' \) for an internal action \( \alpha \notin \Delta \) such that one of the following two conditions holds:

1. \( \exists q_B. (q_C, q_B) \in F_1 \land (q_C', q_B) \in F_1 \land q_B \gg_1 q_C' \land \neg q_B \gg_\Delta \),

2. \( \exists q_B, \alpha, q_A. (q_C, q_B) \in F_1 \land (q_C', q_B') \in F_1 \land q_B \xrightarrow{\alpha} q_B' \land q_B \gg_2 q_B' \land (q_B, q_A) \in F_2 \land (q_B', q_A) \in F_2 \land \neg q_A \gg_\Delta \)

where \( \alpha \) 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:

\[
\begin{align*}
\exists q_B. \exists q_A. & \quad (q_C, q_B) \in F_1 \land (q_C', q_B') \in F_1 \land q_B\ll B b' \land (b' = \true \land (b \land pc \gg_X pc')) \land (\neg b \land pc \gg_X pc') \\
\end{align*}
\]

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 \( \{F_1, F_2, F_3, F_4, \text{idle}\} \).

\[
q \gg q' \iff \exists t. (q(\text{current\_val}) = q(n_t), q(pc_t)) \gg_L (q'(\text{current\_val}) = q'(n_t), q'(pc_t))
\]

All that remains is the instantiation of \( \gg_X \) and \( \gg_Y \). The order \( \gg_X \) is empty, since \( q(\text{current\_val}) = q(n_t) \) implies that \( q(pc_t) = F_3 \), and execution of thread \( t \) corresponds to a successful SC operation. We define \( F_2 \gg_X F_3 \), representing a retry since this order allows \( t \) to make progress towards making \( q(\text{current\_val}) = q(n_t) \) 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 \( \text{backoff}_t \) for a thread \( t \) that causes the FAI executed by thread \( t \) to be delayed. Abstractly, \( \text{backoff}_t \) 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 \( \text{backoff}_t \) 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 (\( \tau \)) 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

Weak Progressive Forward Simulation Is Necessary and Sufficient


The inductive definition maps the new action each commuting diagram to choose the abstract action sequence. It maps all finite traces of construction guarantees that all the two corresponding traces always satisfy the state that is chosen by $\alpha$ described above when is defined to return commuting diagrams that map program steps one-to-one. Formally, intuitively in addition to the commuting diagrams of the forward simulation this defines obligation for a weak progressive forward simulation. To be useful for constructing traces nonempty choice, so $q$ 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 $\rho_C$ the trace $f(\rho_C)$ is infinite as well. This then allows to schedule actions for any prefix.

Now, define $f$ 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 $\rho_C$. 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 $q_C \xrightarrow{a} q_C'$ and $q_A \xrightarrow{a} q_A'$. 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 $\sigma$ 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 $\sigma$ 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 $\text{dom}(f_n) = \{ (q_C \in T(P \times C), S_C) : \# \sigma_C \leq n \}$, $\text{cod}(f_n) \subseteq T(P \times A)$, such that $f_0 \subseteq f_1 \subseteq \ldots$ inductively as follows:

$$f_0 = \{ (\epsilon, \epsilon) \}$$

$$f_{n+1} = f_n \cup \{ (\sigma_C \cdot a, f(\sigma_C) \cdot a) \mid \sigma_C \cdot a \in T(P \times C, S_C), \# \sigma_C = n, \alpha = m(\text{state}(\sigma_C).\text{obj}, a, \text{state}(f(\sigma_C)).\text{obj}) \}$$

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

The states $(q_P, q_C) = \text{state}(\sigma_C)$ and $(q_P, q_A) = \text{state}(f_n(\sigma_C))$ 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 $\sigma$ and $\sigma' \subseteq \sigma$, then $f_n(\sigma') \subseteq f_n(\sigma)$.

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 \( \pi_C \) is an infinite trace from \( T(P \times C, S_C) \), and \( \sigma^n_A := f(\pi_C^{<n}) \), then \( \sigma^0_A \sqsubseteq \sigma^1_A \sqsubseteq \ldots \). 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 \( \sigma^n_A \) always eventually increases. Then the sequences converges to an infinite sequence \( f^{lim}(\pi_C) = \pi_A \in T(P \times A) \). Otherwise, the \( \sigma^n_A \) eventually become a constant finite trace \( \sigma_A \) and we can set \( f^{lim}(\pi_C) = \sigma_A \). In this case the final state \( state(\sigma_A) \) 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 \( state(\sigma_A) \) can be fixed with an infinite sequence \( \pi_A \) 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 \( \sigma_A \) is a prefix of some \( f(\pi_C) \) such that \( \pi_C \) is an infinite trace in \( T(P \times C, S_C) \). Before we can do this properly, a number of lemmas is needed.

\[ \textbf{Lemma A.1.} \quad f(\sigma_C)|_{I_0} = \sigma_C|_{I_0} \text{ for all } \sigma_C \in T(P \times C, S_C). \]

\textbf{Proof.} This should be obvious from the construction, since the forward simulation guarantees that \( m(q_C, a, q_A)|_{I_0} = a|_{I_0} \) for all \( a \in I \cup R \), while \( a \in I_0 \) is mapped by identity. \( \blacksquare \)

\[ \textbf{Lemma A.2.} \quad \text{For two finite traces } \sigma_C, \sigma_C' \in T(P \times C, S_C): \text{ if } f(\sigma_C) \text{ and } f(\sigma_C') \text{ have the same program actions in } I_0, \text{ then } \sigma_C \text{ is a prefix of } \sigma_C' \text{ or vice versa, and the longer one just adds internal actions of } C. \]

\textbf{Proof.} Lemma A.1 implies \( \sigma_C|_{I_0} = \sigma_C'|_{I_0} \). If the lemma would be wrong, then there would be a maximal common prefix \( \sigma_0 \) and two actions \( a \neq a' \) such that \( \sigma_0 \cdot a \subseteq \sigma_C \) and \( \sigma_0 \cdot a' \subseteq \sigma_C' \). The case where both \( a \) and \( a' \) are external actions is impossible, because otherwise the external actions in \( \sigma_C \) and \( \sigma_C' \) 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 \( a \neq a' \). \( \blacksquare \)

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

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

\textbf{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 \( f(\pi_C^{<n}) = f(\varepsilon) = \varepsilon \) \( n \) is the biggest index where the length of \( f(\pi_C^{<n}) \) is still less or equal to \( \# \sigma \).

\[ \textbf{Lemma A.4.} \quad \text{Assume } \pi_C, \pi_C' \in T(P \times C, S_C). \text{ if } \sigma_A \text{ is a prefix of both } f(\pi_C) \text{ and } f(\pi_C'), \text{ then there is } m \text{ such that } \pi_C^{<m} = \pi_C' \sqsubseteq m \text{ and } \sigma_A \sqsubseteq f(\pi_C^{<m}). \]

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 \( f \).

\textbf{Proof.} Since \( \sigma_A \subseteq f(\pi_C) \) 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 \( \sigma_A \) 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 \( \sigma_A|_{I_0} = f(\pi_C^{<n'}) \). By Lemma A.2 above, it follows that \( \pi_C^{<n} \) is a prefix of \( \pi_C' \) 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 \( \sigma_A \), 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 \( \sigma_A \) of any \( f(\pi_C) \), where \( \pi_C \in T(P \times C, S_C) \). There are two cases. Either \( \sigma_A \) is not a prefix of \( f^{lim}(\pi_C) \). Then it has the form \( f^{lim}(\pi_C)\sigma'_A \) where \( \sigma'_A \) 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 scheduled is then the next action of this sequence. Otherwise the definition uses Lemma A.3 to find unique index \( n \), such that \( f(\pi_C^{<n}) \subseteq \sigma_A \subseteq f(\pi_C^{<n} \cdot \alpha) \). where \( \alpha = m(\text{state}(\pi_C^{<n}), \text{obj}[n], \text{state}(f(\pi_C^{<n})). \text{obj}) \neq \epsilon \). Since \( \sigma_A \) is a proper prefix, there is an event \( a \), such that \( \sigma_A \cdot a \subseteq f^{lim}(\pi_C^{<n}) \cdot \alpha \), and \( a \) is an element of \( \alpha \). If \( a \) is an external action in \( I_P \), then \( a \) must be equal to \( \pi_C[n] \) (\( \alpha \) contains either \( \pi_C[n] \) if it is an external action, or no external action at all). In this case, we set \( S_A(\sigma_A) := \pi_C(\pi_C^{<n}) \). Note that \( a \) is enabled and in \( S_C(\pi_C^{<n}) \) in this case. Otherwise, when \( a \notin I_P \), we set \( S_A(\sigma_A) := \{a\} \).

**Theorem A.5.** \( S_A \) is well-defined.

**Proof.** Assume that \( \sigma_A \) 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 \( \sigma_A \subseteq f(\pi_C^{<m}) \). If \( \sigma_A \) is a proper prefix of \( f(\pi_C^{<n}) \), 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 \( \pi_C \) and \( \pi'_C \) could be different. If one of them is internal (i.e. not in \( I_P \)), 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 \( I_P \), but in \( S_C(\pi_C^{<m}) \). However, in this case \( S_A(f(\pi_C^{<n})) \) 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 \( \sigma_A \subseteq f(\pi_C) \), 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 \( \sigma_A \cdot a \subseteq f(\pi_C') \).

**Proof.** The case, where \( \sigma_A \not\subseteq f^{lim}(\pi_C) \) 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 \( f^{lim}(\pi_C^{<n}) \subseteq \sigma_A \subseteq f^{lim}(\pi_C^{<n+1}) \). Let \( \pi_C^{<n+1} = (\pi_C^{<n} \cdot a) \) and \( a = m(\text{state}(\pi_C^{<n}), \text{obj}, a, \text{state}(f(\pi_C^{<n})). \text{obj}) \).

**Case 1:** \( a \notin \Sigma_C \). Then \( a = a, a_1 = a \) 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 \( \pi'_1 \) with \( (\pi_C^{<n}) \cdot a \subseteq \pi'_1 \cdot \pi_1 \) has the required prefix \( \pi_C \cdot a_1 \) such that \( \sigma_A \cdot a_1 = f(\pi_C^{<n}) \cdot a \).
Case 2: \( a \notin \Sigma_C \). Then \( \alpha \) is a nonempty sequence of internal actions and \( \alpha \) is the only continuation of \( f(\pi_C^{<n}) \) compatible with \( S_A \). \( \sigma_A \) is \( f(\pi_C^{<n}) \) concatenated with some proper prefix of \( \alpha \).

“\( \Rightarrow \)”: If \( \sigma_A \cdot a \in T(P \times A, S_A) \), then \( a \) must be the next element in \( \alpha \). 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 \( \sigma_A \) in \( \pi'_1 \) 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 \((q_{P}^{ini}, q_{C}^{ini}, \sigma)) \in F_1 \). Now let \((q_{C}, (q_{P}, q_{C}, \sigma)) \in F_1 \) and let \( q_{C} \xrightarrow{a} q'_{C} \) 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.

Since \( \text{last}(\sigma) = q_{C} \), 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:

\((q_{P}^{ini}, q_{C}^{ini}, \sigma) \xrightarrow{g(q_{C}')} (q_{P}^{ini}, q_{C}, \sigma \cdot g(q_{C}')) \xrightarrow{a} (q_{P}^{ini}, q_{C}, \sigma \cdot g(q_{C}') \cdot a)\)

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.

Since \( \text{last}(\sigma) = q_{C} \), we get \( g(a, q_{C}') \in S_C(\sigma) \). By definition of \( S_C \) we hence get

\((q_{P}^{ini}, q_{C}^{ini}, \sigma) \xrightarrow{g(a, q_{C}')} (q_{a}, q_{C}, \sigma \cdot g(q_{C}')) \xrightarrow{a} (q_{rec(a)}, q_{C}', \sigma \cdot g(q_{C}') \cdot a) \xrightarrow{rec(a)} (q_{P}^{ini}, q_{C}', \sigma \cdot g(q_{C}') \cdot a \cdot rec(a))\)

We furthermore get \((q_{C}, (q_{P}^{ini}, q_{C}, \sigma \cdot g(q_{C}') \cdot a \cdot rec(a))) \in F_1 \).
We then say that an LTS is $q \times P$ simulation. This forward simulation is weak progressive, since the concrete system never executes more with this history where reach $h$, the initial states are the same since the initial state of deterministic systems is unique. Given we show that given two finite traces this is furthermore a weak progressive forward simulation as the matching steps are never executed more.

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

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

We then say that an LTS is $\Delta$-deterministic if the set of minimally reachable states for every history $h$ consists of a single element which we write $\text{minstate}_L(h)$.

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

Proof. We first prove that $P \times C \times S_C$ and $P \times A \times S_A$ are $\Gamma_P$-deterministic. To do so we show that given two finite traces $\sigma_1$ and $\sigma_2$ 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 $\Sigma_P$-deterministic too.

Lemma 6.4. If $C \Delta$-refines $A$ and $A$ is $\Delta$-deterministic, then there exists a $\Delta$-forward simulation.

Proof. Define the forward simulation $F$ as

$$F = \{(q_C, q_A) \mid \exists h \in \Delta^*, \text{reach}_C(h, q_C) \land q_A = \text{minstate}_A(h)\}$$

where $\text{reach}_C(h, q_C)$ and $\text{minstate}_A(h)$ 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 $q_C \Rightarrow q'_C$ 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 $q_A = \text{minstate}_A(h)$. 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 $\sigma$ of the remaining steps then has $a|_\Delta = \sigma|_\Delta$ and $q_A \xrightarrow{a}_A q'_A$ as required, and $(q'_C, q'_A) \in F$ holds by definition.

Lemma 6.5. There exists a weak progressive $\Sigma_P$-forward simulation $(F_2, \Rightarrow_2)$ between $P \times C \times S_C$ and $P \times A \times S_A$.

Proof. Since strong observational refinement implies $\Gamma_P$-refinement, and the abstract system is $\Sigma_P$-deterministic (Lemma B.2) Lemma 6.4 ensures that a $\Sigma_P$-forward simulation $F$ exists. This forward simulation is weak progressive, since the concrete system never executes more
than one internal action in a row ($S_C(a' \cdot a)$ 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 $q_p = q''_p$ and there is an internal action $a$ with $q_C \rightarrow a q_C'$ 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 $a \in \{g(q), g(e, q), \text{div}, \text{rec}(e)\}$. The proofs for each of these are trivial, but for completeness, we consider each of these in turn. We have the following transitions:

- $a = g(q)$. We have $(q'^{\text{ini}}_p, q_A, \sigma) \xrightarrow{g(q)} (q'^{\text{ini}}_p, q_A, \sigma \cdot g(q))$.
- $a = g(e, q)$. We have $(q'^{\text{ini}}_p, q_A, \sigma) \xrightarrow{g(e, q)} (q_e, q_A, \sigma \cdot g(e, q))$.
- $a = \text{div}$. We have $(q'^{\text{ini}}_p, q_A, \sigma) \xrightarrow{\text{div}} (q_{\text{div}}, q_A, \sigma)$.
- $a = \text{rec}(e)$. We have $(q_{\text{rec}(e)}, q_A, \sigma) \xrightarrow{\text{rec}(e)} (q'^{\text{ini}}_p, q_A, \sigma \cdot \text{rec}(e))$.

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:

- $a = e \in (I \cup R)$. We have $(q_e, q_A, \sigma) \xrightarrow{e} (q_{\text{rec}(e)}, q'_A, \sigma \cdot e)$.
- $a \in \Sigma_A \setminus (I \cup R)$. 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. $(q^{\text{ini}}_p, \_, \_, \_) \gg_3 (q_e, \_, \_, \_)$
2. $(q_{\text{rec}(e)}, \_, \_, \_) \gg_3 (q^{\text{ini}}_p, \_, \_, \_)$
3. $(q^{\text{ini}}_p, \_, \_, \_) \gg_3 (q_{\text{div}}, \_, \_, \_)$

Note that from $(q_{\text{div}}, \_, \_, \_)$, the only possible transition is an $\rightarrow$ step, where $a \in \Sigma_A \setminus (I \cup R)$, which is non-stuttering. Similarly, from $(q_e, \_, \_, \_)$, the only possible transition is $\rightarrow$, 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 $(q_{\text{rec}(e)}, \_, \_, \_)$ is guaranteed to reduce wrt. $\gg_3$, as are transitions corresponding to $g(e, q)$ and div from $(q^{\text{ini}}_p, \_, \_, \_)$.

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

Suppose there exists an infinite run

$$(q'^{\text{ini}}_p, q_A, \sigma) \xrightarrow{g(q)} (q'^{\text{ini}}_p, q_A, \sigma \cdot g(q)) \xrightarrow{g(q)} (q'^{\text{ini}}_p, q_A, \sigma \cdot g(q_1) \cdot g(q_2)) \xrightarrow{g(q)} \ldots$$

By construction, $P \times A \times S_A$ is an “abstraction” of $P \times C \times S_C$ such that $T(P \times A \times S_A) \upharpoonright R = T(P \times C \cdot S_C) \upharpoonright R$, thus, $T(P \times C \cdot S_C) \upharpoonright R$. Thus, there exists a $q_C$ such that $\text{last}(\sigma) = g(q_C)$ and

$$(q'^{\text{ini}}_p, q_C, \sigma) \xrightarrow{g(q)} (q'^{\text{ini}}_p, q_1, \sigma \cdot g(q_1) \cdot \tau_1) \xrightarrow{g(q)} (q'^{\text{ini}}_p, q_2, \sigma \cdot g(q_1) \cdot \tau_1, g(q_2) \cdot \tau_2) \xrightarrow{g(q)} \ldots$$
where $\tau_k \in \Sigma_C \setminus (I \cup R)$ for all $k$. Note that the definition of $S_C$ enforces a $\div(q_{C^k}, \tau_k)$ transition for each $\div(q_{C^k}, q_{\alpha^k})$ 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 $(q^\init_P, q_C, \sigma)$. 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 $(q^\init_P, q_A, \sigma)$. Now, $P \times A \times S_A$ contains a run with final state $\sigma$. Therefore, $P \times A \times S_A$ also contains a run

$$(q^\init_P, q_A, \sigma) \xrightarrow{\div} (q_{\div}, q_A, \sigma) \xrightarrow{r_1} (q_{\div}, q_1, \sigma) \xrightarrow{r_2} (q_{\div}, q_2, \sigma) \xrightarrow{r_3} \ldots$$

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 $\Delta$-forward simulation from $C$ to $B$, and $(F_2, \gg_2)$ one from $B$ to $A$. Then there exists a weak progressive $\Delta$-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 $\Delta$-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 $\alpha^k$ is a sequence of internal actions. Some $\alpha^k$ may be empty, so states may occur several times. It is, however, not possible that there is a final state $q_B^1$ such that $q_B^1 = q_B^{k+1} = \ldots$, since then $q_C^0 \gg q_C^{k+1} \gg \ldots$ would be implied, contradicting well-foundedness of $\gg_1$. Note that $q_B^1$ cannot start a diverging run in this case, otherwise $q_C^1 \gg q_C^{k+1}$ 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 $k$ $(q_A^k, q_B^k) \in F_2$ and $q_A^k \xrightarrow{\beta^k} q_A^{k+1}$ holds. Again the $\beta^k$ are sequences of internal actions, and the existence of a final state with $q_A^0 = q_A^{k+1} = \ldots$ 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. ▶