Almost Fair Simulations

ARTHUR CORRENSON, CISPA Helmholtz Center for Information Security, Germany
IONA KUHN, Saarland University, Germany
BERND FINKBEINER, CISPA Helmholtz Center for Information Security, Germany

It is well known that liveness properties cannot be proven using standard simulation arguments. This issue has been mitigated by extending standard notions of simulation for transition systems to *fairness-preserving simulations* for systems equipped with an additional fairness condition modeling liveness assumptions and/or liveness requirements. In the context of automated verification of finite-state systems, proofs by simulation are an appealing method as there exist efficient algorithms to find a simulation between two systems. However, applications of fair simulation to interactive verification have been much less studied. Perhaps one reason is that the definitions of fair simulation relations typically involve non-trivial nestings of inductive and coinductive relations, making them particularly difficult to use and to reason about. In this paper, we argue that in many cases, stronger notions of fair simulation involving more controlled alternations of fixed points are sufficient. Starting from known fair simulation techniques, we progressively build up a family of *almost fair* simulation relations for transition systems equipped with a Büchi fairness condition. The simulation relations we present can all be equipped with intuitive reasoning rules, leading to elegant deductive systems to prove fair trace inclusion. We mechanized our simulation relations and their associated deductive systems in the Rocq proof assistant, proved their soundness, and we demonstrate their use through a selection of examples.

CCS Concepts: • **Theory of computation** → **Automata over infinite objects**; *Program verification.*

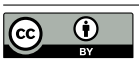
Additional Key Words and Phrases: Fairness, Interactive Verification, Coinduction

1 Introduction

Simulation proofs provide a systematic technique to reduce proofs of trace inclusion between programs into simple *local* reasoning about states and transitions. In the context of program verification, the problem of checking whether a program satisfies a given safety specification (e.g. specifications stating that *"nothing bad ever happens"*) can always be reformulated as a trace inclusion between a source transition system modeling the program and a target transition system that nondeterministically produces safe behaviors. In turn, simulation can immediately be applied as a simple technique to prove safety properties.

Unfortunately, this technique does not immediately work for liveness specifications (e.g. specifications stating that *"something good should eventually happen"*) such as termination or response properties. Indeed, modeling liveness specifications (and also programs with liveness assumptions) as transition systems typically requires augmenting the transition systems with an additional *fairness condition* describing which executions are legitimate, and which are not. In this context, proving trace inclusion means proving that any trace engendered by a fair execution of the source transition system can be reproduced via a fair execution of the target transition system. Standard simulation techniques would only ensure that any execution of the source can be mimicked by an arbitrary execution of the target, disregarding the fairness conditions. Instead, we need a notion of simulation that filters out unfair executions of the source, as these do not need to be simulated,

Authors' Contact Information: Arthur Correnson, CISPA Helmholtz Center for Information Security, Saarbrücken, Germany, arthur.correnson@cispa.de; Iona Kuhn, Saarland University, Saarbrücken, Germany, ioku00001@stud.uni-saarland.de; Bernd Finkbeiner, CISPA Helmholtz Center for Information Security, Saarbrücken, Germany, finkbeiner@cispa.de.



and further enforces that the remaining fair executions of the source can be simulated by *fair* executions of the target.

It is important to note that fairness conditions are typically *global* conditions inspecting an entire execution to determine whether it is fair or not. This contrasts with one of the main benefits of simulation techniques, which is *precisely* to reduce global reasoning about executions to local reasoning about states and transitions. In turn, designing fairness-preserving simulation relations is non-trivial and it requires to carefully approximate the global fairness conditions with more local checks on states and transitions.

To this extent, several extended notion of simulation such as direct simulation, delay simulation, and fair simulation have been proposed in the literature [2, 8, 11]. However, the focus has been mostly on the design of efficient algorithms to automatically construct simulations between two finite-state transition systems, or on algorithms to minimize large systems by computing their quotient modulo a fairness-preserving simulation relation [5]. The use of fairness-preserving simulation relations for interactive verification, and in particular interactive verification of liveness properties, has been comparatively much less explored.

In this paper, we propose to re-explore already established notions of fairness-preserving simulation, but from the point of view of interactive deductive verification inside a proof assistant. More precisely, we focus on fairness-preserving simulation of Büchi automata, an expressive computational model that is commonly used in the literature on automated verification of linear time properties. We present the following contributions:

(1) Our first contribution is to formalize direct simulation and delay simulation of Büchi automata in the Rocq proof assistant. While both notions are well-established in the literature, their rigorous formalization in a proof assistant turns out to be fairly technical. Further, by leveraging the framework of *parameterized coinduction*, we demonstrate that both direct and delay simulation can be presented in the form of a deductive system to prove language inclusion of Büchi automata interactively.

(2) Through a selection of examples, we precisely identify limitations of delay simulation and argue that for the purpose of interactive proofs of liveness properties, weaker notions of fairness-preserving simulations are required. We nonetheless observe that in the specific case where the left-hand automaton is a safety automaton (i.e., when we only have liveness requirements and no liveness assumptions), a simpler notion of *right-biased* delay simulation can be sufficient.

(3) To mitigate the limitations of direct and delay simulation in the case where the left-hand automaton is *not* a safety automaton (i.e., when we have both liveness assumptions and liveness requirements), we present two novel notions of fairness-preserving simulation: *double delay simulation*, and *repeated delay simulation*. Both are significantly weaker than delay simulation, but they remain sound for language inclusion of Büchi automata. Further, they preserve the simplicity and ease-of-use of delay simulation and can also be equipped with an intuitive set of reasoning rules. We mechanized these new notions of simulation and their proof of soundness in Rocq.

2 Preliminaries

Before proceeding with a description of the contributions, we start by introducing necessary background on labeled transition systems, Büchi automata, and parameterized coinduction, which all play a central role throughout the paper.

2.1 Labeled Transition Systems

A labeled transition system (LTS) is a triple $(\mathcal{S}, \mathcal{E}, \mathcal{I}, \rightarrow)$ where $\mathcal{S}$ is a set of states, $\mathcal{E}$ is a set of events, $\mathcal{I} \subseteq \mathcal{S}$ is a set of initial states, and $\rightarrow \subseteq \mathcal{S} \times \mathcal{E} \times \mathcal{S}$ is a labeled transition relation. $s_1 \xrightarrow{e} s_2$ indicates that it is possible to transition from s_1 to s_2 while emitting/reading the event e. Given a LTS $TS = (\mathcal{S}, \mathcal{E}, \mathcal{I}, \rightarrow)$, the set of *traces* that can be produced starting from a given state $s \in \mathcal{S}$ is the set of infinite sequences of events $\tau \in \mathcal{E}^\omega$ characterized by the following coinductive relation $Traces_{TS} \subseteq \mathcal{S} \times \mathcal{E}^\omega$.

Definition 2.1 (Traces). $Traces_{TS} \triangleq \nu T.\{\ (s, e \cdot \tau) \mid \exists s \xrightarrow{e} s'.\ (s', \tau) \in T\ \}$

For a specific state $s \in \mathcal{S}$ we note $Traces_{TS}(s) \triangleq \{\tau \mid (s, \tau) \in Traces_{TS}\}$. For a set of states $S \subseteq \mathcal{S}$ we note $Traces_{TS}(S) \triangleq \bigcup_{s \in S} Traces_{TS}(s)$. We will often consider the set $Traces_{TS}(\mathcal{I})$ of all *initial* traces of *TS*. Alternatively, we note $Traces(TS) \triangleq Traces_{TS}(\mathcal{I})$. When clear from context, we omit the subscript *TS*.

2.2 Büchi Automata

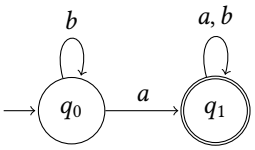
A Büchi automaton $(\mathcal{S}, \mathcal{E}, \mathcal{I}, \mathcal{F}, \rightarrow)$ is a LTS extended with a set of accepting states $\mathcal{F} \subseteq \mathcal{S}$ that should be visited infinitely often over the course of an execution for its associated trace to be considered valid. Traces that can be produced via infinitely many visits to accepting states are said to be in the language of the automaton. Formally, the language of a state s of an automaton A is characterized by the following coinductive-inductive relation $\mathcal{L}_A \subseteq \mathcal{S} \times \mathcal{E}^\omega$.

Definition 2.2 (Language).

$$\mathcal{L}_A \triangleq \nu L.\mu X.\ \{\ (s, e \cdot \tau) \mid \exists s \xrightarrow{e} s'.\ (s', \tau) \in X\ \} \cup \{\ (s, e \cdot \tau) \mid s \in \mathcal{F} \wedge \exists s \xrightarrow{e} s'.\ (s', \tau) \in L\ \}$$

As for traces, we use the notations $\mathcal{L}_A(s) \triangleq \{\tau \mid (s, \tau) \in \mathcal{L}_A\}$, $\mathcal{L}_A(S) \triangleq \bigcup_{s \in S} \mathcal{L}_A(s)$, and $\mathcal{L}(A) \triangleq \mathcal{L}_A(\mathcal{I})$.

Contrary to LTSs, which can only model safety requirements, Büchi automata can also model specifications with a liveness component. For example, the following automaton over events $\mathcal{E} = \{a, b\}$ expresses the requirement that an event "a" must eventually be emitted.



In this automaton, the only accepting state is marked with a double circle. Viewed as a LTS, this automaton accepts any infinite sequence of events a and b. However, the addition of the accepting set $\mathcal{F} = \{q_1\}$ effectively filters out traces containing only b's.

We note that Büchi automata are usually assumed to have finite state spaces (i.e., $|\mathcal{S}| < \infty$). This limits the specifications that can be encoded to so called *omega-regular properties* [3]. The advantage of considering only finite-state automata is that it gives access to efficient decision procedures to solve problems such as language inclusion. This observation is the basis of automata-based model-checking Linear Temporal Logics. In this paper, we are not concerned with automated verification. Instead, we aim to develop interactive deductive proof techniques. In this setting, assuming the finiteness of the state-space is an artificial limitation.

2.3 (Parameterized) Coinduction

Let A be a set, and $(\mathcal{P}(A), \subseteq, \cap, \cup, A, \emptyset)$ be the complete lattice of subsets of A (ordered by set inclusion). A result due to Tarski [17] guarantees that any monotone functor $F : \mathcal{P}(A) \xrightarrow{mon} \mathcal{P}(A)$ has a greatest fixed point noted νF. Further, νF is exactly the union of all postfixed points of F:

$$\nu F \triangleq \bigcup \{ R \mid R \subseteq F(R) \}$$

An immediate consequence of this result is that any coinductive predicate $\nu F \subseteq A$ has a systematic proof technique associated with it. Indeed, let $X \subseteq A$ and suppose we want to prove that $X \subseteq \nu F$. Since νF is larger than any postfixed point of F, it suffices to exhibit a postfixed point of R of F containing X. In this context, R is usually referred to as a *coinduction hypothesis*.

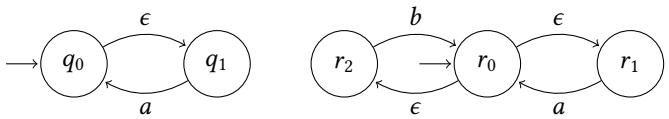
Lemma 2.3 (Coinduction Principle). $X \subseteq \nu F \iff \exists R, X \subseteq R \wedge R \subseteq F(R)$

A common use case for proofs by coinduction are proofs of trace inclusion by simulation. Given two LTSs $TS_1 = (\mathcal{S}_1, \mathcal{E}, \mathcal{I}_1, \rightarrow_1)$ and $TS_2 = (\mathcal{S}_2, \mathcal{I}_2, \mathcal{E}, \rightarrow_2)$ with the same set of events, to prove that they are *trace included* (i.e., $Traces(\mathcal{I}_1) \subseteq Traces(\mathcal{I}_2)$) it is well known that it suffices to show that $\mathcal{I}_1 \times \mathcal{I}_2 \subseteq \texttt{sim}$ where $\texttt{sim}$ is the binary relation defined coinductively as follows.

Definition 2.4 (Standard Simulation).

$$\texttt{simF}(X) \triangleq \{ (s_1, s_2) \mid \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2' \wedge (s_1', s_2') \in X \}$$
$$\texttt{sim} \triangleq \nu \texttt{simF}$$

Example 1. As an example, consider the two following LTSs labeled with events $\mathcal{E} = \{a, b, \epsilon\}$. The left-hand one could represent a program repeatedly emitting an event "a" and the right-hand one a safety specification requiring that only event "a" or event "b" is ever emitted.



It is not difficult to check that $R = \{(q_0, r_0), (q_1, r_1)\}$ contains $\mathcal{I}_1 \times \mathcal{I}_2 = \{(q_0, r_0)\}$ and is a postfixed point of $\texttt{simF}$. Therefore, by Lemma 2.3, $\mathcal{I}_1 \times \mathcal{I}_2 \subseteq \texttt{sim}$, and the left-hand system satisfies the right-hand specification. ◊

For this simple example, guessing a postfixed point up-front was not difficult. For larger systems, this can rapidly become more challenging. Starting with at least $\mathcal{I}_1 \times \mathcal{I}_2$ is necessary, but in practice it is not sufficient (in the above example, $\mathcal{I}_1 \times \mathcal{I}_2$ is not a postfixed point!), requiring to restart the proof with a slightly larger guess until we eventually find a large enough one. In the context of automated verification, this trial and error process to prove simulation is not necessarily a limitation, and intermediate results can be automatically cached and re-exploited. However, in the context of interactive deductive verification, in particular inside a proof assistant, it is cumbersome. In [12], it has been observed that even in an interactive setting, there is actually no strict need to guess the coinduction hypothesis up-front. Better, there is actually no need to ever provide a postfixed point at all and it generally suffices to collect fragments of a postfixed point instead [12, 19]. The key idea supporting this technique is to replace the usual fixed point operator ν with a parameterized version of it, $G_F(H)$, where H is a current *guess* for (a fragment of) the coinduction hypothesis. Formally, G is defined as follows:

$$G_F(H) \triangleq \nu X. F(X \cup H)$$

It is not difficult to see that G coincides with the standard greatest fixed point operator when $H = \emptyset$ (i.e., $G_F(\emptyset) = \nu F$). In particular, it means that proving $X \subseteq G_F(\emptyset)$ is equivalent to proving $X \subseteq \nu F$. The benefit is that the parameterized greatest fixed point admits the following *incremental* reasoning rules:

$$\textsc{Init}\ \frac{X \subseteq G_F(\emptyset)}{X \subseteq \nu.F} \qquad \textsc{Accumulate}\ \frac{X \subseteq G_F(H \cup X)}{X \subseteq G_F(H)} \qquad \textsc{Step}\ \frac{X \subseteq F(H \cup G_F(H))}{X \subseteq G_F(H)}$$

The rule Init initializes a proof by *parameterized coinduction* by replacing a goal of the form $X \subseteq \nu F$ with the equivalent $X \subseteq G_F(\emptyset)$. The rule Accumulate extends the current guess H with the subset X whose inclusion in the greatest fixed point should be proven. Finally, the rule Step enables to make progress by unfolding the greatest fixed point underlying the definition of G. We note that by definition of G, every element previously accumulated in H can be used after applying the rule Step. To better illustrate how to exploit the rule of parameterized coinduction, we revisit the previous example and present a detailed incremental proof that $\mathcal{I}_1 \times \mathcal{I}_2 \subseteq \mathtt{sim}$.

Example 2 (Example 1, revisited).

$$\begin{aligned}
& \{(q_0, r_0)\} \in \mathtt{sim} \\
\text{By Init} \quad \Longleftarrow\ & \{(q_0, r_0)\} \in G_{\mathtt{simF}}(\emptyset) \\
\text{By Accumulate} \quad \Longleftarrow\ & \{(q_0, r_0)\} \in G_{\mathtt{simF}}(\{q_0, r_0\}) \\
\text{By Step} \quad \Longleftarrow\ & \exists r_0 \xrightarrow{\epsilon} r.(q_1, r) \in \{q_0, r_0\} \cup G_{\mathtt{simF}}(\{q_0, r_0\}) \\
\text{Pick } r = r_1 \quad \Longleftarrow\ & (q_1, r_1) \in G_{\mathtt{simF}}(\{q_0, r_0\}) \\
\text{By Step} \quad \Longleftarrow\ & \exists r_1 \xrightarrow{a} r.(q_0, r) \in \{q_0, r_0\} \cup G_{\mathtt{simF}}(\{q_0, r_0\}) \\
\text{Pick } r = r_0 \quad \Longleftarrow\ & (q_0, r_0) \in \{q_0, r_0\} \cup G_{\mathtt{simF}}(\{q_0, r_0\}) \\
\Longleftarrow\ & (q_0, r_0) \in \{q_0, r_0\}
\end{aligned}$$

◊

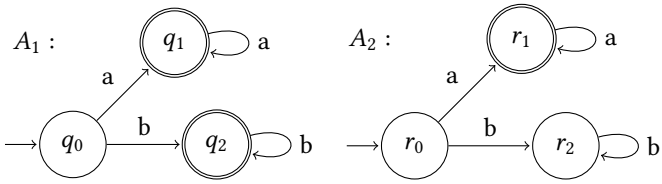
We note that in this example, even though $\{(q_0, r_0)\}$ is not a postfixed point of simF, we never needed to accumulate more that just the initial states! We however note that it is sometimes necessary to accumulate more than just the currently targeted subset X. For this purpose, the following Invariant rule can easily be derived as a corollary of Accumulate.

$$\textsc{Invariant}\ \frac{X \subseteq H' \qquad H' \subseteq G_F(H' \cup H)}{X \subseteq G_F(H)}$$

Proof. Suppose (1) $X \subseteq H'$, and (2) $H' \subseteq G_F(H' \cup H)$. By Accumulate and (2) we obtain $H' \subseteq G_F(H)$. By (1) it follows that $X \subseteq H' \subseteq G_F(H)$. □

3 Direct Simulation

In the previous section, we discussed the use of simulation combined with parameterized coinduction as an interactive proof technique to show trace inclusion between two LTSs. A natural question to ask is whether the same incremental proof technique can also be used to prove *language inclusion* of LTSs equipped with a fairness condition such as Büchi automata. Clearly, the standard notion of simulation is not suitable to check inclusion of Büchi automata, as it does not even mention accepting sets. For example, consider the following two automata:



Both automata have the same traces and they are even (bi)similar. However, they do not have the same languages! Indeed, the word b^ω is accepted by A_1 but rejected by A_2 as the only corresponding execution for b^ω loops in the state r_2, which is not an accepting state. To overcome this issue, a naive extension of standard simulation would be to synchronize visits to accepting states. In other words, whenever the left-state is accepting, we require the right-state to be accepting as well. This extension is usually referred to as *direct simulation*, and can be defined coinductively as follows:

$$\texttt{fsim}_{\texttt{direct}} \triangleq \nu X.\{\ (s_1, s_2) \mid (s_1 \in \mathcal{F}_1 \implies s_2 \in \mathcal{F}_2) \wedge \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'.\ (s_1', s_2') \in X\ \}$$

Since all visits to a left-accepting state are exactly synchronized with a visit to a right-accepting state, it is not too difficult to see that $\texttt{fsim}_{\texttt{direct}}$ offers a sound proof technique for language inclusion.

Theorem 3.1 (Soundness). $(s_1, s_2) \in \texttt{fsim}_{\texttt{direct}} \implies \mathcal{L}(s_1) \subseteq \mathcal{L}(s_2)$

Proof. The proof goes by coinduction on $\mathcal{L}(s_2)$ (remember that $\mathcal{L}$ is a coinductive predicate) and then by induction on the inductive part of $\mathcal{L}(s_1)$. □

Contrary to standard simulation, direction simulation (rightfully) rejects the two automata from the previous example. However, it can prove language inclusion of the previous example in the other direction (i.e., $\mathcal{L}(A_2) \subseteq \mathcal{L}(A_1)$).

Example 3. We prove that $\mathcal{L}_{A_2}(r_0) \subseteq \mathcal{L}_{A_1}(q_0)$ by showing that (r_0, q_0) is included in $\texttt{fsim}_{\texttt{direct}}$. We pick $R = \{(r_0, q_0), (r_1, q_1), (r_2, q_2)\}$ and we observe that $(r_0, q_0) \in R \subseteq \texttt{fsimF}_{\texttt{direct}}(R)$. Thus, by Lemma 2.3, $(r_0, q_0) \in \texttt{fsim}_{\texttt{direct}}$ and by Theorem 3.1 it follows that $\mathcal{L}_{A_2}(r_0) \subseteq \mathcal{L}_{A_1}(q_0)$. ◊

3.1 A Simple Deductive System via Parameterized Coinduction

Although direct simulation is a very strong notion of fairness-preserving simulation, and thus limited in applicability (we discuss these limitations in Section 3.2), its simplicity makes it a good starting point for interactive proofs. Although guessing a direct simulation is not difficult for small Büchi automata, for larger automata, and in particular automata with infinite state spaces, guessing a direct simulation might be more difficult. Instead, as discussed in the preliminaries, one can exploit *parameterized coinduction* to obtain a set of customized reasoning rules (a deductive system) for proving that pairs of states are contained in $\texttt{fsim}_{\texttt{direct}}$.

The deductive system we propose is operating on triples $\boxed{H} \vdash s_1 \preccurlyeq s_2$ and $\boxed{\boxed{H}} \vdash s_1 \preccurlyeq s_2$ where H is a relation between states representing a fragment of a direct simulation, and s_1 and s_2 are two states. Intuitively, a solid box $\boxed{H}$ indicates that H is *guarded* and cannot immediately be used, whereas a dashed box $\boxed{\boxed{H}}$ indicates that H can be used to conclude that the target state are in direct simulation. This notation is borrowed from [4] and [6], and these triples are defined in terms of the parameterized greatest fixed point operator as follows:

$$\boxed{H} \vdash s_1 \preccurlyeq s_2 \triangleq (s_1, s_2) \in G_{\texttt{fsimF}_{\texttt{direct}}}(H)$$
$$\boxed{\boxed{H}} \vdash s_1 \preccurlyeq s_2 \triangleq (s_1, s_2) \in H \cup G_{\texttt{fsimF}_{\texttt{direct}}}(H)$$

$$\textsc{Final}\ \frac{s_2 \in \mathcal{F}_2 \qquad \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'. \boxed{H} \vdash s_1' \preccurlyeq s_2'}{\boxed{H} \vdash s_1 \preccurlyeq s_2} \qquad \textsc{Step}\ \frac{s_1 \notin \mathcal{F}_1 \qquad \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'. \boxed{H} \vdash s_1' \preccurlyeq s_2'}{\boxed{H} \vdash s_1 \preccurlyeq s_2}$$

$$\textsc{Cycle}\ \frac{(s_1, s_2) \in H}{\boxed{H} \vdash s_1 \preccurlyeq s_2} \qquad \textsc{Guard}\ \frac{\boxed{H} \vdash s_1 \preccurlyeq s_2}{\boxed{H} \vdash s_1 \preccurlyeq s_2} \qquad \textsc{Invariant}\ \frac{(s_1, s_2) \in H' \qquad \forall (s_1', s_2') \in H'. \boxed{H \cup H'} \vdash s_1' \preccurlyeq s_2'}{\boxed{H} \vdash s_1 \preccurlyeq s_2}$$

Fig. 1. Rules for $\mathtt{fsim}_{\mathtt{direct}}$

Note that here, $\mathtt{fsimF}_{\mathtt{direct}}$ denotes the monotone functor underlying the definition of $\mathtt{fsim}_{\mathtt{direct}}$. Throughout the paper, we reuse this notation convention heavily: for any coinductive definition of the form $\mathtt{rel} \triangleq \nu X.F(X)$, we note $\mathtt{relF}$ for F.

Since our triples are defined in terms of the parameterized greatest fixed point operator (see Section 2.3), they inherit the reasoning principles of parameterized coinduction. In particular, $\boxed{\emptyset} \vdash s_1 \preccurlyeq s_2 \iff (s_1, s_2) \in \mathtt{fsim}_{\mathtt{direct}}$ and, additionally, the rules presented in Figure 1 can immediately be derived.

The Final and Step rules are stepping through the two targeted automata, ensuring that every step in the left-hand automaton can be reproduced by at least one step in the right-hand automaton. We note that following the intuition of direct simulation, progress can only be made via Step and Final if the current left-state is rejecting ($s_1 \notin \mathcal{F}_1$) or if the current right-state is accepting ($s_2 \in \mathcal{F}_2$). Additionally, we note that making progress via Step or Final always releases the guard. The remaining rules Cycle, Guard and Invariant are for handling the parameter H. Cycle allows to conclude a proof whenever H is unguarded and already contains the targeted pair of states. The rule Guard restores the guard around an unguarded hypothesis. Finally, the rule Invariant extends the current hypothesis H with a relation H'. Important, H' needs to contain at least the currently targeted pair of states. Further, after applying Invariant, the proof resumes from an arbitrary pair of states in H' and the guard is maintained (thus requiring to later make progress via Final or Step).

Example 4. We prove that $\mathcal{L}_{A_2}(r_0) \subseteq \mathcal{L}_{A_1}(q_0)$ by showing that $\boxed{\emptyset} \vdash r_0 \preccurlyeq q_0$.

$$\begin{aligned}
& \boxed{\emptyset} \vdash r_0 \preccurlyeq q_0 \\
\text{By Step and Guard} \quad \Longleftarrow\ & \boxed{\emptyset} \vdash r_1 \preccurlyeq q_1 \wedge \boxed{\emptyset} \vdash r_2 \preccurlyeq q_2 \\
\text{By Invariant} \quad \Longleftarrow\ & \boxed{(r_1, q_1)} \vdash r_1 \preccurlyeq q_1 \wedge \boxed{(r_2, q_2)} \vdash r_2 \preccurlyeq q_2 \\
\text{By Final as } q_1, q_2 \in \mathcal{F}_2 \quad \Longleftarrow\ & \boxed{(r_1, q_1)} \vdash r_1 \preccurlyeq q_1 \wedge \boxed{(r_2, q_2)} \vdash r_2 \preccurlyeq q_2 \\
\text{By Cycle} \quad \Longleftarrow\ & (r_1, q_1) \in \{(r_1, q_1)\} \wedge (r_2, q_2) \in \{(r_2, q_2)\}
\end{aligned}$$

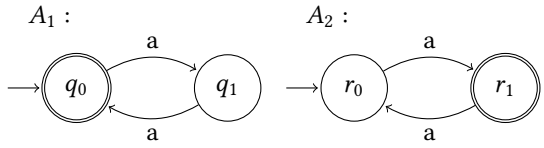
We note that in this proof, we omitted the brackets around sets of pairs for readability (e.g., we noted $\boxed{(r_1, q_1)}$ instead of $\boxed{\{(r_1, q_1)\}}$). We adopt this notation for the remaining of the paper. ◊

3.2 Limitations of Direct Simulation

In the previous section, we have demonstrated how direct simulation, a simple extension of the standard notion of simulation, can be used to prove language inclusion of Büchi automata. Furthermore, we showed that combined with parameterized coinduction, direct simulation can be

presented as a simple and intuitive deductive system to prove language containment between two Büchi automata. Nonetheless, direct simulation is still way too strong a simulation relation and it is easy to see that $\texttt{fsim}_{\texttt{direct}}$ is incomplete.

Example 5 (Incompleteness of $\texttt{fsim}_{\texttt{direct}}$). Let us consider the two following Büchi automata A_1 and A_2.



Clearly, $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$ as both automata accepts exactly the language $L = \{a^\omega\}$. However, $(q_0, r_0) \notin \texttt{fsim}_{\texttt{direct}}$. The only accepting run for a^ω in A_1 visits the accepting state at even positions, whereas the only accepting run in A_2 only visits an accepting state at odd positions. To be more formal, for these two specific automata, the largest direct simulation is $\texttt{fsim}_{\texttt{direct}} = \{(q_0, r_1), (q_1, r_0), (q_1, r_1)\}$ and $(q_0, r_0) \notin \texttt{fsim}_{\texttt{direct}}$. ◊

Beyond this specific example, simulation techniques are notoriously incomplete without making further assumptions on the transition systems, even without having to handle fairness conditions [1, 3, 15]. In fact, any refinement of standard simulation (i.e., any relation $R \subseteq \texttt{sim}$) is necessarily incomplete for language inclusion (because $\texttt{sim}$ is incomplete for trace inclusion, and trace inclusion is equivalent to language inclusion when marking all states as accepting). In spite of this inherent source of incompleteness, the simplicity of direct simulation makes it an appealing proof technique. Further, as demonstrated in Example 3, direct simulation is well-suited for interactive deductive reasoning.

4 Delayed Notions of Simulation

4.1 Delay Simulation

Although any refinement of standard simulation is necessarily incomplete for language inclusion, direct simulation is unnecessarily strong: by definition, it is sensitive to the exact number of computation steps that separate visits to accepting states. This renders the technique essentially unusable for any verification task. Indeed, in practice, a program (combined with fairness assumptions) will typically execute for a data-dependent number of steps in between intermediate goals. On the other hand, specification automata are fixed and require more precisely timed visits to accepting states. Consequently, $\texttt{fsim}_{\texttt{direct}}$ simulation is too strong to reason about programs and specifications.

A weaker notion of fairness-preserving simulation is delay simulation [8]. As for direct simulation, delay simulation extends standard simulation by further enforcing that infinitely many visits to a left-accepting state are simulated by infinitely many visits to a right-accepting state. However, contrary to direct simulation, delay simulation does not force visits to accepting states to be exactly synchronized. Instead, whenever a left-accepting state is encountered, delay simulation permits postponing (for a bounded number of steps) the moment when a corresponding right-accepting state is going to be reached. This intuition is achieved by nesting a coinductive and an inductive relation.

Definition 4.1 (Delay Simulation). Let $(\mathcal{S}_1, \mathcal{E}, \mathcal{I}_1, \rightarrow_1, \mathcal{F}_1)$ and $(\mathcal{S}_2, \mathcal{E}, \mathcal{I}_2, \rightarrow_2, \mathcal{F}_2)$ be two Büchi automata over the same set of events $\mathcal{E}$. We define the two relations $\mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}$ and $\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$ as follows:

$$
\begin{aligned}
&\mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}(X) \triangleq \mu Y. \\
&\quad \text{(R-Final)} \quad \{\ (s_1, s_2) \mid s_2 \in \mathcal{F}_2 \wedge \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'.\ (s_1', s_2') \in X\ \} \cup \\
&\quad \text{(R-Delay)} \quad \{\ (s_1, s_2) \mid \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'.\ (s_1', s_2') \in Y\ \} \\
&\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}} \triangleq \nu X. \\
&\quad \text{(L-to-R)} \quad \{\ (s_1, s_2) \mid (s_1, s_2) \in \mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}(X)\ \} \cup \\
&\quad \text{(L-Step)} \quad \{\ (s_1, s_2) \mid s_1 \notin \mathcal{F}_1 \wedge \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'.\ (s_1', s_2') \in X\ \}
\end{aligned}
$$

Our formulation of delay simulation combines an inductive relation $\mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}$ and a coinductive relation $\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$. The coinductive relation $\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$ is the entry point, and it tracks left-accepting states. Whenever a left-accepting state is encountered, $\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$ transfers the control to $\mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}$, which intuitively requires to reach a right-accepting state. More precisely, $\mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}$ contains all pairs of states from which, no matter the steps taken by the left-hand automaton, the right-hand automaton can always mimic these steps in such a way that a right-accepting state is eventually found. Once such a state is attained, $\mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}$ transfers the control back to $\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$. It is important to note that when switching from $\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$ to $\mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}$ (see (L-to-R) in Definition 4.1), no computation steps need to be taken. However, switching back from $\mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}$ to $\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$ (see (R-final) in Definition 4.1) does require taking a step. Without this subtle difference, $\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$ would contain any pair of states (s_1, s_2) with $s_2 \in \mathcal{F}_2$, and it would not be a sound proof technique for language inclusion. In general, defining fair simulation relations require extra care. Errors like off-by-one-step errors are easily introduced and one has to be particularly careful to make sure that *progress* is always made before corecursing.

We further note that there are other equivalent ways to define $\mathtt{fsim}_{\mathtt{delay}}$. For example, instead of requiring to take a step before switching back to $\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$ in (R-Final), one could also take a step before switching to $\mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}$ in (L-to-R) and add a third rule allowing to corecurse when a right-accepting state is encountered in $\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$. In principle, the style of definition does not matter much. However, we observed that in practice, the exact choice of formalization can significantly impact the complexity of the soundness proof.

Before proving the soundness of $\mathtt{fsim}_{\mathtt{delay}}$ for language inclusion, we make two useful observations. First, we observe that $\mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}(\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}) \subseteq \mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$.

Lemma 4.2. $\mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}(\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}) \subseteq \mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$

Proof. Let $(s_1, s_2) \in \mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}(\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}})$, we have to show that $(s_1, s_2) \in \mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$. By unfolding $\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$, it is equivalent to prove $(s_1, s_2) \in \mathtt{fsimF}^{\mathsf{L}}_{\mathtt{delay}}(\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}})$. Using the (L-to-R) disjunct, it is enough to prove $(s_1, s_2) \in \mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}(\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}})$. This is exactly our assumption. □

Another useful observation is that $\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$ is a postfixed point of $\mathtt{simF}$. By Lemma 2.3, this implies that $\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$ is stronger than standard simulation (i.e., $\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}} \subseteq \mathtt{sim}$).

Lemma 4.3. $(s_1, s_2) \in \mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}} \implies \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'.\ (s_1', s_2') \in \mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$

Proof. Suppose $(s_1, s_2) \in \mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$. By unfolding the greatest fixed point we know that in particular $(s_1, s_2) \in \mathtt{fsimF}^{\mathsf{L}}_{\mathtt{delay}}(\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}})$. We then proceed by case analysis on $\mathtt{fsimF}^{\mathsf{L}}_{\mathtt{delay}}$. The (L-Step) case is trivial. In the (L-to-R) case, we have $(s_1, s_2) \in \mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}(\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}})$ and by Lemma 4.2 it immediately follows that $(s_1, s_2) \in \mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$. □

Using Lemma 4.2 and Lemma 4.3, we are ready to prove the soundness of $\mathtt{fsim}_{\mathtt{delay}}$.

Theorem 4.4 (Soundness of $\mathtt{fsim}_{\mathtt{delay}}$). $(s_1, s_2) \in \mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}} \implies \mathcal{L}(s_1) \subseteq \mathcal{L}(s_2)$

Proof. (Sketch) The proof proceeds by coinduction on $\mathcal{L}(s_2)$, and then by induction on the inductive part of $\mathcal{L}(s_1)$. In the cases where the state s_1 is final, we do a second induction on $\mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}$, applying Lemma 4.3 along the way until a right-accepting state is reached. We then conclude by coinduction. In the case where s_1 is not final, we conclude using Lemma 4.3 and the induction hypothesis. □

4.2 Basic Deductive System and Examples

Instead of giving examples of proofs by delay simulation using the raw definition of $\mathtt{fsim}_{\mathtt{delay}}$, we instead directly present its associated deductive system. The deductive system is operating on the following three kinds of triples:

$$\boxed{H} \vdash_{\mathsf{d}} s_1 \preccurlyeq_{\mathsf{L}} s_2 \triangleq (s_1, s_2) \in G_{\mathtt{fsimF}^{\mathsf{L}}_{\mathtt{delay}}}(H)$$

$$\boxed{\boxed{H}} \vdash_{\mathsf{d}} s_1 \preccurlyeq_{\mathsf{L}} s_2 \triangleq (s_1, s_2) \in H \cup G_{\mathtt{fsimF}^{\mathsf{L}}_{\mathtt{delay}}}(H)$$

$$\boxed{H} \vdash_{\mathsf{d}} s_1 \preccurlyeq_{\mathsf{R}} s_2 \triangleq (s_1, s_2) \in \mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}(H \cup G_{\mathtt{fsimF}^{\mathsf{L}}_{\mathtt{delay}}}(H))$$

Here, the subscripts L and R indicates whether we are currently tracking **L**eft-accepting states (with $\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$) or currently searching for a **R**ight-accepting one (with $\mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}$). As for $\mathtt{fsim}_{\mathtt{direct}}$, specializing the rules of parameterized coinduction (see Section 2.3) immediately gives a basic set of reasoning principles which we list in Figure 2.

$$\text{L-to-R}\ \frac{\boxed{H} \vdash_{\mathsf{d}} s_1 \preccurlyeq_{\mathsf{R}} s_2}{\boxed{H} \vdash_{\mathsf{d}} s_1 \preccurlyeq_{\mathsf{L}} s_2} \qquad \text{L-Step}\ \frac{s_1 \notin \mathcal{F}_1 \qquad \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'. \boxed{\boxed{H}} \vdash_{\mathsf{d}} s_1' \preccurlyeq_{\mathsf{L}} s_2'}{\boxed{H} \vdash_{\mathsf{d}} s_1 \preccurlyeq_{\mathsf{L}} s_2}$$

$$\text{R-Delay}\ \frac{\forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'. \boxed{H} \vdash_{\mathsf{d}} s_1' \preccurlyeq_{\mathsf{R}} s_2'}{\boxed{H} \vdash_{\mathsf{d}} s_1 \preccurlyeq_{\mathsf{R}} s_2} \qquad \text{R-Final}\ \frac{s_2 \in \mathcal{F}_2 \qquad \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'. \boxed{\boxed{H}} \vdash_{\mathsf{d}} s_1' \preccurlyeq_{\mathsf{L}} s_2'}{\boxed{H} \vdash_{\mathsf{d}} s_1 \preccurlyeq_{\mathsf{R}} s_2}$$

$$\text{L-Cycle}\ \frac{(s_1, s_2) \in H}{\boxed{\boxed{H}} \vdash_{\mathsf{d}} s_1 \preccurlyeq_{\mathsf{L}} s_2} \qquad \text{L-Guard}\ \frac{\boxed{H} \vdash_{\mathsf{d}} s_1 \preccurlyeq_{\mathsf{L}} s_2}{\boxed{\boxed{H}} \vdash_{\mathsf{d}} s_1 \preccurlyeq_{\mathsf{L}} s_2}$$

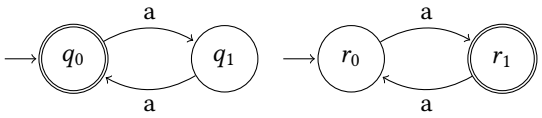
$$\text{L-Invariant}\ \frac{(s_1, s_2) \in H' \qquad \forall (s_1', s_2') \in H'. \boxed{H \cup H'} \vdash_{\mathsf{d}} s_1' \preccurlyeq_{\mathsf{L}} s_2'}{\boxed{H} \vdash_{\mathsf{d}} s_1 \preccurlyeq_{\mathsf{L}} s_2}$$

Fig. 2. Basic rules for $\mathtt{fsim}_{\mathtt{delay}}$

The rules L-to-R, L-Step, R-Delay and R-Final are immediately obtained by unfolding the definition of $\mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}$ and by instantiating the Step rule of parameterized coinduction with $\mathtt{fsimF}^{\mathsf{L}}_{\mathtt{delay}}$ as the underlying monotone functor. We note that without further assumptions, the context H can only be exploited and modified (via the rules L-Cycle, L-Guard, and L-Invariant)

when the goal is an L-triple $\boxed{H} \vdash_{\mathsf{d}} s_1 \preccurlyeq_{\mathsf{L}} s_2$ or $\boxed{\boxed{H}} \vdash_{\mathsf{d}} s_1 \preccurlyeq_{\mathsf{L}} s_2$! Indeed, when the goal is a R-triple $\boxed{H} \vdash_{\mathsf{d}} s_1 \preccurlyeq_{\mathsf{R}} s_2$, the context H is guarded by an application of the inductive relation $\mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}$, thus forbidding to exploit the rules of parameterized coinduction. In Section 4.3, we discuss this limitation, and we show that with a little more work, R-triples can also be equipped with additional rules to modify the context. For now, we focus on an example using only the basic rules of Figure 2.

Example 6. Recall the following two automata from the previous section:



We already discussed that there exists no direct simulation between these two automata. However, they can be proven to be language included by delay simulation. In particular, we prove that $\boxed{\emptyset} \vdash_{\mathsf{d}} q_0 \preccurlyeq_{\mathsf{L}} r_0$.

We start by using the L-Invariant rule to add the current states to the context.

$$\boxed{\emptyset} \vdash_{\mathsf{d}} q_0 \preccurlyeq_{\mathsf{L}} r_0$$
$$\text{By L-Invariant} \impliedby \boxed{(q_0, r_0)} \vdash_{\mathsf{d}} q_0 \preccurlyeq_{\mathsf{L}} r_0$$

As s_0 is final, the only rule we can apply to make further progress is the L-to-R rule, thus replacing the current L-triple with an R-triple, and initiating a search for a right-accepting state.

$$\text{By L-to-R} \impliedby \boxed{(q_0, r_0)} \vdash_{\mathsf{d}} q_0 \preccurlyeq_{\mathsf{R}} r_0$$

As r_0 is not final, the only option is to apply rule R-Delay to investigate all possible ways to transition out of (q_0, r_0). Here, the only option is to move to (q_1, r_1) via the event a.

$$\text{By R-Delay} \impliedby \boxed{(q_0, r_0)} \vdash_{\mathsf{d}} q_1 \preccurlyeq_{\mathsf{R}} r_1$$

Since r_1 is final, we can circle back to an L-triple with R-Final.

$$\text{By R-Final} \impliedby \boxed{\boxed{(q_0, r_0)}} \vdash_{\mathsf{d}} q_0 \preccurlyeq_{\mathsf{L}} r_0$$

Now, we are back in (q_0, r_0) and importantly, the guard around the context is released. Hence, we can use the L-Cycle rule to conclude the proof.

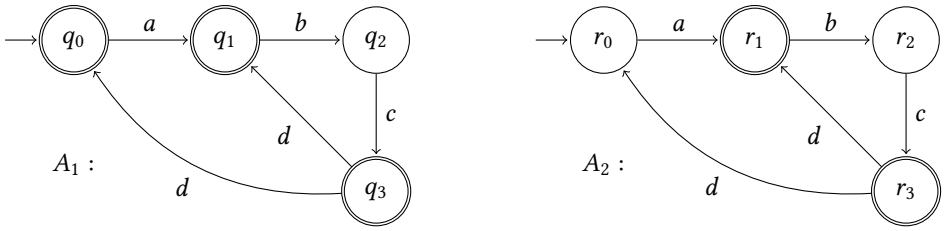
$$\text{By L-Cycle} \impliedby (q_0, r_0) \in \{(q_0, r_0)\}$$

◊

4.3 Additional Reasoning Rules

Strictly by following the definition of our triples, the rules of parameterized coinduction cannot be applied to R-triples. In particular, with the current definition of $\mathtt{fsim}_{\mathtt{delay}}$ it would be incorrect to use the context H to prove a R-triple. Indeed, this would allow us to exploit any cycle to abandon an obligation to reach a right-accepting state. Even though using the context during a proof of an R-triple is unsound, extending it by accumulating pairs of states visited during the proof of an R-triple can be a convenient feature to avoid redundant proof steps. We demonstrate this observation with an example.

Example 7. Consider the following two Büchi automata:



Both automata have exactly the same states and transitions. However, A_1 has its initial state marked as accepting but not A_2. This difference does not change the language recognized by A_2 and $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$ (in fact, we even have $\mathcal{L}(A_1) = \mathcal{L}(A_2)$). We prove that $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$ by delay simulation.

$$\boxed{\emptyset} \vdash_{\mathsf{d}} q_0 \preccurlyeq_{\mathsf{L}} r_0$$

$$\text{By L-Invariant} \quad \Longleftarrow \quad \boxed{(q_0, r_0)} \vdash_{\mathsf{d}} q_0 \preccurlyeq_{\mathsf{L}} r_0$$

$$\text{By L-to-R and R-Delay} \quad \Longleftarrow \quad \boxed{(q_0, r_0)} \vdash_{\mathsf{d}} q_1 \preccurlyeq_{\mathsf{R}} r_1$$

$$\text{By 2 times R-Delay} \quad \Longleftarrow \quad \boxed{(q_0, r_0)} \vdash_{\mathsf{d}} q_3 \preccurlyeq_{\mathsf{R}} r_3$$

$$\text{By R-Final} \quad \Longleftarrow \quad \boxed{(q_0, r_0)} \vdash_{\mathsf{d}} q_1 \preccurlyeq_{\mathsf{L}} r_1 \wedge \boxed{(q_0, r_0)} \vdash_{\mathsf{d}} q_0 \preccurlyeq_{\mathsf{L}} r_0$$

At this point, the goal $\boxed{(q_0, r_0)} \vdash_{\mathsf{d}} q_0 \preccurlyeq_{\mathsf{L}} r_0$ can be immediately discharged by L-Cycle. However, we can not yet conclude for $\boxed{(q_0, r_0)} \vdash_{\mathsf{d}} q_1 \preccurlyeq_{\mathsf{L}} r_1$. We therefore have to extend our current invariant.

$$\boxed{(q_0, r_0)} \vdash_{\mathsf{d}} q_1 \preccurlyeq_{\mathsf{L}} r_1$$

$$\text{By L-Guard and L-Invariant} \quad \Longleftarrow \quad \boxed{(q_0, r_0), (q_1, r_1)} \vdash_{\mathsf{d}} q_1 \preccurlyeq_{\mathsf{L}} r_1$$

$$\text{By L-to-R and 2 times R-Delay} \quad \Longleftarrow \quad \boxed{(q_0, r_0), (q_1, r_1)} \vdash_{\mathsf{d}} q_3 \preccurlyeq_{\mathsf{L}} r_3$$

$$\text{By R-Final and L-Cycle} \quad \Longleftarrow \quad (q_0, r_0) \in \{(q_0, r_0), (q_1, r_1)\} \wedge (q_1, r_1) \in \{(q_0, r_0), (q_1, r_1)\}$$

Observe that the last steps of reasoning are duplicated! Indeed, we already encountered the pair of states (q_1, r_1) earlier in the proof when we proved the R-triple $\boxed{(q_0, r_0)} \vdash_{\mathsf{d}} q_1 \preccurlyeq_{\mathsf{R}} r_1$. Intuitively, at this point, we would have wanted to extend our context with (q_1, r_1), thus allowing us to immediately discharge the two goals generated by the first application of R-Final. Unfortunately, the current rules do not allow to extend the context while establishing a R-triple. ◊

Even though the rules of parameterized coinduction do not immediately allow us to accumulate pairs of states visited during the proof of a R-triples, by definition all such pairs are still guaranteed to reach a right-accepting state. Intuitively, nothing should prevent us from remembering this fact by accumulating pairs of states in the context. In the following, we prove that this intuition is indeed correct. More precisely, we prove that the following R-Invariant rule is sound:

$$\text{R-Invariant} \quad \frac{(s_1, s_2) \in H' \qquad H' \subseteq \mathcal{F}_1 \times \mathcal{S}_2 \qquad \forall (s_1', s_2') \in H'.\ \boxed{H \cup H'} \vdash_{\mathsf{d}} s_1 \preccurlyeq_{\mathsf{R}} s_2}{\boxed{H} \vdash_{\mathsf{d}} s_1 \preccurlyeq_{\mathsf{R}} s_2}$$

To establish the soundness of R-Invariant, we first observe that when $s_1 \in \mathcal{F}_1$, $\boxed{H} \vdash s_1 \preccurlyeq_{\mathsf{L}} s_2$ also implies $\boxed{H} \vdash s_1 \preccurlyeq_{\mathsf{R}} s_2$.

Lemma 4.5. $s_1 \in \mathcal{F}_1 \implies \boxed{H} \vdash s_1 \preccurlyeq_{\mathsf{L}} s_2 \implies \boxed{H} \vdash s_1 \preccurlyeq_{\mathsf{R}} s_2$

Proof. Suppose $s_1 \in \mathcal{F}_1$ and $\boxed{H} \vdash s_1 \preccurlyeq_{\mathsf{R}} s_2$. By unfolding the definition of L-triples, we have two cases. Either $(s_1, s_2) \in \mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}(H \cup G_{\mathtt{fsim}\mathsf{F}^{\mathsf{L}}_{\mathtt{delay}}}(H))$. This is exactly the definition of $\boxed{H} \vdash s_1 \preccurlyeq_{\mathsf{R}} s_2$. Otherwise, $s_1 \notin \mathcal{F}_1$ and for every successor s_1' of s_1, there is a successor of s_2' of s_2 with $\boxed{\boxed{H}} \vdash s_1 \preccurlyeq_{\mathsf{L}} s_2$. This case is contradictory with the assumption that $s_1 \in \mathcal{F}_1$. □

Lemma 4.6. *The rule R-Invariant is sound.*

Proof. Let (i) $H' \subseteq \mathcal{F}_1 \times \mathcal{S}_2$ and suppose (ii) that $(s_1, s_2) \in H'$. Further, assume that (iii) $\forall (s_1', s_2') \in H'. \boxed{H \cup H'} \vdash_{\mathsf{d}} s_1' \preccurlyeq_{\mathsf{R}} s_2'$. From these assumptions, we have to derive $\boxed{H} \vdash_{\mathsf{d}} s_1 \preccurlyeq_{\mathsf{R}} s_2$. By (i) and (ii) we can deduce that $s_1 \in \mathcal{F}_1$. We can then use Lemma 4.5 to establish the following chain of implications:

$$
\begin{aligned}
& \boxed{H} \vdash_{\mathsf{d}} s_1 \preccurlyeq_{\mathsf{R}} s_2 \\
\text{By Lemma 4.5 and } s_1 \in \mathcal{F}_1 \quad \Longleftarrow\ & \boxed{H} \vdash_{\mathsf{d}} s_1 \preccurlyeq_{\mathsf{L}} s_2 \\
\text{By L-Invariant using } H' \quad \Longleftarrow\ & \forall (s_1', s_2') \in H'. \boxed{H \cup H'} \vdash_{\mathsf{d}} s_1' \preccurlyeq_{\mathsf{L}} s_2' \\
\text{By L-to-R} \quad \Longleftarrow\ & \forall (s_1', s_2') \in H'. \boxed{H \cup H'} \vdash_{\mathsf{d}} s_1' \preccurlyeq_{\mathsf{R}} s_2'
\end{aligned}
$$

The last statement is exactly (iii), which concludes the proof. □

Using the new rule R-Invariant, we can now rework our previous example by accumulating (q_1, r_1) into the context the first time it is encountered. The corresponding proof is summarized as follows:

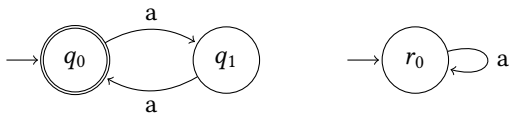
$$
\begin{aligned}
& \boxed{\emptyset} \vdash_{\mathsf{d}} q_0 \preccurlyeq_{\mathsf{L}} r_0 \\
\text{By L-Invariant} \quad \Longleftarrow\ & \boxed{(q_0, r_0)} \vdash_{\mathsf{d}} q_0 \preccurlyeq_{\mathsf{L}} r_0 \\
\text{By L-to-R and R-Delay} \quad \Longleftarrow\ & \boxed{(q_0, r_0)} \vdash_{\mathsf{d}} q_1 \preccurlyeq_{\mathsf{R}} r_1 \\
\textbf{By R-Invariant} \text{ and } q_1 \in \mathcal{F}_1 \quad \Longleftarrow\ & \boxed{(q_0, r_0), (q_1, r_1)} \vdash_{\mathsf{d}} q_1 \preccurlyeq_{\mathsf{R}} r_1 \\
\text{By 2 times R-Delay} \quad \Longleftarrow\ & \boxed{(q_0, r_0), (q_1, r_1)} \vdash_{\mathsf{d}} q_3 \preccurlyeq_{\mathsf{R}} r_3 \\
\text{By R-Final} \quad \Longleftarrow\ & \boxed{\boxed{(q_0, r_0), (q_1, r_1)}} \vdash_{\mathsf{d}} q_1 \preccurlyeq_{\mathsf{L}} r_1 \wedge \boxed{\boxed{(q_0, r_0), (q_1, r_1)}} \vdash_{\mathsf{d}} q_0 \preccurlyeq_{\mathsf{L}} r_0 \\
\text{By L-Cycle} \quad \Longleftarrow\ & (q_0, r_0) \in \{(q_0, r_0), (q_1, r_1)\} \wedge (q_1, r_1) \in \{(q_0, r_0), (q_1, r_1)\}
\end{aligned}
$$

4.4 Right-Biased Delay Simulation

In the definition of delay simulation, it is very important that the disjunct L-Step is *not* in the scope of the inner least fixed point. Otherwise, the relation does not provide a sound technique for language inclusion. Consider the following (incorrect) reformulation of delay simulation where the inner least fixed point operator has been pulled back to be at the same level as the outer greatest fixed point.

$$
\begin{aligned}
& \mathtt{wrong} \triangleq \nu X.\mu Y. \\
\text{(Final)} \quad & \{\ (s_1, s_2) \mid s_2 \in \mathcal{F}_2 \wedge \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'.\ (s_1', s_2') \in X\ \} \cup \\
\text{(Delay)} \quad & \{\ (s_1, s_2) \mid \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'.\ (s_1', s_2') \in Y\ \} \cup \\
\text{(Step)} \quad & \{\ (s_1, s_2) \mid s_1 \notin \mathcal{F}_1 \wedge \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'. (s_1, s_2) \in X\ \}
\end{aligned}
$$

This definition essentially merges $\texttt{fsim}^{\mathsf{L}}_{\texttt{delay}}$ and $\texttt{fsim}^{\mathsf{R}}_{\texttt{delay}}$ into a single coinductive-inductive predicate. Unfortunately, it is not too difficult to see that this attempt at "simplifying" $\texttt{fsim}_{\texttt{delay}}$ gives a notion of simulation that is unsound for language inclusion. As a counterexample, consider the following two automata:



Clearly, $\mathcal{L}(q_0) = \{a^\omega\}$ and it is not included in $\mathcal{L}(r_0) = \emptyset$. However, we can prove that $(q_0, r_0) \in \texttt{wrong}$ using $\{(q_0, r_0), (q_1, r_0)\}$ as a coinduction hypothesis. Indeed, from (q_0, r_0) it suffices to use Delay once to reach (q_1, r_0). From there, since q_1 is not final, we can use Step to cycle back to (q_0, r_0) and conclude, even though we never used Final to show that an accepting state can be reached in the right-hand automaton.

To fix this wrong definition, we can either remove the Delay disjunct, or the Step disjunct. Removing Delay gives back direct simulation. However, if we remove Step and keep Delay, we obtain the following *right-biased* notion of simulation (we note $\texttt{fsim}_{\texttt{rb}}$).

Definition 4.7 (Right-biased Simulation).

$$
\begin{aligned}
&\texttt{fsim}_{\texttt{rb}} \triangleq \nu X.\mu Y.\\
&\qquad\text{(Final)}\quad \{\ (s_1, s_2) \mid s_2 \in \mathcal{F}_2 \wedge \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'.\ (s_1', s_2') \in X\ \} \cup\\
&\qquad\text{(Delay)}\quad \{\ (s_1, s_2) \mid \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'.\ (s_1', s_2') \in Y\ \}
\end{aligned}
$$

We note that by removing the disjunct (Step), we lose the ability to exploit the fairness assumption of the left-hand automaton (hence the name *right-biased*). Indeed, (Step) allowed us to use the coinduction hypothesis after taking a computation step from a non-accepting left-state. This enables to use cyclic reasoning to ignore non-accepting loops in the left-hand automaton. Without (Step), we have no choice but to visit a right-accepting state in order to eventually get access to the coinduction hypothesis using (Final).

The relation $\texttt{fsim}_{\texttt{rb}}$ gives a sound technique for language inclusion. However, since it ignores the fairness condition of the left-hand automaton, it is more accurately phrased as a technique to guarantee that any *trace* of the left-hand automaton is in the language of the right-hand one.

Theorem 4.8. $(s_1, s_2) \in \texttt{fsim}_{\texttt{rb}} \implies \mathit{Traces}(s_1) \subseteq \mathcal{L}(s_2)$

Proof. (Sketch) Clearly, $\texttt{fsim}_{\texttt{rb}}$ is contained in $\texttt{sim}$ and therefore, it guarantees trace inclusion. Further, the disjunct (Final) enforces that any execution of the left-hand automaton is simulated by an execution of the right-hand automaton that steps through an $\mathcal{F}_2$-state infinitely many times. This guarantees that traces of s_1 are in the language of s_2. □

Interestingly, we observe that even without the ability to exploit fairness assumptions, $\texttt{fsim}_{\texttt{rb}}$ can still be used as a general method for interactive proofs of arbitrary specifications expressed as a Büchi automaton. In particular, this includes any specification expressed in Linear Temporal Logic [3].

As for the previous relations presented so far, $\mathtt{fsim}_{\mathtt{rb}}$ can be presented in the form of a deductive system. The two core rules are the following Final and Delay rules:

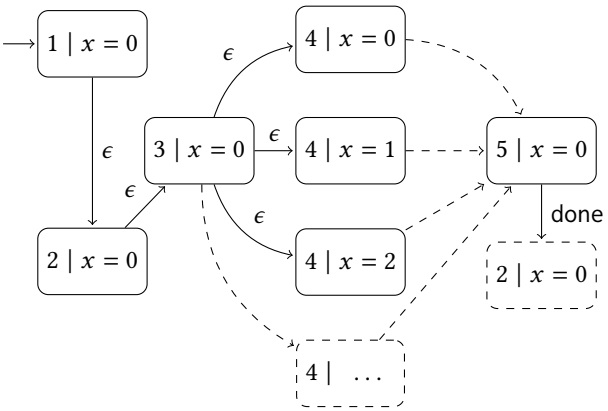
$$\textsc{Final}\quad \frac{s_2 \in \mathcal{F}_2 \qquad \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'. \boxed{\boxed{H}} \vdash_{\mathtt{rb}} s_1' \preccurlyeq s_2'}{\boxed{H} \vdash_{\mathtt{rb}} s_1 \preccurlyeq s_2} \qquad \textsc{Delay}\quad \frac{\forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'. \boxed{H} \vdash_{\mathtt{rb}} s_1' \preccurlyeq s_2'}{\boxed{H} \vdash_{\mathtt{rb}} s_1 \preccurlyeq s_2}$$

We note that while Final releases the guard around the context, Delay does not! Naturally, as in Figure 1 and Figure 2, additional rules Cycle, Guard, and Invariant can also be derived.

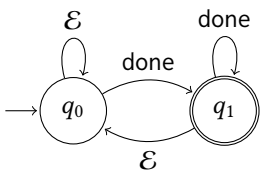
Example 8. As an example, consider the following program (top left), its representation as an infinite-state transition system (on the right), and its specification (bottom left):

Program:

(1) $x \leftarrow 0$
(2) **loop**:
(3) $\quad x \leftarrow n \in \mathbb{N}$
(4) $\quad$ **while**$(x > 0) : x \leftarrow x - 1$
(5) $\quad$ **print**(*"done"*)



Specification:



The program operates a single variable $x \in \mathbb{N}$, initialized to be 0. Then, it enters an infinite loop. At each iteration of this outer loop, a new value $n \in \mathbb{N}$ is determined. The program then counts down to 0 from n and prints the message "done". This example could model, for example, a reactive system continuously receiving user inputs (x), and processing it before providing an answer. The specification we wish to prove is that the program is *"done"* infinitely many times. In LTL notation, we want to prove $\square\diamond$ done. We give a proof using the rules of $\mathtt{fsim}_{\mathtt{rb}}$.

$$\begin{aligned} & \boxed{\emptyset} \vdash_{\mathtt{rb}} (1,0) \preccurlyeq q_0 \\ \text{By Delay (2 times)} \quad \Longleftarrow\ & \boxed{\emptyset} \vdash_{\mathtt{rb}} (3,0) \preccurlyeq q_0 \end{aligned}$$

We use the invariant $I = \{\ ((3,0), q_0)\ \}$.

$$\begin{aligned} \text{By Invariant} \quad \Longleftarrow\ & \boxed{\mathrm{I}} \vdash_{\mathtt{rb}} (3,0) \preccurlyeq q_0 \\ \text{By Delay and } x \leftarrow n \in \mathbb{N} \quad \Longleftarrow\ & \forall n \in \mathbb{N}.\ \boxed{\mathrm{I}} \vdash_{\mathtt{rb}} (4,n) \preccurlyeq q_0 \end{aligned}$$

From there, the proof goes by induction on $n \in \mathbb{N}$. The case of $n = 0$ is simple:

$$\begin{aligned}
& \boxed{I} \vdash_{\mathtt{rb}} (4,0) \preccurlyeq q_0 \\
\text{By Delay and since } \neg(0>0) \quad & \Longleftarrow \boxed{I} \vdash_{\mathtt{rb}} (5,0) \preccurlyeq q_0 \\
\text{By Delay and by printing } \textit{“done”} \quad & \Longleftarrow \boxed{I} \vdash_{\mathtt{rb}} (2,0) \preccurlyeq q_1 \\
\text{By Final (since } q_1 \text{ is final)} \quad & \Longleftarrow \boxed{\boxed{I}} \vdash_{\mathtt{rb}} (3,0) \preccurlyeq q_0 \\
\text{By Cycle} \quad & \Longleftarrow ((3,0), q_0) \in I
\end{aligned}$$

Finally, we cover the inductive case. By induction, we get to assume $\boxed{I} \vdash_{\mathtt{rb}} (4,n) \preccurlyeq q_0$, and we have to show $\boxed{I} \vdash_{\mathtt{rb}} (4,n+1) \preccurlyeq q_0$.

$$\begin{aligned}
& \boxed{I} \vdash_{\mathtt{rb}} (4,n+1) \preccurlyeq q_0 \\
\text{By Delay, } n+1>0 \text{ and } x \leftarrow x-1 \quad & \Longleftarrow \boxed{I} \vdash_{\mathtt{rb}} (4,n) \preccurlyeq q_0
\end{aligned}$$

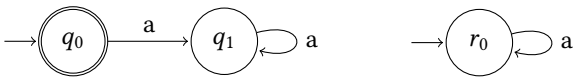
We can now conclude using our induction hypothesis. ◊

5 Simulations with Repeated Delay

5.1 The Problem of Spurious Left-Accepting States

In the previous section, we observed that for transition systems without fairness assumptions, a very simple notion of right-biased simulation is sufficient to prove liveness properties encoded as Büchi automata. For more challenging examples, we might however need to exploit fairness assumptions in order to prove that a liveness objective is accomplished. Direct simulation and delay simulation both have the ability to exploit fairness assumptions, but in a very restricted way. The main limitation of delay simulation is that *as soon* as an accepting state is visited in the left, it forces to visit an accepting state in the right at a later point. In particular, this means that a left automaton with unnecessarily many accepting states will be more difficult to deal with. As an extreme example of this phenomenon, let us consider the following example:

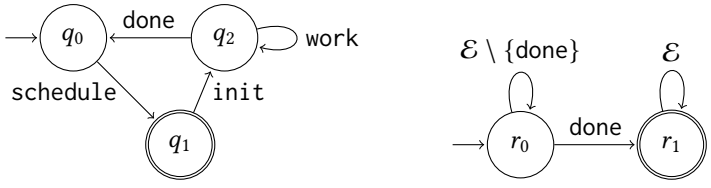
Example 9. Consider the following two automata.



Both have an empty language ($\mathcal{L}(q_0) = \mathcal{L}(r_0) = \emptyset$), and in principle, we would like to be able to prove that $\boxed{\emptyset} \vdash_{\mathtt{d}} q_0 \preccurlyeq_{\mathtt{L}} r_0$. Unfortunately, since q_0 is accepting, the only rule that can be applied initially is L-to-R and we have to prove $\boxed{\emptyset} \vdash_{\mathtt{d}} q_0 \preccurlyeq_{\mathtt{R}} r_0$. This of course not possible as the right automaton does not have any accepting state. ◊

This example is somewhat artificial. Indeed, since the left automaton does not have any word anyway, the accepting state q_0 is spurious and can be removed. Without it, we would have $(q_0, r_0) \in \mathtt{fsim}_{\mathtt{delay}}$, as expected. Unfortunately, this bad pattern can happen in practice with accepting states that cannot be removed because they belong to a cycle.

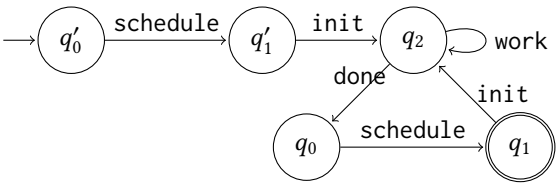
Example 10. Consider the following two automata:



The left automaton could model a scheduler which controls the execution of a program. The accepting state q_1 models two assumptions: the execution of the program is scheduled infinitely many times, and once the program execution is scheduled, it always terminates and hands the control back to the scheduler. With this interpretation in mind, the right automaton specifies that the program should be executed until completion at least once (i.e., the transition done needs to be taken). We note that here, the fact that q_1 is accepting is crucial.

It is easy to see that the left automaton is language included in the right one. Unfortunately, this fact cannot be established by delay simulation. Indeed, suppose we wish to prove $\boxed{\emptyset} \vdash_{\mathtt{d}} q_0 \preccurlyeq_{\mathtt{L}} r_0$. Since $q_0 \notin \mathcal{F}_1$, the best we can do is to use L-Step to go to $\boxed{\emptyset} \vdash_{\mathtt{d}} q_1 \preccurlyeq_{\mathtt{L}} r_0$. Now, since $q_1 \in \mathcal{F}_1$, we have no other choice than using L-to-R and it remains to prove $\boxed{\emptyset} \vdash_{\mathtt{d}} q_1 \preccurlyeq_{\mathtt{R}} r_0$. From there, we have to show that no matter which path out of q_1 is taken, we are guaranteed to reach a right-accepting state from r_0. Unfortunately, this is not the case! Indeed, we have to first go to $\boxed{\emptyset} \vdash_{\mathtt{d}} q_2 \preccurlyeq_{\mathtt{R}} r_0$ by R-Delay, but since there is a self-loop $q_2 \xrightarrow{\mathtt{work}} q_2$, the right-hand automaton is stuck in the non-accepting state r_0. ◊

Even though the previous example cannot be handled by $\mathtt{fsim}_{\mathtt{delay}}$ because $(q_2, r_0) \notin \mathtt{fsim}^{\mathtt{R}}_{\mathtt{delay}}$, we observe that $(q_2, r_0) \in \mathtt{fsim}^{\mathtt{L}}_{\mathtt{delay}}$! Further, since the Büchi acceptance condition requires executions to visit infinitely many times an accepting state, the first $n \in \mathbb{N}$ visits to a left-accepting state can always be ignored. Said otherwise, for this specific example, it would be correct to replace the left-hand automaton with the following one, where the two first steps have been explicitly unrolled, and the first visit to an accepting state is ignored:



Clearly, the unrolled version is language equivalent to the initial one. However, we can now prove $\boxed{\emptyset} \vdash q_0' \preccurlyeq_{\mathtt{L}} r_0$. It suffices to use two times L-Step to move to $\boxed{\emptyset} \vdash q_2 \preccurlyeq_{\mathtt{L}} r_0$. Since we have not encountered any left-accepting state on the way, we are still focusing on a L-triple. From there, it is not too difficult to conclude by using $\{(q_2, r_0), (q_0, r_1), (q_1, r_1), (q_2, r_1)\}$ as a coinduction hypothesis. Instead of having to explicitly modify the left transition system, the purpose of this section is to develop several new notions of delay simulation that have built-in support to emulate this type of reasoning.

5.2 Double Delay Simulation

In the previous example, we encountered the problem that delay simulation tracks *every* left-accepting state, and each time one is encountered, it forces us to prove that a right-accepting state can be later reached. Instead, we would like to be able to temporarily ignore left-accepting state, and *choose* when to start matching them with right-accepting states. So long as we only allow to ignore left-accepting states for a bounded number of computation steps, this still gives a sound proof technique for language inclusion.

A first idea to achieve this intuition is to prefix the greatest fixed point underlying the definition of $\mathtt{fsim}_{\mathtt{delay}}$ with an extra least fixed point allowing to skip the first n left-accepting states. We call the corresponding relation *double delay simulation* and define it as follows:

Definition 5.1 (Double Delay Simulation).

$$
\begin{aligned}
&\mathtt{wait}(X) \triangleq \mu Y.\\
&\quad\text{(Wait)}\quad \{\ (s_1, s_2) \mid \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'.\ (s_1', s_2') \in Y\ \} \cup\\
&\quad\text{(Commit)}\quad \{\ (s_1, s_2) \mid (s_1, s_2) \in X\ \}\\
&\mathtt{fsim}_{\mathtt{2delay}} \triangleq \mathtt{wait}(\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}})
\end{aligned}
$$

Concretely, to prove that $(s_1, s_2) \in \mathtt{fsim}_{\mathtt{2delay}}$ we can either decide to *commit* to track left-accepting state right away by proving $(s_1, s_2) \in \mathtt{fsim}_{\mathtt{delay}}$. Alternatively, we can also decide to *wait* for one computation step, and instead show that every successor s_1' of s_1 can be matched with a successor s_2' of s_2 such that (s_1', s_2') is again in $\mathtt{fsim}_{\mathtt{2delay}}$. Since `delay` is an inductive predicate, we can only wait for finitely many successive computation steps before having to switch to $\mathtt{fsim}_{\mathtt{delay}}$, which guarantees soundness for language inclusion.

Theorem 5.2 (Soundness). $(s_1, s_2) \in \mathtt{fsim}_{\mathtt{2delay}} \implies \mathcal{L}(s_1) \subseteq \mathcal{L}(s_2)$

Proof. By induction on the definition of `wait`, exploiting the soundness of $\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$ for the base case. □

We observe that $\mathtt{fsim}_{\mathtt{2delay}}$ is strictly weaker than $\mathtt{fsim}_{\mathtt{delay}}$. In particular, it is weak enough to cover the previous example.

Theorem 5.3. $\mathtt{fsim}_{\mathtt{2delay}}$ *is strictly weaker than* $\mathtt{fsim}_{\mathtt{delay}}$.

Proof. The fact that $\mathtt{fsim}_{\mathtt{delay}} \subseteq \mathtt{fsim}_{\mathtt{2delay}}$ immediately follows from the Commit disjunct in the definition of `delay`. It remains to show that there exists at least one pair of Büchi automata such that $\mathtt{fsim}_{\mathtt{2delay}}$ contains strictly more pairs of states than $\mathtt{fsim}_{\mathtt{delay}}$. For the two automata from Example 10, we already discussed that $(q_0, r_0) \notin \mathtt{fsim}_{\mathtt{delay}}$. However, we show that $(q_0, r_0) \in \mathtt{fsim}_{\mathtt{2delay}}$. We start by unfolding the underlying least fixed point `wait` twice (exploiting the first disjunct (Wait)) to get to $(q_2, r_0) \in \mathtt{fsim}_{\mathtt{2delay}}$. We then *commit* to prove that a right-accepting state can be reached by unfolding `wait` and exploiting the second disjunct (Commit). From there it suffices to show that $(q_2, r_0) \in \mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$. This can be done using the rules of Figure 2.

$$
\begin{aligned}
&\boxed{\emptyset} \vdash_{\mathsf{d}} q_2 \preccurlyeq_{\mathsf{L}} r_0\\
\text{By L-Invariant} \quad \Longleftarrow\quad &\boxed{(q_2, r_0)} \vdash_{\mathsf{d}} q_2 \preccurlyeq_{\mathsf{L}} r_0\\
\text{By L-Step} \quad \Longleftarrow\quad &\boxed{(q_2, r_0)} \vdash_{\mathsf{d}} q_0 \preccurlyeq_{\mathsf{L}} r_1 \wedge \boxed{(q_2, r_0)} \vdash_{\mathsf{d}} q_2 \preccurlyeq_{\mathsf{L}} r_0
\end{aligned}
$$

By L-Cycle, the second conjunct is trivially discharged, and it only remains to prove $\boxed{(q_2, r_0)} \vdash_{\mathsf{d}} q_0 \preccurlyeq_{\mathsf{L}} r_1$.

$$
\begin{aligned}
& \boxed{(q_2, r_0)} \vdash_{\mathsf{d}} q_0 \preccurlyeq_{\mathsf{L}} r_1 \\
\text{By L-Invariant} \quad \Longleftarrow \quad & \boxed{(q_2, r_0), (q_0, r_1)} \vdash_{\mathsf{d}} q_0 \preccurlyeq_{\mathsf{L}} r_1 \\
\text{By L-Step and } q_0 \notin \mathcal{F}_1 \quad \Longleftarrow \quad & \boxed{\boxed{(q_2, r_0), (q_0, r_1)}} \vdash_{\mathsf{d}} q_1 \preccurlyeq_{\mathsf{L}} r_1 \\
\text{By L-Invariant} \quad \Longleftarrow \quad & \boxed{(q_2, r_0), (q_0, r_1), (q_1, r_1)} \vdash_{\mathsf{d}} q_1 \preccurlyeq_{\mathsf{L}} r_1 \\
\text{By L-to-R and R-Delay} \quad \Longleftarrow \quad & \boxed{(q_2, r_0), (q_0, r_1), (q_1, r_1)} \vdash_{\mathsf{d}} q_2 \preccurlyeq_{\mathsf{R}} r_1 \\
\text{By R-Final and } r_1 \in \mathcal{F}_2 \quad \Longleftarrow \quad & \boxed{\boxed{(q_2, r_0), (q_0, r_1), (q_1, r_1)}} \vdash_{\mathsf{d}} q_2 \preccurlyeq_{\mathsf{L}} r_1 \wedge \\
& \boxed{\boxed{(q_2, r_0), (q_0, r_1), (q_1, r_1)}} \vdash_{\mathsf{d}} q_0 \preccurlyeq_{\mathsf{L}} r_1
\end{aligned}
$$

By L-Cycle, the right conjunct is discharged and it remains to show

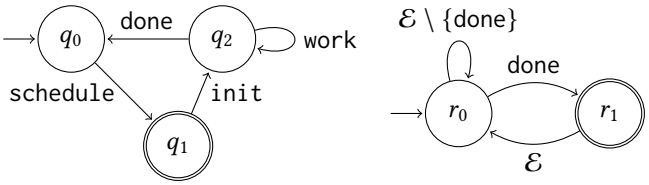
$$
\begin{aligned}
& \boxed{(q_2, r_0), (q_0, r_1), (q_1, r_1)} \vdash_{\mathsf{d}} q_2 \preccurlyeq_{\mathsf{L}} r_1 \\
\text{By L-Invariant} \quad \Longleftarrow \quad & \boxed{(q_2, r_0), (q_0, r_1), (q_1, r_1), (q_2, r_1)} \vdash_{\mathsf{d}} q_2 \preccurlyeq_{\mathsf{L}} r_1 \\
\text{By L-Step and } q_2 \notin \mathcal{F}_1 \quad \Longleftarrow \quad & \boxed{\boxed{(q_2, r_0), (q_0, r_1), (q_1, r_1), (q_2, r_1)}} \vdash_{\mathsf{d}} q_2 \preccurlyeq_{\mathsf{L}} r_1 \wedge \\
& \boxed{\boxed{(q_2, r_0), (q_0, r_1), (q_1, r_1), (q_2, r_1)}} \vdash_{\mathsf{d}} q_0 \preccurlyeq_{\mathsf{L}} r_1
\end{aligned}
$$

And by L-Cycle we are done since both (q_2, r_1) and (q_0, r_1) are part of the coinduction hypothesis. □

5.3 Repeated Delay

With $\mathtt{fsim}_{\mathtt{2delay}}$, left-accepting states can only be ignored at the beginning of a proof. Once we commit to track left-accepting states, we cannot use Wait anymore, and every subsequent visit to a left-accepting state will force us to prove that a right-accepting state can be reached. For the previous example, this was not an issue as it was enough to unroll the first two steps of the left automaton, skipping unnecessary $\mathcal{F}_1$-states until we are positioned right before the last event allowing us to conclude that our liveness goal is fulfilled. In general, this style of reasoning can be applied for reachability/termination specifications where, once an objective has been attained once, the left-automata is unconstrained. For richer specifications, this strategy does not always work.

Example 11 (Double Delay Simulation is still too strong). Consider the following example. The left automaton is the same as in Example 10, but instead of requiring for the event done to be produced at least once, we require to be done infinitely many times. Concretely, the specification automaton now has a back edge forcing to transition from r_1 back to r_0.



With this modification of the specification, the right automaton does not double delay simulate the left one (i.e., $(q_0, r_0) \notin \mathtt{fsim}_{\mathtt{2delay}}$). We can use the double delay trick once to reach (q_0, r_1),

but then the specification automaton cycles back to r_0 and forces us to prove $(q_1, r_0) \in \mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$. At this point, we cannot use Wait anymore, and since $q_1 \in \mathcal{F}_1$, we have to prove that r_1 can be reached again using only the rules of $\mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}$. As discussed in the previous section, this is not possible because of the self loop in q_2. ◊

Fortunately, there is *a priori* no reason to restrict ourselves to using the `wait` trick only at the beginning of a proof. Instead, every time a right-accepting state has been visited, it is correct to again temporarily ignore further visits to a left-accepting state. To concretize this intuition, we define the following *repeated delay simulation* (we note $\mathtt{fsim}_{\mathtt{rdelay}}$).

Definition 5.4 (Repeated Delay Simulation).

$$\mathtt{fsim}^{\mathsf{R}}_{\mathtt{rdelay}}(X) \triangleq \mu Y.$$

$$\text{(R-Final)} \quad \{\ (s_1, s_2) \mid s_2 \in \mathcal{F}_2 \wedge \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'.\ (s_1', s_2') \in \mathtt{fsim}^{\mathsf{W}}_{\mathtt{rdelay}}(X)\ \} \cup$$

$$\text{(R-Delay)} \quad \{\ (s_1, s_2) \mid \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'.\ (s_1', s_2') \in Y\ \}$$

$$\mathtt{fsim}^{\mathsf{L}}_{\mathtt{rdelay}} \triangleq \nu X.$$

$$\text{(L-to-R)} \quad \{\ (s_1, s_2) \mid (s_1, s_2) \in \mathtt{fsim}^{\mathsf{R}}_{\mathtt{rdelay}}(X)\ \} \cup$$

$$\text{(L-Step)} \quad \{\ (s_1, s_2) \mid s_1 \notin \mathcal{F}_1 \wedge \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'.\ (s_1', s_2') \in X\ \}$$

$$\mathtt{fsim}^{\mathsf{W}}_{\mathtt{rdelay}}(X) \triangleq \mu Y.$$

$$\text{(W-Wait)} \quad \{\ (s_1, s_2) \mid \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'.\ (s_1', s_2') \in Y\ \} \cup$$

$$\text{(W-Commit)} \quad \{\ (s_1, s_2) \mid (s_1, s_2) \in X\ \}$$

$$\mathtt{fsim}_{\mathtt{rdelay}} \triangleq \mathtt{fsim}^{\mathsf{W}}_{\mathtt{rdelay}}(\mathtt{fsim}^{\mathsf{L}}_{\mathtt{rdelay}})$$

The intuition behind this definition is the following: Initially, we work with $\mathtt{fsim}^{\mathsf{W}}_{\mathtt{rdelay}}$, which gives access to the *unrolling rules* W-Wait and W-Commit, allowing us to ignore finitely many left-accepting states (same as `wait`). Whenever we desire, we can commit to track left-accepting state and instead switch to the coinductive relation $\mathtt{fsim}^{\mathsf{L}}_{\mathtt{rdelay}}$ (similar to $\mathtt{fsim}^{\mathsf{L}}_{\mathtt{delay}}$). As soon as a left-accepting state is encountered while using $\mathtt{fsim}^{\mathsf{L}}_{\mathtt{rdelay}}$, the control is transferred to the inductive relation $\mathtt{fsim}^{\mathsf{R}}_{\mathtt{rdelay}}$ (similar to $\mathtt{fsim}^{\mathsf{R}}_{\mathtt{delay}}$), which forces us to reach a right-accepting state in finitely many steps. Once a right-accepting state is found, instead of cycling back to $\mathtt{fsim}^{\mathsf{L}}_{\mathtt{rdelay}}$ (as in $\mathtt{fsim}_{\mathtt{delay}}$), we instead cycle back to $\mathtt{fsim}^{\mathsf{W}}_{\mathtt{rdelay}}$.

Theorem 5.5 (Soundness). $(s_1, s_2) \in \mathtt{fsim}_{\mathtt{rdelay}} \implies \mathcal{L}(s_1) \subseteq \mathcal{L}(s_2)$

Proof. (Sketch) The proof proceeds by coinduction on $\mathcal{L}(s_2)$ and by induction on the inductive part of $\mathtt{fsim}^{\mathsf{W}}_{\mathtt{rdelay}}$, the inductive part of $\mathcal{L}(s_1)$ and the inductive definition of $\mathtt{fsim}^{\mathsf{R}}_{\mathtt{rdelay}}$ (in this precise order). The technical details are rather tedious, and we refer the curious readers to our Rocq development for a complete proof. □

As for all the simulation relations presented so far, $\mathtt{fsim}_{\mathtt{rdelay}}$ can be presented as a deductive system. We omit the exact definition of the triples associated with $\mathtt{fsim}_{\mathtt{rdelay}}$. Instead, we directly present (a selection) of reasoning rules in Figure 3.

$$\textsc{L-to-R}\ \frac{\boxed{H} \vdash_{\mathtt{rd}} s_1 \preccurlyeq_{\mathsf{R}} s_2}{\boxed{H} \vdash_{\mathtt{rd}} s_1 \preccurlyeq_{\mathsf{L}} s_2} \qquad \textsc{L-Step}\ \frac{s_1 \notin \mathcal{F}_1 \qquad \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'. \boxed{\boxed{H}} \vdash_{\mathtt{rd}} s_1' \preccurlyeq_{\mathsf{L}} s_2'}{\boxed{H} \vdash_{\mathtt{rd}} s_1 \preccurlyeq_{\mathsf{L}} s_2}$$

$$\textsc{R-Delay}\ \frac{\forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'. \boxed{H} \vdash_{\mathtt{rd}} s_1' \preccurlyeq_{\mathsf{R}} s_2'}{\boxed{H} \vdash_{\mathtt{rd}} s_1 \preccurlyeq_{\mathsf{R}} s_2} \qquad \textsc{R-Final}\ \frac{s_2 \in \mathcal{F}_2 \qquad \forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'. \boxed{H} \vdash_{\mathtt{rd}} s_1' \preccurlyeq_{\mathsf{W}} s_2'}{\boxed{H} \vdash_{\mathtt{rd}} s_1 \preccurlyeq_{\mathsf{R}} s_2}$$

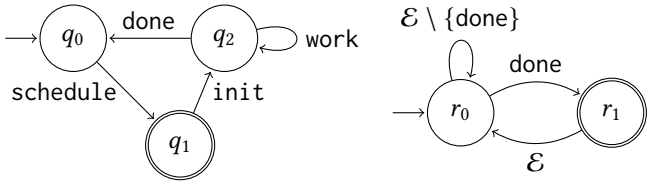
$$\textsc{W-Wait}\ \frac{\forall e. \forall s_1 \xrightarrow{e} s_1'. \exists s_2 \xrightarrow{e} s_2'. \boxed{H} \vdash_{\mathtt{rd}} s_1' \preccurlyeq_{\mathsf{W}} s_2'}{\boxed{H} \vdash_{\mathtt{rd}} s_1 \preccurlyeq_{\mathsf{W}} s_2} \qquad \textsc{W-Commit}\ \frac{\boxed{\boxed{H}} \vdash_{\mathtt{rd}} s_1 \preccurlyeq_{\mathsf{L}} s_2}{\boxed{H} \vdash_{\mathtt{rd}} s_1 \preccurlyeq_{\mathsf{W}} s_2}$$

$$\textsc{L-Cycle}\ \frac{(s_1, s_2) \in H}{\boxed{\boxed{H}} \vdash_{\mathtt{rd}} s_1 \preccurlyeq_{\mathsf{L}} s_2} \qquad \textsc{L-Guard}\ \frac{\boxed{H} \vdash_{\mathtt{rd}} s_1 \preccurlyeq_{\mathsf{L}} s_2}{\boxed{\boxed{H}} \vdash_{\mathtt{rd}} s_1 \preccurlyeq_{\mathsf{L}} s_2} \qquad \textsc{L-Invariant}\ \frac{(s_1, s_2) \in H' \qquad \forall (s_1', s_2') \in H'. \boxed{H \cup H'} \vdash_{\mathtt{rd}} s_1' \preccurlyeq_{\mathsf{L}} s_2'}{\boxed{H} \vdash_{\mathtt{rd}} s_1 \preccurlyeq_{\mathsf{L}} s_2}$$

Fig. 3. Selection of proof rules for $\mathtt{fsim}_{\mathtt{rdelay}}$

We showcase the use of $\mathtt{fsim}_{\mathtt{rdelay}}$ by applying the rules of Figure 3 to the example of the beginning of this section (which was neither covered by $\mathtt{fsim}_{\mathtt{delay}}$ nor by $\mathtt{fsim}_{\mathtt{2delay}}$).

Example 12. Recall the following two automata from Example 11:



We want to show $\boxed{\emptyset} \vdash_{\mathtt{rd}} q_0 \preccurlyeq_{\mathsf{W}} r_0$. We start by applying the W-Wait rule twice.

$$\boxed{\emptyset} \vdash_{\mathtt{rd}} q_0 \preccurlyeq_{\mathsf{W}} r_0$$
$$\text{By W-Wait} \impliedby \boxed{\emptyset} \vdash_{\mathtt{rd}} q_1 \preccurlyeq_{\mathsf{W}} r_0$$
$$\text{By W-Wait} \impliedby \boxed{\emptyset} \vdash_{\mathtt{rd}} q_2 \preccurlyeq_{\mathsf{W}} r_0$$

Then, we are in (q_2, r_0). From there, we commit and accumulate.

$$\text{By W-Commit and L-Guard} \impliedby \boxed{\emptyset} \vdash_{\mathtt{rd}} q_2 \preccurlyeq_{\mathsf{L}} r_0$$
$$\text{By L-Invariant} \impliedby \boxed{(q_2, r_0)} \vdash_{\mathtt{rd}} q_2 \preccurlyeq_{\mathsf{L}} r_0$$

Since q_2 is not final, we can apply the L-Step rule. However, because q_2 has two outgoing transitions, our goal splits into two goals. The first transition stays in (q_2, r_0) and thus we can conclude directly.

$$\text{By L-Step} \impliedby \boxed{\boxed{(q_2, r_0)}} \vdash_{\mathtt{rd}} q_2 \preccurlyeq_{\mathsf{L}} r_0$$
$$\text{By L-Cycle} \impliedby (q_2, r_0) \in \{(q_2, r_0)\}$$

The second transition goes to (q_0, r_1). As r_1 is final, we can apply the L-to-R rule followed by R-Final to conclude.

$$\text{By L-Step and L-Guard} \impliedby \boxed{(q_2, r_0)} \vdash_{\mathtt{rd}} q_0 \preccurlyeq_{\mathtt{L}} r_1$$
$$\text{By L-to-R} \impliedby \boxed{(q_2, r_0)} \vdash_{\mathtt{rd}} q_0 \preccurlyeq_{\mathtt{R}} r_1$$
$$\text{By R-Final} \impliedby \boxed{(q_2, r_0)} \vdash_{\mathtt{rd}} q_1 \preccurlyeq_{\mathtt{W}} r_0$$

Thus, we are now in (q_1, r_0) and back in the wait relation. We apply once the W-Wait rule to get to (q_2, r_0). Then, we commit and can directly conclude as we already have this state in our relation.

$$\text{By W-Wait} \impliedby \boxed{(q_2, r_0)} \vdash_{\mathtt{rd}} q_2 \preccurlyeq_{\mathtt{W}} r_0$$
$$\text{By W-Commit} \impliedby \boxed{\boxed{(q_2, r_0)}} \vdash_{\mathtt{rd}} q_2 \preccurlyeq_{\mathtt{L}} r_0$$
$$\text{By L-Cycle} \impliedby (q_2, r_0) \in \{(q_2, r_0)\}$$

◊

6 Related Work

6.1 Fairness Preserving Simulations for Büchi Automata

There is a large body of literature studying fairness-preserving notions of simulation for Büchi automata [2, 8, 10, 11, 13]. Grumberg and Long introduced a first notion of simulation for transition systems augmented with a fairness condition [10]. To be precise, their definition covers transition systems equipped with a Street condition (note that such transition systems can be translated to Büchi automata, and vice versa). Aziz et al. [2] later extended the notion of simulation presented by Grumberg and Long [10] into several alternative notions of bisimulation for transition systems equipped with a Muller fairness condition (again, these can be translated to Büchi automata, and vice versa). However, as later observed by Henzinger et al. [11], the definitions of Grumberg and Long [10] and Aziz et al. [2] are not *local*: both refer to fragments of executions instead of only referring to states and transitions. Consequently, these notions of fairness-preserving simulation are computationally expensive to automate, and they are also not well-suited for interactive proofs. Henzinger et al. [11] introduced *Fair Simulation* to overcome some of these limitations. Contrary to the definitions of Grumberg and Long [10] and Aziz et al. [2], the notion of fair simulation introduced by Henzinger et al. [11] is more local. In particular, there are efficient algorithms to automatically prove fair simulation. Nonetheless, applications of fair simulation to interactive proofs are not investigated by Henzinger et al. [11]. Further, to the best of our knowledge, none of the simulation techniques listed above have been mechanized in an interactive proof assistant.

6.2 Interactive Proofs using Simulation Relations

Over the course of the last decade, a variety of techniques and tools to integrate simulation techniques into interactive proof assistants have been developed. For example, our paper builds on the framework of *parameterized coinduction* introduced by Hur et al. [12] as a systematic methodology to develop interactive proofs by coinduction. Parameterized coinduction has been successfully applied to a wide range of problems including behavioral inclusion of reactive and impure programs modeled as interaction trees [18]. It has also been used to develop powerful deductive systems to reason about weak notions of stream equivalence [19].

Another notable application of simulation techniques to interactive verification in a proof assistant is the Simuliris framework [9]. Simuliris combines a simulation relation with the mechanized concurrent separation logic Iris [14] in order to verify the correctness of optimizations for concurrent programs. At its core, Simuliris features a *fair termination preserving* simulation relation: it

ensures that terminating behaviors of the left program can be reproduced by the right program, and it further allows the left program to have diverging behaviors, so long as the right program can mimic these diverging behaviors via a fair execution. However, Simuliris cannot exploit fairness assumptions on the left program, nor does it support reasoning about more general liveness properties (other than termination/divergence).

The recently developed notion of *freely stuttering simulation* (FreeSim) [4] is also related to our work. The core idea of FreeSim is to simplify interactive proofs of behavioral inclusion by simulation in the presence of *stutter steps* (i.e., computation steps that do not produce events). In this setting, the right-hand system is typically allowed to stutter for finitely many computation steps until, eventually, synchronous progress is made and both the left-hand and the right-hand systems are emitting the same event (at which point the coinduction hypothesis can be used to close cycles). FreeSim proposes a systematic approach to further weaken this principle by instead allowing for *asynchronous* progress to be made, effectively enabling (under additional constraints) to exploit the coinduction hypothesis after a stutter step in the right-hand system. Techniques developed by Cho et al. [4] are similar in spirit to the techniques we employ in our notions of fairness-preserving simulation with repeated delay. We note, however, that FreeSim does not handle behavioral inclusions in the presence of fairness assumptions/requirements.

A more recent approach that is connected to ours is the notion of *Fair Operational Semantics* (FOS) introduced by Lee et al. [16]. In essence, FOS proposes to model fairness assumptions of (concurrent) programs as an operational semantics. This is achieved by labelling the operational semantics of a programming language with specific events. As part of this framework, the authors show that it is possible to develop notions of simulation to prove behavioral inclusions in the presence of fairness assumptions modeled by FOS [16]. We note however that, as stated by the authors themselves, the notion of fairness supported by FOS differs in several subtle aspects with other notions of fairness that are commonly used in the automata-based model checking literature [16]. In particular, it is not yet clear whether arbitrary liveness specifications (and in particular, arbitrary LTL specifications) can easily be encoded using FOS. In contrast, the notions of simulation we developed in this paper apply to arbitrary Büchi automata, and by extension to any omega-regular specification (including in particular, LTL-definable specifications).

7 Conclusion and Future Work

In this paper, we re-investigated, from the point of view of interactive proofs, several notions of fairness-preserving simulations that are typically employed in the context of automata-based model-checking. Starting from *direct simulation*, arguably the simplest notion of fairness-preserving simulation, we progressively constructed several improvements allowing to handle increasingly complex examples. Along the way, we demonstrated that by combining carefully designed fairness-preserving simulation relations with parameterized coinduction, we can obtain simple deductive systems to prove fair trace inclusion between transition systems modelling programs and Büchi automata modelling temporal specifications. We believe that this idea deserves further investigation. In particular, it could be used as a basis to develop a fully-featured mechanized program logic for temporal reasoning.

Data-Availability Statement

All notions of simulation presented throughout the paper, their proofs of soundness, as well as all examples and counterexamples have been formalized in the Rocq proof assistant. The sources and their documentation are available at the following address [7]:

https://zenodo.org/records/15658188

Acknowledgements

This work was supported by the European Research Council (ERC) Grant HYPER (No. 101055412). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. A. Correnson and I. Kuhn carried out this work as members of the Saarbrücken Graduate School of Computer Science.

References

[1] Martin Abadi and Leslie Lamport. 1991. The existence of refinement mappings. *Theoretical Computer Science* 82, 2 (1991), 253–284. doi:10.1016/0304-3975(91)90224-P

[2] Adnan Aziz, Vigyan Singhal, Felice Balarin, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. 1994. Equivalences for fair Kripke structures. In *Automata, Languages and Programming*, Serge Abiteboul and Eli Shamir (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 364–375.

[3] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of model checking*. MIT Press, 249–253.

[4] Minki Cho, Youngju Song, Dongjae Lee, Lennard Gäher, and Derek Dreyer. 2023. Stuttering for Free. *Proc. ACM Program. Lang.* 7, OOPSLA2, Article 281 (Oct. 2023), 28 pages. doi:10.1145/3622857

[5] Lorenzo Clemente. 2011. Büchi automata can have smaller quotients. In *International Colloquium on Automata, Languages, and Programming*. Springer, 258–270.

[6] Arthur Correnson and Bernd Finkbeiner. 2025. Coinductive Proofs for Temporal Hyperliveness. *Proc. ACM Program. Lang.* 9, POPL, Article 53 (Jan. 2025), 28 pages. doi:10.1145/3704889

[7] Arthur Correnson and Iona Kuhn. 2025. *Artifact for Almost Fair Simulations*. doi:10.5281/zenodo.15658187

[8] Kousha Etessami, Thomas Wilke, and Rebecca A. Schuller. 2001. Fair Simulation Relations, Parity Games, and State Space Reduction for Büchi Automata. In *Automata, Languages and Programming*, Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 694–707.

[9] Lennard Gäher, Michael Sammler, Simon Spies, Ralf Jung, Hoang-Hai Dang, Robbert Krebbers, Jeehoon Kang, and Derek Dreyer. 2022. Simuliris: a separation logic framework for verifying concurrent program optimizations. *Proc. ACM Program. Lang.* 6, POPL, Article 28 (Jan. 2022), 31 pages. doi:10.1145/3498689

[10] Orna Grumberg and David E. Long. 1994. Model checking and modular verification. *ACM Trans. Program. Lang. Syst.* 16, 3 (May 1994), 843–871. doi:10.1145/177492.177725

[11] Thomas A. Henzinger, Orna Kupferman, and Sriram K. Rajamani. 1997. Fair simulation. In *CONCUR '97: Concurrency Theory*, Antoni Mazurkiewicz and Józef Winkowski (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 273–287.

[12] Chung-Kil Hur, Georg Neis, Derek Dreyer, and Viktor Vafeiadis. 2013. The power of parameterization in coinductive proof. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Rome, Italy) *(POPL '13)*. Association for Computing Machinery, New York, NY, USA, 193–206. doi:10.1145/2429069.2429093

[13] Milka Hutagalung, Martin Lange, and Etienne Lozes. 2013. Revealing vs. Concealing: More Simulation Games for Büchi Inclusion. In *Language and Automata Theory and Applications*, Adrian-Horia Dediu, Carlos Martín-Vide, and Bianca Truthe (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 347–358.

[14] Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer. 2015. Iris: Monoids and Invariants as an Orthogonal Basis for Concurrent Reasoning. *SIGPLAN Not.* 50, 1 (Jan. 2015), 637–650. doi:10.1145/2775051.2676980

[15] Leslie Lamport and Stephan Merz. 2022. Prophecy Made Simple. *ACM Trans. Program. Lang. Syst.* 44, 2, Article 6 (April 2022), 27 pages. doi:10.1145/3492545

[16] Dongjae Lee, Minki Cho, Jinwoo Kim, Soonwon Moon, Youngju Song, and Chung-Kil Hur. 2023. Fair Operational Semantics. *Proc. ACM Program. Lang.* 7, PLDI, Article 139 (June 2023), 24 pages. doi:10.1145/3591253

[17] Alfred Tarski. 1955. A lattice-theoretical fixpoint theorem and its applications. (1955).

[18] Li-yao Xia, Yannick Zakowski, Paul He, Chung-Kil Hur, Gregory Malecha, Benjamin C. Pierce, and Steve Zdancewic. 2019. Interaction trees: representing recursive and impure programs in Coq. *Proc. ACM Program. Lang.* 4, POPL, Article 51 (Dec. 2019), 32 pages. doi:10.1145/3371119

[19] Yannick Zakowski, Paul He, Chung-Kil Hur, and Steve Zdancewic. 2020. An equational theory for weak bisimulation via generalized parameterized coinduction. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs* (New Orleans, LA, USA) *(CPP 2020)*. Association for Computing Machinery, New York, NY, USA, 71–84. doi:10.1145/3372885.3373813

Received 2025-02-27; accepted 2025-06-27