Completing Almost Fair Simulations

Arthur Correnson
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Iona Kuhn
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Bernd Finkbeiner
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Abstract

The paper *Almost Fair Simulations* recently introduced a collection of deductive systems for interactive proofs of language inclusion between Büchi automata. These deductive systems enable intuitive proofs via cyclic reasoning principles, but are unfortunately incomplete for *fair similarity*, a standard notion of refinement for Büchi automata. In this paper, we address this shortcoming by presenting a new deductive system for language inclusion of Büchi automata that preserves the simplicity of *Almost Fair Simulations*, with the additional benefit of being complete for fair similarity. We mechanized the soundness and the completeness proofs of our new system in the Rocq proof assistant. The proofs rely on a new technique we call *nested parameterized coinduction*, an adaptation of Hur's et al. *parameterized coinduction* for the difficult case of proofs by coinduction-induction-coinduction.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Theory of computation → Logic and verification

Keywords and phrases Fair Simulation, Deductive Systems, Interactive Proof Assistants, Coinduction, Language Containment

Digital Object Identifier 10.4230/LIPIcs.ITP.2026.20

Supplementary Material *Software*: `https://doi.org/10.5281/zenodo.20322554`

Funding 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.

Acknowledgements We thank the anonymous ITP reviewers for their useful and detailed feedback, which guided the writing of the final version of this paper. A. Correnson and I. Kuhn carried out this work as members of the Saarbrücken Graduate School of Computer Science.

Declaration about GenAI/LLM use GenAI (LLMs) was not used to produce code, definitions, theorems or proofs in this paper.

1 Introduction

Fair simulations are a convenient technique for proving that a system equipped with a *fairness condition* (e.g., a concurrent program with a fair scheduler) is a *fair refinement* of another one (i.e., fair executions of a system can be simulated by fair executions of another one). While originally developed for the purpose of algorithmic verification, fair simulations have recently been increasingly used as a basis for designing mechanized program logics and deductive proof systems. The underlying motivation for this "repurposing" is twofold: First, verifying *liveness* properties of programs – a task that is pushing mechanized program logics to their limits – can be elegantly rephrased as a fair refinement problem [13, 12, 20]. Further, simulation proofs are relatively simple to develop in a proof assistant. In particular, since

(fair) similarity relations are typically defined as greatest fixpoints, one can rely on powerful techniques such as parameterized coinduction [11] to prove that a system (fairly) simulates another one. This approach has proven to work well for *strong* notions of (fair) similarity relations of the form $\nu\mu$ (i.e., coinductive-inductive relations) [8, 12, 5]. Unfortunately, weaker notions of fair similarity are typically characterized by predicates of the form $\nu\mu\nu$ (i.e., coinductive-inductive-coinductive relations). For these, proofs remain particularly tedious to mechanize in an interactive theorem prover.

An example of $\nu\mu\nu$-type similarity relation is fair similarity for Büchi automata [10], which we just call *fair similarity* from now on. *Almost Fair Simulations*[1] were introduced by Correnson et al. [5] as a collection of $\nu\mu$-type strengthening of fair similarity, specifically designed to avoid tedious coinductive-inductive-coinductive proofs. Thanks to the simpler fixpoint structure, the authors were able to obtain simple deductive systems based on *parameterized coinduction* [11]. Concretely, their deductive systems enable the incremental discovery of a (fair) simulation between two automata via cyclic reasoning principles. These principles enable concise and understandable proofs, but the trade-off is that dropping the last ν in $\nu\mu\nu$ renders the systems incomplete for fair similarity. This raises a question: Can we actually keep the simple cyclic reasoning principles, but achieve completeness for fair similarity?

Contributions

In this paper, we extend *Almost Fair Simulations* [5] by introducing a sound and complete deductive proof system for fair similarity of Büchi automata. Our system supports cyclic reasoning and incremental construction of a simulation via *nested parameterized coinduction*, an adaption of parameterized coinduction [11] for proofs by coinduction-induction-coinduction. In summary, we present the following contributions:

1. We present a sound and complete deductive system for fair similarity of Büchi automata. The system and its proof are mechanized in the Rocq proof assistant. We also formally proved that this deductive system is strictly more powerful than the ones presented in *Almost Fair Simulations* [5].
2. To justify the soundness and completeness of our deductive system, we introduce *nested parameterized coinduction* (nested paco, for short), an adaption of parameterized coinduction specifically tailored for proofs of inclusions in $\nu\mu\nu$-type predicates. Such proofs typically require constantly unfolding and refolding nested fixpoints, which can be particularly tedious in a proof assistant. Instead, nested paco hides these tedious fixpoint manipulations behind a simple fixpoint-free interface. While originally developed specifically for the soundness proof of our deductive system, we show that nested paco constitutes a sound and complete technique for proving inclusions in a large class of coinductive-inductive-coinductive predicates.
3. We provide a Rocq proof showing that fair similarity (and hence our deductive system) is sound for Büchi language inclusion. We note that, in the literature, fair similarity is usually defined in game-theoretic terms, and the soundness for language inclusion emerges as an immediate consequence of the definition. In this paper, obtaining soundness is less immediate because we start from a fixpoint-theoretic definition of fair similarity.

[1] We use the term "Almost Fair Simulations" to refer to the collection of simulation techniques (and their associated deductive systems) presented in the paper with the same name [5].

2 Preliminaries

Before presenting our deductive system and its properties, we briefly recall some necessary preliminaries on fixpoints, proofs by (co)induction, and Büchi automata.

Fixpoints and Proofs by Induction/Coinduction

Let U be a set and consider the usual lattice $(\mathcal{P}(U), \subseteq)$ of subsets of U ordered by set inclusion. Let further $F : \mathcal{P}(U) \to \mathcal{P}(U)$ be a function from subsets of U to subsets of U. We say that F is monotone if $\forall X, Y \in \mathcal{P}(U).\, X \subseteq Y \implies F(X) \subseteq F(Y)$, and we say that a subset $X \subseteq U$ is a *fixpoint* of F if $X = F(X)$. A result due to Tarski [19] guarantees that if F is monotone, it has both a *least* fixpoint μF and a *greatest* fixpoint νF. Further, it can be shown that $\mu F = \bigcap\{\, X \mid F(X) \subseteq X\}$ and $\nu F = \bigcup\{\, X \mid X \subseteq F(X)\}$. As a direct consequence of these two equations, we obtain powerful reasoning principles to prove inclusions of the form $\mu F \subseteq X$ (induction), and $X \subseteq \nu F$ (coinduction).

▶ **Theorem 1** (Proofs by Induction/Coinduction).

$$\text{(Coinduction)} \quad X \subseteq \nu F \iff \exists H.\, X \subseteq H \wedge H \subseteq F(H)$$
$$\text{(Induction)} \quad \mu F \subseteq X \iff \exists H.\, H \subseteq X \wedge F(H) \subseteq H$$

These two principles can be viewed as techniques to prove inclusions by exhibiting a suitable "invariant" H, usually called a *coinduction hypothesis* (in proofs by coinduction) or an *induction hypothesis* (in proofs by induction). Throughout the paper, we often use specific instances of Theorem 1 with $H = X$ as a (co)induction hypothesis.

$$\text{(direct coinduction)}\ X \subseteq F(X) \Rightarrow X \subseteq \nu F \qquad \text{(direct induction)}\ F(X) \subseteq X \Rightarrow \mu F \subseteq X$$

$$\text{(strong induction)} \quad F(X \cap \mu F(X)) \subseteq X \iff \mu F \subseteq X$$

In the remainder of the paper, we often write $\nu X.\, F(X)$ instead of $\nu(\lambda X.\, F(X))$ to inline fixpoint definitions. We also use the terms *fixpoint unfolding* and *fixpoint folding* to refer to the inclusions $\nu F \subseteq F(\nu F)$ and $F(\nu F) \subseteq \nu F$ respectively (both are immediate consequences of νF being a fixpoint of F). We use the same conventions for least fixpoints.

Büchi Automata

▶ **Definition 2** (Büchi Automata). *A non-deterministic Büchi automaton (NBA) is a tuple $(S, \Sigma, F, \to)$, where S is a set of states, Σ is an alphabet of symbols, $F \subseteq S$ is a set of accepting states, and $\to \subseteq S \times \Sigma \times S$ is a transition relation.*

For the remainder of this section, let $A = (S, \Sigma, F, \to)$ be a NBA. Given two states $s, s' \in S$ and a letter $\sigma \in \Sigma$, we write $s \xrightarrow{\sigma} s'$ instead of $(s, \sigma, s') \in \to$ to indicate that s can transition to s' after reading σ. An infinite word $w \in \Sigma^\omega$ is said to be in the *language* of a state $s \in S$ if there exists an infinite run on w that visits the accepting set F infinitely many times. That is, if there exists an infinite sequence

$$s \xrightarrow{w(0)} s_1 \xrightarrow{w(1)} s_2 \xrightarrow{w(2)} \dots$$

such that $s_i \in F$ for infinitely many i's. Such a sequence of transitions is called a *fair run* on w (also called *accepting run* in the literature). We formalize this intuition by defining a coinductive-inductive predicate $\texttt{Lang}_A \subseteq S \times \Sigma^\omega$. The intention is that $(s, w) \in \texttt{Lang}_A$ if and only if there is an accepting run on w from s.

▶ **Definition 3** (Language accepted by a state of a NBA).

$$\begin{aligned}\texttt{LangF}_A(L_\nu, L_\mu) \triangleq\ & \{\ (s, \sigma \cdot w) \mid \exists s' \in S.\ s \xrightarrow{\sigma} s' \wedge L_\mu(s', w)\ \} \cup \\ & \{\ (s, \sigma \cdot w) \mid s \in F \wedge \exists s' \in S.\ s \xrightarrow{\sigma} s' \wedge L_\nu(s', w)\ \} \\ \texttt{Lang}_A \triangleq\ & \nu L_\nu.\mu L_\mu.\ \texttt{LangF}_A(L_\nu, L_\mu)\end{aligned}$$

The intuition behind this nested fixpoint definition is the following. First, let us observe that by unfolding the greatest and the least fixpoint, we obtain the following equalities

$$\texttt{Lang}_A = \mu L_\mu.\texttt{LangF}_A(\texttt{Lang}_A, L_\mu) = \texttt{LangF}_A(\texttt{Lang}_A, \texttt{Lang}_A)$$

By definition of $\texttt{LangF}_A$, we therefore have the following equivalence

$$(s, \sigma \cdot w) \in \texttt{Lang}_A \iff \exists s'.\ s \xrightarrow{\sigma} s' \wedge (s', w) \in \texttt{Lang}_A \tag{1}$$

By "iterating" the left-to-right implication from any pair $(s, w) \in \texttt{Lang}_A$, we obtain a run on w from s. It remains to explain why this run is necessarily fair. The subtlety is that, in the definition of $\texttt{Lang}_A$, transitions from a non-accepting state are handled *inductively*. This ensures that for any $s \notin F$, and $(s, w) \in \texttt{Lang}_A$, the run generated by iterating (1) eventually visits an accepting state. This fact is easily proven by induction on the inductive part of $\texttt{Lang}_A$. This explains why $(s, w) \in \texttt{Lang}_A$ implies existence of an accepting run on w from s. The converse implication can be proved by coinduction on $\texttt{Lang}_A$, and using the right-to-left part of (1).

We note that NBAs are typically assumed to have finite state spaces and finite alphabets. Under these assumptions, there exist complete decision procedures for the language inclusion problem [2]. Since this paper is concerned with deductive systems and not algorithms, finiteness does not play a role, and we do not assume it. As a result, our deductive system can be used to reason about automata with an infinite state space and/or an infinite alphabet.

3 A Deductive System for Language Inclusion

3.1 Proving Language Inclusion by Simulation

In this paper, we are interested in proving language inclusion between NBAs. More specifically, we are interested in mechanizing proofs of language inclusion in an interactive proof assistant. For the reminder of this section, let us fix two automata $A_1 = (S_1, \Sigma, F_1, \to_1)$ and $A_2 = (S_2, \Sigma, F_2, \to_2)$ with the same alphabet Σ. We sometimes omit the subscripts 1 and 2 when they are clear from context.

Given a pair of states $s_1 \in S_1$ and $s_2 \in S_2$, we would like a simple state-based proof technique to check $\texttt{Lang}_{A_1}(s_1) \subseteq \texttt{Lang}_{A_2}(s_2)$. In order to prove $\texttt{Lang}_{A_1}(s_1) \subseteq \texttt{Lang}_{A_2}(s_2)$, we need to establish that every fair run of s_1 can be mimicked by a fair run of s_2. Omitting fairness for a moment, a standard state-based approach to prove that every run of s_1 can be mimicked by a run of s_2 would be to find a simulation relation $R \subseteq S_1 \times S_2$. Existence of a simulation relation is characterized by the following coinductive predicate `sim`.

▶ **Definition 4** (Strong Similarity).

$$\begin{aligned}\texttt{simF}(X) \triangleq\ & \{\ (s_1, s_2) \mid \forall \sigma. \forall s_1 \xrightarrow{\sigma} s_1'. \exists s_2 \xrightarrow{\sigma} s_2' \wedge (s_1', s_2') \in X\ \} \\ \texttt{sim} \triangleq\ & \nu\, \texttt{simF}\end{aligned}$$

Intuitively, $\texttt{sim}(s_1, s_2)$ if and only if for every transition $s_1 \xrightarrow{\sigma} s_1'$, one can find a transition $s_2 \xrightarrow{\sigma} s_2'$ such that $\texttt{sim}(s_1', s_2')$, again. The important result is that $\texttt{sim}(s_1, s_2)$ implies $\texttt{Traces}_{A_2}(s_1) \subseteq \texttt{Traces}_{A_1}(s_2)$ (where `Traces` is the same as `Lang` (see Definition 3), but without any requirement regarding visits to accepting states). Further, since `sim` is defined as a greatest fixpoint, $\texttt{sim}(s_1, s_2)$ can be proven by coinduction. In the Rocq proof assistant, one can even use *parameterized coinduction* [11], a powerful technique to streamline mechanized proofs by coinduction.

If we wanted to prove trace inclusion, `sim` would be a good starting point. Unfortunately it completely disregards the acceptance condition of Büchi automata. Consequently, it is not a sound technique for language inclusion for Büchi automata. Instead, we would need to strengthen `sim` to somewhat incorporate the requirement that *fair* runs from s_1 are simulated by *fair* runs from s_2. This intuitive idea was originally introduced by Grumberg and Long [9] and further explored by Henzinger et al. [10]. Later on, existence of a *fair simulation* between Büchi automata was shown to be equivalent to winning a certain *parity game* [7]. In this game, positions are pairs of states, and two players P_1 and P_2 take turns picking transitions in a pair of automata. Player P_2 wins the game from a position (s_1, s_2) if and only if there exists a fair simulation between s_1 and s_2. Importantly, a known result in game-theory is that the winning region of such parity games (i.e., the set of positions from which player P_2 wins) is determined by a nested fixpoint construction [21]. Applying this result to the specific case of fair similarity of Büchi automata yields the following coinductive-inductive-coinductive definition.

▶ **Definition 5** (Fair Similarity)**.**

$$
\begin{aligned}
\texttt{fairsim} = \nu X.\mu Y.\nu Z.\\
&\{\ (s_1, s_2) \mid s_2 \in F_2 \wedge \forall \sigma. \forall s_1 \xrightarrow{\sigma} s_1'. \exists s_2 \xrightarrow{\sigma} s_2'.\ (s_1', s_2') \in X\ \}\\
\cup\ &\{\ (s_1, s_2) \mid \forall \sigma. \forall s_1 \xrightarrow{\sigma} s_1'. \exists s_2 \xrightarrow{\sigma} s_2'.\ (s_1', s_2') \in Y\ \}\\
\cup\ &\{\ (s_1, s_2) \mid s_1 \notin F_1 \wedge \forall \sigma. \forall s_1 \xrightarrow{\sigma} s_1'. \exists s_2 \xrightarrow{\sigma} s_2'.\ (s_1', s_2') \in Z\ \}
\end{aligned}
$$

Intuitively, the outer greatest fixpoint tracks visits to an accepting state in the right-hand automaton. The least fixpoint models the fact that visits to F_2 states are allowed to be separated by *finitely* many game rounds where F_2 is not visited. Finally, the inner greatest fixpoint is used to exploit non-accepting cycles in the left-hand automaton: if the left-hand automaton is stuck in a rejecting loop (i.e., a loop visiting no F_1 states), there is no need to ensure fairness in the right-hand side either.

The key theorem, that the rest of the paper builds on, is that fair similarity gives a sound technique for language inclusion [10, 9, 7]. We note, however, that `fairsim` is incomplete for language inclusion if A_2 is allowed to be a non-deterministic automaton [5, 10].

▶ **Theorem 6.** $\forall s_1 \in S_1. \forall s_2 \in S_2.\ \texttt{fairsim}(s_1, s_2) \implies \texttt{Lang}_{A_1}(s_1) \subseteq \texttt{Lang}_{A_2}(s_2)$

Proof. The proof goes by coinduction on $\texttt{Lang}_{A_2}$, and then by successive induction on the inductive part of `fairsim`, and the inductive part of $\texttt{Lang}_{A_1}$. More precisely, we use the following coinduction hypothesis (`CH`) and the following two induction hypotheses (`IH1`, `IH2`):

$$
\begin{aligned}
\texttt{CH} &\triangleq \{(s_2, w) \mid \exists s_1.\ \texttt{fairsim}(s_1, s_2) \wedge \texttt{Lang}_{A_1}(q_1, w)\}\\
\texttt{IH1} &\triangleq \{(s_1, s_2) \mid \forall w.\ \texttt{Lang}_{A_1}(s_1, w) \Rightarrow (s_2, w) \in \mu L.\ \texttt{LangF}_{A_2}(\texttt{CH}, L)\}\\
\texttt{IH2} &\triangleq \{(s_1, w) \mid \forall s_2.\ (s_1, s_2) \in (\nu Z.\texttt{F}(\texttt{fairsim}, \texttt{IH1}, Z)) \Rightarrow (s_2, w) \in \mu L.\texttt{LangF}_{A_2}(\texttt{CH}, L)\}
\end{aligned}
$$

where $\mathtt{F}$ is the function such that $\mathtt{fairsim} = \nu X.\mu Y.\nu Z.\ \mathtt{F}(X,Y,Z)$ (see Definition 5). To use these hypotheses, we start by observing that the statement of the theorem can be equivalently rephrased as $\mathtt{CH} \subseteq \mathtt{Lang}_{A_2}$. Since $\mathtt{Lang}_{A_2}$ is a greatest fixpoint, this inclusion can be proven by coinduction.

$$\begin{aligned} & \mathtt{CH} \subseteq \mathtt{Lang}_{A_2} \\ \text{Definition of Lang} \quad \Longleftarrow \quad & \mathtt{CH} \subseteq \nu L_\nu.\mu L_\mu.\ \mathtt{LangF}_{A_2}(L_\nu, L_\mu) \\ \text{By coinduction} \quad \Longleftarrow \quad & \mathtt{CH} \subseteq \mu L_\mu.\ \mathtt{LangF}_{A_2}(\mathtt{CH}, L_\mu) \end{aligned}$$

By definition of $\mathtt{CH}$ and $\mathtt{IH1}$, this last inclusion can be equivalently rephrased as $\mathtt{fairsim} \subseteq \mathtt{IH1}$. Since $\mathtt{fairsim}$ hides a least fixpoint, the proof can continue by induction.

$$\begin{aligned} & \mathtt{fairsim} \subseteq \mathtt{IH1} \\ \text{By def of fairsim} \quad \Longleftarrow \quad & (\nu X.\mu Y.\nu Z.\ \mathtt{F}(X,Y,Z)) \subseteq \mathtt{IH1} \\ \text{By unfolding } \nu X \quad \Longleftarrow \quad & (\mu Y.\nu Z.\ \mathtt{F}(\mathtt{fairsim}, Y, Z)) \subseteq \mathtt{IH1} \\ \text{By induction} \quad \Longleftarrow \quad & (\nu Z.\ \mathtt{F}(\mathtt{fairsim}, \mathtt{IH1}, Z)) \subseteq \mathtt{IH1} \end{aligned}$$

Finally, by definition of $\mathtt{IH1}$ and $\mathtt{IH2}$, this last inclusion can be equivalently rephrased as $\mathtt{Lang}_{A_1} \subseteq \mathtt{IH2}$. Since $\mathtt{Lang}_{A_1}$ also hides a least fixpoint, this inclusion can also be proven by induction. In fact, our proof uses *strong* induction.

$$\begin{aligned} & \mathtt{Lang}_{A_1} \subseteq \mathtt{IH2} \\ \text{Definition of Lang} \quad \Longleftarrow \quad & (\nu L_\nu.\mu L_\mu.\ \mathtt{LangF}_{A_1}(L_\nu, L_\mu)) \subseteq \mathtt{IH2} \\ \text{By unfolding } \nu L_\nu \quad \Longleftarrow \quad & (\mu L_\mu.\ \mathtt{LangF}_{A_1}(\mathtt{Lang}_{A_1}, L_\mu)) \subseteq \mathtt{IH2} \\ \text{By strong induction} \quad \Longleftarrow \quad & \mathtt{LangF}_{A_1}(\mathtt{Lang}_{A_1}, (\mu L_\mu.\ \mathtt{Lang}_{A_1}) \cap \mathtt{IH2}) \subseteq \mathtt{IH2} \end{aligned}$$

After unfolding the definition of $\mathtt{IH2}$ and set inclusion, it remains to show

$$(s_1, w) \in \mathtt{LangF}_{A_1}(\ldots) \Rightarrow (s_1, s_2) \in (\nu Z.\mathtt{F}(\ldots)) \Rightarrow (s_2, w) \in \mu L_\mu.\ \mathtt{LangF}_{A_2}(\mathtt{CH}, L_\mu)$$

for all s_1, s_2, w. The remainder of the proof proceeds by case analysis on $(s_1, w) \in \mathtt{LangF}_{A_1}(\ldots)$ and $(s_1, s_2) \in \nu Z.\,\mathtt{F}(\ldots)$. This generates a total of 6 cases (corresponding to the 2 clauses in $\mathtt{LangF}$ and the 3 clauses in $\mathtt{F}$). Each case is easily covered by choosing the appropriate clause of $\mathtt{LangF}_{A_2}$ (depending on whether s_2 is accepting or not), and using the induction/coinduction hypotheses. ◀

As an application of Theorem 6, since $\mathtt{fairsim}$ is defined as a coinductive-inductive-coinductive predicate, it could in principle be used as is to prove language inclusion by coinduction-induction-coinduction in an interactive proof assistant. Unfortunately, as demonstrated in the proof of Theorem 6, such nested proofs are quite tedious to write: we need to guess suitable induction and coinduction hypothesis up-front, and to carefully unfold and refold nested fixpoints. The more nestings of ν's and μ's we have, the more tedious it gets. It is therefore desirable to have higher-level reasoning principles.

For similarity relations involving a single greatest fixpoint (but potentially several nested least fixpoints), techniques such as *parameterized coinduction* [11], the *companion approach* [17], or *tower induction* [18] can be directly used to facilitate the mechanization of proofs. For example, Correnson et al. proposed deductive systems to prove language inclusion of NBAs based on parameterized coinduction [5]. Their deductive systems are built on top of fairness preserving similarity relations with a simpler $\nu\mu$ structure, and are thus

strictly finer than `fairsim` (we prove this result in Theorem 12). As a result, these systems are sound but incomplete for fair similarity. Intuitively, a single greatest fixpoint only allows to either exploit a non-accepting cycle in the left-hand automaton or an accepting cycle in the right-hand automaton, but never truly both at the same time. This intuition is covered in greater details in Example 11.

For weaker similarity relations involving deeper nestings of fixpoints (such as `fairsim`), obtaining deductive systems requires more work: a naive use of parameterized coinduction is possible, but as we will see in Section 4.1, it does not immediately translate into usable proof rules.

3.2 A Deductive System for Fair Similarity

We propose a new deductive system for mechanized proofs of language inclusion, based on fair similarity. Contrary to the systems of Correnson et al. [5], ours is complete for Henzinger's et al. fair similarity [10] (i.e., Definition 5). In some cases, it also enables simpler proofs.

The general idea of fair simulation is to refine the standard notion of similarity, with the additional requirements that infinitely many visits to left-accepting states must be simulated by infinitely many visits to right-accepting ones. This requirement is implemented by tracking non-accepting cycles in the left-hand automaton, and accepting cycles in the right-hand automaton. Our deductive system therefore employs a combination of two *guarded contexts* to track exactly these two things. Concretely, our deductive system operates on proof quadruples of the following forms:

$$\boxed{H_O};\boxed{H_I} \vdash s_1 \preccurlyeq s_2 \qquad \boxdot{H_O};\boxed{H_I} \vdash s_1 \preccurlyeq s_2 \qquad \boxed{H_O};\boxdot{H_I} \vdash s_1 \preccurlyeq s_2$$

where H_O and H_I are binary relations between states, and s_1 and s_2 are two states we wish to prove to be language included (i.e., $\texttt{Lang}(s_1) \subseteq \texttt{Lang}(s_2)$). We note that we refer to the contexts $\boxed{H_O}$ and $\boxed{H_I}$ as *guarded* contexts because they represent assumptions that are not always freely available during a proof. A solid box (e.g., $\boxed{H_I}$) indicates that assumptions H_I cannot yet be exploited, whereas a dashed box (e.g., $\boxdot{H_I}$) indicates that assumptions H_I are available. This notation is borrowed from previous papers on coinductive proofs [3, 5, 4].

Intuitively, the relations H_O and H_I can be thought of as fragments of fair simulation relations, i.e., they represent states that are assumed to be fairly similar. In other words, a quadruple $\boxed{H_O};\boxed{H_I} \vdash s_1 \preccurlyeq s_2$ can be read as "s_1 and s_2 are fairly similar, under the (guarded) assumption that states related in H_O and H_I are also fairly similar". The subtlety is that, during proofs, H_O will exclusively contain states that are known to be fairly similar because of accepting cycles in the right-hand automaton, while H_I will exclusively contain states that are known to be fairly similar because of rejecting cycles in the left-hand automaton (i.e., cycles with no accepting states). The indices O and I stand for "outer" and "inner", as the contexts intuitively correspond to the outer and inner ν's in the definition of fair similarity (see Definition 5).

This intuition, although partly inaccurate, is already enough to state a few properties associated with these quadruples. Namely, when $H_O = H_I = \emptyset$ (i.e., no states are assumed to be fairly similar), validity of a quadruple implies language inclusion.

▶ **Theorem 7.** $\forall s_1 \in S_1. \forall s_2 \in S_2.\ \boxed{\emptyset};\boxed{\emptyset} \vdash s_1 \preccurlyeq s_2 \iff \texttt{fairsim}(s_1, s_2)$

▶ **Theorem 8.** $\forall s_1 \in S_1. \forall s_2 \in S_2.\ \boxed{\emptyset};\boxed{\emptyset} \vdash s_1 \preccurlyeq s_2 \implies \texttt{Lang}_{A_1}(s_1) \subseteq \texttt{Lang}_{A_2}(s_2)$

We delay the formal proof of these statements until Section 4.2. In the meantime, we introduce the core reasoning rules of the deductive system. We present the rules divided into three main categories: *accumulation rules* to make hypotheses on states, *cycle rules* to use unguarded hypotheses, and *step rules* to release the guards.

Accumulation Rules

The first two rules are *accumulation rules* used to insert new pairs of states in the two guarded contexts. At any time during a proof of a quadruple $\boxed{H_O};\boxed{H_I} \vdash s_1 \preccurlyeq s_2$, any set H' containting (s_1, s_2) can be accumulated in H_O or in H_I. This can be thought of as making the assumptions that all pairs in H' are similar (including in particular the current pair (s_1, s_2)). After extending the context, we need to check the validity of the new assumptions by proving a quadruple for *all* pairs in H', not just for (s_1, s_2). As will be discussed below, assumptions in H_O can be exploited after visiting an accepting state of the right-hand automaton, while assumptions in H_I can be exploited after a visit to a non-accepting state of the left-hand automaton. This corresponds to the intuition that H_O is used to discover accepting cycles in the right-hand automaton, while H_I is used to discover rejecting cycles in the left-hand one.

$$\textsc{AccOut}\quad \frac{(s_1, s_2) \in H' \qquad \forall (s_1', s_2') \in H'.\ \boxed{H_O \cup H'};\boxed{\emptyset} \vdash s_1' \preccurlyeq s_2'}{\boxed{H_O};\boxed{H_I} \vdash s_1 \preccurlyeq s_2}$$

$$\textsc{AccIn}\quad \frac{(s_1, s_2) \in H' \qquad \forall (s_1', s_2') \in H'.\ \boxed{H_O};\boxed{H_I \cup H'} \vdash s_1' \preccurlyeq s_2'}{\boxed{H_O};\boxed{H_I} \vdash s_1 \preccurlyeq s_2}$$

It is important to note that these two accumulation rules are *not* exactly symmetric: When accumulating in the outer context H_O (via the rule AccOut), the inner context has to be emptied. In contrast, accumulating in the inner context H_I does not have any impact on the outer context. The reason why is explained in detail in Section 4.1.

As we will see later, it is often convenient to just accumulate the singleton $\{(s_1, s_2)\}$. This gives the following two derived rules, which are particularly useful when manipulating finite-state automata.

$$\textsc{SingleAccOut}\quad \frac{\boxed{H_O \cup (s_1, s_2)};\boxed{\emptyset} \vdash s_1 \preccurlyeq s_2}{\boxed{H_O};\boxed{H_I} \vdash s_1 \preccurlyeq s_2} \qquad \textsc{SingleAccIn}\quad \frac{\boxed{H_O};\boxed{H_I \cup (s_1, s_2)} \vdash s_1 \preccurlyeq s_2}{\boxed{H_O};\boxed{H_I} \vdash s_1 \preccurlyeq s_2}$$

Cycle Rules

The next two rules enable the usage of accumulated pairs of states. If a context is unguarded, we can use any assumption in the context to finish the proof. So, if the current state pair is in that context, we can conclude. Again, there is one rule for each context.

$$\textsc{CycleOut}\quad \frac{(s_1, s_2) \in H_O}{\boxed{H_O};\boxed{H_I} \vdash s_1 \preccurlyeq s_2} \qquad \textsc{CycleIn}\quad \frac{(s_1, s_2) \in H_I}{\boxed{H_O};\boxed{H_I} \vdash s_1 \preccurlyeq s_2}$$

Step Rules

In the previous paragraphs, we introduced the accumulation rules, which can be used to add pairs to the context, and the cycle rules, to use assumptions in the contexts when they are opened. The missing part is how to *open* the contexts. The *step rules* define exactly this. To release the guard around the outer context (respectively inner), we need to validate that the current state pair is indeed part of an accepting (respectively rejecting) cycle. To do so, we need to explore all possible outgoing transitions of the left state, and match them with appropriate transitions from the right state. Depending on whether the current states are accepting or not, this will give access (or not) to one of the contexts. This is implemented by the following three rules:

$$\textsc{Accept}\quad \frac{s_2 \in F_2 \qquad \forall \sigma. \forall s_1 \xrightarrow{\sigma} s_1'. \exists s_2 \xrightarrow{\sigma} s_2'.\ \boxed{H_O};\boxed{\emptyset} \vdash s_1' \preccurlyeq s_2'}{\boxed{H_O};\boxed{H_I} \vdash s_1 \preccurlyeq s_2}$$

$$\textsc{Reject}\quad \frac{s_1 \notin F_1 \qquad \forall \sigma. \forall s_1 \xrightarrow{\sigma} s_1'. \exists s_2 \xrightarrow{\sigma} s_2'.\ \boxed{H_O};\boxed{H_I} \vdash s_1' \preccurlyeq s_2'}{\boxed{H_O};\boxed{H_I} \vdash s_1 \preccurlyeq s_2}$$

$$\textsc{Step}\quad \frac{\forall \sigma. \forall s_1 \xrightarrow{\sigma} s_1' \exists s_2 \xrightarrow{\sigma} s_2'.\ \boxed{H_O};\boxed{\emptyset} \vdash s_1' \preccurlyeq s_2'}{\boxed{H_O};\boxed{H_I} \vdash s_1 \preccurlyeq s_2}$$

The three rules are very similar, they only differ by (1) the side condition on the current state, and (2) which guarded assumptions become available.

The first rule, Accept, demands that s_2 is an accepting state. If so, taking a step grants access to the outer context (thus enabling to discover right-accepting cycles). Similarly, the rule Reject demands s_1 is *not* an accepting state. If so, taking a step grants access to the inner context (thus enabling to discover left-rejecting cycles). The last rule Step has no side condition, meaning that one can always step through finitely many transitions, ignoring the acceptance conditions. In this case, guards are not released.

Observe that both Accept and Step reset the inner context. Without this, our deductive system would be unsound for language inclusion. We give a more detailed justification in Section 3.3. For now, we can already mention that this subtlety stems from the different nature of accepting and rejecting cycles: a cycle is rejecting in an NBA if *no state* in the cycle is accepting, whereas a cycle is accepting if at least *one state* in the cycle is accepting.

Guard Rules

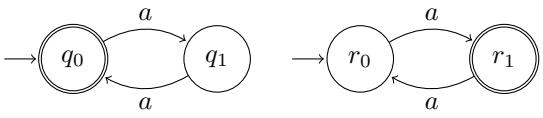
The last two rules state that the guards around the context can always be freely restored since restricting access to an assumption is always sound. Again, there is one rule per context.

$$\textsc{GuardOut}\quad \frac{\boxed{H_O};\boxed{H_I} \vdash s_1 \preccurlyeq s_2}{\boxed{H_O};\boxed{H_I} \vdash s_1 \preccurlyeq s_2} \qquad \textsc{GuardIn}\quad \frac{\boxed{H_O};\boxed{H_I} \vdash s_1 \preccurlyeq s_2}{\boxed{H_O};\boxed{H_I} \vdash s_1 \preccurlyeq s_2}$$

Note that in proofs these two rules are applied so often that we use them implicitly. More precisely, whenever one of the two contexts is unguarded, and we want to use a rule which needs a guarded context, we apply one of these two rules.

Having presented all rules of the deductive system, we now look at two simple examples, which demonstrate how we use the two contexts.

▶ **Example 9.** Consider the following two automata:



Both automata accept the language a^ω, however, the accepting states are not aligned. We want to show $\texttt{Lang}(q_0) \subseteq \texttt{Lang}(r_0)$, thus we prove $\boxed{\emptyset};\boxed{\emptyset} \vdash q_0 \preccurlyeq r_0$. Intuitively, in the proof, we only have one cycle we want to exploit, namely the accepting one ($r_0 \xrightarrow{a} r_1 \xrightarrow{a} r_0$) in the right-hand automaton.

The proof starts by accumulating the current state pair in the outer context, because we want to use it for an accepting cycle in the right-hand automaton.

$$\boxed{\emptyset};\boxed{\emptyset} \vdash q_0 \preccurlyeq r_0$$
$$\text{By AccOut} \impliedby \boxed{(q_0, r_0)};\boxed{\emptyset} \vdash q_0 \preccurlyeq r_0$$

Next, the proof continues with a step rule. As q_0 only has one outgoing transition, namely $q_0 \xrightarrow{a} q_1$, which we match with $r_0 \xrightarrow{a} r_1$, we get one new goal.

$$\text{By Step} \impliedby \boxed{(q_0, r_0)};\boxed{\emptyset} \vdash q_1 \preccurlyeq r_1$$

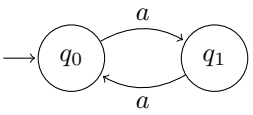
As r_0 is not an accepting state, the guard around the outer context is not released. The guard around the inner context also cannot be released as q_0 is an accepting state.

As we are now in r_1, which is an accepting state, we use the Accept rule. This then opens the guard around the outer context.

$$\text{By Accept} \impliedby \boxed{(q_0, r_0)};\boxed{\emptyset} \vdash q_0 \preccurlyeq r_0$$

Now, the proof can be concluded by an application of the rule CycleOut. ⌟

▶ **Example 10.** Consider the following automaton, which describes the empty language.



In fact, both states q_0 and q_1 accept the empty language. In particular, this implies $\texttt{Lang}(q_0) \subseteq \texttt{Lang}(q_1)$. We validate this claim by proving $\boxed{\emptyset};\boxed{\emptyset} \vdash q_0 \preccurlyeq q_1$.

Contrary to the example above (where we exploited an accepting cycle in the right), this time we want to exploit a non-accepting cycle ($q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_0$). Thus, we start by accumulating the current pair of states in the inner context.

$$\boxed{\emptyset};\boxed{\emptyset} \vdash q_0 \preccurlyeq q_1$$
$$\text{By AccIn} \impliedby \boxed{\emptyset};\boxed{(q_0, q_1)} \vdash q_0 \preccurlyeq q_1$$

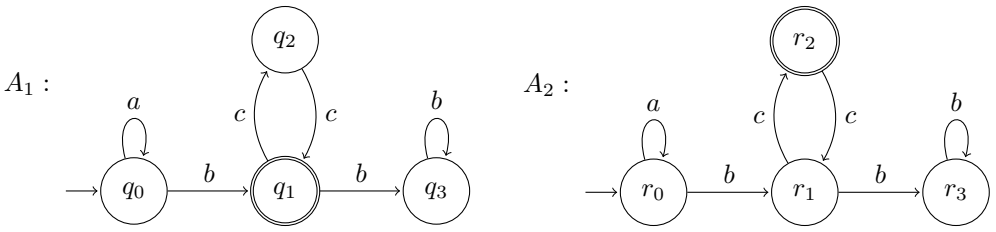
The proof continues by performing two steps using the step rule Reject. Note that both states only have one outgoing transition. Thus, the first application brings us to the state pair (q_1, q_0) and the second application brings us back to (q_0, q_1).

$$\text{By Reject and GuardIn} \Longleftarrow \boxed{\emptyset}; \boxed{(q_0, q_1)} \vdash q_1 \preccurlyeq q_0$$
$$\text{By Reject} \Longleftarrow \boxed{\emptyset}; \boxed{(q_0, q_1)} \vdash q_0 \preccurlyeq q_1$$

Note that in both cases the guard around the inner context is released as both q_0 and q_1 are not accepting. After the second application, we are back at (q_0, q_1) and hence, we can conclude using the rule CycleIn. ⌟

It turns out that the two examples above can also be handled by *Almost Fair Simulations* [5]. The limits of *Almost Fair Simulations* become apparent when looking at an example which requires simultaneous reasoning about accepting and non-accepting cycles. The following example demonstrates this.

▶ **Example 11.** Consider the following two automata:

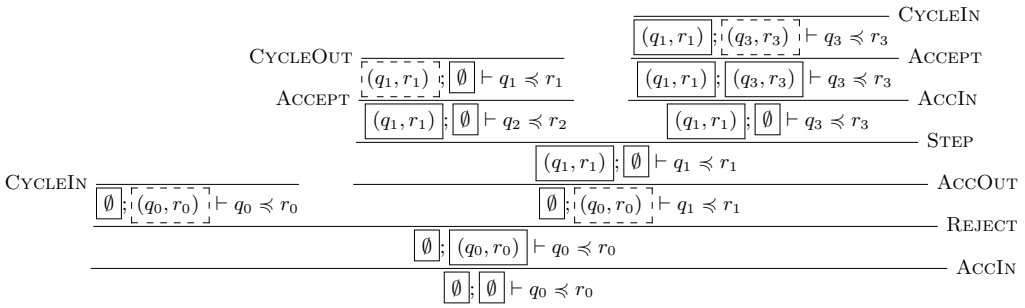


It is not difficult to see that $\texttt{Lang}(q_0) \subseteq \texttt{Lang}(r_0)$: the two automata are almost identical. The only difference is that the accepting states are in different places, but this does not impact the accepted words. In fact, one can even prove $\texttt{Lang}(q_0) = \texttt{Lang}(r_0) = (a^*)b(c^\omega)$.

Proving $\texttt{Lang}(q_0) \subseteq \texttt{Lang}(r_0)$ is not possible using *repeated-delay simulation*, the weakest relation presented in *Almost Fair Simulations* [5]. Most of the fairness preserving simulation relations given in *Almost Fair Simulations* [5] are based on notions of similarity with *delay* [7]. The main intuition behind delayed simulations, is that any visit to a left-accepting state will immediately force a subsequent visit to a right-accepting state *on all paths* and *within a finite number of steps*. As a consequence, seeing an accepting state in the left-hand automaton forbids exploiting non-accepting cycles until an accepting state in the right-hand automaton is reached. This is precisely the reason why this example is not provable by delayed simulation. Indeed, after starting the proof in (q_0, r_0), we eventually reach (q_1, r_1). Since q_1 is accepting, we are required to see an accepting state in the right-hand automaton in all outgoing paths from r_1. This is impossible because of the rejecting loop $r_3 \xrightarrow{b} r_3$.

Repeated-delay simulation [5] attempts to mitigate this problem by extending the standard notion of delayed similarity [7] with an additional least fixpoint. Although this helps in many cases (see [5] for a more detailed explanation), it unfortunately does not help in this particular example. In fact, we can formally prove that q_0 and r_0 are not repeated-delay similar (see the accompanying Rocq code). However, this example can be handled by fair similarity. In particular, we can use our deductive system to show $\texttt{fairsim}(q_0, r_0)$.

The intuition of the proof is to use the inner context to exploit the two rejecting cycles $q_0 \xrightarrow{a} q_0$ and $q_3 \xrightarrow{a} q_3$, while simultaneously discovering the accepting cycle $r_1 \xrightarrow{c} r_2 \xrightarrow{c} r_1$ using the outer context. This is done by accumulating (q_0, r_0) and (q_3, r_3) in the inner context, and (q_1, r_1) in the outer one. The rest of the proof goes by stepping through the automata, using the cycle rules to conclude when possible.



This example gives evidence that having the two greatest fixpoints in `fairsim` indeed enables more reasoning power (as opposed to similarity relations with a single greatest fixpoint).

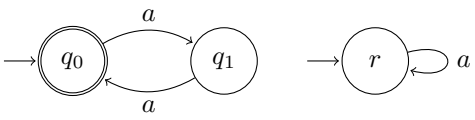
▶ **Theorem 12.** *Repeated-delay similarity is a strictly stronger relation than fair similarity (and thus a strictly weaker proof technique).*

Proof. The example above justifies the *strict* part, so it remains to show that repeated-delay similarity implies fair similarity. Conceptually, the proof goes by nested coinduction on the fixpoint definition of fair similarity. Finding the right coinduction and induction hypotheses is particularly tricky. However, as our deductive system is sound for fair similarity (see Theorem 20), it can be used to greatly simplify the mechanization of the proof. ◀

3.3 On the Necessity of Resetting the Inner Context

We already mentioned that in the two rules Accept and Step, the inner context needs to be emptied. Without this subtlety, the deductive system would not be sound. The necessity to reset the inner context is demonstrated by the following example.

▶ **Example 13.** Consider the following two automata:



For the sake of contradiction, let us assume we have access to the following "idealized" proof rule:

$$\textsc{StepUnsound}\ \frac{\forall \sigma. \forall s_1 \xrightarrow{\sigma} s_1'. \exists s_2 \xrightarrow{\sigma} s_2'.\ \boxed{H_O};\boxed{H_I} \vdash s_1' \preccurlyeq s_2'}{\boxed{H_O};\boxed{H_I} \vdash s_1 \preccurlyeq s_2}$$

With this rule, we can (wrongly) show that the left-hand automaton is language included in the right-hand one. However, this is not true: $\texttt{Lang}(q_0) = a^\omega$, but $\texttt{Lang}(r_0) = \emptyset$. The (wrong) proof goes as follows: First, we accumulate the initial state pair (q_0, r) in the inner context. Second, we use the unsound rule to transition to the pair (q_1, r). Lastly, we can use Reject since q_1 is not accepting. This last step opens the guard around the inner context and thus, we can conclude using the inner cycle rule, even though $q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_0$ is *not* a rejecting cycle.

$$
\begin{aligned}
& \boxed{\emptyset}\,;\boxed{\emptyset} \vdash q_0 \preccurlyeq r \\
\text{By AccIn} \quad \Longleftarrow\ & \boxed{\emptyset}\,;\boxed{(q_0, r)} \vdash q_0 \preccurlyeq r \\
\text{By StepUnsound} \quad \Longleftarrow\ & \boxed{\emptyset}\,;\boxed{(q_0, r)} \vdash q_1 \preccurlyeq r \\
\text{By Reject} \quad \Longleftarrow\ & \boxed{\emptyset}\,;\boxed{(q_0, r)} \vdash q_0 \preccurlyeq r \\
\text{By CycleIn} \quad \Longleftarrow\ & (q_0, r) \in \{(q_0, r)\}
\end{aligned}
$$

The critical point is that a cycle is rejecting if and only if it *only* contains non-accepting states. Here, because StepUnsound does *not* reset the context, it enabled us to step through a right-accepting state in the middle of establishing a rejecting cycle.

Observe that there is no need for such a reset when establishing right-accepting cycles. In particular, a cycle is accepting so long as it contains at least one accepting state, but it can otherwise contain non-accepting ones. ⌟

4 Soundness and Completeness

In this section, we prove soundness and completeness of our deductive system for fair similarity. In *Almost Fair Simulations* [5], deductive systems (for fair similarity) have a single guarded context which intuitively correspond to the fact that they deal with a single greatest fixpoint. The soundness of these systems is justified by relying on the framework of *parameterized coinduction* [11], which offers an elegant semantic interpretation of the guarded context and its associated accumulation rule. To achieve completeness for fair similarity, we need to handle *two* nested greatest fixpoints, and hence *two* associated guarded contexts. It turns out that parameterized coinduction also provides a semantic interpretation for the two contexts, modulo a little bit of fixpoint reasoning. This section explains the process.

4.1 Nested Parameterized Coinduction

In a typical simulation proof, we are interested in proving statements of the form $X \subseteq \nu F$ where X is a relation between states of a transition system (often a unique pair of states (s_1, s_2)). These statements (inclusions into greatest fixpoints) are provable by coinduction. For example, to prove $(s_1, s_2) \in \texttt{sim}$, it suffices to exhibit a *simulation relation* R such that $(s_1, s_2) \in R$ and $R \subseteq \texttt{simF}(R)$. This technique is already very useful because it enables proving *global* properties about the behavior of systems (e.g., language inclusion) by *local* reasoning about states and transitions. The limitation of direct proofs by coinduction though is that they require guessing an entire simulation relation *up front* [11]. In contrast, *parameterized coinduction* (short Paco) enables a more incremental proof style, where a simulation relation is progressively discovered by accumulation [11].

The key idea to achieve this incremental proof style is to replace greatest fixpoints νF with a *parameterized* variant $\texttt{G}_F(H) \triangleq \nu X.F(H \cup X)$, where H represents a fragment of a coinduction hypothesis. $\texttt{G}_F(\emptyset) = \nu F$, meaning any proof of $X \subseteq \nu F$ can be replaced by a proof of $X \subseteq \texttt{G}_F(\emptyset)$. Further, the following two key reasoning principles can be derived [11]:

$$
\text{Paco-Accumulate}\ \frac{X \subseteq \texttt{G}_F(X \cup H)}{X \subseteq \texttt{G}_F(H)} \qquad \text{Paco-Step}\ \frac{X \subseteq F(H \cup \texttt{G}_F(H))}{X \subseteq \texttt{G}_F(H)}
$$

In combination, these two rules enable the incremental discovery of coinduction hypotheses. The reader might have recognized an *accumulation* principle (Paco-Accumulate) enabling to insert the currently targeted set X in the current "context"; and a *stepping* principle

(Paco-Step), granting access to said "context" after an F-step. These are reminiscent of the core reasoning rules of our deductive system. We shall see that, indeed, Paco provides a suitable semantic interpretation of our deductive system with guarded contexts. There is still one problem to address: Paco only talks about *one* context. In contrast, our deductive system employs two. To give a semantic interpretation to the two contexts, we will sequentially parameterize the two greatest fixpoints underlying the definition of `fairsim` using the Paco construction (`G`).

First, let us observe that the definition of `fairsim` (see Definition 5) can be put into the following form

$$\texttt{fairsim} \triangleq \nu X.\mu Y.\nu Z.\ \texttt{Accept}(X)\ \cup\ \texttt{Step}(Y)\ \cup\ \texttt{Reject}(Z)$$

where `Accept`, `Step` and `Reject` are the following predicate transformers:

$$\begin{aligned}
\texttt{Accept}(X) &\triangleq \{\ (s_1,s_2) \mid s_2 \in F_2 \wedge \forall\sigma.\forall s_1 \xrightarrow{\sigma} s_1'.\exists s_2 \xrightarrow{\sigma} s_2'.\ (s_1',s_2') \in X\ \}\\
\texttt{Step}(Y) &\triangleq \{\ (s_1,s_2) \mid \forall\sigma.\forall s_1 \xrightarrow{\sigma} s_1'.\exists s_2 \xrightarrow{\sigma} s_2'.\ (s_1',s_2') \in Y\ \}\\
\texttt{Reject}(Z) &\triangleq \{\ (s_1,s_2) \mid s_1 \notin F_1 \wedge \forall\sigma.\forall s_1 \xrightarrow{\sigma} s_1'.\exists s_2 \xrightarrow{\sigma} s_2'.\ (s_1',s_2') \in Z\ \}
\end{aligned}$$

What we want is a parameterized variant $\texttt{fairsim}(H_O, H_I)$. We decompose the construction of this $\texttt{fairsim}(H_O, H_I)$ in three phases: First, we will parameterize the outer ν to obtain a construction $\texttt{outer}(H_O)$ with a single parameter H_O. This will justify the outer accumulation rule. Then, we will parameterize the inner ν to obtain a construction $\texttt{inner}(H_O, H_I)$, this time with two contexts. This will justify the inner accumulation rule. Finally, we will show that one can go back and forth between `inner` and `outer`. In the end, we will define $\texttt{fairsim}(H_O, H_I) \triangleq \texttt{inner}(H_O, H_I)$.

Since `fairsim` itself is defined as a greatest fixpoint, we can start by simply parameterizing the outer ν using the standard Paco construction. This motivates the following definition $\texttt{outer}(H_O)$.

$$\begin{aligned}
&\texttt{outerF}(X) \triangleq \mu Y.\nu Z.\ \texttt{Accept}(X)\ \cup\ \texttt{Step}(Y)\ \cup\ \texttt{Reject}(Z)\\
&\texttt{outer}(H_O) \triangleq \texttt{G}_{\texttt{outerF}}(H_O)
\end{aligned}$$

By definition of `G`, $\texttt{outer}(\emptyset) = \texttt{fairsim}$. This will enable us to derive an accumulation principle for H_O as a consequence of Paco-Accumulate. To obtain the remaining reasoning principles, and in particular the inner accumulation principle, we introduce the `inner` construction. Starting from $\texttt{outer}(H_O)$, the idea is to fold and unfold the first μ and ν to pull the inner ν outwards and then parameterize the inner ν.

$$\begin{aligned}
\texttt{outer}(H_O) &= \texttt{G}_{\texttt{outerF}(H_O)}\\
(\text{definition of } G) &= \nu X.\mu Y.\nu Z.\ \texttt{Accept}(H_O \cup X) \cup \texttt{Step}(Y) \cup \texttt{Reject}(Z)\\
(\text{unfold } \nu X) &= \mu Y.\nu Z.\ \texttt{Accept}(H_O \cup \texttt{outer}(H_O)) \cup \texttt{Step}(Y) \cup \texttt{Reject}(Z)\\
(\text{unfold } \mu Y) &= \nu Z.\ \texttt{Accept}(H_O \cup \texttt{outer}(H_O)) \cup \texttt{Step}(\mu Y.\nu Z.\ldots) \cup \texttt{Reject}(Z)\\
(\text{fold } \texttt{outer}) &= \nu Z.\ \underbrace{\texttt{Accept}(H_O \cup \texttt{outer}(H_O))\ \cup\ \texttt{Step}(\texttt{outer}(H_O))\ \cup\ \texttt{Reject}(Z)}_{\texttt{innerF}(H_O,Z)}
\end{aligned}$$

Having exposed the inner ν, we can parameterize it to obtain the following definition:

$$\begin{aligned}
&\texttt{innerF}(H_O, Z) \triangleq \texttt{Accept}(H_O \cup \texttt{outer}(H_O))\ \cup\ \texttt{Step}(\texttt{outer}(H_O)\ \cup\ \texttt{Reject}(Z))\\
&\texttt{inner}(H_O, H_I) \triangleq \texttt{G}_{\texttt{innerF}(H_O)}(H_I)
\end{aligned}$$

From the definition of `inner`, using Paco-Step and Paco-Accumulate (and a bit of fixpoint reasoning) we can derive the following reasoning rules.

$$\textsc{Inner-Accumulate-O}\ \frac{X \subseteq \texttt{inner}(H_O \cup X, \emptyset)}{X \subseteq \texttt{inner}(H_O, H_I)} \qquad \textsc{Inner-Accumulate-I}\ \frac{X \subseteq \texttt{inner}(H_O, H_I \cup X)}{X \subseteq \texttt{inner}(H_O, H_I)} \qquad \textsc{Inner-Accept}\ \frac{X \subseteq \texttt{Accept}(H_O \cup \texttt{outer}(H_O))}{X \subseteq \texttt{inner}(H_O, H_I)}$$

$$\textsc{Inner-Step}\ \frac{X \subseteq \texttt{Step}(\texttt{outer}(H_O))}{X \subseteq \texttt{inner}(H_O, H_I)} \qquad \textsc{Inner-Reject}\ \frac{X \subseteq \texttt{Reject}(H_I \cup \texttt{inner}(H_O, H_I))}{X \subseteq \texttt{inner}(H_O, H_I)}$$

▶ **Theorem 14.** *The rules* Inner-Accumulate-I*,* Inner-Accept*,* Inner-Step *and* Inner-Reject *are sound.*

Proof. The rule Inner-Accumulate-I is a direct consequence of Paco-Accumulate applied to `inner`. The three rules Inner-Accept, Inner-Reject, and Inner-Step are immediate consequences of Paco-Step applied to `inner`. ◀

We have two problems left to solve: First, proving Inner-Accumulate-O. Second, the rules Inner-Accept and Inner-Step have premises mentioning $\texttt{outer}(H_O)$. Ideally, we would like everything to be formulated in terms of `inner` so that the system is closed. Both problems are addressed by observing that we can go back and forth between `inner` and `outer`.

▶ **Theorem 15** (Inner-Outer)**.** $\texttt{inner}(H_O, \emptyset) \subseteq \texttt{outer}(H_O)$

▶ **Theorem 16** (Outer-Inner)**.** $\texttt{outer}(H_O) \subseteq \texttt{inner}(H_O, H_I)$

Hence, when proving $X \subseteq \texttt{outer}(H_O)$, one can instead try to prove $\texttt{inner}(H_O, \emptyset)$. In this case, we start with no inner assumptions (i.e. $H_I = \emptyset$). Similarly, when proving $X \subseteq \texttt{inner}(H_O, H_I)$, one can *drop* all assumptions H_I and prove $X \subseteq \texttt{outer}(H_O)$ instead. The soundness of Inner-Accumulate-O is obtained (bottom-up) by going from `inner` to `outer`, accumulating in `outer`, and going back to `inner`. In the process, the inner assumptions H_I are lost.

▶ **Theorem 17.** *The rule Inner-Acccumulate-O is sound.*

Finally, the premise of Inner-Accept and Inner-Step can be reformulated purely in terms of `inner` thanks to Theorem 15 (and exploiting the monotonicity of `Step` and `Accept`). As a result, we obtain the following principles:

$$\textsc{Inner-Accept'}\ \frac{X \subseteq \texttt{Accept}(H_O \cup \texttt{inner}(H_O, \emptyset))}{X \subseteq \texttt{inner}(H_O, H_I)} \qquad \textsc{Inner-Step'}\ \frac{X \subseteq \texttt{Step}(\texttt{inner}(H_O, \emptyset))}{X \subseteq \texttt{inner}(H_O, H_I)}$$

The loss of the assumption H_I corresponds to the resets of the inner context in our deductive system (see Sections 3.2 and 3.3).

▶ **Theorem 18.** *The rules* Inner-Accept' *and* Inner-Step' *are sound.*

Proof. By monotonicity of `Step` and `Accept` and Theorem 15. ◀

4.2 Soundness and Completeness of our Deductive System

Soundness

The soundness of the rules presented in Section 3.2 is justified using the following semantic model (note that we use $\vDash$ instead of $\vdash$):

$$\boxed{H_O};\boxed{H_I} \vDash s_1 \preccurlyeq s_2 := (s_1, s_2) \in \texttt{inner}(H_O, H_I)$$
$$\boxed{};\boxed{H_I} \vDash s_1 \preccurlyeq s_2 := (s_1, s_2) \in H_O \cup \texttt{inner}(H_O, H_I)$$
$$\boxed{H_O};\boxed{} \vDash s_1 \preccurlyeq s_2 := (s_1, s_2) \in H_I \cup \texttt{inner}(H_O, H_I)$$

Note that we do not need to give a semantics to quadruples where the two contexts are ungarded, because such quadruples cannot be reached using the rules of Section 3.2. Since $\texttt{inner}(\emptyset, \emptyset) = \texttt{fairsim}$, this semantic model is sound and complete for fair similarity. From the soundness of the model, we easily derive the soundness of the deductive system itself.

▶ **Theorem 19.** $\forall s_1. \forall s_2. \boxed{\emptyset};\boxed{\emptyset} \vDash s_1 \preccurlyeq s_2 \iff \texttt{fairsim}(s_1, s_2)$

▶ **Theorem 20** (Soundness). $\forall s_1. \forall s_2. \boxed{\emptyset};\boxed{\emptyset} \vdash s_1 \preccurlyeq s_2 \implies \texttt{fairsim}(s_1, s_2)$

Proof. We prove $\boxed{\emptyset};\boxed{\emptyset} \vdash s_1 \preccurlyeq s_2 \implies \boxed{\emptyset};\boxed{\emptyset} \vDash s_1 \preccurlyeq s_2$ by induction on the derivation $\boxed{\emptyset};\boxed{\emptyset} \vdash s_1 \preccurlyeq s_2$, using the rules of `inner` presented in the previous subsection to cover all cases. ◀

Completeness

In our Rocq development, we initially did not explicitly define our deductive system (e.g., as an inductive predicate representing valid derivations). Instead, we directly developed the semantic model (`inner`) and obtained the rules as theorems. This approach is sufficient to prove soundness, but it does not justify completeness. Indeed, even though we already proved that the semantic model itself is complete (Theorem 19), this does not justify that the *finite* set of rules presented in Section 3.2 is sufficient to prove all statements of the form $\texttt{fairsim}(s_1, s_2)$. Fortunately, the finite set of rules of Section 3.2 turns out to be complete for fair similarity. This result is obtained as a consequence of Paco being a complete proof technique for inclusions into greatest fixpoints.

▶ **Theorem 21** (Completeness). $\forall s_1, s_2. \texttt{fairsim}(s_1, s_2) \implies \boxed{\emptyset};\boxed{\emptyset} \vdash s_1 \preccurlyeq s_2$

Proof (Sketch). Let s_1, s_2 be two states and suppose $\texttt{fairsim}(s_1, s_2)$. Intuitively, the nested $\nu\mu\nu$ definition of `fairsim` guarantees the existence of a first post-fixpoint R_1 (for the outer ν), and a second post-fixpoint R_2 (for the inner ν). The accumulation of R_1 in H_O and R_2 in H_I suffices to conclude the proof. A bit more formally, the proof starts by accumulating `fairsim` using AccOut and then proceeds by induction on the inductive part of `fairsim`. The first case (disjunct `Accept`) is covered using the rules Accept and CycleOut. The second case (disjunct `Step`) is covered using Step and the induction hypothesis. The last case (disjunct `Reject`) is covered using AccIn followed by Reject and CycleIn. ◀

Nested Parameterized Coinduction, beyond Fair Similarity

We note that the proofs discussed in this section are independent of the exact definition of `Accept`, `Step`, and `Reject`. In fact, the `inner` construction and its associated reasoning rules offer a sound and complete deductive system for proofs of inclusion in any predicate of the form $\nu X.\mu Y.\nu Z.\ F_1(X) \cup F_2(Y) \cup F_3(Z)$ where F_1, F_2, F_3 are monotone predicate transformers.

5 Deductive System in the Proof Assistant

We used the proof assistant Rocq to formalize all results presented in this paper. To further simplify the usage of our deductive system, we defined tactics to partially automate the selection of proof rules and the verification of side conditions.

The three main tactics are `iBranch`, `iStep`, and `iCycle`. The tactic `iBranch` tries applying one of the step rules, and generates one proof obligation per successor of the left-hand state. Each proof obligation requires the users to exhibit a matching successor of the right-hand state. After using `iBranch`, the tactic `iStep` s can be used to choose a successor s of the current right-hand state. We note that the side conditions of the three step rules Accept, Reject and Step are not mutually exclusive. Therefore, `iBranch` needs to decide which one to pick. To determine which of the three step rules should be applied, `iBranch` uses the following heuristic: First, it tries to apply the Accept rule (if the current right-hand state is accepting). If this fails, it tries to apply the Reject rule (if the current left-hand state is non-accepting). If this also fails, it applies the rule Step, which can always be applied since it does not have any side condition. Finally, the tactic `iCycle` tries applying CycleIn or CycleOut to conclude a proof depending on which of the two contexts is ungarded.

We find that using these tactics, proving language inclusion of two Büchi automata in Rocq is concise and closely follows the intuitive steps without overwhelming the user with unnecessary technical details. For example, the proof of Example 9 in Rocq looks as follows:

```
Lemma sim_q0_r0_tacs:
  fairsim ... ∅ ∅ q0 r0.
Proof.
  iAccOut.                    (* Goal: [(q0, r0)] ; [∅] ⊢ q0 ≼ r0 *)
  iBranch. iStep r1.          (* Goal: [(q0, r0)] ; [∅] ⊢ q1 ≼ r1 *)
  iBranch. iStep r0.          (* Goal: (q0, r0) ; [∅] ⊢ q0 ≼ r0 *)
  iCycle.
Qed.
```

6 Related Work

Fair Simulations, Games, and Automata

The idea of proofs by simulation can be traced back to Milner [15]. It was rapidly adapted to support reasoning about transition systems equipped with various fairness conditions [16, 1]. For the specific case of Büchi automata, which play a central role in model-checking of temporal properties, standard approaches include direct simulation [6], delayed simulation [7] and fair simulation [10]. These simulation approaches were all introduced with algorithmic verification in mind, mostly for the purpose of implementing automata simplification procedures [7], or as a cheap (but incomplete) method to automatically check language inclusion [6]. Their use for mechanized deductive reasoning has been comparatively much less explored. In this space, Correnson et al. recently introduced *Almost Fair Simulations*, a family of deductive systems for interactive proofs of language inclusion [5]. Their systems are mechanized in Rocq, and feature ergonomic reasoning principles based on a single guarded context. They are however restricted to notions of simulation with a single greatest fixpoint, resulting in an unnecessary loss of expressiveness. Our approach addresses this limitation by introducing reasoning principles supporting one more nesting of greatest fixpoint.

Previous work by Lichter and Smolka focused on formalizing a large body of results related to Büchi automata in the constructive type theory of Rocq [14]. Their development mostly focuses on proving the decidability of logics based on automata construction. To our knowledge, it does not include any result about fair simulation. We note that their formalization of the semantics of Büchi automata is based on explicit quantification on runs represented as sequences indexed with natural numbers. In contrast, our development is based on coinductive types and predicates.

Interactive Proofs by Coinduction

Coinduction is commonly used in the context of mechanized simulation proofs. For various technical reasons, it is also known for pushing proof assistants to their limits. The introduction of parameterized coinduction by Hur et al. [11] greatly contributed to making coinduction more applicable. Since then, several other frameworks to facilitate proofs by coinduction have been developed. For example, Pous introduced *the companion approach* [17] as an alternative to parameterized coinduction with a strong emphasis on support for so called "up-to techniques", a powerful tool to simplify coinductive proofs by working with much smaller coinduction hypotheses. Later, Zakowski et al. proposed *generalized parameterized coinduction* (Gpaco) [22], a generalization of Paco specifically designed to facilitate reasoning about coinductive-inductive relations (i.e. $\nu\mu$) which typically occur when working with weak notions of similarity or bisimilarity. In principle, these approaches can also be used to reason about any deeper nestings of fixpoints (after all, deeper nestings $\nu\mu\nu\mu\ldots$ are just special cases of greatest fixpoints). However, as shown in Section 4.1, obtaining simple high-level reasoning rules may require additional work for deeply nested fixpoints. We believe that the approach presented in Section 4.1 to obtain reasoning rules for $\nu\mu\nu$-type predicates should easily be generalizable to arbitrary nestings. This would lead to deductive systems with n guarded contexts, one per ν. Another interesting direction would be to investigate the role of up-to techniques in the context of coinductive-inductive-coinductive relations such as the one discussed in this paper.

Recently, a range of mechanized program logics based on notions of fair refinement have been introduced. These include Simuliris [8], a separation logic for refinements preserving fair termination of concurrent programs; Trillium [20], another (family of) separation logic(s) based on fair refinement and dedicated to liveness proofs; and Lilo [13], a separation logic for liveness proofs, based on *fair operational semantics* [12]. To the best of our knowledge, these are all based on relatively strong notions of fairness preserving simulation (of type $\nu\mu$).

7 Conclusion and Future Work

We introduced a sound and complete deductive system for fair similarity of Büchi automata, mechanized in the Rocq proof assistant. To justify the soundness and completeness of our system, we introduced nested parameterized coinduction as an intermediate technique to handle the $\nu\mu\nu$ nesting of fixpoints underlying the definition of fair similarity.

Although we originally introduced nested parameterized coinduction specifically for proofs of fair similarity, it could in principle be used as a basis to develop deductive systems for other problems. In particular, it can already be used as an incremental proof technique to prove the existence of a winning strategy in any three-colors parity game. Beyond simulation proofs, these games play a central role in other applications such as reactive synthesis.

Future work includes the extension of the approach to deeper nestings of greatest and least fixpoints. This would enable the construction of deductive systems for a richer class of games (e.g., arbitrary parity games).

References

1 Adnan Aziz, Vigyan Singhal, Felice Balarin, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. Equivalences for fair kripke structures. In Serge Abiteboul and Eli Shamir, editors, *Automata, Languages and Programming*, pages 364–375, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg. doi:10.1007/3-540-58201-0_82.

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

3 Minki Cho, Youngju Song, Dongjae Lee, Lennard Gäher, and Derek Dreyer. Stuttering for free. *Proc. ACM Program. Lang.*, 7(OOPSLA2), October 2023. doi:10.1145/3622857.

4 Arthur Correnson and Bernd Finkbeiner. Coinductive proofs for temporal hyperliveness. *Proc. ACM Program. Lang.*, 9(POPL), 2025. doi:10.1145/3704889.

5 Arthur Correnson, Iona Kuhn, and Bernd Finkbeiner. Almost fair simulations. *Proc. ACM Program. Lang.*, 9(ICFP), August 2025. doi:10.1145/3747512.

6 David L. Dill, Alan J. Hu, and Howard Wong-Toi. Checking for language inclusion using simulation preorders. In Kim G. Larsen and Arne Skou, editors, *Computer Aided Verification*, pages 255–265, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

7 Kousha Etessami, Thomas Wilke, and Rebecca A. Schuller. Fair simulation relations, parity games, and state space reduction for Büchi automata. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Automata, Languages and Programming*, pages 694–707, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. doi:10.1007/3-540-48224-5_57.

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

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

10 Thomas A Henzinger, Orna Kupferman, and Sriram K Rajamani. Fair simulation. *Information and Computation*, 173(1):64–81, 2002. doi:10.1006/inco.2001.3085.

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

12 Dongjae Lee, Minki Cho, Jinwoo Kim, Soonwon Moon, Youngju Song, and Chung-Kil Hur. Fair operational semantics. *Proc. ACM Program. Lang.*, 7(PLDI), June 2023. doi:10.1145/3591253.

13 Dongjae Lee, Janggun Lee, Taeyoung Yoon, Minki Cho, Jeehoon Kang, and Chung-Kil Hur. Lilo: A higher-order, relational concurrent separation logic for liveness. *Proc. ACM Program. Lang.*, 9(OOPSLA1), April 2025. doi:10.1145/3720525.

14 Moritz Lichter and Gert Smolka. Constructive analysis of s1s and büchi automata, 2018. arXiv:1804.04967.

15 Robin Milner. Equivalences on program schemes. *J. Comput. Syst. Sci.*, 4(3):205–219, 1970. doi:10.1016/S0022-0000(70)80021-6.

16 David Michael Ritchie Park. Concurrency and automata on infinite sequences. In Peter Deussen, editor, *Theoretical Computer Science, 5th GI-Conference, Karlsruhe, Germany, March 23-25, 1981, Proceedings*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer, 1981. doi:10.1007/BFB0017309.

17 Damien Pous. Coinduction all the way up. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, pages 307–316, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2933575.2934564.

18 Steven Schäfer and Gert Smolka. Tower induction and up-to techniques for ccs with fixed points. In Peter Höfner, Damien Pous, and Georg Struth, editors, *Relational and Algebraic Methods in Computer Science*, pages 274–289, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-57418-9_17.

19 Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955. doi:10.2140/pjm.1955.5.285.

20 Amin Timany, Simon Oddershede Gregersen, Léo Stefanesco, Jonas Kastberg Hinrichsen, Léon Gondelman, Abel Nieto, and Lars Birkedal. Trillium: Higher-order concurrent and distributed separation logic for intensional refinement. *Proc. ACM Program. Lang.*, 8(POPL), 2024. doi:10.1145/3632851.

21 Igor Walukiewicz. Monadic second-order logic on tree-like structures. *Theoretical Computer Science*, 275(1):311–346, 2002. doi:10.1016/S0304-3975(01)00185-2.

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