Process Equivalence Checking as Abstract Interpretation*

Benjamin Bisping

SAMOVAR, Télécom SudParis, Insitut Polytechnique de Paris, Palaiseau, France
https://bbisping.de

Abstract. We flesh out that *equivalence checking* constitutes a form of backward-complete *abstract interpretation* on pairs of processes. The first part of the paper illustrates this framing for trace semantics on finite-state systems. We then extend the approach to uniformly handle simulation, failure equivalence, and ready simulation in a single abstract interpretation.

Keywords: Equivalence checking · Abstract interpretation · Program difference · Bisimulation · Traces · Ready simulation · Process algebra

1 Introduction

How do abstract interpretation and equivalence checking relate? Both are core concepts in the analysis of programs and processes. And while abstract interpretation brought some order to the jungle of static analyses, equivalence checking often appears as a more ad hoc activity.

Abstract interpretation is a general theory for sound approximation of semantics, and a powerful tool for designing static analyses due to Cousot and Cousot [5]. Instead of the whole semantics $[\![p]\!]$ of a process p, one computes an abstract value $\widehat{[\![p]\!]}$ that approximates it with respect to relevant properties.

Process equivalence checking answers the question whether two processes p and q are equivalent, $p \sim_N q$, where N is a semantic equivalence notion, for instance, bisimulation or trace equivalence. This can be rephrased as a question about semantics: Do p and q have the same meaning, $[\![p]\!] = [\![q]\!]$ with respect to N? So, equivalence checking can be seen as a special case of semantic analysis, where the question is not about properties of a *single* process, but about the relationship between *two processes*: What observations may one make of one process but not of the other? Are there differences $[\![p]\!] \setminus [\![q]\!] \neq \emptyset$? This suggests:

* This wip article comes out of my first experiment to co-develop a paper and a formalization with GitHub Copilot. Starting from notes and proof sketches for a chapter that I had envisioned for my PhD thesis [2] but then scrapped, I asked Copilot (mostly running GPT-5.4) to formalize definitions and proofs, similar to a student supervision. (I enjoyed the experience far less than with students…) Send me your thoughts to benjamin.bisping@telecom-sudparis.eu!

Funding: My work is partially funded through a project with *Orange Innovation*.

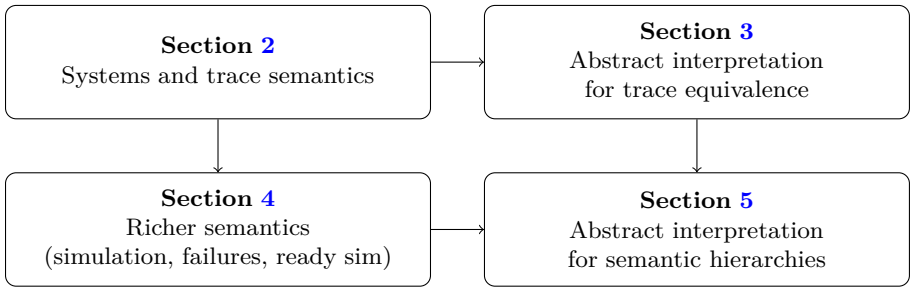

Figure 1: Logical structure of the paper.

Equivalence checking algorithms can be understood as abstract interpretation: They aim to find out just enough about one process's semantics to distinguish it from the other, computing an abstract representation of the difference. This abstraction must be *complete* in the sense that, if there is a concrete distinguishing observation, an abstract one must be found.

Contribution. This paper develops the perspective of equivalence-checking-as-abstract-interpretation in detail. We first consider only classical yes/no equivalence checking, and then show how the abstract interpretation view naturally extends to hierarchies of distinctions. The paper is less about one more equivalence algorithm than about a uniform way of representing and computing semantic differences. We proceed in two major layers as illustrated in Figure 1:

- For preliminaries, Section 2 defines trace semantics for the Calculus of Communicating Systems (in a way that is mostly agnostic towards the formalism) and derives the corresponding equivalences and differences. We give a *concrete transformer* D_{Tr} that collects trace differences as a least fixpoint.
- Section 3 presents an *abstract transformer* $D^{\sharp}_{\mathrm{Tr}}$ for D_{Tr} in the context of a backward-complete abstract interpretation whose least fixpoint characterizes the presence of distinguishing traces in a *computable* way.
- Section 4 recalls richer semantics for simulation, failure, and ready simulation semantics from the linear-time–branching-time spectrum [18].
- With Section 5, we give a single abstract interpretation that computes abstract differences for all these semantics *at once*.
- Section 6 connects the approach to related work, in particular, to Ranzato and Tapparo [14] and to Bisping [2].

Formalization. All definitions and theorems are supported by a Lean *auto-formalization*, available at https://benkeks.github.io/eq-checking-abstract-interpretation/. The appendix contains more traditional proofs that came out of playing back and forth between proof blueprints and autoformalization.

2 Trace Semantics

Trace preorder and equivalence are among the simplest and most fundamental semantic notions for processes. This section introduces some notation for trace difference and equivalence on the small CCS language, and shows how they can be characterized by a concrete transformer.

2.1 Trace Observations on Sequential CCS

Our framework needs some formalism with semantics. For illustration purposes, we will use finite-state terms in the Calculus of Communicating Systems [11]:

Definition 1 (CCS syntax)**.** Assume given a set of process names $\mathcal{X}$ and action names $\mathcal{A}$. The set of processes CCS is defined by the following grammar:

$$p \ ::= \ a.p \ \mid \ p + p \ \mid \ 0 \ \mid \ X \qquad \text{with } a \in \mathcal{A}, X \in \mathcal{X}.$$

Definition 2 (Derivative sets)**.** Given a finite assignment $\mathcal{V} \colon \mathcal{X} \to \mathsf{CCS}$ for process names, we define one-step behavior by *derivative sets*

$$\mathrm{Der}(p, a) \subseteq \mathsf{CCS} \qquad (a \in \mathcal{A}),$$

as the least family satisfying:

$$\mathrm{Der}(0, a) := \emptyset, \qquad \mathrm{Der}(b.p, a) := \begin{cases} \{p\} & \text{if } a = b, \\ \emptyset & \text{otherwise} \end{cases}$$

$$\mathrm{Der}(p + q, a) := \mathrm{Der}(p, a) \cup \mathrm{Der}(q, a), \qquad \mathrm{Der}(X, a) := \mathrm{Der}(\mathcal{V}(X), a).$$

For sets of processes $P \subseteq \mathsf{CCS}$, we use the lifted derivative operator

$$\mathrm{Der}(P, a) := \bigcup_{p \in P} \mathrm{Der}(p, a).$$

For the *immediately enabled actions* of process p, we write

$$\mathrm{En}(p) := \{a \in \mathcal{A} \mid \mathrm{Der}(p, a) \neq \emptyset\}.$$

We define *trace observations* by inductive rules from the derivations.

Definition 3 (Denotational trace observations)**.** Define $[\![p]\!]_{\mathrm{Tr}}$ as the least *set of traces* $\langle a_1 \rangle \ldots \langle a_n \rangle \top$ with $a_i \in \mathcal{A}$ derivable through:[1]

$$\frac{}{\top \in [\![p]\!]_{\mathrm{Tr}}} \qquad \frac{p' \in \mathrm{Der}(p, a) \qquad t \in [\![p']\!]_{\mathrm{Tr}}}{\langle a \rangle t \in [\![p]\!]_{\mathrm{Tr}}}$$

We denote by $\mathsf{Obs}_{\mathrm{Tr}}$ the language of all finite (but unbounded-length) trace observations (for a fixed $\mathcal{A}$-set). We identify $\top$ with the empty trace and work with finite traces only.[2]

[1] Lean inductive: Trace.TraceSem

[2] We will write traces, and later trees of behavior, using the syntax of Hennessy–Milner logic [7], but treat the formulas as purely syntactic objects without a satisfaction relation.

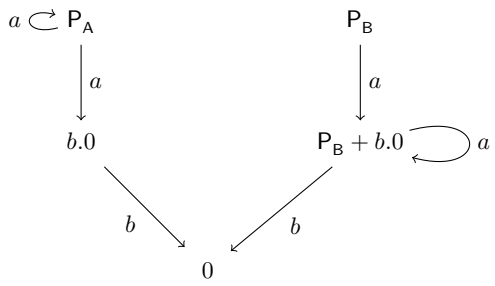

Figure 2: Transition system for $\mathsf{P_A}$ and $\mathsf{P_B}$ of Example 4, showing only reachable states.

So the trace-observation semantics is directly denotational: It maps each CCS term p to the language $[\![p]\!]_{\mathrm{Tr}} \subseteq \mathsf{Obs}_{\mathrm{Tr}}$ of traces derivable from these rules.

Example 4 (Looping order)**.** Consider two CCS processes over actions $\mathcal{A} = \{a, b\}$:

$$\mathsf{P_A} = a.\mathsf{P_A} + a.b.0 \qquad \mathsf{P_B} = a.(\mathsf{P_B} + b.0)$$

Both processes can perform action a, but they differ subtly in their branching structure: $\mathsf{P_A}$ makes a non-deterministic a-choice at the top level: either repeat as $\mathsf{P_A}$, or perform b then stop. $\mathsf{P_B}$ performs a, then offers the choice: either repeat, or perform b then stop. Figure 2 shows their transition system.

For the trace denotations of $b.0$, $\mathsf{P_A}$ and $\mathsf{P_B}$, we compute by Definition 3:

$$[\![b.0]\!]_{\mathrm{Tr}} = \{\top, \langle b\rangle\top\}$$

$$[\![\mathsf{P_A}]\!]_{\mathrm{Tr}} = \{\langle a\rangle^n\top \mid n \geq 0\} \cup \{\langle a\rangle^n\langle b\rangle\top \mid n \geq 1\} = [\![\mathsf{P_B}]\!]_{\mathrm{Tr}}.$$

2.2 Equivalences and Preorders through Semantic Differences

Let's frame equivalence and preorder checking in terms of differences of semantics. It is common to consider processes equivalent, $p \sim q$, if they have the same semantics, $[\![p]\!] = [\![q]\!]$; likewise, $p \preceq q$ corresponds to $[\![p]\!] \subseteq [\![q]\!]$. In terms of differences:

Definition 5 (Difference)**.** The trace difference between p and q is defined as:[3]

$$\Delta_{\mathrm{Tr}}(p, q) := [\![p]\!]_{\mathrm{Tr}} \setminus [\![q]\!]_{\mathrm{Tr}}.$$

Overloaded to a set of processes $Q \subseteq \mathsf{CCS}$, we write:

$$\Delta_{\mathrm{Tr}}(p, Q) := [\![p]\!]_{\mathrm{Tr}} \setminus \bigcup_{q \in Q} [\![q]\!]_{\mathrm{Tr}}.$$

[3] Lean def: Trace.TraceDifference

Definition 6 (Trace preorder and trace equivalence)**.** Two processes are trace-preordered iff no trace observation of p is missing in q:

$$p \preceq_{\mathrm{Tr}} q :\iff \Delta_{\mathrm{Tr}}(p, q) = \emptyset.$$

They are trace-equivalent iff preorder holds in both directions:

$$p \sim_{\mathrm{Tr}} q :\iff p \preceq_{\mathrm{Tr}} q \wedge q \preceq_{\mathrm{Tr}} p.$$

Example 7 (Trace equivalence and distinction in the running example)**.** Applying this to Example 4:

Since $[\![\mathsf{P_A}]\!]_{\mathrm{Tr}} = [\![\mathsf{P_B}]\!]_{\mathrm{Tr}}$, we have $\Delta_{\mathrm{Tr}}(\mathsf{P_A}, \mathsf{P_B}) = \emptyset$ and $\Delta_{\mathrm{Tr}}(\mathsf{P_B}, \mathsf{P_A}) = \emptyset$, so $\mathsf{P_A} \sim_{\mathrm{Tr}} \mathsf{P_B}$.

By contrast, consider $b.0$ and $\mathsf{P_B} + b.0$ (the intermediate state reachable from $\mathsf{P_B}$ by one a-step, as in Figure 2). We have $[\![b.0]\!]_{\mathrm{Tr}} = \{\top, \langle b\rangle\top\}$, $[\![\mathsf{P_B} + b.0]\!]_{\mathrm{Tr}} = \{\langle a\rangle^n\top \mid n \geq 0\} \cup \{\langle a\rangle^n\langle b\rangle\top \mid n \geq 0\}$. Since every trace of $b.0$ appears in $\mathsf{P_B} + b.0$, we have $\Delta_{\mathrm{Tr}}(b.0, \mathsf{P_B} + b.0) = \emptyset$. Thus $b.0 \preceq_{\mathrm{Tr}} \mathsf{P_B} + b.0$.

However, in the other direction, $\Delta_{\mathrm{Tr}}(\mathsf{P_B} + b.0, b.0) = \{\langle a\rangle^n\top \mid n \geq 1\} \cup \{\langle a\rangle^n\langle b\rangle\top \mid n \geq 1\} \neq \emptyset$, an infinite set of distinguishing traces! Thus $\mathsf{P_B} + b.0 \not\preceq_{\mathrm{Tr}} b.0$; the two processes are preordered in one direction only, not trace-equivalent.

With this view, equivalence and preorder checking reduce to checking emptiness of behavioral differences. But let us first find out how to enumerate differences if they exist.

2.3 Concrete Trace Differences

Definition 5 takes the set-theoretic difference of two denotation sets, thus of two things that can be expressed as least fixpoints. But for the equivalence checking view, we need to make the difference collection for process pairs itself explicit as *one single* least fixpoint construction. In the abstract-interpretation view of Section 3, this will be our *concrete transformer* for trace differences. In order to express it in terms of single steps in spite of non-determinism, we need to build difference systems connecting processes and *sets* of processes (instead of plain process pairs).

Definition 8 (Concrete predecessor transformer for trace differences)**.** Let

$$DS := (\mathsf{CCS} \times 2^{\mathsf{CCS}}) \to 2^{\mathsf{Obs}_{\mathrm{Tr}}}$$

be the space of *difference systems*, ordered pointwise by subset inclusion:[4]

$$\delta_1 \sqsubseteq \delta_2 :\iff \forall p, Q.\ \delta_1(p, Q) \subseteq \delta_2(p, Q).$$

Define the monotone[5] function[6] $D_{\mathrm{Tr}} \colon DS \to DS$ recursively on traces:

$$\top \in D_{\mathrm{Tr}}(\delta)(p, Q) :\iff Q = \emptyset,$$
$$\langle a\rangle u \in D_{\mathrm{Tr}}(\delta)(p, Q) :\iff \exists p' \in \mathrm{Der}(p, a).\ u \in \delta(p', \mathrm{Der}(Q, a)).$$

[4] Lean def: Trace.DiffSys
[5] Lean theorem: Trace.dTr_mono
[6] Lean def: Trace.DTr

Intuitively, D_{Tr} propagates distinguishing traces backward along derivatives: $\top$ distinguishes p from Q exactly when $Q = \emptyset$, and a prefixed trace $\langle a \rangle u$ distinguishes p from Q whenever some successor $p' \in \mathrm{Der}(p, a)$ is distinguished from $\mathrm{Der}(Q, a)$ by u.

Proposition 9 (Concrete trace difference as least fixpoint)**.** *For all $p \in \mathsf{CCS}$ and $Q \subseteq \mathsf{CCS}$,*[7]

$$\Delta_{\mathrm{Tr}}(p, Q) = \mathrm{lfp}(D_{\mathrm{Tr}})(p, Q).$$

Proof in Appendix A.

With Definition 6, trace-preorder reduces to a fixpoint-emptiness check:

$$p \preceq_{\mathrm{Tr}} q \iff \mathrm{lfp}(D_{\mathrm{Tr}})(p, \{q\}) = \emptyset.$$

But this check is formulated in terms of an object that will often contain infinitely many distinguishing traces, where a classical lfp-fixpoint iteration will not terminate. From the computational perspective, it would be desirable to only compute as much as we need (and in finite time). This jump is precisely what *abstract interpretation* is about, leading us to the next section.

3 Distinguishability via Abstract Interpretation

We will now frame trace equivalence checking as an abstract interpretation. For this, we abstract the previous concrete transformer D_{Tr} to an abstract transformer $D^\sharp_{\mathrm{Tr}}$ that computes *just enough* to decide trace preorder. The goal of the following definitions is to ensure that p is trace-distinguishable from each process in Q precisely when:

$$\mathsf{T} \in \mathcal{D}_{\mathrm{Tr}}(p, Q).$$

The concrete trace itself is abstracted away, T is a purely syntactic marker, and $\mathcal{D}_{\mathrm{Tr}}(\cdot, \cdot) := \mathrm{lfp}(D^\sharp_{\mathrm{Tr}})$ is defined in Definition 10 below. Theorem 13 will show the definitions to fit our bill.

3.1 Abstract Interpretation Components for Traces

Abstract interpretation generally connects a concrete domain C to an abstract one A by a Galois connection (α, γ) with $\alpha(c) \sqsubseteq_A \hat{a} \iff c \sqsubseteq_C \gamma(\hat{a})$. The concrete and abstract semantics are given in terms of transformers on concrete and abstract difference systems.

Definition 10 (Abstract interpretation for trace differences)**.** We instantiate the abstract-interpretation framework for trace differences as follows.

Concrete and abstract domains.

$$C := 2^{\mathsf{Obs}_{\mathrm{Tr}}}, \qquad \sqsubseteq_C := \subseteq,$$

$$A := \{\emptyset, \{\mathsf{T}\}\}, \qquad \sqsubseteq_A := \subseteq .$$

[7] Lean theorem: Trace.traceDifferenceToSet_eq_lfpDTr

Abstraction and concretization.

$$\alpha(c) := \begin{cases} \emptyset & \text{if } c = \emptyset, \\ \{\mathsf{T}\} & \text{if } c \neq \emptyset, \end{cases} \qquad \gamma(a) := \begin{cases} \emptyset & \text{if } a = \emptyset, \\ \mathsf{Obs}_{\mathrm{Tr}} & \text{if } a = \{\mathsf{T}\}. \end{cases}$$

Concrete and abstract transformers. As concrete transformer we already have D_{Tr} from Definition 8. We accompany it by an *abstract* transformer: Given an abstract difference system $\hat{\delta} \colon (\mathsf{CCS} \times 2^{\mathsf{CCS}}) \to 2^{\{\mathsf{T}\}}$ with pointwise subset order, define $D^{\sharp}_{\mathrm{Tr}}$ by[8]

$$D^{\sharp}_{\mathrm{Tr}}(\hat{\delta})(p, Q) := \begin{cases} \{\mathsf{T}\} & \text{if } Q = \emptyset, \\ \bigcup_{a \in \mathcal{A}} \bigcup_{p' \in \mathrm{Der}(p,a)} \hat{\delta}(p', \mathrm{Der}(Q, a)) & \text{otherwise.} \end{cases}$$

Abstract interpretation. The abstract result is computed as the least fixpoint of the abstract transformer:

$$\mathcal{D}_{\mathrm{Tr}}(\cdot, \cdot) := \mathrm{lfp}(D^{\sharp}_{\mathrm{Tr}}).$$

Example 11 (Abstract trace differences for the running example)**.** We continue with the running example $\mathsf{P_A} = a.\mathsf{P_A} + a.b.0$ and $\mathsf{P_B} = a.(\mathsf{P_B} + b.0)$. Recall $\mathrm{Der}(\{b.0\}, a) = \emptyset$ since $b.0$ has no a-transition. Define $\hat{\delta}_0 := \emptyset$ and $\hat{\delta}_{n+1} := D^{\sharp}_{\mathrm{Tr}}(\hat{\delta}_n)$.

$\mathsf{P_B} + b.0$ **is distinguishable from** $b.0$**.** We show $\mathsf{T} \in \mathcal{D}_{\mathrm{Tr}}(\mathsf{P_B} + b.0, \{b.0\})$, i.e. $\mathsf{P_B} + b.0 \not\preceq_{\mathrm{Tr}} b.0$.[9] The derivation takes two transformer steps: $\hat{\delta}_1(\mathsf{P_B} + b.0, \emptyset) = \{\mathsf{T}\}$ by the empty-Q clause. Since $\mathsf{P_B} + b.0 \in \mathrm{Der}(\mathsf{P_B} + b.0, a)$ and $\mathrm{Der}(\{b.0\}, a) = \emptyset$, the next iterate yields $\hat{\delta}_2(\mathsf{P_B} + b.0, \{b.0\}) = D^{\sharp}_{\mathrm{Tr}}(\hat{\delta}_1)(\mathsf{P_B} + b.0, \{b.0\}) = \hat{\delta}_1(\mathsf{P_B} + b.0, \emptyset) \cup \hat{\delta}_1(0, \{0\}) = \{\mathsf{T}\} \cup \emptyset = \{\mathsf{T}\}$. Thus $\mathsf{T} \in \mathcal{D}_{\mathrm{Tr}}(\mathsf{P_B} + b.0, \{b.0\})$.

$b.0$ **is not distinguishable from** $\mathsf{P_B} + b.0$**.** We show $\mathcal{D}_{\mathrm{Tr}}(b.0, \{\mathsf{P_B} + b.0\}) = \emptyset$, i.e. $b.0 \preceq_{\mathrm{Tr}} \mathsf{P_B} + b.0$.[10] The graph from $(b.0, \{\mathsf{P_B} + b.0\})$ closes immediately under the zero valuation. At the entry pair, $b.0$ has only one derivative 0 via b, and $\mathrm{Der}(\{\mathsf{P_B} + b.0\}, b) = \{0\}$, so: $D^{\sharp}_{\mathrm{Tr}}(\hat{\delta}_0)(b.0, \{\mathsf{P_B} + b.0\}) = \hat{\delta}_0(0, \{0\}) = \emptyset$. At the successor pair, 0 has no derivatives at all, so $D^{\sharp}_{\mathrm{Tr}}(\hat{\delta}_0)(0, \{0\}) = \emptyset$ as well. So no iterate can introduce a marker at the entry pair, and therefore $\mathcal{D}_{\mathrm{Tr}}(b.0, \{\mathsf{P_B} + b.0\}) = \emptyset$.

3.2 Correctness and Preorder Checking

We show that the marker analysis is exact for emptiness checking of concrete trace difference by establishing that the relationship between concrete and abstract transformers is backward-complete, that is, that $D^{\sharp}_{\mathrm{Tr}}$ mirrors D_{Tr} exactly in the abstract domain.

[8] Lean def: Trace.DTrSharp
[9] Lean theorem: Trace.RunningExample.abstractDiff_PBb0_b0
[10] Lean theorem: Trace.RunningExample.not_abstractDiff_b0_PBb0

Lemma 12 (Backward completeness for the trace abstraction)**.** *The abstraction is backward-complete.*[11]

$$\alpha \circ D_{\mathrm{Tr}} = D^{\sharp}_{\mathrm{Tr}} \circ \alpha.$$

Proof in Appendix A.

Combining Lemma 12 with Proposition 9 and fixpoint machinery, we obtain:

Theorem 13 (Precise approximation of trace-difference non-emptiness)**.** *For all* $p \in \mathsf{CCS}$ *and* $Q \subseteq \mathsf{CCS}$,[12]

$$\mathsf{T} \in \mathcal{D}_{\mathrm{Tr}}(p, Q) \iff \Delta_{\mathrm{Tr}}(p, Q) \neq \emptyset.$$

Proof in Appendix A.

As an immediate consequence, preorder checking reduces to a single abstract non-emptiness query:[13]

$$p \preceq_{\mathrm{Tr}} q \iff \mathcal{D}_{\mathrm{Tr}}(p, \{q\}) = \emptyset.$$

Equivalence follows according to Definition 6. In summary, the abstraction is exact for the emptiness question underlying preorder/equivalence checking with respect to traces.

So far, we have just re-framed trace equivalence checking in a way that is a little unconventional from the equivalence perspective, and conventional from the abstract interpretation perspective. The next sections will show how the shifted view enables more generic algorithms.

4 Simulation, Failures, and Ready Simulation

As a second round of preliminaries, this section recalls a part of the linear-time–branching-time spectrum [18], namely the one between trace equivalence and ready simulation. This is the "denotational part" of the hierarchy, where finite (recursion-free) processes have finite sets of observations as their semantics.

We consider two orthogonal observation capabilities: alternatives ("branching" / logical conjunction) and immediately impossible actions ("refusals" / negations of depth one). With these, we can characterize failure, simulation, and ready-simulation preorders. Mirroring Section 2, we culminate in establishing a concrete transformer to collect ready-simulation observations with Proposition 18.

[11] Lean theorem: Trace.backwardComplete_DTr
[12] Lean theorem: Trace.markerPresence_iff_concreteDiffNonempty
[13] Lean theorem: Trace.tracePreorder_iff_no_marker

4.1 Concrete Observations Beyond Traces

Trace semantics only captures sequences of actions. To distinguish more behaviors, we introduce richer observation structures. In Section 2, observations were linear: either $\top$ (empty trace) or $\langle a \rangle t$ (action a followed by observation t). We now add conjunctions of possible follow-up trees and of immediately impossible actions. We continue to write the observations as formulas of Hennessy–Milner logic (HML) [7], now of the form

$$\mathsf{Obs}_{\mathrm{RS}}: \quad \varphi ::= \bigwedge \left(\{ \langle a_s \rangle \varphi_s \mid s \in S \} \cup \{ \neg \langle f \rangle \mid f \in F \} \right)$$

for finite index sets S and $F \subseteq \mathcal{A}$.

The idea here is that an observation is a tree of possibilities, where each node has a set of successor observations S, guarded by actions, and a set of refusals F, actions that are impossible at that point. Think of a program with which you interact, where you can undo actions to explore other paths, and where you always see which actions are possible/impossible at the specific state.

Definition 14 (Ready simulation observations)**.** For *ready simulation*, we define $[\![p]\!]_{\mathrm{RS}}$ as the least set of observations derivable by:

$$\frac{F \subseteq \mathcal{A} \setminus \mathrm{En}(p) \qquad S \text{ finite} \qquad \forall s \in S.\, p_s \in \mathrm{Der}(p, a_s) \wedge t_s \in [\![p_s]\!]_{\mathrm{RS}}}{\bigwedge \left(\{ \langle a_s \rangle t_s \mid s \in S \} \cup \{ \neg \langle f \rangle \mid f \in F \} \right) \in [\![p]\!]_{\mathrm{RS}}}$$

Here S indexes positive branch *occurrences*; different indices may carry the same action label, so sibling branches with the same action are allowed.

Observe that ready simulation observations contain all information needed to describe trace preorder of Section 2 (if $F = \emptyset$ and there is at most one positive branch occurrence at each node). If we identify unary conjunction $\bigwedge\{o\}$ with the observation o itself and the empty conjunction $\bigwedge \emptyset$ with $\top$, then we have identical objects. So we consider *trace observations* $\mathsf{Obs}_{\mathrm{Tr}}$ to equal the set of action sequences ending in $\top$ of the form $\bigwedge\{\langle a_1 \rangle \cdots \bigwedge\{\langle a_n \rangle \top\} \cdots\}$ (equivalently: with at most one positive branch occurrence, and no negative tests, at each node).

One can define more notions of equivalence by filtering for certain subsets of ready simulation differences:

Definition 15 (Subsets of ready simulation observations)**.**

- Let $\mathsf{Obs}_{\mathrm{S}}$ denote the set of *simulation observations* with $F = \emptyset$ throughout (no negations anywhere).
- Let $\mathsf{Obs}_{\mathrm{F}}$ denote the set of *failure observations* with at most one positive branch occurrence at each node, and with $F = \emptyset$ whenever a positive branch occurrence is present.

These induce preorders and differences analogously to Section 2:

$$\Delta_{\mathrm{RS}}(p, Q) := [\![p]\!]_{\mathrm{RS}} \setminus \bigcup_{q \in Q} [\![q]\!]_{\mathrm{RS}}$$

and define, for $N \in \{\mathrm{S}, \mathrm{F}\}$:

$$\Delta_N(p, Q) := \Delta_{\mathrm{RS}}(p, Q) \cap \mathsf{Obs}_N, \qquad p \preceq_N q :\Longleftrightarrow\ \Delta_N(p, \{q\}) = \emptyset.$$

Note that we align with the old definition of $\Delta_{\mathrm{Tr}}(\cdot, \cdot)$.

Example 16 (Simulation vs. failures on the running example)**.** We now return to the processes $\mathsf{P_A} = a.\mathsf{P_A} + a.b.0$ and $\mathsf{P_B} = a.(\mathsf{P_B} + b.0)$ of Example 4, and note that they are incomparable with respect to simulation and failures:

- **Simulation side:** $\mathsf{P_A} \preceq_{\mathrm{S}} \mathsf{P_B}$, but not conversely. A distinguishing simulation observation for the opposite direction is $\bigwedge\{\langle a\rangle \bigwedge\{\langle a\rangle\top, \langle b\rangle\top\}\}$ ("after a, require both a and b").[14]
- **Failure side:** $\mathsf{P_B} \preceq_{\mathrm{F}} \mathsf{P_A}$, but not conversely. A distinguishing failure observation for the opposite direction is $\bigwedge\{\langle a\rangle\ \bigwedge\{\neg\langle b\rangle\}\}$ ("after a, refuse b").[15]

4.2 Collecting Ready Simulation Differences

To match the trace development, we make the concrete computation explicit at transformer level.

Definition 17 (Concrete predecessor transformer for ready-simulation differences)**.** Let

$$DS_{RS} := (\mathsf{CCS} \times 2^{\mathsf{CCS}}) \to 2^{\mathsf{Obs}_{\mathrm{RS}}}$$

be the space of ready-simulation difference systems, ordered pointwise by inclusion:

$$\delta_1 \sqsubseteq \delta_2 :\Longleftrightarrow\ \forall (p, Q).\, \delta_1(p, Q) \subseteq \delta_2(p, Q).$$

Define the concrete predecessor transformer[16]

$$D_{\mathrm{RS}} \colon DS_{RS} \to DS_{RS}$$

pointwise by

$$\begin{aligned}
&\textstyle\bigwedge(\{\langle a_s\rangle u_s \mid s \in S\} \cup \{\neg\langle f\rangle \mid f \in F\}) \ \in\ D_{\mathrm{RS}}(\delta)(p, Q)\\
&\quad :\Longleftrightarrow\ \exists S \text{ finite},\ (a_s)_{s\in S},\ (u_s)_{s\in S},\ F \subseteq \mathcal{A} \setminus \mathrm{En}(p),\ Q_F \subseteq Q,\ (Q_s \subseteq Q)_{s\in S}.\\
&\qquad \wedge\ Q \subseteq Q_F \cup \bigcup_{s\in S} Q_s\\
&\qquad \wedge\ \forall q \in Q_F.\, F \cap \mathrm{En}(q) \neq \emptyset\\
&\qquad \wedge\ \forall s \in S.\, \exists p_s \in \mathrm{Der}(p, a_s),\ u_s \in \delta(p_s, \mathrm{Der}(Q_s, a_s)).
\end{aligned}$$

This is the direct RS analogue of D_{Tr}, but with two important differences. First, an important coherence condition for conjunctions: the right-hand states in Q may be split across different local failure reasons. A conjunctive observation

[14] Lean theorem: Ready.RunningExample.abstractRSDiffExact_PB_PA_at_S
[15] Lean theorem: Ready.RunningExample.abstractRSDiffExact_PA_PB_at_F
[16] Lean def: Ready.DRS

distinguishes p from Q when every right-hand state is ruled out by at least one chosen branch, not necessarily the same one. Second, each positive branch s passes only its local competitor set $Q_s \subseteq Q$ (not all of Q) into the recursive call $\delta(p_s, \mathrm{Der}(Q_s, a_s))$; the trace case is recovered by taking $Q_s = Q$. Crucially, the positive side is indexed by branch occurrences rather than action labels, so different siblings may carry the same action.

Proposition 18 (Concrete ready-simulation difference as least fixpoint)**.** *For all $p \in \mathsf{CCS}$ and $Q \subseteq \mathsf{CCS}$,*[17]

$$\Delta_{\mathrm{RS}}(p, Q) = \mathrm{lfp}(D_{\mathrm{RS}})(p, Q).$$

As in the trace case, this least-fixpoint characterization is the concrete basis for the abstract-interpretation step in Section 5.

5 Checking Hierarchies of Equivalences in One Abstract Interpretation

This section explains how to decide trace, simulation, failure, and ready-simulation equivalences in a single abstract interpretation by overlaying a more refined abstract domain over the ready-simulation observations of Section 4. Structurally, it mirrors Section 3.

5.1 Abstract-interpretation Components and Capability Domain

Let us make explicit the abstract-interpretation ingredients, the backward-complete transformer, and the correctness statement.

Definition 19 (Capability lattice for strong observations)**.** Let

$$\mathsf{RS} := \{\mathsf{sim}, \mathsf{fail}\}.$$

An observation capability is a subset $c \subseteq \mathsf{RS}$, ordered by inclusion. So we obtain the three more core capability combinations:

$$\begin{aligned} \mathsf{T} &:= \emptyset \\ \mathsf{S} &:= \{\mathsf{sim}\} \\ \mathsf{F} &:= \{\mathsf{fail}\}. \end{aligned}$$

Hence $(2^{\mathsf{RS}}, \subseteq)$ forms a diamond lattice (see Figure 3) with $\mathsf{T} \subseteq \mathsf{S} \subseteq \mathsf{RS}$ and $\mathsf{T} \subseteq \mathsf{F} \subseteq \mathsf{RS}$, while S and F are incomparable.

The abstract interpretation will, for each pair (p, Q), record the collection of all *minimal* capabilities that suffice to distinguish p from each process in Q. So the abstract value has the shape $\mathcal{D}(p, Q) \subseteq 2^{\mathsf{RS}}$. This must be a set rather than a single capability because the same pair (p, Q) may be distinguishable in different incomparable ways, for instance both with capability S and with capability F. We only store the minimal sufficient capabilities, i.e. antichains in $(2^{\mathsf{RS}}, \subseteq)$:

[17] Lean theorem: Ready.rsDifferenceToSet_eq_lfpDRS

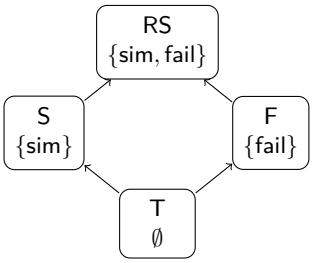

Figure 3: Capability lattice of observation strengths.

Definition 20 (Abstract interpretation for ready-simulation differences)**.** We instantiate the abstract-interpretation framework for ready-simulation differences as follows.

Concrete and abstract domains.

$$C := 2^{\mathsf{Obs}_{\mathrm{RS}}}, \qquad \sqsubseteq_C := \subseteq,$$

$$A := \{X \subseteq 2^{\mathsf{RS}} \mid \forall c, d \in X.\, c \subseteq d \Rightarrow c = d\},$$

$$X \sqsubseteq_A Y \iff \uparrow X \subseteq \uparrow Y, \qquad \uparrow X := \{d \subseteq \mathsf{RS} \mid \exists c \in X.\, c \subseteq d\}.$$

Abstraction and concretization. Define $\mathsf{req}(o)$, the least capability required to express o as:

$$\mathsf{req}\left(\bigwedge(\{\langle a_s\rangle u_s \mid s \in S\} \cup \{\neg\langle f\rangle \mid f \in F\})\right) := \mathsf{req}_{\mathsf{shape}}(S, F) \cup \bigcup_{s \in S} \mathsf{req}(u_s)$$

$$\text{where } \mathsf{req}_{\mathsf{shape}}(S, F) := \{\mathsf{sim} \mid 1 < |S|\} \cup \{\mathsf{fail} \mid F \neq \emptyset\}.$$

Then, define abstraction and concretization by

$$\alpha_{cap}(X) := \{c \subseteq \mathsf{RS} \mid X \cap \mathsf{Obs}_c \neq \emptyset \wedge \nexists d \subset c.\, X \cap \mathsf{Obs}_d \neq \emptyset\},$$

$$\gamma_{cap}(Y) := \{o \in \mathsf{Obs}_{\mathrm{RS}} \mid \exists c \in Y.\, c \subseteq \mathsf{req}(o)\},$$

where the *capability-threshold fragment* Obs_c is defined by

$$\mathsf{Obs}_c := \{o \in \mathsf{Obs}_{\mathrm{RS}} \mid \mathsf{req}(o) \subseteq c\}.$$

Concrete and abstract transformers. As concrete transformer we already have D_{RS} from Definition 17. We accompany it by an abstract transformer: Given an abstract difference system $\hat{\delta} \colon (\mathsf{CCS} \times 2^{\mathsf{CCS}}) \to \mathcal{A}$, define the candidate

capability set

$$U_{p,Q}(\hat{\delta}) := \Big\{ c \subseteq \mathsf{RS} \;\Big|\; \exists S \text{ finite},\ (a_s)_{s\in S},\ (u_s)_{s\in S},\ F \subseteq \mathcal{A} \setminus \mathrm{En}(p),\ Q_F \subseteq Q,\ (Q_s \subseteq Q)_{s\in S},\ (c_s)_{s\in S}.$$
$$\wedge\ Q \subseteq Q_F \cup \bigcup_{s\in S} Q_s$$
$$\wedge\ \forall q \in Q_F.\, F \cap \mathrm{En}(q) \neq \emptyset$$
$$\wedge\ \forall s \in S.\, \exists p_s \in \mathrm{Der}(p, a_s).\, c_s \in \hat{\delta}(p_s, \mathrm{Der}(Q_s, a_s))$$
$$\wedge\ c_s \subseteq \mathsf{req}(u_s)$$
$$\wedge\ \mathsf{req}_{\mathsf{shape}}(S, F) \cup \bigcup_{s\in S} \mathsf{req}(u_s) \subseteq c \Big\}.$$

Let $\mathrm{Min}(X) := \{c \in X \mid \nexists d \in X.\, d \subset c\}$. Then define[18]

$$D^{\sharp}_{\mathrm{RS}}(\hat{\delta})(p, Q) := \mathrm{Min}(U_{p,Q}(\hat{\delta})).$$

Abstract interpretation. The abstract result is computed as the least fixpoint of the abstract transformer:

$$\mathcal{D}(\cdot,\cdot) := \mathrm{lfp}(D^{\sharp}_{\mathrm{RS}}).$$

The named observation sets of Definition 15 are exactly the four capability-threshold fragments at the lattice points of Definition 19:

$$\mathsf{Obs}_{\mathrm{Tr}} = \mathsf{Obs}_{\mathsf{T}}, \qquad \mathsf{Obs}_{\mathrm{S}} = \mathsf{Obs}_{\mathsf{S}}, \qquad \mathsf{Obs}_{\mathrm{F}} = \mathsf{Obs}_{\mathsf{F}}, \qquad \mathsf{Obs}_{\mathrm{RS}} = \mathsf{Obs}_{\mathsf{RS}}.$$

This is immediate from the definitions: $\mathsf{req}(o) \subseteq \mathsf{T} = \emptyset$ iff o uses no sim/fail features (i.e. $o \in \mathsf{Obs}_{\mathrm{Tr}}$), $\mathsf{req}(o) \subseteq \mathsf{S}$ iff o uses no failure features (i.e. $o \in \mathsf{Obs}_{\mathrm{S}}$), and so on.

The aim of the abstract interpretation definition is to capture for each notion $N \in \{\mathsf{T}, \mathsf{S}, \mathsf{F}, \mathsf{RS}\}$, that the pair (p, Q) is distinguishable under N iff $\exists c \in \mathcal{D}(p, Q).\, c \subseteq N$. Crucially, the antichain representation does not force premature combination: if one can distinguish both by a simulation observation and by a failure observation, we may have $\mathcal{D}(p, Q) = \{\mathsf{S}, \mathsf{F}\}$, which does *not* collapse to $\{\mathsf{RS}\}$. This preserves that either capability alone already suffices.

Example 21 (Abstract capability differences for the running example)**.** The capability transformer stabilizes after two steps (failure side) and three steps (simulation side) on the relevant pairs:

$$\mathcal{D}(\mathsf{P_A}, \{\mathsf{P_B}\}) = \{\mathsf{F}\}, \qquad \mathcal{D}(\mathsf{P_B}, \{\mathsf{P_A}\}) = \{\mathsf{S}\}.$$

Failure side.[19] First, $\hat{\delta}_1(b.0, \{\mathsf{P_B} + b.0\}) = \{\mathsf{F}\}$, by taking $S = \emptyset$, $F = \{a\}$, $Q_F = \{\mathsf{P_B} + b.0\}$, so $\mathsf{req}_{\mathsf{shape}}(S, F) = \mathsf{F}$ and there are no child capabilities to add.

[18] Lean def: Ready.bestDRSExplicit

[19] Lean theorem: Ready.RunningExample.lfpDRSAbsExactCanon_PA_PB_eq_singleton_F

Then one predecessor step with singleton branch family $S = \{1\}$, action label $a_1 = a$, and $Q_1 = \{\mathsf{P_B}\}$ only propagates that child, because $\mathsf{req}_{\mathsf{shape}}(\{1\}, \emptyset) = \mathsf{T}$. Hence $\hat{\delta}_2(\mathsf{P_A}, \{\mathsf{P_B}\}) = \{\mathsf{F}\}$.

Simulation side.[20] At step 1, the empty-Q base case gives $\hat{\delta}_1(p, \emptyset) = \{\mathsf{T}\}$ for all p. Using these as child values, $\hat{\delta}_2(\mathsf{P_B} + b.0, \{\mathsf{P_A}, b.0\}) = \{\mathsf{S}\}$, by taking $S = \{1, 2\}$, action labels $a_1 = a$ and $a_2 = b$, $F = \emptyset$, $Q_1 = \{b.0\}$, $Q_2 = \{\mathsf{P_A}\}$; both children have empty competitor sets after derivatives, so they contribute only T from step 1, and the branching shape forces S. One more singleton predecessor step with $S = \{1\}$, $a_1 = a$, and $Q_1 = \{\mathsf{P_A}\}$ preserves that child capability, since again $\mathsf{req}_{\mathsf{shape}}(\{1\}, \emptyset) = \mathsf{T}$. Hence $\hat{\delta}_3(\mathsf{P_B}, \{\mathsf{P_A}\}) = \{\mathsf{S}\}$.

Lemma 22 (Backward completeness for the capability abstraction)**.** *The abstract interpretation is backward complete:*[21]

$$\alpha_{cap} \circ D_{\mathrm{RS}} = D^{\sharp}_{\mathrm{RS}} \circ \alpha_{cap}.$$

Proof in Appendix A.

As in the trace section, the least-fixpoint consequence is an exact abstraction result: As $\mathrm{lfp}(D^{\sharp}_{\mathrm{RS}}) = \alpha_{cap}(\mathrm{lfp}(D_{\mathrm{RS}}))$ pointwise, the abstract fixpoint retains exactly the minimal capabilities of the concrete ready-simulation difference.

The unified abstract interpretation now follows the same completeness pattern as Theorem 13 for traces:

Theorem 23 (Exactness for capability-threshold checking)**.** *For all $p \in \mathit{CCS}$, $Q \subseteq \mathit{CCS}$, and $N \in \{\mathsf{T}, \mathsf{S}, \mathsf{F}, \mathsf{RS}\}$,*[22]

$$\exists c \in \mathrm{lfp}(D^{\sharp}_{\mathrm{RS}})(p, Q).\, c \subseteq N \iff \Delta_{\mathrm{RS}}(p, Q) \cap \mathsf{Obs}_N \neq \emptyset.$$

Proof in Appendix A.

Thus *one* least-fixpoint in the abstract domain contains the information to check for all four preorders.

6 Conclusion and Related Work

With this paper, we have shown how to uniformly check trace, simulation, failure, and ready-simulation preorders under a single abstract-interpretation framework. The abstract domain captures what kinds of observations can distinguish the compared processes, thereby modeling a kind of *abstract difference*.

Ranzato and Tapparo [13, 14, 15] offer a closely related line of work: They show how partition refinement algorithms for bisimulation and simulation equivalence can be justified as *greatest* fixed points in *forward-complete* abstract

[20] Lean theorem: Ready.RunningExample.lfpDRSAbsExactCanon_PB_PA_eq_singleton_S

[21] Lean theorem: Ready.backwardComplete_bestDRS

[22] Lean theorem: Ready.thresholdWitness_lfpBestDRS_iff_rsDifferenceThreshold

interpretations. Our approach is slightly more general in that we can handle ranges of preorders in one algorithm. There is a clear duality with us using *least* fixed points and *backward* completeness.

The idea of addressing hierarchies of equivalences in one pass is explored in Bisping [2]. Its underlying game-theoretic publications are related in a way resembling the abstraction link in this paper: Bisping, Jansen, and Nestmann [4] collect specific distinguishing formulas, while Bisping [3] abstracts them into energy vectors. The potential merit of lifting generalized equivalence into an abstract-interpretation framework is that similar algorithms might become derivable almost automatically, for instance by applying approaches for the synthesis of abstract transformers like Kalita *et al.* [9]. This paper does not use game models, but these (e.g. also Tan [17], Shukla, Hunt III, and Rosenkrantz [16]) should usually offer more efficient algorithms to compute the relevant fixpoints than naive iteration.

On the more applied side of things, Partush and Yahav [12] and Delmas and Miné [6] explore how to compute abstract differences of numerical programs starting from "correlating programs," where differences of variables are tracked. Their state-based view differs fundamentally from ours, which is observation-based, but the underlying idea of abstracting differences is similar.

A perk of the abstract-interpretation perspective is that it provides a starting point for algorithms that over-approximate the differences. Such approximations are useful for security queries [e.g. 8] on infinite-state systems.

Galois connections have appeared in several recent papers on behavioral equivalences [1, 2, 10], suggesting fundamental links between the topics of abstract interpretation and behavioral equivalences. This paper hopes to take a few steps towards further uncovering these.

References

1. Beohar, H., Gurke, S., König, B., Messing, K.: Hennessy-Milner Theorems via Galois Connections. In: Klin, B., Pimentel, E. (eds.) 31st EACSL Annual Conference on Computer Science Logic (CSL 2023), 12:1–12:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2023). https://doi.org/10.4230/LIPIcs.CSL.2023.12
2. Bisping, B.: Generalized Equivalence Checking of Concurrent Programs. PhD thesis, Technische Universität Berlin (2025). https://doi.org/10.14279/depositonce-24813. https://generalized-equivalence-checking.equiv.io/.
3. Bisping, B.: Process Equivalence Problems as Energy Games. In: Enea, C., Lal, A. (eds.) Computer Aided Verification (CAV 2023), pp. 85–106. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-37706-8_5
4. Bisping, B., Jansen, D.N., Nestmann, U.: Deciding All Behavioral Equivalences at Once: A Game for Linear-Time–Branching-Time Spectroscopy. Logical Methods in Computer Science **18**(3) (2022). https://doi.org/10.46298/lmcs-18(3:19)2022

5. Cousot, P., Cousot, R.: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In: Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 1977), pp. 238–252. ACM (1977). https://doi.org/10.1145/512950.512973
6. Delmas, D., Miné, A.: Analysis of Software Patches Using Numerical Abstract Interpretation. In: Chang, B.-Y.E. (ed.) Static Analysis (SAS 2019), pp. 225–246. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-32304-2_12
7. Hennessy, M., Milner, R.: On observing nondeterminism and concurrency. In: *LNCS*. Vol. 85: Automata, Languages and Programming. Ed. by J.W. de Bakker and J. van Leeuwen, pp. 299–309. Springer, Berlin (1980). https://doi.org/10.1007/3-540-10003-2_79
8. Horne, R., Mauw, S., Yurkov, S.: When privacy fails, a formula describes an attack: A complete and compositional verification method for the applied π-calculus. Theoretical Computer Science **959**, 113842 (2023). https://doi.org/10.1016/j.tcs.2023.113842
9. Kalita, P.K., Muduli, S.K., D'Antoni, L., Reps, T., Roy, S.: Synthesizing abstract transformers. Proc. ACM Program. Lang. **6**(OOPSLA2) (2022). https://doi.org/10.1145/3563334
10. König, B., Messing, K.: Witnesses for Fixpoint Games on Lattices, (2026). https://doi.org/10.48550/arXiv.2603.11908. Preprint.
11. Milner, R.: A calculus of communicating systems. Springer (1980)
12. Partush, N., Yahav, E.: Abstract Semantic Differencing for Numerical Programs. In: Logozzo, F., Fähndrich, M. (eds.) Static Analysis (SAS 2013), pp. 238–258. Springer Berlin Heidelberg, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38856-9_14
13. Ranzato, F., Tapparo, F.: An efficient simulation algorithm based on abstract interpretation. Information and Computation **208**(1), 1–22 (2010). https://doi.org/10.1016/j.ic.2009.06.002
14. Ranzato, F., Tapparo, F.: Generalized Strong Preservation by Abstract Interpretation. Journal of Logic and Computation **17**(1), 157–197 (2007). https://doi.org/10.1093/logcom/exl035
15. Ranzato, F., Tapparo, F.: Generalizing the Paige–Tarjan algorithm by abstract interpretation. Information and Computation **206**(5), 620–651 (2008). https://doi.org/10.1016/j.ic.2008.01.001
16. Shukla, S.K., Hunt III, H.B., Rosenkrantz, D.J.: HORNSAT, Model Checking, Verification and Games: extended abstract. In: *LNCS*. Vol. 1102: Computer Aided Verification (CAV 1996). Ed. by R. Alur and T.A. Henzinger, pp. 99–110. Springer, Berlin (1996). https://doi.org/10.1007/3-540-61474-5_61
17. Tan, L.: An Abstract Schema for Equivalence-Checking Games. In: Cortesi, A. (ed.) Verification, Model Checking, and Abstract Interpretation, Third International Workshop, (VMCAI 2002), Revised Papers, pp. 65–78. Springer (2002). https://doi.org/10.1007/3-540-47813-2_5
18. van Glabbeek, R.J.: The linear time–branching time spectrum: extended abstract. In: *LNCS*. Vol. 458: Theories of Concurrency: Unification and Extension (CONCUR 1990). Ed. by J.C.M. Baeten and J.W. Klop, pp. 278–297. Springer, Berlin (1990). https://doi.org/10.1007/BFb0039066

A Paper-style Proofs

For convenience, we here provide "paper-style" versions of proofs for the facts that appear througout the paper. Links to the corresponding Lean formalizations can be found in the footnotes of the main paper body.

Proof of Proposition 9

Proof. We show mutual inclusion between $\Delta_{\mathrm{Tr}}(p, Q)$ and $\mathrm{lfp}(D_{\mathrm{Tr}})(p, Q)$ by taking $\mathrm{lfp}(D_{\mathrm{Tr}})$ to be the intersection of all pre-fixpoints δ satisfying $D_{\mathrm{Tr}}(\delta) \sqsubseteq \delta$. (This works thanks to the Knaster-Tarski theorem: On the complete lattice $(DS, \sqsubseteq)$ with monotone endofunction D_{Tr}, the least fixpoint coincides with the greatest lower bound (intersection) of all pre-fixpoints.)

$\Delta_{\mathrm{Tr}}(\cdot,\cdot)$ **is contained in every pre-fixpoint.** Let $\delta \in DS$ with $D_{\mathrm{Tr}}(\delta) \sqsubseteq \delta$. We show $\Delta_{\mathrm{Tr}}(p, Q) \subseteq \delta(p, Q)$ for all (p, Q) simultaneously, by induction on the derivation of $t \in [\![p]\!]_{\mathrm{Tr}}$, with Q universally quantified in the induction hypothesis (so that in the step case the IH can be applied with the shifted set $\mathrm{Der}(Q, a)$).

- Base $t = \top$: if $\top \in \Delta_{\mathrm{Tr}}(p, Q)$ then $Q = \emptyset$, so, by definition of D_{Tr}, the proposition $D_{\mathrm{Tr}}(\delta)(p, \emptyset)(\top)$ holds, and thus $\top \in \delta(p, \emptyset)$.
- Step $t = \langle a\rangle u$: if $\langle a\rangle u \in \Delta_{\mathrm{Tr}}(p, Q)$, then there is $p' \in \mathrm{Der}(p, a)$ with $u \in \Delta_{\mathrm{Tr}}(p', \mathrm{Der}(Q, a))$. By induction hypothesis, $u \in \delta(p', \mathrm{Der}(Q, a))$. Hence $D_{\mathrm{Tr}}(\delta)(p, Q)(\langle a\rangle u)$ holds, so $\langle a\rangle u \in \delta(p, Q)$.

Therefore $\Delta_{\mathrm{Tr}}(\cdot,\cdot) \sqsubseteq \mathrm{lfp}(D_{\mathrm{Tr}})$.

$\Delta_{\mathrm{Tr}}(\cdot,\cdot)$ **is itself a pre-fixpoint.** We show $D_{\mathrm{Tr}}(\Delta_{\mathrm{Tr}}(\cdot,\cdot))(p, Q) \subseteq \Delta_{\mathrm{Tr}}(p, Q)$ for all (p, Q).

- Base $t = \top$: $D_{\mathrm{Tr}}(\Delta_{\mathrm{Tr}}(\cdot,\cdot))(p, Q)(\top)$ asserts $Q = \emptyset$, in which case $\Delta_{\mathrm{Tr}}(p, \emptyset) = [\![p]\!]_{\mathrm{Tr}} \ni \top$.
- Step $t = \langle a\rangle u$: D_{Tr} gives $p' \in \mathrm{Der}(p, a)$ with $u \in \Delta_{\mathrm{Tr}}(p', \mathrm{Der}(Q, a))$, i.e. $u \in [\![p']\!]_{\mathrm{Tr}}$ and $u \notin [\![q']\!]_{\mathrm{Tr}}$ for all $q' \in \mathrm{Der}(Q, a)$. Hence $\langle a\rangle u \in [\![p]\!]_{\mathrm{Tr}}$ and $\langle a\rangle u \notin [\![q]\!]_{\mathrm{Tr}}$ for all $q \in Q$, so $\langle a\rangle u \in \Delta_{\mathrm{Tr}}(p, Q)$.

Combining both parts, $\mathrm{lfp}(D_{\mathrm{Tr}}) \sqsubseteq \Delta_{\mathrm{Tr}}(\cdot,\cdot) \sqsubseteq \mathrm{lfp}(D_{\mathrm{Tr}})$, gives $\Delta_{\mathrm{Tr}}(p, Q) = \mathrm{lfp}(D_{\mathrm{Tr}})(p, Q)$. □

Proof of Lemma 12

Proof. Fix an arbitrary valuation $\delta \in DS$ and pair (p, Q). We show $\alpha(D_{\mathrm{Tr}}(\delta))(p, Q) = D^{\sharp}_{\mathrm{Tr}}(\alpha(\delta))(p, Q)$ by case analysis on Q.

- **Case $Q = \emptyset$:** By the base clause of Definition 8, $\top \in D_{\mathrm{Tr}}(\delta)(p, \emptyset)$, so $D_{\mathrm{Tr}}(\delta)(p, \emptyset) \neq \emptyset$ and hence $\alpha(D_{\mathrm{Tr}}(\delta))(p, \emptyset) = \{\mathsf{T}\}$. On the other side, $D^{\sharp}_{\mathrm{Tr}}(\alpha(\delta))(p, \emptyset) = \{\mathsf{T}\}$ by the $Q = \emptyset$ clause of Definition 10. Both sides equal $\{\mathsf{T}\}$.

– **Case $Q \neq \emptyset$:** On the left, $\alpha(D_{\mathrm{Tr}}(\delta))(p,Q) = \{\mathsf{T}\}$ iff $D_{\mathrm{Tr}}(\delta)(p,Q) \neq \emptyset$ iff (by the step clause) there is $a \in \mathcal{A}$ with $p' \in \mathrm{Der}(p,a)$ and $\delta(p', \mathrm{Der}(Q,a)) \neq \emptyset$, i.e., $\alpha(\delta)(p', \mathrm{Der}(Q,a)) = \{\mathsf{T}\}$. On the right, $D^\sharp_{\mathrm{Tr}}(\alpha(\delta))(p,Q) = \bigcup_a \bigcup_{p' \in \mathrm{Der}(p,a)} \alpha(\delta)(p', \mathrm{Der}(Q,a))$ by the $Q \neq \emptyset$ clause, and this set contains T under the same condition. Hence both sides agree. □

Proof of Theorem 13

Proof. The key step is Lemma 12: backward completeness, $\alpha \circ D_{\mathrm{Tr}} = D^\sharp_{\mathrm{Tr}} \circ \alpha$. Using this, $\alpha(\mathrm{lfp}(D_{\mathrm{Tr}}))$ is itself an abstract pre-fixpoint of $D^\sharp_{\mathrm{Tr}}$,[23] so the abstract lfp (as smallest pre-fixpoint) satisfies $\mathrm{lfp}(D^\sharp_{\mathrm{Tr}}) \leq \alpha(\mathrm{lfp}(D_{\mathrm{Tr}}))$. Moreover, every abstract pre-fixpoint $\hat{X}$ yields (via γ) a concrete pre-fixpoint of D_{Tr} (using backward completeness and $\alpha \circ \gamma = \mathrm{id}_A$, which holds since $\alpha(\emptyset) = \emptyset$ and $\alpha(\mathsf{Obs}_{\mathrm{Tr}}) = \{\mathsf{T}\}$), so $\alpha(\mathrm{lfp}(D_{\mathrm{Tr}}))$ lies below every abstract pre-fixpoint,[24] giving $\alpha(\mathrm{lfp}(D_{\mathrm{Tr}})) \leq \mathrm{lfp}(D^\sharp_{\mathrm{Tr}})$. Therefore the abstract least fixpoint coincides pointwise with α of the concrete least fixpoint.[25]

Combining this with the previous identification of marker analysis with the abstract lfp gives[26]

$$\mathsf{T} \in \mathrm{lfp}(D^\sharp_{\mathrm{Tr}})(p,Q) \iff \alpha(\mathrm{lfp}(D_{\mathrm{Tr}}))(p,Q) = \{\mathsf{T}\}.$$

Unfolding α (which maps non-empty sets to $\{\mathsf{T}\}$ and $\emptyset$ to $\emptyset$)[27] gives

$$\mathsf{T} \in \mathcal{D}_{\mathrm{Tr}}(p,Q) \iff \mathrm{lfp}(D_{\mathrm{Tr}})(p,Q) \neq \emptyset.$$

By Proposition 9, $\Delta_{\mathrm{Tr}}(p,Q) = \mathrm{lfp}(D_{\mathrm{Tr}})(p,Q)$, hence

$$\mathsf{T} \in \mathcal{D}_{\mathrm{Tr}}(p,Q) \iff \Delta_{\mathrm{Tr}}(p,Q) \neq \emptyset.$$ □

Proof of Lemma 22

Proof. Fix a valuation $\delta\colon (\mathsf{CCS} \times 2^{\mathsf{CCS}}) \to 2^{\mathsf{Obs}_{\mathrm{RS}}}$ and a pair (p,Q). An observation in $D_{\mathrm{RS}}(\delta)(p,Q)$ is generated by choosing an indexed family of positive branch occurrences S, action labels $(a_s)_{s \in S}$, a negative side $F \subseteq \mathcal{A} \setminus \mathrm{En}(p)$, a refusal side $Q_F \subseteq Q$, branch-specific competitor sets $(Q_s \subseteq Q)_{s \in S}$ covering Q, and for each $s \in S$ a branch witness $u_s \in \delta(p_s, \mathrm{Der}(Q_s, a_s))$. The resulting observation has required capability

$$\mathsf{req}_{\mathsf{shape}}(S,F) \cup \bigcup_{s \in S} \mathsf{req}(u_s).$$

In particular, taking $c_s := \mathsf{req}(u_s)$ (the exact minimal child capability, which lies in $\alpha_{cap}(\delta)$ since $u_s \in \delta \cap \mathsf{Obs}_{\mathsf{req}(u_s)}$) satisfies the $c_s \subseteq \mathsf{req}(u_s)$ condition of $U_{p,Q}$.

[23] Lean theorem: Trace.alpha_lfpDTr_is_prefixpoint
[24] Lean theorem: Trace.alpha_lfpDTr_le_of_abstract_prefixpoint
[25] Lean theorem: Trace.lfpDTrSharp_iff_alpha_lfpDTr
[26] Lean theorem: Trace.markerPresence_iff_alpha_lfpDTr
[27] Lean theorem: Trace.markerPresence_iff_lfpDTr_nonempty

Passing from δ to $\alpha_{cap}(\delta)$ therefore loses no information relevant to capability thresholds: it forgets the concrete branch observations u_s but keeps enough child-capability information to reconstruct the same threshold behavior. The pointwise definition of $D^\sharp_{\mathrm{RS}}$ reconstructs exactly the set of such capability combinations, and the final pruning step removes non-minimal supersets. But this is exactly what α_{cap} does when applied to the concrete set $D_{\mathrm{RS}}(\delta)(p, Q)$: it extracts the minimal capabilities of all concrete distinguishing observations generated in one step.

Hence, pointwise in (p, Q),

$$\alpha_{cap}(D_{\mathrm{RS}}(\delta))(p, Q) = D^\sharp_{\mathrm{RS}}(\alpha_{cap}(\delta))(p, Q). \qquad \square$$

Proof of Theorem 23

Proof. By backward completeness (Lemma 22),

$$\mathrm{lfp}(D^\sharp_{\mathrm{RS}}) = \alpha_{cap}(\mathrm{lfp}(D_{\mathrm{RS}}))$$

pointwise. By Proposition 18, the concrete side is $\Delta_{\mathrm{RS}}(p, Q) = \mathrm{lfp}(D_{\mathrm{RS}})(p, Q)$. So it remains only to unfold α_{cap}: for any concrete observation set $X \subseteq \mathsf{Obs}_{\mathrm{RS}}$ and threshold N,

$$\exists c \in \alpha_{cap}(X).\, c \subseteq N \iff X \cap \mathsf{Obs}_N \neq \emptyset,$$

because $\alpha_{cap}(X)$ stores exactly the minimal capabilities under which observations in X occur. Applying this to $X = \Delta_{\mathrm{RS}}(p, Q)$ gives the displayed equivalence. The singleton corollary is the case $Q = \{q\}$. $\square$