DECITING ALL BEHAVIORAL EQUIVALENCES AT ONCE: A GAME FOR LINEAR-TIME–BRANCHING-TIME SPECTROSCOPY [*]

BENJAMIN BISPING [a], DAVID N. JANSEN [b,c], AND UWE NESTMANN [a]

[a] Technische Universität Berlin, Berlin, Germany
  e-mail address: {benjamin.bisping, uwe.nestmann}@tu-berlin.de

[b] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
  e-mail address: dnjansen@ios.ac.cn

[c] University of Chinese Academy of Sciences, Beijing, China

ABSTRACT. We introduce a generalization of the bisimulation game that finds distinguishing Hennessy–Milner logic formulas from every finitary, subformula-closed language in van Glabbeek's linear-time–branching-time spectrum between two finite-state processes. We identify the relevant dimensions that measure expressive power to yield formulas belonging to the coarsest distinguishing behavioral preorders and equivalences; the compared processes are equivalent in each coarser behavioral equivalence from the spectrum. We prove that the induced algorithm can determine the best fit of (in)equivalences for a pair of processes.

1. INTRODUCTION

Have you ever looked at two system models and wondered what would be the finest notions of behavioral equivalence to equate them—or, conversely: the coarsest ones to distinguish them? We often run into this situation when analyzing models and, especially, when devising examples for teaching. We then find ourselves fiddling around with whiteboards and various tools, each implementing different equivalence checkers. Would it not be nice to *decide all equivalences at once?*

Example 1.1. Consider the CCS process $P_1 = a.(b+c)+a.d$, shown in Figure 1. It describes a machine that can be activated (a) and then either is in a state where one can choose from b and c or where it can only be deactivated again (d). P_1 shares a lot of properties with $P_2 = a.(b+d)+a.(c+d)$. For example, they have the same traces (and the same completed traces). Thus, they are *(completed) trace equivalent.*

But they also have differences. For instance, P_1 has a run where it executes a and then cannot do d, while P_2 does not have such a run. Hence, they are *not failure equivalent.* Moreover, P_1 may perform a and then choose from b and c, and P_2 cannot. This renders the two processes also *not simulation equivalent.* Failure equivalence and simulation equivalence

---

$$P_1$$

$a \swarrow \qquad \searrow a$

$b + c \qquad\qquad d$

$b, c \searrow \qquad \swarrow d$

0

$$P_2$$

$a \swarrow \qquad \searrow a$

$b + d \qquad\qquad c + d$
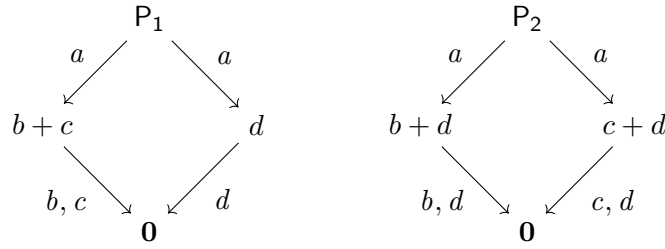
$b, d \searrow \qquad \swarrow c, d$

0

Figure 1: Processes for Example 1.1.

are incomparable—that is, neither one follows from the other one. *Both* are maximally coarse ways of telling the processes apart. Other inequivalences, like bisimulation inequivalence, are implied by both. Together, they make (completed) trace equivalence the finest notion to equate the processes.

In the following, we present a uniform game-based way of finding the most fitting notions of (in)equivalence for process pairs like in Example 1.1.

Our approach is based on the fact that notions of process equivalence can be characterized by two-player games. The defender's winning region in the game corresponds to pairs of equivalent states, and the attacker's winning strategies correspond to distinguishing formulas in Hennessy–Milner logic (HML).

Each notion of equivalence in van Glabbeek's famous linear-time–branching-time spectrum [vG90] can be characterized by a subset of HML with specific distinguishing power. Some of the notions are incomparable. So, often a process pair that is equivalent with respect to one equivalence is distinguished by a set of slightly coarser or incomparable equivalences, without any one of them alone being *the* coarsest way to distinguish the pair. As with the spectrum of light where a mix of wave lengths shows to us as a color, there is a "mix" of distinguishing capabilities involved in establishing whether a specific equivalence is finest. We present an algorithm that is meant to analyze what is in the mix.

Contributions. More precisely, this paper makes the following contributions:

- We *rechart the linear-time–branching-time spectrum of observation languages using "formula prices"* that capture six dimensions of expressive capabilities used in HML formulas (Subsection 2.4).
- We introduce a *special bisimulation game* that neatly characterizes the distinguishing formulas of HML for pairs of states in finite transition systems (Subsection 3.2).
- We show how to *enumerate the relevant distinguishing formulas* using the attacker's winning region in this game (Subsection 3.5).
- We define an algorithm that constructs a *finite set of distinguishing formulas guaranteed to contain observations of the weakest possible observation languages,* which can be seen as a "spectroscopy" of the differences between two processes (Subsection 3.6).
- We present a small *web tool that is able to run the algorithm on finite-state processes* and output a visual representation of the results (Subsection 4.1). We also report on the distinctions it finds for all the finitary examples from the report version of the linear-time–branching-time spectrum [vG01].
- Additionally, we quickly report on a *browser computer game based on the spectroscopy mechanics* (Subsection 4.2).

We frame the contributions by a roundtrip through the basics of HML, games and the spectrum (Section 2), a discussion of related work (Section 5), and concluding remarks on future lines of research (Section 6).

Compared to the original conference version of the paper [BN21], this paper includes an important correction of the spectroscopy game in Subsection 3.2. This allows us to now prove correctness of the algorithm in Subsection 3.7. Why the change has been necessary is discussed in Subsection 3.8. On the road to establishing the new correctness result, the journal version pays more attention to the fine points of how we assign formulas expressiveness prices. We are also glad to devote more space to presenting more examples, proofs and subalgorithms, and to also cover the enabledness preorder.

2. Preliminaries: HML, Games, and the Spectrum

We use the concepts of transition systems, games, observations, and notions of equivalence, largely due to the wake of Hennessy and Milner's seminal paper [HM80].

2.1. Transition Systems and Hennessy–Milner Logic. *Labeled transition systems* capture a discrete world view, where there is a current state and a branching structure of possible state changes to future states.

Definition 2.1 (Labeled transition system)**.** A *labeled transition system* (LTS) is a triple $(\mathcal{P}, \Sigma, \rightarrow)$ where $\mathcal{P}$ is the set of *states*, Σ the set of *actions*, and $\rightarrow \subseteq \mathcal{P} \times \Sigma \times \mathcal{P}$ the *transition relation.*

Figure 1 displays the labeled transition system for Example 1.1. In our examples, we use a small fragment of CCS to describe certain transition system states. This fragment is called BCCSP in [vG01]. It contains just *action prefixing* ($a.P$), *summation* ($P_1 + P_2$), and the *completed process* (**0**). (Note that LTSs are more powerful than the simple examples; in particular, our tool is also able to handle recursively defined CCS terms.) This fragment only needs two rules for its semantics:

(1) $a.P \xrightarrow{a} P$ for $P \in$ CCS and $a \in \Sigma$, and
(2) $P_1 + P_2 \xrightarrow{a} P_i'$ if there is $i \in \{1, 2\}$ such that $P_i \xrightarrow{a} P_i'$.

Silently assuming commutativity and associativity, we sometimes write $P_1 + P_2 + \cdots + P_n$.

Hennessy–Milner logic [HM80] describes *observations* (or "tests") on such a system.

Definition 2.2 (Hennessy–Milner logic)**.** Given an alphabet Σ, the syntax of *Hennessy–Milner logic* formulas, HML$[\Sigma]$, is inductively defined as follows:

Observations: If $\varphi \in$ HML$[\Sigma]$ and $a \in \Sigma$, then $\langle a \rangle \varphi \in$ HML$[\Sigma]$.
Conjunctions: If $\varphi_i \in$ HML$[\Sigma]$ for all i from an index set I, then $\bigwedge_{i \in I} \varphi_i \in$ HML$[\Sigma]$.
Negations : If $\varphi \in$ HML$[\Sigma]$, then $\neg \varphi \in$ HML$[\Sigma]$.

Intuitively, $\langle a \rangle \varphi$ means that one can observe a system transition labeled by a and then continue to make observation(s) φ. Conjunction and negation work as known from propositional logic. So, $\langle a \rangle \neg \langle d \rangle \mathsf{T}$ can be read as "one can observe a in such a way that afterwards one cannot observe a d." We will provide a common game semantics for HML in the following subsection.

We often write $\bigwedge \{\varphi_0, \varphi_1, \dots\}$ for $\bigwedge_{i \in I} \varphi_i$. T denotes $\bigwedge \varnothing$, the nil-element of the syntax tree, and $\langle a \rangle$ is a short-hand for $\langle a \rangle \mathsf{T}$. We also implicitly assume that formulas are flattened

in the sense that conjunctions do not contain other conjunctions as immediate subformulas. We will sometimes talk about the syntax tree height of a formula and consider the height of T to equal 0.

In principle, Definition 2.2 can also be read to be infinitary with respect to branching degree or recursion depth. Allowing infinite index sets I enables formulas like $\bigwedge_{n\in\mathbb{N}}\langle a\rangle^n$, "one can observe an arbitrary number of as." A coinductive reading makes formulas like $\langle a\rangle^\omega$, "one can observe an infinite sequence of as," possible. Note that the formulas both have infinite height. Obviously, this might add a lot of expressiveness to HML. For the scope of this paper, we are concerned with finite formulas only. The prices we are going to define do not distinguish properly between different infinitely branching or infinitely deep formulas.

2.2. **Game Semantics of HML.** Let us fix some notions for *Gale–Stewart-style reachability games* where the defender wins all infinite plays.

Definition 2.3 (Games). A *reachability game* $\mathcal{G}[g_0] = (G, G_{\mathrm{d}}, \rightarrowtail, g_0)$ is played on a directed graph consisting of

- a set of *game positions* (vertices) G, partitioned into
  - a set of *defender positions* $G_{\mathrm{d}} \subseteq G$
  - a set of *attacker positions* $G_{\mathrm{a}} := G \setminus G_{\mathrm{d}}$,
- a set of *game moves* (edges) $\rightarrowtail\, \subseteq G \times G$, and
- an *initial position* $g_0 \in G$.

Definition 2.4 (Plays and wins). We call the paths $g_0 g_1 \ldots \in G^\infty$ with $g_i \rightarrowtail g_{i+1}$ *plays* of $\mathcal{G}[g_0]$. They may be finite or infinite. The defender *wins* infinite plays. If a finite play $g_0 \ldots g_n \not\rightarrowtail$ is stuck, the stuck player loses: The defender wins if $g_n \in G_{\mathrm{a}}$, and the attacker wins if $g_n \in G_{\mathrm{d}}$.

Definition 2.5 (Strategies and winning strategies). A (positional, non-deterministic) *attacker strategy* is a subset of the moves starting in attacker states, $F \subseteq (G_{\mathrm{a}} \times G) \cap \rightarrowtail$. Similarly, a *defender strategy* is a subset of the moves starting in defender states, $F \subseteq (G_{\mathrm{d}} \times G) \cap \rightarrowtail$. If (fairly) picking elements of strategy F ensures a player to win, F is called a *winning strategy* for this player. The player with a winning strategy for $\mathcal{G}[g_0]$ is said to *win* $\mathcal{G}[g_0]$. If F is a function on G_{a} or G_{d}, respectively, we call it a *deterministic* strategy.

The games we use in this paper essentially are parity games only colored by 0. So they are positionally determined. This means, for each possible initial position, exactly one of the two players has a positional deterministic winning strategy F. We call this partitioning of the game positions the winning regions.

Definition 2.6 (Winning regions). The set $W_{\mathrm{a}} \subseteq G$ of all positions g where the attacker wins $\mathcal{G}[g]$ is called the *attacker winning region*. (The defender winning region W_{d} is defined analogously.)

It is well-known that winning regions of *finite* reachability games can be computed in linear time of the number of game moves. (An algorithm for this is discussed in Subsection 3.4.) This is why the *spectroscopy game* that we introduce in Subsection 3.2 can easily be used in algorithms. It derives from the following semantics game for HML, where the defender tries to prove a formula and the attacker tries to falsify it.

Definition 2.7 (HML game). For a transition system $\mathcal{S} = (\mathcal{P}, \Sigma, \rightarrow)$, the HML *game* $\mathcal{G}^{\mathcal{S}}_{\mathsf{HML}}[g_0] = (G, G_\mathrm{d}, \rightarrowtail, g_0)$ is played on $G = \mathcal{P} \times \mathsf{HML}[\Sigma]$, where the defender controls observations and negated conjunctions, that is $(p, \langle a \rangle \varphi) \in G_\mathrm{d}$ and $(p, \neg \bigwedge_{i \in I} \varphi_i) \in G_\mathrm{d}$ (for all φ, p, I), and the attacker controls the rest.

- The defender can perform the moves:
  - $(p, \langle a \rangle \varphi) \quad\quad \rightarrowtail (p', \varphi) \quad\;$ if $p \xrightarrow{a} p'$ and
  - $(p, \neg \bigwedge_{i \in I} \varphi_i) \rightarrowtail (p, \neg \varphi_i) \quad$ with $i \in I$;
- and the attacker can move:
  - $(p, \neg \langle a \rangle \varphi) \quad\;\; \rightarrowtail (p', \neg \varphi) \quad$ if $p \xrightarrow{a} p'$,
  - $(p, \bigwedge_{i \in I} \varphi_i) \quad \rightarrowtail (p, \varphi_i) \quad\quad$ with $i \in I$, and
  - $(p, \neg \neg \varphi) \quad\quad\;\; \rightarrowtail (p, \varphi)$.

Like in other logical games in the Ehrenfeucht–Fraïssé tradition, the attacker plays the conjunctions and universal quantifiers, whereas the defender plays the disjunctions and existential quantifiers. For instance, $(p, \langle a \rangle \varphi)$ is declared as defender position, since $\langle a \rangle \varphi$ is meant to become true precisely if *there exists* a state p' reachable $p \xrightarrow{a} p'$ where φ is true.

As every move strictly reduces the height of the formula, the game must be finite-depth (and cycle-free) for finite-height formulas, and, for image-finite systems and formulas, also finite. It is determined and the following semantics is total.

Definition 2.8 (HML semantics). For a transition system $\mathcal{S}$, the *semantics of* HML is given by defining that φ is true at p in $\mathcal{S}$, written $[\![\varphi]\!]^{\mathcal{S}}_p$, iff the defender wins $\mathcal{G}^{\mathcal{S}}_{\mathsf{HML}}[(p, \varphi)]$.

Example 2.9. Continuing Example 1.1, $[\![\langle a \rangle \neg \langle d \rangle]\!]^{\mathsf{CCS}}_{\mathsf{P}_2}$ is false: No matter whether the defender plays to $(b + d, \neg \langle d \rangle)$ or to $(c + d, \neg \langle d \rangle)$, the attacker wins by moving to the stuck defender position $(\mathbf{0}, \neg \mathsf{T})$. (Recall that T is the empty conjunction.)

2.3. **The Spectrum of Behavioral Equivalences.** For different theoretical and practical applications, a universe of notions of behavioral equivalence has been developed. Those equivalences are often defined in terms of relations on transition system state spaces or sets of executions. For the purpose of our paper, we rather focus on the correspondence between HML observation languages and notions of behavioral equivalence. In this framework, equivalence of processes means that the same observations are true for them.

Definition 2.10 (Distinguishing formula). A formula φ *distinguishes* state p from q iff $[\![\varphi]\!]_p$ is true and $[\![\varphi]\!]_q$ is not.[1]

Example 2.11. The formula $\langle a \rangle \neg \langle d \rangle$ distinguishes P_1 from P_2 in Example 1.1 (but not the other way around). The formula $\langle a \rangle \bigwedge \{\langle b \rangle, \langle d \rangle\}$ distinguishes P_2 from P_1.

Definition 2.12 (Observational preorders and equivalences). A set of observations, $\mathcal{O}_X \subseteq \mathsf{HML}[\Sigma]$, *preorders* two states p, q, written $p \preceq_X q$, iff no formula $\varphi \in \mathcal{O}_X$ distinguishes p from q. If $p \preceq_X q$ and $q \preceq_X p$, then the two are X-equivalent, written $p \sim_X q$.

Example 2.13 (Enabledness preorder and equivalence). $\mathcal{O}_\mathrm{E} = \{\langle a \rangle \mid a \in \Sigma\} \cup \{\mathsf{T}\}$ defines the preorder on what actions are initially enabled at two compared processes $\preceq_\mathrm{E}$, respectively the equivalence $\sim_\mathrm{E}$. Effectively, it means that $p \preceq_\mathrm{E} q$ iff, for all $p \xrightarrow{a} p'$, there is a q' such that $q \xrightarrow{a} q'$. (The $\{\mathsf{T}\}$ is there in order to ensure compatibility with upcoming definitions.)

---

[1]Here, and in the following, we usually leave the transition system $\mathcal{S}$ implicit.

$$\text{bisimulation}$$

$$\text{2-nested simulation}$$

$$\text{ready simulation}$$

$$\text{ready trace} \quad \text{possible futures}$$

$$\text{simulation} \quad \text{failure trace} \quad \text{readiness} \quad \text{impossible futures}$$

$$\text{failures}$$
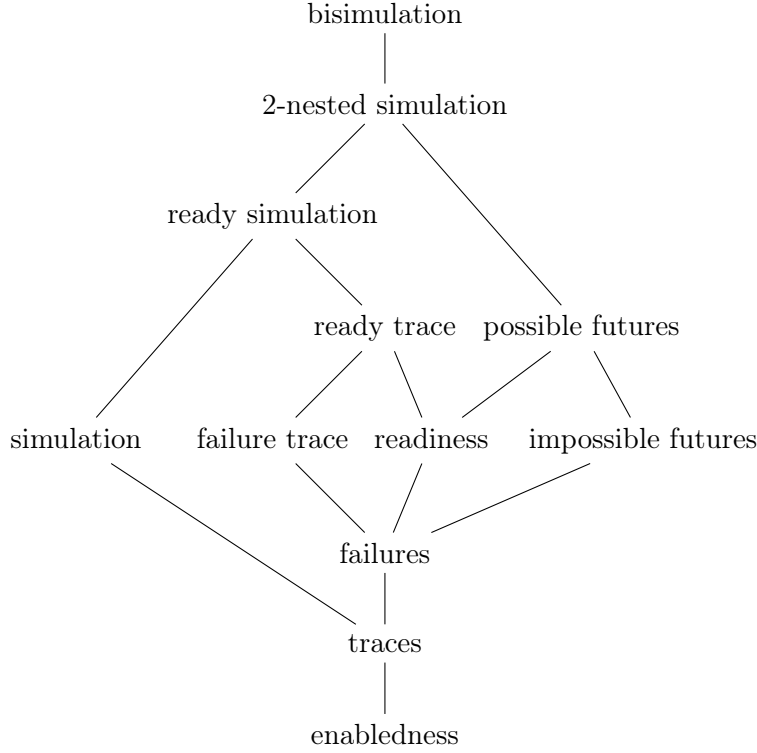
$$\text{traces}$$

$$\text{enabledness}$$

Figure 2: Hierarchy of equivalences/preorders becoming finer towards the top. Lines mean inclusion of observation languages from top to bottom.

The enabledness equivalence $\sim_{\mathrm{E}}$ is presumably the coarsest equivalence that one may encounter in the literature. There are many noteworthy finer equivalences. A broad overview is given in Figure 2.

Definition 2.14 (Linear-time–branching-time observation languages [vG01]). The linear-time–branching-time spectrum is a lattice of observation languages (and of entailed process preorders and equivalences). Every observation language $\mathcal{O}_X$ can perform trace observations, that is, $\mathsf{T} \in \mathcal{O}_X$ and, if $\varphi \in \mathcal{O}_X$, then $\langle a \rangle \varphi \in \mathcal{O}_X$. At the more linear-time side of the spectrum we have:

- *trace observations* $\mathcal{O}_{\mathrm{T}}$:   Just trace observations,
- *failure observations* $\mathcal{O}_{\mathrm{F}}$: $\bigwedge_{i \in I} \neg \langle a_i \rangle \in \mathcal{O}_{\mathrm{F}}$,
- *readiness observations* $\mathcal{O}_{\mathrm{R}}$: $\bigwedge_{i \in I} \varphi_i \in \mathcal{O}_{\mathrm{R}}$ if all $\varphi_i \in \{\langle a \rangle, \neg \langle a \rangle \mid a \in \Sigma\}$,
- *failure trace observations* $\mathcal{O}_{\mathrm{FT}}$: $\bigwedge_{i \in I} \varphi_i \in \mathcal{O}_{\mathrm{FT}}$ if $\varphi_0 \in \mathcal{O}_{\mathrm{FT}}$ and $\varphi_i = \neg \langle a_i \rangle$ for $i \neq 0$,
- *ready trace observations* $\mathcal{O}_{\mathrm{RT}}$: $\bigwedge_{i \in I} \varphi_i \in \mathcal{O}_{\mathrm{RT}}$ if $\varphi_0 \in \mathcal{O}_{\mathrm{RT}}$ and $\varphi_i \in \{\langle a \rangle, \neg \langle a \rangle \mid a \in \Sigma\}$ for $i \neq 0$,
- *impossible futures* $\mathcal{O}_{\mathrm{IF}}$: $\bigwedge_{i \in I} \neg \varphi_i \in \mathcal{O}_{\mathrm{IF}}$ if all $\varphi_i \in \mathcal{O}_{\mathrm{T}}$, and
- *possible futures* $\mathcal{O}_{\mathrm{PF}}$: $\bigwedge_{i \in I} \varphi_i \in \mathcal{O}_{\mathrm{PF}}$ if all $\varphi_i \in \{\psi, \neg \psi \mid \psi \in \mathcal{O}_{\mathrm{T}}\}$.

At the more branching-time side, we have simulation observations. Every simulation observation language $\mathcal{O}_{X\mathrm{S}}$ permits trace observation construction and has full conjunctive capacity, that is, if $\varphi_i \in \mathcal{O}_{X\mathrm{S}}$ for all $i \in I$, then $\bigwedge_{i \in I} \varphi_i \in \mathcal{O}_{X\mathrm{S}}$.

- *simulation observations* $\mathcal{O}_{1\text{S}}$: Just simulation (and trace) observations,
- *n-nested simulation observations* $\mathcal{O}_{n\text{S}}$: $\neg\varphi \in \mathcal{O}_{n\text{S}}$ if $\varphi \in \mathcal{O}_{(n-1)\text{S}}$,
- *ready simulation observations* $\mathcal{O}_{\text{RS}}$: $\neg\langle a\rangle \in \mathcal{O}_{\text{RS}}$, and
- *bisimulation observations* $\mathcal{O}_{\text{B}}$:   The same as $\bigcup_{n\in\mathbb{N}} \mathcal{O}_{n\text{S}}$, which is exactly $\mathsf{HML}[\Sigma]$.

The observation languages of the spectrum differ in how many of the syntactic features of HML one will encounter when descending into a formula's syntax tree. We will come back to this in Subsection 2.4.

    The languages of Definition 2.14 and $\mathcal{O}_{\text{E}}$ can be ordered by subset relations between them. The resulting inclusion structure is depicted in Figure 2. For two observation languages $\mathcal{O}_X$ and $\mathcal{O}_Y$ with $\mathcal{O}_X \subseteq \mathcal{O}_Y$, it is clear that $\mathcal{O}_X$ has at most as much distinctive capability as $\mathcal{O}_Y$, and thus $p \preceq_Y q$ implies $p \preceq_X q$. (Pay attention to the subset relation and the implication running in opposite directions!) Thus, the equivalences referred to in Figure 2 imply one-another downwards. Those implications usually are strict, albeit not for every transition system.

    Note that we consider $\bigwedge\{\varphi\}$ to be an alias for φ. With this aliasing, all the listed observation languages are *closed* in the sense that all subformulas and partial conjunctions (conjunctions of subsets) within each observation are themselves part of that language.

Definition 2.15 (Closed observation language)**.** We say that an observation language $\mathcal{O}_X$ is *closed* if:

- $\langle a\rangle\varphi \in \mathcal{O}_X$ implies $\varphi \in \mathcal{O}_X$,
- $\bigwedge\Phi \in \mathcal{O}_X$ implies $\Phi \subseteq \mathcal{O}_X$ and $\bigwedge\Phi' \in \mathcal{O}_X$ for every $\Phi' \subseteq \Phi$, and
- $(\neg\varphi) \in \mathcal{O}_X$ implies $\varphi \in \mathcal{O}_X$.

Proposition 2.16. *The languages of Definition 2.14 and $\mathcal{O}_{\text{E}}$ of Example 2.13 are closed.*

Proof. This can be seen by examining the definition. The aliasing is necessary for all cases with negations under conjunctions. For instance, for failure observations $\mathcal{O}_{\text{F}}$, we have to prove that $\bigwedge_{i\in I}\neg\langle a_i\rangle \in \mathcal{O}_{\text{F}}$ implies $\langle a_i\rangle \in \mathcal{O}_{\text{F}}$ and $\neg\langle a_i\rangle \in \mathcal{O}_{\text{F}}$ for all $i \in I$, as well as $\bigwedge_{i\in I'}\neg\langle a_i\rangle \in \mathcal{O}_{\text{F}}$ with $I' \subseteq I$. We easily see $\langle a_i\rangle \in \mathcal{O}_{\text{F}}$ because $\langle a_i\rangle \in \mathcal{O}_{\text{T}} \subseteq \mathcal{O}_{\text{F}}$. The second term is not mentioned by the definition, but has $\bigwedge\{\neg\langle a_i\rangle\}$ as an alias. The alias is mentioned by the definition, as are all $\bigwedge_{i\in I'}\neg\langle a_i\rangle \in \mathcal{O}_{\text{F}}$ with $I' \subseteq I$. The other observation languages are equally immediate.   □

    The languages of Definition 2.14 and $\mathcal{O}_{\text{E}}$ thus are *inductive* in the sense that all observations with finite syntax tree height must be built from smaller observations of the same language. This is convenient in proofs by structural induction.

Example 2.17. As explained in Examples 1.1 and 2.11, process P_1 has a run where it executes a and then cannot do d, while P_2 does not have such a run, so $\mathsf{P}_1 \not\preceq_{\text{F}} \mathsf{P}_2$. This can be expressed by the HML formula $\langle a\rangle\neg\langle d\rangle \in \mathcal{O}_{\text{F}}$, which distinguishes P_1 from P_2.—Moreover, P_2 cannot simulate the transition $\mathsf{P}_1 \xrightarrow{a} b + c$, so $\mathsf{P}_1 \not\preceq_{1\text{S}} \mathsf{P}_2$. This can be expressed by the distinguishing formula $\langle a\rangle\bigwedge\{\langle b\rangle, \langle c\rangle\} \in \mathcal{O}_{1\text{S}}$. (This formula happens to be in $\mathcal{O}_{\text{R}}$, the readiness observations, as well.)

    As a more complex example from [vG01, p. 21], see Figure 3. $\mathsf{P}_3 = a.(b+c.d)+a.(f+c.e)$ can be distinguished from $\mathsf{P}_4 = a.(b+c.e) + a.(f+c.d)$ in several ways:

- P_4 cannot simulate the transition $\mathsf{P}_3 \xrightarrow{a} b + c.d$. This means that $\mathsf{P}_3 \not\preceq_{1\text{S}} \mathsf{P}_4$. This corresponds to the HML formula $\langle a\rangle\bigwedge\{\langle b\rangle, \langle c\rangle\langle d\rangle\} \in \mathcal{O}_{1\text{S}}$.
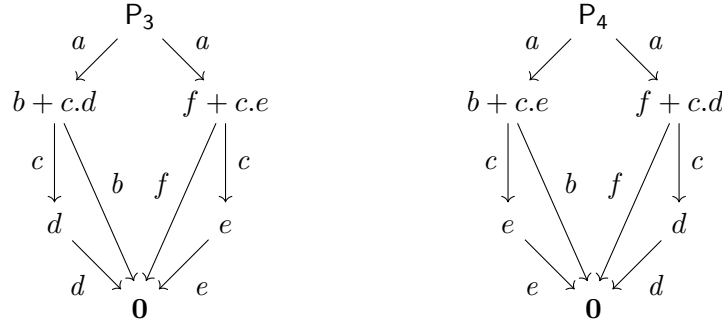
Figure 3: Additional processes for Example 2.17. (They differ by the position of d and e.)

- $\mathsf{P_3}$ has the failure trace $\varnothing a\{f\}c\varnothing d\varnothing$, which is not a failure trace of $\mathsf{P_4}$, so $\mathsf{P_3} \not\preceq_{\mathrm{FT}} \mathsf{P_4}$. This corresponds to the HML formula $\langle a\rangle\bigwedge\{\neg\langle f\rangle, \langle c\rangle\langle d\rangle\} \in \mathcal{O}_{\mathrm{FT}}$.
- $\langle a, \{b, c.d\}\rangle$ is an impossible future of $\mathsf{P_3}$, meaning that after action sequence a it may reach a state where no trace from $\{b, c.d\}$ can be executed, but this is not an impossible future of $\mathsf{P_4}$, so $\mathsf{P_3} \not\preceq_{\mathrm{IF}} \mathsf{P_4}$. This corresponds to the HML formula $\langle a\rangle\bigwedge\{\neg\langle b\rangle, \neg\langle c\rangle\langle d\rangle\} \in \mathcal{O}_{\mathrm{IF}}$.

The three formulas mentioned (and a variant of each one, using the other branch of $\mathsf{P_3}$) distinguish $\mathsf{P_3}$ from $\mathsf{P_4}$. Note that none of the three languages are contained in another; they are all minimal ways to tell the processes apart.

Remark 2.18. Like Kučera and Esparza [KE99], who studied the properties of "good" observation languages, we glimpse over completed trace, completed simulation and possible worlds observations in Definition 2.14, because these observations need a special exhaustive $\bigwedge_{a\in\Sigma}\varphi_a$, where the φ_a are deactivated actions for completed traces, and more complex trees for possible worlds. While it could be provided for with additional operators, it would break the closure property of observation languages, without giving much in return. For instance, for $\Sigma = \{a, b\}$, completed trace and completed simulation observations contain the observation $\bigwedge\{\neg\langle a\rangle, \neg\langle b\rangle\}$, but not its subformula $\neg\langle a\rangle$.

2.4. **Pricing Formulas.** In our following quest for the coarsest behavioral preorders distinguishing two states, we actually are interested in the formulas that are part of the *minimal observation languages* from the spectrum (Definition 2.14). We can think of the amount of HML-expressiveness used by a formula as its *price*. So, our first contribution is overlaying the spectrum with a price metric.

Definition 2.19 (Formula price lattice). The formula price lattice **Pr** is the (complete) lattice over $(\mathbb{N} \cup \{\infty\})^6$ with the partial order $\sqsubseteq$ defined by pointwise comparison, that is, $e \sqsubseteq e'$ iff $e_j \leq e'_j$ and $\bigsqcup E = e'$ iff $e'_j = \sup_{e\in E} e_j$, for all $j = 1, \ldots, 6$.

We use the six dimensions to catch the price structure of the spectrum from Definition 2.14. Intuitively, we employ the following metrics (listed in the order of the dimensions):

1. Observations: How many observations $\langle a\rangle \ldots$ may one pass at most when descending down the syntax tree? (So we count levels of observations, not the total number of observations.) This is called the "depth" of modal operator nesting for a formula in [HM85, Mil90].

2. **Conjunctions:** How often may one run into a conjunction? Negations in the beginning or following an observation are counted as implicit conjunctions.
3. **Positive Deep Branches:** How many positive deep branches may appear in each conjunction? We call conjuncts of the form $\langle a \rangle$ *positive flat branches,* and other positive branches *positive deep branches.*
4. **Positive Branches:** How many positive branches may appear in each conjunction, regardless of their depth?
5. **Negations:** How many negations may be visited when descending? In other variants of HML, this is sometimes called the number of "alternations" between $\diamond$ and $\square$.
6. **Negated Observations:** How many observations can happen under each negation? In other words, what is the maximum observation depth of the negative parts of the formula?

Each dimension expresses some aspect of the complexity of formulas in the spectrum of Definition 2.14. Intuitively, the notions of equivalence in the spectrum differ by how linear observations (traces) and branching observations (conjunctions) can be mixed. At the branching points, they differ in the kinds of lower bounds (positive branches) and upper bounds (negative branches) that may be imposed on follow-up behavior. This also gives an intuitive explanation of why negations under observations count as implicit conjunctions in our metric: they form an upper bound on the behavior at the point where the trace observation stops.

More formally, we compute the six dimensions as follows:

Definition 2.20 (Formula expressiveness prices)**.** We define the *(expressiveness) price* of a formula, $\mathsf{expr} \colon \mathsf{HML} \to \mathbf{Pr}$ recursively by:

$$
\mathsf{expr}(\langle a \rangle \varphi) = \begin{pmatrix} 1 + \mathsf{expr}_1(\varphi) \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \sqcup \mathsf{expr}(\widehat{\varphi}) \qquad \mathsf{expr}(\neg \varphi) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 + \mathsf{expr}_5(\varphi) \\ \mathsf{expr}_1(\varphi) \end{pmatrix} \sqcup \mathsf{expr}(\varphi)
$$

$$
\mathsf{expr}(\bigwedge_{i \in I} \varphi_i) = \begin{pmatrix} 0 \\ \sup_{i \in I} 1 + \mathsf{expr}_2(\varphi_i) \\ |pb| - |pf| \\ |pb| \\ 0 \\ 0 \end{pmatrix} \sqcup \bigsqcup_{i \in I} \mathsf{expr}(\varphi_i) \qquad \begin{aligned} \widehat{\varphi} &= \begin{cases} \bigwedge\{\varphi\}, & \text{if } \exists \varphi'. \varphi = \neg \varphi' \\ \varphi, & \text{otherwise} \end{cases} \\[1em] pb &= \{\varphi_i \mid \nexists \varphi'. \varphi_i = \neg \varphi'\} \\ pf &= \{\varphi_i \mid \exists a \in \Sigma. \varphi_i = \langle a \rangle\} \end{aligned}
$$

In these calculations, $\mathsf{expr}_j(\,\cdot\,)$ stands for the jth dimension of the expressiveness price, pb is the set of positive branches of a conjunction, and pf is the set of positive flat branches.

The price of a standalone formula φ is $\mathsf{expr}(\widehat{\varphi})$. (We also apply $\widehat{}$ in the definition of $\mathsf{expr}(\langle a \rangle \varphi)$ to ensure that negations following an observation are counted as implicit conjunctions.)

Example 2.21. Let us calculate the prices of the formulas in Example 2.17. For $\langle a \rangle \neg \langle d \rangle$ (that distinguishes P_1 from P_2), the price of subformula $\neg \langle d \rangle$ is calculated with an additional conjunction: $\mathsf{expr}(\widehat{\neg \langle d \rangle}) = \mathsf{expr}(\bigwedge\{\neg \langle d \rangle\}) = (1, 1, 0, 0, 1, 1)$. This leads to $\mathsf{expr}(\widehat{\langle a \rangle \neg \langle d \rangle}) = (2, 1, 0, 0, 1, 1)$.—The other distinguishing formula $\langle a \rangle \bigwedge\{\langle b \rangle, \langle c \rangle\}$ has price $(2, 1, 0, 2, 0, 0)$.

Table 1: Expressiveness price bounds per observation language.

Observation language	Observations	Conjunctions	Positive deep br.	Positive branches	Negations	Negated observations
enabledness $\mathcal{O}_\mathrm{E}$	1	0	0	0	0	0
trace $\mathcal{O}_\mathrm{T}$	∞	0	0	0	0	0
failure $\mathcal{O}_\mathrm{F}$	∞	1	0	0	1	1
readiness $\mathcal{O}_\mathrm{R}$	∞	1	0	∞	1	1
failure-trace $\mathcal{O}_\mathrm{FT}$	∞	∞	1	1	1	1
ready-trace $\mathcal{O}_\mathrm{RT}$	∞	∞	1	∞	1	1
impossible-future $\mathcal{O}_\mathrm{IF}$	∞	1	0	0	1	∞
possible-future $\mathcal{O}_\mathrm{PF}$	∞	1	∞	∞	1	∞
ready-simulation $\mathcal{O}_\mathrm{RS}$	∞	∞	∞	∞	1	1
$(n{+}1)$-nested-simulation $\mathcal{O}_{(n+1)\mathrm{S}}$	∞	∞	∞	∞	n	∞
bisimulation $\mathcal{O}_\mathrm{B}$	∞	∞	∞	∞	∞	∞

The formulas that distinguish $\mathsf{P_3}$ from $\mathsf{P_4}$ have the prices:

$$\mathsf{expr}(\widehat{\langle a\rangle \bigwedge\{\langle b\rangle, \langle c\rangle\langle d\rangle\}}) = (3,1,1,2,0,0) \qquad \mathsf{expr}(\widehat{\langle a\rangle \bigwedge\{\neg\langle f\rangle, \langle c\rangle\langle d\rangle\}}) = (3,1,1,1,1,1)$$

$$\mathsf{expr}(\widehat{\langle a\rangle \bigwedge\{\neg\langle b\rangle, \neg\langle c\rangle\langle d\rangle\}}) = (3,1,0,0,1,2)$$

For both process pairs, there are multiple minimal-price distinguishing formulas, which are incomparable. This reflects our earlier observation that there are multiple minimal languages to tell the processes apart. We will make this more exact in Lemma 2.23 below.

We say that a formula φ_1 *dominates* φ_2 if φ_1 has lower or equal values than φ_2 in each dimension of the metrics with at least one entry strictly lower, for which we write $\mathsf{expr}(\widehat{\varphi_1}) \sqsubset \mathsf{expr}(\widehat{\varphi_2})$.

Example 2.22. A locally more expensive formula may pay off as part of a bigger global formula. For example, $\neg\langle a\rangle$ is dominated by $\langle b\rangle$ as $(1,1,0,0,1,1) \sqsupset (1,0,0,0,0,0)$. But inserted into the context $\bigwedge\{\neg\langle c\rangle,\ \cdot\ \}$, the more expensive negation $\neg\langle a\rangle$ leads to price $(1,1,0,0,1,1)$ as opposed to the price $(1,1,0,1,1,1)$ after inserting the positive observation $\langle b\rangle$.

Note that this inversion of domination can only happen for the dimensions expr_3 (positive deep branches) and expr_4 (positive branches), as all other dimensions in Definition 2.20 are clearly monotonic. Then the context must contain a conjunction $\bigwedge_{i\in I}\varphi_i$ and the local formula must be a conjunct φ_i, the cheaper local formula is an observation formula and the other a negation formula.

Table 1 gives an overview of how many syntactic HML-features the observation languages of the spectrum (Definition 2.14) may use at most—these are the least upper bounds of the prices for the contained observations. So, we are talking *budgets,* in the price analogy.

Lemma 2.23. *A formula φ is in an observation language $\mathcal{O}_X$ with expressiveness price bound e_X from Table 1 precisely if its price is within the bound, that is $\mathsf{expr}(\widehat{\varphi}) \sqsubseteq e_X$.*

Proof Sketch. $\varphi \in \mathcal{O}_X$ implies $\mathsf{expr}(\widehat{\varphi}) \sqsubseteq e_X$ as the e_X in the table exactly are the least upper bounds $\bigsqcup_{\varphi \in \mathcal{O}_X} \mathsf{expr}(\widehat{\varphi})$. That $\mathsf{expr}(\widehat{\varphi}) \sqsubseteq e_X$ implies $\varphi \in \mathcal{O}_X$, is more involved. Basically it amounts to constructing the fiber function expr^{-1} and noticing that a partial evaluation of $\mathsf{expr}^{-1}(e_X)$ looks exactly like the cases in Definition 2.14. For failures for instance, one would get $\mathsf{expr}^{-1}(e_\mathrm{F}) = \{\langle a \rangle \varphi \mid \varphi \in \mathsf{expr}^{-1}(e_\mathrm{F})\} \cup \{\mathsf{T}\} \cup \{\bigwedge_{a \in A} \neg \langle a \rangle \mid A \subseteq \Sigma\}$. □

Note that no strict subset of these dimensions distinguishes all languages of Definition 2.14 and $\mathcal{O}_\mathrm{E}$.

Example 2.24. We can now compare the formula prices calculated in Example 2.21 with the expressiveness price bounds: The price of $\langle a \rangle \neg \langle d \rangle \in \mathcal{O}_\mathrm{F}$ is $(2, 1, 0, 0, 1, 1)$, which is cheaper than the price bound $e_\mathrm{F} = (\infty, 1, 0, 0, 1, 1)$ of $\mathcal{O}_\mathrm{F}$.—The other distinguishing formula $\langle a \rangle \bigwedge \{\langle b \rangle, \langle c \rangle\} \in \mathcal{O}_{1\mathrm{S}} \cap \mathcal{O}_\mathrm{R}$ has price $(2, 1, 0, 2, 0, 0)$, which is cheaper than the price bounds of $\mathcal{O}_{1\mathrm{S}}$ and $\mathcal{O}_\mathrm{R}$.

The formulas that distinguish P_3 from P_4 are within the following price bounds:

$$\mathsf{expr}(\overline{\langle a \rangle \bigwedge \{\langle b \rangle, \langle c \rangle \langle d \rangle\}}) = (3, 1, 1, 2, 0, 0) \sqsubseteq (\infty, \infty, \infty, \infty, 0, \infty), \text{ the price bound of } \mathcal{O}_{1\mathrm{S}}$$
$$\sqsubseteq (\infty, 1, \infty, \infty, 1, \infty), \text{ the price bound of } \mathcal{O}_\mathrm{PF}$$
$$\sqsubseteq (\infty, \infty, 1, \infty, 1, 1), \text{ the price bound of } \mathcal{O}_\mathrm{RT}$$
$$\mathsf{expr}(\overline{\langle a \rangle \bigwedge \{\neg \langle f \rangle, \langle c \rangle \langle d \rangle\}}) = (3, 1, 1, 1, 1, 1) \sqsubseteq (\infty, \infty, 1, 1, 1, 1), \text{ the price bound of } \mathcal{O}_\mathrm{FT}$$
$$\sqsubseteq (\infty, 1, \infty, \infty, 1, \infty), \text{ the price bound of } \mathcal{O}_\mathrm{PF}$$
$$\mathsf{expr}(\overline{\langle a \rangle \bigwedge \{\neg \langle b \rangle, \neg \langle c \rangle \langle d \rangle\}}) = (3, 1, 0, 0, 1, 2) \sqsubseteq (\infty, 1, 0, 0, 1, \infty), \text{ the price bound of } \mathcal{O}_\mathrm{IF}$$

We can see that these prices are below some further price bounds printed in gray. However, these bounds are less relevant, as $\mathcal{O}_\mathrm{FT} \subset \mathcal{O}_\mathrm{RT}$ and $\mathcal{O}_\mathrm{IF} \subset \mathcal{O}_\mathrm{PF}$.

Not every conceivable observation language can soundly be characterized by our metric. For instance, the "two-as-may-happen-equivalence" with observation language $\{\langle a \rangle \langle a \rangle, \mathsf{T}\}$ would have coordinates $(2, 0, 0, 0, 0, 0)$ but does not contain the formula $\langle a \rangle$ even though $\mathsf{expr}(\langle a \rangle) = (1, 0, 0, 0, 0, 0)$ is below. However, all common closed observation languages have characteristic coordinates in our price lattice.

Remark 2.25 (New prices). Definition 2.20 of formula expressiveness prices and the spectrum characterization differ from the conference version [BN21] in that the fourth dimension counts positive branches instead of positive *flat* branches and in that the last dimension "negated observations" measures the observation depth instead of negated syntax tree height. We will use these two changes in the proof of Theorem 3.14.

Not counting flat positive branches resolves some technical problems that existed around readiness and failure trace languages. The advantage of this definition of formula price is that we have $\mathsf{expr}(\bigwedge \{\langle a \rangle \langle a \rangle, \langle b \rangle\}) \sqsubseteq \mathsf{expr}(\bigwedge \{\langle a \rangle \langle a \rangle, \langle b \rangle \langle b \rangle\})$. This would not hold in the definition of [BN21], because $\bigwedge \{\langle a \rangle \langle a \rangle, \langle b \rangle\}$ contains the positive flat branch $\langle b \rangle$ while the other formula does not contain any positive flat branches.

The measuring of negated height in terms of observations is desirable for technical reasons. It is more stable under the insertion of virtual conjunctions in front of negations.

This way, $\neg\langle a\rangle\bigwedge\{\neg\langle b\rangle, \neg\langle a\rangle\}$, $\neg\langle a\rangle\bigwedge\{\neg\langle b\rangle\}$ and $\neg\langle a\rangle\neg\langle b\rangle$ all have $\mathsf{expr}(\hat{\ })=(2,2,0,0,2,2)$ as price. Using syntax tree height, the sixth dimension would be 3 for the last formula and 4 for the other two. Also, if we would not include an implicit conjunction before each negation, the last formula would have a smaller value in the second dimension.

2.5. **Double Negations and Negated Conjunctions.** When searching distinguishing formulas in the observation languages considered, double negations and negated conjunctions are not useful and can be suppressed.

A *double negation* $\neg\neg\varphi$ is more expensive than φ in most contexts. Only in a conjunction, a double negation can mask a positive branch; for example, $\langle a\rangle\bigwedge\{\neg\neg\langle b\rangle, \neg\langle c\rangle\neg\langle d\rangle\}$ is strictly cheaper than $\langle a\rangle\bigwedge\{\langle b\rangle, \neg\langle c\rangle\neg\langle d\rangle\}$. Note that a formula containing a double negation is in 3-nested simulation $\mathcal{O}_{3S}$ or a larger language, where the number of positive (deep) branches is irrelevant. Therefore, double negations are not useful, and we forbid them in distinguishing formulas.

Negated conjunctions are disjunctions. (In this paragraph, we use the abbreviation $\bigvee_{i\in I}\varphi_i$ for $\neg\bigwedge_{i\in I}\neg\varphi_i$.) They can be replaced by (almost always cheaper) formulas without disjunction. For example, $\langle a\rangle\neg\bigwedge\{\langle b\rangle, \langle c\rangle\}$ distinguishes $a.b$ and $a.c$ from $a.(b+c)$. This formula is equivalent to $\bigvee\{\langle a\rangle\neg\langle b\rangle, \langle a\rangle\neg\langle c\rangle\}$. Note that the disjuncts $\langle a\rangle\neg\langle b\rangle$ and $\langle a\rangle\neg\langle c\rangle$ also distinguish $a.c$ and $a.b$, respectively, from $a.(b+c)$.

In general, any formula containing a disjunction can be transformed into a kind of disjunctive normal form $\bigvee_{i\in I}\varphi_i$, where the φ_i do not contain any disjunctions. When a concrete process p needs to be distinguished from some process q, one of the disjuncts φ_i can be used instead of the whole formula. The transformation to disjunctive normal form will almost always lead to disjuncts that are cheaper than or as cheap as the original formula. The only exception is the case where a disjunction masks a double negation in a conjunction; for example, $\langle a\rangle\bigwedge\{\neg\bigwedge\{\neg\langle b\rangle\}, \neg\langle c\rangle\neg\langle d\rangle\}$ corresponds to $\langle a\rangle\bigwedge\{\neg\neg\langle b\rangle, \neg\langle c\rangle\neg\langle d\rangle\}$. As explained above, this only happens in 3-nested simulation $\mathcal{O}_{3S}$ and larger languages, where the number of positive (deep) branches is irrelevant. Therefore, like double negations, negated conjunctions are not useful, and we forbid them in distinguishing formulas as well.

3. A Game to Find Distinguishing Formulas

This section introduces our main contribution: the spectroscopy game (Definition 3.3), and how to build all interesting distinguishing HML formulas from its winning region (Definition 3.6).

3.1. **The Abstract Observation Preorder Problem.** In what follows, we generalize the problem whether an observation language preorders two states (Definition 2.12) in two ways:

(1) We not only consider one observation language $\mathcal{O}_X$ from Definition 2.14 or $\mathcal{O}_E$ but all of them at the same time.

(2) We do not compare one process p to another q, but rather one p to a set $Q\subseteq\mathcal{P}$.

The first generalization is the main objective of this paper. The second one enables the construction of a uniform algorithm. The abstracted problem thus becomes:

Problem 3.1 (Abstract observation preorder problem)**.** Given a process p and a set of processes Q, what are the observation languages from Definition 2.14 (including $\mathcal{O}_{\mathrm{E}}$) for which p is preordered to every $q \in Q$?

Our approach to solve this problem looks for the set of minimal languages to tell the processes apart. We characterize these minimal distinguishing languages through a set of coordinates from the price lattice (Definition 2.19), where every coordinate is justified by a distinguishing formula with this price. In line with Subsection 2.5, we do not care about formulas with double negations and negated conjunctions.

Problem 3.2 (Cheapest distinction problem)**.** Given a process p and a set of processes Q, what is the set of minimal prices (according to Definition 2.20) of formulas with neither double negations nor negated conjunctions that distinguish p from every $q \in Q$? What are illuminating witness formulas for each such price?

There is a straightforward way of turning the problem whether an observation language $\mathcal{O}_X$ preorders p and q into a game: Have the attacker pick a supposedly distinguishing formula $\varphi \in \mathcal{O}_X$, and then have the defender choose whether to play the HML game (Definition 2.7) for $\llbracket \neg \varphi \rrbracket_p$ or for $\llbracket \varphi \rrbracket_q$. One can examine the attacker winning strategies comprised of price-minimal formulas and solve Problem 3.2. This direct route will yield infinite games for infinite $\mathcal{O}_X$—and all the languages from Definition 2.14 are infinite.

To bypass the infinity issue, the next subsection will introduce a variation of this game *where the attacker gradually chooses their attacking formula implicitly.* In particular, this means that the attacker decides which observations to play. In return, the defender does not need to pick a side in the beginning and may postpone the decision where (on the side of $\llbracket \varphi \rrbracket_q$) an observation leads. Postponing decisions here means that the defender state is modeled non-deterministically, moving to multiple process states at once. The mechanics are analogous to the standard powerset construction when transforming non-deterministic finite automata into deterministic ones. In effect, the attacker tries to show that a formula $\varphi \in \mathcal{O}_X$ distinguishes p from every $q \in Q$, and the defender tries to prove that no such formula exists.

3.2. **The Spectroscopy Game.** Let us now look at the "spectroscopy game." It forms the heart of this paper. Figure 4 gives a graphical representation.

Definition 3.3 (Spectroscopy game)**.** For a transition system $\mathcal{S} = (\mathcal{P}, \Sigma, \rightarrow)$, the L-labeled *spectroscopy game* $\mathcal{G}_{\triangle}^{\mathcal{S}}[g_0] = (G, G_{\mathrm{d}}, \rightarrowtail, g_0)$ with $L = \{\neg, \wedge, *, \langle a \rangle \mid a \in \Sigma\}$ consists of

- *attacker positions* $(p, Q)_{\mathsf{a}} \in G_{\mathsf{a}}$ with $p \in \mathcal{P}$, $Q \in \mathbf{2}^{\mathcal{P}}$,
- *post-conjunction attacker positions* $(p, Q)_{\mathsf{a}}^{\curlywedge} \in G_{\mathsf{a}}$, where $Q \in \mathbf{2}^{\mathcal{P}}$ contains at least two elements; the symbol "$\curlywedge$" indicates that conjunct challenge moves are not allowed there,
- *defender positions* $(p, \mathcal{Q})_{\mathsf{d}} \in G_{\mathsf{d}}$ where $\mathcal{Q} \subseteq \mathbf{2}^{\mathcal{P}}$ is a partition of a subset of $\mathcal{P}$,

and four kinds of moves:

- *observation moves* $(p, Q)_{\mathsf{a}} \overset{\langle a \rangle}{\rightarrowtail} (p', \{q' \mid \exists q \in Q.\, q \xrightarrow{a} q'\})_{\mathsf{a}}$ if $p \xrightarrow{a} p'$,
  $(p, Q)_{\mathsf{a}}^{\curlywedge} \overset{\langle a \rangle}{\rightarrowtail} (p', \{q' \mid \exists q \in Q.\, q \xrightarrow{a} q'\})_{\mathsf{a}}$ if $p \xrightarrow{a} p'$,
- *conjunct challenges* $(p, Q)_{\mathsf{a}} \overset{\triangle}{\rightarrowtail} (p, \mathcal{Q})_{\mathsf{d}}$ if $\mathcal{Q}$ is a partition of Q that is not trivial (i.e. $\mathcal{Q} \neq \{Q\}$),
- *conjunct answers* $(p, \mathcal{Q})_{\mathsf{d}} \overset{*}{\rightarrowtail} (p, Q)_{\mathsf{a}}^{\curlywedge}$ if $Q \in \mathcal{Q}$ is not a singleton,
  $(p, \mathcal{Q})_{\mathsf{d}} \overset{*}{\rightarrowtail} (p, \{q\})_{\mathsf{a}}$ if $\{q\} \in \mathcal{Q}$, and
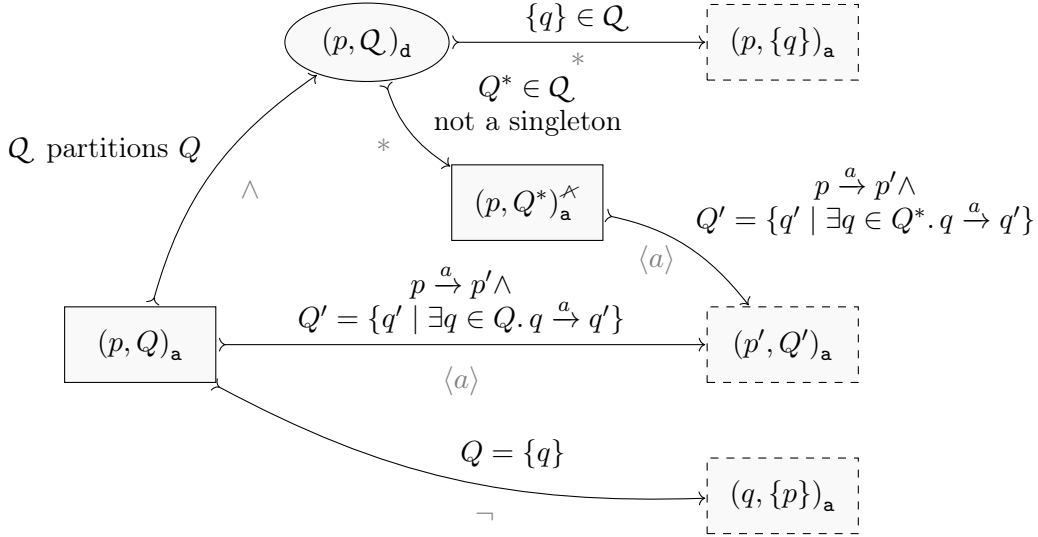- *negation moves* $(p, \{q\})_{\mathsf{a}} \overset{\neg}{\rightarrowtail} (q, \{p\})_{\mathsf{a}}$.

Figure 4: Schematic spectroscopy game $\mathcal{G}_\triangle$ of Definition 3.3.

Attacker moves are labeled with the syntactic HML constructs from which they originate. This does not change expressive power but is helpful for formula reconstruction in the next section. Accordingly, attacker strategies for spectroscopy games are subsets of labeled moves.

Example 3.4. Let us say we want to compare the processes $a.b$ and $a.(a+b) + a.b.b + a$. Figure 5 shows the game graph. Recall from Definition 2.4 that the attacker wins in defender positions that are stuck, concretely in position $(\mathbf{0}, \varnothing)_\mathsf{d}$. The defender wins infinite plays, concretely from position $(\mathbf{0}, \{\mathbf{0}\})_\mathsf{a}$, where the only play loops infinitely in place. Elsewhere, the player that can force the play into one of these two positions wins. The black part represents the attacker winning region, which corresponds to processes that can be distinguished by HML formulas. The edge labels already hint at how to construct such formulas. The (thin, small) red part corresponds to processes that cannot be distinguished and thus are won by the defender. We will make this more precise in the next subsections.

3.3. **Relationship to Bisimulation.** Comparing $\mathcal{G}_\triangle$ to the standard bisimulation game from the literature (amending games originating from [Sti96] with symmetry moves, see e.g. [BNP20]), we can easily transfer attacker strategies from there. In the standard bisimulation game, the attacker will play $(p, q) \rightarrowtail (a, p', q)$ with $p \xrightarrow{a} p'$ and the defender has to answer by $(a, p', q) \rightarrowtail (p', q')$ with $q \xrightarrow{a} q'$. In the spectroscopy game, the attacker can enforce analogous moves by playing

$$(p, \{q\})_\mathsf{a} \xrightarrowtail{\langle a \rangle} (p', Q')_\mathsf{a} \xrightarrowtail{\triangle} (p', \{\{q^*\} \mid q^* \in Q'\})_\mathsf{d},$$

which will make the defender pick $(p', \{\{q^*\} \mid q^* \in Q'\})_\mathsf{d} \xrightarrowtail{*} (p', \{q'\})_\mathsf{a}$.

The opposite direction of transfer is not so easy, as the attacker has more ways of winning in $\mathcal{G}_\triangle$. But this asymmetry is precisely why we have to use the spectroscopy game instead of the standard bisimulation game if we want to learn about, for example, interesting failure-trace attacks.
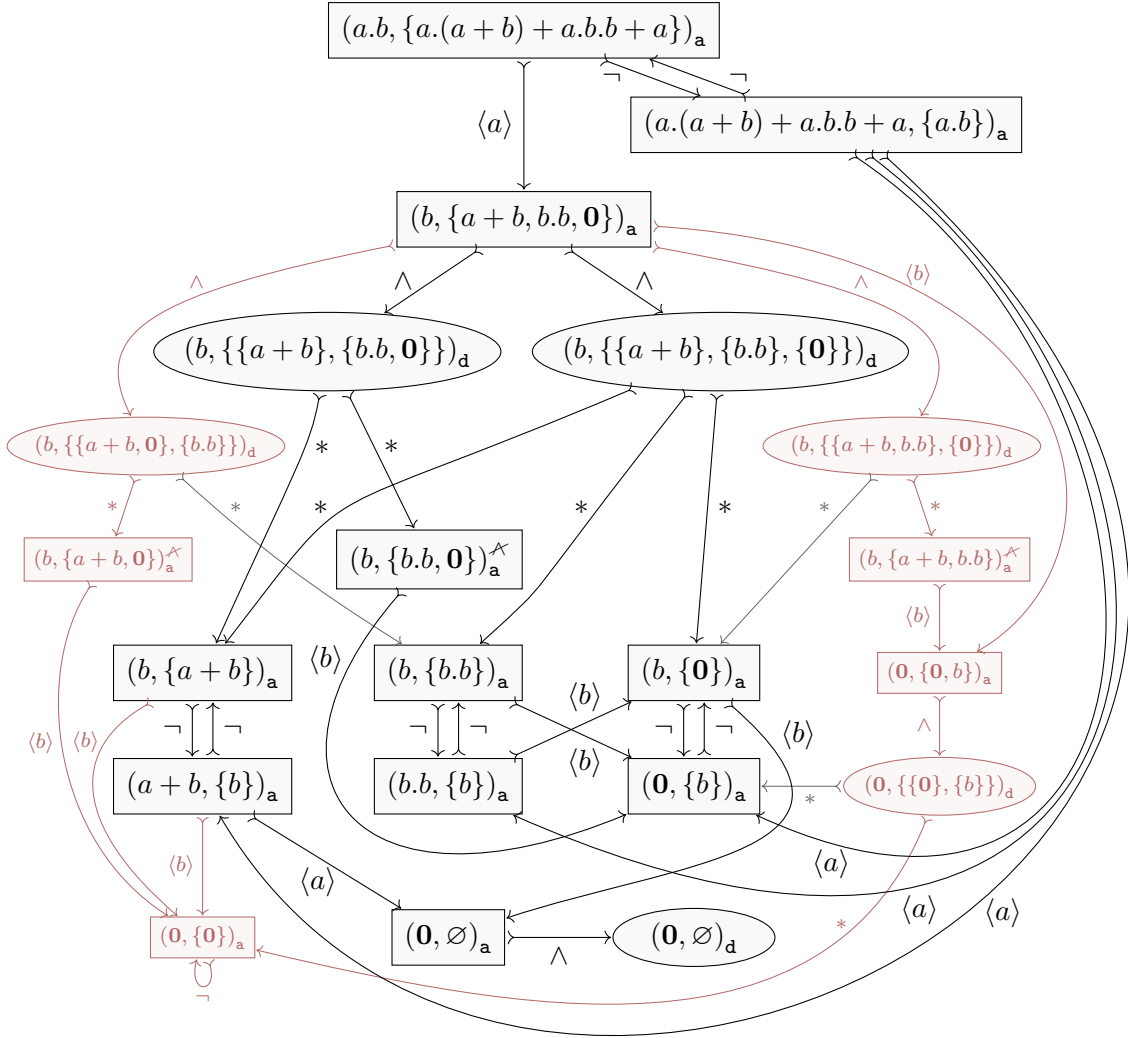
Figure 5: The spectroscopy game that distinguishes process $a.b$ from $a.(a + b) + a.b.b + a$. Parts where the defender wins are drawn thin, small, and red.

Indeed, the spectroscopy game does characterize bisimilarity. Bisimilarity denotes the existence of a bisimulation relation $\mathcal{R}$ that has the properties that $\mathcal{R} \subseteq \mathcal{R}^{-1}$ (symmetry) and that $(\mathcal{R}^{-1}) \cdot (\xrightarrow{a}) \subseteq (\xrightarrow{a}) \cdot (\mathcal{R}^{-1})$ for all $a \in \Sigma$ (simulation).

Lemma 3.5. *The spectroscopy game $\mathcal{G}_\triangle[(p_0, \{q_0\})_\mathsf{a}]$ is won by the defender precisely if p_0 and q_0 are bisimilar.*

Proof. We use that bisimilarity is equivalent to the existence of a bisimulation relation, and that winning is equivalent to the existence of a positional strategy.

- Construct the relation $\mathcal{R} = \{(p, q) \mid (p, \{q\})_\mathsf{a} \in W_\mathsf{d}\}$. This must be a symmetric simulation (and thus a bisimulation).
  - Symmetry: Assume $(p, q) \in \mathcal{R}$. As the attacker can play from $(p, \{q\})_\mathsf{a}$ to $(q, \{p\})_\mathsf{a}$, the two can only be in the defender winning region together. Thus $(q, p) \in \mathcal{R}$.

– Simulation: Assume $(p, q) \in \mathcal{R}$ and $p \xrightarrow{a} p'$. Then there is an attacker move $(p, \{q\})_{\mathsf{a}}$ to $(p', Q')_{\mathsf{a}}$ for $Q' = \{q' \mid q \xrightarrow{a} q'\}$. We must have $(p', Q')_{\mathsf{a}} \in W_{\mathsf{d}}$ (otherwise, $(p, \{q\})_{\mathsf{a}}$ would be winning for the attacker); this excludes in particular $Q' = \varnothing$. If Q' is a singleton, say $Q' = \{q'\}$, we get $(p', q') \in \mathcal{R}$ and the simulation is proven. Otherwise, the attacker can take a conjunct challenge to $(p', \{\{q'\} \mid q' \in Q'\})_{\mathsf{d}}$. There must be a move from this position to some $(p', \{q_0'\})_{\mathsf{a}}$ that is still in the defender winning region W_{d}. As $q_0' \in Q'$, we can be sure that $q \xrightarrow{a} q_0'$, so that $(p', q_0') \in \mathcal{R}$ completes this case.

If $\mathcal{G}_\triangle[(p_0, \{q_0\})_{\mathsf{a}}]$ is won by the defender, this means $(p_0, \{q_0\})_{\mathsf{a}} \in W_{\mathsf{d}}$ and thus $(p_0, q_0) \in \mathcal{R}$.

• Assume $\mathcal{R}$ is a bisimulation relation and $(p_0, q_0) \in \mathcal{R}$. Construct the defender strategy $F((p, \mathcal{Q})_{\mathsf{d}}) = \{(p, Q)_{\mathsf{a}} \mid Q \in \mathcal{Q} \wedge \exists q \in Q. \, (p, q) \in \mathcal{R}\}$. Starting with $(p_0, \{q_0\})_{\mathsf{a}}$ whatever the attacker plays and whatever moves the defender chooses from F, we prove that the play will maintain the invariant that for attacker positions $(p, Q)_{\mathsf{a}}$ or $(p, Q)_{\mathsf{a}}^{\not\curvearrowright}$ there will always be $q \in Q$ such that $(p, q) \in \mathcal{R}$, and similarly for defender positions $(p, \mathcal{Q})_{\mathsf{d}}$ there will always be $Q \in \mathcal{Q}$ and $q \in Q$ such that $(p, q) \in \mathcal{R}$.

– Observation moves: At $(p, Q)_{\mathsf{a}}$, the attacker moves to $(p', Q')_{\mathsf{a}}$ with $p \xrightarrow{a} p'$ and Q' reachable by $q \xrightarrow{a}$. By the invariant, there was $q \in Q$ such that $(p, q) \in \mathcal{R}$. As $\mathcal{R}$ is a simulation, there must be $q' \in Q'$ with $(p', q') \in \mathcal{R}$.

– Conjunct challenges: At $(p, Q)_{\mathsf{a}}$, the attacker moves to $(p, \mathcal{Q})_{\mathsf{d}}$. As $\mathcal{Q}$ covers Q, the invariant is maintained.

– Conjunct answers: At $(p, \mathcal{Q})_{\mathsf{d}}$, the defender moves to some $g' \in F((p, \mathcal{Q})_{\mathsf{d}})$. Because of the invariant, the set cannot be empty. Because of the definition of F, the new position must keep the invariant.

– Negation moves: At $(p, \{q\})_{\mathsf{a}}$, the attacker moves to $(q, \{p\})_{\mathsf{a}}$. Due to $\mathcal{R}$ being symmetric, the invariant holds.

Because of the invariant, the defender cannot get stuck following F, and thus wins.   □

3.4. **Deciding the Spectroscopy Game.** Due to the partition construction over subsets of $\mathcal{P}$, the worst-case game size is proportional to the Bell number $\mathbf{B}(1 + |\mathcal{P}|)$, which is somewhat worse than exponential. Going at least exponential is necessary, as we want to also characterize weaker preorders like the trace preorder, where exponential $\mathcal{P}$-subset or Σ^*-word constructions cannot be circumvented. However, for moderate real-world systems, such constructions will not necessarily show their full exponential blow-up (cf. [CH93]). In our case, we went beyond exponential complexity in order to correctly address failure trace and ready trace languages, as will be explained in Subsection 3.8.

To be more precise, the game size is asymptotically described by the number of conjunct answers, $|\mathcal{P}|^2 \cdot \mathbf{B}(1 + |\mathcal{P}|)$. In detail: There are $|\mathcal{P}| \cdot 2^{|\mathcal{P}|}$ potential attacker positions. For every such position, there are Bell-number many conjunct challenges and corresponding defender positions (so $|\mathcal{P}| \cdot \mathbf{B}(1 + |\mathcal{P}|)$ in total, each) with up to $|\mathcal{P}|^2 \cdot \mathbf{B}(1 + |\mathcal{P}|)$ conjunct answers. Moreover, there are up to $|p \xrightarrow{} \cdot|$ observation moves for each attacker position, totalling to $|\xrightarrow{}| \cdot 2^{|\mathcal{P}|}$. Finally, there are the $|\mathcal{P}|^2$ negation moves, which clearly are dominated by the other complexities.

For our approach in the next section, we will need to know (and process) the whole attacker winning region of size $\mathcal{O}(|\mathcal{P}| \cdot \mathbf{B}(1 + |\mathcal{P}|))$. We use Algorithm 1 to compute the winning region of reachability games (derived from [Grä07]). It starts by assuming that every position is won by the defender, and then proceeds by visiting positions where the defender does not have any options or where the attacker has an option. Every move in the

```
1  def compute_winning_region(𝒢 = (G, G_d, ↣)):
2      defender_options := [g ↦ n | g ∈ G ∧ n = |{p | g ↣ p}|]
3      attacker_win := {}
4      def propagate_attacker_win(g):
5          if g ∉ attacker_win :
6              attacker_win := attacker_win ∪ {g}
7              for g_p ∈ (· ↣ g):
8                  defender_options[g_p] := defender_options[g_p] − 1
9                  if g_p ∈ G_a ∨ defender_options[g_p] = 0 :
10                     propagate_attacker_win(g_p)
11     for g ∈ G_d:
12         if defender_options[g] = 0 :
13             propagate_attacker_win(g)
14     return attacker_win
```

Algorithm 1: Algorithm for deciding the attacker winning region W_a of a reachability game $\mathcal{G}$ in linear time of $|\!\!\rightarrowtail\!\!|$ and linear space of $|G|$.

game will lead to at most one invocation of the inner-most operations, which renders the algorithm linear-time with respect to game moves. The algorithm keeps information on every game position, making the space complexity linear in the number of game positions and thus Bell-number super-exponential with respect to $|\mathcal{P}|$.

3.5. **Building Distinguishing Formulas from Attacker Strategies.** Attacker strategies in $\mathcal{G}_\triangle$ correspond to sets of HML formulas. If the strategies are winning, the formulas are distinguishing.

Definition 3.6 (Strategy formulas)**.** Given a (positional, non-deterministic, labeled) attacker strategy $F \subseteq (G_a \times L \times G) \cap {\rightarrowtail}$ for the spectroscopy game $\mathcal{G}_\triangle$, the set of *strategy formulas*, Strat_F, is inductively defined by:
- If $\varphi \in \mathsf{Strat}_F(g'_a)$ and $(g_a, \langle b \rangle, g'_a) \in F$, then $\langle b \rangle \varphi \in \mathsf{Strat}_F(g_a)$,
- if $\varphi \in \mathsf{Strat}_F(g'_a)$ and $(g_a, \neg, g'_a) \in F$, then $\neg \varphi \in \mathsf{Strat}_F(g_a)$,
- if $\varphi \in \mathsf{Strat}_F(g_d)$ and $(g_a, \wedge, g_d) \in F$, then $\varphi \in \mathsf{Strat}_F(g_a)$, and
- if $\varphi_{g'_a} \in \mathsf{Strat}_F(g'_a)$ for all $g'_a \in I = \{g'_a \mid g_d \overset{*}{\rightarrowtail}_\triangle g'_a\}$ then $\bigwedge_{g'_a \in I} \varphi_{g'_a} \in \mathsf{Strat}_F(g_d)$.

Example 3.7. The attacks $(\mathsf{P}_1, \{\mathsf{P}_2\})_a \overset{\langle a \rangle}{\rightarrowtail} (b + c, \{b + d, c + d\})_a \overset{\wedge}{\rightarrowtail}\overset{*}{\rightarrowtail}\overset{\neg}{\rightarrowtail}\overset{\langle d \rangle}{\rightarrowtail} (\mathbf{0}, \varnothing)_a \overset{\wedge}{\rightarrowtail}$ on the system of Example 1.1 give rise to the formula $\langle a \rangle \bigwedge \{\neg \langle d \rangle \mathsf{T}\}$, which can be written as $\langle a \rangle \neg \langle d \rangle$.

The definition will never generate disjunctive subformulas: The target of a negation move is always a position $(p, \{q\})_a$ with singleton $\{q\}$, and only observation moves and negation moves are generated in such positions.

Definition 3.8 (Winning strategy graph)**.** Given the attacker winning region W_a and a starting position $g_0 \in W_a$, the *attacker winning strategy graph* F_a is the subset of the ${\rightarrowtail}$graph that can be visited from g_0 when following all ${\rightarrowtail}$-edges unless they lead out of W_a.

This graph can be cyclic; cycles are of the form $(p, \{q\})_a \overset{\neg}{\rightarrowtail} (q, \{p\})_a \overset{\neg}{\rightarrowtail} (p, \{q\})_a$, or introduced by recursive processes. (Remember that our examples are acyclic, but more

complex LTSs could contain cycles.) However, if the attacker plays inside their winning region according to F_a, they will always have paths to their final winning positions. So even though the attacker could loop (and thus introduce double negations or lose), they can always end the game and *win* in the sense of Definition 2.5 without generating double negations.

Example 3.9. Figure 6 continues the comparison of $a.b$ with $a.(a + b) + a.b.b + a$ from Example 3.4. The game constructs the following three distinguishing formulas:

- $\langle a\rangle\bigwedge\{\neg\langle a\rangle, \langle b\rangle\neg\langle b\rangle\}$. This formula has price $(3, 2, 1, 1, 1, 1)$ and shows that the processes can be distinguished by a failure trace, namely $\varnothing a\{a\}b\{b\}$: this is a failure trace of $a.b$ but not of $a + a.b.b + a.(a + b)$.
- $\neg\langle a\rangle\langle a\rangle$, with price $\mathsf{expr}(\widehat{\cdot}) = (2, 1, 0, 0, 1, 2)$. This formula shows that the processes are not in the impossible-futures preorder.
- $\langle a\rangle\bigwedge\{\neg\langle a\rangle, \neg\langle b\rangle\langle b\rangle, \langle b\rangle\}$. This formula has price $(3, 1, 0, 1, 1, 2)$ and shows that the processes can be distinguished by the possible-futures preorder. While this formula is strictly more expensive than $\neg\langle a\rangle\langle a\rangle$ (and therefore will be suppressed in the final output), it cannot be deleted altogether because it may be needed as part of a larger formula in other states. In particular, its negation is strictly cheaper than the negation of the first constructed formula, similar to Example 2.22.

The construction of other formulas is aborted as soon as they are recognized as more expensive in every relevant context, as we will discuss in the next subsection.

Let us quickly prove the soundness of the strategy formulas obtained from winning strategies.

Theorem 3.10. *If W_a is the attacker winning region of the spectroscopy game $\mathcal{G}_\triangle$ and F_a the derived strategy graph, every $\varphi \in \mathsf{Strat}_{F_a}((p, Q)_a)$ distinguishes p from every $q \in Q$.*

Proof. We proceed by induction on the structure of Strat_{F_a} with arbitrary p, Q.

- Assume $\varphi \in \mathsf{Strat}_{F_a}((p', Q')_a)$, and $((p, Q)_a, \langle b\rangle, (p', Q')_a) \in F_a$. By induction hypothesis, φ must be true for p' and false for all $q' \in Q'$. Due to the structure of $\mathcal{G}_\triangle$, we know that $p \xrightarrow{b} p'$ and that Q' contains all b-reachable states for Q. Thus, $\langle b\rangle\varphi \in \mathsf{Strat}_{F_a}((p, Q)_a)$ must be true for p and false for all $q \in Q$. Likewise, in the case of $((p, Q)_a^\lightning, \langle b\rangle, (p', Q')_a) \in F_a$, we find that $\langle b\rangle\varphi \in \mathsf{Strat}_{F_a}((p, Q)_a^\lightning)$ is distinguishing.
- Assume $\varphi \in \mathsf{Strat}_{F_a}((p', Q')_a)$ and $((p, Q)_a, \neg, (p', Q')_a) \in F_a$. By the construction of $\mathcal{G}_\triangle$, $Q = \{p'\}$ and $Q' = \{p\}$. By induction hypothesis, φ must be true for p' and false for p. So, $\neg\varphi \in \mathsf{Strat}_{F_a}((p, Q)_a)$ must be true for p and false for all elements of $\{p'\} = Q$.
- Assume $\varphi_{g_a'} \in \mathsf{Strat}_{F_a}(g_a')$ for all $g_a' \in I = \{g_a' \mid g_d \xrightarrow{*}_\triangle g_a'\}$, and also $((p, Q)_a, \wedge, g_d) \in F_a$. Due to the construction of $\mathcal{G}_\triangle$, $Q = \{q' \mid (p', \{q'\})_a \in I\} \cup \{q' \mid \exists Q'.\, q' \in Q' \wedge (p', Q')_a^\lightning \in I\}$ and $p' = p$. By induction hypothesis, every $\varphi_{g_a'}$ is true for p and false for all its respective $q' \in Q'$. So, the conjunction $\bigwedge_{g_a' \in I}\varphi_{g_a'} \in \mathsf{Strat}_{F_a}((p, Q)_a)$ must be distinguishing for p and Q. $\qquad\square$

Note that the theorem is only one-way, as every distinguishing formula can neutrally be extended by stating that some additional clause that is true for *both* processes does hold. Definition 3.6 will not find such bloated formulas.

We prove that a neatly constructed subset of the cheapest strategy formulas suffices for reaching completeness in Theorem 3.14.

$(a.b, \{a.(a+b) + a.b.b + a\})_\mathsf{a}$

$\langle a\rangle\bigwedge\{\neg\langle a\rangle, \langle b\rangle\neg\langle b\rangle\}; \langle a\rangle\bigwedge\{\neg\langle a\rangle, \neg\langle b\rangle\langle b\rangle, \langle b\rangle\};$
$\neg\langle a\rangle\langle a\rangle; \neg\neg\langle a\rangle\bigwedge\{\neg\langle a\rangle, \langle b\rangle\langle b\rangle, \langle b\rangle\}$

$\langle a\rangle$

$(b, \{a+b, b.b, \mathbf{0}\})_\mathsf{a}$

$\bigwedge\{\neg\langle a\rangle, \langle b\rangle\neg\langle b\rangle\};$
$\bigwedge\{\neg\langle a\rangle, \langle b\rangle\langle b\rangle, \langle b\rangle\}; \bigwedge\{\neg\langle a\rangle, \neg\langle b\rangle\langle b\rangle, \langle b\rangle\}$

$(a.(a+b) + a.b.b + a, \{a.b\})_\mathsf{a}$

$\langle a\rangle\langle a\rangle; \langle a\rangle\langle b\rangle\langle b\rangle; \langle a\rangle\neg\langle b\rangle\langle b\rangle;$
$\langle a\rangle\neg\langle b\rangle; \neg\langle a\rangle\bigwedge\{\neg\langle a\rangle, \langle b\rangle\neg\langle b\rangle\};$
$\neg\langle a\rangle\bigwedge\{\neg\langle a\rangle, \neg\langle b\rangle\langle b\rangle, \langle b\rangle\}; \neg\neg\langle a\rangle\langle a\rangle$

$\wedge$

$(b, \{\{a+b\}, \{b.b, \mathbf{0}\}\})_\mathsf{d}$

$\bigwedge\{\neg\langle a\rangle, \langle b\rangle\neg\langle b\rangle\}$

$\wedge$

$(b, \{\{a+b\}, \{b.b\}, \{\mathbf{0}\}\})_\mathsf{d}$

$\bigwedge\{\neg\langle a\rangle, \langle b\rangle\neg\langle b\rangle, \langle b\rangle\};$
$\bigwedge\{\neg\langle a\rangle, \neg\langle b\rangle\langle b\rangle, \langle b\rangle\}$

$*$

$(b, \{b.b, \mathbf{0}\})_\mathsf{a}^{\mathcal{A}}$

$\langle b\rangle\neg\langle b\rangle$

$(b, \{a+b\})_\mathsf{a}$

$\neg\langle a\rangle$

$\langle b\rangle$

$(b, \{b.b\})_\mathsf{a}$

$\langle b\rangle\neg\langle b\rangle; \neg\langle b\rangle\langle b\rangle; \neg\neg\langle b\rangle\langle b\rangle$

$(b, \{\mathbf{0}\})_\mathsf{a}$

$\langle b\rangle; \neg\neg\langle b\rangle$

$\langle a\rangle$

$\neg$

$(a+b, \{b\})_\mathsf{a}$

$\langle a\rangle; \neg\neg\langle a\rangle$

$\neg$

$(b.b, \{b\})_\mathsf{a}$

$\langle b\rangle\langle b\rangle; \neg\langle b\rangle\neg\langle b\rangle; \neg\neg\langle b\rangle\langle b\rangle$

$\langle b\rangle$

$(\mathbf{0}, \{b\})_\mathsf{a}$

$\neg\langle b\rangle$

$\langle b\rangle$

$\langle a\rangle$

$(\mathbf{0}, \varnothing)_\mathsf{a}$

T

$\wedge$

$(\mathbf{0}, \varnothing)_\mathsf{d}$

T

$\langle a\rangle$
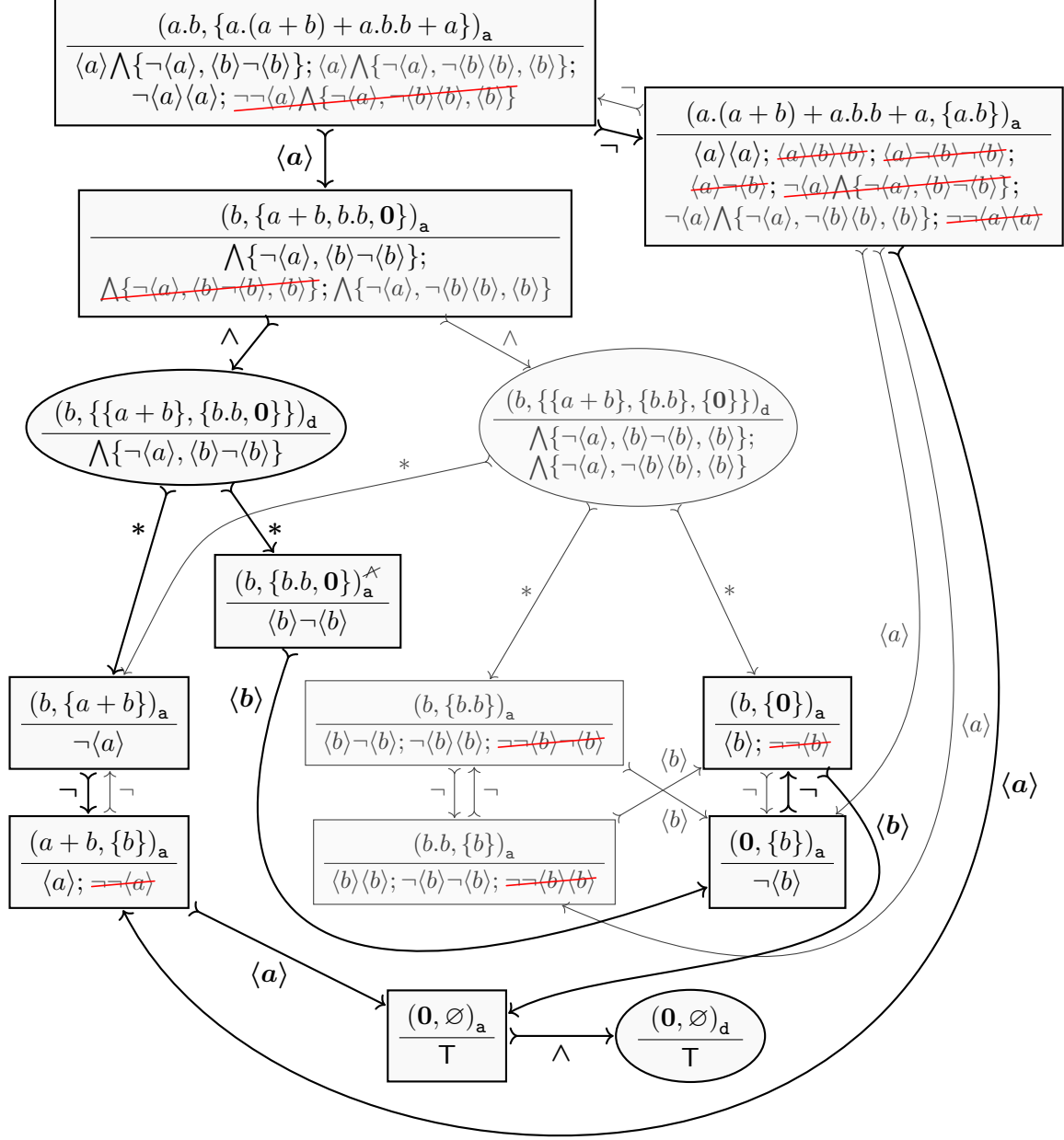
$\langle a\rangle$

$\langle b\rangle$

Figure 6: The winning region of the spectroscopy game of Figure 5, together with distinguishing formulas. Every node displays in the top half the game position and in the bottom half the relevant cheap strategy formulas in $\mathsf{Strat}_{F_\mathsf{a}}$ for that position. Formulas that are recognized as too expensive to be interesting are ~~pruned away~~, as will be explained more formally in Definition 3.12. Edges that are used to construct the main distinguishing formulas are **_thick._**

```
 1  def game_spectroscopy(S, p₀, q₀):
 2  │   G△ˢ = (G, Gₐ, ↦) := construct_spectroscopy_game(S)
 3  │   Wₐ := compute_winning_region(G△ˢ)
 4  │   if (p₀, {q₀})ₐ ∈ Wₐ :
 5  │   │   Fₐ := winning_graph(G△ˢ, Wₐ, (p₀, {q₀})ₐ)
 6  │   │   strats[] := ∅
 7  │   │   todo := [(p₀, {q₀})ₐ]
 8  │   │   while  todo ≠ []:
 9  │   │   │   g := todo.dequeue()
10  │   │   │   sg := strats[g]
11  │   │   │   if sg = undefined :
12  │   │   │   │   strats[g] := ∅
13  │   │   │   gg' := {g' | (g, ·, g') ∈ Fₐ ∧ strats[g'] = undefined}
14  │   │   │   if gg' = ∅ :
15  │   │   │   │   sg' = prune_dominated(Strat'_{Fₐ,strats}(g))
16  │   │   │   │   if sg ≠ sg' :
17  │   │   │   │   │   strats[g] := sg'
18  │   │   │   │   │   todo.enqueue_each_end({g* | (g*, ·, g) ∈ Fₐ ∧ g* ∉ todo})
19  │   │   │   else:
20  │   │   │   │   todo.enqueue_each_front(gg')
21  │   │   return strats[(p₀, {q₀})ₐ]
22  │   else:
23  │   │   R := {(p, q) | (p, {q})ₐ ∈ Gₐ \ Wₐ}
24  │   │   return R
```

Algorithm 2: Spectroscopy procedure.

Due to the cycles in the game graph, $\mathsf{Strat}_{F_\mathrm{a}}$ usually yields infinitely many formulas, unbounded in length and price. The next section will discuss how to find a finite subset of $\mathsf{Strat}_{F_\mathrm{a}}$ that solves Problem 3.2.

3.6. **Retrieving Cheapest Distinguishing Formulas.** We are now at the point where the recharting of the spectrum from Subsection 2.4 pays off. With the pricing metric on formulas, the coarsest ways of telling two states p, q apart are precisely given by the formulas $\Phi_\Delta \subseteq \mathsf{Strat}_{F_\mathrm{a}}((p, \{q\})_\mathrm{a})$ that are not dominated by any other formula in the set.

There might well be multiple such minimal-price formulas $\varphi_\Delta \in \Phi_\Delta$ with differing expr prices for the same pair of processes. Due to Lemma 2.23, the processes then are distinguished with respect to every equivalence X where the budget vector e_X from Table 1 is above the price of one of the formulas, that is, where $\mathsf{expr}(\widehat{\varphi_\Delta}) \sqsubseteq e_X$ for a $\varphi_\Delta \in \Phi_\Delta$. At the same time, the processes are preordered with respect to all the other notions of equivalence from the table, because these do not have sufficient distinguishing capabilities.

The extraction of cheapest formulas from $\mathsf{Strat}_{F_\mathrm{a}}$ presents itself as a very special kind of "shortest-distance" problem (cf. [Moh02]) between the positions immediately won by the attacker and the initial game position. It is just that the "distances" we have to keep track of are no numbers but sets of formulas. Those are peculiar as we have seen in Example 2.22.

We cannot really calculate with them but only compare their expr values and construct more complex formula sets.

The rest of this subsection is about designing a *fixed point algorithm* around this strange problem space. In order to be efficient, the algorithm should discard distinguishing formulas that are already recognized as not-cheapest as early as possible.

Algorithm 2 gives an overview of how the results of previous subsections and two following definitions play together. It constructs the spectroscopy game $\mathcal{G}^{\mathcal{S}}_{\triangle}$ (Definition 3.3), finds the attacker winning region using Algorithm 1 and computes its attacker winning strategy graph F_{a} (Definition 3.8). If the attacker cannot win, the algorithm returns a bisimulation relation derived from the defender winning region (Lemma 3.5). Otherwise, it constructs the distinguishing formulas: It keeps a map strats of strategy formulas that have been *found so far* and a list of game positions todo that have to be updated. In every round, we take a game position g from todo. If some of its successors have not been visited yet, we add them to the top of the work list. Otherwise we call $\mathsf{Strat}'_{F_{\mathrm{a}},\mathsf{strats}}(\mathsf{g})$ (to be introduced in Definition 3.11) to compute distinguishing formulas using the follow-up formulas strats. From the found formulas, prune_dominated removes formulas that are recognized as too expensive to be interesting (to be introduced in Definition 3.12). If the result changes strats[g], we enqueue each game predecessor to propagate the update there.

The fixed point approach needs Strat_F from Definition 3.6 in a more stepwise formulation that is not recursive. The function $\mathsf{Strat}'_{F,strats}$ mostly corresponds to Definition 3.6 with the twist that tentative follow-ups are used instead of recursion.

Definition 3.11 (Tentative strategy formulas). Given a labeled attacker strategy $F \subseteq (G_{\mathrm{a}} \times L \times G)$ for the spectroscopy game $\mathcal{G}_{\triangle}$ and an approximation of strategy formulas $strats \colon G \to \mathsf{HML}[\Sigma]$, the *next tentative strategy formulas,* $\mathsf{Strat}'_{F,strats}$, are defined for $g_{\mathrm{a}} \in G_{\mathrm{a}}$ and $g_{\mathrm{d}} \in G_{\mathrm{d}}$ as:

$$
\begin{aligned}
\mathsf{Strat}'_{F,strats}(g_{\mathrm{a}}) = \quad & \{\langle b \rangle \varphi \mid (g_{\mathrm{a}}, \langle b \rangle, g'_{\mathrm{a}}) \in F \wedge \varphi \in strats(g'_{\mathrm{a}})\} \\
& \cup \{\neg\varphi \mid (g_{\mathrm{a}}, \neg, g'_{\mathrm{a}}) \in F \wedge \varphi \in strats(g'_{\mathrm{a}})\} \\
& \cup \{\varphi \mid (g_{\mathrm{a}}, \wedge, g_{\mathrm{d}}) \in F \wedge \varphi \in strats(g_{\mathrm{d}})\} \\
\mathsf{Strat}'_{F,strats}(g_{\mathrm{d}}) = \quad & \{\textstyle\bigwedge_{g_{\mathrm{a}} \in I} \varphi_{g_{\mathrm{a}}} \mid I = \{g_{\mathrm{a}} \mid g_{\mathrm{d}} \overset{*}{\rightarrowtail}_{\triangle} g_{\mathrm{a}}\} \wedge \forall g_{\mathrm{a}} \in I.\, \varphi_{g_{\mathrm{a}}} \in strats(g_{\mathrm{a}})\}
\end{aligned}
$$

Intuitively, *strats* corresponds to intermediate candidates for reaching the target region in a shortest-distance problem. $\mathsf{Strat}'_{F,strats}$ corresponds to the addition of outgoing edge weights to the candidates when updating nodes. For a shortest-distance algorithm, we need one more ingredient: the criterion to select best candidates. In conventional shortest-distance problems, the minimum function (min) plays this role. Strat' already implicitly does something comparable to min: It forms the union of all the formula sets that are implied by outgoing game moves. This union on its own has the problem that formulas may be unfolded beyond all bounds, accumulating more and more information in arbitrarily expensive distinguishing formulas. For the fixed point algorithm to terminate, there must be a pointwise bound on the growth of *strats* with only finitely many possible sets of formulas below. For instance, one could restrict the formulas to have expressiveness prices below $(|S|, |S|, |S|, |S|, |S|, |S|)$. This would guarantee a safe solution, but it can result in insane running times. Instead, we use the following function:

Definition 3.12 (Pruning of dominated formulas). prune_dominated is defined as

$$\text{prune\_dominated}(\mathcal{O}) = \{\varphi \in \mathcal{O} \mid \quad (\nexists \varphi'. \, \varphi = \neg\neg\varphi')$$
$$\wedge \, (\nexists \psi \in \mathcal{O}. \, \text{expr}(\widehat{\psi}) \sqsubset \text{expr}(\widehat{\varphi})$$
$$\wedge \, ((\exists a, \varphi'. \, \varphi = \langle a \rangle \varphi') \rightarrow (\exists a, \psi'. \, \psi = \langle a \rangle \psi'))$$
$$\wedge \, ((\exists \varphi'. \, \varphi = \neg\varphi') \rightarrow (\exists \psi'. \, \psi = \neg\psi')))\}.$$

prune_dominated removes all formulas that are dominated with respect to the metrics by any other formula in this set, where observations may only be dominated by observations, and negations only by negations (but double negations are always dropped because we forbid them in distinguishing formulas). We need to keep minimum-price negations because of inversion of dominance in cases like Example 2.22. We also need to keep minimum-price observation formulas, at least in game positions $(p, \{q\})_\mathsf{a}$, because only such formulas are allowed under a negation. Other dominated formulas will not be important later on to find the cheapest options.

Example 3.13. If we look back at Figure 6 (used for Example 3.9), we can now see why it is important that dominated formulas are pruned away *at the right steps.* In the figure, pruned formulas are ~~striked out~~.

If a game position $(p, \{q\})_\mathsf{a}$ allows a positive and a negative distinguishing formula, we need to maintain the cheapest ones of either kind. For example, we kept both $\langle b \rangle \neg \langle b \rangle$ and the (more expensive) $\neg \langle b \rangle \langle b \rangle$ in $(b, \{b.b\})_\mathsf{a}$, which allowed to construct two minimal-price distinguishing formulas in $(b, \{\{a + b\}, \{b.b\}, \{\mathbf{0}\}\})_\mathsf{d}$: one formula with two positive branches and a small depth of negated observations, the other with one positive branch but deeper negated observations. However, one of them is pruned away in $(b, \{a + b, b.b, \mathbf{0}\})_\mathsf{a}$ because the latter position has another successor that generates a strictly cheaper formula.

All in all, the algorithm structure in Algorithm 2 is mostly usual fixed point machinery. It terminates because, for each state in a finite transition system, there must be a bound on the distinguishing mechanisms necessary with respect to our metrics. Strat' will only generate finitely many formulas under this bound. prune_dominated ensures that not too many more formulas are generated.

3.7. **Correctness of the Algorithm.** We will now prove that Algorithm 2—in spite of the pruning—generates enough formulas to solve Problem 3.2.

Theorem 3.14. *Assume a finite formula φ that distinguishes p from every $q \in Q$. Assume also that φ does not contain double negations or negated conjunctions (i.e. does not have subformulas of the form $\neg\neg\psi$ or $\neg\bigwedge_{i \in I}\psi_i$). Then we claim about the sets $\text{strats}[(p, Q)_\mathsf{a}]$ and $\text{strats}[(p, Q)_\mathsf{a}^{\curvearrowright}]$ of distinguishing formulas produced by Algorithm 2:*

(1) *$\text{strats}[(p, Q)_\mathsf{a}]$ contains a formula that is cheaper than or as cheap as φ.*
(2) *If φ is an observation formula, $\text{strats}[(p, Q)_\mathsf{a}^{\curvearrowright}]$ contains a formula that is cheaper than or as cheap as φ. (This necessarily is an observation formula.)*
(3) *If φ is an observation formula or a negation formula and Q is a singleton, $\text{strats}[(p, Q)_\mathsf{a}]$ contains a formula of the same kind that is cheaper than or as cheap as φ.*

Proof. First, $\text{strats}[(p, \varnothing)_\mathsf{a}] = \{\mathsf{T}\}$, and T is the cheapest of all formulas, so the claims hold if $Q = \varnothing$. In the rest of the proof we assume that $Q \neq \varnothing$ (and therefore $\varphi \neq \mathsf{T}$).

We prove the three claims of the theorem simultaneously by induction over the structure of φ with arbitrary p and Q.

Case $\varphi = \langle a \rangle \psi$: This means that there exists p' such that $p \xrightarrow{a} p'$ and ψ distinguishes p' from $Q' = \{q' \mid \exists q \in Q. \, q \xrightarrow{a} q'\}$. The game graph contains an observation move $(p, Q)_{\mathsf{a}} \overset{\langle a \rangle}{\rightarrowtail} (p', Q')_{\mathsf{a}}$ or $(p, Q)_{\mathsf{a}}^{\not\wedge} \overset{\langle a \rangle}{\rightarrowtail} (p', Q')_{\mathsf{a}}$.

Then we apply the induction hypothesis to $(p', Q')_{\mathsf{a}}$ and ψ, and we conclude that a formula $\psi' \in \mathsf{strats}[(p', Q')_{\mathsf{a}}]$ is found such that $\mathsf{expr}(\widehat{\psi'}) \sqsubseteq \mathsf{expr}(\widehat{\psi})$. Therefore, either $\langle a \rangle \psi' \in \mathsf{strats}[(p, Q)_{\mathsf{a}}]$ (respectively $\langle a \rangle \psi' \in \mathsf{strats}[(p, Q)_{\mathsf{a}}^{\not\wedge}]$); or, if $\langle a \rangle \psi'$ is pruned away, this must have been justified by a formula strictly cheaper than $\langle a \rangle \psi'$ in $\mathsf{strats}[(p, Q)_{\mathsf{a}}]$ (or $\langle a \rangle \psi' \in \mathsf{strats}[(p, Q)_{\mathsf{a}}^{\not\wedge}]$), and this cheaper formula must have been an observation formula according to Definition 3.12 (this is needed to prove Claim (3) if $|Q| = 1$). This proves all three claims.

Case $\varphi = \bigwedge_{i \in I} \psi_i$: (Only Claim (1) needs to be proven.) Note that $I \neq \varnothing$. Assign to every $q \in Q$ a conjunct ψ_i that distinguishes p from q; we denote this conjunct with ψ^q. If every $q \in Q$ is assigned the same formula ψ_{i_0} (e.g. if Q is a singleton), apply the induction hypothesis to $(p, Q)_{\mathsf{a}}$ and ψ_{i_0}.

Otherwise, define a partition of Q as follows:

$$Q^- = \{\{q\} \mid \psi^q \text{ is a negation formula}\}$$
$$Q^+ = \{\{q \mid \psi^q = \psi_i\} \mid \psi_i \text{ is an observation formula, for some } i \in I\} \setminus \{\varnothing\}$$
$$Q = Q^- \cup Q^+.$$

The set Q^- contains singleton blocks for every process in Q that is associated with a negation formula. Every block in Q^+ is associated with an observation formula. As we excluded nested conjunctions, each ψ_i can only be a negation or an observation formula. The game graph contains a conjunct challenge $(p, Q)_{\mathsf{a}} \overset{\triangle}{\rightarrowtail} (p, Q)_{\mathsf{d}}$. We apply the induction hypothesis to the targets of the corresponding conjunct answers:

- If some $\{q'\} \in Q$ is a singleton, we apply the induction hypothesis Claim (3) to $(p, \{q'\})_{\mathsf{a}}$ and formula $\psi^{q'}$. Then there will be a formula of the same kind as $\psi^{q'}$, denoted $\psi'_{\{q'\}} \in \mathsf{strats}[(p, \{q'\})_{\mathsf{a}}]$, with $\mathsf{expr}(\widehat{\psi'_{\{q'\}}}) \sqsubseteq \mathsf{expr}(\widehat{\psi^{q'}})$.

- If some $Q' \in Q$ is not a singleton, then $Q' \in Q^+$ and $\psi^{q'}$ (for $q' \in Q'$) is an observation formula. We apply the induction hypothesis Claim (2) to $(p, Q')_{\mathsf{a}}^{\not\wedge}$ and $\psi^{q'}$. Then there will be an observation formula, denoted $\psi'_{Q'} \in \mathsf{strats}[(p, Q')_{\mathsf{a}}^{\not\wedge}]$, with $\mathsf{expr}(\widehat{\psi'_{Q'}}) \sqsubseteq \mathsf{expr}(\widehat{\psi^{q'}})$.

Further note that if $\psi^{q'} = \langle a \rangle$ is a positive flat branch, then the formula found $\psi'_{Q'}$ is also a positive flat branch (otherwise it would be more expensive than $\psi^{q'}$). Then, the conjunction $\bigwedge_{Q' \in Q} \psi'_{Q'}$ satisfies:

- It contains no more positive branches than $\bigwedge_{i \in I} \psi_i$;[2]
- It contains no more positive deep branches than $\bigwedge_{i \in I} \psi_i$;
- For every $\psi'_{Q'}$, there is a ψ_i such that $\mathsf{expr}(\widehat{\psi'_{Q'}}) \sqsubseteq \mathsf{expr}(\widehat{\psi_i})$.

From this, we can conclude that $\mathsf{expr}(\widehat{\bigwedge_{Q' \in Q} \psi'_{Q'}}) \sqsubseteq \mathsf{expr}(\widehat{\bigwedge_{i \in I} \psi_i})$.

---

[2] But the number of positive *flat* branches (with observation height 1) could have been increased. This is why the first change described in Remark 2.25 is necessary.

Now either $\bigwedge_{Q' \in Q} \psi'_{Q'} \in \mathsf{strats}[(p, Q)_\mathsf{a}]$, or a formula that is strictly cheaper is in $\mathsf{strats}[(p, Q)_\mathsf{a}]$. Claim (1) holds in both cases.

Case $\varphi = \neg\psi$: Note that ψ is an observation formula. Assume for now that Q is a singleton $\{q\}$. The game graph contains a negation move $(p, \{q\})_\mathsf{a} \rightarrowtail (q, \{p\})_\mathsf{a}$.

We apply the induction hypothesis Claim (3) to $(q, \{p\})_\mathsf{a}$ and ψ and conclude that there is some observation formula $\psi' \in \mathsf{strats}[(q, \{p\})_\mathsf{a}]$ with $\mathsf{expr}(\widehat{\psi'}) \sqsubseteq \mathsf{expr}(\widehat{\psi})$. But then, either $\neg\psi' \in \mathsf{strats}[(p, \{q\})_\mathsf{a}]$, or a negation formula strictly cheaper than $\neg\psi'$ is in $\mathsf{strats}[(p, \{q\})_\mathsf{a}]$. This proves Claims (1) and (3) for singleton Q.

Now if Q contains more than one element, only Claim (1) needs to be proven. The game graph contains a conjunct challenge $(p, Q)_\mathsf{a} \rightarrowtail (p, \{\{q\} \mid q \in Q\})_\mathsf{d}$, and we can apply the above argumentation to the targets $(p, \{q\})_\mathsf{a}$ of the conjunct answers. So, for every $q \in Q$, there is some negation formula $\neg\psi^q \in \mathsf{strats}[(p, \{q\})_\mathsf{a}]$ with $\mathsf{expr}(\widehat{\neg\psi^q}) \sqsubseteq \mathsf{expr}(\widehat{\neg\psi})$. Consequently, either the conjunction $\bigwedge_{q \in Q} \neg\psi^q \in \mathsf{strats}[(p, Q)_\mathsf{a}]$, or a formula that is strictly cheaper is in $\mathsf{strats}[(p, Q)_\mathsf{a}]$. As formula prices do not count negative branches in a conjunction, $\mathsf{expr}(\widehat{\bigwedge_{q \in Q} \neg\psi^q}) = \mathsf{expr}(\bigwedge_{q \in Q} \neg\psi^q) \sqsubseteq \mathsf{expr}(\bigwedge\{\neg\psi\}) = \mathsf{expr}(\widehat{\neg\psi})$. So the claim on formula prices holds.[3] $\square$

In Subsection 2.4, we have chosen to describe limits on upper bounds by the depth of negated observations (dimension 6). An alternative choice could have been to restrict the number of negative deep branches, but that would make the correctness proof of Theorem 3.14 more difficult. The proof uses the technical property that a formula with more negative branches is not more expensive than a similar formula with fewer (but more than zero) negative branches. A formula with one negative branch gives another, technical, reason why negations under observations should count as implicit conjunctions.

Using the theorem, we can state the correctness of the algorithm easily.

Corollary 3.15. *Assume a spectroscopy game position $(p, Q)_\mathsf{a}$. For every language in Definition 2.14 (and, in relevant cases, $\mathcal{O}_\mathrm{E}$) that distinguishes p from every $q \in Q$, Algorithm 2 will find a minimal-price formula in the distinguishing formulas (without double negations or negated conjunctions) in this language.*

Proof. Assume that some observation language $\mathcal{O}_X$ distinguishes p from every $q \in Q$. In line with Subsection 2.5, we can limit our attention to formulas without double negations or negated conjunctions. Then there exists some formula $\varphi \in \mathcal{O}_X$ with minimal formula price that distinguishes p from every $q \in Q$.

Now apply Theorem 3.14 to φ. Algorithm 2 will therefore find a formula with the same formula price (price-minimality implies that there is no cheaper distinguishing formula than φ), which consequently is in the same distinguishing language $\mathcal{O}_X$. $\square$

Example 3.16. Now that we have proven the main correctness theorem, we can conclude even more about the minimal-price formulas found in Example 3.9. As stated there, the processes are not preordered by failure traces nor by impossible futures ($a.b \not\preceq_\mathrm{FT} a.(a + b) + a.b.b + a$ and $a.b \not\preceq_\mathrm{IF} a.(a + b) + a.b.b + a$), but Corollary 3.15 also implies that these are the coarsest ways of telling the processes apart. The processes are in every preorder in Figure 2 that is not above one of these two. So, we can conclude that they are simulation-preordered and readiness-preordered.

---

[3]If $\varphi = \neg\langle a\rangle\neg\langle b\rangle$, the algorithm might find a formula of the form $\neg\langle a\rangle\bigwedge\{\neg\langle b\rangle, \neg\langle a\rangle\}$. The latter formula should not be more expensive than the former. This is why the second change in Remark 2.25 is necessary.

3.8. **Differences to the Conference Version.** The game and algorithm as presented here have an important change when compared to the conference version from TACAS'21 [BN21].

In the conference version, each attacker position had only one outgoing conjunction move. Definition 3.3 now has one conjunction move from $(p, Q)_{\mathsf{a}}$ *per partition* of Q. This adds a lot of possible game positions and moves. However, this has been necessary to fix an error in the algorithm.

In order to understand the problem, let us reexamine the graph in Figure 6. The TACAS'21 version would only contain the move from $(b, \{a + b, b.b, \mathbf{0}\})_{\mathsf{a}}$ to the finest partition $(b, \{\{a + b\}, \{b.b\}, \{\mathbf{0}\}\})_{\mathsf{d}}$ but not the one to $(b, \{\{a + b\}, \{b.b, \mathbf{0}\}\})_{\mathsf{d}}$. This move, however, was instrumental in finding the failure-trace formula $\langle a \rangle \bigwedge \{\langle b \rangle \neg \langle b \rangle, \neg \langle a \rangle\}$ in Example 3.9.

So, without the correction, the algorithm published originally picks the ready-trace formula $\langle a \rangle \bigwedge \{\langle b \rangle, \langle b \rangle \neg \langle b \rangle, \neg \langle a \rangle\}$, showing that the processes can be distinguished by a ready trace, namely $\{a\}a\{b\}b\varnothing$. While this formula does distinguish $a.b$ from $a.(a + b) + a.b.b + a$, and even is semantically equivalent to the failure-trace formula above, it is not in the cheapest possible language. Erroneously assuming ready traces to be a minimal language to tell the processes apart leads to the wrong conclusion that the processes would be failure-trace-preordered, which they are not.

The TACAS'21 version lacked a theorem like Theorem 3.14. The paper, however, was confident that a similar result could be established. In light of the problem presented here, it is clear that the original algorithm was not complete in the sense of the theorem.

There are several ways of correcting the problem. We have chosen the least invasive fix by considering all partitions and not just the finest ones. Together with some slight simplifications of metric and pruning, this allowed us to provide a completeness proof while staying close to the conference version.

4. Linear-Time–Branching-Time Spectroscopy in the Browser

There are two implementations of the algorithm described in this article. In Subsection 4.1, we describe the analysis of transition systems in a web tool built around the algorithm. In Subsection 4.2, we give a short account of a computer game implementation.

4.1. **A Webtool for Equivalence Spectroscopy.** We have implemented the game and the generation of minimal distinguishing formulas in the "Linear-Time–Branching-Time Spectroscope," a Scala.js program that can be run in the browser on `https://concurrency-theory.org/ltbt-spectroscope/`.

The tool (screenshot in Figure 7) consists of a text editor to input finite-state CCS-style processes (including recursively defined processes) and a view of the transition system graph. When queried to compare two processes, the tool yields the cheapest distinguishing HML-formulas it can find for both directions. From the found formulas, the tool infers the finest fitting preorders for the relevant pairs of processes. In the process, distinguishing formulas that do not contribute to this question are discarded in order to reduce the noise of the output.

To "benchmark" the quality of the distinguishing formulas, we have run the algorithm on all the finitary counterexample processes from the report version of "The Linear-Time–Branching-Time Spectrum" [vG01]. Table 2 reports the output of our tool, on how to distinguish certain processes. The results match the (in)equivalences given in [vG01]. In some cases, the tool finds slightly better ways of distinction using impossible futures equivalence,
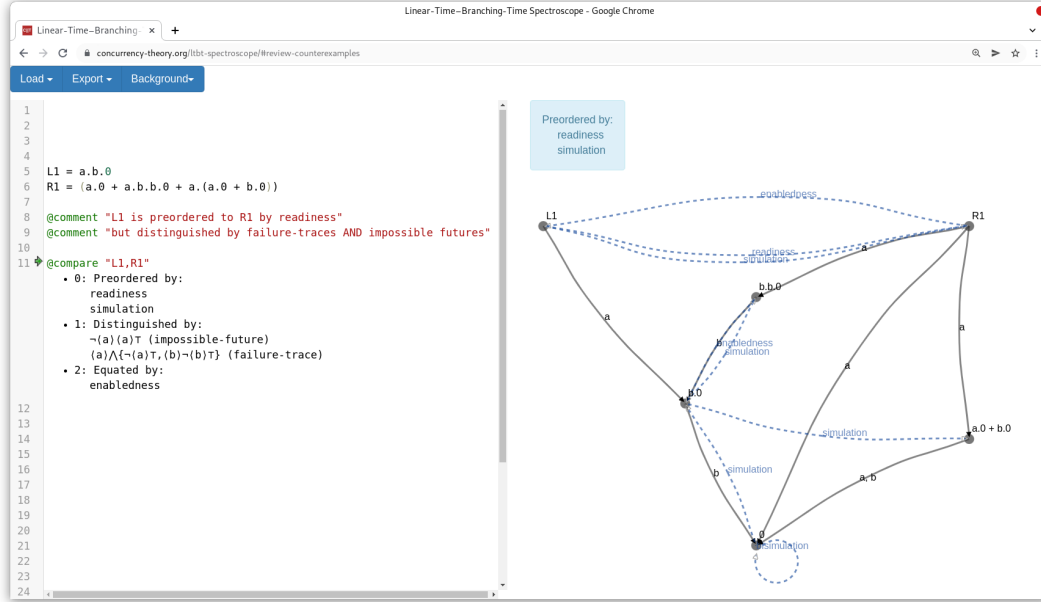
Figure 7: Screenshot of a linear-time–branching-time spectroscopy of the processes from Example 3.4.

Table 2: Formulas found by our implementation for some interesting processes from [vG01].

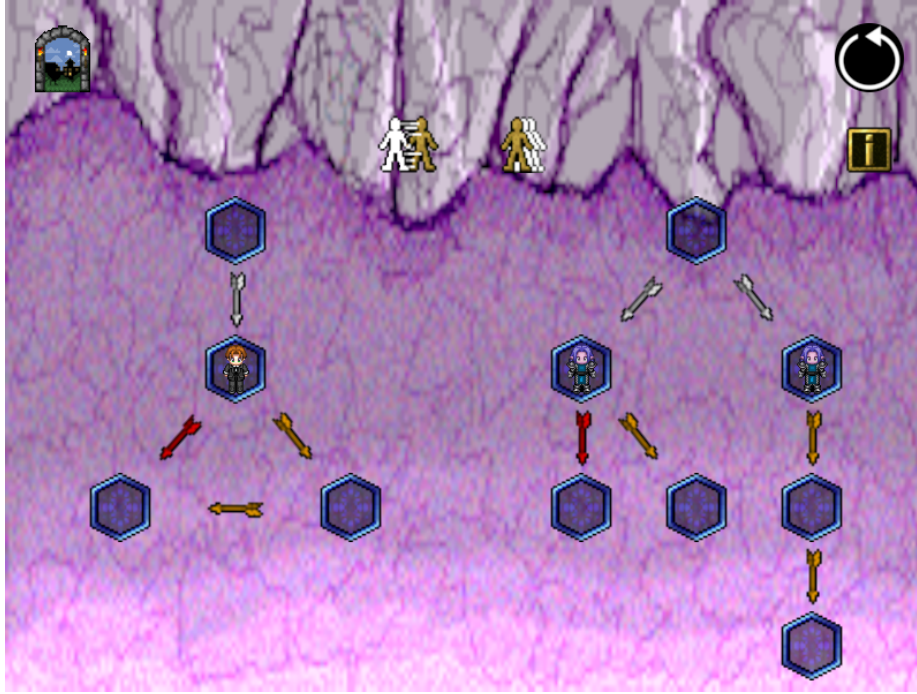| p	q	Cheapest distinguishing formulas found	From
$a.b + a$	$a.b$	$\langle a \rangle \neg \langle b \rangle \in \mathcal{O}_F$	p. 13
$a.b + a.(b + c)$	$a.(b + c)$	$\langle a \rangle \neg \langle c \rangle \in \mathcal{O}_F$	p. 16
P_3	P_4	The formulas explained in Example 2.17	p. 21
$a.b + a.(b+c) + a.c$	$a.b + a.c$	$\langle a \rangle \bigwedge \{\langle c \rangle, \langle b \rangle\} \in \mathcal{O}_R \cap \mathcal{O}_S$	p. 24
$a.(b + a.(b+c.d) + a.c.e) + a.(a.c.d + a.(c.e + b))$	$a.(a.(b + c.d) + a.c.e) + a.(a.c.d + a.(c.e + b) + b)$	$\langle a \rangle \bigwedge \{\langle b \rangle, \langle a \rangle \bigwedge \{\langle c \rangle \langle d \rangle, \langle b \rangle\}\} \in \mathcal{O}_{RT} \cap \mathcal{O}_S,$ $\langle a \rangle \bigwedge \{\neg \langle b \rangle, \langle a \rangle \bigwedge \{\langle c \rangle \langle d \rangle, \neg \langle b \rangle\}\} \in \mathcal{O}_{FT}$	p. 27
$a.(b.c + b.d)$	$a.b.c + a.b.d$	$\langle a \rangle \bigwedge \{\langle b \rangle \langle c \rangle, \langle b \rangle \langle d \rangle\} \in \mathcal{O}_{PF} \cap \mathcal{O}_S$	p. 31
$a.b.c + a.(b.c + b.d)$	$a.(b.c + b.d)$	$\langle a \rangle \neg \langle b \rangle \langle d \rangle \in \mathcal{O}_{IF}$	p. 34
$a.b + a + a.c$	$a.b + a.(b+c) + a.c$	$\langle a \rangle \bigwedge \{\neg \langle b \rangle, \neg \langle c \rangle\} \in \mathcal{O}_F$	p. 38
$a.b.c + a.(b.c + b)$	$a.(b.c + b)$	$\langle a \rangle \neg \langle b \rangle \neg \langle c \rangle \in \mathcal{O}_B$	p. 42

Figure 8: Screenshot of the browser game "The Spectroscopy Invaders."

which was not known at the time of the original paper. All the computed formulas are reasonably small.

For each of the examples (from papers) we have considered, the browser's capacities sufficed to run the algorithm with pruning in 30 to 250 milliseconds. This does not mean that one should expect the algorithm to work for systems with thousands of states. There, the exponentialities of game and formula construction would hit. However, such big instances would usually stem from preexisting models where one would very much hope for the designers to already know under which semantics to interpret their model. The practical applications of our browser tool are more on the research side: When devising compiler optimizations, encodings, or distributed algorithms, it can be very handy to fully grasp the equivalence structure of isolated instances. The Linear-Time–Branching-Time Spectroscope supports this process.

4.2. **A Spectroscopy Browser Game.** In order to make the spectroscopy game more explainable, Trzeciakiewicz [Trz21] implemented the computer game "The Spectroscopy Invaders" where one plays the attacker. A play of the game corresponds to constructing a distinguishing formula. To reach higher scores, one has to construct minimal formulas in the sense of this paper. Under the hood, a TypeScript implementation of Algorithm 2 is used. The game can be played in the browser at `https://concurrency-theory.org/ltbt-game/`.

A screenshot of the game is given in Figure 8. The player's task is to distinguish $w.(r+y.y)$ (left) from $w.(r+y) + w.y.y$ (right). The current game position is $(r + y.y, \{r + y, y.y\})_a$, as indicated by the human figure in the left LTS and the two elves in the right LTS. The player can click the "conjunction" button 🎄 , which will split the position into $(r + y.y, \{r + y\})_a$

and $(r + y.y, \{y.y\})_{\mathsf{a}}$. After that, the player can click on a successor state in the left LTS to indicate an observation move.

Trzeciakiewicz [Trz21] limits the scope to trace, failure, possible-future, simulation, and bisimulation equivalences, excluding readiness, ready traces, and failure traces. In effect, the spectrum can be relieved of the issues that come from situations where positive deep branches in a conjunction have to be counted. This is another way of circumventing the problems discussed in Subsection 3.8. In particular, this allows the game mechanics to stay clear of non-trivial partitions, as they did originally in [BN21].

Moreover, the game is single-player: there is no defender picking a conjunction answer. Instead, the attacker has to name attacks for every right-hand state. Due to nested conjunctions, the game positions thus actually are sets of $(p, Q)_{\mathsf{a}}$ tuples.

5. Related Work and Alternatives

The game and the algorithm presented fill a blank spot in between the following previous directions of work:

Distinguishing Formulas in General. Cleaveland [Cle91] showed how to restore (small but non-minimal) distinguishing formulas for bisimulation equivalence from the execution of a bisimilarity checker based on the splitting of blocks. He mentioned as possible future work to extend the construction to other notions of the spectrum. We are not aware of any place where this has previously been done completely. Korver [Kor92] extended this work to branching bisimulation; he needed to extend HML with an until modality that restricts intermediate internal steps. For example, $\varphi_1\langle a\rangle\varphi_2$ means "the current state can take an a-step to a state that satisfies φ_2, while every intermediate state visited before the a-step satisfies φ_1." However, the algorithm in [Kor92] would not always produce a minimal formula in any sense. There are related islands like the encoding between CTL and failure traces by Bruda and Zhang [BZ10]. There is also more recent work like Jasper et al. [JSS20] extending to the generation of characteristic invariant formulas for bisimulation classes, and like Wißmann et al. [WMS21] about generating distinguishing formulas for bisimulation in a general coalgebraic setting. Previous algorithms for bisimulation inequivalence tend to generate formulas that alternate $\langle a\rangle$ and $[b] \equiv \neg\langle b\rangle\neg$ observations. Such formulas can not as easily be linked to the spectrum as ours.

Game Characterizations of the Spectrum. After Shukla et al. [SHR96] had shown how to characterize many notions of equivalence by HORNSAT games, Chen and Deng [CD08] presented a hierarchy of games characterizing all the equivalences of the linear-time–branching-time spectrum. The games from [CD08] allow word moves and thus are infinite already for finite transition systems with cycles. Therefore, they cannot be applied as easily as ours in algorithms. Constructing distinguishing formulas from attacker strategies of these games would be less convenient than in our solution. Their parametric approach is comparable to fixing maximal price budgets *ex ante*. Our on-the-fly picking of minimal prices provides more flexibility.

Using Game Characterizations for Distinguishing Formulas. There is recent work by Mika-Michalski et al. [KMS20] on constructing distinguishing formulas using games in a more abstract coalgebraic setting focussed on the absence of bisimulation. The game and formula generation there, however, cannot easily be adapted for our purpose of performing a *spectroscopy* also for weaker notions.

Generalizations of the Spectrum. As a by-product of our work, Subsection 2.4 has recharted the linear-time–branching-time spectrum in a way that corresponds nicely with our algorithm. Each coordinate in the price lattice can be seen as characterizing a conceivable notion of equivalence. Thus, this characterization generalizes (a part of) the [vG01]-spectrum. For instance, we have not considered the revivals semantics [Ros09], but it can easily be characterized by $e_\mathrm{R} \sqcap e_\mathrm{FT} = (\infty, 1, 0, 1, 1, 1)$. So let us briefly mention important generalizations of the spectrum.

For instance, Lange et al. [LLV14] introduce a *higher-order dyadic μ-calculus* where each single formula characterizes a notion of equivalence. This, too, allows a more unified computational treatment of the spectrum's members and many conceivable siblings.

The most complete *observational generalization* of [vG01] has been provided by de Frutos Escrig et al. [dFGPR13]. Their main dimensions are five layers of simulation, which correspond to classes of localizable annotations on an observaion tree (viz.: no local observations, deadlock detection, enabled actions, enabled traces, enabled trees), and in which way to compare these annotations. The second main dimension actually is a combination of 5 to 9 sub-dimensions: (1) whether to compare only along one linear path, deterministic branching paths or all branching paths, whether to check for (2) superset and/or (3) subset relationship between annotations, and whether to do (each of) this (4) along the paths and/or (5) at the leafs of observation trees. Their lattice cannot only cover interesting notions of equivalence unreachable by ours (for instance impossible-futures-along-a-trace), but is also nicely linked to a unified system of axiomatic characterizations. Still, their characterization is less general than our price lattice in other regards. For instance, it cannot cover counting equivalences like i-step bisimilarity, $(i, \infty, \infty, \infty, \infty, \infty)$ or n-nested similarity.

Apartness and Simulation Distances. Published around the same time as the conference version of the present paper, Geuvers and Jacobs [GJ21] discussed *apartness* as the inductively defined dual of bisimulation. This is closely related to the approach of the present paper. We also tackle the hierarchy of equivalence problems by rather addressing the dual question of how easy it is to tell two processes apart. In this view, the expressiveness prices output by our algorithm provide information on "how far apart" two models are.

Another view on "how far apart" models are is provided by *(bi)simulation distances* as researched by Černý et al. [ČHR12], and by Romero Hernández and de Frutos Escrig [RdF12]. In this line of work, distances quantify how many and how relevant changes need to be made to a process in order to make it equivalent to another process with respect to a fixed semantics. This is orthogonal to what we are doing: Bisimulation distances are about "how much cheating" in a (bi)simulation game the defender would need to win. Our multidimensional discrete expressiveness prices are about "how much semantics" the attacker needs to win in a bisimulation game without cheating.

Alternatives. One can also find the finest notion of equivalence between two states by *gradually minimizing* the transition system with ever coarser equivalences from bisimulation to trace equivalence until the states are conflated (possibly also trying branches of the spectrum). Within a big tool suite of highly optimized algorithms this should be quite efficient. We preferred the game approach, because it can uniformly be extended to the whole spectrum and also has the big upside of explaining the in-equivalences by distinguishing formulas.

A variation of our approach, which we have already tried, is to run the formula search on a *directed acyclic subgraph* of the winning strategy graph. For our purpose of finding most fitting equivalences, DAG-ification may preclude the algorithm from finding the right formulas. On the other hand, if one for instance is mainly interested in a short distinguishing formula with low depth, one can speed up the process with DAG-ification by the order of remaining game rounds.

Optimizations. The algorithm in this article has more than exponential time complexity. To improve on that, one could reduce the number of conjunct challenges generated as follows. First one only explores the finest conjunct challenge move $(p, Q)_\mathsf{a} \overset{\triangle}{\rightarrowtail} (p, \{\{q\} \mid q \in Q\})_\mathsf{d}$, and if that succeeds and leads to a formula with multiple positive branches, one only needs to explore one or two additional conjunct challenges to find out whether there is a formula with a single positive (deep) branch. Only formulas with one positive (deep) branch may be in a smaller language than the previously found formula. This faster variant would still generate distinguishing formulas from the cheapest languages (i.e. they solve Problem 3.1) but these formulas are not always the cheapest ones in themselves (i.e. they do not solve Problem 3.2). This would require to interleave the game graph construction (line 2 of our Algorithm 2) with the generation of distinguishing formulas (lines 5–20).

There are some other avenues for small optimizations: For instance, the pruning of Definition 3.12 can be changed to take game positions into account. If the right-hand state set Q of $(p, Q)_\mathsf{a}$ has more than one element, dominated formulas can be pruned regardless of their kind. Also, the game construction can be sped up a bit by stopping the construction of attacker moves in positions that cannot be winning for the attacker. In particular, this is the case for $(p, Q)_\mathsf{a}$ if $p \in Q$. The downside of this is that it becomes a little more difficult to retrieve a proper bisimulation relation in positions where the attacker does not win.

6. Conclusion

In this paper, we have established a convenient way of finding distinguishing formulas that use a minimal amount of expressiveness.

System analysis tools can employ the algorithm to tell their users in more detail *how equivalent* two process models are. While the generic approach is costly, instantiations to more specific, symbolic, compositional, on-the-fly or depth-bounded settings may enable wider applications. There are also some algorithmic tricks (like building the concrete formulas only after having found the price bounds and heuristics in handling the game graph) we have not explored in this paper.

More clever ways of characterizing the spectrum, the formula prices, and instances of admissible pruning might further improve the applicability of the approach.

So far, we have only looked at *strong* notions of equivalence [vG90]. It is an interesting question how to extend our algorithm, so it also deals with *weak* notions of equivalence [vG93].

These equivalences abstract over τ-actions representing "internal activity" and correspond to observation languages with a special temporal $\langle\epsilon\rangle$-observation (cf. [GFM20]). Such an extension would generalize work on weak game characterizations such as de Frutos Escrig et al.'s [dFKW17] and our own [BN19, BNP20, BM21, SMJ16].

A roadblock to expanding the approach to the weak spectrum lies in the fact that, with the conventional modal characterizations of the weak equivalences, the observation languages are not closed. Subformulas would need to be more expensive than their parent formulas. For instance, $\langle\epsilon\rangle\langle a\rangle\langle\epsilon\rangle\mathsf{T}$ is a valid weak trace observation but its subformula $\langle a\rangle\langle\epsilon\rangle\mathsf{T}$ is not. In order to apply our algorithm, one would need a different modal language, perhaps similar to Korver's until operator [Kor92].

The vision is to arrive at *one* certifying algorithm that can yield finest equivalences and cheapest distinguishing formulas as witnesses for the whole spectrum.

References

[Bis21]   Benjamin Bisping. Linear-time–branching-time spectroscope: TACAS 2021 edition, 2021. Archived on Zenodo. `doi:10.5281/zenodo.4475878`.

[Bis22]   Benjamin Bisping. Linear-time–branching-time spectroscope v0.2.0, 2022. Archived on Zenodo. `doi:10.5281/zenodo.6726494`.

[BM21]   Benjamin Bisping and Luisa Montanari. A game characterization for contrasimilarity. In Ornela Dardha and Valentina Castiglioni, editors, *Proceedings Combined 28th International Workshop on Expressiveness in Concurrency and 18th Workshop on Structural Operational Semantics*, volume 339 of *Electronic Proceedings in Theoretical Computer Science*, pages 27–42. Open Publishing Association, 2021. `doi:10.4204/EPTCS.339.5`.

[BN19]   Benjamin Bisping and Uwe Nestmann. Computing coupled similarity. In Tomáš Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems: TACAS*, volume 11427 of *LNCS*, pages 244–261. Springer, Cham, 2019. `doi:10.1007/978-3-030-17462-0_14`.

[BN21]   Benjamin Bisping and Uwe Nestmann. A game for linear-time–branching-time spectroscopy. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems: TACAS*, volume 12651 of *LNCS*, pages 3–19. Springer, Cham, 2021. `doi:10.1007/978-3-030-72016-2_1`.

[BNP20]   Benjamin Bisping, Uwe Nestmann, and Kirstin Peters. Coupled similarity: the first 32 years. *Acta Informatica*, 57(3–5):439–463, 2020. `doi:10.1007/s00236-019-00356-4`.

[BZ10]   Stefan D. Bruda and Zhiyu Zhang. Model checking is refinement: From computation tree logic to failure trace testing. In José Cordeiro, Maria Virvou, and Boris Shishkov, editors, *Proceedings of the 5th International Conference on Software and Data Technologies: ICSOFT,* Volume 2, pages 173–178. SciTePress, Setúbal, 2010. `doi:10.5220/0003006801730178`.

[CD08]   Xin Chen and Yuxin Deng. Game characterizations of process equivalences. In G. Ramalingam, editor, *Programming Languages and Systems: APLAS*, volume 5356 of *LNCS*, pages 107–121. Springer, Berlin, 2008. `doi:10.1007/978-3-540-89330-1_8`.

[CH93]   Rance Cleaveland and Matthew Hennessy. Testing equivalence as a bisimulation equivalence. *Formal Aspects of Computing*, 5(1):1–20, 1993. `doi:10.1007/BF01211314`.

[ČHR12] Pavol Černý, Thomas A. Henzinger, and Arjun Radhakrishna. Simulation distances. *Theoretical Computer Science*, 413(1):21–35, 2012. Quantitative Aspects of Programming Languages (QAPL 2010). `doi:10.1016/j.tcs.2011.08.002`.

[Cle91] Rance Cleaveland. On automatically explaining bisimulation inequivalence. In E. M. Clarke and R. P. Kurshan, editors, *Computer-Aided Verification: CAV'90*, volume 531 of *LNCS*, pages 364–372. Springer, Berlin, 1991. `doi:10.1007/BFb0023750`.

[dFGPR13] David de Frutos Escrig, Carlos Gregorio Rodríguez, Miguel Palomino, and David Romero Hernández. Unifying the Linear Time-Branching Time Spectrum of Strong Process Semantics. *Logical Methods in Computer Science*, 9(2:11):1–74, 2013. `doi:10.2168/LMCS-9(2:11)2013`.

[dFKW17] David de Frutos Escrig, Jeroen J. A. Keiren, and Tim A. C. Willemse. Games for bisimulations and abstraction. *Logical Methods in Computer Science*, 13(4:15):1–40, 2017. `doi:10.23638/LMCS-13(4:15)2017`.

[GFM20] Maciej Gazda, Wan Fokkink, and Vittorio Massaro. Congruence from the operator's point of view: Syntactic requirements on modal characterizations. *Acta Informatica*, 57(3–5):329–351, 2020. `doi:10.1007/s00236-019-00355-5`.

[GJ21] Herman Geuvers and Bart Jacobs. Relating apartness and bisimulation. *Logical Methods in Computer Science*, 17(3:15):1–35, 2021. `doi:10.46298/LMCS-17(3:15)2021`.

[Grä07] Erich Grädel. Finite model theory and descriptive complexity. In Erich Grädel, Phokion G. Kolaitis, Leonid Libkin, Maarten Marx, Joel Spencer, Moshe Y. Vardi, Yde Venema, and Scott Weinstein, editors, *Finite Model Theory and Its Applications*, Texts in Theoretical Computer Science, pages 125–230. Springer, Berlin, 2007. `doi:10.1007/3-540-68804-8_3`.

[HM80] Matthew Hennessy and Robin Milner. On observing nondeterminism and concurrency. In J. W. de Bakker and J. van Leeuwen, editors, *Automata, Languages and Programming*, volume 85 of *LNCS*, pages 299–309. Springer, Berlin, 1980. `doi:10.1007/3-540-10003-2_79`.

[HM85] Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985. `doi:10.1145/2455.2460`.

[JSS20] Marc Jasper, Maximilian Schlüter, and Bernhard Steffen. Characteristic invariants in Hennessy–Milner logic. *Acta Informatica*, 57(3–5):671–687, 2020. `doi:10.1007/s00236-020-00376-5`.

[KE99] Antonín Kučera and Javier Esparza. A logical viewpoint on process-algebraic quotients. In Jörg Flum and Mario Rodriguez-Artalejo, editors, *Computer Science Logic: CSL*, volume 1683 of *LNCS*, pages 499–514. Springer, Berlin, 1999. `doi:10.1007/3-540-48168-0_35`.

[KMS20] Barbara König, Christina Mika-Michalski, and Lutz Schröder. Explaining non-bisimilarity in a coalgebraic approach: Games and distinguishing formulas. In Daniela Petrişan and Jurriaan Rot, editors, *Coalgebraic Methods in Computer Science: CMCS*, volume 12094 of *LNCS*, pages 133–154. Springer, Cham, 2020. `doi:10.1007/978-3-030-57201-3_8`.

[Kor92] Henri Korver. Computing distinguishing formulas for branching bisimulation. In K. G. Larsen and A. Skou, editors, *Computer Aided Verification: CAV'91*, volume 575 of *LNCS*, pages 13–23. Springer, Berlin, 1992. `doi:10.1007/3-540-55179-4_3`.

[LLV14] Martin Lange, Etienne Lozes, and Manuel Vargas Guzmán. Model-checking process equivalences. *Theoretical Computer Science*, 560(3):326–347, 2014. `doi:10.1016/j.tcs.2014.08.020`.

[Mil90] Robin Milner. Operational and algebraic semantics of concurrent processes. In Jan van Leeuwen, editor, *Handbook of theoretical computer science, Vol. B: Formal methods and semantics*, pages 1201–1242. Elsevier, Amsterdam, 1990. `doi:10.1016/B978-0-444-88074-1.50024-X`.

[Moh02] Mehryar Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002. `doi:10.25596/jalc-2002-321`.

[RdF12] David Romero Hernández and David de Frutos Escrig. Defining distances for all process semantics. In Holger Giese and Grigore Rosu, editors, *Formal Techniques for Distributed Systems: FMOODS and FORTE*, volume 7273 of *LNCS*, pages 169–185. Springer, Heidelberg, 2012. `doi:10.1007/978-3-642-30793-5_11`.

[Ros09] A. W. Roscoe. Revivals, stuckness and the hierarchy of CSP models. *Journal of Logic and Algebraic Programming*, 78(3):163–190, 2009. `doi:10.1016/j.jlap.2008.10.002`.

[SHR96] Sandeep K. Shukla, Harry B. Hunt III, and Daniel J. Rosenkrantz. HORNSAT, model checking, verification and games: extended abstract. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer Aided Verification: CAV*, volume 1102 of *LNCS*, pages 99–110. Springer, Berlin, 1996. `doi:10.1007/3-540-61474-5_61`.

[SMJ16]   Rick Smetsers, Joshua Moerman, and David N. Jansen. Minimal separating sequences for all pairs of states. In Adrian-Horia Dediu, Jan Janoušek, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and automata theory and applications: LATA*, volume 9618 of *LNCS*, pages 181–193. Springer, [s.l.], 2016. `doi:10.1007/978-3-319-30000-9_14`.

[Sti96]   Colin Stirling. Modal and temporal logics for processes. In Faron Moller and Graham Birtwistle, editors, *Logics for Concurrency: structure versus automata*, volume 1043 of *LNCS*, pages 149–237. Springer, Berlin, 1996. `doi:10.1007/3-540-60915-6_5`.

[Trz21]   Mariusz Trzeciakiewicz. Linear-time–branching-time spectroscopy as an educational web browser game. Bachelor's thesis, Technische Universität, Berlin, 2021. URL: `https://github.com/Marii19/the-spectroscopy-invaders`.

[vG90]   R. J. van Glabbeek. The linear time–branching time spectrum: extended abstract. In J. C. M. Baeten and J. W. Klop, editors, *CONCUR'90*, volume 458 of *LNCS*, pages 278–297. Springer, Berlin, 1990. `doi:10.1007/BFb0039066`.

[vG93]   R. J. van Glabbeek. The linear time–branching time spectrum II: The semantics of sequential systems with silent moves; extended abstract. In Eike Best, editor, *CONCUR'93*, volume 715 of *LNCS*, pages 66–81. Springer, Berlin, 1993. `doi:10.1007/3-540-57208-2_6`.

[vG01]   R. J. van Glabbeek. The linear time–branching time spectrum I: The semantics of concrete, sequential processes. In J. A. Bergstra, A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*, chapter 1, pages 3–99. Elsevier, Amsterdam, 2001. `doi:10.1016/B978-044482830-9/50019-9`.

[WMS21]   Thorsten Wißmann, Stefan Milius, and Lutz Schröder. Explaining behavioural inequivalence generically in quasilinear time. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory: CONCUR*, volume 203 of *LIPIcs*, pages 32:1–32:18. Schloss Dagstuhl: Leibniz-Zentrum für Informatik, Saarbrücken, Germany, 2021. `doi:10.4230/LIPICS.CONCUR.2021.32`.