TECHNISCHE UNIVERSITÄT BERLIN
FAKULTÄT IV – ELEKTROTECHNIK UND INFORMATIK
MODELLE UND THEORIE VERTEILTER SYSTEME

MASTER'S THESIS

Computing Coupled Similarity

BENJAMIN BISPING
ID: 321187
`info@bbisping.de`

Supervised by
Prof. Dr. Uwe NESTMANN
Prof. Dr. Holger HERMANNS

Berlin
April 2018

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Benjamin Bisping
Berlin, 18. April 2018

Digital Version

This master's thesis is accompanied by digital content, which is available from

`https://coupledsim.bbisping.de`.

The Scala implementations and Isabelle/HOL formalizations referenced in this thesis can be found there. I pledge to keep the online version live for at least ten years, that is, at least until December 2028. Moreover, the printed version contains a DVD with the digital content.

Acknowledgements

I'd like to thank everybody who helped me develop and finish this thesis (in order of appearance).

- Uwe Nestmann and Kirstin Peters for introducing me to coupled simulations.
- My colleagues at Modelle und Theorie Verteilter Systeme, especially my awesome office mate David Karcher, master of super-exponential normal forms, who did a lot to support me in my research.
- Holger Hermanns and the Dependable Systems and Software group, especially Felix Freiberger, for their hospitality during my visit to Saarbrücken; Ralf Wimmer for providing me with material on Sigref.
- The people who attended my talk at D-CON 2018 and discussed games and simulations with me, especially Stephan Mennicke.
- My lovely writing mates Hai-Hsin and Baum, who currently are in the midst of writing their master's thesis and dissertation. I wish them lots of success!

Abstract

Coupled similarity is a notion of equivalence for systems with internal actions. It has its applications in contexts where internal choices must transparently be distributed in time or space, for example, in process calculi encodings or in action refinements. To the author's knowledge, no tractable algorithms for the computation of the coupled simulation preorder and coupled similarity have been presented up to now. Accordingly, there is no tool support for coupled similarity.

In this master's thesis, we present three algorithms to compute coupled similarity: An algorithm based on reduction to strong bisimilarity, a coinductive fixed-point algorithm, and a game-theoretic algorithm. The game-theoretic algorithm runs in $\mathcal{O}(|\overset{\hat{\cdot}}{\Rightarrow}|\,|S|)$ space and time where $\overset{\hat{\cdot}}{\Rightarrow}$ is the weak transition relation and S the state space of the input transition system. This matches the time complexity of the best known simulation preorder algorithms. Additionally, we survey definitions of coupled simulation from the literature and show that deciding the coupled simulation preorder is at least as hard as deciding the weak simulation preorder.

The thesis is accompanied by an Isabelle/HOL formalization of key results, as well as by a Scala.js web tool implementation of the discussed algorithms, and a parallelized version of the game algorithm using the Apache Flink framework.

Zusammenfassung

Gekoppelte Similarität ist ein Gleichheitsbegriff auf Transitionssystemen mit internen Ereignissen. Ihre Anwendungen liegen dort, wo interne Entscheidungen in Raum oder Zeit zu verteilen sind, ohne dass das im Modell als Unterschied gilt. Beispiele dafür sind Prozess-Kalkül-Übersetzungen und Aktionsverfeinerungen. Bisher sind keine effizienten Algorithmen für gekoppelte Similarität und ihre Quasiordnung vorgestellt worden. Entsprechend gibt es auch keine Unterstützung durch Tools.

In dieser Master-Arbeit entwickeln wir drei Algorithmen zur Berechnung von gekoppelten Similaritätsrelationen: Eine Reduktion auf starke Bisimilarität, einen koinduktiven Fixpunkt-Algortihmus und einen spieltheoretischen Algorithmus. Letzterer läuft in Zeit- und Speicher-Komplexität von $\mathcal{O}(|\overset{\hat{\cdot}}{\Rightarrow}|\,|S|)$, wobei $\overset{\hat{\cdot}}{\Rightarrow}$ die schwache Schrittrelation bezeichnet und S den Zustandsraum des eingegebenen Transitionssystems. Das entspricht in der Zeitkomplexität den besten bekannten Algorithmen der schwachen Simulations-Quasiordnung. Des Weiteren ordnen wir Definitionen von gekoppelter Similarität aus der Literatur und zeigen, dass das Entscheidungsproblem der gekoppelten Simulations-Quasiordnung mindestens so komplex ist wie das der schwachen Simulations-Quasiordnung.

Eine Isabelle/HOL-Formalisierung der Kernergebnisse, ein die Algorithmen in Scala.js implementierendes Webtool und eine parallelisierte Version des spieltheoretischen Algorithmus für die Apache-Flink-Plattform begleiten die Arbeit.

Contents

Chapter 1

Introduction

In a computer system, a lot of things are going on that we cannot observe. Systems with the same observable behavior in one context might expose differing behavior in other contexts. This can be because seemingly identical visible actions actually lead to different internal system states (*non-determinism*), or because the system commits to a course of events internally (*internal choice*).

When is it justified to consider two systems—or system models—to be *equivalent*? Computer science has developed a multitude of notions of equivalence for such systems. One interesting notion is *coupled similarity*. It adds a weak form of symmetry (*coupling*) to weak similarity.

Coupled similarity hits a sweet spot within the *linear-time branching-time spectrum* [vG93]. At that spot, one can encode between brands of process calculi [NP00, HWPN15], name a branching-time semantics for Communicating Sequential Processes [vG17], distribute synchronizations [PS92], and refine atomic actions [Ren00, DW03].

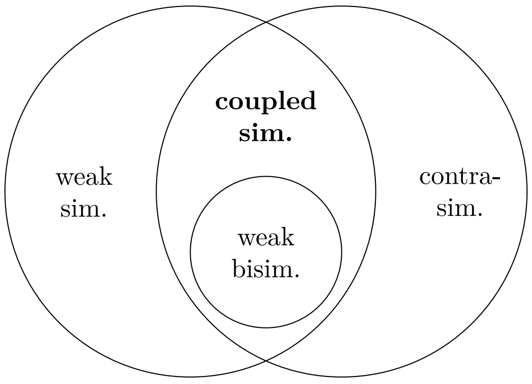


Figure 1.1: Notions of equivalence for systems with internal actions.

Figure 1.1 gives a first intuition where we are in the spectrum. Coupled similarity does distinguish systems by their branching behavior, but it does not see the precise structure of internal transitions and choices. It is *stronger than weak similarity* as it notices deadlocks but *weaker than weak bisimilarity* as the latter captures internal choice structure. It also is *finer than contrasimilarity* as it can tell non-determinism introduced by internal choices from non-determinism introduced by branching visible steps. The set of coupled simulations is a proper superset of the set of weak bisimulations and precisely the intersection of weak simulations and contrasimulations.

1.1 Computing Coupled Similarity

This thesis is not (only) about how neat coupled similarity is but primarily about how to decide which states of a finite transition system are coupled-similar.

Computing coupled similarity is an interesting challenge because the aforementioned sweet spot comes at a price: Coupled similarity is virtually the finest notion of equivalence for systems with internal actions where the equivalence and the corresponding behavioral preorder differ decisively. This makes the computation significantly more complicated than for finer relations such as weak bisimilarity.

While there are efficient and optimized algorithms for weak bisimilarity [BGRR16], up to now, no algorithms for coupled similarity, being just slightly coarser, have been developed. Parrow and Sjödin [PS94] briefly addressed the question of computing coupled similarity as possible future work. Apparently, the future is now.

1.2 This Thesis

After an introduction to the field in Chapter 2, we make the following contributions:

- We *survey pre-existing definitions of coupled simulation* and prove their coincidences (Section 3.1).
- We *characterize the coupled simulation preorder by a game* (Section 3.4).
- We *reduce weak simulation to coupled simulation*, thereby showing that deciding coupled simulation is comparably hard—presumably cubic (Section 3.5).
- Picking up the thread of [PS94], we develop an exponential-time *coupled similarity algorithm based on their normalization procedure* (Section 4.2).
- We present a more straight-forward polynomial-time *coupled simulation fixed-point algorithm* and prove its correctness (Section 4.3).
- We propose a *game-theoretic coupled simulation algorithm* (Section 4.4), which runs in cubic time and can be improved further (Section 4.5).
- We *implement the game algorithm for massively parallel computation using Apache Flink* and benchmark its performance (Chapter 5).

To unfold these contributions, the core chapters (Chapter 2, 3, 4) each follow the *trinity of characterizations* for notions of behavioral equivalence, consisting of relation properties, axiomatizations, and games. Figure 1.2 explicates this matrix structure. Chapter 2 introduces transition system models and the aspects of the trinity. Chapter 3 applies the trinity to characterize coupled similarity. Chapter 4 turns the characterizations into algorithms. The trinity roughly has the following sides:

Relational and coinductive definitions of equivalences imply greatest-fixed-point algorithms (Sections 2.2, 3.1, and 4.3).

Axiomatizations by equational laws entail reductions to strong bisimulation by normalization (Sections 2.3, 3.3, and 4.2).

Game characterizations map the equivalence problem to the computation of game winning regions (Sections 2.4, 3.4, and 4.4).

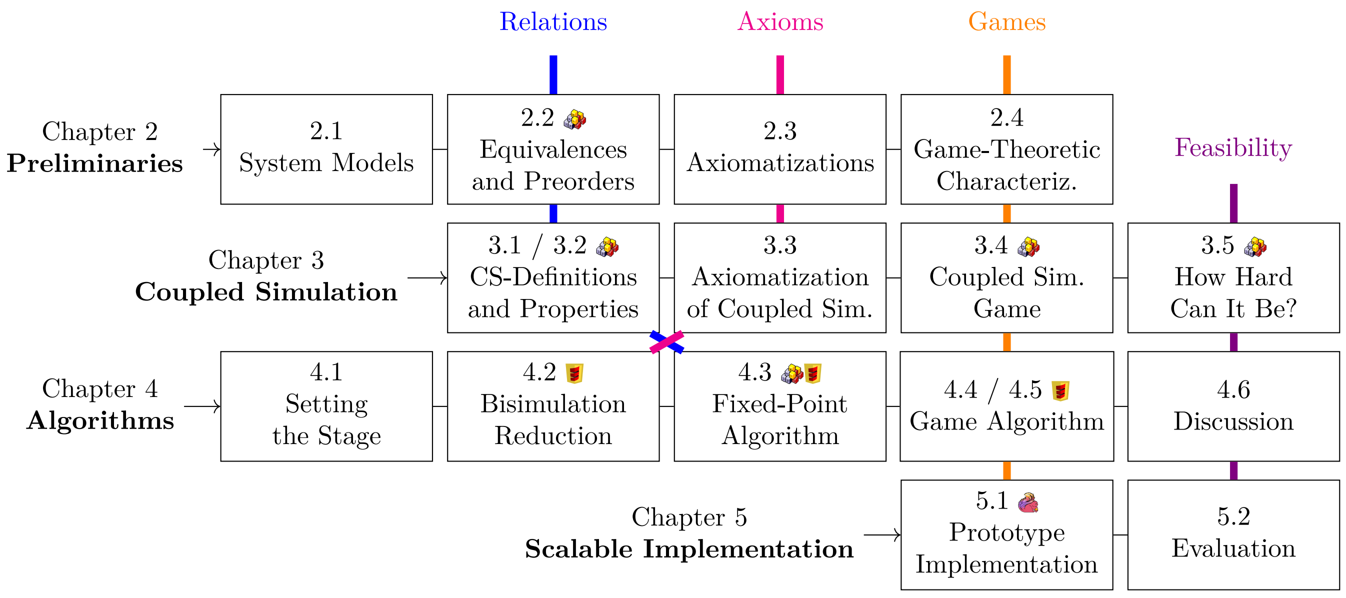


Figure 1.2: Structure of the thesis (with icons for accompanying artifacts).

The trinity actually is more of a quadrity in that there also are modal-logical characterizations, which lead to relational coarsest partition algorithms. This aspect has been factored out of the scope of this thesis. We briefly address it in the conclusion.

1.3 Accompanying Artifacts

This document is accompanied by machine checkable proofs in Isabelle/HOL, a web tool for experiments with the algorithms implemented in Scala.js, and a parallel implementation of the game-based main algorithm employing Apache Flink and Scala.

Isabelle/HOL (icon:) is an interactive proof assistant. It can be used to formalize mathematical definitions and proofs in a machine-checkable way. The proofs tend to be a little more verbose than corresponding pen-and-paper proofs but achieve a higher level of maintainability and confidence than the latter. Our formalization can be found on `https://coupledsim.bbisping.de/isabelle/`. To keep the presentation compact, most lemmas in this thesis just go with a hint or proof sketch and a link to the full-blown Isabelle/HOL proof.

Scala.js () is a programming language that builds a bridge between the functional world of mathematics and the object-oriented world of application development. It can be compiled to JavaScript and then run in any modern browser. The source code of our implementation is available from `https://coupledsim.bbisping.de/code/`. Most of it is written in functional Scala.js. For a quick start, please refer to `https://coupledsim.bbisping.de/code/README.md`. The compiled tool itself runs on `https://coupledsim.bbisping.de/` and contains all relevant example transition systems and presented algorithms.

Apache Flink/Scala () is a framework for parallelized big data applications. Our implementation of the coupled simulation game algorithm using Apache Flink can be found at `https://coupledsim.bbisping.de/code/flink/`.

Chapter 2

Preliminaries

As it is the custom, this chapter introduces some standard notions and notations[1] for equivalences on labeled transition systems with internal actions. We will look at examples of how notions of equivalence close to coupled similarity are characterized in terms of relations, axioms, and games.

The focus of this chapter lies on properties that are important for the characterization of coupled similarity later on. A more detailed introduction to the field can be found in [San12]. Readers familiar with the field can fast-forward to the next chapter.

2.1 System Models

Labeled transition systems and process calculi are two of the core concepts in describing the behavior of distributed interactive systems—and what in the world is not a distributed interactive system in some way?

2.1.1 Transition Systems

Labeled transition systems capture a discrete world view, where there is a current state and a branching structure of possible state changes ("transitions") to future states. They, for instance, can be used as semantical models for flowcharts, computer programs, and board games.

Definition 2.1 (Labeled transition system)**.** A *labeled transition system* (LTS) is a tuple $\mathcal{S} = (S, \Sigma_\tau, \rightarrow)$ where

- S is a set of *states*,
- Σ_τ is a set of *actions* containing a special *internal action* $\tau \in \Sigma_\tau$, and
- $\rightarrow \subseteq S \times \Sigma_\tau \times S$ is the *transition relation.*[2]

We write $\Sigma := \Sigma_\tau \setminus \{\tau\}$ for the set of *visible actions* and $p \xrightarrow{\alpha} p'$ for $(p, \alpha, p') \in \rightarrow$. We use p, q to range over states, α, β to range over actions, and a, b to range over visible actions.

[1]Later on, the nomenclature on page 58 can be used to disentangle notation. Other hints for notation: Formulas are implicitly parenthesized from the right. We use $\cdot$ to abstract functions and relations: $f(\cdot)$ stands for $\lambda x.\, f(x)$, which is identified with the relation $\{(x, y) \mid y = f(x)\}$. In infix notation, we omit the outer dots, so $\xrightarrow{\cdot} = \cdot \xrightarrow{\cdot} \cdot$.

[2] locale `Transition_Systems.lts`

Definition 2.2 (Weak transition relation)**.** The *weak transition relation* $\overset{\hat{\cdot}}{\Rightarrow}$ is defined as the reflexive transitive closure of internal steps $\overset{\hat{\tau}}{\Rightarrow} := \xrightarrow{\tau}^{*}$ combined with $\overset{\hat{a}}{\Rightarrow} := \overset{a}{\Rightarrow}$ where $\overset{\alpha}{\Rightarrow} := \overset{\hat{\tau}}{\Rightarrow} \xrightarrow{\alpha} \overset{\hat{\tau}}{\Rightarrow}$.[3] We write $p \overset{\vec{a}}{\Rightarrow} p'$, where $\vec{a} \in \Sigma^*$, iff there is a sequence of steps $p = p_0 \overset{a_0}{\Rightarrow} p_1 \overset{a_1}{\Rightarrow} \cdots \overset{a_{n-1}}{\Rightarrow} p_n = p'$ or, for the empty word $\vec{a} = \langle\rangle$, iff $p \overset{\hat{\tau}}{\Rightarrow} q$.

So, the difference between $\overset{\tau}{\Rightarrow}$ and $\overset{\hat{\tau}}{\Rightarrow}$ is that the first needs at least one $\xrightarrow{\tau}$-step whereas the latter is reflexive. As a shorthand for $\overset{\hat{\tau}}{\Rightarrow}$, we also write just $\Rightarrow$.

We call an $\overset{\alpha}{\Rightarrow}$-step "weak" whereas an $\xrightarrow{\alpha}$-step is referred to as "strong." In general, $p \xrightarrow{\alpha} p'$ entails $p \overset{\alpha}{\Rightarrow} p'$, and $p \overset{\alpha}{\Rightarrow} p'$ entails $p \overset{\hat{\alpha}}{\Rightarrow} p'$.

A visible action $a \in \Sigma$ is said to be *weakly enabled* in p iff there is p' such that $p \overset{a}{\Rightarrow} p'$.

Definition 2.3 (Stability and divergence)**.** A state p is called *stable* iff it has no τ-transitions, $p \not\xrightarrow{\tau}$. Stable states are *maximal* with respect to $\Rightarrow$. For a stable state p, weak and strong steps coincide $p \overset{\alpha}{\Rightarrow} \cdot = p \xrightarrow{\alpha} \cdot$.

A state p is called *divergent* iff it is possible to perform an infinite sequence of τ-transitions beginning in this state, $p \xrightarrow{\tau}^{\omega}$. For a divergent state p', $p \Rightarrow p'$ implies that p also is divergent, that is: divergence is *downward closed* under $\Rightarrow$.

2.1.2 Process Calculi

Transition systems arise naturally as an operational semantics for programs written in some programming language. Concurrency theory aims to provide abstract ways of understanding distributed computation. To this end, the field has developed a number of *process calculi*—formal languages the dynamics of concurrent systems can be expressed in. These languages are *algebraic* in the sense that they are built around term equalities that enable calculations on the terms of the languages.

This thesis is not about the nuances of these process calculi. Nevertheless, it is worthwhile to understand that the relevance of the different notions of equivalence we shall discuss, including coupled similarity, is linked to these nuances.

For our purposes, it is enough to introduce the following fragment of MILNER'S *Calculus of Communicating Systems* (CCS) [Mil89] without value passing:

Definition 2.4 (CCS)**.** The following grammar gives the language of *Calculus of Communicating Systems* expressions $\mathsf{CCS}[\mathcal{I}, \Sigma_{in}]$ for a set of process names $\mathcal{I}$ and a set of visible input actions Σ_{in}, which induce corresponding output actions $\Sigma_{out} := \{\bar{a} \mid a \in \Sigma_{in}\}$ and $\Sigma := \Sigma_{in} \cup \Sigma_{out}$:

$$\begin{gathered} P, Q \quad ::= \quad K \quad | \quad \alpha.P \quad | \quad P + Q \quad | \quad P \mid Q \quad | \quad P \setminus L, \\ \text{where } K \in \mathcal{I},\ \alpha \in \Sigma_\tau, \text{ and } L \subseteq \Sigma. \end{gathered}$$

Under a process valuation $\mathcal{D} \colon \mathcal{I} \to \mathsf{CCS}[\mathcal{I}, \Sigma_{in}]$, the operational semantics of $\mathsf{CCS}[\mathcal{I}, \Sigma_{in}]$ is the transition system $\mathcal{S}_{\mathcal{D}, \mathsf{CCS}[\mathcal{I}, \Sigma_{in}]} = (\mathsf{CCS}[\mathcal{I}, \Sigma_{in}], \Sigma_\tau, \to)$ on process expressions, where $\xrightarrow{\alpha}$ for $\alpha \in \Sigma_\tau$ is defined by the following rules. (We define $\bar{\bar{a}} = a$.)

Recursion $K \xrightarrow{\alpha} P'$ if $P \xrightarrow{\alpha} P'$ and $\mathcal{D}(K) = P$.

A process name occurrence K behaves like the referenced process.

[3] locale `Weak_Transition_Systems.lts_tau`

Action prefix $\alpha.P \xrightarrow{\alpha} P$.

An action prefix $\alpha.P$ can do an α-step and then behave like P.

Choice $P + Q \xrightarrow{\alpha} P'$ if $P \xrightarrow{\alpha} P'$ or $Q \xrightarrow{\alpha} P'$.

A choice is resolved towards the first process that makes a step.

Parallel step $P \mid Q \xrightarrow{\alpha} P' \mid Q$ if $P \xrightarrow{\alpha} P'$; and $P \mid Q \xrightarrow{\alpha} P \mid Q'$ if $Q \xrightarrow{\alpha} Q'$.

Processes in a parallel composition can perform steps with interleaving concurrency.

Communication $P \mid Q \xrightarrow{\tau} P' \mid Q'$ if $P \xrightarrow{a} P'$ and $Q \xrightarrow{\bar{a}} Q'$.

Parallel processes can advance with a synchronized step on a shared action that is output by one of them and input to the other process. The message is consumed and becomes internal behavior τ.

Restriction $P \setminus L \xrightarrow{\alpha} P' \setminus L$ if $P \xrightarrow{\alpha} P'$ and $\alpha, \bar{\alpha} \notin L$.

A restricted process cannot communicate certain events L to the environment.

We write $K \overset{\text{def}}{=} P$ to set $\mathcal{D}(K)$ to P. We assume that there always is a stuck process $\mathbf{0}$, which can be derived by letting $\mathbf{0} \overset{\text{def}}{=} (a.\mathbf{0}) \setminus \{a\}$ where a is some $a \in \Sigma$. We use just α as an abbreviation for the process term $\alpha.\mathbf{0}$.

CCS is extremely expressive due to its choice operator. One can easily encode arbitrary finitely branching transition systems $\mathcal{S} = (S, \Sigma_\tau, \rightarrow)$ with the valuation $\mathcal{D}_\mathcal{S} \colon S \rightarrow \mathsf{CCS}[S, \Sigma]$:

$$\mathcal{D}_\mathcal{S}(s) := \begin{cases} \alpha_1.s'_1 + \ldots + \alpha_n.s'_n & \text{for } \{(\alpha_i, s'_i) \mid 1 \leq i \leq n\} = s \xrightarrow{\cdot} \cdot \\ \mathbf{0} & \text{if } s \not\rightarrow \end{cases}$$

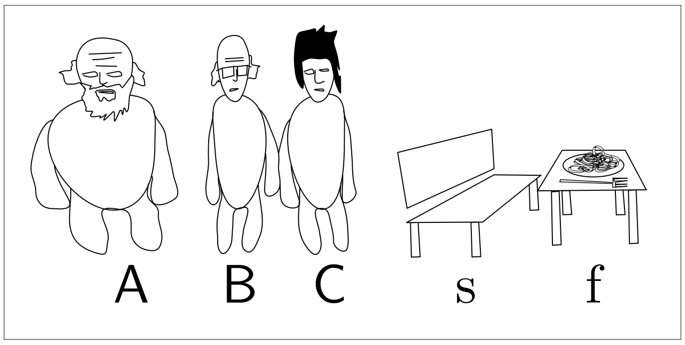
If $\mathcal{S}$ is finite, the $\mathcal{D}_\mathcal{S}$-structure is finite as well.

Now, it seems natural to wonder when to consider two processes as behaviorally equivalent. For example, $\mathbf{0}$ and $\mathbf{0} \mid \mathbf{0}$ obviously have the same set of outgoing transitions (the empty set) but are different states in the transition system $\mathcal{S}_{\mathcal{D},\mathsf{CCS}[\mathcal{I},\Sigma_{in}]}$. To discuss equivalences, let us introduce a more enlightening example.

2.1.3 Running Example: Exclusively Dining Philosophers

Let us see how transition systems and CCS can be used to model a concurrent situation. More abstract variants of the following example have been used in the literature to motivate coupled similarity [NP00, PS92, San12].

Example 2.5 (Gradually committing philosophers). Three philosophers A, B and C want to eat pasta. To do so, they must first sit down on a bench s and grab a fork f.

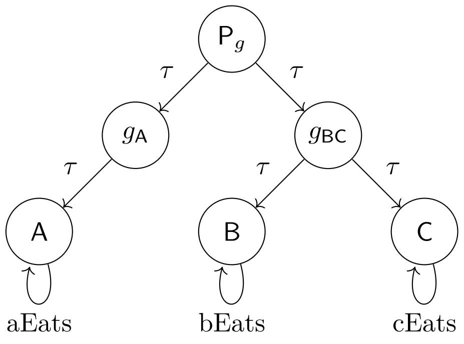


Unfortunately, only either A alone or the thinner B and C can fit on the bench simultaneously and there is just one fork. (The philosophers have the skill of eating with one fork.) From the outside, we are only interested in the fact who of them eats. So we consider the whole bench-and-fork business internal to the system.

The following CCS structure models the situation. The resources correspond to output actions (which can only be consumed once) and the obtaining of the resources corresponds to matching input actions.

$$\begin{aligned}
\mathsf{P}_g &\overset{\text{def}}{=} \big(\bar{\mathrm{s}} \mid \bar{\mathrm{f}} \mid \mathrm{s.f.}\mathsf{A} \mid \mathrm{s.}(\mathrm{f.}\mathsf{B} \mid \mathrm{f.}\mathsf{C})\big) \setminus \{\mathrm{s}, \mathrm{f}\}\\
\mathsf{A} &\overset{\text{def}}{=} \mathrm{aEats.}\mathsf{A}\\
\mathsf{B} &\overset{\text{def}}{=} \mathrm{bEats.}\mathsf{B}\\
\mathsf{C} &\overset{\text{def}}{=} \mathrm{cEats.}\mathsf{C}
\end{aligned}$$

This structure has the following transition system with initial state P_g as its semantics. Notice that the internal communication concerning the resource allocation turns into internal τ-actions, which in P_g, g_{A}, and g_{BC} *gradually decide* who is going to eat the pasta.

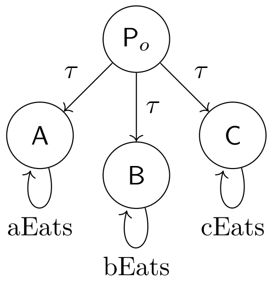


One might now be inclined to ponder that exactly one of the philosophers will get both resources and that we thus could model the situation just as well in the following way.

Example 2.6 (One-step non-determinism philosophers)**.** If we merge s and f into a single resource sf, the model becomes:

$$\mathsf{P}_o \overset{\text{def}}{=} \big(\overline{\mathrm{sf}} \quad\mid\quad \mathrm{sf.}\mathsf{A} \mid \mathrm{sf.}\mathsf{B} \mid \mathrm{sf.}\mathsf{C}\big) \setminus \{\mathrm{sf}\}$$

The corresponding transition system concentrates the internal choice to the starting node P_o.



We refer to the philosopher system containing the components of P_g and P_o (sharing $\mathsf{A}, \mathsf{B}, \mathsf{C}$) as $\mathcal{S}_\mathsf{P}$. It can also be inspected at `https://coupledsim.bbisping.de/#phil` or in Figure 2.2.

So, are P_g and P_o behaviorally equivalent? This depends on the notion of equivalence we employ. They obviously can reach different transitions. A notion of equivalence then tells us which aspects of the state changes really matter—which are "observable."

Some notions will consider P_g and P_o equivalent, some will reject the equivalence. Sometimes this is a philosophical question. But, more often than not, encodings between formalisms, and reasonable refinements of specifications only become possible if the notion of equivalence is coarse enough to accept gradual and one-step internal commitment as equivalent. Spoiler: Coupled similarity is one of the finest branching-time equivalences considering P_g and P_o equivalent.

2.2 Equivalences and Preorders

Let us introduce weak simulation, weak bisimulation, and contrasimulation. They are the most relevant notions of equivalence related to coupled similarity.

The framework for arbitrary $\mathcal{S} = (S, \Sigma_\tau, \rightarrow)$ is the following: We define a property "*X-simulation*." If a relation on the state space $\mathcal{R} \subseteq S \times S$ fulfills the property, we call it an X-simulation on $\mathcal{S}$. The property induces a preorder $\sqsubseteq_X^\mathcal{S}$, the "$X$-simulation *preorder*," with $p \sqsubseteq_X^\mathcal{S} q$ iff there is an X-simulation $\mathcal{R}$ on $\mathcal{S}$ with $(p, q) \in \mathcal{R}$. The preorder then induces an *equivalence* $\equiv_X^\mathcal{S} = \sqsubseteq_X^\mathcal{S} \cap \sqsubseteq_X^{\mathcal{S}\,-1}$, the "*$X$-similarity*."[4]

There are hundreds of notions of equivalence in the linear-time branching-time spectrum [vG90, vG93] that can be characterized in this way. In this section, we use the framework for the main neighbors of coupled simulation in the spectrum.

2.2.1 Weak Similarity and Weak Simulation Preorder

Weak simulation is a variant of simulation illustrated by Figure 2.1(A). A *simulation* maps each state to other states that can do at least as much as it and therefore "simulate" it. "Simulation" means for a pair of states p, q that, for every α-step that can be performed in the first state p, the second *simulating* state q can *answer* with a matching step leading to a state that simulates the target of the first step. The answer part is rendered red in the figure. For *weak* simulation, the answer may use the additional freedom of weak steps $\overset{\hat{\cdot}}{\Rightarrow}$ as opposed to just strong steps $\overset{\cdot}{\rightarrow}$. Intuitively, a state p is simulated by q if p cannot expose more behavior than q. More formally:

Definition 2.7 (Weak simulation)**.** A *weak simulation* is a relation $\mathcal{R} \subseteq S \times S$ such that, for all $(p, q) \in \mathcal{R}$, $p \xrightarrow{\alpha} p'$ implies that there is a q' such that $q \overset{\hat{\alpha}}{\Rightarrow} q'$ and $(p', q') \in \mathcal{R}$.

The *weak simulation preorder* relates two states, $p \sqsubseteq_{WS} q$, iff there is a weak simulation $\mathcal{R}$ such that $(p, q) \in \mathcal{R}$. Two states are *weakly similar,* $p \equiv_{WS} q$, iff $p \sqsubseteq_{WS} q$ and $q \sqsubseteq_{WS} p$. The relation $\equiv_{WS}$ is called weak similarity.[5]

Remark 2.8. Another, trivially equivalent, way of expressing the definition would be: $\mathcal{R}$ is a weak simulation iff $\mathcal{R}^{-1} \xrightarrow{\alpha} \subseteq \overset{\hat{\alpha}}{\Rightarrow} \mathcal{R}^{-1}$ for all $\alpha \in \Sigma_\tau$.

[4]In the following, we omit the system $\mathcal{S}$ where it is clear from the context. Also, $\overset{\cdot}{\Rightarrow}$, Σ, etc. are implicitly instantiated by fixed $\mathcal{S}$. Sometimes we just write "X-simulation" to refer to the preorder.

[5] definition `Weak_Relations.weak_simulation`

$$\begin{tikzcd}
p \arrow[r, dashed, bend left, "\mathcal{R}"] \arrow[d, "\alpha"] \arrow[dr, phantom, "\Searrow"] & q \arrow[d, Rightarrow, "\hat{\alpha}"] \\
p' \arrow[r, dashed, bend left, "\mathcal{R}"'] & q'
\end{tikzcd}$$

A: Weak simulation

$$\begin{tikzcd}
p \arrow[r, dashed, bend left, "\mathcal{R}"] \arrow[d, Rightarrow, "\hat{\alpha}"] \arrow[dr, phantom, "\Swarrow"] & q \arrow[d, "\alpha"] \\
p' \arrow[r, dashed, bend left, "\mathcal{R}"'] & q'
\end{tikzcd}$$

B: Weak bisimulation (2)

$$\begin{tikzcd}
p \arrow[r, dashed, bend left, "\mathcal{R}"] \arrow[d, Rightarrow, "\vec{a}"] \arrow[dr, phantom, "\Searrow"] & q \arrow[d, Rightarrow, "\vec{a}"] \\
p' & q' \arrow[l, dashed, bend left, "\mathcal{R}"]
\end{tikzcd}$$

C: Contrasimulation

$$\begin{tikzcd}
p \arrow[r, dashed, bend left, "\mathcal{R}"] \arrow[dr, phantom, "\Searrow"] & q \arrow[d, Rightarrow] \\
 & q' \arrow[ul, dashed, bend left, "\mathcal{R}"]
\end{tikzcd}$$

D: Coupling

Figure 2.1: Illustration of weak relations on transition systems.

Example 2.9. Some weak simulations on $\mathcal{S}_{\mathsf{P}}$:

- $\mathcal{R}_1 = \emptyset$ (the empty relation)
- $\mathcal{R}_2 = \Delta_S$ (the identity relation)
- $\mathcal{R}_3 = \Rightarrow^{-1}$ (the inverted τ-closure)
- $\mathcal{R}_4 = \{g_{\mathsf{A}}, \mathsf{A}\} \times \{g_{\mathsf{A}}, \mathsf{A}\}$
- $\mathcal{R}_5 = (\mathcal{R}_3 \cup \mathcal{R}_4 \cup \{(\mathsf{P}_g, \mathsf{P}_o), (\mathsf{P}_o, \mathsf{P}_g)\})^*$

The weak simulation $\mathcal{R}_5$ justifies $\mathsf{P}_g \equiv_{WS} \mathsf{P}_o$.

Actually, there is no weak simulation on $\mathcal{S}_{\mathsf{P}}$ that is greater than $\mathcal{R}_5$—that is: $\mathcal{R} \subseteq \mathcal{R}_5$ for all weak simulations $\mathcal{R}$. The greatest weak simulation always equals $\sqsubseteq_{WS}$. So, in this case, $\mathcal{R}_5 = \sqsubseteq_{WS}$.

Lemma 2.10. *The weak simulation preorder can be characterized,*

1. *by the union:*

$$\sqsubseteq_{WS} = \bigcup\{\mathcal{R} \mid \mathcal{R} \textit{ is a weak simulation}\}, \textit{ and}$$

2. *coinductively by the rule:*

$$\frac{\forall p', \alpha.\ p \xrightarrow{\alpha} p' \longrightarrow \exists q'.\ q \overset{\hat{\alpha}}{\Rightarrow} q' \ \wedge\ p' \sqsubseteq_{WS} q'}{p \sqsubseteq_{WS} q}.$$

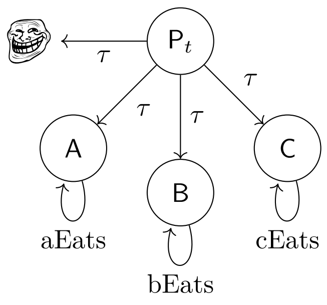
The coinductive characterization under number 2 means that $\sqsubseteq_{WS}$ is the greatest relation $\mathcal{R}$ where $(p, q) \in \mathcal{R}$ locally implies weak simulation for all p, q. For more material on coinduction, please refer to [San12].

Example 2.9 has shown that weak similarity is coarse enough for the gradually committing philosophers P_g and the one-step committing philosophers P_o to be equivalent. However, the weak simulation preorder is too coarse for many applications that work with some basic ideas of fairness, namely that all paths matter. It is blind to spurious deadlocks as illustrated by the following example.

Example 2.11 (Trolled philosophers)**.** Although it is more common for philosophers to struggle with trolley problems, let us imagine for the sake of the example that our philosophers, like other scientists [Sch15], are having a *troll problem.* A hidden troll (😏) enters the stage of Example 2.6 and might grab the resource sf.

$$\mathsf{P}_t \overset{\text{def}}{=} \left(\bar{\text{sf}} \mid \text{sf}.\mathsf{A} \mid \text{sf}.\mathsf{B} \mid \text{sf}.\mathsf{C} \mid \quad \text{sf}.\text{😏}\right) \setminus \{\text{sf}, \text{😏}\}$$

This yields a transition system with one new deadlock node (marked by 😏).



Consider the components of P_o and P_t in the same system. The troll state cannot perform any actions and thus is weakly simulated by all other states. So, the $\mathsf{P}_t \xrightarrow{\tau}$ 😏-transition can be simulated by any $\mathsf{P}_o \xrightarrow{\tau}$-step. As this transition is the only difference between P_o and P_t, weak similarity considers them equivalent $\mathsf{P}_o \equiv_{WS} \mathsf{P}_t$.

For most system models, it indeed is relevant whether a hidden process can block a choice or not. It matters whether the philosophers starve. Weak similarity is too coarse to make this differentiation. So, let us look at a finer notion of equivalence.

2.2.2 Weak Bisimilarity

Weak bisimilarity probably is the best-known notion of equivalence for transition systems with internal steps. It adds a converse requirement to the definition of weak similarity, which is depicted in Figure 2.1(B).

Definition 2.12 (Weak bisimulation)**.** A *weak bisimulation* is a relation $\mathcal{R} \subseteq S \times S$ such that, for all $(p, q) \in \mathcal{R}$,

- $p \xrightarrow{\alpha} p'$ implies that there is a q' such that $q \overset{\hat{\alpha}}{\Rightarrow} q'$ and $(p', q') \in \mathcal{R}$, and
- $q \xrightarrow{\alpha} q'$ implies that there is a p' such that $p \overset{\hat{\alpha}}{\Rightarrow} p'$ and $(p', q') \in \mathcal{R}$.

Two states are called *weakly bisimilar,* $p \equiv_{WB} q$, iff there is a weak bisimulation $\mathcal{R}$ such that $(p, q) \in \mathcal{R}$. (For bisimulation, equivalence $\equiv_{WB}$ and preorder $\sqsubseteq_{WB}$ coincide.)

Definition 2.12 says in other words that $\mathcal{R}$ is a (weak) bisimulation iff $\mathcal{R}$ and $\mathcal{R}^{-1}$ are (weak) simulations.[6] Weak bisimilarity is the greatest symmetric weak simulation.

[6] lemma `Weak_Relations.weak_bisim_weak_sim`

Example (2.5, 2.6, 2.11 continued)**.** Returning to our running example, we see that weak bisimilarity notices the troll so that $\mathsf{P}_o \not\equiv_{WB} \mathsf{P}_t$. This is because the troll state 🧌 cannot simulate any other state, and thus cannot be covered by any bisimulation containing $(\mathsf{P}_o, \mathsf{P}_t)$ answering to $\mathsf{P}_t \xrightarrow{\tau}$ 🧌.

But weak bisimilarity also notices the differing internal branching structure of P_o and P_g. P_o can reach no state with the same set of enabled actions like g_{BC}. So, $\mathsf{P}_g \xrightarrow{\tau} g_{\mathsf{BC}}$ cannot be bisimulated by P_o, and thus $\mathsf{P}_o \not\equiv_{WB} \mathsf{P}_g$.

The second part of the example shows that weak bisimilarity is too strong for settings where we need to abstract away from the structure of gradual internal choices. The next section introduces a remedy: contrasimilarity.

2.2.3 Contrasimilarity and Contrasimulation Preorder

Contrasimulation is a slight modification of simulation, alternating the simulation direction. The answering process must have a successor state that is simulated by the target state of the *challenging* transitions, as shown in Figure 2.1(C). The direction of the red $\mathcal{R}$ is inverse to ordinary simulation. Moreover, challenge and answer may be words of actions $\vec{a}$ instead of single actions $\hat{\alpha}$.

Definition 2.13 (Contrasimulation)**.** A *contrasimulation* is a relation $\mathcal{R} \subseteq S \times S$ such that, for all $(p, q) \in \mathcal{R}$, $p \overset{\vec{a}}{\Rightarrow} p'$ implies that there is a q' such that $q \overset{\vec{a}}{\Rightarrow} q'$ and $(q', p') \in \mathcal{R}$.[7]

Pay attention to the swap of sides between p' and q' on the right-hand side of the definition if compared to weak simulation! $\sqsubseteq_C$ and $\equiv_C$ are implied as with weak similarity.

Remark 2.14. Definition 2.13 becomes *extensionally weaker* if one replaces the occurrences of $\overset{\vec{a}}{\Rightarrow}$ by $\overset{\hat{\alpha}}{\Rightarrow}$.[8] If one then replaces the "$p \overset{\hat{\alpha}}{\Rightarrow} p'$" on the left-hand side by "$p \xrightarrow{\alpha} p'$", the extension of the definition grows again.[9] This is noteworthy because many other weak simulation-like properties have equivalent definitions with weak and strong challenge transitions as well as with single step or transition path formulations.

Let us return to our running example.

Example (2.5, 2.6, 2.11 continued II)**.** The differing internal branching structure of P_o and P_g is invisible to contrasimilarity by the same witnesses as for weak similarity. A visualization of such a (contra-)simulation on $\mathcal{S}_{\mathsf{P}}$ can be seen in Figure 2.2. The asymmetric parts of the relation are drawn in red.

Just as weak bisimilarity, contrasimilarity notices the troll so that $\mathsf{P}_o \not\equiv_C \mathsf{P}_t$. Again, this is because the troll state 🧌 cannot simulate any other state, and, consequently, there cannot be any contrasimulation answer for $\mathsf{P}_t \xrightarrow{\tau}$ 🧌.

Contrasimulation shares some lemma schemes with simulation. For example, contrasimulation and symmetry together also imply for a relation to be a bisimulation.

Proposition 2.15. *If $\mathcal{R}$ is a contrasimulation and a symmetric relation, then $\mathcal{R}$ is a weak bisimulation.*[10]

[7] definition `Weak_Relations.contrasim`
[8] lemma `Weak_Relations.contrasim_step_weaker_than_seq`
[9] lemma `Weak_Relations.contrasim_challenge_strength_does_not_imply`
[10] lemma `Weak_Relations.symm_contrasim_implies_weak_bisim`

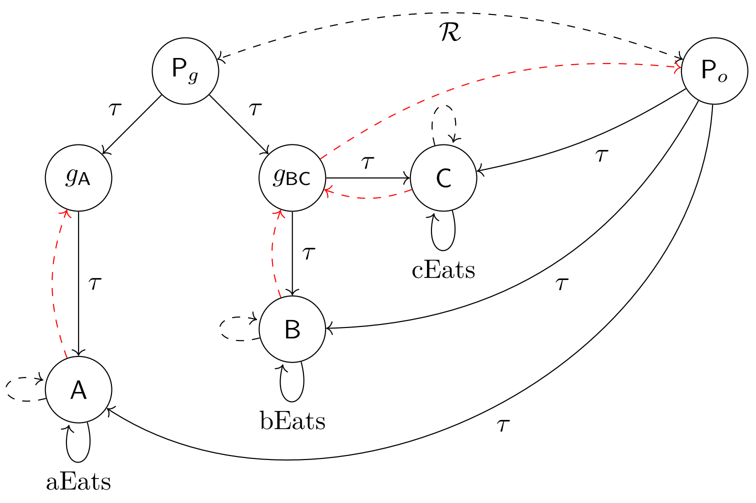


Figure 2.2: A non-maximal (contra/coupled/weak) simulation on the philosopher system $\mathcal{S}_{\mathsf{P}}$.

Although contrasimulation looks like a variation of simulation, it is in a certain way "closer" to the symmetry of the greatest bisimulation than weak simulation is. This is due to the *coupling property* of contrasimulation, illustrated in Figure 2.1(D):

Proposition 2.16 (Coupling)**.** *If $\mathcal{R}$ is a contrasimulation, then $(p, q) \in \mathcal{R}$ implies there is a q' such that $q \Rightarrow q'$ and $(q', p) \in \mathcal{R}$.*[11]

Coupling can be thought of as a relaxed form of symmetry—"weak symmetry" so to speak. For a relation to be symmetric, $\mathcal{R}^{-1} \subseteq \mathcal{R}$ must be true whereas coupling boils down to $\mathcal{R}^{-1} \subseteq {\Rightarrow}\mathcal{R}$. In particular, if coupling holds for a relation, it must be symmetric on the stable states, and τ-free systems only have stable states, so:

Corollary 2.17. *If $\xrightarrow{\cdot}$ contains no τ-steps, and $\mathcal{R}$ is a contrasimulation, then $\mathcal{R}$ is symmetric and thus a bisimulation.*[12]

Note that there is no analogous proposition for simulation. A *weak simulation* within a τ-free system is just a *strong simulation*, but not necessarily a bisimulation. This motivates our claim that contrasimulation is, in this respect, "closer to bisimulation" than weak simulation, even though the two are incomparable in the mathematical sense.

So, does this close our case? One indeed can argue that contrasimilarity has the desired properties concerning the transparency of distributed choices and the closeness to weak bisimilarity. However, it also entails equalities concerning drifting choice points that will seem strange to people who are not writing parallelizing compilers.

Example 2.18 (Religious philosophers)**.** For centuries, philosophers usually have been persons of faith. So, maybe they would say a grace before starting their meal (this example is narrowed down to only two philosophers):

$$\mathsf{P}_r \overset{\text{def}}{=} \left(\;\; \overline{\text{sf}} \;\; \mid \;\; \text{sf.grace.}\mathsf{A} \mid \text{sf.grace.}\mathsf{B} \;\; \right) \;\; \setminus \; \{\text{sf}\}$$

[11] lemma `Weak_Relations.contrasim_implies_coupling`
[12] lemma `Weak_Relations.taufree_contrasim_implies_weak_bisim`

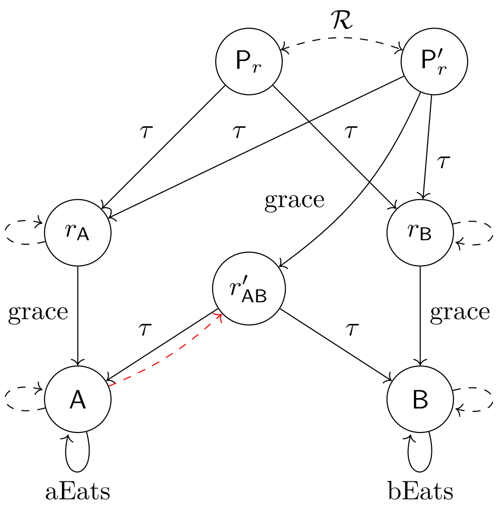


Figure 2.3: Contrasimulation for religious philosophers.

Interestingly, contrasimilarity does not care whether they say their grace before or after the acquirement of the resource, as can be seen by the transition system in Figure 2.3 with additional state P'_r (also at `https://coupledsim.bbisping.de/#phil-religious`). There, $\mathcal{R}$ is a contrasimulation, yielding $\mathsf{P}_r \equiv_C \mathsf{P}'_r$.

P'_r in Figure 2.3 can do quite the same as P_r—with the difference that it may also postpone the internal choice to r'_{AB}, that is, *after* the occurrence of grace. Note that $\mathcal{R}$ is no weak simulation. Indeed, there cannot be any mutually relating P_r and P'_r as there is no state that also weakly enables at least as many actions as r'_{AB} (that is, $\{\text{aEats}, \text{bEats}\}$) and consequently $\mathsf{P}'_r \overset{\text{grace}}{\rightarrow} r'_{\mathsf{AB}}$ cannot be simulated by P_r.

Example 2.18 already points to the fact that contrasimulation is partly blind to the structure of internal choice and non-determinism. $\sqsubseteq_C$ cannot tell the difference between a non-deterministic choice by visible steps and a deterministic visible step followed by an internal choice. In a nutshell: Non-determinism of visible steps can be postponed into internal choice! This is made explicit by the characteristic axiom of contrasimulation in Voorhoeve and Mauw's study [VM01] being

$$a.P + a.Q \;\equiv_C\; a.(\tau.P + \tau.Q).$$

Note however that contrasimilarity still is stronger than the well-known weak trace equivalence when it comes to drifting choices (see [Bel13, Figure 5]). There still are P, Q such that

$$a.P + a.Q \;\not\equiv_C\; a.(P + Q).$$

The choice drifting is crucial for compilers that try to analyze which code depends on what other computations [Bel13]. From the point of view of such a compiler, seeing that grace is bound to happen either way, it seems perfectly natural to allow it to occur independently of the internal choice.

But imagine a setting where the grace-event is actually a way to communicate to the environment that a group of processes has made up their mind on who will continue a course of action. There, we would have difficulties to accept P_r and P'_r to be congruent.

2.2.4 Equivalences, Differences, Intersections

Time to wrap up our observations.

- Weak bisimilarity maintains $\mathsf{P}_o \not\equiv_{WB} \mathsf{P}_g$. It is too strong to ignore the distribution of choices.
- Weak similarity considers $\mathsf{P}_o \equiv_{WS} \mathsf{P}_g$, but also $\mathsf{P}_o \equiv_{WS} \mathsf{P}_t$. It is so far away from weak bisimilarity that it is oblivious to spurious deadlocks. Still, it respects the temporal order of internal choices and visible actions, $\mathsf{P}_r \not\equiv_{WS} \mathsf{P}'_r$.
- Contrasimilarity also considers $\mathsf{P}_o \equiv_C \mathsf{P}_g$, and indeed maintains that $\mathsf{P}_o \not\equiv_C \mathsf{P}_t$. But, strangely, $\mathsf{P}_r \equiv_C \mathsf{P}'_r$. It partially allows internal choices and external behavior to be drifted. This breaks some intuitive temporal assumptions we might hold about our models. Moreover, contrasimilarity is unwieldy when it comes to proofs due to its definition in terms of transition sequences instead of single weak transitions.

So, if one looks for a notion of equivalence $\equiv_X$, where distributed choice is transparent ($\mathsf{P}_o \equiv_X \mathsf{P}_g$), while deadlocks *and* temporal drifts are noticed ($\mathsf{P}_o \not\equiv_X \mathsf{P}_t$ and $\mathsf{P}_r \not\equiv_X \mathsf{P}'_r$), one will have to turn to some equivalence in the intersection of $\equiv_{WS}$ and $\equiv_C$ while staying coarser than $\equiv_{WB}$. This is what we shall do in Chapter 3—coupled similarity is precisely the intersection of $\equiv_{WS}$ and $\equiv_C$.

If the application demands it, some *other refinements for notions of equivalence* can easily be added on top. Classical examples include the preservation of some state property such as stability or divergence. For instance, a behavioral preorder $\sqsubseteq_{\Delta X}$ is *divergence respecting* iff $p \sqsubseteq_{\Delta X} q$ implies that, if p diverges, then q does so too.

2.3 Axiomatizations

An important way of characterizing notions of equivalence in the context of process calculi is in terms of *complete axiomatizations.* These consist of sets of equational laws on process terms that equate two processes precisely when the corresponding notion of equivalence is meant to. Axiomatizations are of particular interest to us because the complete axiomatization of coupled similarity by Parrow and Sjödin [PS94] hints at the possibility of employing the axiomatization for a decision algorithm. This section introduces the concept of axiomatization for weak bisimilarity.

2.3.1 Soundness and Completeness

An axiomatization $\mathcal{A}$ is a set of *equational laws* that can be used to rewrite terms by substituting subterms accordingly. Expressions P and Q are considered equal if they can be transformed into one-another, for which we write $\mathcal{A} \vdash P = Q$. The correspondence between an axiomatization and an equivalence is understood in terms of soundness and completeness.

Definition 2.19. With respect to an equivalence $\equiv_X$, an axiomatization $\mathcal{A}$ is

sound iff $\mathcal{A} \vdash P = Q$ implies $P \equiv_X Q$, and

complete iff $P \equiv_X Q$ implies $\mathcal{A} \vdash P = Q$.

Soundness virtually always requires $\equiv_X$ to be a *congruence* (with respect to the operators occurring in $\mathcal{A}$) and the $\mathcal{A}$-laws to be valid.

Definition 2.20. An equivalence $\equiv_X$ on expressions is a *congruence* for an n-ary operator *op* iff $p_i \equiv_X q_i$ for $i \in \{1, \dots, n\}$ implies $op(p_1, \dots, p_n) \equiv_X op(q_1, \dots, q_n)$.

If we let a preorder $\sqsubseteq_X$ take the place of $\equiv_X$ in the definition, we say that $\sqsubseteq_X$ is a *precongruence* and that *op* is *monotone* with respect to $\sqsubseteq_X$.

Once soundness is established, it usually becomes feasible to justify a *normalization* procedure on terms in order to prove completeness.

2.3.2 Axiomatizing Weak Bisimilarity

Milner [Mil89, 7.4f.] provides a complete axiomatization for weak bisimilarity on *finite-state* CCS processes.

A process expression is called *finite-state* iff it only consists of prefix, choice, process names, and a **0**-process. If it additionally lacks recursion, then the process is called *finite.* Finite-state processes are isomorphic to finite transition systems, and finite processes correspond to acyclic finite transition systems.

To be exact, the axiomatization does not work with ordinary weak bisimilarity, but with *rooted weak bisimilarity.* This is because weak bisimilarity is no congruence for the operators of CCS.

Example 2.21. $\tau.\mathbf{0} \equiv_{WB} \mathbf{0}$ and $\mathrm{a} \equiv_{WB} \mathrm{a}$, but $\tau.\mathbf{0} + \mathrm{a} \not\equiv_{WB} \mathrm{a}$.

The example shows that $\equiv_{WB}$ is no congruence with respect to the operator $+$. In its place, we need to use rooted weak bisimilarity, which is often called "weak bisimulation congruence."

Definition 2.22 (Rooted weak bisimilarity)**.** Two states are *rooted weakly bisimilar*, written $p \equiv_{WB^c} q$, iff

1. for all p' and α with $p \xrightarrow{\alpha} p'$, there is a q' such that $q \overset{\alpha}{\Rightarrow} q'$ and $p' \equiv_{WB} q'$, and
2. for all q' and α with $q \xrightarrow{\alpha} q'$, there is a p' such that $p \overset{\alpha}{\Rightarrow} p'$ and $p' \equiv_{WB} q'$.

In contrast to weak bisimilarity, rooted weak bisimilarity requires that initial τ-steps are bisimulated by at least one real τ-step (and not stuttering). After that, the definition falls back to standard weak bisimilarity $\equiv_{WB}$. The result is slightly finer than weak bisimilarity, $\equiv_{WB^c} \subseteq \equiv_{WB}$.

This minimal deviation from the original requirements of $\equiv_{WB}$ actually is enough to transform weak bisimilarity into a congruence for finite-state CCS processes. Example 2.21 is solved because $\tau.\mathbf{0} \not\equiv_{WB^c} \mathbf{0}$.

Definition 2.23 ($\equiv_{WB^c}$ axioms)**.** We define the following axiom system $\mathcal{A}_{WB}$:

$$\begin{array}{lrcl}
\textbf{AC1} & P + Q & = & Q + P \\
\textbf{AC2} & P + (Q + R) & = & (P + Q) + R \\
\textbf{AC3} & P + P & = & P \\
\textbf{AC4} & P + \mathbf{0} & = & P \\
\textbf{AT1} & \alpha.\tau.P & = & \alpha.P \\
\textbf{AT2} & P + \tau.P & = & \tau.P \\
\textbf{AT3} & \alpha.(P + \tau.Q) & = & \alpha.(P + \tau.Q) + \alpha.Q
\end{array}$$

The laws **AC1**–**AC4** are quite straight-forward axioms for the choice operator. They are valid because the corresponding nodes in transition systems according to the operational CCS semantics (Definition 2.4) have identical outgoing transitions.

The laws **AT1**–**AT3** are referred to as the "τ-laws." They capture the intuition that the observer of a system cannot tell how much internal activity occurs between visible activity. They as well are valid for $\equiv_{WB^c}$.

While soundness of the laws is not really a problem, completeness takes some proof effort. The proof relies on a normalization of process terms into *WB*-normal forms ("full standard forms" in [Mil89]), and this normalization is what is of real interest to us in our pursuit of decision procedures for process equalities.

Definition 2.24 (*WB*-normal form)**.** A process term P is in *WB-normal form* iff

1. it has the structure $P = \alpha_1.P_1 + \alpha_2.P_2 + \ldots + \alpha_n.P_n$ (or $P = \mathbf{0}$),
2. every P_i is in *WB*-normal form, and
3. $P \overset{\alpha}{\Rightarrow} P'$ implies there is an i such that $\alpha_i = \alpha$ and $P_i = P'$.

Every finite process can soundly be rewritten into such a normal form using a method of *saturation* [Mil89], adding all the implied weak transitions as explicit process prefixes. With slight modifications, this applies to finite-state processes also.

On the transition systems of such normal forms weak and strong steps coincide, that is, $\overset{\cdot}{\Rightarrow} = \overset{\cdot}{\rightarrow}$. Consequently, weak bisimilarity $\equiv_{WB}$ and strong bisimilarity $\equiv_B$ *almost* match. The trick to make them match exactly is adding τ-loops to every state. This addition is transparent to weak bisimilarity (though not to rooted weak bisimilarity), and creates transition systems where $\overset{\hat{\cdot}}{\Rightarrow} = \overset{\cdot}{\rightarrow}$ and thus $\equiv_{WB} = \equiv_{SB}$.

This illustrates how normalization procedures in completeness proofs often justify how to rewrite models such that the equivalence coincides with strong bisimilarity on the result system. The proofs tell us how to reduce deciding one notion of equivalence to deciding strong bisimilarity, which is a common approach (cf. Aceto et al. "The algorithmics of bisimilarity" [AIS11]).

For weak bisimilarity, one might argue that it is obvious to reduce the problem to strong bisimilarity on $\overset{\hat{\cdot}}{\Rightarrow}$ by computing the weak transition relation. For coupled similarity however, we are going to encounter a not-so-obvious reduction in Section 4.2 based on an axiomatization we discuss in Section 3.3.

2.4 Game-Theoretic Characterizations

There is a tradition of characterizing logics in terms of specialized two-player games with perfect information [Hod13]. Notions of equivalence correspond to certain modal logics describing observations on systems [vG93]. In this light, it comes as no surprise that notions of (bi-)similarity can also be expressed in terms of games. We here introduce *simple games* in the spirit of the "finite games" from [Grä07], but we will not demand finiteness for now. We then cite the characterizations of weak similarity and bisimilarity as a preparation for our characterization of coupled similarity in Section 3.4.

2.4.1 Simple Games

Checking whether two states are related by a (bi-)simulation preorder $\sqsubseteq_X$ can also be seen as a *game* along the lines of coinductive characterizations like the one in Lemma 2.10. One player, the *attacker*, challenges that $p \sqsubseteq_X q$, while the other player, the *defender*, has to name witnesses for the existential quantifications of the definition. For the preorders discussed here, quite simple games suffice.

Definition 2.25 (Simple games)**.** A *simple game* $\mathcal{G}[p_0]$ consists of

- a (countable) set of *game positions* G,
    - partitioned into a set of *defender positions* $G_d \subseteq G$
    - and *attacker positions* $G_a := G \backslash G_d$,
- and a graph of *game moves* $\rightarrowtail \subseteq G \times G$,

where $p_0 \in G$ names an *initial position.*[13]

The positions G_a and G_d specify whose turn it is.

Definition 2.26 (Plays and wins)**.** We call the paths $p_0 p_1 ... \in G^\infty$ with $p_i \rightarrowtail p_{i+1}$ *plays* of $\mathcal{G}[p_0]$. The defender *wins* all infinite plays. If a finite play $p_0 \dots p_n$ is stuck, that is, if $p_n \not\rightarrowtail$, then the stuck player loses: The defender wins if $p_n \in G_a$, and the attacker wins if $p_n \in G_d$.

A strategy tells a player how to win. For our purposes, we are only interested in the winning strategies of the defender.

Definition 2.27 (Strategies and winning strategies)**.** A *defender strategy* is a (usually partial) mapping from initial play fragments to next moves $f \subseteq \{(p_0 ... p_n, p') \mid p_n \in G_d \wedge p_n \rightarrowtail p'\}$. A play p follows a strategy f iff, for each move $p_i \rightarrowtail p_{i+1}$ with $p_i \in G_d$, $p_{i+1} = f(p_0 ... p_i)$. If every such play is won by the defender, f is a *winning strategy* for the defender. The player with a winning strategy for $\mathcal{G}[p_0]$ is said to *win* $\mathcal{G}[p_0]$.

Definition 2.28 (Winning regions and determinacy)**.** The *winning region* W_σ of player $\sigma \in \{a, d\}$ for a game $\mathcal{G}$ is the set of states p_0 from which player σ wins $\mathcal{G}[p_0]$. We call $\mathcal{G}$ determined if $G = W_d \cup W_a$.

Proposition 2.29. *All simple games are determined.*

Proof. (Sketch) The winning condition for the defender from Definition 2.26 obviously is a safety property ("Don't run into certain deadlocks."). Safety properties correspond to closed sets of plays. The Gale–Stewart theorem guarantees games that have closed sets as winning conditions to be determined. □

2.4.2 Characterizing Weak Similarity and Bisimilarity with Games

The coinductive characterization of $\sqsubseteq_{WS}$ in Lemma 2.10 can easily be transformed into a simple game [Sti93]. We skip the proofs for this. They are not so different from what we shall do for coupled similarity in Section 3.4 in more detail.

[13] locale `Simple_Game.simple_game`

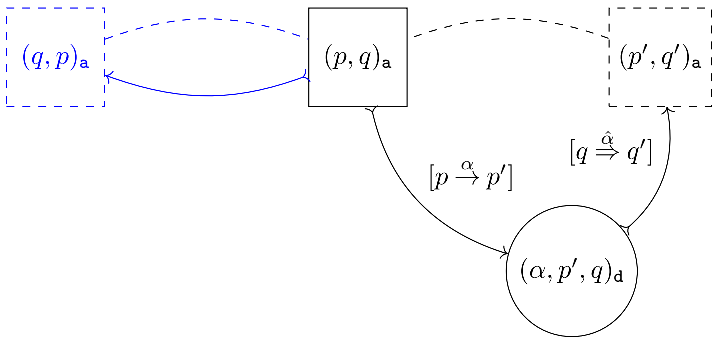

$(q,p)_\mathtt{a}$
$(p,q)_\mathtt{a}$
$(p',q')_\mathtt{a}$
$[p \xrightarrow{\alpha} p']$
$[q \overset{\hat{\alpha}}{\Rightarrow} q']$
$(\alpha,p',q)_\mathtt{d}$

Figure 2.4: Schematic weak (bi-)simulation game.

Definition 2.30 ($\sqsubseteq_{WS}$ game)**.** The *weak simulation game* $\mathcal{G}_{WS}[p_0] = (G, G_a, \rightarrowtail, p_0)$ for a transition system $\mathcal{S} = (S, \Sigma_\tau, \rightarrow)$ consists of

- *attacker nodes* $(p,q)_\mathtt{a} \in G_a$ with $p, q \in S$, and
- *defender nodes* $(\alpha, p, q)_\mathtt{d} \in G_d$ for situations where a simulation challenge has been formulated,

and two kinds of moves

- *challenges* $(p,q)_\mathtt{a} \rightarrowtail (\alpha, p', q)_\mathtt{d}$ if $p \xrightarrow{\alpha} p'$, and
- *answers* $(\alpha, p', q)_\mathtt{d} \rightarrowtail (p', q')_\mathtt{a}$ if $q \overset{\hat{\alpha}}{\Rightarrow} q'$.

The right-hand side of Figure 2.4 (the black part) gives a schematic illustration of this game. The square nodes represent kinds of attacker positions, and the circle nodes kinds of defender positions. Arrows stand for schematic game moves. The dashed positions are places where the game returns to attacker positions of the initial kind, but with a new variable assignment.

Proposition 2.31. *The defender wins $\mathcal{G}_{WS}[(p,q)_\mathtt{a}]$ precisely if $p \sqsubseteq_{WS} q$.*

$\mathcal{G}_{WS}$ can easily be extended with *swaps* of sides to establish *symmetry* of the characterized relation, thus capturing weak *bi*-similarity. Symmetry swaps are represented by the blue part of Figure 2.4.

Definition 2.32 ($\equiv_{WB}$ game)**.** The *weak bisimulation game* $\mathcal{G}_{WB}$ for a transition system $\mathcal{S} = (S, \Sigma_\tau, \rightarrow)$ is just $\mathcal{G}_{WS}$ with one additional kind of moves:

- *symmetry swaps* $(p,q)_\mathtt{a} \rightarrowtail (q,p)_\mathtt{a}$.

This means that the attacker, where they see fit, can choose to attack on the right-hand side instead of only the left-hand side.

Proposition 2.33. *The defender wins $\mathcal{G}_{WB}[(p,q)_\mathtt{a}]$ precisely if $p \equiv_{WB} q$.*

Such games are sufficiently simple for algorithms on them to also be quite simple. That is why they can serve as a nice intermediate level of abstraction for algorithms computing behavioral equivalence relations. The details of this are provided in Section 4.4.

Chapter 3

Coupled Simulation

Coupled similarity strikes a deal between a lot of nice properties of weak similarity, contrasimilarity, and weak bisimilarity. As visualized in Figure 1.1, it sits right in the middle between them.

We here introduce relational, axiomatic and game-theoretic characterizations of coupled similarity and its preorder, which turn into algorithms in the next chapter. We moreover prove properties of the coupled simulation preorder that justify tricks in the algorithms of the next chapter.

3.1 Definitions of Coupled Simulation

A multitude of definitions for coupled simulation and coupled similarity have been proposed. Our thesis uses the coupled simulation presented by Rob van Glabbeek [vG17]. This section introduces the definition from [vG17] and compares it to other pre-existing notions of coupled simulation. Such a survey seems advisable because, over the years, there have been varying formulations of coupled simulation, which, to some extend, also differ in range.

3.1.1 Coupled Simulation [vG17]

Van Glabbeek's definition just merges the properties weak simulation and coupling familiar from Figure 2.1.

Definition 3.1 (Coupled simulation)**.** A *coupled simulation* is a relation $\mathcal{R} \subseteq S \times S$ such that, for all $(p, q) \in \mathcal{R}$,

- $p \xrightarrow{\alpha} p'$ implies there is a q' such that $q \overset{\hat{\alpha}}{\Rightarrow} q'$ and $(p', q') \in \mathcal{R}$ (*simulation*), and
- there exists a q' such that $q \Rightarrow q'$ and $(q', p) \in \mathcal{R}$ (*coupling*).

The *coupled simulation preorder* relates two processes, $p \sqsubseteq_{CS} q$, iff there is a coupled simulation $\mathcal{R}$ such that $(p, q) \in \mathcal{R}$. *Coupled similarity* relates two processes, $p \equiv_{CS} q$, iff $p \sqsubseteq_{CS} q$ and $q \sqsubseteq_{CS} p$.[1]

Example 3.2 (Chapter 2 examples)**.** As the relation $\mathcal{R}$ on the philosopher system $\mathcal{S}_\mathsf{P}$ from Figure 2.2 is a weak simulation, it fulfills the first condition of Definition 3.1.

[1] definition `Coupled_Simulation.coupled_simulation`

Because it also is a contrasimulation, and thus a coupling by Proposition 2.16, it lives up to the second condition. Consequently, $\mathcal{R}$ is a coupled simulation, and $\mathsf{P}_g \equiv_{CS} \mathsf{P}_o$.

The formulation of coupled simulation implying weak simulation directly yields that there cannot be a coupled simulation where there cannot be a weak simulation. This, for example, implies that choice drifting from Example 2.18 is distinguished, and $\mathsf{P}_r \not\equiv_{CS} \mathsf{P}'_r$ for the variants of the religious philosophers.

The demand for coupling, on the other hand, detects the troll from Example 2.11. The -state does not simulate any other state and has no τ-successors and therefore cannot be *coupled.* So, the $\mathsf{P}_t \xrightarrow{\tau}$ -transition cannot be simulated by P_o, which warrants $\mathsf{P}_o \not\equiv_{CS} \mathsf{P}_t$.

The example demonstrates right away that the coupled simulation preorder and coupled similarity fit our bill outlined in the previous chapter. We shall dive deeper into their properties in Section 3.2. Let us first continue our survey of definitions for coupled simulation from the literature.

3.1.2 Stability-Coupled Simulation [PS92, PS94]

The historically first definition of coupled simulation is due to Parrow and Sjödin [PS92, PS94]. It works with *two* simulations that are coupled *at the stable states.* It is nowadays referred to as "S-coupled simulation" (for example in [San12]).

Definition 3.3 (S-coupled simulation)**.** An *S-coupled simulation* is a pair of weak simulations $\mathcal{R}_1, \mathcal{R}_2^{-1} \subseteq S \times S$, which are mutually "S-coupled" (not in the sense of the previous sections!) at the stable states, that is:

- $(p, q) \in \mathcal{R}_1$ and p stable implies $(p, q) \in \mathcal{R}_2$, and
- $(p, q) \in \mathcal{R}_2$ and q stable implies $(p, q) \in \mathcal{R}_1$.

S-*coupled similarity* relates two processes, $p \equiv_{SCS} q$, iff there is an S-coupled simulation $(\mathcal{R}_1, \mathcal{R}_2)$ such that $(p, q) \in \mathcal{R}_1 \cap \mathcal{R}_2$.

S-coupled similarity on states of divergence-free systems is an equivalence relation. However, S-coupled similarity is no fit notion of equivalence for general systems, that may contain divergences.

Example 3.4 ($\equiv_{SCS}$ lack of transitivity)**.** Consider the transition system given in Figure 3.1 (example from [PS94, p. 567]). We have $p_1 \equiv_{SCS} q_1$ and $q_1 \equiv_{SCS} r_1$ but $p_1 \not\equiv_{SCS} r_1$. So, $\equiv_{SCS}$ is not transitive and thus not an equivalence relation.

Note that coupled similarity, as we defined it in Subsection 3.1.1, does not suffer from this shortcoming. It maintains that $q_1 \not\equiv_{CS} r_1$ because it also demands coupling on q_4.

The example shows that $\equiv_{SCS}$ and $\equiv_{CS}$ actually are different notions. Still, for many systems, they match.[2]

Lemma 3.5. *If $\mathcal{S}$ is a divergence-free system, then, provided that p and q are equally stable ("stability-rooted"), $p \equiv^{\mathcal{S}}_{CS} q$ if and only if $p \equiv^{\mathcal{S}}_{SCS} q$.*[3]

[2][PS94] and [San12] claim a variant of this to hold without the assumption of shared stability. Following the construction from [San12], our Isabelle proof still needs this extra assumption and it is not obvious how to dispose of it.

[3] theorem `Coupled_Simulation.divergence_free_coupledsims_coincidence`

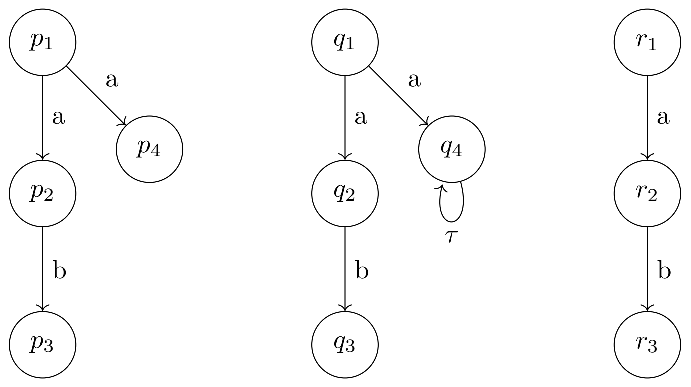


Figure 3.1: Processes for Example 3.4.

So, in the absence of divergence, stability-rooted S-coupled similarity has the nice properties of coupled similarity we shall discuss in Section 3.2. The literature suggests that these are precisely the cases where S-coupled similarity is useful. Consequently, it seems wise to consider S-coupled similarity just as an alternative characterization of coupled simulation for these special cases. But even then, it usually is not as handy as [vG17]-coupled simulation, which motivates our choice of the latter over the first.

3.1.3 Coupled Simulation Using Pairs [PS94, San12]

The conclusion section of [PS94] already discusses the problems of S-coupled similarity in the face of divergences and proposes "weakly coupled simulation" as a remedy. This proposal is what usually is meant with the name "coupled simulation" since [PS94], for example in [San12, NP00, Ren00]. To avoid confusion with Definition 3.1, let us call this definition *pair-based coupled simulation.*

Definition 3.6 (Pair-based coupled simulation)**.** A *pair-based coupled simulation* is a pair of weak[4] simulations $\mathcal{R}_1, \mathcal{R}_2^{-1} \subseteq S \times S$ that are mutually coupled, that is:

- $(p, q) \in \mathcal{R}_1$ implies there is q' with $q \Rightarrow q'$ and $(p, q') \in \mathcal{R}_2$, and
- $(p, q) \in \mathcal{R}_2$ implies there is p' with $p \Rightarrow p'$ and $(p', q) \in \mathcal{R}_1$.

The difference to Definition 3.3 is that coupling now is phrased like in Definition 3.1 and required not only on the stable states.

It turns out that this concept of coupled simulation has some inherent redundancy and coincides with single-relation *coupled simulation* as presented in Definition 3.1.

Lemma 3.7. *The pair-based definition and* van Glabbeek*'s definition of coupled similarity coincide.*

- *If $\mathcal{R}$ is a [vG17]-coupled simulation, then $(\mathcal{R}, \mathcal{R}^{-1})$ is a pair-based [San12]-coupled simulation.*[5]

[4][PS94, San12] here just speak of "simulation", not "weak simulation." But, from their examples and lemmas, it seems reasonable to read a "weak" into their definitions. [NP00] explicitly requires weak simulations. [Ren00] points out that actually requiring finer *delay simulations* is enough. Delay simulations are like weak simulations with $\Rightarrow\xrightarrow{a}$ instead of $\Rightarrow\xrightarrow{a}\Rightarrow$ in the visible simulation answer. More on this in Subsection 3.2.2.

[5] lemma `Coupled_Simulation.coupledsim_gla17_resembles_sangiorgi12`

- *If $(\mathcal{R}_1, \mathcal{R}_2)$ is a pair-based [San12]-coupled simulation, then $\mathcal{R}_1 \cup \mathcal{R}_2^{-1}$ is a [vG17]-coupled simulation.*[6]

In this light, the label "weakly coupled" from [PS94] has to be considered extremely misleading. In Example 3.4, we already encountered a situation where the allegedly "weakly coupled" simulation actually requires *more coupling* than S-coupled simulation.

Coupled simulations with two relations are considered more tedious for proofs than ordinary simulations and bisimulations [San12, p. 179]. This might be one of the reasons why coupled simulations have not been widely used up to now. Fortunately, van Glabbeek's definition [vG17] is more convenient.

3.1.4 Barbed and Reduction Coupled Simulation [PvG15]

Fournet and Gonthier [FG05] as well as Peters and van Glabbeek [PvG15] present a variation of coupled simulation for reduction semantics, that is, for systems where all transitions are τ-labeled, and other properties of reachable states matter.

Definition 3.8 (Reduction coupled simulation)**.** A *reduction coupled simulation* is a relation $\mathcal{R} \subseteq S \times S$ such that, for all $(p, q) \in \mathcal{R}$ and p' with $p \Rightarrow p'$,

- there exists a q' such that $q \Rightarrow q'$ and $(p', q') \in \mathcal{R}$, and
- there exists a q' such that $q \Rightarrow q'$ and $(q', p') \in \mathcal{R}$.

This property of $\mathcal{R}$ is meant to be accompanied by some further constraints, for instance to respect *barbs*. Then, it can be used to generate notions of reduction coupled similarity.

As one would hope, given that [PvG15] and [vG17] share an author, reduction coupled simulation indeed is a special case of the coupled simulation definition we are using in this thesis.

Lemma 3.9. *For reduction-semantical transition systems, Definition 3.8 and Definition 3.1 have the same extension.*

- *Every [vG17]-coupled simulation also is a* [PvG15]*-reduction-coupled simulation.*[7]
- *If a system only has τ-transitions, then every* [PvG15]*-reduction-coupled simulation also is a [vG17]-coupled simulation.*[8]

Interestingly, Theorem 4 of [FG05] shows that barbed reduction coupled similarity and *fair testing equivalence* coincide for the local π-calculus. The *local π-calculus* is a distributable variant of the π-calculus where processes may not receive on free or received names. Two processes p, q are *fair testing equivalent* iff, for every context and barb, all successors of p may reach the barb precisely if all reachable successors of q may. We note this because it points to the fact that coupled similarity may come close to or even touch the—usually coarser—world of testing equivalences.

[6] lemma `Coupled_Simulation.coupledsim_sangiorgi12_impl_gla17`
[7] lemma `Coupled_Simulation.coupledsim_gla17_implies_gp15`
[8] lemma `Coupled_Simulation.coupledsim_gp15_implies_gla17_on_tau_systems`

3.1.5 Coupled Simulation as Special Contrasimulation?

The standard definitions of coupled simulation introduce it as a special brand of simulations. But coupled simulation also is a special case of contrasimulation.[9] Indeed, coupled simulation (as defined in Definition 3.1) is *precisely* the intersection of the two concepts. (So, Figure 1.1 from the introduction does not only relate equivalences, "similarities", but already simulation relations.) The "surprising" direction of this, actually is quite obvious in the light of the coupling property (Proposition 2.16) of contrasimulation.

Lemma 3.10. *$\mathcal{R}$ is a coupled simulation* precisely if *$\mathcal{R}$ is a weak simulation and a contrasimulation.*[10]

Now, it suggests itself to look for a concise characterization of coupled simulation in terms of contrasimulation. In the literature, there seems to be only one instance where this has been tried. Voorhoeve and Mauw's tech report on contrasimilarity [VM00] defines coupled simulation as a special case of contrasimulation (notation adapted):

> "A coupled simulation is a contrasimulation $\mathcal{R}$ satisfying
> $\{x \mid \exists y.(x,y) \in \mathcal{R}\} = \{y \mid \exists x.(x,y) \in \mathcal{R}\}$."

Unfortunately, this characterization is too weak. For instance, consider the contrasimulation $\mathcal{R}$ on the religious philosopher system from Figure 2.3. Its reflexive closure $\mathcal{R} \cup \Delta_S$ still is a contrasimulation[11] and matches the definition. But it is no weak simulation and thus no coupled simulation by the arguments from Example 2.18.

So, whatever it is [VM00] is defining, it is not compatible with the common notions of coupled simulation, which all require coupled simulations to be weak simulations. In the journal version [VM01], the definition of coupled simulation has been dropped.

The previous reasoning also disproves Kučera and Schnoebelen's Remark 6 of [KS06] that contrasimulation and coupled simulation would coincide on divergence-free processes—the transition system in Figure 2.3 is divergence-free.

In summary, there appears to exist some confusion concerning the exact relationship of coupled simulation and contrasimulation. It would be interesting to look deeper into this, but that is beside the point of this thesis.

3.2 Properties of Coupled Similarity

The coupled simulation preorder has a few characteristics that render it quite suited for proofs and models. Some of them are particularly relevant for our algorithms in the following chapter. From now on, we stick to coupled simulation as specified in Definition 3.1. Results nonetheless carry over to the other definitions according to the established coincidences.

3.2.1 Order Properties and Coinduction

The coupled simulation preorder $\sqsubseteq_{CS}$ has the properties everyone expects from a decent notion of equivalence.

[9] lemma `Coupled_Simulation.coupledsim_implies_contrasim`
[10] lemma `Coupled_Simulation.coupled_simulation_iff_weak_sim_and_contrasim`
[11] lemma `Weak_Relations.contrasim_union,contrasim_reflexive`

Lemma 3.11. $\sqsubseteq_{CS}$ *forms a preorder, that is, it is reflexive*[12] *and transitive.*[13] *As a consequence, coupled similarity* $\equiv_{CS}$ *is an equivalence relation.*[14]

Coupled simulation is closed under set union,[15] and the coupled simulation preorder is itself a coupled simulation[16] (and thus the greatest coupled simulation).

Lemma 3.12. *The coupled simulation preorder can be characterized,*

1. by the union*:*

$$\sqsubseteq_{CS} = \bigcup\{\mathcal{R} \mid \mathcal{R} \textit{ is a coupled simulation}\}, \textit{ and}$$

2. coinductively[17] *by the rule:*[18]

$$\frac{\forall p', \alpha.\ p \xrightarrow{\alpha} p' \longrightarrow \exists q'.\ q \overset{\hat{\alpha}}{\Rightarrow} q' \ \wedge\ p' \sqsubseteq_{CS} q' \qquad q \Rightarrow q' \qquad q' \sqsubseteq_{CS} p}{p \sqsubseteq_{CS} q}.$$

Especially the coinductive characterization is important because it allows to conduct proofs for $p \sqsubseteq_{CS} q$ between some states p and q coinductively. For this thesis, it also plays an important role in motivating the following game characterization (Section 3.4) and algorithms (Sections 4.3ff.).

3.2.2 Strengthening Coupled Simulation

Like weak similarity and bisimilarity, yet unlike contrasimilarity, coupled similarity can be defined with a weaker formulation of the simulation challenge.

Lemma 3.13. *The "$p \xrightarrow{\alpha} p'$" on the left-hand side of the simulation property from Definition 3.1 may be replaced by "$p \overset{\hat{\alpha}}{\Rightarrow} p'$" while the extension of the definition remains the same.*[19]

This is a logically stronger formulation of coupled simulation, which is helpful when coupled simulation appears as a premise in a proof.

Coupled similarity can also be defined employing an effectively stronger concept than weak simulation, namely *delay simulation*. Delay simulations are defined in terms of a "shortened" weak step relation $\overset{\hat{\alpha}}{\Rightarrow\!\!\!\!>}$ where $\overset{\hat{\tau}}{\Rightarrow\!\!\!\!>} := \Rightarrow$ and $\overset{\hat{a}}{\Rightarrow\!\!\!\!>} := \Rightarrow\xrightarrow{a}$. So the difference between $\overset{\hat{a}}{\Rightarrow\!\!\!\!>}$ and $\overset{\hat{a}}{\Rightarrow}$ lies in the fact that the latter can move on with τ-steps after the strong $\xrightarrow{a}$-step in its construction.

Definition 3.14 (Coupled delay simulation)**.** A *coupled delay simulation* is a relation $\mathcal{R} \subseteq S \times S$ such that, for all $(p, q) \in \mathcal{R}$,

- $p \xrightarrow{\alpha} p'$ implies there is a q' such that $q \overset{\hat{\alpha}}{\Rightarrow\!\!\!\!>} q'$ and $(p', q') \in \mathcal{R}$ (*delay simulation*),
- and there exists a q' such that $q \Rightarrow q'$ and $(q', p) \in \mathcal{R}$ (*coupling*).

[12] lemma `Coupled_Simulation.coupledsim_refl`
[13] lemma `Coupled_Simulation.coupledsim_trans`
[14] lemma `Coupled_Simulation.coupled_similarity_equivalence`
[15] lemma `Coupled_Simulation.coupledsim_union`
[16] lemma `Coupled_Simulation.coupled_sim_by_is_coupled_sim`
[17] coinductive `Coupled_Simulation.greatest_coupled_simulation`
[18] lemma `Coupled_Simulation.gcs_eq_coupled_sim_by`
[19] lemma `Coupled_Simulation.coupled_simulation_weak_premise`

The only difference to Definition 3.1 is the use of $\overset{\hat{\alpha}}{\Rightarrow\!\!\rhd}$ instead of $\overset{\hat{\alpha}}{\Rightarrow}$. As $\overset{\hat{\alpha}}{\Rightarrow\!\!\rhd} \subseteq \overset{\hat{\alpha}}{\Rightarrow}$, every coupled delay simulation is a coupled simulation. However, not every coupled simulation also is a delay (coupled) simulation.[20] So, Definition 3.14 is not simply a stricter formulation, but indeed extensionally smaller than coupled simulation. Still, the *greatest* coupled simulation $\sqsubseteq_{CS}$ *is* a delay coupled simulation so that we can add the following characterization of the coupled simulation preorder.

Lemma 3.15. *$p \sqsubseteq_{CS} q$ precisely if there is a coupled delay simulation $\mathcal{R}$ such that $(p, q) \in \mathcal{R}$.*[21]

Rensink [Ren00] uses a definition for pair-based coupled simulation that is based on delay steps as well. For action-labeled coupled simulation on a single relation, the delay simulation approach seems not to have been used up to now, which is why we refrain from taking delay coupled simulations as canonical coupled simulations here.

By any account, it is a nice property because it means that algorithms (and proofs) do not have to bother with the extra load of fuzziness caused by the trailing τ-steps of $\overset{\hat{\cdot}}{\Rightarrow}$. Once symmetry is demanded instead of coupling, delay simulation and weak simulation lead to different notions of equivalence, namely delay bisimilarity and weak bisimilarity. Delay bisimilarity is strictly stronger and has important use cases, for example, applicative bisimulation on the λ-calculus [Pit11].

3.2.3 Internal Steps and Choice

The coupled simulation preorder is conceptionally linked to the internal choice structure of a system. One basis for this is that it contains the inverted τ-closure of the transition system.

Lemma 3.16. *If $q \Rightarrow p$, then $p \sqsubseteq_{CS} q$.*[22]

Intuitively, this is logical from the weak simulation nature of coupled similarity: p can obviously do at least as much as its τ-successor q, because it can perform a τ-step and then steal the abilities from q. And due to the reflexive nature of $\sqsubseteq_{CS}$, q can answer its own coupling challenge.

A direct implication is that strongly connected τ-components are coupled-similar:

Corollary 3.17. *If p and q are on a τ-cycle, that means $p \Rightarrow q$ and $q \Rightarrow p$, then $p \equiv_{CS} q$.*[23]

This corollary is a commonplace property of equivalences in the linear-time branching-time spectrum with silent actions. Weak equivalences usually allow for internal steps before answering a challenge. Therefore, if $p \Rightarrow q$ and $q \Rightarrow p$, then p and q can steal the transitions from each other when equivalence is questioned.

What does it mean, if we know for an internal choice between p and q that $p \sqsubseteq_{CS} q$? Interestingly: that there is no real choice.

Lemma 3.18. *Let $pq_{\sqcap} \in S$ such that $pq_{\sqcap} \xrightarrow{\alpha} pq'$ if and only if $\alpha = \tau$ and $pq' \in \{p, q\}$. Then, $p \sqsubseteq_{CS} q$ if and only if $pq_{\sqcap} \equiv_{CS} q$.*[24]

[20] Consider for example $\mathcal{R} = \{(\mathrm{a}.\tau, \mathrm{a}.\tau), (\tau, \mathbf{0}), (\mathbf{0}, \tau), (\mathbf{0}, \mathbf{0})\}$.
[21] lemma `Coupled_Simulation.coupled_sim_by_eq_delay_coupled_simulation`
[22] lemma `Coupled_Simulation.coupledsim_step`
[23] lemma `Coupled_Simulation.strongly_tau_connected_coupled_similar`
[24] lemma `Coupled_Simulation.coupledsim_choice_join`

We formulated the lemma for transition systems, borrowing the symbol $\sqcap$ commonly used for internal choice in the process calculus Communicating Sequential Processes (CSP). In CCS, the lemma would read

$$P \sqsubseteq_{CS} Q \text{ if and only if } \tau.P + \tau.Q \equiv_{CS} Q.$$

This is to say, internal choice acts as a least upper bound (join) in the $(\mathsf{CCS}, \sqsubseteq_{CS})$ semi-lattice with $\equiv_{CS}$ as equality.

Ordinary coupled simulation is blind to divergence. In particular, it cannot distinguish two states whose outgoing transitions only differ in an additional τ-loop at the second state.

Lemma 3.19. $p \equiv_{CS} q$ *if* $q \overset{\cdot}{\rightarrow} \cdot = p \overset{\cdot}{\rightarrow} \cdot \cup \{(\tau, q)\}$.[25]

This lemma, together with the τ-cycle compression from Corollary 3.17, allows to convert finite systems with divergence into $\equiv_{CS}$-equivalent systems without divergence, thereby bridging the gap between $\equiv_{CS}$ and $\equiv_{SCS}$.

The previous lemmas stem from the weak simulation nature of $\sqsubseteq_{CS}$. However, adapting words from [vG17], $p \sqsubseteq_{CS} q$ does not only mean that "p is *ahead* of q" (weak simulation, Lemma 3.16ff.), but also that "q can *catch up* to p" (coupling).

The important property of coupled similarity setting it apart from weak similarity is that it enforces more symmetry:

Lemma 3.20. *Assume $\mathcal{S}$ is finite and has no τ-cycles. Then $p \sqsubseteq_{CS} q$ and $p \overset{\hat{\alpha}}{\Rightarrow} p'$ with stable p' imply there is a stable q' such that $q \overset{\hat{\alpha}}{\Rightarrow} q'$ and $p' \equiv_{CS} q'$.*[26]

In other words: If we ignore intermediate instable internal choice points and only consider $\overset{\hat{\cdot}}{\Rightarrow}$-steps leading to τ-maximal states, coupled similarity virtually resembles weak bisimilarity on finite divergence-free systems. The next section looks at the connection of weak bisimilarity and coupled similarity in more depth.

3.3 Axiomatization of Coupled Similarity

One of the most important works on coupled similarity is Parrow and Sjödin's "The complete axiomatization of Cs-congruence" [PS94]. They provide an axiomatization of $\equiv_{SCS}$-congruence for finite CCS processes, which is based on Milner's axiomatization of $\equiv_{WB}$ [Mil89], we reported in Section 2.3.

3.3.1 Coupled Simulation Congruence

Coupled similarity is no congruence with respect to the CCS choice operator. It suffers from the same shortcoming as weak bisimilarity in Example 2.21, namely that $\tau.\mathbf{0} \equiv_{CS} \mathbf{0}$ and $\mathrm{a} \equiv_{CS} \mathrm{a}$, but $\tau.\mathbf{0} + \mathrm{a} \not\equiv_{CS} \mathrm{a}$.[27] However, its largest CCS congruence has a nicer characterization than the special treatment for initial τ-steps of rooted weak bisimilarity in Definition 2.22.

[25] lemma `Coupled_Simulation.coupledsim_tau_loop_ignorance`

[26] lemma `Coupled_Simulation.coupledsim_eventual_symmetry`

[27] On CCS descendants without the expressive power of $+$, $\equiv_{CS}$ usually is a congruence. One important example is the asynchronous π-calculus [NP00, Prop. 2.4.4].

Definition 3.21 (Rooted coupled similarity)**.** Two states are *rooted coupled similar*, written $p \equiv_{CS^c} q$, iff $p \equiv_{CS} q$, and p stable precisely if q stable.

Proofs that $\equiv_{CS^c}$ indeed is a congruence and that it is the largest one can be found in [San12]. Lemma 3.5 already yields that $\equiv_{CS^c}$ matches the congruence [PS94] similarly derives from $\equiv_{SCS}$ on finite processes because their transition systems are cycle-free and finite-state, and hence divergence-free. Therefore, the axiomatization from [PS94] for $\equiv_{SCS}$-congruence is just as well an axiomatization for $\equiv_{CS^c}$. Now, let us have a look at the axiomatization.

3.3.2 A New τ-Law

Coupled similarity is coarser than weak bisimilarity. In order to be complete, its axiomatization must declare *more* processes to be equal than the weak bisimilarity axiom system $\mathcal{A}_{WB}$ (Definition 2.23).

Definition 3.22 ($\equiv_{CS^c}$ axioms)**.** We define $\mathcal{A}_{CS}$ as an extension of $\mathcal{A}_{WB}$ from Definition 2.23 with the new law

$$\textbf{CS} \quad \tau.(\tau.P + Q) \quad = \quad \tau.P + Q.$$

The axiom basically says that τ-steps to instable states *can be skipped.*

Example 3.23. If we encode the philosopher transition system P_g from Example 2.5 as a finite-state process, we can rewrite it using **CS** to match P_o.

$$\begin{aligned}
\mathsf{P}_g \quad & \overset{\text{(encode)}}{\equiv_B} \quad \tau.\tau.\mathsf{A} + \tau.(\tau.\mathsf{B} + \tau.\mathsf{C}) \\
& \overset{\textbf{CS}}{=} \quad \tau.\tau.\mathsf{A} + (\tau.\mathsf{B} + \tau.\mathsf{C}) \\
& \overset{\textbf{AT1}}{=} \quad \tau.\mathsf{A} + \tau.\mathsf{B} + \tau.\mathsf{C} \qquad \overset{\text{(encode)}}{\equiv_B} \; \mathsf{P}_o
\end{aligned}$$

The example shows that $\mathcal{A}_{CS}$ is sufficiently complete to cover that $\mathsf{P}_g \equiv_{CS} \mathsf{P}_o$. But is it completely complete?

3.3.3 CS-Normalization

To prove completeness for finite CCS processes, [PS94] adapts Milner's *WB*-normal form (Definition 2.24). The crucial trick is to include a closure over the partial choices of instable choice points.

Definition 3.24 (*CS*-normal form)**.** A process term P is in *CS-normal form* iff

1. it has the structure $P = \alpha_1.P_1 + \ldots + \alpha_n.P_n$ (or $P = \mathbf{0}$),
2. every P_i is in *CS*-normal form,
3. $P \overset{\alpha}{\Rightarrow} P'$ implies there is an i such that $\alpha_i = \alpha$ and $P_i = P'$,
4. **new:** if $\alpha_i = \tau$ for some i, then P_i is stable, and
5. **new:** whenever $P_i = \beta_1.Q_1 + \ldots + \beta_m.Q_m$ for some i with $\beta_j = \tau$ for some j, then for every partial choice $\hat{P} = \hat{Q}_1 + \cdots + \tau.Q_j + \cdots + \hat{Q}_m$ of P_i where $\hat{Q}_k \in \{\mathbf{0}, \beta_k.Q_k\}$ for all $k \neq j$, there is some l such that the l-th branch of P is *CS*-congruent to the partial choice, that is, $P_l \equiv_{CS^c} \hat{P}$ with $\alpha_l = \alpha_i$.

Conditions 1, 2, and 3 are identical to *WB*-normal forms from Definition 2.24. Condition 4 is in line with the property of coupled similarity that τ-steps to instable states can be skipped. Condition 5 then demands the presence of all partial commits that could be hidden in the trailing $\Rightarrow$-part of a weak visible transition. We shall return to the properties of such normal forms in Definition 4.5, where there also is an illustration of how a transition system changes with normalization in Figure 4.1.

On *CS*-normal forms, coupled similarity and weak bisimilarity coincide.

Lemma 3.25. *If P and Q are CS-normal forms, then $P \equiv_{CS} Q$ precisely if $P \equiv_{WB} Q$, (and $P \equiv_{CS^c} Q$ precisely if $P \equiv_{WB^c} Q$). (Proof cf. Lemma 20 in [PS94].)*

Moreover, every finite CCS processes can be rewritten to a *CS*-normal form (Lemma 18 in [PS94]). As Parrow and Sjödin's induction proof for this makes use of the reference to $\equiv_{CS^c}$ for a structurally smaller term in clause 5 of the definition, the implied normalization is recursive.

Parrow and Sjödin conjecture that their normalization and the completeness proof can be extended to finite-state processes by adding the recursion laws from [Mil89]. Because this would enable divergent processes, one would have to use the variant of coupled similarity that remains an equivalence relation in this case (Definition 3.1.3). However, they do not elaborate on the question what consequences recursion would have on the normal forms. Due to condition 3, *CS*-normal forms cannot be divergent processes. We shall deal with this problem in Section 4.2.

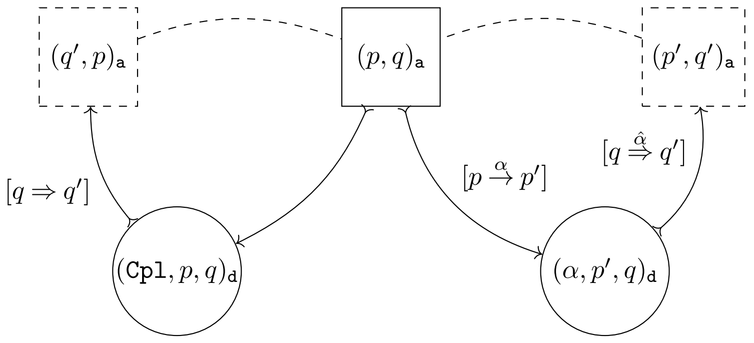
3.3.4 Axiomatization in CSP

As a side note, let us mention that coupled similarity in its divergence-sensitive flavor $\equiv_{\Delta CS}$ (cf. final remark in Subsection 2.2.4) is a congruence for the wide-spread process calculus CSP. In particular, van Glabbeek [vG17] shows that $\sqsubseteq_{\Delta CS}$ is a precongruence for "all"[28] CSP operators except for action prefixing and the related throw operator, for which, nevertheless, $\equiv_{\Delta CS}$ is a congruence. He moreover gives a complete axiomatization of $\equiv_{\Delta CS}$ within recursion-free CSP without interrupts. Most relevant laws of the standard CSP axiom set are provable from van Glabbeek's axiom system. This shows that many of the intuitions behind the interplay of synchronizations and choices that lie behind the design of CSP correspond to the notions implicit in coupled similarity.

3.4 Coupled Simulation as a Game

As introduced in Section 2.4, checking preorders with coinductive characterizations like the one of $\sqsubseteq_{CS}$ in Lemma 3.12 can be expressed by games between an *attacker*, challenging that $p \sqsubseteq_{CS} q$, and a *defender*, naming witnesses for the existential quantifications of the coinduction premises. Let us recall that the difference between the weak simulation game and the weak bisimulation game consists in the opportunity of the attacker to switch the direction of the game, which captures symmetry. For the coupled simulation game, we introduce a game *in between* the two games.

[28]This "all" has to be understood with some reservations, since there seemingly are indefinitely many CSP operators out there.



$(q', p)_{\mathtt{a}}$
$(p, q)_{\mathtt{a}}$
$(p', q')_{\mathtt{a}}$
$[q \Rightarrow q']$
$[p \xrightarrow{\alpha} p']$
$[q \overset{\hat{\alpha}}{\Rightarrow} q']$
$(\mathtt{Cpl}, p, q)_{\mathtt{d}}$
$(\alpha, p', q)_{\mathtt{d}}$

Figure 3.2: Schematic coupled simulation game.

3.4.1 The Coupled Simulation Game

The *coupled simulation game* proceeds as follows: For $p \sqsubseteq_{CS} q$, the attacker may question that simulation holds by selecting p' and α with $p \xrightarrow{\alpha} p'$. The defender then has to name a q' with $q \overset{\hat{\alpha}}{\Rightarrow} q'$, whereupon the attacker may go on to challenge $p' \sqsubseteq_{CS} q'$. For coupled simulation, the attacker may moreover demand the defender to name a coupling witness q' with $q \Rightarrow q'$ whereafter $q' \sqsubseteq_{CS} p$ stands to questions. If the defender runs out of answers, they lose; if the game continues forever, they win.[29] This can be modeled by a simple game, whose schema is given in Figure 3.2, as follows.

Definition 3.26 ($\sqsubseteq_{CS}$ game)**.** The *coupled simulation game* $\mathcal{G}^{\mathcal{S}}_{CS}[p_0] = (G, G_a, \rightarrowtail, p_0)$ for a transition system $\mathcal{S} = (S, \Sigma_\tau, \rightarrow)$ consists of

- *attacker nodes* $(p, q)_{\mathtt{a}} \in G_a$ with $p, q \in S$,
- *simulation defender nodes* $(\alpha, p, q)_{\mathtt{d}} \in G_d$ for situations where a simulation challenge has been formulated, and
- **new:** *coupling defender nodes* $(\mathtt{Cpl}, p, q)_{\mathtt{d}} \in G_d$ for situations where coupling is challenged,

and four kinds of moves

- *simulation challenges* $(p, q)_{\mathtt{a}} \rightarrowtail (\alpha, p', q)_{\mathtt{d}}$ if $p \xrightarrow{\alpha} p'$,
- *simulation answers* $(\alpha, p', q)_{\mathtt{d}} \rightarrowtail (p', q')_{\mathtt{a}}$ if $q \overset{\hat{\alpha}}{\Rightarrow} q'$,
- **new:** *coupling challenges* $(p, q)_{\mathtt{a}} \rightarrowtail (\mathtt{Cpl}, p, q)_{\mathtt{d}}$, and
- **new:** *coupling answers* $(\mathtt{Cpl}, p, q)_{\mathtt{d}} \rightarrowtail (q', p)_{\mathtt{a}}$ if $q \Rightarrow q'$.[30]

The differences to the weak simulation game (Definition 2.30) are marked as "**new**." They enable something quite similar to the symmetry swaps of Definition 2.32 with the innovation that the defender has a say in the direction change. They may choose a q' with $q \Rightarrow q'$ from where to continue after the swap ("coupling answers"). If we were to restrict the coupling answers to q' where $q = q'$ instead of $q \Rightarrow q'$, we would again end up with a game characterizing weak bisimilarity.

Intuitively, it is clear, that this is a game *in between* the weak simulation game $\mathcal{G}_{WS}$ and the weak bisimulation game $\mathcal{G}_{WB}$. The attacker can do more than in $\mathcal{G}_{WS}$,

[29] The attacker never runs out of options as they can always question coupling.
[30] locale `CS_Game.cs_game`

namely challenge coupling, and the defender can do more than in $\mathcal{G}_{WB}$, namely make a $\Rightarrow$-step before a symmetry attack may proceed. This is in line with our observation from Subsection 2.2.3 that coupling is symmetry relaxed by $\Rightarrow$.

3.4.2 Correctness of the Game

Let us now see that the defender's winning region of $\mathcal{G}_{CS}$ indeed corresponds to $\sqsubseteq_{CS}$. To this end, we first show how to construct winning strategies for the defender from a coupled simulation, and then establish the opposite direction.

Lemma 3.27. *Let $\mathcal{R}$ be a coupled simulation and $(p_0, q_0) \in \mathcal{R}$. Then the defender wins $\mathcal{G}_{CS}[(p_0, q_0)_{\mathtt{a}}]$ with the following positional strategy:*

- *If the current play fragment ends in a simulation defender node $(\alpha, p', q)_{\mathtt{d}}$, move to some attacker node $(p', q')_{\mathtt{a}}$ with $(p', q') \in \mathcal{R}$ and $q \overset{\hat{\alpha}}{\Rightarrow} q'$;*
- *if the current play fragment ends in a coupling defender node $(\mathtt{Cpl}, p, q)_{\mathtt{d}}$, move to some attacker node $(q', p)_{\mathtt{a}}$ with $(q', p) \in \mathcal{R}$ and $q \Rightarrow q'$.*[31]

Proof. Following the specified strategy, the game can only reach attacker nodes $(p, q)_{\mathtt{a}}$ where $(p, q) \in \mathcal{R}$.[32] Therefore, and because $\mathcal{R}$ is a coupled simulation, the attacker can only lead the defender to positions where the moves of the strategy are allowed[33] and the "some"-formulations really specify moves. This can be used to name a next move for every play fragment ending in a defender node. Consequently, the defender does not get stuck and wins. □

Lemma 3.28. *Let f be a winning strategy for the defender in $\mathcal{G}_{CS}[(p_0, q_0)_{\mathtt{a}}]$. Then*

$$\mathcal{R} = \{(p, q) \mid \textit{some } \mathcal{G}_{CS}[(p_0, q_0)_{\mathtt{a}}]\textit{-play fragment consistent with } f \textit{ ends in } (p, q)_{\mathtt{a}}\}$$

is a coupled simulation.[34]

Proof. The proof relies on the fact, that only $(p, q)_{\mathtt{a}}$ from where the defender knows how to win (using f) can be considered when checking the coupled simulation property of $\mathcal{R}$.

If we challenge the simulation property, that is, demand a q' such that $q \overset{\hat{\alpha}}{\Rightarrow} q'$ and $(p', q') \in \mathcal{R}$ given that $p \xrightarrow{\alpha} p'$, then we know that there is a simulation challenge game move $(p, q)_{\mathtt{a}} \rightarrowtail (\alpha, p', q)_{\mathtt{d}}$. Because f is winning, it must know a simulation answer move $(\alpha, p', q)_{\mathtt{d}} \rightarrowtail (p', q')_{\mathtt{a}}$ and this move gives us the q' we are looking for due to the definitions of $\mathcal{G}_{CS}$ and $\mathcal{R}$.

Analogously, if we challenge coupling, that is, look for a q' such that $q \Rightarrow q'$ and $(q', p) \in \mathcal{R}$, then we know there is a coupling challenge game move $(p, q)_{\mathtt{a}} \rightarrowtail (\mathtt{Cpl}, p, q)_{\mathtt{d}}$. Because f is winning, it must know a coupling answer move $(\mathtt{Cpl}, p, q)_{\mathtt{d}} \rightarrowtail (q', p)_{\mathtt{a}}$ and this move again gives us the desired q'. □

With minimal fiddling, this completes our game-theoretic characterization of the coupled simulation preorder.

[31] lemma `CS_Game.coupledsim_implies_winning_strategy`
[32] lemma `CS_Game.strategy_from_coupledsim_retains_coupledsim`
[33] lemma `CS_Game.strategy_from_coupledsim_sound`
[34] lemma `CS_Game.winning_strategy_implies_coupledsim`

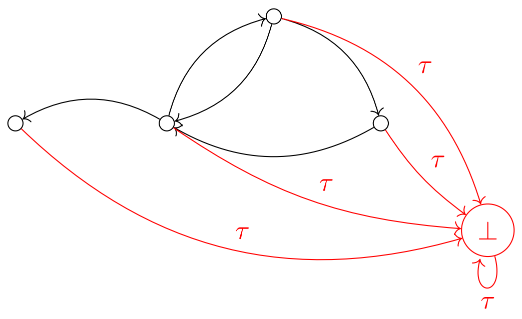


Figure 3.3: Example for $\mathcal{S}^{\perp}$ from Theorem 3.30 ($\mathcal{S}$ in black, $\mathcal{S}^{\perp} \backslash \mathcal{S}$ in red).

Theorem 3.29. *The defender wins $\mathcal{G}_{CS}[(p,q)_{\mathtt{a}}]$ precisely if $p \sqsubseteq_{CS} q$.*[35]

The game theoretic characterization is the basis for our main algorithm in Section 4.4 and its parallel implementation in Chapter 5.

3.5 How Hard Can It Be?

We conclude this chapter with a few thoughts on how the properties of coupled similarity make it compare to related notions of equivalence complexity-wise. The general rule of thumb for notions of equivalence in between strong bisimilarity and trace equivalence is that *deciding the coarser ones is harder than deciding the finer ones.* This holds true for coupled similarity.

3.5.1 Reduction of Weak Simulation to Coupled Simulation

It is well-established that deciding variants of similarity is at least as hard as deciding corresponding variants of bisimilarity, and actually tends to be harder [KM02]. So, as coupled similarity lies in between weak bisimilarity and weak similarity, the question suggests itself whether coupled similarity rather inherits the complexity of weak similarity or bisimilarity. The best known algorithms for weak similarity must also construct the weak simulation preorder. Their running time is cubic in the state space size, whereas sub-cubic weak bisimilarity algorithms exist.

Unfortunately, deciding the weak simulation preorder can be reduced to deciding the coupled simulation preorder.

Theorem 3.30. *Every decision algorithm for $\sqsubseteq_{CS}^{\mathcal{S}}$ with running time $\mathcal{O}(g(t))$ can be used to decide $\sqsubseteq_{WS}^{\mathcal{S}}$ in running time $\mathcal{O}(g(2t))$, where $t = |{\rightarrow}|$ is the number of transitions in system $\mathcal{S}$ and g is some strictly increasing function.*

Proof. Let $\mathcal{S} = (S, \Sigma_\tau, \rightarrow)$ be an arbitrary transition system and $\perp \notin S$. Then

$$\mathcal{S}^{\perp} := \Big(S \cup \{\perp\}, \quad \Sigma_\tau, \quad \dot{\rightarrow} \cup \{(p, \tau, \perp) \mid p \in S \cup \{\perp\}\} \Big)$$

extends $\mathcal{S}$ with a sink $\perp$ that can be reached by a τ-step from everywhere. (For an illustration see Figure 3.3.) Note that $\sqsubseteq_{WS}^{\mathcal{S}}$ and $\sqsubseteq_{WS}^{\mathcal{S}^{\perp}}$ resemble each other; that is, for $p, q \neq \perp$, $p \sqsubseteq_{WS}^{\mathcal{S}} q$ exactly if $p \sqsubseteq_{WS}^{\mathcal{S}^{\perp}} q$.[36] On $\mathcal{S}^{\perp}$, coupled simulation preorder and

[35] theorem `CS_Game.winning_strategy_iff_coupledsim`
[36] lemma `Weak_Relations.simulation_sink_invariant`

weak simulation preorder coincide, $\sqsubseteq^{\mathcal{S}^\perp}_{WS} = \sqsubseteq^{\mathcal{S}^\perp}_{CS}$, because $\perp$ is τ-reachable everywhere, and, for each p, $\perp \sqsubseteq^{\mathcal{S}^\perp}_{CS} p$ discharges the coupling constraint of coupled simulation.[37]

Because $\sqsubseteq^{\mathcal{S}}_{WS}$ can be decided by deciding $\sqsubseteq^{\mathcal{S}^\perp}_{CS}$, a decision procedure for $\sqsubseteq_{CS}$ also induces a decision procedure for $\sqsubseteq_{WS}$. The transformation increases the amount of transitions by a factor of at most 2 (it adds $\mathcal{O}(|S|) \subseteq \mathcal{O}(t)$ new transitions) and can itself be done in t time, so that the resulting algorithm runs in $\mathcal{O}(t + g(2t)) = \mathcal{O}(g(2t))$ time. $\square$

As $\mathcal{O}(g(2t)) = \mathcal{O}(g(t))$ in the polynomial world of weak simulation, deciding $\sqsubseteq_{CS}$ cannot have a lower complexity class than deciding $\sqsubseteq_{WS}$. The best known algorithms for strong simulation $\sqsubseteq_S$ without transition labeling [HHK95, RT08] run in $\mathcal{O}(|S|\,|{\rightarrow}|)$ time. Of these Ranzato and Tapparo's algorithm [RT10] has the upside that it actually is bounded in terms of the number of simulation equivalence classes $|S_{/\equiv_S}|$ and not the full size of S, that is, by $\mathcal{O}(|S_{/\equiv_S}|\,|{\rightarrow}|)$. They advocate the use of the method by Dovier et al. [DPP04] to transform transition-labeled systems to state-labeled ones. This method consists in replacing every labeled transition $p \xrightarrow{\alpha} p'$ by an α-labeled state $p\alpha p'$ and two unlabeled transitions $p \rightarrow p\alpha p' \rightarrow p'$. For a naive reduction from weak simulation to strong simulation, this would amount to a significantly higher complexity of $\mathcal{O}(|\overset{\hat{\cdot}}{\Rightarrow}|^2)$, because each weak step would induce its own new state. However, all the $|\overset{\hat{\cdot}}{\Rightarrow}|$ new states can be merged to $\mathcal{O}(|\Sigma_\tau|\,|S|)$ states with out-degree 1. Thus, we can confidently conjecture that implementations of [RT10] specialized for weak systems would have time bounds of $\mathcal{O}(|S_{/\equiv_{WS}}|\,|\overset{\hat{\cdot}}{\Rightarrow}|)$.

3.5.2 The Costs of Weakness

We already observed that coarseness comes at a price in the linear-time branching-time spectrum. This is also true for the *weak transition relation* $\overset{\hat{\cdot}}{\Rightarrow}$, which plays an important role in the definition of coupled simulation.

In theory, both the real weak transition relation $\dot{\Rightarrow} \; = \; \Rightarrow\dot{\rightarrow}\Rightarrow$ and the conventional transition relation $\dot{\rightarrow}$ are bounded in size by $|\Sigma_\tau|\,|S|^2$. But for most transition systems, $\dot{\Rightarrow}$ tends to be much bigger in size.

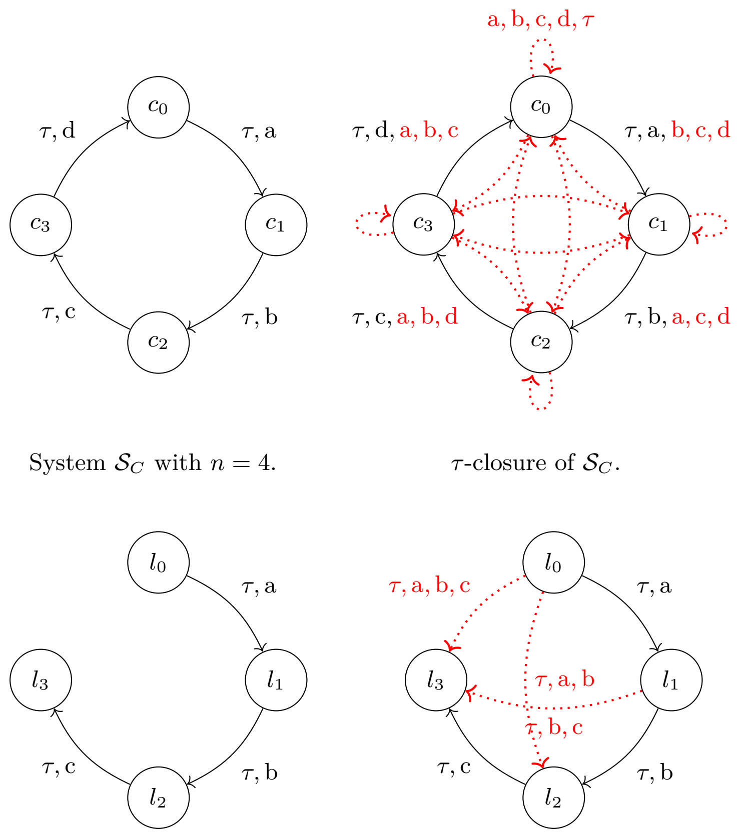
Example 3.31. Consider cyclic systems like $\mathcal{S}_C$ and linear systems like $\mathcal{S}_L$ from Figure 3.4, where $|\Sigma| \leq |S|$.[38] The systems are given on the left-hand side whereas the right-hand side displays the weak step closure $\dot{\Rightarrow}$ of the systems. The difference is highlighted in red. For readability, some labels of Σ_τ-labeled edges are omitted.

- System $\mathcal{S}_C$ contains a cycle of τ-steps. Consequently, the transitive τ-closure $\Rightarrow$ relates all states and $\dot{\Rightarrow} \; = S \times \Sigma_\tau \times S$. For cyclic systems like $\mathcal{S}_C$, the weak step relation is of cubic size in the number of original transitions, that is, $|\dot{\Rightarrow}| = n^2(n+1) \in \mathcal{O}(n^3)$ where $n = \frac{|\dot{\rightarrow}|}{2} = |S|$.

- System $\mathcal{S}_L$ is linearized by τ-steps. Consequently, $\Rightarrow$ is a total order and the weak step relation prepones all the steps of "greater" states. For linear systems like $\mathcal{S}_L$, the weak step relation is not as dense as for systems like $\mathcal{S}_C$ but still of cubic size $|\dot{\Rightarrow}| = \sum\limits_{k \in \{1..n\}} \frac{(k+2)(k-1)}{2} \in \mathcal{O}(n^3)$ where $n = \frac{|\dot{\rightarrow}|+2}{2} = |S|$.

[37] lemma `Coupled_Simulation.tau_sink_sim_coupledsim`

[38] Also available at `https://coupledsim.bbisping.de/#cyclic-linear`.



System $\mathcal{S}_C$ with $n = 4$. τ-closure of $\mathcal{S}_C$.

System $\mathcal{S}_L$ with $n = 4$. τ-closure of $\mathcal{S}_L$.

Figure 3.4: Worst-case transition systems for the size of $\dot{\Rightarrow}$.

So, quite sparse $\dot{\rightarrow}$-graphs can generate dense $\dot{\Rightarrow}$-graphs.

Moreover, the computation of the transitive closure also has significant time complexity. Algorithms for transitive closures on graphs (V, E) usually are in $\mathcal{O}(|V|\,|E|)$.[39] This means computing $\Rightarrow$ would already be in $\mathcal{O}(|S|\,|\xrightarrow{\tau}|)$, thus cubic in the size of the state space.

One might argue that the size of weakly τ-connected components in big state spaces often is bounded by a lower parameter than $|\xrightarrow{\tau}|$, so that the $\mathcal{O}(|S|\,|\xrightarrow{\tau}|)$-complexity is not actualized. Still, the general theoretical complexity of the whole weak equivalence algorithm cannot be much better than cubic if the transition closure is computed.

So, aside from the complexity inherited from simulation, coupled simulation additionally is affected by the inherent complexity of the weak transition relation. Consequently, one of the questions in the next chapter is how to keep the weak transition blow-up at bay.

[39]The theoretical bound is somewhat lower, since the problem can be reduced to matrix multiplication, for which the best algorithm currently known (Le Gall) runs in $\mathcal{O}(|V|^{2.372864})$ time.

Chapter 4

Algorithms

The previous chapters have laid the groundwork for three approaches towards computing coupled similarity. In this chapter, we turn the axiomatization of $\equiv_{CS}$ from [PS94] into a reduction algorithm, develop a fixed-point algorithm from the $\sqsubseteq_{CS}$-definition of [vG17], and adapt our game characterization to decide $\sqsubseteq_{CS}$.

4.1 Setting the Stage

First, it is necessary to explicate some assumptions about the setting our coupled similarity algorithms are expecting.

Obviously, in finite time, they can only deal with a finite portion of a transition system. So, it does not hurt to assume that the systems are restricted to be finite and free of disconnected junk states and actions.

Assumption 4.1. *The transition systems $\mathcal{S} = (S, \Sigma_\tau, \rightarrow)$ we are working on are finite (meaning S and Σ_τ are finite). Their data structures use linear space and can be traversed in linear time. Σ and S only contain actions and states appearing in $\rightarrow$.*

We should make two more assumptions concerning the context of *minimizations* in which our coupled simulation algorithms are applied.

4.1.1 Dealing with the τ-Closure

A lot of headaches can be avoided by ruling out τ-cycles right from the start.

Assumption 4.2. *The input transition systems of our core algorithm are τ-cycle-free.*

This is no hack because the cycles can actually be removed quite easily. First, they can be conflated to τ-loops because strongly τ-connected states are coupled similar (Corollary 3.17). This is accomplished by Tarjan's $\mathcal{O}(|\xrightarrow{\tau}|)$-algorithm for strongly connected components in directed graphs [Tar72]. An implementation can be found in `de.bbisping.coupledsim.algo.transform.TauLoopCompression`. Second, because τ-loops are invisible to coupled simulation by Lemma 3.19, we can remove or ignore the remaining τ-loops.

(Sub-)systems like the cyclic $\mathcal{S}_C$ from Example 3.31 then collapse to single states and thus lose their space complexity. If the system has no τ-cycles, for example, as it is divergence-free like the linear $\mathcal{S}_L$, the compression will have no effect. Either way, after the compression, we can be sure that the τ-edges form a directed acyclic graph (DAG), which algorithms can handle more efficiently.

At this point, it even is questionable whether to explicitly build the τ-closure $\overset{\hat{\tau}}{\Rightarrow}$, which would otherwise seem a reasonable preparation for weak equivalence algorithms. But the τ-DAG already is a good data structure to walk for the equivalence algorithms.[1] Indeed, sub-cubic time complexities for weak bisimulation [RT08, Li09, BGRR16], branching bisimulation [GJKW17], and other weak equivalences can be achieved by abstaining from the computation of the transitive closure. The developers of the CSP refinement checker *FDR4* present a few more thoughts on this in [BGRR16], and we will run into the question again in Subsection 4.5.4.

4.1.2 Minimization

The τ-cycle compression actually is only an instance of a more general approach in equivalence algorithms, namely to minimize the input system with finer notions of equivalence. Minimization means that one uses an equivalence relation $\equiv_X$ and builds the quotient system:

Definition 4.3 (Quotient system)**.** Given an equivalence relation $\equiv_X$ and a transition system $\mathcal{S} = (S, \Sigma_\tau, \rightarrow)$, the *quotient system* is defined as $\mathcal{S}_{/\equiv_X} := (S_{/\equiv_X}, \Sigma_\tau, \rightarrow_{/\equiv_X})$ where $S_{/\equiv_X}$ is the set of equivalence classes, and

$$p \xrightarrow{\alpha}_{/\equiv_X} q \text{ for } p, q \in S_{/\equiv_X} \text{ iff there are } p_0 \in p, q_0 \in q \text{ such that } p_0 \xrightarrow{\alpha} q_0.$$

Instead of $S_{/\equiv_X}$, it might often be more convenient to take the set of equivalence class representatives. A straight-forward algorithm for this transformation is implemented in `de.bbisping.coupledsim.algo.transform.BuildQuotientSystem`.

Our last—in a way: meta—assumption is that our algorithms run in the context of such minimizations:

Assumption 4.4. *The result of our $\sqsubseteq_{CS}$-algorithms should be usable to minimize systems and to tell whether two certain states are equivalent.*

Minimized systems are more manageable for proofs and presentations. For example, van Glabbeek and Weijland [vGW96] report that Peterson's mutual exclusion algorithm minimized by coupled similarity has only nine states as opposed to fourteen states in weak bisimulation semantics. The quotient systems have the nice property that within them their equivalences equal the identity relation. Together with some normalization procedure, this can be used to generate unique-up-to-isomorphism representatives for equivalent transition systems. A lot of details have to be taken care of if one pursues fully minimal unique representatives [EHS+13].

Moreover, minimizations can be used to speed up algorithms. For instance, if two states are strongly bisimilar, then they also are coupled similar. So, no coupled simulation algorithm will tell them apart. Consequently, they can be represented by a single state right from the beginning. As finer equivalence algorithms tend to have better space and time complexities, this can significantly push the boundary of what system sizes can be tackled by an implementation.

[1] One might argue that loop compression somewhat is a part of the transitive closure problem. For example, it also is the first step of Boost's widely used transitive closure implementation. `http://www.boost.org/doc/libs/1_66_0/libs/graph/doc/transitive_closure.html`

4.2 The Bisimulation Approach

A common approach for notions of equivalence, is to construct algorithms that solve the equivalence problem by *reducing it to strong bisimilarity.* Bisimilarity, then, can be decided by tried and tested algorithms. For instance, the PAIGE–TARJAN algorithm [PT87] computes bisimilarity relations in $\mathcal{O}(|S| \log |{\rightarrow}|)$ time and with little space overhead.

Such a reduction also is possible for coupled similarity building on PARROW and SJÖDIN'S axiomatization from Section 3.3. But, things might get messy.

The normalization algorithm quickly turns out to be intractable, which is why this section remains sketchy around the fine points of implementation and correctness of the algorithm.

4.2.1 The Reduction

PARROW and SJÖDIN'S [PS94] completeness proof for their axiomatization of $\equiv_{CS}$ relies on normalizing process terms. As they point out, this normalization induces a bisimulation-reduction-based coupled similarity algorithm. Their normalization can $\equiv_{CS}$-soundly rewrite finite (recursion-free) CCS process terms such that $\equiv_{CS}$ and $\equiv_{WB}$ coincide.

There are three gaps that are to be filled for our setting: The normal forms must be lifted to transition systems; the normalization must be described algorithmitically; and the problem of divergent τ-cycles must be addressed.

Lifting Definition 3.24 of *CS*-normal forms from CCS processes to transition systems, we get:

Definition 4.5 (CS-normal TS)**.** A transition system $\mathcal{S} = (S, \rightarrow, \Sigma_\tau)$ is in *CS-normal form* iff

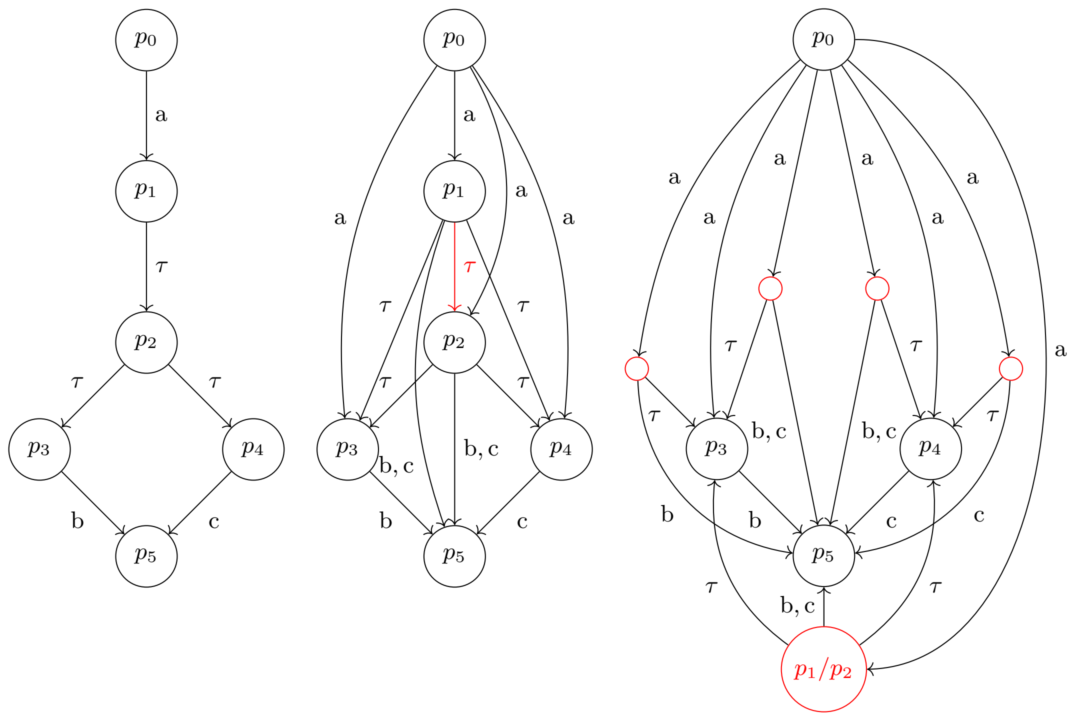
1. $p \overset{\alpha}{\Rightarrow} p'$ implies $p \overset{\alpha}{\rightarrow} p'$,
2. $p \overset{\tau}{\rightarrow} p'$ implies p' stable, and
3. if $p \overset{\alpha}{\rightarrow} p'$ and p' instable, then for every subset of p'-transitions, $\hat{P} \subseteq \{(\beta, p'') \mid p' \overset{\beta}{\rightarrow} p''\}$, containing at least one τ-transition, there exists $\hat{p}$ such that $p \overset{\alpha}{\rightarrow} \equiv_{CS^c}^{\mathcal{S}'} \hat{p}$, where $\mathcal{S}' \coloneqq \Big(S \cup \{\hat{p}\},\ \Sigma_\tau,\ \rightarrow \cup\, (\{\hat{p}\} \times \hat{P})\Big)$.

Conditions 1, 2, and 3 correspond to clauses 3, 4, and 5 from Definition 3.24. There, we already observed that the last condition ensures saturation of the system with all partial commits that could be hidden in the trailing $\Rightarrow$-part of a weak visible transition. In order to see what this means, let us consider the following example from [PS94].

Example 4.6. Figure 4.1 gives a system, its $\dot{\Rightarrow}$-closed version, and its minimal CS-normal form with respect to Definition 4.5.

The $\dot{\Rightarrow}$-closed system already suffers from the blow-up in transitions discussed in Subsection 3.5.2. To make it obey condition 2 from Definition 4.5, it suffices to remove the τ-step between the two instable states p_1 and p_2 (highlighted in red).

However, condition 3 then demands all partial choices that could have been skipped between p_1 and $\{p_3, p_4\}$ by $p_0 \overset{a}{\Rightarrow}$-steps to be present. So, p_1 and p_2 are shuffled into



Original system. $\dot{\Rightarrow}$-closed system. CS-normal form.

Figure 4.1: CS-normalization from [PS94].

the five red states in the right-hand side system. For instance, the left-most red state expresses the partial commit that the a-step has already committed not to perform c but still can prepend the b-step with τ. The result would be even bigger if it did not exploit the fact that p_1 and p_2 are bisimilar after the computation of the $\dot{\Rightarrow}$-closure and the removal of instable τ-steps.

Such normal forms can actually be computed. Lemma 18 of [PS94] gives an existence proof, which motivates the algorithm of this section. Lifting their lemma to transition systems, we get:

Lemma 4.7. *([PS94]) For every finite $\mathcal{S} = (S, \Sigma_\tau, \rightarrow)$ with acyclic $\rightarrow$, there is a finite $\mathcal{S}'$ in CS-normal form such that, for each $p, q \in S$, $p \equiv^{\mathcal{S}}_{CS} q$ precisely if $p \equiv^{\mathcal{S}'}_{CS} q$.*

For $\mathcal{S}'$ in normal form, $p \equiv^{\mathcal{S}'}_{CS} q$ precisely if $p \equiv^{\mathcal{S}'}_{WB} q$.

Like Parrow and Sjödin [PS94], we assume that this carries over to general finite transition systems. This assumption is based on the impression that the main trouble arises from τ-cycles. But by Assumption 4.2, these have been erased as described in Subsection 4.1.1.

4.2.2 The Algorithm

The core of the reduction approach outlined in Algorithm 1 is the normalization procedure in construct_cs_normalization, the implementation of which can be found in `de.bbisping.coupledsim.algo.transform.PS94CSNormalForms`.

Line 2 builds the weak transition closure with the restriction that τ-steps may only reach stable states. This corresponds to the first two requirements of Definition 4.5.

```
1  def construct_cs_normalization($\mathcal{S} = (S, \Sigma_\tau, \to)$):
2      $\rightsquigarrow := \{(p, \alpha, p') \mid p \overset{\alpha}{\Rightarrow} p' \land (\alpha = \tau \longrightarrow p' \overset{\tau}{\not\to})\}$
3      $\twoheadrightarrow := \{(p, \alpha, P) \mid$
4          $\exists p'.\, p \overset{\alpha}{\rightsquigarrow} p' \land \neg p' \overset{\tau}{\not\to}$
5          $\land\, P \subseteq \{(\beta, p'') \mid p' \overset{\beta}{\rightsquigarrow} p''\}$
6          $\land\, \exists p''.\, (\tau, p'') \in P\}$
7      $S_P := \{P \mid \cdot \overset{\cdot}{\twoheadrightarrow} P \neq \emptyset\}$
8      $\to' := \rightsquigarrow \cup \twoheadrightarrow \cup \{(P, \beta, p'') \mid P \in S_P \land (\beta, p'') \in P\}$
9      return $(S \cup S_P, \Sigma_\tau, \to')$
10 def compute_cs_by_normalization($\mathcal{S} = (S, \Sigma_\tau, \to)$):
11     $\mathcal{S}_{\text{old}} := \bot$
12     while $\mathcal{S}_{\text{old}} \neq \mathcal{S}$:
13         $\mathcal{S}_{\text{old}} := \mathcal{S}$
14         $\mathcal{S} :=$ construct_cs_normalization($\mathcal{S}$)
15     $\mathcal{R} :=$ strong_bisim(tau_refl($\mathcal{S}$))
16     return $\mathcal{R} \cap (S \times S)$
```

Algorithm 1 : Bisimulation-reduction-based algorithm for coupled similarity $\equiv_{CS}$.

Lines 3 to 7 cover the third requirement: the presence of partial commits. The new partial commit states are represented by the sets of their outgoing edges because two partial commits that lead to the same set of remaining moves can be considered equivalent. τ-loops might cause trouble here, but we do not bother with them in line with Assumption 4.2.

The newly added partial commit states do not necessarily fulfill requirement 1 and 3. Some of the implied steps could be missing for them and they might not be closed under partial commits. However, this can be dealt with by repeating construct_cs_normalization until a fixed point[2] is reached. This is what is happening in lines 12ff. of compute_cs_by_normalization.

On the resulting $\mathcal{S}$, $p \equiv^{\mathcal{S}}_{CS} q$ if and only if $p \equiv^{\mathcal{S}}_{WB} q$ by Lemma 4.7. The clauses it fulfills are a superset of the demands of the normalization for weak bisimilarity from Subsection 2.3.2. So, everything one needs to complete the reduction to strong bisimilarity is the reflexive closure on τ-steps of the transition relation. This is done in line 15. Afterwards, the equivalences determined for the initial states carry over to the original system, which is why $\mathcal{R} \cap (S \times S)$ is returned. Note that this rests on the fact that no on-the-fly merging (as in Example 4.6) of states is happening.

4.2.3 Complexity

The reduction Algorithm 1 is exponential in time and space. The target states of $\twoheadrightarrow$ are constructed from an only slightly filtered power set of outgoing weak transitions (line 8), the size of which is exponential in the original outgoing transition count.

The complexity is bounded by the branching degree of the special weak transition

[2]A fixed point x of a function $f\colon A \to A$ is a value where $x = f(x)$. The next pages repeatedly draw on this concept. An introduction to fixed points can be found in Nielson, Nielson, and Hankin's "Principles of Program Analysis" [NNH15].

```
def fp_step$_{(S,\Sigma_\tau,\rightarrow)}(\mathcal{R})$:
    return $\{(p,q) \in \mathcal{R} \mid$
        $(\forall p', \alpha.\ p \xrightarrow{\alpha} p' \longrightarrow \exists q'.\, (p',q') \in \mathcal{R} \land q \overset{\hat{\alpha}}{\Rightarrow} q')$
        $\land\ (\exists q'.\, q \Rightarrow q' \land (q',p) \in \mathcal{R})\}$
def fp_compute_cs($\mathcal{S} = (S, \Sigma_\tau, \rightarrow)$):
    $\mathcal{R} := S \times S$
    while fp_step$_\mathcal{S}(\mathcal{R}) \neq \mathcal{R}$:
        $\mathcal{R}$ := fp_step$_\mathcal{S}(\mathcal{R})$
    return $\mathcal{R}$
```
Algorithm 2 : Fixed-point algorithm for the coupled simulation preorder $\sqsubseteq_{CS}$.

relation $\rightsquigarrow$. In practice, bounds like this one often mean that the exponentiality of an algorithm remains of limited impact. For instance, the reduction of weak trace equivalence to strong bisimilarity from [CH93] is exponential in theory but usable in practice. For the present algorithm however, the complexity indeed makes it practically intractable because $\rightsquigarrow$ inherits the boosted branching degree of $\dot{\Rightarrow}$.

Even if one combined the algorithm with some on-the-fly minimization techniques, an exponential amount of additional states and transitions would still be necessary.

Its intractability is the reason why we do not focus on refining the algorithm and proving its correctness. Instead, we continue with other algorithms right away.

This section has illustrated that the reduction of coupled similarity to weak/strong bisimilarity as hinted at by [PS94] is possible, but indeed much more expensive than the polynomial τ-closure-based reductions of notions in between weak and strong bisimilarity. Luckily, there are less expensive approaches.

4.3 The Fixed-Point Approach

The coinductive characterization of $\sqsubseteq_{CS}$ in Lemma 3.12 induces an extremely simple polynomial-time algorithm to compute the coupled simulation preorder as a *greatest fixed point*. This section introduces the algorithm and proves its correctness.

4.3.1 The Algorithm

Roughly speaking, the algorithm first considers the universal relation between states, $S \times S$, and then proceeds by removing every pair of states from the relation that would contradict the coupling or the simulation property. Its pseudo code is depicted in Algorithm 2. An implementation can be found in `de.bbisping.coupledsim.algo.cs.FixedPointCoupledSimilarity`.

fp_step plays the role of removing the tuples that would immediately violate the simulation or coupling property from the relation. Of course, such a pruning might invalidate tuples that were not rejected before. Therefore, fp_compute_cs repeats the process until fp_step$_\mathcal{S}(\mathcal{R}) = \mathcal{R}$, that is, until $\mathcal{R}$ is a fixed point of fp_step$_\mathcal{S}$.

4.3.2 Correctness

It is quite straight-forward to show that Algorithm 2 indeed computes $\sqsubseteq_{CS}$ because of the resemblance between fp_step and the coupled simulation property itself, and because of the monotonicity of fp_step.

Lemma 4.8. *If* $\mathsf{fp\_step}_{\mathcal{S}}(\mathcal{R}) = \mathcal{R}$*, then* $\mathcal{R}$ *is a coupled simulation.*[3]

So, we can be sure that our algorithm, if it terminates, finds some coupled simulation. But is it the $\sqsubseteq_{CS}$-relation—which is to say the *greatest* one? First, let us note that fp_step is monotone on the subset lattice over $2^{S \times S}$.

Lemma 4.9. $\mathcal{R}_1 \subseteq \mathcal{R}_2$ *implies* $\mathsf{fp\_step}_{\mathcal{S}}(\mathcal{R}_1) \subseteq \mathsf{fp\_step}_{\mathcal{S}}(\mathcal{R}_2)$.[4]

This guarantees the existence of a greatest fixed point by the Knaster–Tarski theorem. This greatest fixed point coincides with $\sqsubseteq_{CS}$.

Lemma 4.10. *If* $\mathcal{R}$ *is the greatest fixed point of* fp_step*, then* $\mathcal{R} = \sqsubseteq_{CS}$.[5]

On finite labeled transition systems, that is, with finite S and $\rightarrow$, the while loop of fp_compute_cs is guaranteed to terminate at the greatest fixed point of fp_step (by a dual variant of the Kleene fixed-point theorem).

Lemma 4.11. *For finite transition systems* $\mathcal{S}$*,* $\mathsf{fp\_compute\_cs}(\mathcal{S})$ *computes the greatest fixed point of* $\mathsf{fp\_step}_{\mathcal{S}}$.[6]

Wrapping up the previous two lemmas, we receive that fp_compute_cs is correct.

Theorem 4.12. *For finite transition systems* $\mathcal{S}$*,* $\mathsf{fp\_compute\_cs}(\mathcal{S})$ *returns* $\sqsubseteq_{CS}^{\mathcal{S}}$.[7]

The proof is intuitively and machine-verifiably correct. Because of its simplicity, we can trust implementations of Algorithm 2 to faithfully return sound and complete $\sqsubseteq_{CS}$-relations. Therefore, we can use this algorithm to generate reliable results within test suites for the behavior of other $\sqsubseteq_{CS}$-implementations.

4.3.3 Complexity

The space complexity, given by the maximal size of $\mathcal{R}$, clearly is in $\mathcal{O}(|S|^2)$. Time complexity takes some inspection of the algorithm. For our considerations, we assume that $\hat{\Rightarrow}$ has been pre-computed, which can slightly increase the space complexity to $\mathcal{O}(|\Sigma|\,|S|^2)$.

Lemma 4.13. *The running time of* fp_compute_cs *is in* $\mathcal{O}(|\Sigma|\,|S|^6)$.

Proof. Checking the simulation property for a tuple $(p, q) \in \mathcal{R}$ means that for all $\mathcal{O}(|\Sigma|\,|S|)$ outgoing $p\!\rightarrow$-transitions, each has to be matched by a $q\hat{\Rightarrow}$-transition with identical action, of which there are at most $|S|$. So, simulation checking costs $\mathcal{O}(|\Sigma|\,|S|^2)$ time per tuple. Checking the coupling can be approximated by $\mathcal{O}(|S|)$ per tuple. Simulation dominates coupling. The amount of tuples that have to be checked is in $\mathcal{O}(|S|^2)$. Thus, the overall complexity of one invocation of fp_step is in $\mathcal{O}(|\Sigma|\,|S|^4)$.

[3] lemma `CS_Fixpoint_Algo.fp_fp_step`
[4] lemma `CS_Fixpoint_Algo.mono_fp_step`
[5] lemma `CS_Fixpoint_Algo.gfp_fp_step_gcs`
[6] lemma `CS_Fixpoint_Algo.gfp_fp_step_while`
[7] lemma `CS_Fixpoint_Algo.coupled_sim_fp_step_while`

Because every invocation of fp_step decreases the size of $\mathcal{R}$ or leads to termination, there can be at most $\mathcal{O}(|S|^2)$ invocations of fp_step in fp_compute_cs. Checking whether fp_step changes $\mathcal{R}$ can be done without notable overhead. In conclusion, we arrive at an overall time complexity of $\mathcal{O}(|\Sigma|\,|S|^6)$. $\square$

In Subsection 3.5.1, we already noticed that computing $\sqsubseteq_{CS}$ is at least as complex as computing $\sqsubseteq_{WS}$. The observation now, that ensuring the coupling condition costs less than ensuring simulation, increases hope that computing $\sqsubseteq_{CS}$ is not more complex than computing $\sqsubseteq_{WS}$.

Lemma 4.13 seems not to be extra-ordinarily noteworthy. Still, in combination with Theorem 4.12, it already proves that computing coupled similarity is in **P** and thus less complex than computing trace equivalence, which is **PSPACE**-complete [CH93]. Parrow and Sjödin [PS94] while hinting at the exponential algorithm from Section 4.2 formulated as an open research question whether $\equiv_{CS}$ can be decided in **P**, and—to the author's knowledge—we are hereby explicitly answering the question for the first time. Given that deciding the equivalences surrounding $\equiv_{CS}$ in the linear-time branching-time spectrum is in **P**, a different result for $\equiv_{CS}$ would have been extremely surprising.

Now, it does not take much energy to spot that applying the filtering in fp_step to each and every tuple in $\mathcal{R}$ in every step, would not be necessary. Only after a tuple (p, q) has been removed from $\mathcal{R}$, the algorithm does really need to find out whether this was the last witness for the $\exists$-quantification in the clause of another tuple. While this observation could inspire various improvements, let us fast-forward to the game-theoretic approach in the next section, which elegantly explicates the witness structure of a coupled similarity problem.

4.4 The Game-Theoretic Approach

Section 3.4 introduced a simple two-player game characterization of the coupled simulation preorder and proved its correctness. Deciding winning regions of finite simple games is in linear running time. Going down this route, the game approach induces a nice and reliable way of computing the coupled simulation preorder $\sqsubseteq_{CS}$ with running time cubic in the count of transition system states.

4.4.1 Deciding the Coupled Simulation Game

It is well-known that the winning regions of finite simple games can be computed in linear time. Variants of the standard algorithm for this task can be found in [Grä07] and [Kre16]. A slightly simplified[8] version of it is given in Algorithm 3 and implemented in the Scala trait `de.bbisping.coupledsim.game.WinningRegionComputation`. Intuitively, the algorithm first assumes that the defender wins everywhere and then sets off a chain reaction beginning in defender deadlock nodes, which "turns" all the nodes won by the attacker.

The algorithm compute_winning_region proceeds as follows: Initially, it determines

[8]The difference is that Grädel's formulation is for well-founded games and might consider situations with cycles undecided. Interestingly, matters actually become *simpler* if infinite plays are considered wins for the defender: We can just unfold the winning region of the attacker and consider every other position a win for the defender due to the determinacy property from Proposition 2.29.

```
 1 def compute_winning_region($\mathcal{G} = (G, G_a, \rightarrowtail)$):
 2     predecessors := $[g \mapsto (\cdot \rightarrowtail g) \mid g \in G]$
 3     num := $[g \mapsto n \mid g \in G \wedge n = \|\{p \mid g \rightarrowtail p\}\|]$
 4     win := $[g \mapsto \mathtt{d} \mid g \in G]$
 5     def propagate_attacker_win($g$):
 6         if win$[g] = \mathtt{d}$ :
 7             win$[g]$ := $\mathtt{a}$
 8             for $g_p \in$ predecessors($g$):
 9                 num$[g_p]$ := num$[g_p] - 1$
10                 if $g_p \in G_a \vee$ num$[g_p] = 0$ :
11                     propagate_attacker_win($g_p$)
12     for $g \in G_d$:
13         if num$[g] = 0$ :
14             propagate_attacker_win($g$)
15     return win
```

Algorithm 3 : Algorithm for deciding simple games.

```
1 def game_compute_cs($\mathcal{S}$):
2     $\mathcal{G}^{\mathcal{S}}_{CS} = (G, G_a, \rightarrowtail)$ := obtain_cs_game($\mathcal{S}$)
3     win := compute_winning_region($\mathcal{G}^{\mathcal{S}}_{CS}$)
4     $\mathcal{R}$ := $\{(p, q) \mid (p, q)_\mathtt{a} \in G_a \wedge$ win$[(p, q)_\mathtt{a}] = \mathtt{d}\}$
5     return $\mathcal{R}$
```

Algorithm 4 : Game algorithm for the coupled simulation preorder $\sqsubseteq_{CS}$.

the predecessors (line 2) and the number of successors (line 3, num) for each node. It first assumes that every node is won by the defender d (line 4, win). Consequently, num also equals the number of assumed winning defender moves and this will be the core invariant. The defender loses nodes where they have no move to a winning node. So, the recursion starts at defender nodes without winning moves (line 12). propagate_attacker_win (line 5) is called every time when it is discovered that a node is actually won by the attacker. Then the attacker also wins the predecessors of the node they own and propagate_attacker_win is recursively called on these nodes in line 11. For each predecessor, num is decreased by one because it loses the current node as a winning defender move. If this was their last move, predecessors owned by the defender are also won by the attacker, which is propagated in line 11 as well.

The algorithm runs in linear time of the game moves because every node can only turn once. It can be used to decide arbitrary finite simple games and, consequently, also to decide our coupled simulation game $\mathcal{G}_{CS}$ from Subsection 3.4.1.

Using compute_winning_region, it is only a matter of a few lines to determine the coupled simulation preorder for a system $\mathcal{S}$ as shown in game_compute_cs in Algorithm 4. One starts by constructing the corresponding game $\mathcal{G}^{\mathcal{S}}_{CS}$ using a function obtain_cs_game, we consider given by Definition 3.26. Then, one calls compute_winning_region and collects the attacker nodes won by the defender for the result. The implementation, including the construction of the game, can be found in `de.bbisping.coupledsim.algo.cs.GameCoupledSimilarityPlain`.

4.4.2 Correctness

The correctness of game_compute_cs in Algorithm 4 is mostly guaranteed by the proofs conducted in Subsection 3.4.2.

Theorem 4.14. *For a finite labeled transition systems $\mathcal{S}$,* game_compute_cs($\mathcal{S}$) *from Algorithm 4 returns $\sqsubseteq^{\mathcal{S}}_{CS}$.*

Proof. Theorem 3.29 states that the defender wins $\mathcal{G}^{\mathcal{S}}_{CS}[(p,q)_{\mathsf{a}}]$ exactly if $p \sqsubseteq^{\mathcal{S}}_{CS} q$. As compute_winning_region($\mathcal{G}^{\mathcal{S}}_{CS}$), according to [Grä07, Kre16], returns where the defender wins, line 4 of Algorithm 4 precisely assigns $\mathcal{R} = \sqsubseteq^{\mathcal{S}}_{CS}$. ☐

4.4.3 Complexity

The complexity arguments from [Grä07] yield linear complexity for deciding the game by Algorithm 3.

Proposition 4.15. *For a game $\mathcal{G} = (G, G_a, \rightarrowtail)$,* compute_winning_region *(Algorithm 3) runs in $\mathcal{O}(|G| + |\rightarrowtail|)$ time and space.*

In order to tell the overall complexity of the resulting algorithm, we have to look at the size of $\mathcal{G}^{\mathcal{S}}_{CS}$ depending on the size of $\mathcal{S}$.

Lemma 4.16. *Consider the coupled simulation game $\mathcal{G}^{\mathcal{S}}_{CS} = (G, G_a, \rightarrowtail)$ for varying $\mathcal{S} = (S, \Sigma_\tau, \rightarrow)$. The growth of the game size $|G| + |\rightarrowtail|$ is in $\mathcal{O}(|\overset{\hat{\cdot}}{\Rightarrow}|\,|S|)$.*

Proof. Let us reexamine Definition 3.26. There are $|S|^2$ attacker nodes. Collectively, they can formulate $\mathcal{O}(|\overset{\cdot}{\rightarrow}|\,|S|)$ simulation challenges and $|S|^2$ coupling challenges. There are $\mathcal{O}(|\overset{\hat{\cdot}}{\Rightarrow}|\,|S|)$ simulation answers and $\mathcal{O}(|\Rightarrow|\,|S|)$ coupling answers. Of these, $\mathcal{O}(|\overset{\hat{\cdot}}{\Rightarrow}|\,|S|)$ dominates the others. ☐

Lemma 4.17. game_compute_cs *(Algorithm 4) runs in $\mathcal{O}(|\overset{\hat{\cdot}}{\Rightarrow}|\,|S|)$ time and space.*

Proof. Proposition 4.15 and Lemma 4.16 already yield that line 3 is in $\mathcal{O}(|\overset{\hat{\cdot}}{\Rightarrow}|\,|S|)$ time and space. Definition 3.26 is completely straight-forward, so the complexity of building $\mathcal{G}^{\mathcal{S}}_{CS}$ in line 2 equals its output size $\mathcal{O}(|\overset{\hat{\cdot}}{\Rightarrow}|\,|S|)$, which coincides with the complexity of computing $\overset{\hat{\cdot}}{\Rightarrow}$. The filtering in line 4 is in $\mathcal{O}(|S|^2)$ (upper bound for attacker nodes) and thus does not influence the overall complexity. ☐

With the estimate that $\mathcal{O}(|\overset{\hat{\cdot}}{\Rightarrow}|) = \mathcal{O}(|\Sigma|\,|S|^2)$, this would amount to $\mathcal{O}(|\Sigma|\,|S|^3)$. The game algorithm thus has cubic complexity with respect to the state space. Recalling that Section 3.5 already predicted cubic complexity due to the simulation nature of coupled simulation, the presented game algorithm should be decent enough as a basis for further work. Therefore, the remainder of the chapter focuses on improving it.

4.5 Optimizing the Game Algorithm

A huge advantage of the game-theoretical approach is that it adds an intuitive and manageable layer of abstraction. In many situations, we can use additional knowledge we have about $\sqsubseteq_{CS}$ to reduce the game size.

```
def discover_game($G_0$, $successors$):
    predecessors := [] with default $\emptyset$
    num := []
    todo := $G_0$
    discovered := $G_0$
    while todo $\neq \emptyset$:
        choose $g_c \in$ todo
        todo := todo $\setminus \{g_c\}$
        num$[g_c]$ := $|successors(g_c)|$
        for $g_n \in successors(g_c)$:
            predecessors$[g_n]$ := predecessors$[g_n] \cup \{g_c\}$
            if $g_n \notin$ discovered :
                discovered := discovered $\cup \{g_n\}$
                todo := todo $\cup \{g_n\}$
    return (discovered, predecessors, num)
```

Algorithm 5 : Game discovery.

4.5.1 Discovering What Matters

The $\mathcal{G}_{CS}$-game from Section 3.4 can directly be used to answer the question: "For every p, q in S, is $p \sqsubseteq_{CS} q$?" But usually we are not interested in *every* p and q.

One common case is to ask the question only for two initial states p_0 and q_0, for example, because the relationship of two program fragments starting in these states is of interest. To determine $p_0 \sqsubseteq_{CS} q_0$, only some part of $\sqsubseteq_{CS}$ must be decided. As a consequence, we would not need the full-blown game graph. For some examples, this might even render the $\sqsubseteq_{CS}$-problem on infinite transition systems decidable.

Another instance where we wonder whether $p \sqsubseteq_{CS} q$ just for a subset of $S \times S$ is if we already know that $p \not\sqsubseteq_{CS} q$ for some p, q. For example, if p has more weakly enabled actions than q, we know that $p \not\sqsubseteq_{CS} q$ without invoking the algorithm.

For these instances, it is reasonable not to build the whole game graph but only *the portion that matters.*

discover_game of Algorithm 5 constructs only the fragment of a game that is reachable from some relevant game positions G_0 via moves specified by a function $successors \colon G \to 2^G$. It returns precisely the information that is needed in the initialization of the winning region algorithm from Subsection 4.4.1. It is an ordinary graph search algorithm, so it runs in linear time of the discovered edges and linear space of the reached vertices. The implementation can be found in `de.bbisping.coupledsim.game.GameDiscovery`.

4.5.2 Over- and Under-Approximations

The game can be downsized tremendously once we take additional over- and under-approximation information into account.

Definition 4.18. An *over-approximation* of $\sqsubseteq_{CS}$ is a relation $\mathcal{R}_O$ of that we know that $\sqsubseteq_{CS} \subseteq \mathcal{R}_O$. Conversely, an *under-approximation* of $\sqsubseteq_{CS}$ is a relation $\mathcal{R}_U$ where $\mathcal{R}_U \subseteq \sqsubseteq_{CS}$.

For instance, an obvious under-approximation would be the identity function Δ_S as we do not have to ask whether $p \sqsubseteq_{CS} p$. It is true no matter the structure of $\mathcal{S}$.

Over-approximations can be combined to acquire higher precision: If $\mathcal{R}_{O1}$ and $\mathcal{R}_{O2}$ are over-approximations then so is $\mathcal{R}_{O1} \cap \mathcal{R}_{O2}$. The dual holds for under-approximations ($\mathcal{R}_{U1} \cup \mathcal{R}_{U2}$).

Regarding the game, over-approximations tell us where the defender *can* win, and under-approximations tell us where the attacker is doomed to lose. They can be used to eliminate "boring" parts of the game.

Given an over-approximation $\mathcal{R}_O$, when discovering the game, it only makes sense to add moves from defender nodes to attacker nodes $(p, q)_{\mathtt{a}}$ if $(p, q) \in \mathcal{R}_O$. There just is no need to allow the defender moves we already know cannot be winning for them. So, the discovery can skip outgoing game moves from defender nodes to attacker nodes if they leave the over-approximation. Likewise, only subsets of $\mathcal{R}_O$ should be used as starting point for the discovery from Subsection 4.5.1—the other attacker nodes do not matter because they would be won by the attacker anyways.

Given an under-approximation $\mathcal{R}_U$, we can ignore all the outgoing moves of $(p, q)_{\mathtt{a}}$ if $(p, q) \in \mathcal{R}_U$. Without moves, $(p, q)_{\mathtt{a}}$ is sure to be won by the defender, which is in line with the claim of the approximation.

`de.bbisping.coupledsim.algo.cs.GameCoupledSimilarity` implements a variant of the coupled simulation game where the game graph can be kept small using approximations.

Two examples for approximations that are easily supplyable once the τ-closure has been computed are:

Corollary 4.19. $\Rightarrow^{-1}$ *is an under-approximation of* $\sqsubseteq_{CS}$*. (Cf. Lemma 3.16)*

Lemma 4.20. $\mathcal{R} = \{(p, q) \mid$ *all actions weakly enabled in* p *are weakly enabled in* $q\}$ *is an over-approximation of* $\sqsubseteq_{CS}$.[9]

The trouble with over-approximations is that, as we already discussed, coarser notions of equivalence tend to have more expensive algorithms than finer ones. The question thus is which helpful coarser notions of equivalence are easily computable in the context of our coupled simulation preorder algorithm.

4.5.3 Over-Approximation by Maximal Weak Steps

That coupled simulation is "almost bisimulation" on steps to stable states in finite systems (Lemma 3.20) can be used for a comparably cheap and precise over-approximation.

The idea is to compute the strong bisimulation on the system $\mathcal{S}_{\Rightarrow|} = (S, \Sigma_\tau, \Rightarrow|)$, where *maximal weak steps*, $p \overset{\alpha}{\Rightarrow}| \, p'$, exist iff $p \overset{\hat{\alpha}}{\Rightarrow} p'$ and p' is stable, that is, $p' \overset{\tau}{\not\to}$. Let $\equiv_{\Rightarrow|}$ be the biggest symmetric relation where

$$p \equiv_{\Rightarrow|} q \text{ and } p \overset{\alpha}{\Rightarrow}| \, p' \text{ implies there is } q' \text{ such that } p' \equiv_{\Rightarrow|} q' \text{ and } q \overset{\alpha}{\Rightarrow}| \, q'.$$

By Lemma 3.20, $\equiv_{CS} \subseteq \equiv_{\Rightarrow|}$. Using this and exploiting that the existential quantification can be hidden in the relation concatenation of $\overset{\alpha}{\Rightarrow}|$ and $\equiv_{\Rightarrow|}$, Lemma 3.20 can be rewritten to:

[9] corollary `Coupled_Simulation.coupledsim_enabled_subs`

$$p \sqsubseteq_{CS} q \text{ and } p \overset{\alpha}{\Rightarrow}\!| \, p' \text{ implies } q \overset{\alpha}{\Rightarrow}\!| \equiv_{\Rightarrow|} p'.$$

This yields that the subset relationship on the $\overset{\alpha}{\Rightarrow}\!| \equiv_{\Rightarrow|}$-signatures makes for a neat over-approximation:

Lemma 4.21. $\mathcal{R}_{\Rightarrow|} = \{(p, q) \mid (p \overset{\alpha}{\Rightarrow}\!| \cdot) \subseteq (q \overset{\alpha}{\Rightarrow}\!| \equiv_{\Rightarrow|} \cdot)\}$ *is an over-approximation of* $\sqsubseteq_{CS}$ *on finite systems.*

Computing $\equiv_{\Rightarrow|}$ can be expected to be cheaper than computing $\equiv_{WB}$. After all, $\overset{\cdot}{\Rightarrow}\!|$ is just a subset of $\overset{\hat{\cdot}}{\Rightarrow}$. However, filtering $S \times S$ using subset checks to create $\mathcal{R}_{\Rightarrow|}$ might well be *quartic,* $\mathcal{O}(|S|^4)$, or worse. Nevertheless, one can argue that with a reasonable algorithm design and for many real-world examples, $\overset{\alpha}{\Rightarrow}\!| \equiv_{\Rightarrow|}$ will be sufficiently bounded in branching degree, in order for the over-approximation to do more good than harm.

For everyday system designs, $\mathcal{R}_{\Rightarrow|}$ is a tight approximation of $\sqsubseteq_{CS}$. On the philosopher system $\mathcal{S}_P$ from Example 2.6f., they even coincide. In some situations, $\mathcal{R}_{\Rightarrow|}$ degenerates to the shared enabledness relation (Lemma 4.20), which is to say it becomes comparably useless. One example for this are the systems created by the reduction from weak simulation to coupled simulation in Subsection 3.5.1 after τ-cycle removal. There, all $\Rightarrow\!|$-steps are bound to end in the same one τ-sink state $\bot$.

Our implementation computes $\equiv_{\Rightarrow|}$ using a *signature refinement algorithm* in `de.bbisping.coupledsim.algo.sigref.BigStepEquivalence`. Signature refinement [WHH+06, Wim11] describes a class of iterative algorithms for coarsest partitions where algorithms for bisimulation-like equivalences can be expressed concisely by specifying a “signature.” The signature describes for every state which information about the current partitions of successor states can distinguish it from other states. The advantage of this class of algorithms is their maintainability and their distributability [BO03].

4.5.4 Gamifying the τ-Closure

The inherently transitive nature of games gives us the opportunity to avoid the computation of the τ-closure. The exploration of the τ-closure necessary to find a tuple from $\overset{\hat{\alpha}}{\Rightarrow}$ as simulation answer or a tuple from $\Rightarrow$ as coupling answer can also be accomplished using steps between defender nodes.

Definition 4.22. The *τ-closure-free coupled simulation game* $\mathcal{G}^{\tau}_{CS}[p_0] = (G, G_a, \rightarrowtail, p_0)$ has the same kinds of nodes as $\mathcal{G}_{CS}[p_0]$ from Definition 3.26 and the following kinds of moves:

- *simulation challenges* $(p, q)_{\mathtt{a}} \rightarrowtail (\alpha, p', q)_{\mathtt{d}}$ if $p \xrightarrow{\alpha} p'$,
- **new:** *simulation answer postponements* $(\alpha, p', q)_{\mathtt{d}} \rightarrowtail (\alpha, p', q')_{\mathtt{d}}$ if $q \xrightarrow{\tau} q'$,
- **new:** *simulation answer steps* $(\alpha, p', q)_{\mathtt{d}} \rightarrowtail (\tau, p', q')_{\mathtt{d}}$ if $q \xrightarrow{\alpha} q'$,
- **new:** *simulation answer resolutions* $(\tau, p, q)_{\mathtt{d}} \rightarrowtail (p, q)_{\mathtt{a}}$,
- *coupling challenges* $(p, q)_{\mathtt{a}} \rightarrowtail (\mathtt{Cpl}, p, q)_{\mathtt{d}}$,
- **new:** *coupling steps* $(\mathtt{Cpl}, p, q)_{\mathtt{d}} \rightarrowtail (\mathtt{Cpl}, p, q')_{\mathtt{d}}$ if $q \xrightarrow{\tau} q'$, and
- **modified:** *coupling resolutions* $(\mathtt{Cpl}, p, q)_{\mathtt{d}} \rightarrowtail (q, p)_{\mathtt{a}}$.

This reformulation relies on the τ-cycle removal from Subsection 4.1.1. If they had not been removed, such τ-cycles would break the game as they would introduce spurious infinite winning runs for the defender.

Following Definition 4.22, the game can be constructed without the costly computation of the τ-closure. Interestingly, this modification at the same time reduces the amount of moves in the game. We are saving twice.

Now, the old dominating amount of simulation answers in $\mathcal{O}(|\overset{\hat{\cdot}}{\Rightarrow}|\,|S|)$ is superseded by the amounts of simulation answer postponements $\mathcal{O}(|\Sigma_\tau|\,|\xrightarrow{\tau}|\,|S|)$ and simulation answer steps $\mathcal{O}(|\dot{\rightarrow}|\,|S|)$. $\mathcal{O}(|\Rightarrow|\,|S|)$ coupling answers are replaced by $\mathcal{O}(|\xrightarrow{\tau}|\,|S|)$ coupling steps (and $\mathcal{O}(|S|^2)$ resolutions). So, for sensible transition systems $\mathcal{O}(|\Sigma_\tau|\,|\xrightarrow{\tau}|\,|S|)$ becomes the new size bound. Although this bound is effectively smaller than $\mathcal{O}(|\overset{\hat{\cdot}}{\Rightarrow}|\,|S|)$, it still is cubic in the state space.

4.6 Discussion

Of the algorithms presented, clearly the game-theoretic one is the most attractive. Its cubic $\mathcal{O}(|\overset{\hat{\cdot}}{\Rightarrow}|\,|S|)$ complexity and its flexibility make it a good choice for many applications. However, the game graph of cubic size uses a lot of space.

The fixed-point algorithm has the advantage of a tendentially lower memory usage, its simplicity, and verified dependability. Its $\mathcal{O}(|S|^6)$ time complexity is daunting. It might however be a good basis for improvements in the spirit of the simulation algorithms in [HHK95].

The bisimulation reduction algorithm is neither simple nor efficient. This comes from the peculiar normalization procedure in [PS94]. Unless a trick is devised to get around the partial commit closure, it is not really fit for application.

Further improvements of the algorithms could be possible by "symbolic refinements." A classical approach in simulation algorithms [HHK95, GPP03, vGP08, RT10] is to represent $\mathcal{R}$ by a partition relation pair (P, R), where the relation R is only defined on partitions of S instead of the whole graph. Partitions of graphs (aka. "colorings") can be stored very efficiently and a lot of tuples of $\mathcal{R}$ can be saved this way. This might bring down some of the $|S|$ components of the complexity terms to $|S_{/\equiv_{CS}}|$. Once we assume that the transition system has been minimized with respect to $\equiv_{WB}$ before our computations, so that $|S| = |S_{/\equiv_{WB}}|$, the partition relation pair trick would not save a lot. $|S_{/\equiv_{WB}}|$ and $|S_{/\equiv_{CS}}|$ will be almost equal for many systems.

The other route of symbolic refinements are binary decision diagrams (BDDs), where $\mathcal{R}$ and $\overset{\hat{\cdot}}{\Rightarrow}$ can be stored and modified in a kind of on-the-fly zip format [Wim11, WHH$^+$06, Bul11]. This would be connected with some technicalities and would hamper distributability. But it can lead to significant complexity savings as long as one disregards the complexity of the conversions from and to the BDD representation.

Chapter 5

A Scalable Implementation

How applicable are our algorithms in practice? Coupled similarity has restrictions on the size of systems it can cope with as it inherits from similarity that it is computationally harder than bisimilarity.

The experimental results by Ranzato and Tapparo [RT10] suggest that their simulation algorithm and the algorithm by Henzinger, Henzinger, and Kopke [HHK95] only work on comparably small systems. The necessary data structures quickly consume gigabytes of RAM. So, the bothering question is not so much whether some highly optimized C++-implementation can do the job in milliseconds for small problems, but how to implement the algorithm such that large-scale systems are feasible at all.

To give first answers, this chapter presents a scalable and distributable prototype implementation of the coupled simulation game algorithm using the stream processing framework Apache Flink.

5.1 Prototype Implementation

We base our implementation on the game algorithm from Section 4.4 and the optimizations presented in Section 4.5. The implementation is a vertical prototype in the sense that every feature to get from a transition system to its coupled simulation preorder is present, but there is no big variety of options in the process.

5.1.1 Apache Flink

Apache Flink [CKE+15] (`https://flink.apache.org/`) is a platform for computations on large data sets built around a universal data-flow engine. Once an algorithm has been adapted to the data-flow way of describing computation, it in principle is easy to distribute and replicate the program parts to use whole server farms of memory and computational power.

An Apache Flink program is a graph whose nodes represent data sources and sinks, intermediate data sets, and operations. The programs can easily be built using Scala code. Its API supports iterative algorithms. Flink ships with the library *Gelly*[1] for graph algorithms.

[1]Gelly—Flink Graph API, `https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/libs/gelly/`.

5.1.2 Algorithm Stages

`de.bbisping.coupledsim.flink.CoupledSimulationFlink` describes a pipeline to compute the coupled simulation preorder. Abstractly, it first minimizes the transition graph using a finer equivalence relation, then computes an over-approximation, uses the result to build the interesting part of the coupled simulation game, and finally computes the winning regions of the game. The phases, in detail, are:

Import Reads a CSV representation of the transition system $\mathcal{S}$. Transition systems are encoded as comma separated values of edge sets, which is one of the default formats for graphs in Flink/Gelly. Every line represents one transition in the form *sourceID*, *targetID*, `"`*actionLabel*`"`. The IDs are positive integers, the actions are strings. The internal action (τ) is denoted by `i`. At runtime, the actions are recoded to integers.

Minimize Computes an equivalence relation under-approximating $\equiv_{CS}$ on the transition system. The partition structure is represented as a coloring C, that is, as a mapping from state IDs to "colors" (integers). This mapping then is used to minimize the transition graph, obtaining $\mathcal{S}_M$, and to restore the full $\sqsubseteq^{\mathcal{S}}_{CS}$ relation later on. This stage should at least compress τ-cycles if there are any. Currently, there is only a weak bisimulation ($\equiv^{\mathcal{S}}_{WB}$) and a delay bisimulation ($\equiv^{\mathcal{S}}_{DB}$) option. To reduce the cost of the weak bisimulation computation, we prepend it by a strong bisimulation minimization.

Compute over-approximation Determines an equivalence relation over-approximating $\equiv^{\mathcal{S}_M}_{CS}$. The result is a mapping σ from states to *signatures* (sets of colors) such that $p \sqsubseteq^{\mathcal{S}_M}_{CS} q$ implies $\sigma(p) \subseteq \sigma(q)$. In the prototype, only the maximal weak step equivalence $\equiv_{\Rightarrow|}$ from Subsection 4.5.3 is available.

Build game graph Constructs the τ-closure-free coupled simulation game $\mathcal{G}^{\tau\,\mathcal{S}_M}_{CS}$ for $\mathcal{S}_M$ (Definition 4.22) with attacker states restricted according to the over-approximation signatures σ. The algorithm in `de.bbisping.coupledsim.flink.CoupledSimulationGameDiscovery` is a distributable version of the iterative game space discovery from Subsection 4.5.1.

Compute winning regions Decides for $\mathcal{G}^{\tau\,\mathcal{S}_M}_{CS}$ where the attacker has a winning strategy. `de.bbisping.coupledsim.flink.SimpleGameDecider` implements a distributable version of Algorithm 3, following the scatter-gather scheme from [KVH18]. If a game node is discovered to be won by the attacker, it *scatters* the information to its predecessors. Every game node *gathers* information on its winning successors. Defender nodes count down their degrees of freedom starting at their game move out-degrees.

Output Finally, the results can be output or verified. The winning regions directly imply $\sqsubseteq^{\mathcal{S}_M}_{CS}$. To work with $\sqsubseteq^{\mathcal{S}}_{CS}$, the tuples from $\sqsubseteq^{\mathcal{S}_M}_{CS}$ must be multiplied with C to obtain $\sqsubseteq^{\mathcal{S}}_{CS} = C \sqsubseteq^{\mathcal{S}_M}_{CS} C^{-1}$. At output, the algorithm can also check that the computed $\sqsubseteq^{\mathcal{S}}_{CS}$ indeed is a coupled simulation. Due to the massive size of $\sqsubseteq^{\mathcal{S}}_{CS}$ and $\overset{\hat{\cdot}}{\Rightarrow}$, this might not always be possible in a reasonable amount of time.

system	S	$\dot{\rightarrow}$	$\overset{\hat{\cdot}}{\Rightarrow}^2$	$S_{/\equiv_{DB}}$	$S_{/\equiv_{CS}}$	$\sqsubseteq_{CS}^{S_{/\equiv_{CS}}}$	time/s
`phil`	10	14	86	6	5	11	7.9
`ltbts`	88	98	2,599	27	25	38	8.2
`vasy_0_1`	289	1,224	52,641	9	9	9	10.0
`vasy_1_4`	1,183	4,464	637,585	4	4	4	9.6
`vasy_5_9`	5,486	9,676	1,335,325	112	112	112	10.6
`cwi_1_2`	1,952	2,387	593,734	67	67	137	12.9
`cwi_3_14`	3,996	14,552	15,964,021	2	2	2	18.8
`vasy_8_24`	8,879	24,411	2,615,500	170	169	232	13.1
`vasy_8_38`	8,921	38,424	46,232,423	193	193	193	9.4
`vasy_10_56`	10,849	56,156	842,087	2,112	2,112	3,932	29.8
`vasy_25_25`	25,217	25,216	50,433	25,217	25,217	25,217	243.1

Table 5.1: Sample systems, sizes, and benchmark results.

5.1.3 Signature Refinement

Unfortunately, there seems to be no prior implementation of bisimulation computation in Apache Flink. De Lange et al. [LdLF+13] have realized bisimulation for the akin technology stack HADOOP/MapReduce using a signature-based approach. Signature-based approaches indeed are the natural pick for our setting because of their distributability. The computations of $\equiv_{WB}$ and $\equiv_{\Rightarrow|}$ rely on a small ad-hoc signature refinement implementation of bisimulation in `de.bbisping.coupledsim.flink.SignatureRefinement`. Due to some technicalities that would require a lot of fiddling, our implementation, for now, only is correct as long as the 32-bit hash values of the signature sets do not collide. So, starting at about 10,000 input states, there is a danger ($> 1\,\%$) of erroneous results. A more mature solution would be desirable. Especially, in order to tackle bigger examples, the explicit computation of the τ-closure would have to be optimized away as in [BGRR16].

5.2 Evaluation

Experimental evaluation shows that the approach can cope with the smaller examples of the "Very Large Transition Systems (VLTS) Benchmark Suite" [Gar17] (`vasy_*` and `cwi_*` up to 50,000 transitions). On small examples, we also tested that the output matches the return values of the verified fixed-point $\sqsubseteq_{CS}$-algorithm from Section 4.3. These samples include, among others, the philosopher system `phil` containing P_g and P_o from Example 2.5f. and `ltbts`, which consists of the finitary separating examples from the linear-time branching-time spectrum [vG93, p. 73].

Table 5.1 summarizes the results for some of our test systems with pre-minimization by delay bisimilarity and over-approximation by maximal weak step equivalence. The first two value columns give the system sizes in number of states S and transitions $\dot{\rightarrow}$. The next two columns present derived properties, namely the size of the (weak) delay step relation $\overset{\hat{\cdot}}{\Rightarrow}$,[2] and the number of partitions with respect to delay bisimulation $S_{/\equiv_{DB}}$. The following columns enumerate the sizes of the resulting coupled simulation preorders represented by the partition relation pair $(S_{/\equiv_{CS}}, \sqsubseteq_{CS}^{S_{/\equiv_{CS}}})$, where $S_{/\equiv_{CS}}$ is

[2]The reported size of $\overset{\hat{\cdot}}{\Rightarrow}$ actually only is an upper estimate! As the algorithm works on a system minimized by strong bisimilarity at this stage, the exact size is unknown.

the partitioning of S with respect to coupled similarity $\equiv_{CS}$, and $\sqsubseteq_{CS}^{\mathcal{S}_{/\equiv_{CS}}}$ the coupled simulation preorder projected to this quotient. The last column reports the running time of the programs on an Intel i7-7500U CPU (2.70GHz) with 2 GB Java Virtual Machine heap space.

The systems in Table 5.1 are a superset of the VLTS systems for which Ranzato and Tapparo [RT10] report their algorithm *SA* to terminate. Regarding complexity, SA is the best simulation algorithm known. In the [RT10]-experiments, the C++ implementation ran out of 2 GB RAM for `vasy_10_56` and `vasy_25_25` but finished much faster than our setup for most smaller examples. Their time advantage on small systems comes as no surprise as the start-up of the whole Apache Flink pipeline induces heavy overhead costs of about 8 seconds even for tiny examples like `phil`.

However, on bigger examples such as `vasy_18_73` and `vasy_40_60` both implementations fail. This is in stark contrast to *bi*-simulation implementations, which usually cope with much larger systems single-handedly [Li09, BGRR16].[3]

The experiments yield further evidence for claims from the thesis. The weak step relations $\overset{\hat{\cdot}}{\Rightarrow\!\!\triangleright}$ (and its superset $\overset{\hat{\cdot}}{\Rightarrow}$) can indeed become abhorrently large, which is in line with the theoretical considerations from Subsection 3.5.2. For all tested VLTS systems, $S_{/\equiv_{WB}}$ equals $S_{/\equiv_{CS}}$ (and, with the exception of `vasy_8_24`, $S_{/\equiv_{DB}}$). This supports the prediction from Section 4.6 that partition relation refinement algorithms cannot gain much in a setting where a finer weak/delay bisimulation minimization has already been applied. The preorder $\sqsubseteq_{CS}^{\mathcal{S}_{/\equiv_{CS}}}$ also matches the identity in 6 of 9 examples.

This observation about the effective closeness of $\sqsubseteq_{CS}$ and $\equiv_{WB}$ is two-fold. On the one hand, it brings into question how meaningful coupled similarity is for minimization. After all, it takes a lot of space and time to come up with the output that delay bisimilarity already minimized everything that could be minimized. On the other hand, the observation suggests that the considered VLTS samples are based around models that do not need—or maybe even do avoid—the expressive power of $\equiv_{WB}$. This is further evidence for the case that $\sqsubseteq_{CS}$ *has a more sensible level of precision than* $\equiv_{WB}$.

The experiments moreover demonstrate that the optimizations from Section 4.5 really are necessary for larger systems. Table 5.2 lists the sizes of the game graphs without and with maximal weak step over-approximation ($\rightarrowtail$ and $\rightarrowtail_\sigma$). Without over-approximation, `vasy_10_56` and `vasy_25_25` ran out of memory ("o.o.m."). The over-approximation stage, which accounts for half of the program running time, creates the opportunity to bring down game size decisively. Due to the τ-trick from Subsection 4.5.4, the game size can remain a fraction of the weak transition relation size.

system	$S_{/\equiv_{DB}}$	$\rightarrowtail$	$\rightarrowtail_\sigma$.
`phil`	6	320	201
`ltbts`	27	4,529	399
`vasy_0_1`	9	543	67
`vasy_1_4`	4	105	30
`vasy_5_9`	112	68,686	808
`cwi_1_2`	67	46,871	1,559
`cwi_3_14`	2	19	10
`vasy_8_24`	170	275,585	3,199
`vasy_8_38`	193	303,819	2,163
`vasy_10_56`	2,112	o.o.m.	74,269
`vasy_25_25`	25,217	o.o.m.	126,083

Table 5.2: Plain vs. optimized game size.

[3]`vasy_40_60` is an anomaly where Paige–Tarjan-based algorithms outperform signature-based algorithms by orders of magnitude; for `vasy_25_25`, it is the other way around.

Chapter 6

Conclusion

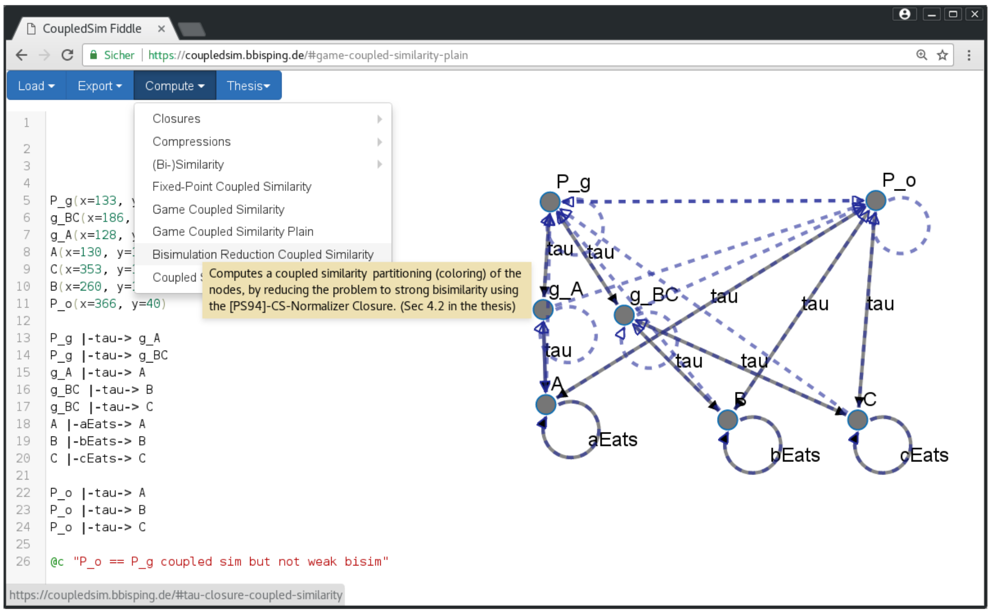
They say that all good things come to an end. So, ω-traces and defender-winning plays presumably are no good things. This thesis has the chance of being good, given that it has an end—even more than one if one also counts the ends intentionally left loose.

The core of this thesis has been to introduce a game algorithm to decide the coupled simulation preorder and thereby coupled similarity. This algorithm runs in cubic time with respect to the state space size of the input transition system. Since we have shown that weak simulation reduces to coupled simulation, the theoretical time bounds presumably cannot get much better.

Nevertheless, it would be interesting whether our algorithms can be modified to require *sub-cubic space*. A few pointers on how this could be possible can be found in the discussion of symbolic refinements (Section 4.6). Related to this is the question how to turn the *modal-logical characterization* of coupled similarity into an algorithm. Van Glabbeek [vG93] presents a characterization that basically is Hennesy–Milner modal logic (HML) with a *weakened negation* operator $\neg\!\!\!\neg$, where $\neg\!\!\!\neg\varphi$ is true in a state iff it can reach a state where not φ by $\Rightarrow$-steps. Some similarity algorithms, including Ranzato and Tapparo's [RT10], iteratively split partitions along the question which states can be distinguished by modal-logic formulas in light of the distinctions determined so far. Such algorithms can certainly be amended with the additional separating power of $\neg\!\!\!\neg$. However, we suspect that many algorithmic tricks stop being an option as $\neg\!\!\!\neg$ is not distributive.

We have not touched on the question how the *maximal weak step bisimilarity* we used for over-approximations relates to the standard notions of equivalence. Also, the closeness of coupled similarity and delay similarity, which usually is neglected in the literature, could be researched further. We were able to show that there are open questions concerning the relationship of coupled similarity and contrasimilarity. Addressing these questions could pave the way to *employing our algorithms to decide contrasimilarity*, for which there also seems to be no tool support.

Our scalable implementation of the coupled simulation game algorithm has shown that the game approach *can cope with similar system sizes when compared to the available simulation algorithms.* The implementation could be a basis for further experiments with notions of equivalence in Apache Flink or a blueprint for a GPU version of the algorithm.


CoupledSim Fiddle
Sicher | https://coupledsim.bbisping.de/#game-coupled-similarity-plain
Load
Export
Compute
Thesis
Closures
Compressions
(Bi-)Similarity
Fixed-Point Coupled Similarity
Game Coupled Similarity
Game Coupled Similarity Plain
Bisimulation Reduction Coupled Similarity
Coupled
P_g
P_o
g_A
g_BC
A
B
C
tau
aEats
bEats
cEats
https://coupledsim.bbisping.de/#tau-closure-coupled-similarity

Computes a coupled similarity partitioning (coloring) of the nodes, by reducing the problem to strong bisimilarity using the [PS94]-CS-Normalizer Closure. (Sec 4.2 in the thesis)

```



P_g(x=133, y
g_BC(x=186,
g_A(x=128, y
A(x=130, y=
C(x=353, y=
B(x=260, y=
P_o(x=366, y=40)

P_g |-tau-> g_A
P_g |-tau-> g_BC
g_A |-tau-> A
g_BC |-tau-> B
g_BC |-tau-> C
A |-aEats-> A
B |-bEats-> B
C |-cEats-> C

P_o |-tau-> A
P_o |-tau-> B
P_o |-tau-> C

@c "P_o == P_g coupled sim but not weak bisim"
```

Figure 6.1: CoupledSim Fiddle web tool at `coupledsim.bbisping.de` showing the philosopher system $\mathcal{S}_\mathsf{P}$ (gray) and $\sqsubseteq_{CS}^{\mathcal{S}_\mathsf{P}}$ (blue).

Moreover, we have *verified a simple coupled simulation algorithm* and *continued work on a normalization-based reduction* of coupled similarity to bisimilarity. Development, testing, and presentation of new algorithms for notions of equivalence is made simple by our web tool on `https://coupledsim.bbisping.de` (for a screenshot, see Figure 6.1).

By benchmarking our Flink-implementation with the VLTS suite, we have established that *coupled similarity and weak bisimilarity coincide for the considered systems*. This points back to a line of thought by van Glabbeek and Weijland [vGW96] that, for many applications, branching, delay and weak bisimilarity will coincide with coupled similarity. Where they do not, usually coupled similarity or a coarser notion of equivalence is called for. Apparently, the designers of the case studies behind the VLTS systems (unintentionally?) took care to stay in the realm where various branching-time equivalences resemble coupled similarity. The higher semantic precision of weak bisimilarity and the like is uncalled for, and coupled similarity probably does right to dispense with it. However, real case studies—and maybe an embedding into existing tool landscapes like FDR [GRABR14], CADP [GLMS13], or LTSmin [KLM$^+$15]—would be necessary to gain deeper insights here.

It would be interesting to have a better way of telling ex ante *whether coupled similarity and finer notions of equivalence coincide* for a given transition system. The normality property of Section 4.2, for example, describes a class of transition systems where the hierarchy between coupled simulation and strong bisimulation collapses. Can one abstract this property further, or narrow it down to a property where only the hierarchy between branching bisimulation and coupled simulation is bridged? Such a result could help turn the normalization-based reduction into a reliable and tractable algorithm. This—and many other interesting research approaches—would be facilitated by our Isabelle/HOL formalization for various notions of coupled simulation and their properties.

Bibliography

[AIS11] Luca Aceto, Anna Ingolfsdottir, and Jiri Srba. *The algorithmics of bisimilarity*, pages 100–172. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2011. 16

[Bel13] Christian J. Bell. Certifiably sound parallelizing transformations. In *International Conference on Certified Programs and Proofs*, pages 227–242. Springer, 2013. 13

[BGRR16] Alexandre Boulgakov, Thomas Gibson-Robinson, and A. W. Roscoe. Computing maximal weak and other bisimulations. *Formal Aspects of Computing*, 28(3):381–407, 2016. 2, 35, 50, 51

[BO03] Stefan Blom and Simona Orzan. Distributed branching bisimulation reduction of state spaces. *Electronic Notes in Theoretical Computer Science*, 89(1):99–113, 2003. 46

[Bul11] Peter E. Bulychev. Game-theoretic simulation checking tool. *Programming and Computer Software*, 37(4):200, 2011. 47

[CH93] Rance Cleaveland and Matthew Hennessy. Testing equivalence as a bisimulation equivalence. *Formal Aspects of Computing*, 5(1):1–20, 1993. 39, 41

[CKE$^+$15] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache Flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015. 48

[DPP04] Agostino Dovier, Carla Piazza, and Alberto Policriti. An efficient algorithm for computing bisimulation equivalence. *Theoretical Computer Science*, 311(1):221–256, 2004. 32

[DW03] John Derrick and Heike Wehrheim. Using coupled simulations in non-atomic refinement. In *International Conference of B and Z Users*, pages 127–147, Berlin, Heidelberg, 2003. Springer. 1

[EHS$^+$13] Christian Eisentraut, Holger Hermanns, Johann Schuster, Andrea Turrini, and Lijun Zhang. The quest for minimal quotients for probabilistic automata. In Nir Piterman and Scott A. Smolka, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 16–31, Berlin, Heidelberg, 2013. Springer. 35

[FG05] Cédric Fournet and Georges Gonthier. A hierarchy of equivalences for asynchronous calculi. *The Journal of Logic and Algebraic Programming*, 63(1):131–173, 2005. 22

[Gar17] Hubert Garavel. The VLTS benchmark suite, 2017. Jointly created by CWI/SEN2 and INRIA/VASY as a CADP resource. 50

[GJKW17] Jan Friso Groote, David N. Jansen, Jeroen J. A. Keiren, and Anton J. Wijs. An $\mathcal{O}(m \log n)$ algorithm for computing stuttering equivalence and branching bisimulation. *ACM Transactions on Computational Logic (TOCL)*, 18(2):13:1–13:34, 2017. 35

[GLMS13] Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. CADP 2011: A toolbox for the construction and analysis of distributed processes. *International Journal on Software Tools for Technology Transfer*, 15(2):89–107, 2013. 53

[GPP03] Raffaella Gentilini, Carla Piazza, and Alberto Policriti. From bisimulation to simulation: Coarsest partition problems. *Journal of Automated Reasoning*, 31(1):73–103, 2003. 47

[Grä07] Erich Grädel. Finite model theory and descriptive complexity. In E. Grädel, P.G. Kolaitis, L. Libkin, M. Marx, J. Spencer, M.Y. Vardi, Y. Venema, and S. Weinstein, editors, *Finite Model Theory and Its Applications*, Texts in Theoretical Computer Science. An EATCS Series, pages 125–230. Springer Berlin Heidelberg, 2007. 16, 41, 43

[GRABR14] Thomas Gibson-Robinson, Philip Armstrong, Alexandre Boulgakov, and A. W. Roscoe. FDR3: A modern refinement checker for CSP. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 187–201. Springer Berlin Heidelberg, 2014. 53

[HHK95] Monika Rauch Henzinger, Thomas A. Henzinger, and Peter W. Kopke. Computing simulations on finite and infinite graphs. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin*, pages 453–462, 1995. 32, 47, 48

[Hod13] Wilfrid Hodges. Logic and games. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2013 edition, 2013. 16

[HWPN15] Meike Hatzel, Christoph Wagner, Kirstin Peters, and Uwe Nestmann. Encoding CSP into CCS. In *Proceedings of the Combined 22th International Workshop on Expressiveness in Concurrency and 12th Workshop on Structural Operational Semantics, and 12th Workshop on Structural Operational Semantics, EXPRESS/SOS*, pages 61–75, 2015. 1

[KLM$^+$15] Gijs Kant, Alfons Laarman, Jeroen Meijer, Jaco van de Pol, Stefan Blom, and Tom van Dijk. LTSmin: High-performance language-independent model checking. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 692–707. Springer, 2015. 53

[KM02] Antonín Kučera and Richard Mayr. Why is simulation harder than bisimulation? In Luboš Brim, Mojmír Křetínský, Antonín Kučera, and Petr Jančar, editors, *CONCUR 2002 — Concurrency Theory: 13th International Conference Brno, Czech Republic, August 20–23, 2002 Proceedings*, pages 594–609. Springer Berlin Heidelberg, 2002. 31

[Kre16] Stephan Kreutzer. Logic, games, automata. Lecture notes, 2016. 41, 43

[KS06] Antonín Kučera and Philippe Schnoebelen. A general approach to comparing infinite-state systems with their finite-state specifications. *Theoretical Computer Science*, 358(2-3):315–333, 2006. 23

[KVH18] Vasiliki Kalavri, Vladimir Vlassov, and Seif Haridi. High-level programming abstractions for distributed graph processing. *IEEE Transactions on Knowledge and Data Engineering*, 30(2):305–324, 2018. 49

[LdLF$^+$13] Yongming Luo, Yannick de Lange, George HL Fletcher, Paul De Bra, Jan Hidders, and Yuqing Wu. Bisimulation reduction of big graphs on MapReduce. In *British National Conference on Databases*, pages 189–203. Springer, 2013. 50

[Li09] Weisong Li. Algorithms for computing weak bisimulation equivalence. In *Theoretical Aspects of Software Engineering, 2009. TASE 2009. Third IEEE International Symposium on*, pages 241–248. IEEE, 2009. 35, 51

[Mil89] Robin Milner. *Communication and Concurrency.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989. 5, 15, 16, 26, 28

[NNH15] Flemming Nielson, Hanne R. Nielson, and Chris Hankin. *Principles of program analysis.* Springer, 2015. 38

[NP00] Uwe Nestmann and Benjamin C. Pierce. Decoding choice encodings. *Information and Computation*, 163(1):1–59, 2000. 1, 6, 21, 26

[Pit11] Andrew Pitts. Howe's method for higher-order languages. In Davide Sangiorgi and Jan Rutten, editors, *Advanced Topics in Bisimulation and Coinduction*, pages 197–232. Cambridge University Press, 2011. 25

[PS92] Joachim Parrow and Peter Sjödin. Multiway synchronization verified with coupled simulation. In W.R. Cleaveland, editor, *CONCUR '92: Third International Conference on Concurrency Theory Stony Brook, NY, USA, August 24–27, 1992 Proceedings*, pages 518–533. Springer Berlin Heidelberg, 1992. 1, 6, 20

[PS94] Joachim Parrow and Peter Sjödin. The complete axiomatization of Cs-congruence. In Patrice Enjalbert, Ernst W. Mayr, and Klaus W. Wagner, editors, *STACS 94: 11th Annual Symposium on Theoretical Aspects of Computer Science Caen, France, February 24–26, 1994 Proceedings*, pages 555–568. Springer Berlin Heidelberg, 1994. 2, 14, 20, 21, 22, 26, 27, 28, 34, 36, 37, 39, 41, 47

[PT87] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987. 36

[PvG15] Kirstin Peters and Rob J. van Glabbeek. Analysing and comparing encodability criteria. In *Proceedings of the Combined 22th International Workshop on Expressiveness in Concurrency and 12th Workshop on Structural Operational Semantics, and 12th Workshop on Structural Operational Semantics, EXPRESS/SOS*, pages 46–60, 2015. 22

[Ren00] Arend Rensink. Action contraction. In *International Conference on Concurrency Theory*, pages 290–305. Springer, 2000. 1, 21, 25

[RT08] Francesco Ranzato and Francesco Tapparo. Generalizing the Paige–Tarjan algorithm by abstract interpretation. *Information and Computation*, 206(5):620–651, 2008. Special Issue: The 17th International Conference on Concurrency Theory (CONCUR 2006). 32, 35

[RT10] Francesco Ranzato and Francesco Tapparo. An efficient simulation algorithm based on abstract interpretation. *Information and Computation*, 208(1):1–22, 2010. 32, 47, 48, 51, 52

[San12] Davide Sangiorgi. *Introduction to Bisimulation and Coinduction.* Cambridge University Press, New York, NY, USA, 2012. 4, 6, 9, 20, 21, 22, 27

[Sch15] Christiane Schulte. Der deutsch-deutsche Schäferhund – Ein Beitrag zur Gewaltgeschichte des Jahrhunderts der Extreme. *Totalitarismus und Demokratie*, 12(2):319–334, 2015. 10

[Sti93] Colin Stirling. Modal and temporal logics for processes. Technical report, Department of Computer Science, University of Edinburgh, 1993. 17

[Tar72] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. 34

[vG90] Rob J. van Glabbeek. The linear time–branching time spectrum. In *International Conference on Concurrency Theory*, pages 278–297. Springer, 1990. 8

[vG93] Rob J. van Glabbeek. The linear time–branching time spectrum II. In *International Conference on Concurrency Theory*, pages 66–81. Springer, 1993. 1, 8, 16, 50, 52

[vG17] Rob J. van Glabbeek. A branching time model of CSP. *CoRR*, 2017. 1, 19, 21, 22, 26, 28, 34

[vGP08] Rob J. van Glabbeek and Bas Ploeger. Correcting a space-efficient simulation algorithm. In *International Conference on Computer Aided Verification*, pages 517–529. Springer, 2008. 47

[vGW96] Rob J. van Glabbeek and W. Peter Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM (JACM)*, 43(3):555–600, 1996. 35, 53

[VM00] Marc Voorhoeve and Sjouke Mauw. Impossible futures and determinism. Computing science reports Vol. 0014, Technische Universiteit Eindhoven, Eindhoven, 2000. 23

[VM01] Marc Voorhoeve and Sjouke Mauw. Impossible futures and determinism. *Information Processing Letters*, 80(1):51–58, 2001. 13, 23

[WHH$^+$06] Ralf Wimmer, Marc Herbstritt, Holger Hermanns, Kelley Strampp, and Bernd Becker. Sigref: A symbolic bisimulation tool box. In Susanne Graf and Wenhui Zhang, editors, *Automated Technology for Verification and Analysis: 4th International Symposium, ATVA 2006, Beijing, China, October 23-26, 2006. Proceedings*, pages 477–492. Springer Berlin Heidelberg, 2006. 46, 47

[Wim11] Ralf Wimmer. *Symbolische Methoden für die probabilistische Verifikation – Zustandsraumreduktion und Gegenbeispiele.* Dissertation, Albert-Ludwigs-Universität Freiburg, Freiburg im Breisgau, Germany, January 2011. 46, 47

Nomenclature

iff	“If and only if” in definitions
$\lvert A\rvert$	Number of elements of A
Δ_A	Identity relation on A
Σ^*, Σ^ω, Σ^∞	Language of finite, infinite, finite-or-infinite words over alphabet Σ
2^A	Power set of A (set of subsets)
$A_{/\equiv_X}$	Quotient set of A by equivalence $\equiv_X$ (notation also used for quotient transition systems, Def. 4.3)
$\mathcal{R}^{-1}$	Inverse relation of $\mathcal{R}$
$\mathcal{R}^*$	Reflexive transitive closure of relation $\mathcal{R}$
$\equiv_C, \sqsubseteq_C$	Contrasimilarity, contrasimulation preorder (Def. 2.13)
$\equiv_{CS}, \sqsubseteq_{CS}, \equiv_{CS^c}$	Coupled similarity, coupled simulation preorder (Def. 3.1), rooted coupled similarity (Def. 3.21)
$\equiv_B$	Strong bisimilarity (like weak bisimilarity, Def. 2.12, but with $\rightarrow$-answers instead of $\overset{\hat{}}{\Rightarrow}$-answers)
$\equiv_S, \sqsubseteq_S$	Strong similarity (like weak similarity, Def. 2.7, but with $\rightarrow$-answers instead of $\overset{\hat{}}{\Rightarrow}$-answers), strong simulation preorder
$\equiv_{WB}, \equiv_{WB^c}$	Weak bisimilarity (Def. 2.12), rooted weak bisimilarity (Def. 2.22)
$\equiv_{WS}, \sqsubseteq_{WS}$	Weak similarity, weak simulation preorder (Def. 2.7)
$\xrightarrow{\alpha}$	Transition relation (Def. 2.1)
$\overset{\alpha}{\not\rightarrow}$	Absence of a transition (Disabledness of α)
$\overset{\alpha}{\Rightarrow}$	Weak transition relation (Def. 2.2)
$\overset{\hat{\alpha}}{\Rightarrow}$	Weak transition relation reflexive on τ (Def. 2.2)
$\overset{\hat{\alpha}}{\Rightarrow\!\!\triangleright}$	Weak delay transition relation reflexive on τ (Def. 3.14)
$\rightarrowtail$	Game move (Def. 2.25)