Structural Operational Semantics for Weak Bisimulations

Bard Bloom*

TR 93-1373
August 1993

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

* bard@cs.cornell.edu Supported by NSF grant (CCR-9003441).

Structural Operational Semantics for Weak Bisimulations

Bard Bloom*
Cornell University

August 10, 1993

Abstract

In this study, we present rule formats for four main notions of bisimulation with silent moves. Weak bisimulation is a congruence for any process algebra defined by *WB cool rules*; we have similar results for rooted weak bisimulation (Milner's "observational equivalence"), branching bisimulation, and rooted branching bisimulation. The theorems stating that, say, observational equivalence is an appropriate notion of equality for CCS are corollaries of the results of this paper. We also give sufficient conditions under which equational axiom systems can be generated from operational rules. Indeed, many equational axiom systems appearing in the literature are instances of this general theory.

1 Introduction

Process algebras serve the same role in concurrency that the λ-calculus does in sequential block-structured languages, isolating the essential features of the area while abstracting away inessential details. As there are a vast variety of fundamentally different concurrent settings (varying process synchrony, communication mechanism, failure, and so forth), there is a need for many process algebras. Developing the right basic theory for the first few process algebras was a labor of several years. This paper is part of a continuing investigation in the metatheory of process algebra, with the goal of developing a body of mathematics to make the development and use of process algebras simpler.

The multiplicity of computational models is reflected in several ways in process algebras. Most obviously, the operations in the algebra must match the operations in the model: describing broadcasting systems requires broadcasting operations in the algebra. Somewhat more subtly, the basic notion of *process equivalence* — that is, the definition of what it means for two processes p and q to mean the same thing — will also vary with the model of concurrency. Notions of equivalence which work perfectly for point-to-point communication (*e.g.*, failures equivalence [BHR84]) are not correct for broadcasting.

*bard@cs.cornell.edu. Supported by NSF grant (CCR-9003441).

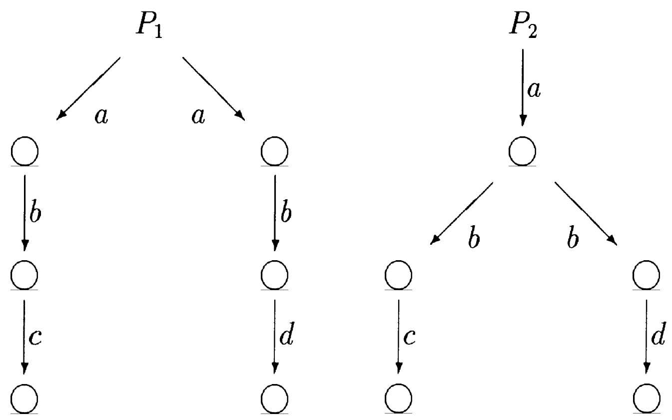


Figure 1: Linear vs. Branching Time

A wide range of process equivalences have been proposed and used [vG93]; in process algebra, they range from *strong bisimulation* (the finest reasonable notion in this setting) to *weak partial trace equivalence* (the coarsest notion, and not often applicable).

There are many ways to characterize equivalences, two of which are relevant to our discussion. In many process algebras, there is a *silent move* τ, an action which represents internal computation. Some equivalences, called *strong* equivalences, treating τ as just another action, without special status. Other equivalences, the *weak* equivalences, do their best to ignore τ actions that are computationally irrelevant. For example, let $\tau\tau$ be a process which does two τ actions and then stops. Then $\tau\tau$ and τ are distinguished by most strong equivalences, but identified by most weak ones. Some τ actions cannot be ignored; the process $a+b$ is able to accept either an a or a b signal, but the process $\tau a + \tau b$ autonomously chooses which it will accept, and will reject the other.

Another important characterization of process equivalences is *linear* and *branching* time [Gla90]. Loosely, linear-time equivalences observe the behavior of a process as it runs along a single execution, with very limited ability to observe possible alternate paths of computation. Branching-time equivalences consider the whole tree of possible executions. For example, the processes P_1 and P_2 of Figure 1 are linear-time equivalent in all standard linear-time equivalences; for example, they have the same traces, *abc* and *abd*. They are distinguished by most branching-time equivalences; P_1 makes an important decision on its first step, while P_2 delays it until its second.

Most process algebras use the notion of a *transition*. The relation $p \xrightarrow{a} q$ says that the process p can perform the atomic action a and thereafter behave like the process q. Most process algebras are given by a *Structural Operational Semantics* (SOS), defining the transition relation by induction on the structure of the term p.

Given the multiplicity of models of concurrency, notions of equivalence, and process algebras, it seems worthwhile to study the *metatheory* of process algebra: theorems which apply to large families of languages, rather than merely specific ones. Such theories will greatly facilitate the development of new process algebras. For example, [BIM90, GV92, Gro90] show that *any* language defined by suitable forms of structured operational rules enjoy certain desirable properties: in particular, all such languages respect strong bisimulation. In [ABV92a], we extend this theory: any language described by the *GSOS* rules of [BIM90] can in fact be given an equational axiom system with a single infinitary induction principle, which is complete for proving equalities between programs.

Most of these studies have treated strong equivalences. There have only been a few studies concerning methatheory of SOSses for weak equivalences. Most of these studies have been aimed at calculating the finest appropriate weak process equivalence. [Blo90] introduced several technical tools used in later studies, but its notion of equivalence had some ability to count τ moves and was therefore judged inappropriate as a weak process equivalence. [Uli92] was considerably more appealing as a study of equivalences, describing the class of *ISOS* rules, and precisely characterizing the notion of process equivalence they generate. Ulidowski's notion is appealing in many ways, but misses certain important uses of process algebras: *e.g.*, processes with polling loops can diverge, and Ulidowski's theory doesn't place many requirements on divergent processes.

In this study, we give SOS rule formats for important weak variants of bisimulation, *weak bisimulation*, and *branching bisimulation*, and their (more useful) rooted versions. Weak bisimulation [Mil81] is perhaps the most obvious variant of bisimulation designed to ignore τ moves; its theory is not quite as clean as that of strong bisimulation. Branching bisimulation [vGW89] is a finer notion, with better algebraic properties. Neither notion is quite right for process algebra: they are not congruences with respect to certain operations, mainly nondeterministic choice $+$. Both can be slightly modified, to *rooted weak bisimulation* (Milner's *observational equivalence* [Mil89a]) and *rooted branching bisimulation*, capable of handling $+$ and maintaining their other properties. We discuss the implications of these rule formats on equational axiom systems.

1.1 Strong and Weak Bisimulation

Strong bisimulation [Par81, Mil83, Mil84] is the finest generally-accepted notion of process equivalence in this setting. Informally, two processes are strongly bisimilar iff they make the same decisions at the same times. Formally,

Definition 1.1 *A binary relation $\smile$ between processes is a* strong bisimulation relation *iff, whenever $p \smile p'$, then*

- *If $p \xrightarrow{a} q$ for some action a and process q, then $p' \xrightarrow{a} q'$ for some process q' such that $q \smile q'$.*
- *Vice versa; that is, if $p' \xrightarrow{a} q'$ for some action a and process q', then $p \xrightarrow{a} q$ for some process q such that $q \smile q'$.*

Processes p and p' are strongly bisimilar, *$p \leftrightarroweq p'$, iff there is some strong bisimulation relation $\smile$ such that $p \smile p'$.*

Strong bisimulation enjoys a rich and powerful theory. There are complete equational axiom systems (some requiring induction principles) for a wide variety of theories. [BW90, Mil89b, ABV92a]. There is an elegant equivalent logical characterization via *Hennessy-Milner Logic* [HM85]. Definition 1.1 gives a useful method for showing two real programs equivalent [CPS89, WBB92]: one guesses a relation $\smile$ relating the programs, and verifies that it is indeed a strong bisimulation relation. Checking this is local, involving only one step of computation. Despite some theoretical concerns (*e.g.*, strong bisimulation is *too* strong, capable of distinguishing processes which ought to be identified [Abr87, BIM88, BM90]), strong bisimulation is a central part of concurrency theory.

1.1.1 Weak Bisimulation

Most process algebras, including CCS and ACP, have a *silent* or *hidden* action, τ. This action marks internal computation: for example, processes will take τ steps as they compute; and when processes communicate on a hidden channel, all that is visible outside is a τ action. The informal intent is that τ's are (mostly) irrelevant, and that processes which differ only in the number and positioning of τ's are equivalent. As stated before, we would like to have τ and $\tau\tau$ identified.

Strong bisimulation, and strong equivalences in general, do not meet this informal intent. τ's are as visible as any other action, and $\tau \not\underline{\leftrightarrow} \tau\tau$. Accordingly, several researchers have defined weak versions of bisimulation, which pay less attention to τ's but still have the essential flavor of bisimulation. The first such relation, called simply *weak bisimulation*, is based on the *weak transition relation* $\xRightarrow{a}$, defined by:

Definition 1.2 *We define multi-step transition relations $p \xrightarrow{s} q$ by $p \xrightarrow{\epsilon} p$, and if $p \xrightarrow{s} q$ and $q \xrightarrow{a} r$ then $p \xrightarrow{sa} r$. This is extended to sets of string S in existentially: $p \xrightarrow{S} q$ iff $\exists s \in S . p \xrightarrow{s} q$. Let $\tau^* = \{\varepsilon, \tau, \tau\tau, \ldots\}$.*

Finally, we define $\xRightarrow{\alpha}$ for $\alpha \in \mathsf{Act}_\tau$ to be $\xrightarrow{\tau^*\alpha\tau^*}$ if $\alpha \in \mathsf{Act}$, and $\xrightarrow{\tau^*}$ if $\alpha = \tau$.

Weak bisimulation is defined like strong bisimulation, using the relation $\xRightarrow{\alpha}$ for $\alpha \in \mathsf{Act}_\tau$ instead of $\xrightarrow{a}$ for $a \in \mathsf{Act}$.

Definition 1.3 *A relation $\smile$ between processes is a* weak bisimulation relation *iff, whenever $p \smile p'$ and $p \xRightarrow{\alpha} q$ for some $\alpha \in \mathsf{Act}_\tau$ and process q, then there is a q' such that $p' \xRightarrow{\alpha} q'$ and $q \smile q'$, and vice versa.*

Processes p and p' are weakly bisimilar*, $p \underline{\Leftrightarrow} p'$, if there is a weak bisimulation relation $\smile$ with $p \smile p'$.*

For example, $a \underline{\Leftrightarrow} \tau a$. Note that $a \not\underline{\leftrightarrow} \tau a$, as the one can take an a-step and the other cannot. Strong bisimulation is mathematically quite nice, but weak bisimulation is somewhat less so. For example, strong bisimulation is a congruence with respect to all CCS (and a vast range of other) operations. Weak bisimulation is a congruence with for most CCS operations, but not for $+$. For example, $a \underline{\Leftrightarrow} \tau a$, but $a + b \not\underline{\Leftrightarrow} \tau a + b$; the latter process can take a τ step to the process b, which the former cannot match. The standard approach to fixing this is to take *weak bisimulation congruence*, also known as *rooted weak bisimulation*, rooted τ-bisimulation [BW90], and rather anomalously as *observational equivalence* [Mil84, Mil89a]. This relation is defined as follows:

Definition 1.4 *Processes p and p' are* rooted weakly bisimilar, *$p \,\underline{\leftrightarrow}_r\, p'$, iff whenever $p \xrightarrow{\beta} q$ for some $\beta \in \mathsf{Act}_\tau$, then $p' \xrightarrow{\tau^*\beta\tau^*} q' \,\underline{\leftrightarrow}\, q$, and vice versa.*

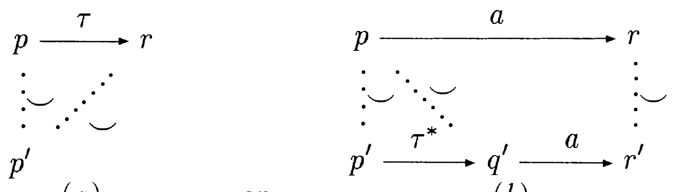
This is a direct definition of $\underline{\leftrightarrow}_r$ in terms of $\underline{\leftrightarrow}$; and it differs from $\underline{\leftrightarrow}$ only when $\beta = \tau$: the rooted relation insists that p' take at least one τ-step. Rooted weak bisimulation works correctly: all CCS, and indeed most other process algebra operations respect it.

1.1.2 Branching Bisimulation

Detailed philosophical justification for branching bisimulation may be found in [vGW89, vG93]. Briefly, branching bisimulation is the finest weak analog of bisimulation in the testing scenarios of [vG93], and the finest notion which admits an expansion theorem. Furthermore, the informal description of bisimulation, "two processes making the same decisions at the same time," is imperfectly captured by the formal definition. Consider the weakly bisimilar processes $p_1 = a(\tau x + y)$ and $p_2 = a(\tau x + y) + ax$. The process p_2 can, on an a-transition, choose to forgo the chance of performing y. The process p_1 cannot make the same choice in just the a-move; discarding the y requires an a-move followed by a τ. This conceptual imperfection is fixed by *branching bisimulation*; p_1 and p_2 are not branching bisimilar.

Definition 1.5 *$\smile$ is a branching bisimulation relation if, for all actions $\beta \in \mathsf{Act}_\tau$ and for all $p \smile p'$, then*

1. *If $p \xrightarrow{\beta} r$, then either:*
   - *(a) $\beta = \tau$ and $r \smile p'$, or*
   - *(b) there are q', r' such that $p' \xrightarrow{\tau^*} q' \xrightarrow{\beta} r'$, $p \smile q'$, and $r \smile r'$:*



(a) or (b)

2. *Vice versa.*

As usual, p and p' are branching bisimilar, $p \,\underline{\leftrightarrow}_b\, p'$, if there is a branching bisimulation relation relating them.

Branching bisimulation suffers from the same congruence problem as weak bisimulation: $a \,\underline{\leftrightarrow}_b\, \tau a$, but $a + b \,\not\underline{\leftrightarrow}_b\, \tau a + b$. The solution is a bit finer than for weak bisimulation. Rooted branching bisimulation does one step of strong bisimulation, then turns into branching bisimulation.

Definition 1.6 *Two processes p and q are* rooted branching bisimilar *if, whenever $p \xrightarrow{a} q$, then $p' \xrightarrow{a} q'$ and $q \,\underline{\leftrightarrow}_b\, q'$; and vice versa.*

The theory of $\leftrightarroweq_{rb}$ is quite nice: for example, rooted branching bisimulation is a congruence with respect to CCS, has a complete axiom system on finite processes, and so forth.

We refer to the relations of weak and branching bisimulation, rooted and otherwise, collectively as "silent bisimulations."

1.2 Structured Operational Semantics and Process Equivalences

Most process calculi are given behavior by *structured operational semantics* (SOS), rules which define the behavior of composite processes in terms of the behavior of their components. A well-designed SOS rule system has many of the advantages of denotational semantics; e.g, structural induction is a viable proof technique, as we will see repeatedly in this study. Furthermore, SOS rules are generally fairly easy to read. For example, the interleaving parallel operation $p \parallel q$ is easily described by SOS rules: for each action α, there are rules:

$$\frac{x_1 \xrightarrow{\alpha} y_1}{x_1 \parallel x_2 \xrightarrow{\alpha} y_1 \parallel x_2} \qquad \frac{x_2 \xrightarrow{\alpha} y_2}{x_1 \parallel x_2 \xrightarrow{\alpha} x_1 \parallel y_2} \tag{1}$$

The purpose of this study, like many others in this line of research, is to determine the relation between the SOS definition of a process algebra, and the equational theory. We will give some quite general rule formats which guarantee that process algebras respect the four silent bisimulations described in Section 1.1, and a new variant which may be of some use.

The SOS definition of a process algebra or programming language describes the intended behavior of processes. It is not intended to suggest the preferred implementation of the language, any more than the β-rule is intended suggest the preferred implementation of functional languages. It is often useful to specify languages using rules which appear rather unreasonable, knowing that the implementation need not reflect the rules.

Two common categories of "unreasonable" or "unimplementable" features of rules are *copying* of processes and *negative tests*. Both features arise quite naturally in the simplest specifications of standard operations. Our rule formats for weak process equivalences allow copying in a fairly general way. However, negative tests defy both implementation and weak bisimulations, and are thus excluded from our rule format.

1.2.1 Copying

For example, one of the rules of the specification of a `while` loop is:

$$\frac{x_1 \xrightarrow{\texttt{true}} y_1}{\textsf{while } x_1 \textsf{ do } x_2 \xrightarrow{\tau} (x_2\,; \textsf{while } y_1 \textsf{ do } x_2)}$$

That is, if the test x_1 signals "true," the loop body is executed once and the loop is re-executed. This specification makes a copy of the loop body x_2. A typical implementation [Plo81, WBB92] does not copy x_2; it simply uses a loop around the code.

A similar operation is Milner's $!p$, which effectively turns p into a reentrant server:

$$\frac{x \xrightarrow{a} x'}{!x \xrightarrow{a} x' \| !x}$$

This spawns a new server x' to handle each request a. This will probably be implemented by spawning new processes on each machine running x — which may, in general, be a distributed system [BC90, BCG91].

Many other operations can be specified by copying rules. For example, a process implementing a distributed service might produce a signal ℓ if its load gets too high, piggybacked on top of its normal service message. The system's response might be to bring up another processor running the original server code, running in parallel with the original process. Such a service could conveniently be specified by rules including:

$$\frac{x_1 \xrightarrow{a} y_1}{\mathsf{server}(x_1, x_2) \xrightarrow{a} \mathsf{server}(y_1, x_2)} \qquad \frac{x_1 \xrightarrow{a} y_{11}, x_1 \xrightarrow{\ell} y_{12}}{\mathsf{server}(x_1, x_2) \xrightarrow{a} \mathsf{server}(y_{11}|x_2, x_2)}$$

where a ranges over server actions, the x_1 argument represents the current state of the server, and the x_2 argument represents the initial state of a new server processor.

1.2.2 Negative Rules vs. Weak Bisimulation

For strong notions of process equivalence, many operations are best described using *negative rules*; *viz.*, those with antecedents of the form $x \xrightarrow{a}\!\!\!\!\!\not\;\;$, which are satisfied if x can*not* perform an a action. Consider the following "full sequencing" operation $p\,;q$, which runs p until it stops, then runs q:

$$\frac{x_1 \xrightarrow{a} y_1}{x_1\,; x_2 \xrightarrow{a} y_1\,; x_2} \qquad \frac{x_2 \xrightarrow{a} y_2,\ \forall b.\left(x_1 \overset{b}{\not\longrightarrow}\right)}{x_1\,; x_2 \xrightarrow{a} y_2} \tag{2}$$

It is impossible to define this operation without negative rules, though most process algebras define approximations which are adequate for programming, and much easier to implement. There are several other reasonable operations which can only be defined with negative rules; *e.g.*, *polling* operations which branch on whether or not a process is ready to communicate.

Unfortunately, negative rules seem incompatible with weak process equivalences. One of the fundamental laws of silent bisimulations is KFAR, *Koomen's Fair Abstraction Rule* [BW90]. Intuitively, this rule says that that nondeterministic choice can be implemented by a polling loop. Indeed, delay-insensitive implementations of process algebra based languages [WBB92] implement selective communication (the main practical application of nondeterministic choice in this setting) via polling loops. As these implementations are correct up to weak or branching bisimulation and (probably) not much stronger, KFAR is practically and philosophically important.

However, KFAR seems incompatible with negative rules. A basic consequence of KFAR is that, if we have two processes p_1 and p_2 such that $p_1 \underset{\tau^*}{\overset{\tau^*}{\rightleftarrows}} p_2$, then p_1 and p_2 ought to be considered the same process. That is, the decision of whether or not $p_1 \overset{a}{\not\longrightarrow}$ should involve all such p_2. As $\left\{p_2 | p_1 \underset{\tau^*}{\overset{\tau^*}{\rightleftarrows}} p_2\right\}$ can be unmanageable — infinite and not even recursive — testing p_1's ability to do an a is necessarily challenging.

We are not aware of any way to interpret negative tests $x \overset{a}{\not\longrightarrow}$ in a way that respects silent bisimulations and has any chance of being implementable. Consider the weakly bisimilar processes of Figure 2.

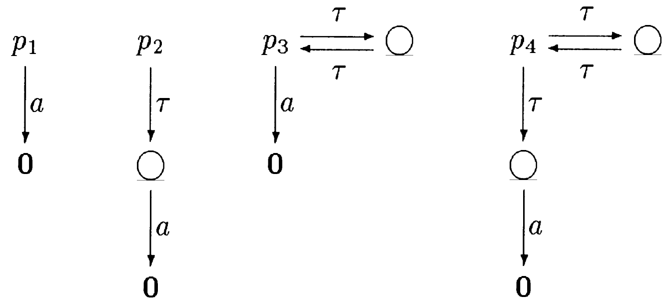


Figure 2: Weakly Bisimilar Processes

These processes must be indistinguishable in any language respecting weak bisimulation, as they are weakly bisimilar. The most obvious interpretation of $x \overset{a}{\not\longrightarrow}$ is simply to test that x has no a transition. With this interpretation, $\neg(p_1 \overset{a}{\not\longrightarrow})$ but $p_2 \overset{a}{\not\longrightarrow}$; a rule which can test for $x \overset{a}{\not\longrightarrow}$ could distinguish these equivalent processes.

A variant interpretation which has been proposed several times [Vaa91, Uli92] is that negative rules only be interpreted in *stable states*; that is, x satisfies the test $x \overset{a}{\not\longrightarrow}$ iff x can perform τ-moves to reach a state which has neither a nor τ transitions. This is implementable (by running the process until such time as it reaches a stable state), and does address the primary difficulty with KFAR; the set of processes $\overset{\tau^*}{\underset{\tau^*}{\rightleftarrows}}$ to a stable process p is just $\{p\}$. For example, $p_4 \overset{b}{\not\longrightarrow}$, as $p_4 \xrightarrow{\tau} a$ which is stable and cannot perform a b. This interpretation *does* work for Ulidowski's weak equivalence, but it fails for weak bisimulation. For example, $p_1 \overset{b}{\not\longrightarrow}$ in this interpretation, but $\neg(p_3 \overset{b}{\not\longrightarrow})$ as p_3 is not in a stable state, and indeed cannot reach one by performing τ moves.

As there seems to be no good interpretation of negative tests, we do not include them in our languages for weak or branching bisimulation. The languages for rooted branching bisimulation can tolerate them to a limited degree.

1.3 Results

In this paper, we give rule formats for the four silent bisimulations of Section 1.1, as well as a variant of rooted weak bisimulation. The basic intuition behind the classes is fairly similar — basically, we require operations to wait calmly for their operands to be ready. We call this property *coolness*: we thus have WB cool languages, which respect Weak Bisimulation; RBB cool languages respecting Rooted Branching Bisimulation, and so forth. Weak bisimulation is discussed in Section 3, rooted weak bisimulation and a technically convenient variant called "strongly rooted weak bisimulation" in Section 4, branching bisimulation in Section 5, and rooted branching bisimulation in Section 6. It is worth noting that the methods used for branching bisimulation are very similar to those for weak bisimulation, though the proofs have twice as many cases.

In Section 7, we discuss the implication of these rule formats for equational theories. It is well-known that neither weak nor branching bisimulation has appealing (*e.g.*, finite) axiom systems for some basic operations like parallel composition. However, the rooted versions generally admit equational axiom systems. The method of [ABV92a] for generating equational axiom systems applies to all SRWB cool and RBB cool languages (and preserves their SRWB cool or RBB cool character.) We give decidable sufficient conditions under which RWB cool languages have [ABV92a]-style axiom systems. The equational axiom systems the [ABV92a] method generates are not complete for any decent programming language.

1.3.1 Useful Corollaries: simply cool languages

In the rest of the paper, our results are quite general, and accordingly hard to apply. We expect that the following straightforward corollaries of our main theorems will prove useful in most cases appearing in practice; they are enough to cover most CCS-like process calculi known to date.

Definition 1.7 *A language $\mathcal{L}$ defined by SOS rules is* simply RB cool *if*

1. *All rules ρ for all operation symbols f of $\mathcal{L}$ have the form:*

$$\rho = \frac{\left\{x_i \xrightarrow{a_i} y_i | i \in I(\rho)\right\}}{f(x_1, \ldots, x_n) \xrightarrow{b} t}$$

   where $I(\rho)$ is a set of numbers telling which arguments of f take actions under rule ρ.

2. *For all rules ρ, and all $i \in I(\rho)$, $\mathcal{L}$ contains a* patience *rule:*

$$\frac{x_i \xrightarrow{\tau} y_i}{f(\vec{x}) \xrightarrow{\tau} f(x_1, \ldots, x_{i-1}, y_i, x_{i+1}, \ldots, x_n)}$$

3. *No rules have hypotheses mentioning τ's, except the patience rules required by clause 2.*

The definition for weak bisimulation differs only slightly:

Definition 1.8 *A language is* simply WB cool *iff it is simply RB cool, and furthermore for every t in clause 1, x_i does not appear in t when $i \in I(\rho)$.*

CCS without + is a simply WB cool (and, *a fortiori*, simply RB cool) language.

Rule formats for rooted silent bisimulations are built by partitioning the rules into two classes: *tame* operations, which respect the silent bisimulation from the previous definitions, and *wild* operations, which exploit the rootedness condition for one step and then evolve into tame operations.

Definition 1.9 *A GSOS language $\mathcal{L}$ is* simply RWB cool *if the operations in $\mathcal{L}$ can be partitioned into* tame *and* wild *operations, such that:*

1. *The targets of all rules (the t's above) only mention tame operations.*
2. *The sublanguage consisting only of tame operations is WB cool.*
3. *Every rule ρ for a wild operation has the form*

$$\frac{\left\{x_i \xrightarrow{a_i} y_i | i \in I(\rho)\right\}}{f(\vec{x}) \xrightarrow{b} t}$$

4. *For each such rule, there are rules (possibly derived rules):*

$$\frac{x_i \xrightarrow{\tau} y_i}{f(\vec{x}) \xrightarrow{\tau} t[x_i := y_i]} \qquad \frac{\left\{x_i \xrightarrow{a_i} y_i | i \in I(\rho)\right\}}{t \xrightarrow{b} t[\vec{x} := \vec{y}]} \qquad \frac{x_i \xrightarrow{\tau} y_i}{t \xrightarrow{\tau} t[x_i := y_i]}$$

For example, CCS is a simply RWB cool language. $+$ is a wild operation; all others are tame. Pick one of the rules for $+$, say

$$\frac{x_1 \xrightarrow{a} y_1}{x_1 + x_2 \xrightarrow{a} y_1}$$

If we choose $t = x_1$, then we must have the following rules and derived rules:

$$\frac{x_1 \xrightarrow{\tau} y_1}{x_1 + x_2 \xrightarrow{\tau} y_1} \qquad \frac{x_1 \xrightarrow{a} y_1}{x_1 \xrightarrow{a} y_1} \qquad \frac{x_1 \xrightarrow{\tau} y_1}{x_1 \xrightarrow{\tau} y_1}$$

Indeed, these are all valid CCS rules or derived rules, and hence $+$ is a suitable wild operation.

Rooted branching bisimulation is much easier, largely due to the stronger first-step condition.

Definition 1.10 *A GSOS language $\mathcal{L}$ is* simply RBB cool *iff the operation symbols can be partitioned into* tame *and* wild *operations, such that*

1. *The sublanguage of $\mathcal{L}$ consisting of only tame operations is simply RB cool, and*
2. *Let $f(\vec{x}) \xrightarrow{c} t$ be the conclusion of any rule of $\mathcal{L}$. Then t mentions only tame operations.*

As one would hope, we have the following corollary to most of the main theorems of this paper.

Corollary 1.11

1. *If $\mathcal{L}$ is simply WB cool, then weak bisimulation is a congruence for $\mathcal{L}$.*

2. *If $\mathcal{L}$ is simply RWB cool, then rooted weak bisimulation is a congruence for $\mathcal{L}$.*

3. *If $\mathcal{L}$ is simply RB cool, then branching bisimulation is a congruence for $\mathcal{L}$.*

4. *If $\mathcal{L}$ is simply RBB cool, then rooted branching bisimulation is a congruence for $\mathcal{L}$.*

We also have

Corollary 1.12 *If $\mathcal{L}$ is a simply RBB cool language, then the methods of [ABV92a] generate a simply RBB cool, conservative extension of $\mathcal{L}$, and a set of equations which are sound for rooted branching bisimulation and complete for strong bisimulation.*

A similar fact generally holds for simply RWB cool languages, but it is harder to state in full generality.

2 Preliminaries

In this section, we review some standard and mostly-standard definitions in process algebra.

2.1 Notation

We use fairly standard mathematical notation, though we write $A \pitchfork B$ for $A \cap B = \varnothing$. If $A \cup B = C$ and $A \pitchfork B$, then A and B *partition* the set C; unlike some authors, we allow A and B to be empty.

We often introduce eccentric notations for tuples; e.g., we write write the tuple $\langle t, a, t \rangle$ as $t \xrightarrow{a} t'$, when we are interpreting it as a formula.[1]

We use vector notation, $\vec{x}$, to denote a finite sequence of items named $x_1, x_2, \ldots, x_n$ for some n. We assume that the sequences are of the correct lengths, which are always clear from context or irrelevant. Operations and relations on vectors are to be interpreted pointwise. For example, the equation $\vec{x} = \vec{y}$ means that $\vec{x}$ and $\vec{y}$ have the same length, and that $x_i = y_i$ for each i. To add to the confusion, $\vec{x}$ is occasionally treated as the *set* $\{x_1, x_2, \ldots, x_n\}$.

We denote the empty string by ε. If τ is a symbol, $\tau^+ = \{\tau, \tau\tau, \tau\tau\tau, \ldots\}$ and $\tau^* = \{\varepsilon\} \cup \tau^+$. Similarly, if S is a set of actions or string, then S^* is the Kleene closure of S, the set of all finite strings of elements of S.

2.2 Structural Operational Semantics

As we are investigating the comparative anatomy of process algebras and their definitions, we discuss their definitions in some generality. We are concerned with *GSOS* and related classes of rules; to discuss these, we need some notation.

A process algebra language, or simply "language," includes a finite set of *actions*, and a finite algebraic signature. In this study, we assume that the set of actions Act_τ contains a distinguished element τ, the *silent action*; the remaining actions Act are not required to

[1] When necessary, we add an extra component to the tuple; if we are using the notation $t \overset{a}{\Longrightarrow} t'$ as well, we assume that the representations are disjoint. This fine point is never relevant.

have any significance. The definitions of signature, term, and so forth are fairly standard. A signature Σ is a finite function from a set of *operation symbols* $f, g, h, \ldots$ to natural numbers, their *arities*. Terms t are inductively built from an infinite set of variables $x, y, z, \ldots$ by application of function symbols:

$$t ::= x \mid f(t_1, \ldots, t_n) \quad \text{where } \Sigma(f) = n$$

A term is *closed* if it contains no variable symbols; process algebras generally contain nullary operators, such as the stopped process $\mathbf{0}$, and thus have closed terms. This is important, as closed terms are *processes* which can be executed. There are no binding operators, and in particular no $\mathsf{rec}[x \Leftarrow P]$ operator for recursive process definition a la CCS. Our definitional schema are powerful enough to include general nontermining computations, even without recursion; and any program written with $\mathsf{rec}[x \Leftarrow P]$ over any of our calculi can be written without $\mathsf{rec}[\cdot \Leftarrow \cdot]$ in a suitable variant calculus.

Let $\mathsf{Procs}(\mathcal{L})$ be the set of closed terms of the language $\mathcal{L}$, and $\mathsf{Terms}(\mathcal{L})$ the set of all terms. When $\mathcal{L}$ is clear from context, we simply write Procs and Terms. A term is *univariate* if no variable appears in it more than once.

Actions a are elements of some finite nonempty set Act. Transition formulas describe possible transitions. A *positive transition formula* is a triple, written $t \xrightarrow{a} t'$, where t and t' are terms, and a is an action symbol. A *negative transition formula* is a pair $t \stackrel{a}{\nrightarrow}$. A formula is *primitive* if t and t' are variables: $x \xrightarrow{a} x'$ and $x \stackrel{b}{\nrightarrow}$.

A *GSOS rule* is a pair $\left\langle \mathcal{H}, f(\vec{x}) \xrightarrow{c} t \right\rangle$, where $\mathcal{H}$ is a set of primitive transition formulas, subject to certain conditions. GSOS rules are usually written:

$$\frac{\bigcup_i \left\{ x_i \xrightarrow{a_{ij}} y_{ij} \middle| 1 \leq j \leq m_i \right\} \cup \bigcup_i \left\{ x_i \stackrel{b_{ij}}{\nrightarrow} \middle| 1 \leq j \leq n_i \right\}}{f(x_1, \ldots, x_n) \xrightarrow{c} t}$$

The conditions are: the variables $\vec{x}, \vec{y}$ are all distinct, and the variables in t are at most $\vec{x}$,$\vec{y}$.

In this study, we take the point of view that rules bind all variables appearing in them. In particular, rules which differ only by a bijective renaming of variable names are equivalent.

We use the following terminology to describe parts of a GSOS rule:

- $\mathsf{ante}(\rho) = \mathcal{H}$ is the set of *antecedents*.
- $\mathsf{cons}(\rho) = f(\vec{x}) \xrightarrow{c} t$ is the *consequent*.
- $\mathsf{source}(\rho) = f(\vec{x})$ is the *source*.
- $\mathsf{target}(\rho) = t$ is the *target*.
- c is the *action*.
- The variables x_i are the *source variables* $\mathsf{SourceVars}(\rho)$.
- The variables y_{ij} are the *target variables* $\mathsf{TargetVars}(\rho)$.

Symbol	Usage
a, b, c, d	Actions other than τ; elements of Act
α, β	Actions or τ; elements of Act_τ.
ζ	Actions, τ, or ε; element of $\mathsf{Act}_{\tau,\varepsilon}$
p, q, r, s	Processes
t, u, v	Process terms
x, y, z	Variables

Figure 3: Conventions

For example, the interleaving parallel operation $p \| q$ is defined by the $2 \cdot |\mathsf{Act}|$ rules of (1). Many process algebras have the operations of *action prefixing* and *nondeterministic choice*. For each action α, αp is a process which performs an α and thereafter behaves like p; for all p and q, $p + q$ may behave like either p or q. These have the rules (for each α):

$$\frac{}{\alpha x \xrightarrow{\alpha} x} \qquad \frac{x_1 \xrightarrow{\alpha} y_1}{x_1 + x_2 \xrightarrow{\alpha} y_1} \qquad \frac{x_2 \xrightarrow{\alpha} y_2}{x_1 + x_2 \xrightarrow{\alpha} y_2} \tag{3}$$

If $\mathcal{R}$ is a rule format (that is, a set of rules), an $\mathcal{R}$ *language* is a finite signature Σ of operations together with a finite set of rules $R \subseteq \mathcal{R}$, where all operations in R are in Σ and used with the correct arity.

Null transitions, $x \xrightarrow{\varepsilon} y$, will provide great notational convenience later on: they are intended to mean that x and y are bound to the same term. This can be trivially expressed in the GSOS format by using x everywhere y appears: the following are equivalent.

$$\frac{}{\alpha x \xrightarrow{\alpha} x} \qquad \frac{x \xrightarrow{\varepsilon} y}{\alpha x \xrightarrow{\alpha} y}$$

We set $\mathsf{Act}_{\tau,\varepsilon} = \mathsf{Act} \cup \{\tau, \varepsilon\}$, use ζ to range over $\mathsf{Act}_{\tau,\varepsilon}$, and with a slight pun we use the notation $x \xrightarrow{\zeta} y$ for $\zeta \in \mathsf{Act}_{\tau,\varepsilon}$ to range over $x \xrightarrow{\alpha} y$ and $x \xrightarrow{\varepsilon} y$.

Definition 2.1 *A rule or language which is GSOS except for possibly having null transitions as antecedents is said to be in* GSOS(ε) *format.*

Notation 2.1 *In this study we work with a variety of objects including variables: terms, transition formulas, rules, and so forth. For any such object Ω, $\mathsf{Vars}(\Omega)$ is the set of variables in Ω. If $x \in \mathsf{Vars}(\Omega)$ and ξ is a suitable object, then $\Omega[x := \xi]$ is Ω with ξ substituted for x. Only rules bind variables, and that at the top level only; so the definition is not subtle.*

It is often convenient to work with *substitutions* (traditionally also called *instantiations* when they refer to rules), which are partial functions σ from variable names to Procs. We extend σ to terms t with $\mathsf{Vars}(t) \subseteq \mathsf{dom}(\sigma)$ by:

$$\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n)),$$

and similarly to other constructs including variables.

If ψ is a transition formula, $\dot{\rightsquigarrow}\ \subseteq \mathsf{Procs} \times \mathsf{Act} \times \mathsf{Procs}$, and σ is a substutition, and $\mathsf{Vars}(\psi) \subseteq \mathsf{dom}(\sigma)$, then we may define the relation $\sigma, \dot{\rightsquigarrow}\ \models \psi$:

$$\begin{array}{lllll} \sigma, \dot{\rightsquigarrow} & \models & t \xrightarrow{a} u & \iff & \sigma(t) \stackrel{a}{\rightsquigarrow} \sigma(u) \\ \sigma, \dot{\rightsquigarrow} & \models & t \stackrel{b}{\nrightarrow} & \iff & \nexists q \in \mathsf{Procs}.\, \sigma(t) \stackrel{b}{\rightsquigarrow} q \\ \sigma, \dot{\rightsquigarrow} & \models & t \xrightarrow{\varepsilon} u & \iff & \sigma(t) = \sigma(u) \end{array}$$

We extend $\models$ to sets of formulae:

$$\sigma, \dot{\rightsquigarrow}\ \models H \iff \forall \psi \in H. \big(\sigma, \dot{\rightsquigarrow}\ \models \psi\big).$$

The GSOS theory of [BIM88, Blo89] shows for any GSOS language $\mathcal{L}$, there is a unique appropriate transition relation $\dot{\rightsquigarrow}$; that is, one satisfying the following properties:

1. For any process term $p = f(p_1, \ldots, p_n)$ and action a, if $p \stackrel{a}{\rightsquigarrow} q$, then there exists a rule ρ which enables the transition: that is, there is an instantiation σ on $\mathsf{Vars}(\rho)$ such that:
   (a) $\sigma(x_i) = p_i$ for each i
   (b) $\sigma \models \mathsf{ante}(\rho)$
   (c) $\sigma(\mathsf{cons}(\rho)) = p \xrightarrow{a} q$.
2. For each rule ρ and instantiation σ such that $\sigma, \dot{\rightsquigarrow} \models \mathsf{ante}(\rho)$, then $\sigma, \dot{\rightsquigarrow} \models \mathsf{cons}(\rho)$.

This relation is denoted $\xrightarrow{\cdot}_{\mathcal{L}}$, or simply $\xrightarrow{\cdot}$ when $\mathcal{L}$ is clear from context.

2.3 Ruloid Theorems

We will frequently need to examine the behavior of processes in arbitrary contexts. Our main technical tools for doing so are *Ruloid Theorems* [BIM90, Blo93], which characterize all possible behaviors of all terms in a language, in a form that resembles the original rule format.

Definition 2.2 Ruloids *are like rules, except that the conclusion has the form $t \xrightarrow{a} u$ for some term t, rather than the simpler form $f(\vec{x}) \xrightarrow{a} u$ as required for rules.*

Rules carry *proscriptive* force; that is, they define the behavior of all terms of the language. Ruloids simply carry *descriptive* force; that is, they explain how the rules cause complex terms to compute.

Most well-designed rule formats enjoy a *Ruloid Theorem*. That is, for each term t and $s \in \mathsf{Act}_{\tau}{}^*$, it is possible to calculate a set $\mathcal{R}(t,s)$ of ruloids which precisely capture the circumstances under which t can perform s; and each $\rho \in \mathcal{R}(t,s)$ is still in the same format as the original rules, *mutatis mutandis*. Formally,

Definition 2.3 *A set $\mathcal{R}$ of ruloids is* just right *for a term t and a string s of actions if,*

1. *The conclusion of each $\rho \in \mathcal{R}$ has the form $t \xrightarrow{s} t'$ for some t'.*

2. *$\mathcal{R}$ explains all the transitions possible for instances of t. Formally, let $\sigma : \mathsf{Vars}(t) \to \mathsf{Procs}$, and suppose $\sigma(t) \xrightarrow{s} q$. Then there is some ruloid $\rho \in \mathcal{R}$, and extension $\sigma' \supseteq \sigma$ to $\mathsf{Vars}(\rho)$ such that:*

   (a) $\sigma' \models \mathsf{ante}(\rho)$

   (b) $\sigma'(\mathsf{cons}(\rho)) = \sigma'(t) \xrightarrow{s} q$.

3. *Every ruloid in $\mathcal{R}$ is valid. That is, if $\sigma : \mathsf{Vars}(\rho) \to \mathsf{Procs}$ and $\sigma \models \mathsf{ante}(\rho)$, then $\sigma \models \mathsf{cons}(\rho)$.*

For example, the following set of ruloids for $(x \parallel y) + z$ are just right for that term and the action a:

$$\frac{x \xrightarrow{a} x'}{(x \parallel y) + z \xrightarrow{a} (x' \parallel y)} \qquad \frac{y \xrightarrow{a} y'}{(x \parallel y) + z \xrightarrow{a} (x \parallel y')} \qquad \frac{z \xrightarrow{a} z'}{(x \parallel y) + z \xrightarrow{a} z'}$$

2.4 ε-Presentation

In this study (and many other ones), it is convenient to use rules and ruloids in ε-presentation.

Definition 2.4 *A ruloid ρ in* GSOS(ε) *format is* ε-presented *iff:*

1. *There is at least one antecedent $x \xrightarrow{\zeta} y$ for each source variable x, for some $\zeta \in \mathsf{Act}_{\tau,\varepsilon}$, and*

2. $\mathsf{SourceVars}(\rho) \pitchfork \mathsf{TargetVars}(\rho)$

For example, the first sequencing rule (2) is not ε-presented, as no antecedent mentions x_2. Nor is the rule

$$\frac{x_1 \xrightarrow{a} y_1, \quad x_2 \xrightarrow{\varepsilon} y_2}{x_1 \,;\, x_2 \xrightarrow{a} y_1 \,;\, x_2}$$

Although an antecedent mentions each argument, the source and target variables are not disjoint. However, the equally useful rule

$$\frac{x_1 \xrightarrow{a} y_1, \quad x_2 \xrightarrow{\varepsilon} y_2}{x_1 \,;\, x_2 \xrightarrow{a} y_1 \,;\, y_2}$$

is ε-presented. The precise definition of "equally useful" is *equipotent*:

Definition 2.5 *Two ruloids ρ and ρ' are* equipotent *if, whenever one of them enables a transition $p \xrightarrow{\alpha} q$, then the other does as well.*

Lemma 2.6 *Let $\mathcal{L}$ be any* GSOS(ε) *language. Then there is a* GSOS(ε) *language $\mathcal{L}'$ with the same sets of actions and operations (and hence the same set of terms), and a bijection $\rho \mapsto \rho'$ between their ruloids such that ρ and ρ' are equipotent.*

Proof: Let ρ be a ruloid of $\mathcal{L}$. Let V be the set of variables which either don't appear in the antecedents of ρ, or appear in both source and target. For each $x \in V$, pick a variable x' distinct from $\mathsf{Vars}(\rho)$ and all other x''s. Suppose that $\mathsf{cons}(\rho) = t \xrightarrow{\alpha} u$. Let u' be u with each $x \in V$ replaced by the corresponding x'. Let ρ' be the ruloid:

$$\frac{\mathsf{ante}(\rho) \cup \left\{ x \xrightarrow{\varepsilon} x' | x \in V \right\}}{t \xrightarrow{\alpha} u'}$$

Then clearly ρ and ρ' are equipotent, and ρ' is ε-presented. $\female$

3 Rules for Weak Bisimulation

The simplest kind of operations which respect weak bisimulation are the *patient* ones. Suppose that $p \underline{\Leftrightarrow} p'$, and that $f(p) \xrightarrow{b} g(q)$ via a rule

$$\frac{x \xrightarrow{a} y}{f(x) \xrightarrow{b} g(y)}$$

and a transition $p \xrightarrow{a} q$. All that we know about p' is that $p' \xrightarrow{\tau^*} r_1 \xrightarrow{a} r_2 \xrightarrow{\tau^*} q'$ with $q \underline{\Leftrightarrow} q'$. This suggests that $f(x)$ will have to wait for its argument to perform an a action. So, we need a *patience* rule:

$$\frac{x \xrightarrow{\tau} x'}{f(x) \xrightarrow{\tau} f(x')}$$

which allows its argument to take τ-transitions freely. (It turns out that $g(q)$ does *not* need a patience rule, but this is a theorem.)

Not all operations have, or require, patience rules. The canonical example of an operation that shouldn't have them is prefixing, with rule:

$$\frac{}{ax \xrightarrow{a} x}$$

It would be a mistake to have a patience rule:

$$\textbf{(mistake)} \qquad \frac{x \xrightarrow{\tau} x'}{ax \xrightarrow{\tau} ax'}$$

as (1) this would destroy many essential properties of prefixing, *e.g.*, its ability to guard recursions, and (2) it isn't necessary, as prefixing already does respect weak bisimulation. That is, if $p \underline{\Leftrightarrow} p'$, then ap's only possible move is an a-transition to p; clearly ap' can do an a-transition to p', which is weakly bisimilar to p by hypothesis. More generally, if $f(p)$'s first step of behavior doesn't involve p's behavior, then we don't need patience rules.

So, we only require patience rules for *active arguments* of functions f; that is, those arguments which f allows to run. For example, if we have an operator with the single rule:

$$\frac{x_1 \xrightarrow{a} y_1}{f(x_1, x_2) \xrightarrow{b} g(y_1, x_2)}$$

we would expect to have a patience rule for x_1 but not for x_2.

However, we know from first principles that operations defined by patience rules alone are not as powerful as possible. Bisimulation equivalences, of all sorts, are branching-time equivalences. Thus, it ought to admit *branching tests*: testing for the ability to do both an a and a b simultaneously. That is, it should tolerate copying, in the style of Section 1.2.1. For example, we'd like to have an operation given by a rule of the form:

$$\frac{x \xrightarrow{a} y_1, \quad x \xrightarrow{b} y_2}{k(x) \xrightarrow{c} \mathbf{0}} \tag{4}$$

However, this is somewhat complicated. We can't (with weak bisimulation) expect to detect *simultaneous* transitions. Specifically, $a+b$ is weakly bisimilar to $P_{ab} = \mathsf{rec}\,[x \Leftarrow a + \tau(b + \tau x)]$, which alternately offers a and b:

$$\begin{tikzcd}
 & a+b \arrow[dl, "a"'] \arrow[dr, "b"] & & & P_{ab} \arrow[r, shift left, "\tau"] \arrow[dl, "a"'] & P_{ba} \arrow[l, shift left, "\tau"] \arrow[dr, "b"] & \\
\mathbf{0} & & \mathbf{0} & \Leftrightarrow \quad \mathbf{0} & & & \mathbf{0}
\end{tikzcd} \tag{5}$$

P_{ab} cannot perform an a and a b simultaneously, and thus $k(P_{ab})$ cannot fire by rule (4). Our solution is to require the presence of enough rules and operations so that $k(P_{ab})$ will behave right. Specifically, we insist that there be a two-argument version $k^\star$ of k, with one argument to handle the $\xrightarrow{a}$ test of k and the other for the $\xrightarrow{b}$ test. The operation $k^\star$ has one computational rule, giving $k^\star$ the ability to do (4):

$$\frac{x_1 \xrightarrow{a} y_1, \quad x_2 \xrightarrow{a} y_2}{k^\star(x_1, x_2) \xrightarrow{c} \mathbf{0}} \tag{6}$$

The basic operation k and the derived version $k^\star$ are connected by a pair of rules, allowing k's argument to take a τ move in preparation for doing its two possible behaviors separately. These will be called *bifurcation rules:*

$$\frac{x \xrightarrow{\tau} y}{k(x) \xrightarrow{\tau} k^\star(x, y)} \qquad \frac{x \xrightarrow{\tau} y}{k(x) \xrightarrow{\tau} k^\star(y, x)} \tag{7}$$

Similarly, $k^\star$ has patience rules allowing its arguments to proceed independently:

$$\frac{x_1 \xrightarrow{\tau} y_1}{k^\star(x_1, x_2) \xrightarrow{\tau} k^\star(y_1, x_2)} \qquad \frac{x_2 \xrightarrow{\tau} y_2}{k^\star(x_1, x_2) \xrightarrow{\tau} k^\star(x_1, y_2)} \tag{8}$$

With these rules, we see that $k(P_{ab}) \stackrel{c}{\Longrightarrow} \mathbf{0}$ as desired:

$$k(P_{ab}) \xrightarrow{\tau} k^\star(P_{ab}, P_{ba}) \xrightarrow{c} \mathbf{0}$$

3.1 WB cool Languages

In this section, we work entirely with ε-presented languages. We define a subset of the ε-presented positive GSOS(ε) languages, the *WB cool languages*, in which weak bisimulation is a congruence. For example, CCS without + is a WB cool language. The definition proceeds in two parts. We first give an infinitary version, the *fully WB cool* languages, which places restrictions on all univariate terms. Then we give necessary and sufficient finitary conditions on the rules which guarantee the infinitary version.

Definition 3.1 *The source variable x is* active *in ρ if there is an antecedent $x \xrightarrow{\beta} y$ of ρ (where by our conventions $\beta \in \mathsf{Act}_\tau$ and hence $\beta \neq \varepsilon$). The variable $x \in \mathsf{Vars}(t)$ is active if it is active in any ruloid ρ for t.*

For example, the variable x is active in $k(x)$ and $k^\star(x, x')$, but not in ax. Next, we need to count how many copies of a variable we will need:

Definition 3.2 *Let $x \in \mathsf{Vars}(t)$ for a univariate term t. We define $\mathsf{barb}(t, x)$ to be the maximum number of antecedents $x \xrightarrow{\zeta} y$ in any element of $\rho \in \cup_\beta \mathcal{R}(t, \beta)$.*

For example, $\mathsf{barb}(k(x), x) = 2$, as (4) has two antecedents for x. Recall that we use ε-presented rules and ruloids; among other things, this guarantees that each variable in a rule appears in at least one antecedent. Thus, $\mathsf{barb}(ax, x) = 1$.

Definition 3.3 *A rule ρ with source $f(\vec{x})$ is* straight *if ρ has exactly one antecedent for each argument x_i. Non-straight rules are* branching.

For example, (4) is not straight; (6), (7), and (8) are straight.

Definition 3.4 *A univariate term t is* straight *if $\mathsf{barb}(x, t) = 1$ for each $x \in \mathsf{Vars}(t)$. We define Terms_1^s to be the set of straight univariate terms.*

For example, $k^\star(x_1, x_2)$ is straight, and $k(x)$ is not. Indeed, $k^\star(x_1, x_2)$ is a straight version of $k(x)$, in a sense we will make precise. To describe the correspondence between $k(x)$ and $k^\star(x_1, x_2)$, we several things.

- First, we need a function $(\cdot)^\star$, taking terms to their straightened versions: we will have $(k(x))^\star = k^\star(x_1, x_2)$. Recall that k and $k^\star$ are simply operation symbols; the right hand side of this equation is just a term.
- We need to explain that x_1 and x_2 in $k^\star(x_1, x_2)$ both correspond to x in $k(x)$; we thus have a map $\frac{\mathsf{x}}{\mathsf{x}'}(\cdot) = \frac{\mathsf{x}}{\mathsf{x}'}\Big|_{k(x)}(\cdot) : \{x_1, x_2\} \to \{x\}$, with $\frac{\mathsf{x}}{\mathsf{x}'}(x_1) = \frac{\mathsf{x}}{\mathsf{x}'}(x_2) = x$. The notation $\frac{\mathsf{x}}{\mathsf{x}'}(x')$ is intended to be reminiscent of fractions: we use conventions so that generally $\frac{\mathsf{x}}{\mathsf{x}'}(x') = x$.
- We need *bifurcation* rules, like (7), which allow $k(x)$ to copy its argument and run it along some τ-steps.

- We need a correspondence $\rho \mapsto \rho^\star$ between the rules for $k(x)$ and the rules for $k^\star(x_1, x_2)$. For example, $(4)^\star = (6)$, and the two rules of (7) correspond to the two rules of (8). Note that this is a bijection between the rules for $k(x)$ and those for $k^\star(x_1, x_2)$, and that the rules corresponding to the bifurcating rules are patience rules.

These requirements are formalized in the following definition.

Definition 3.5 *A positive* GSOS(ε) *language* $\mathcal{L}$ *is* fully WB cool *if there is a mapping* $(\cdot)^\star : \mathsf{Terms}_1 \to \mathsf{Terms}_1^s$*, and, for each* $t \in \mathsf{Terms}_1$*, a mapping* $\left.\frac{\mathsf{X}}{\mathsf{X}'}\right|_t(\cdot) : \mathsf{Vars}(t^\star) \to \mathsf{Vars}(t)$*. Let* $\left.\frac{\{\mathsf{X}'\}}{\mathsf{X}}\right|_t(x) = \left\{x' \in \mathsf{Vars}(t^\star) | \frac{\mathsf{X}}{\mathsf{X}'}(x') = x\right\}$*. When* t *is clear from context (which is almost all the time), we write these as simply* $\frac{\mathsf{X}}{\mathsf{X}'}(x')$ *and* $\frac{\{\mathsf{X}'\}}{\mathsf{X}}(x)$*. Note that* $\frac{\mathsf{X}}{\mathsf{X}'}(\cdot)$ *is a substitution, and we may apply it to terms and the like.*

1. *If* t *is straight, then* $\frac{\mathsf{X}}{\mathsf{X}'}(t^\star) = t$.
2. *If* x *is active in* t*, then for all* $x' \in \frac{\{\mathsf{X}'\}}{\mathsf{X}}(x)$*, there is a ruloid* $\mathsf{bi}^t_{x'} \in \mathcal{R}(t, \tau)$*:*

$$\frac{\left\{x \xrightarrow{\tau} v_{x'}\right\} \cup \left\{x_1 \xrightarrow{\varepsilon} v_{x_1'} | \left(\frac{\mathsf{X}}{\mathsf{X}'}(x_1') = x_1\right) \wedge \langle x, x'\rangle \neq \langle x_1, x_1'\rangle\right\}}{t \xrightarrow{\tau} \sigma(t^\star)} \tag{9}$$

   where $v_{x'}$*'s are distinct fresh variables, and* $\sigma(x') = v_{x'}$ *for each* x'*.* $\mathsf{bi}^t_{x'}$ *is the* bifurcation *ruloid of* t *for* x'*; if* t *is straight, it is also called the* patience *ruloid for* x'.[2]
3. *The only ruloids with* τ*'s in the antecedent are the bifurcation ruloids.*
4. *For all* t *and* α*, there is a bijection* $(\cdot)^\star : \mathcal{R}(t, \alpha) \to \mathcal{R}(t^\star, \alpha)$ *such that:*

$$\begin{aligned} \mathsf{ante}(\rho) &= \tfrac{\mathsf{X}}{\mathsf{X}'}(\mathsf{ante}(\rho^\star)) && (10)\\ \mathsf{target}(\rho) &= \mathsf{target}(\rho^\star) && (11) \end{aligned}$$

Note that Definition 3.5 part 4 implies that the variables appearing in ρ and $\rho^\star$ are closely related. For any ruloid ρ, we have

$$\rho = \frac{\left\{x \xrightarrow{\beta} y\right\}}{t \xrightarrow{\alpha} u} \qquad \rho^\star = \frac{\left\{x' \xrightarrow{\beta} y\right\}}{t^\star \xrightarrow{\alpha} u}.$$

Note that $\mathsf{TargetVars}(\rho) = \mathsf{TargetVars}(\rho^\star)$. As $\rho^\star$ is straight, for each $x' \in \mathsf{SourceVars}(\rho^\star)$ there is exactly one antecedent $x' \xrightarrow{\zeta} y$ of $\rho^\star$. There is thus a bijection $\frac{\mathsf{X}'}{\mathsf{Y}}(\cdot)$ from $\mathsf{TargetVars}(\rho^\star)$ to $\mathsf{SourceVars}(\rho^\star)$, where $\frac{\mathsf{X}'}{\mathsf{Y}}(y) = x'$ iff $x' \xrightarrow{\zeta} y \in \mathsf{ante}(\rho)$.[3]

[2] Recall that the source and target variables of a GSOS(ε) ruloid must be disjoint. Thus, we must introduce new variables $v_{x'}$, to avoid possible name clashes.

[3] Note that ρ is an implicit parameter of these functions.

Similarly, for each y, there is a unique $x \in \mathsf{SourceVars}(\rho)$ with an antecedent $x \xrightarrow{\varsigma} y$; let $\frac{\mathsf{x}}{\mathsf{y}}(\cdot) : \mathsf{TargetVars}(\rho) \to \mathsf{SourceVars}(\rho)$ be the function associating x's with y's. It is not in general a bijection (though it is by definition a bijection for straight ruloids); let $\frac{\{\mathsf{y}\}}{\mathsf{x}}(\cdot) : \mathsf{SourceVars}(\rho) \to \wp(\mathsf{TargetVars}(\rho))$ be given by $\frac{\{\mathsf{y}\}}{\mathsf{x}}(x) = \left\{y \middle| \frac{\mathsf{x}}{\mathsf{y}}(y) = x\right\}$.

Note also that if ρ is a bifurcation ruloid, then so is $\rho^\star$ and vice versa.

3.2 Weak Bisimulation is a Congruence for fully WB cool Languages

In this section, we show that weak bisimulation is a congruence for fully WB cool languages. In Section 3.2.1 we develop a more local equivalent definition of weak bisimulation; which we use in Section 3.2.2 to prove the main theorem.

3.2.1 Local Weak Bisimulation Relations

It is convenient to use an alternate version of weak bisimulation, with a much more local character.

Definition 3.6 *$p \stackrel{a}{\multimap} q$ iff $a = \tau$ and $p \xrightarrow{\tau^*} q$, or $a \neq \tau$ and $p \xrightarrow{\tau^* a} q$. A relation $\smile$ is a* local weak bisimulation relation *if, whenever $p \smile p'$ and $p \xrightarrow{\alpha} q$ for some α, then $p' \stackrel{\alpha}{\multimap} q'$ and $q \smile q'$; and vice versa.*

Lemma 3.7 *Every local weak bisimulation relation $\smile$ is a weak bisimulation relation.*

Proof: Suppose that $p \smile p'$ for a local weak bisimulation relation $\smile$, and that $p \stackrel{\alpha}{\Longrightarrow} q$. There are two cases: $\alpha \neq \tau$ and $\alpha = \tau$. We only present the $\alpha \neq \tau$ case. In this case, we have $p \xrightarrow{\tau^m \alpha \tau^n} q$ for some m, n. We may then fill in the following diagram from the left, using the definition of local weak bisimulation relation repeatedly:

$$\begin{array}{ccccccccccccccc}
p & = & p_0 & \xrightarrow{\tau} & p_1 & \xrightarrow{\tau} \cdots \xrightarrow{\tau} & p_m & \xrightarrow{\alpha} & q_1 & \xrightarrow{\tau} \cdots \xrightarrow{\tau} & q_n & = & q \\
 & & \vdots\smile & & \vdots\smile & & \vdots\smile & & \vdots\smile & & \vdots\smile & & \\
p' & = & p'_0 & \stackrel{\tau}{\multimap} & p'_1 & \stackrel{\tau}{\multimap} \cdots \stackrel{\tau}{\multimap} & p'_m & \stackrel{\alpha}{\multimap} & q'_1 & \stackrel{\tau}{\multimap} \cdots \stackrel{\tau}{\multimap} & q'_n & = & q'
\end{array}$$

It is straightforward from this that $p' \stackrel{\alpha}{\Longrightarrow} q'$ and $q \smile q'$ as desired. ♤

3.2.2 Weak bisimulation is a congruence

We use standard bisimulation methodology to show that $p \underline{\leftrightarrow} p'$ implies $f(p) \underline{\leftrightarrow} f(p')$. We construct a relation $\smile$ which includes $\underline{\leftrightarrow}$ and is closed under application of operations. The diacritical conventions used in this proof are summarized in Figure 4.

Definition 3.8 *We define $p \smile p^\bullet$ to hold if:*

$t^\star, \rho^\star$	Straight versions of t and ρ
x	Source variable of ρ
x'	Source variable of $\rho^\star$
y	Target variables of ρ and $\rho^\star$
$p^\bullet, \sigma^\bullet$	Things on the right-hand side of $\smile$.
σ_1	Components after one τ-step of computation.
$\acute{\sigma}$	Components after all their τ-steps.
$\hat{\sigma}$	Components after one α of computation.

Figure 4: Diacritical Conventions

1. $p \underline{\Leftrightarrow} p^\bullet$, or

2. There is a univariate term t, and substitutions σ and $\sigma^\bullet$ on $\mathsf{Vars}(t)$, such that for all $x \in \mathsf{Vars}(t)$, $\sigma(x) \smile \sigma^\bullet(x)$ and $p = \sigma(t)$ and $p^\bullet = \sigma^\bullet(t)$.

3. There is a univariate term t, and substitutions σ on $\mathsf{Vars}(t)$ and $\sigma^\bullet$ on $\mathsf{Vars}(t^\star)$ such that $p = \sigma(t)$ and $p^\bullet = \sigma^\bullet(t^\star)$, and $\forall x' \in \mathsf{Vars}(t^\star).\sigma(\frac{\mathsf{X}}{\mathsf{X}'}(x')) \smile \sigma^\bullet(x')$.

4. $p^\bullet \smile p$ by 3.

Note that $\smile$ is symmetric.

Theorem 3.9 *If $\mathcal{L}$ is a fully WB cool language, then weak bisimulation is a congruence for $\mathcal{L}$.*

Proof: By Lemma 3.10, $\smile$ is a weak bisimulation relation. Suppose that $p_i \underline{\Leftrightarrow} p_i'$ for $i = 1, 2, \ldots$. It suffices to show that $f(\vec{p}) \underline{\Leftrightarrow} f(\vec{p'})$ for all operation symbols f of $\mathcal{L}$. Let $t = f(\vec{x})$, and define $\sigma(x_i) = p_i$ and $\sigma^\bullet(x_i) = p_i^\bullet$; then we have $f(\vec{p}) = \sigma(t) \smile \sigma^\bullet(t) = f(\vec{p'})$ by clause 2. Hence $f(\vec{p}) \underline{\Leftrightarrow} f(\vec{p'})$ as desired.

Lemma 3.10 $\smile$ *is a weak bisimulation relation.*

Proof: It suffices to show that, if $p \smile p^\bullet$ and $p \xrightarrow{\alpha} q$, then there is some $q^\bullet$ such that $p^\bullet \overset{\alpha}{\longrightarrow\!\!\circ}\, q^\bullet$ and $q \smile q^\bullet$. There are four cases, depending on the reason that $p \smile p^\bullet$.

1: This case is trivial.

2: In this case, $p = \sigma(t)$ and $p^\bullet = \sigma^\bullet(t)$, where $\forall x \in \mathsf{Vars}(t).\sigma(x) \smile \sigma^\bullet(x)$. Let ρ be the ruloid enabling $p \xrightarrow{\alpha} q$, and $u = \mathbf{target}(\rho)$. There is an instantiation $\hat{\sigma}$ of $\mathsf{Vars}(t)$, such that $q = \hat{\sigma}(u)$ and for each antecedent $x \xrightarrow{\zeta} y$ of ρ, we have

$$\sigma(x) \xrightarrow{\zeta} \hat{\sigma}(y) \tag{12}$$

By (12) and the induction hypothesis, for each $x \xrightarrow{\zeta} y$, there are terms $\acute{\sigma}^\bullet(x')$ and $\hat{\sigma}^\bullet(y)$ (where $x' = \frac{\mathsf{x}'}{\mathsf{y}}(y)$) such that,

$$\begin{array}{lllllll} \zeta = \varepsilon \;\Rightarrow & \sigma^\bullet(x) & = & \acute{\sigma}^\bullet(x') & = & \hat{\sigma}^\bullet(y) & \smile \;\hat{\sigma}(y) \\ \zeta = \tau \;\Rightarrow & \sigma^\bullet(x) & \xrightarrow{\tau^*} & \acute{\sigma}^\bullet(x') & = & \hat{\sigma}^\bullet(y) & \smile \;\hat{\sigma}(y) \\ \text{Else} & \sigma^\bullet(x) & \xrightarrow{\tau^*} & \acute{\sigma}^\bullet(x') & \xrightarrow{\zeta} & \hat{\sigma}^\bullet(y) & \smile \;\hat{\sigma}(y) \end{array} \tag{13}$$

We have two cases, depending on whether ρ was an action rule or a bifurcation rule.

2.action: In this case, no ζ is τ. We have two subcases, depending on whether or not any $\sigma^\bullet(x)$ takes a τ-step in (13)

2.action.no: In this case, we have $\sigma^\bullet(x) \xrightarrow{\zeta} \hat{\sigma}^\bullet(y)$ for each antecedent $x \xrightarrow{\zeta} y$ of ρ. Hence, ρ applies to $p^\bullet = \sigma^\bullet(t)$:

$$p^\bullet \xrightarrow{\alpha} \hat{\sigma}^\bullet(u) = q^\bullet$$

and $q = \hat{\sigma}(u) \smile \hat{\sigma}^\bullet(u) = q^\bullet$ as desired.

2.action.yes: In this case, there is an antecedent $x_0 \xrightarrow{\zeta_0} y_0$ in $\mathsf{ante}\ (\rho)$ such that $\sigma^\bullet(x_0) \xrightarrow{\tau^+} \acute{\sigma}^\bullet(x_0')$. Let $x_0' = \frac{\mathsf{x}'}{\mathsf{y}}(y_0)$. In this case, there is a bifurcation ruloid (except for variable names) $\mathsf{bi}^t_{x_0'}$, which is

$$\frac{x_0 \xrightarrow{\tau} v_{x_0'}, \left\{ x \xrightarrow{\varepsilon} x' | \frac{\mathsf{x}}{\mathsf{x}'}(x') = x \wedge (x \neq x_0 \vee x' \neq x_0') \right\}}{t \xrightarrow{\tau} t^\star[x_0 := v_{x_0'}]} \tag{14}$$

We know that

$$\sigma^\bullet(x_0) \xrightarrow{\tau} r^\bullet \xrightarrow{\tau^*} \acute{\sigma}^\bullet(x_0') \xrightarrow{\zeta_0} \hat{\sigma}^\bullet(y_0)$$

Let $\sigma_1^\bullet$ be the substitution on $\mathsf{Vars}(t^\star)$ given by:

$$\sigma_1^\bullet(x') = \begin{cases} r^\bullet & x' = x_0' \\ \sigma^\bullet(\frac{\mathsf{x}}{\mathsf{x}'}(x')) & \text{otherwise} \end{cases}$$

By the bifurcation rule (14)

$$p^\bullet \xrightarrow{\tau} \sigma_1^\bullet(t^\star) = p_1^\bullet$$

Now, we will run $p_1^\bullet$ with patience ruloids until it is ready to execute $\rho^\star$ the right way. For each $x' \xrightarrow{\beta} y$ in $\mathsf{ante}\ (\rho^\star)$, we have $\sigma_1^\bullet(x') \xrightarrow{\tau^*} \acute{\sigma}^\bullet(x')$ and for all $x' \xrightarrow{\varepsilon} y$, we have $\acute{\sigma}^\bullet(x') = \sigma_1^\bullet(x') = \sigma^\bullet\left(\frac{\mathsf{x}}{\mathsf{x}'}(x')\right)$. Hence, we have transitions:

$$p_1^\bullet \xrightarrow{\tau^*} \acute{\sigma}^\bullet(t^\star) = p_2^\bullet$$

Now, for each antecedent $x' \xrightarrow{\beta} y$ of $\rho^\star$, we have $\acute{\sigma}^\bullet(x') \xrightarrow{\beta} \hat{\sigma}^\bullet(y)$, and for each antecedent $x' \xrightarrow{\varepsilon} y$, we have $\acute{\sigma}^\bullet(x') = \hat{\sigma}^\bullet(y)$. Hence $\rho^\star$ applies to $p_2^\bullet$:

$$p_2^\bullet \xrightarrow{\alpha} \hat{\sigma}^\bullet(u) = q^\bullet.$$

$q \smile q^\bullet$ by part 3 of the definition.

2.bifurcation: In this case, $\rho = \mathsf{bi}^t_{x'_0}$ for some x'_0. Let $x_0 = \frac{\mathsf{x}}{\mathsf{x'}}(x'_0)$. Let $\hat{\sigma}(x'_0)$ be the τ-child of $\sigma(x_0)$ that caused the transition, and $\hat{\sigma}(x') = \sigma\left(\frac{\mathsf{x}}{\mathsf{y}}(x')\right)$ for other x''s. Then $q = \hat{\sigma}(t^\star)$. By the induction hypothesis, there is a process $\hat{\sigma}^\bullet(x'_0) \smile \hat{\sigma}(x'_0)$ with $\sigma^\bullet(x) \xrightarrow{\tau^n} \hat{\sigma}^\bullet(x'_0)$. For each $x' \neq x'_0$, let $\hat{\sigma}^\bullet(x') = \sigma^\bullet\left(\frac{\mathsf{x}}{\mathsf{y}}(x')\right)$ There are two cases: $n = 0$ and $n > 0$.

2.bifurcation.($n = 0$)**:** We have $\sigma^\bullet(x_0) \smile \hat{\sigma}(x'_0)$. Furthermore, for each $x' \neq x'_0$, $\hat{\sigma}^\bullet(x') = \sigma^\bullet\left(\frac{\mathsf{x}}{\mathsf{x'}}(x')\right) \smile \hat{\sigma}(x')$. Hence

$$q^\bullet = p^\bullet = \sigma^\bullet(t) \smile \hat{\sigma}(t^\star) = q$$

as desired.

2.bifurcation.($n > 0$)**:** This is basically the same as case **2.action.yes**.

3: In this case, we have $p = \sigma(t)$, $p' = \sigma^\bullet(t^\star)$, and for all $x' \in \mathsf{Vars}(t^\star)$ we have $\sigma\left(\frac{\mathsf{x}}{\mathsf{x'}}(x')\right) \smile \sigma^\bullet(x')$. Let ρ be the ruloid and $\hat{\sigma}$ the substitution for target variables enabling the transition $p \xrightarrow{\alpha} q$. That is, for all antecedents $x \xrightarrow{\zeta} y$ of ρ, we have $\sigma(x) \xrightarrow{\zeta} \hat{\sigma}(y)$; and also $p = \sigma(t)$ and $q = \hat{\sigma}(u)$ where $u = \mathsf{target}(\rho)$.

By the induction hypothesis, we have for each antecedent $x' \xrightarrow{\zeta} y$ of $\rho^\star$, a process $\hat{\sigma}^\bullet(y)$ such that

$$\begin{array}{lllllllll} \zeta = \varepsilon & \Rightarrow & \sigma^\bullet(x) & = & \acute{\sigma}^\bullet(x') & = & \hat{\sigma}^\bullet(y) & \smile & \hat{\sigma}(y) \\ \zeta = \tau & \Rightarrow & \sigma^\bullet(x) & \xrightarrow{\tau^*} & \acute{\sigma}^\bullet(x') & = & \hat{\sigma}^\bullet(y) & \smile & \hat{\sigma}(y) \\ \text{Else} & & \sigma^\bullet(x) & \xrightarrow{\tau^*} & \acute{\sigma}^\bullet(x') & \xrightarrow{\zeta} & \hat{\sigma}^\bullet(y) & \smile & \hat{\sigma}(y) \end{array} \tag{15}$$

3.action: Unlike most other cases, we don't need to apply a bifurcation ruloid to $p^\bullet$; it's already bifurcated. By repeated uses of patience ruloids for $\rho^\star$, we have

$$p^\bullet = \sigma^\bullet(t^\star) \xrightarrow{\tau^*} \acute{\sigma}^\bullet(t^\star) = p^\bullet_2.$$

And by $\rho^\star$, we have

$$p^\bullet_2 \xrightarrow{\alpha} \hat{\sigma}^\bullet(u) = q^\bullet$$

Now, we have $q = \hat{\sigma}(u) \smile \hat{\sigma}^\bullet(u) = q^\bullet$ as desired.

3.bifurcation: Otherwise, ρ is a bifurcation rule, $\rho = \mathsf{bi}^t_{x'_0}$. Let $x_0 = \frac{\mathsf{x}}{\mathsf{x'}}(x'_0)$. Then $q = \hat{\sigma}(t^\star)$, where $\sigma(x) \xrightarrow{\tau} \hat{\sigma}(x'_0)$ is the instantiation of the (unique) non-ε antecedent of ρ, and $\hat{\sigma}(x') = \sigma\left(\frac{\mathsf{x}}{\mathsf{x'}}(x')\right)$ for $x' \neq x'_0$.

By induction, we have $\sigma^\bullet(x'_0) \xrightarrow{\tau^*} \hat{\sigma}^\bullet(x'_0) \smile \hat{\sigma}(x'_0)$. Let $\hat{\sigma}^\bullet(x') = \sigma^\bullet(x')$ for $x' \neq x'_0$.

By repeated use of the x'_0 patience rule for $t^\star$, we have

$$p^\bullet = \sigma^\bullet(t^\star) \xrightarrow{\tau^*} \hat{\sigma}^\bullet(t^\star)$$

and $q = \hat{\sigma}(t^\star) \smile \hat{\sigma}^\bullet(t^\star) = q^\bullet$ as desired.

4: In this case, we have $p = \sigma(t^\star)$, $p^\bullet = \sigma^\bullet(t)$, and $\forall x' \in \mathsf{Vars}(t^\star).\sigma(x') \smile \sigma^\bullet\left(\frac{\mathsf{x}}{\mathsf{x}'}(x')\right)$. Let $\rho^\star$ enable the transition $p \xrightarrow{\alpha} q$. As usual we have

$$\begin{array}{llccccccc}
\zeta = \varepsilon & \Rightarrow & \sigma^\bullet(x) & = & \acute{\sigma}^\bullet(x') & = & \hat{\sigma}^\bullet(y) & \smile & \hat{\sigma}(y) \\
\zeta = \tau & \Rightarrow & \sigma^\bullet(x) & \xrightarrow{\tau^*} & \acute{\sigma}^\bullet(x') & = & \hat{\sigma}^\bullet(y) & \smile & \hat{\sigma}(y) \\
\text{Else} & & \sigma^\bullet(x) & \xrightarrow{\tau^*} & \acute{\sigma}^\bullet(x') & \xrightarrow{\zeta} & \hat{\sigma}^\bullet(y) & \smile & \hat{\sigma}(y)
\end{array} \tag{16}$$

There are two cases, depending on whether ρ is an action or bifurcation rule.

4.action: We have two subsubcases, depending on whether there are any τ-moves or not:

4.action.no: In this case, for all $x \xrightarrow{\zeta} y \in \mathsf{ante}(\rho)$, we have $\sigma^\bullet(x) \xrightarrow{\zeta} \hat{\sigma}^\bullet(y)$. So, ρ applies to $p^\bullet = \sigma^\bullet(t)$, giving us the transition

$$p^\bullet \xrightarrow{\alpha} \hat{\sigma}^\bullet(u) = q^\bullet \smile q = \hat{\sigma}(u)$$

as desired.

4.action.yes: Let y_0 be a y such that $\sigma^\bullet(x) \xrightarrow{\tau^+} \hat{\sigma}^\bullet(x')$, where $x'_0 = \frac{\mathsf{x}'}{\mathsf{y}}(y_0)$, and $x_0 = \frac{\mathsf{x}}{\mathsf{y}}(y_0)$. Clearly, x_0 is active in ρ, and hence there is a bifurcation ruloid $\mathsf{bi}^t_{x'_0}$. We have

$$\sigma^\bullet(x_0) \xrightarrow{\tau} \sigma^\bullet_1(x'_0) \xrightarrow{\tau^*} \acute{\sigma}^\bullet(x'_0) \xrightarrow{\beta_0} \hat{\sigma}^\bullet(y_0)$$

Let $\sigma^\bullet_1(x'_1) = \sigma^\bullet(\frac{\mathsf{x}}{\mathsf{x}'}(x'_1))$ for all $x'_1 \neq x'_0$. The bifurcation ruloid $\mathsf{bi}^t_{x'_0}$ enables the transition

$$p^\bullet \xrightarrow{\tau} \sigma^\bullet_1(t^\star) = p^\bullet_1.$$

We have, for all active x', $\sigma^\bullet_1(x') \xrightarrow{\tau^*} \acute{\sigma}^\bullet(x')$; so repeated uses of the patience rules for $t^\star$ gives us

$$p^\bullet_1 \xrightarrow{\tau^*} \acute{\sigma}^\bullet(t^\star) = p^\bullet_2$$

Now, $\rho^\star$ applies to $p^\bullet_2$, giving

$$p^\bullet_2 \xrightarrow{\alpha} \hat{\sigma}^\bullet(u) = q^\bullet \smile q = \hat{\sigma}(u)$$

as desired.

4.bifurcation: In this case, ρ is a bifurcation ruloid $\mathsf{bi}^t_{x'_0}$. Let $\rho^\star$ be the bifurcation ruloid $\mathsf{bi}^{t\star}_{x'_0}$. We have $\sigma(x'_0) \xrightarrow{\tau} \hat{\sigma}(x'_0)$, and let $\hat{\sigma}(x') = \sigma(x')$ for all $x' \neq x'_0$. Let $x_0 = \frac{\mathsf{x}}{\mathsf{x}'}(x'_0)$. By the induction hypothesis, we have $\sigma^\bullet(x_0) \xrightarrow{\tau^n} \hat{\sigma}^\bullet(x'_0) \smile \hat{\sigma}(x'_0)$. Let $\hat{\sigma}^\bullet(x') = \sigma^\bullet\left(\frac{\mathsf{x}}{\mathsf{x}'}(x')\right)$ for all $x' \neq x'_0$. We have two subsubcases, depending on n:

4.bifurcation.($n = 0$): In this case, we have $\sigma^\bullet(x_0) = \hat{\sigma}^\bullet(x'_0) \smile \hat{\sigma}(x'_0)$. For all $x' \neq x'_0$, we have $\sigma^\bullet\left(\frac{\mathsf{x}}{\mathsf{x}'}(x')\right) \smile \sigma(x') = \hat{\sigma}(x')$. Hence,

$$q^\bullet = \sigma^\bullet(t) \smile \hat{\sigma}(t^\star) = q$$

as desired.

4.bifurcation.$(n \neq 0)$: In this case, we have

$$\sigma^\bullet(x_0) \xrightarrow{\tau} \sigma_1^\bullet(x_0') \xrightarrow{\tau^{n-1}} \hat{\sigma}^\bullet(x_0') \smile \hat{\sigma}(x_0').$$

The bifurcation ruloid $\mathsf{bi}^t_{x_0'}$ applies to $p^\bullet$:

$$p^\bullet \xrightarrow{\tau} \sigma_1^\bullet(t^\star)$$

where $\sigma_1^\bullet(x') = \sigma^\bullet\left(\frac{\mathsf{x}}{\mathsf{x}'}(x')\right)$ for all $x' \neq x_0'$. We now use $n-1$ patience ruloids on $\sigma_1^\bullet(t^\star)$:

$$\sigma_1^\bullet(t^\star) \xrightarrow{\tau^{n-1}} \hat{\sigma}^\bullet(t^\star) = q^\bullet$$

Then $q = \hat{\sigma}(t^\star) \smile \hat{\sigma}^\bullet(t^\star) = q^\bullet$ as desired.

♀

3.3 Finitary Characterization: WB cool Languages

As we have seen, fully WB cool languages respect weak bisimulation. Unfortunately, the definition of fully WB cool is infinitary. In this section, we present finitary, decidable necessary and sufficient conditions.

Definition 3.11 *A language is WB cool if the terms $f(\vec{x})$ where f is a function symbol and $\vec{x}$ are distinct variables satisfy the fully WB cool condition.*

The proof that WB cool languages are fully WB cool proceeds by constructing the ruloid sets $\mathcal{R}(t,\alpha)$ for each t and α, and then checking that they satisfy Definition 3.5. The ruloid theorem will be presented as what amounts to a functional program, in which we construct the set of ruloids for each term by induction.

Theorem 3.12 *Let $\mathcal{L}$ be a WB cool language. Then $\mathcal{L}$ is fully WB cool.*

Proof: We first construct sets $\mathcal{R}(t,\alpha)$ for all terms t and actions $\alpha \in \mathsf{Act}_{\tau,\varepsilon}$, and then show that are fully WB cool. The constructions for ε are easy:

$$\mathcal{R}(t,\varepsilon) = \left\{ \frac{\left\{x \xrightarrow{\varepsilon} v_x | x \in \mathsf{Vars}(t)\right\}}{t \xrightarrow{\varepsilon} \sigma(t)} \right\}$$

where $\sigma(x) = v_x$. The constructions for variables are equally easy:

$$\mathcal{R}(x,\alpha) = \left\{ \frac{x \xrightarrow{\alpha} y}{x \xrightarrow{\alpha} y} \right\}.$$

Suppose, then, that $t = f(\vec{t})$ is a univariate term. Consider a rule ρ for f:

$$\rho = \frac{x_i \xrightarrow{\zeta_{ij}} y_{ij}}{f(\vec{x}) \xrightarrow{b} u}$$

Choose ruloids $\rho_{ij} \in \mathcal{R}(t_i, \zeta_{ij})$. These ruloids give conditions under which $t_i \xrightarrow{\zeta_{ij}}$. They have the form:

$$\rho_{ij} = \frac{\left\{z \xrightarrow{\zeta} w | z \xrightarrow{\zeta} w \in \mathsf{ante}(\rho_{ij})\right\}}{t_i \xrightarrow{\zeta_{ij}} u_{ij}}$$

Rename the target variables as necessary in some fixed way so that $\mathsf{Vars}(u_{ij})$ are all disjoint. Let the ruloid $\bar{\rho}$ be:

$$\bar{\rho} = \frac{\bigcup_{ij} \mathsf{ante}(\rho_{ij})}{t \xrightarrow{c} u[y_{ij} := u_{ij}]}$$

Note that by hypothesis the ruloids for $(f(\vec{x}))^\star$ are in the right correspondence with those for $f(\vec{x})$. That is, corresponding to ρ there is a ruloid:

$$\rho^\star = \frac{\left\{x'_{ij} \xrightarrow{\zeta_{ij}} y_{ij} | \frac{\mathsf{X}}{\mathsf{X}'}\left(x'_{ij}\right) = x_i, \quad x_i \xrightarrow{\zeta_{ij}} y_{ij} \in \mathsf{ante}(\rho)\right\}}{(f(\vec{x}))^\star \xrightarrow{c} u}$$

and by induction to each ρ_{ij} there is a valid ruloid $\rho^\star_{ij}$:

$$\rho^\star_{ij} = \frac{\left\{z' \xrightarrow{\zeta} w | \exists z \xrightarrow{\zeta} w \in \mathsf{ante}(\rho_{ij}). \frac{\mathsf{X}}{\mathsf{X}'}(z') = z\right\}}{t^\star_i \xrightarrow{\zeta_{ij}} u_{ij}}$$

Thus, there is a valid ruloid

$$\bar{\rho}^\star = \frac{\bigcup_{ij} \mathsf{ante}(\rho_{ij})}{t^\star \xrightarrow{b} u[y_{ij} := u_{ij}]}$$

where

$$t^\star = (f(\vec{x}))^\star[x'_i := t^\star_i]$$

Furthermore, we define $\left.\frac{\mathsf{X}}{\mathsf{X}'}\right|_t (z)$ for t to be the union of the $\left.\frac{\mathsf{X}}{\mathsf{X}'}\right|_{t_i} (\cdot)$ functions for the t_i. That is, if $z \in \mathsf{Vars}(t)$, then $z \in \mathsf{Vars}(t_i)$ for precisely one i. Let $\left.\frac{\mathsf{X}}{\mathsf{X}'}\right|_t (z)$ be $\left.\frac{\mathsf{X}}{\mathsf{X}'}\right|_{t_i} (z)$. We now check the parts of Definition 3.5.

Part 1: If t is straight, then each t_i is also straight. Hence $\frac{\mathsf{X}}{\mathsf{X}'}(t^\star_i) = t_i$, and furthermore $f(\vec{x})$ is straight; hence $(f(\vec{x}))^\star = f(\vec{x})$, and so $\frac{\mathsf{X}}{\mathsf{X}'}(t^\star) = t$.

Part 2: Suppose that z is active in t, and $z' \in \frac{\{\mathsf{X}'\}}{\mathsf{X}}(z)$. We know that $z \in \mathsf{Vars}(t_i)$ for some i, and x_i is active in $f(\vec{x})$.[4] Bifurcation ruloids for t may be built from the bifurcation ruloids for f and t_i.

[4]Note that the ruloid for an ε transition has only ε's as antecedents.

Part 3: Suppose that $\bar{\rho}$ has a τ in an antecedent $z \xrightarrow{\tau} w$. Then by construction, some ρ_{ij} also has $z \xrightarrow{\tau} w$ as an antecedent. Hence ρ_{ij} is a bifurcation ruloid for t_i. As it proves a conclusion of the form $t_i \xrightarrow{\tau} u_{ij}$, the rule ρ must have an antecedent $x_i \xrightarrow{\tau} y_{ij}$; hence ρ is a bifurcation rule as well. By construction, $\bar{\rho}$ is also a bifurcation ruloid.

Part 4: Straightforward from the definitions.

♀

4 Rooted Weak Bisimulation

Weak bisimulation is not a congruence for most process algebras. Indeed, such basic operations as nondeterministic choice and selective communication are not WB cool and do not respect weak bisimulation. For example, the CCS rules for $+$ are

$$\frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'}$$

for all a. In particular, there are no patience rules. We have $a \underline{\Leftrightarrow} \tau a$, but $a + b \not\underline{\Leftrightarrow} \tau a + b$. For this reason, some researchers [Hen88] introduce patent versions of $+$, with the above rules for all $a \neq \tau$, and two patience rules:

$$\frac{x \xrightarrow{\tau} x'}{x +_H y \xrightarrow{\tau} x' +_H y} \qquad \frac{y \xrightarrow{\tau} y'}{x +_H y \xrightarrow{\tau} y' +_H y}$$

With this notion of choice, one may build a WB cool version of CCS, for which weak bisimulation will be a congruence.

The more standard solution is to use a slightly different notion of equivalence: *rooted weak bisimulation*, called *rooted* τ *bisimulation* in [BW90]; Milner originally called it *observational congruence* [Mil80]. The definition is given in Definition 1.4.

The characterization of RWB cool languages is rather complicated. Rather than do it immediately, we start with a related, slightly stronger notion, *strongly rooted weak bisimulation*, which is quite reminiscent of rooted *branching* bisimulation. SRWB cool languages are much like RWB cool and RBB cool languages, but the definitions and proofs are much simpler. We deal with rooted weak bisimulation proper in the next subsection.

4.1 Strongly Rooted Weak Bisimulation

For expository convenience, we introduce a slightly strengthened version of rooted weak bisimulation. The rule format for rooted weak bisimulation is based on the ideas used in much simpler form here.

Strongly rooted weak bisimulation is a congruence for a wider class of languages than ordinary rooted weak bisimulation. It probably roughly as useful for actual verification as the ordinary relation. The canonical difference is, $a + \tau a \underline{\Leftrightarrow}_r \tau a$, but $a + \tau a \not\underline{\Leftrightarrow}_{sr} \tau a$. The theory of strongly rooted weak bisimulation remains to be developed. Perhaps it will turn out to be good for something.

Definition 4.1 *Processes p and p' are* strongly rooted weakly bisimilar, *$p \mathrel{\underline{\leftrightarrow}_{sr}} q$, iff whenever $p \xrightarrow{\beta} q$ for some $\beta \in \mathsf{Act}_\tau$, then $p' \xrightarrow{\beta} q' \mathrel{\underline{\leftrightarrow}} q$, and vice versa.*

This is a direct definition of $\underline{\leftrightarrow}_{sr}$ in terms of $\underline{\leftrightarrow}$.

We introduce a rule format which guarantees that all operations respect strongly rooted weak bisimulation. Note that $\underline{\leftrightarrow}_{sr}$ is an extremely powerful relation on its first move; it has the full power of *strong* bisimulation on that move. It is not surprising that it respects a rule format which has the full power of GSOS rules available on its first move. After the first move, $\underline{\leftrightarrow}_{sr}$ is simply $\underline{\leftrightarrow}$; so we use the WB cool rule format.

A SRWB cool language, like CCS, then, has two kinds of operations. First, it includes a WB cool sublanguage of *tame* operations — for CCS, all the operations except $+$. Second, it includes some *wild* operations defined by arbitrary GSOS rules — like $+$ — which are required to yield terms in the WB cool sublanguage after the first step of computation. Formally:

Definition 4.2 *A (not necessarily positive)* GSOS(ε) *language $\mathcal{L}$ is SRWB cool iff the function symbols can be partitioned into two sets, of* wild *and* tame *operations, such that*

1. *The targets of all rules in $\mathcal{L}$ are univariate terms mentioning only tame operations.*

2. *The sublanguage of $\mathcal{L}$ given by just the tame operations and all their rules is WB cool.*

Theorem 4.3 *Let $\mathcal{L}$ be a SRWB cool language. Then $\underline{\leftrightarrow}_{sr}$ is a congruence for $\mathcal{L}$.*

Proof: Clearly, if $\sigma(x) \mathrel{\underline{\leftrightarrow}} \sigma'(x)$ for all $x \in \mathsf{dom}(\sigma)$, and t is a univariate term mentioning only tame operations, then $\sigma(t) \mathrel{\underline{\leftrightarrow}} \sigma'(t)$.

Now, suppose that $p_i \mathrel{\underline{\leftrightarrow}_{sr}} p'_i$ for each i. It suffices to show that $f(\vec{p}) \mathrel{\underline{\leftrightarrow}_{sr}} f(\vec{p}')$. Suppose that some rule ρ enables a transition $f(\vec{p}) \xrightarrow{a} q = \sigma(t)$ where $p = \sigma(\mathsf{source}(\rho))$ and $t = \mathsf{target}(\rho)$ and $\sigma \models \mathsf{ante}(\rho)$. For each positive antecedent $x_i \xrightarrow{b} y$ of ρ, we have $p_i \xrightarrow{b} \sigma(y)$. Hence, $p' \xrightarrow{b} \sigma'(y)$ for some $\sigma'(y) \mathrel{\underline{\leftrightarrow}} \sigma(y)$. Furthermore, if $p_i \stackrel{b}{\not\longrightarrow}$, then $p'_i \stackrel{b}{\not\longrightarrow}$ as well. In particular, ρ applies to $f(\vec{p}')$, giving a transition $f(\vec{p}') \xrightarrow{a} \sigma'(\mathsf{target}(\rho))$. By the first paragraph of this proof, we have $\sigma(\mathsf{target}(\rho)) \mathrel{\underline{\leftrightarrow}} \sigma'(\mathsf{target}(\rho))$. As this holds for all transitions from $f(\vec{p})$, and (by symmetry) from $f(\vec{p}')$, we have shown that $f(\vec{p}) \mathrel{\underline{\leftrightarrow}_{sr}} f(\vec{p}')$. ♀

4.2 Rooted Weak Bisimulation

We now adapt the tame/wild idea to rooted weak bisimulation proper. First, we give a more local definition of rooted weak bisimulation; then, exploiting that definition, a rule format respecting it.

Definition 4.4 *A symmetric relation $\sim$ is a* local rooted weak bisimulation *if, whenever $p \sim p'$ and $p \xrightarrow{\alpha} q$, then there is a q' such that $p' \xrightarrow{\tau^*\alpha\tau^*} q' \underline{\leftrightarrow} q$. Two processes are locally rootedly weakly bisimilar, $p \underline{\leftrightarrow}_{lr} p'$ iff there is a local rooted weak bisimulation relating them.*

As with ordinary $\underline{\leftrightarrow}_r$, this is a direct definition in terms of $\underline{\leftrightarrow}$. Local and ordinary rooted weak bisimulation differ formally in that the local relation only looks at a single step of one process, whilst the ordinary relation allows any number of τ steps of the other.

Lemma 4.5 *Local rooted weak bisimulation coincides with rooted weak bisimulation; that is, $p \underline{\leftrightarrow}_{lr} p'$ iff $p \underline{\leftrightarrow}_r p'$.*

Proof: Suppose that $p \underline{\leftrightarrow}_{lr} p'$, and that $p \xrightarrow{\tau^n\alpha\tau^*} r$. We must show that there is a r' such that $p' \xrightarrow{\tau^*\alpha\tau^*} r' \underline{\leftrightarrow} r$. If $n = 0$, then $p \xrightarrow{\alpha} q \xrightarrow{\tau^*} r$ for some q. By $p \underline{\leftrightarrow}_{lr} p'$, we have $p' \xrightarrow{\tau^*\alpha\tau^*} q' \underline{\leftrightarrow} q$. As $q \underline{\leftrightarrow} q'$, we have $q' \xrightarrow{\tau^*} r' \underline{\leftrightarrow} r$. Hence, $p' \xrightarrow{\tau^*\alpha\tau^*} r' \underline{\leftrightarrow} r$ as desired.

Otherwise, $n > 0$, and we have $p \xrightarrow{\tau} q \xrightarrow{\tau^+} r$. From $p \underline{\leftrightarrow}_{lr} p'$ we have $p' \xrightarrow{\tau^*\tau\tau^*} q' \underline{\leftrightarrow} q$, and from weak bisimulation $q' \xrightarrow{\tau^*\alpha\tau^*} r' \underline{\leftrightarrow} r$. Again, putting these together gives us $p' \xrightarrow{\tau^*\alpha\tau^*} r' \underline{\leftrightarrow} r$ as desired.

For the other direction, suppose that $p \underline{\leftrightarrow}_r p'$, and $p \xrightarrow{\alpha} q$. We thus have $p \xrightarrow{\tau^*\alpha\tau^*} q$, with both τ^*'s being empty; hence, the rooted weak bisimulation condition applies, and $p' \xrightarrow{\tau^*\alpha\tau^*} q' \underline{\leftrightarrow} q$ as desired. $\clubsuit$

We can use the characterization as local rooted weak bisimulation to deduce a rule format which respects $\underline{\leftrightarrow}_r$. Some operations, the *tame* ones, will respect $\underline{\leftrightarrow}$; in CCS, this is all operations except $+$. The remaining operations, the *wild* ones, like $+$ need not respect weak bisimulation; but they must have ruloids which force them to respect local rooted weak bisimulation.

For example, suppose that for each rule

$$\frac{x \xrightarrow{a} y}{f(x,z) \xrightarrow{b} t}$$

that there are ruloids of the form:

$$\frac{x \xrightarrow{\tau} y}{f(x,z) \xrightarrow{\tau} t} \qquad \frac{y \xrightarrow{a} z}{t \xrightarrow{b} t[y := z]} \qquad \frac{y \xrightarrow{\tau} z}{t \xrightarrow{\tau} t[y := z]} \tag{17}$$

where t is some tame term. For example, if f were $+$, t would be y.

Suppose $p \underline{\leftrightarrow}_{lr} p'$. Intuitively, the first rule allows $f(p',r)$ to take a τ-transition to an instance of t, handling the first τ moves of p'. The second ruloid lets t do the transition that f did; the third ruloid handles the final τ moves of p'. Suppose we have transitions:

$$p \xrightarrow{\qquad\qquad a \qquad\qquad} q$$

$$p' \xrightarrow{\tau} p'_1 \xrightarrow{a} q'_1 \xrightarrow{\tau} q'$$

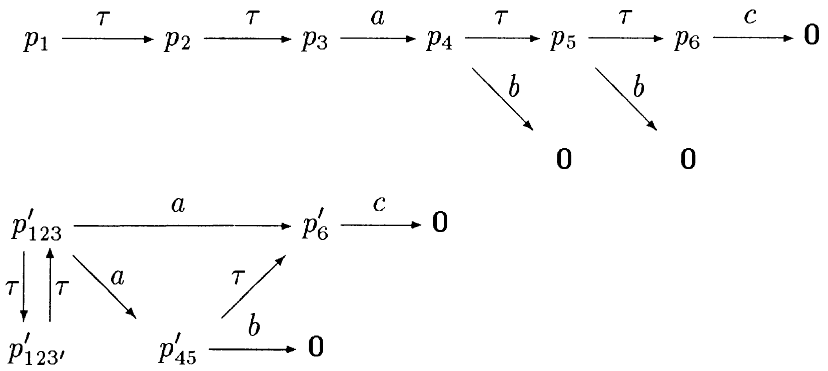


Figure 5: Rooted Weakly Bisimilar Processes

The above rules give (roughly)

$$f(p,r) \xrightarrow{\qquad\qquad\qquad b \qquad\qquad\qquad} t[y := q]$$

$$f(p',r) \xrightarrow{\tau} t[y := p_1'] \xrightarrow{b} t[y := q_1'] \xrightarrow{\tau} t[y := q']$$

In general, though, we need some more structure. Suppose that we have the operation f, whose rules include

$$\frac{x \xrightarrow{a} y}{f(x) \xrightarrow{b} d.y} \qquad \frac{}{f(x) \xrightarrow{d} x}$$

where $d.y$ is prefixing, (3). For example, $f(a.c) \underline{\leftrightarrow} b.d.c + d.a.c$. Consider the processes $p = \tau\tau a(b + \tau(b + \tau c))$ and $p' = a(b + \tau c) + ac + \tau.\tau.p'$; see Figure 5, where the names of the states show the correspondence between the processes. We have $p \underline{\Leftrightarrow}_r p'$. (These processes are chosen to illustrate all the difficulties that arise in the general case.) We will have $f(p) \underline{\Leftrightarrow}_r f(p')$, but this will require several new operations and a new rule for f.

Let us try to match the transition $f(p') \xrightarrow{b} d.c$ with some computation from $f(p)$. We have to do several things. First of all, we need to absorb the transition $p_1 \xrightarrow{\tau} p_2$. When we do this, we lose the option of performing the d action that is available to $f(p)$; we have committed to the transition $f(\ldots) \xrightarrow{\tau^* b \tau^*}$, though it is not yet available. Thus we must have a transition $f(p) \xrightarrow{\tau} g(p_2)$ for some new function g.

g will wait patiently for its argument to perform an a, with a transition $g(p_2) \xrightarrow{\tau} g(p_3)$, and when it does, will perform something like f's a transition: $g(p_3) \xrightarrow{b} h(p_4)$. Note that we cannot simply have $g(p_3)$ move to $d.p_4$, as the latter has the option of eventually doing the string db, but $f(p')$'s child $d.c$ cannot.

The function h intuitively means that we have performed the $f(\ldots) \xrightarrow{\tau^* b \tau^*}$ and are trying to get into a state in which it is appropriate to get to its target $d.c$. It will discard τ actions, in a transition $h(p_4) \xrightarrow{\tau} h(p_5)$. When the process is about to get to the right state, h will become the target of f's original transition: $h(p_5) \xrightarrow{\tau} d.p_6 \underline{\leftrightarrow} d.c$.

Thus, in general, to match a transition from f, we need two intermediate states — g and h in this example — which represent the intent to do that transition, with g being the state before the transition has happened, and h the state afterwards. This rather complicated example inspires the following equally complicated definition. Given the top transition $f \xrightarrow{\alpha} t$ in this diagram, we insist that the rest be possible as well (where there are τ loops at t^1 and t^2):

$$
\begin{tikzcd}
f \arrow[r, "\alpha"] \arrow[d, "\tau"'] \arrow[dr, "\alpha"] & t \arrow[dl, "\alpha"'] \\
\subset t^1 \arrow[r, "\alpha"] & t^2 \supset \arrow[u, "\tau"']
\end{tikzcd}
$$

Definition 4.6 *A positive GSOS language is RWB cool if the operations can be partitioned into* tame *and* wild *operations, such that:*

1. *The target of every rule contains only tame operations.*

2. *The sublanguage of tame operations is WB cool.*

3. *For every rule ρ with target $f(\vec{x}) \xrightarrow{\alpha} t$ for a wild operation symbol f, the following exist:*

   (a) *Terms t^1 and t^2, such that $\mathsf{Vars}(t)$, $\{\vec{x}\}$, $\mathsf{Vars}\left(t^1\right)$, and $\mathsf{Vars}\left(t^2\right)$ are mutually disjoint.*

   (b) *Bijections $\frac{\mathsf{y}}{\mathsf{x1}}(\cdot) : \mathsf{Vars}\left(t^1\right) \to \mathsf{Vars}(t)$ and $\frac{\mathsf{y}}{\mathsf{x2}}(\cdot) : \mathsf{Vars}\left(t^2\right) \to \mathsf{Vars}(t)$. (Let $\frac{\mathsf{x1}}{\mathsf{y}}(\cdot)$ and $\frac{\mathsf{x2}}{\mathsf{y}}(\cdot)$ be the inverses, and let $\frac{\mathsf{x}}{\mathsf{x1}}\left(x^1\right) = \frac{\mathsf{x}}{\mathsf{y}}\left(\frac{\mathsf{y}}{\mathsf{x1}}\left(x^1\right)\right)$.)*

   (c) *The following ruloids, for each x_0 which is active in $f(\vec{x})$, each x_0^1 such that $x_0 = \frac{\mathsf{x}}{\mathsf{x1}}\left(x_0^1\right)$, and each x_0^2 such that $x_0 \xrightarrow{\beta_0} y_0 \in \mathsf{ante}(\rho)$ and $\frac{\mathsf{y}}{\mathsf{x2}}\left(x_0^2\right) = y^0$*

$$
\frac{x_0 \xrightarrow{\tau} x_0^1, \left\{x \xrightarrow{\varepsilon} x^1 \middle| x = \frac{\mathsf{x}}{\mathsf{x1}}\left(x^1\right) \wedge \left(x \neq x_0 \vee x^1 \neq x_0^1\right)\right\}}{f(\vec{x}) \xrightarrow{\tau} t^1} \tag{18}
$$

$$
\frac{\left\{x \xrightarrow{\zeta} x^2 \middle| \left(x \xrightarrow{\zeta} y\right) \in \mathsf{ante}(\rho) \wedge y = \frac{\mathsf{y}}{\mathsf{x2}}\left(x^2\right)\right\}}{f(\vec{x}) \xrightarrow{\alpha} t^2} \tag{19}
$$

$$
\frac{x_0^1 \xrightarrow{\tau} x_0^{1\prime}, \left\{x^1 \xrightarrow{\varepsilon} x^{1\prime} \middle| x^1 \neq x_0^1\right\}}{t^1 \xrightarrow{\tau} t^{1\prime}} \tag{20}
$$

where $t^{1\prime} = t^1[\vec{x}^1 := \vec{x}^{1\prime}]$

$$\frac{\left\{x^1 \xrightarrow{\varsigma} x^2 | x \xrightarrow{\varsigma} y \in \mathsf{ante}(\rho), x = \frac{\mathsf{x}}{\mathsf{x1}}\left(x^1\right), y = \frac{\mathsf{y}}{\mathsf{x2}}\left(x^2\right)\right\}}{t^1 \xrightarrow{\alpha} t^2} \tag{21}$$

$$\frac{\left\{x^1 \xrightarrow{\varsigma} y | x \xrightarrow{\varsigma} y \in \mathsf{ante}(\rho), x = \frac{\mathsf{x}}{\mathsf{x1}}\left(x^1\right)\right\}}{t^1 \xrightarrow{\alpha} t} \tag{22}$$

$$\frac{x_0^2 \xrightarrow{\tau} x_0^{2\prime}, \left\{x^2 \xrightarrow{\varepsilon} x^{2\prime} | x^2 \neq x_0^2\right\}}{t^2 \xrightarrow{\tau} t^{2\prime}} \tag{23}$$

where $t^{2\prime} = t^2[\vec{x}^2 := \vec{x}^{2\prime}]$

$$\frac{x_0^2 \xrightarrow{\tau} y_0, \left\{x^2 \xrightarrow{\varepsilon} y | y = \frac{\mathsf{y}}{\mathsf{x2}}\left(x^2\right) \neq y^0\right\}}{t^2 \xrightarrow{\tau} t} \tag{24}$$

In most cases, t^1 and t^2 will simply be renamings of t; for $+$, where t is a target variable x', t^1 and t^2 are other variables y' and z'.

Theorem 4.7 *Rooted weak bisimulation is a congruence with respect to all RWB cool languages.*

Proof: Let $\mathcal{L}$ be a RWB cool language. It is clear that if t is a univariate term containing only tame operations, and $\sigma(x) \underline{\Leftrightarrow} \sigma^\bullet(x)$ for each x, then

$$\sigma(t) \underline{\Leftrightarrow} \sigma^\bullet(t) \tag{25}$$

Let $p_i \underline{\Leftrightarrow}_r p_i^\bullet$ be vectors of closed terms of $\mathcal{L}$. It suffices to show that $p = f(\vec{p}) \underline{\Leftrightarrow}_{lr} f(\vec{p^\bullet}) = p^\bullet$. Suppose that $p \xrightarrow{\alpha} q$, and let ρ be the rule for f enabling this transition. Let $\sigma(x_i) = p_i$ and $\sigma^\bullet(x_i) = p_i^\bullet$. For each antecedent $x \xrightarrow{\varsigma} y$ of f, we have $\sigma(x) \xrightarrow{\varsigma} \hat{\sigma}(y)$ for some $\hat{\sigma}$. By local weak bisimulation, let $x^1 = \frac{\mathsf{x1}}{\mathsf{y}}(y)$ and $x^2 = \frac{\mathsf{x2}}{\mathsf{y}}(y)$

$$\sigma^\bullet(x) \xrightarrow{\tau^{m_y}} \sigma_b^\bullet\left(x^1\right) \xrightarrow{\varsigma} \sigma_c^\bullet\left(x^2\right) \xrightarrow{\tau^{n_y}} \hat{\sigma}^\bullet(y)$$

where $\hat{\sigma}(y) \underline{\Leftrightarrow} \hat{\sigma}^\bullet(y)$ for each y. What happens next depends on whether or not there are τ actions.

$\forall y. m_y = n_y = 0$: In this case, we have $\sigma^\bullet(x) \xrightarrow{\varsigma} \hat{\sigma}^\bullet(y)$ for each antecedent $x \xrightarrow{\varsigma} y$ of ρ. Hence ρ applies to $p^\bullet$, giving a transition $p^\bullet \xrightarrow{\alpha} q^\bullet = \hat{\sigma}^\bullet(t)$, and $q \underline{\Leftrightarrow} q^\bullet$ by (25).

$\forall y. m_y = 0$, **and** $\exists y_0. n_{y_0} > 0$: In this case, use (19) to get a transition

$$p^\bullet = \sigma^\bullet(f(\vec{x})) \xrightarrow{\beta} \sigma_c^\bullet(t^2)$$

Now, apply the rules (23) $\left(\sum_y n_y\right) - 1$ times, giving us transitions

$$\sigma_c^\bullet(t^2) \xrightarrow{\tau^*} \sigma_d^\bullet(t^2) \tag{26}$$

where $\sigma_c^\bullet(x_0^2) \xrightarrow{\tau^{n_{y_0}-1}} \sigma_d^\bullet(x_0^2) \xrightarrow{\tau} \hat{\sigma}(y_0)$, and $\sigma_d^\bullet(x^2) = \hat{\sigma}^\bullet\left(\frac{\mathsf{y}}{\mathsf{x2}}\left(x^2\right)\right)$ for all other x^2. Then, apply the rule (24), giving a transition

$$\sigma_d^\bullet(t^2) \xrightarrow{\tau} \hat{\sigma}^\bullet(t) = q^\bullet. \tag{27}$$

From (25), we have $q \Leftrightarrow q^\bullet$. That is, $p^\bullet \xrightarrow{\alpha\tau^*} q^\bullet$ as desired.

$\exists y_0.m_{y_0} > 0$: For this case, we use (18) to get a transition

$$p^\bullet \xrightarrow{\tau} \sigma_a^\bullet(t^1)$$

where $\sigma^\bullet(x_0) \xrightarrow{\tau} \sigma_a^\bullet\left(\frac{\mathsf{x1}}{\mathsf{y}}(y_0)\right) \xrightarrow{\tau^{m_{y_0}-1}} \sigma_b^\bullet\left(\frac{\mathsf{x1}}{\mathsf{y}}(y_0)\right)$. Let $\sigma_a^\bullet(x^1) = \sigma^\bullet\frac{\mathsf{x}}{\mathsf{x1}}\left(x^1\right)$ for all other x^1. We then use (20) $\left(\sum_y m_y\right) - 1$ times, giving us transitions

$$\sigma_a^\bullet(t^1) \xrightarrow{\tau^*} \sigma_b^\bullet(t^1)$$

We are now ready to use one of the ruloids giving t^2 a α-action. There are two subcases, depending on whether or not $\exists y_1.m_{y_1} > 0$. If no such y_1 exists, then $\sigma_c^\bullet\left(x^2\right) = \hat{\sigma}^\bullet(y)$ for each $y = \frac{\mathsf{y}}{\mathsf{x2}}\left(x^2\right)$, and so we use (22) to give a transition

$$\sigma_b^\bullet(t^1) \xrightarrow{\alpha} \hat{\sigma}^\bullet(t) = q^\bullet \Leftrightarrow q$$

which proves the theorem in this case.

Otherwise, such a y^1 does exist. In this case, we use (21), giving a transition

$$\sigma_b^\bullet(t^1) \xrightarrow{\alpha} \sigma_c^\bullet(t^2)$$

We then proceed as for (26) and (27), giving $\sigma_c^\bullet(t^2) \xrightarrow{\tau^*} q^\bullet \Leftrightarrow q$.

Thus, rooted weak bisimulation is a congruence ♀

4.3 Discussion

The SRWB cool and RWB cool formats provide an amazing degree of programming power for a language respecting silent bisimulation. They allow operations like $+$, and even some operations which require negative rules: *e.g.*, a one-step priority operator, which gives the action a priority over b:

$$\frac{x \xrightarrow{a} x'}{\theta 1(x) \xrightarrow{a} x'} \qquad \frac{x \xrightarrow{b} x', x \stackrel{a}{\not\longrightarrow}}{\theta 1(x) \xrightarrow{b} x'}$$

Note that the full priority operator,

$$\frac{x \xrightarrow{a} x'}{\theta(x) \xrightarrow{a} \theta(x')} \qquad \frac{x \xrightarrow{b} x', x \xrightarrow{a}\!\!\!\!\!/\;\;}{\theta(x) \xrightarrow{b} \theta(x')}$$

which is not cool. The negative rule forces θ to be wild, but the targets of both rules use θ and thus it must be tame. However, this operation does not respect $\underline{\Leftrightarrow}_{sr}$. Recall the process P_{ba} of (5), which alternately offered a b and an a transition; recall that $P_{ba} \underline{\Leftrightarrow}\ a + b$, though they are not strongly rooted weakly bisimilar. We have $\theta(P_{ba}) \xrightarrow{b} \theta(\mathbf{0})$, but $\theta(a+b) \xrightarrow{b}\!\!\!\!\!/\;\;$ and $\theta(a+b) \xrightarrow{\tau}\!\!\!\!\!/\;\;$; hence θ does not respect weak bisimulation. As $P_{ba} \underline{\Leftrightarrow}\ a+b$, we have $aP_{ba} \underline{\Leftrightarrow}_{sr}\ a(a+b)$. But $\theta(aP_{ba}) \xrightarrow{a} \theta(P_{ba})$ and $\theta(a(a+b)) \xrightarrow{a} \theta(a+b)$, and hence the two are not strongly rooted weak bisimilar.

5 SOSses for branching bisimulation

Branching bisimulation is a finer relation than weak bisimulation. Not surprisingly, a wider class of operations respect it. Here, for example, is an operation which respects branching bisimulation but not weak bisimulation:

$$\frac{x \xrightarrow{a} x'}{h(x) \xrightarrow{a} \mathbf{0}} \qquad \frac{x \xrightarrow{b} x'}{h(x) \xrightarrow{b} x} \qquad \frac{x \xrightarrow{\tau} x'}{h(x) \xrightarrow{\tau} h(x')}$$

Note that this is almost a WB cool operation, with $h^\star = h$. However, the target of the b-rule is x rather than x'; the rule is not ε-presented, and if it were it would not be straight, and thus we would not be allowed to have $h^\star = h$.

Let $p = a + \tau b$ and $q = p + b$. It is easy to verify that $p \underline{\Leftrightarrow}\ q$, but $p \not\underline{\Leftrightarrow}_b\ q$. We have $h(p) \leftrightarrow a + \tau bb$, and $h(q) \xrightarrow{b} q \xrightarrow{a} \mathbf{0}$. In particular, $h(q) \xrightarrow{ba}$, but $h(p)$ cannot take these actions even with τ's interspersed. The two are thus distinguishable by h.

We define fully RB cool languages and related terms in much the same way that we defined fully WB cool languages. Indeed the definition differs only in two places from Definition 3.5: $\mathcal{L}$ is a GSOS language (without ε-transitions) rather than a GSOS(ε) language (requiring ε-transitions.), and the matching between targets, part 4 is slightly different than for Definition 3.5. $\mathsf{barb}(t,x)$ must be redefined: $\mathsf{barb}(t,x)$ must be defined as 1 if no rule for t has x as an antecedent.

Definition 5.1 *A GSOS language $\mathcal{L}$ is* Fully RB cool *if there is a mapping $(\cdot)^\star : \mathsf{Terms}_1 \to \mathsf{Terms}_1^s$, and, for each $t \in \mathsf{Terms}_1$, a mapping $\frac{\mathsf{x}}{\mathsf{x}'}\Big|_t(\cdot) : \mathsf{Vars}(t^\star) \to \mathsf{Vars}(t)$. We use the same auxiliary functions as in Section 3.*

1. *If t is straight, then $\frac{\mathsf{x}}{\mathsf{x}'}(t^\star) = t$.*

2. *If x is active in t, then for all $x'_0 \in \frac{\{\mathsf{x}'\}}{\mathsf{x}}(x)$, there is a ruloid $\mathsf{bi}^t_{x'_0} \in \mathcal{R}(t,\tau)$:*

$$\frac{x \xrightarrow{\tau} v}{t \xrightarrow{\tau} \varphi_{x'_0}(t^\star)} \tag{28}$$

where v is a fresh variable, and

$$\varphi_{x'_0}(x') = \begin{cases} \frac{\mathsf{x}}{\mathsf{x'}}(x') & x' \neq x'_0 \\ v & x' = x'_0 \end{cases}$$

$\mathsf{bi}^{t}_{x'_0}$ *is the* bifurcation *ruloid of t for x'_0; if t is straight, it is also called the* patience *ruloid for x'_0.*

3. *The only ruloids with τ's in the antecedent are the bifurcation ruloids.*

4. *For all t and α, there is a bijection $(\cdot)^{\star} : \mathcal{R}(t,\alpha) \to \mathcal{R}(t^{\star},\alpha)$ such that:*

$$\begin{aligned} \mathsf{ante}(\rho) &= \tfrac{\mathsf{x}}{\mathsf{x'}}(\mathsf{ante}(\rho^{\star})) & (29) \\ \mathsf{target}(\rho) &= \tfrac{\mathsf{x}}{\mathsf{x'}}(\mathsf{target}(\rho^{\star})) & (30) \end{aligned}$$

The functions $\frac{\{\mathsf{x'}\}}{\mathsf{x}}(x)$, $\frac{\mathsf{x'}}{\mathsf{y}}(y)$, and $\frac{\mathsf{x}}{\mathsf{y}}(y)$, are defined as before.

Definition 5.2 *If $\dot{t}$ is not a univariate term, we define $\dot{t}^{\star}$ as follows. For each variable x, choose an infinite number of variables $x_1, x_2, \ldots$, distinct for all x and i, and φ_{strip} the function sending each x_i back to x. Let t be $\dot{t}$ with the i'th occurrence of each variable x replaced by x_i. We define $\dot{t}^{\star}$ to be $t^{\star}$, and $\frac{\mathsf{x}}{\mathsf{x'}}\Big|_{\dot{t}}(x') = \varphi_{\mathsf{strip}}\left(\frac{\mathsf{x}}{\mathsf{x'}}\Big|_{t}(x')\right)$.*

Definition 5.3 *We define $p \smile p^{\bullet}$ to hold if:*

1. *$p \leftrightarrow_b p^{\bullet}$, or*

2. *There is a term t, and substitutions σ and $\sigma^{\bullet}$ on $\mathsf{Vars}(t)$, such that $\forall x \in \mathsf{Vars}(t).\sigma(x) \smile \sigma^{\bullet}(x)$ and $p = \sigma(t)$ and $p^{\bullet} = \sigma^{\bullet}(t)$.*

3. *There is a term t, and substitutions σ on $\mathsf{Vars}(t)$ and $\sigma^{\bullet}$ on $\mathsf{Vars}(t^{\star})$ such that $p = \sigma(t)$ and $p^{\bullet} = \sigma^{\bullet}(t^{\star})$, and $\forall x' \in \mathsf{Vars}(t^{\star}).\sigma(\frac{\mathsf{x}}{\mathsf{x'}}(x')) \smile \sigma^{\bullet}(x')$.*

4. *$p^{\bullet} \smile p$ by 3.*

Note that $\smile$ is symmetric.

Theorem 5.4 *Branching bisimulation is a congruence for every RB cool language.*

Proof: As with Theorem 3.9, it suffices to show that $\smile$ is a branching bisimulation relation; that is, whenever $p \xrightarrow{\alpha} r$, that either

$\boxed{\tau}$: $\alpha = \tau$ and $r \smile p^{\bullet}$:

$$\begin{tikzcd} p \arrow[r, "\tau"] \arrow[d, dotted, no head, "\smile"'] & r \arrow[dl, dotted, no head, "\smile"] \\ p^{\bullet} & \end{tikzcd}$$

or

$\boxed{\triangle}$: there are $q^\bullet$, $r^\bullet$ such that $p^\bullet \xrightarrow{\tau^*} q^\bullet \xrightarrow{\alpha} r^\bullet$, and $p \smile q^\bullet$ and $r \smile r^\bullet$:

$$\begin{tikzcd}
p \arrow[rr, "\alpha"] \arrow[d, dotted, "\smile"'] \arrow[dr, dotted, "\smile"] & & r \arrow[d, dotted, "\smile"] \\
p^\bullet \arrow[r, "\tau^*"] & q^\bullet \arrow[r, "\alpha"] & r^\bullet
\end{tikzcd}$$

The proof that $\smile$ is a branching bisimulation relation is by induction on the reason that $p \smile p^\bullet$, plus an extensive case analysis. Suppose that $p \smile p^\bullet$, and that $p \xrightarrow{\alpha} r$. We use some more diacritical marks beyond Figure 4:

σ_+	The substitution which will prove the theorem
$\ddot{u}$	The target of $\rho^\star$, which in this setting is different from that of ρ.

1: This case is trivial.

2: Suppose that $p = \dot{\sigma}(\dot{t}) \smile \dot{\sigma}^\bullet(\dot{t}) = p^\bullet$, where $\dot{\sigma}(x) \smile \dot{\sigma}^\bullet(x)$ for each $x \in \mathsf{Vars}(\dot{t})$. Now, $\dot{t}$ need not be univariate. However, by definition, there is a univariate term t and substitution $\psi : \mathsf{Vars}(t) \to \mathsf{Vars}(\dot{t})$ such that $\psi(t) = \dot{t}$. Let $\sigma = \dot{\sigma} \circ \psi$ and $\sigma^\bullet = \dot{\sigma}^\bullet \circ \psi$.

Let ρ be the ruloid for t and $\hat{\sigma} \supseteq \sigma$ be the instantiation giving $p \xrightarrow{\alpha} r$. Let $u = \mathbf{target}(\rho)$; note that $r = \hat{\sigma}(u)$. There are two cases, depending on the kind of ruloid ρ is.

2.bifurcation: In this case, for some $x_0 \in \mathsf{Vars}(t)$ and $x_0' \in \mathsf{Vars}(t^\star)$, we have

$$\rho = \frac{x_0 \xrightarrow{\tau} y_0}{t \xrightarrow{\tau} \varphi_{x_0'}(t^\star)}$$

As $\sigma(x_0) \smile \sigma^\bullet(x_0)$ and $\sigma(x_0) \xrightarrow{\tau} \hat{\sigma}(y_0)$, the induction hypothesis shows that $\sigma^\bullet(x_0)$ matches $\sigma(x_0)$'s move. There are two cases, depending on how $\sigma^\bullet(x_0)$ matches $\sigma(x_0)$'s transition.

2.bifurcation.$\boxed{\tau}$: In this case, we have $\hat{\sigma}(y_0) \smile \sigma^\bullet(x_0)$. Let $\sigma_+(x') = \hat{\sigma}(\varphi_{x_0'}(x'))$. We have $\sigma_+(x') \smile \sigma^\bullet(\frac{\mathsf{x}}{\mathsf{x}'}(x'))$ for all $x' \in \mathsf{Vars}(t^\star)$. Hence $r = \sigma_+(t^\star) \smile \sigma^\bullet(t) = p$ by part 4 of the definition of $\smile$.

2.bifurcation.$\boxed{\triangle}$: This case is like **2.action** below, where $\beta_0 = \tau$, and there is only one antecedent of the rule.

2.action: In this case, ρ is any non-bifurcation ruloid. Let $u = \mathbf{target}(\rho)$. For each antecedent $x \xrightarrow{\beta} y$ of ρ, we have:

$$\begin{tikzcd}
\hat{\sigma}(x) \arrow[rr, "\beta"] \arrow[d, dotted, "\smile"'] \arrow[dr, dotted, "\smile"] & & \hat{\sigma}(y) \arrow[d, dotted, "\smile"] \\
\sigma^\bullet(x) \arrow[r, "\tau^*"] & \hat{\sigma}^\bullet(x') \arrow[r, "\beta"] & \hat{\sigma}^\bullet(y)
\end{tikzcd}$$

where $x' = \frac{\mathsf{x}'}{\mathsf{y}}(y)$. There are two subcases, depending on whether or not there are any τ-steps in any transition $\sigma^\bullet(x) \xrightarrow{\tau^*} \acute{\sigma}^\bullet(x')$.

2.action.no: If there are no τ-steps in any transition $\sigma^\bullet(x) \xrightarrow{\tau^*} \acute{\sigma}^\bullet(x')$, then we have $\sigma^\bullet(x) \xrightarrow{\beta} \hat{\sigma}^\bullet(y)$ for each antecedent $x \xrightarrow{\beta} y$ of ρ. So, ρ applies directly to $\sigma^\bullet(t)$, giving us a transition $p^\bullet \xrightarrow{\alpha} \hat{\sigma}^\bullet(u) = r^\bullet$. Let $q^\bullet = p^\bullet$; then we have $\boxed{\triangle}$ as desired.

2.action.yes: Otherwise, for some antecedent $x_0 \xrightarrow{\beta_0} y_0$ of ρ, we have $\sigma^\bullet(x_0) \xrightarrow{\tau} \sigma_1^\bullet(x_0') \xrightarrow{\tau^*} \acute{\sigma}^\bullet(x_0') \xrightarrow{\beta_0} \hat{\sigma}^\bullet(y_0)$. Let $\sigma_1^\bullet(x') = \sigma^\bullet(\frac{\mathsf{x}}{\mathsf{x}'}(x'))$ for all $x' \in \mathsf{Vars}(t^\star) - \{x'\}$. We apply the bifurcation ruloid $\mathsf{bi}^t_{x_0'}$, which gives a transition

$$\sigma^\bullet(t) \xrightarrow{\tau} \sigma_1^\bullet(t^\star) \tag{31}$$

We now apply patience ruloids for $t^\star$ to move all the $\sigma_1^\bullet(x')$'s to $\acute{\sigma}^\bullet(x')$'s:

$$\sigma_1^\bullet(t^\star) \xrightarrow{\tau^*} \acute{\sigma}^\bullet(t^\star) \tag{32}$$

Note that $p = \sigma(t) \smile \acute{\sigma}^\bullet(t^\star) = q^\bullet$ by Definition 5.3 part 3.
The ruloid $\rho^\star$ applies to $\acute{\sigma}^\bullet(t^\star)$, and we have the transition:

$$\acute{\sigma}^\bullet(t^\star) \xrightarrow{\alpha} \hat{\sigma}^\bullet(\ddot{u}). \tag{33}$$

where $\ddot{u} = \mathsf{target}(\rho^\star)$. Recall that $\frac{\mathsf{x}}{\mathsf{x}'}\Big|_t(\ddot{u}) = u$. Let $r^\bullet = \hat{\sigma}^\bullet(u)$. Let $\sigma_+(y) = \hat{\sigma}(y)$ for each y, and $\sigma_+(x') = \sigma(\frac{\mathsf{x}}{\mathsf{x}'}(x'))$ for each x'; then $\sigma_+(\ddot{u}) = \hat{\sigma}(u) = r$. We have $\sigma_+(z) \smile \hat{\sigma}^\bullet(z)$ for each $z \in \vec{x}' \cup \vec{y}$, and hence $r \smile r^\bullet$. This completes the $\boxed{\triangle}$ diagram as desired.

3: In this case, we have $p = \dot{\sigma}(\dot{t})$ and $p^\bullet = \dot{\sigma}^\bullet(\dot{t}^\star)$ for some term $\dot{t}$. As before, we uniquify variables, giving us a univariate term t and substitutions σ and $\sigma^\bullet$ such that $p = \sigma(t)$ and $p^\bullet = \sigma^\bullet(t^\star)$. Let ρ be the ruloid for t and $\hat{\sigma} \supseteq \sigma$ the instantiation giving $p \xrightarrow{\alpha} r$. Let $u = \mathsf{target}(\rho)$, and note that $r = \hat{\sigma}(u)$. There are two cases, depending on whether ρ is a bifurcation or an action ruloid.

3.bifurcation: In this case, ρ has the form

$$\rho = \frac{x_0 \xrightarrow{\tau} y_0}{t \xrightarrow{\tau} \varphi_{x_0'}(t^\star)}$$

We have

$$r = \hat{\sigma}(\varphi_{x_0'}(t^\star))$$

and

$$\rho^\star = \frac{x_0' \xrightarrow{\tau} y_0}{t^\star \xrightarrow{\tau} \varphi_{x_0'}(t^\star)}$$

There are two subsubcases, depending on how $\sigma^\bullet(x_0')$ matches $\sigma(x_0)$'s move.

3.bifurcation.$\boxed{\tau}$: In this case, $\hat{\sigma}(y_0) \smile \sigma^\bullet(x'_0)$. Let $\sigma_+ = \hat{\sigma} \circ \varphi_{x'_0}$. Then for all $x' \in \mathsf{Vars}(t^\star)$, we have $\sigma_+(x') \smile \sigma^\bullet(x')$. Hence $r = \sigma_+(t^\star) \smile \sigma^\bullet(t^\star) = p$ by Definition 5.3 part 2.

3.bifurcation.$\boxed{\triangle}$: This case is about the same as **3.action** below.

3.action: For each antecedent $x \xrightarrow{\beta} y$ of ρ, we have a diagram:

$$\begin{tikzcd}
\hat{\sigma}(x) \arrow[rr, "\beta"] \arrow[d, dash, dotted, "\smile"'] \arrow[dr, dash, dotted, "\smile"] & & \hat{\sigma}(y) \arrow[d, dash, dotted, "\smile"] \\
\sigma^\bullet(x') \arrow[r, "\tau^*"] & \acute{\sigma}^\bullet(x') \arrow[r, "\beta"] & \hat{\sigma}^\bullet(y)
\end{tikzcd}$$

where $x' = \frac{\mathsf{X}'}{\mathsf{Y}}(y)$. Unlike 2, we do not need any subcases, as $p^\bullet$ is already bifurcated. For each antecedent $x' \xrightarrow{\beta} y$ of $\rho^\star$, there is a patience ruloid for $t^\star$, giving us transitions

$$p^\bullet = \sigma^\bullet(t^\star) \xrightarrow{\tau^*} \acute{\sigma}^\bullet(t^\star) = q^\bullet \tag{34}$$

As $\acute{\sigma}(\frac{\mathsf{X}}{\mathsf{X}'}(x')) \smile \acute{\sigma}^\bullet(x')$ for each x', we have $p \smile q^\bullet$. Furthermore, the ruloid $\rho^\star$ applies to $\acute{\sigma}^\bullet(t^\star)$, giving a transition

$$q^\bullet = \acute{\sigma}^\bullet(t^\star) \xrightarrow{\alpha} \hat{\sigma}^\bullet(\ddot{u}) = r^\bullet$$

where $\ddot{u} = \mathsf{target}(\rho^\star)$.

We have $\hat{\sigma}(y) \smile \hat{\sigma}^\bullet(y)$ for each target variable y, and $\hat{\sigma}(\frac{\mathsf{X}}{\mathsf{X}'}(x')) \smile \hat{\sigma}^\bullet(x')$ for each x'. Let $\sigma_+(y) = \hat{\sigma}(y)$ and $\sigma_+(x') = \hat{\sigma}(\frac{\mathsf{X}}{\mathsf{X}'}(x'))$; then $\sigma_+(\ddot{u}) = r$. Hence, $r \smile r^\bullet$ as desired.

4: In this case, we have $p = \dot{\sigma}(\dot{t}^\star)$ and $p^\bullet = \dot{\sigma}^\bullet(\dot{t})$ for some term $\dot{t}$. As before, we uniquify variables, giving us a univariate term t and substitutions σ and $\sigma^\bullet$ such that $p = \sigma(t^\star)$ and $p^\bullet = \sigma^\bullet(t)$.

Let $\rho^\star$ be the ruloid for $t^\star$ and $\hat{\sigma} \supseteq \sigma$ the instantiation giving $p \xrightarrow{\alpha} r$. Let $u = \mathsf{target}(\rho)$ and $\ddot{u} = \mathsf{target}(\rho^\star)$, and note that $r = \hat{\sigma}(\ddot{u})$.

We have two cases, depending on whether ρ is an action or bifurcation rule.

4.bifurcation: If ρ is the bifurcation rule for x'_i, then we must do cases on how $\sigma(x'_i) \smile \sigma^\bullet(x_i)$, where of course $x_i = \frac{\mathsf{X}}{\mathsf{X}'}(x'_i)$.

4.bifurcation.$\boxed{\tau}$: We have $\sigma(x'_i) \xrightarrow{\tau} \sigma_+(x'_i) \smile \sigma^\bullet(x_i)$. Let $\sigma_+(x'_j) = \sigma(x'_j)$ for all $j \neq i$. We have $q = \sigma^+(\ddot{u}) \smile \sigma^\bullet(u) = q^\bullet$ as desired.

4.bifurcation.$\boxed{\triangle}$: This case is much like case 4.action below.

4.action: For each antecedent $x' \xrightarrow{\beta} y$ of $\rho^\star$, we have

$$\begin{tikzcd}
\hat{\sigma}(x') \arrow[rr, "\beta"] \arrow[d, dash, dotted, "\smile"'] \arrow[dr, dash, dotted, "\smile"] & & \hat{\sigma}(y) \arrow[d, dash, dotted, "\smile"] \\
\sigma^\bullet(x) \arrow[r, "\tau^*"] & \acute{\sigma}^\bullet(x') \arrow[r, "\beta"] & \hat{\sigma}^\bullet(y)
\end{tikzcd}$$

where $x = \frac{\mathsf{x}}{\mathsf{x}'}(x')$. As usual, there are two cases, depending on whether or not there are any τ moves.

4.action.yes: In this case, for some x'_0, we have $\sigma^\bullet(x_0) \xrightarrow{\tau} \sigma_1^\bullet(x'_0) \xrightarrow{\tau^*} \acute{\sigma}^\bullet(x'_0)$ Let $\sigma_1^\bullet(x') = \sigma^\bullet(\frac{\mathsf{x}}{\mathsf{x}'}(x'))$ for $x' \neq x'_0$. Apply the bifurcation ruloid for x'_0 to $\sigma^\bullet(t)$, giving us the transition:

$$\sigma^\bullet(t) \xrightarrow{\tau} \sigma_1^\bullet(t^\star)$$

and then use patience ruloids for $t^\star$ to give transitions

$$\sigma_1^\bullet(t^\star) \xrightarrow{\tau^*} \acute{\sigma}^\bullet(t^\star) = q^\bullet$$

It is easily seen that $p = \sigma(t^\star) \smile \acute{\sigma}^\bullet(t^\star) = q^\bullet$. As usual, $\rho^\star$ applies to $\acute{\sigma}^\bullet(t^\star)$, and we have a transition

$$q^\bullet = \acute{\sigma}^\bullet(t^\star) \xrightarrow{\alpha} \hat{\sigma}^\bullet(\ddot{u}) = r^\bullet.$$

As $\hat{\sigma}(y) \smile \hat{\sigma}^\bullet(y)$ and $\sigma(x') \smile \acute{\sigma}^\bullet(x')$ for each y and x', we have $r \smile r^\bullet$ as desired.

4.action.no: In this case, ρ applies directly to $p^\bullet$. We have $\acute{\sigma}^\bullet(x') = \sigma^\bullet(\frac{\mathsf{x}}{\mathsf{x}'}(x'))$ for each x', and a transition

$$p^\bullet = q^\bullet = \sigma^\bullet(t) \xrightarrow{\alpha} \hat{\sigma}^\bullet(u) = r^\bullet.$$

Let $\sigma_+{}^\bullet(y) = \hat{\sigma}^\bullet(y)$ and $\sigma_+{}^\bullet(x') = \sigma^\bullet(\frac{\mathsf{x}}{\mathsf{x}'}(x'))$; then $\sigma_+{}^\bullet(\ddot{u}) = \hat{\sigma}^\bullet(u) = r^\bullet$. We thus have $r = \hat{\sigma}(\ddot{u}) \smile \sigma_+{}^\bullet(\ddot{u}) = r^\bullet$ as desired.

This completes the proof. ♀

5.1 Finitary Characterization: RB cool Languages

As in Section 3.1, we present a finitary characterization of Fully RB cool languages; indeed, we show that requiring the operation symbols to satisfy the Fully RB cool requirement suffices to guarantee that the whole language is Fully RB cool.

Definition 5.5 *A language $\mathcal{L}$ is RB cool iff the terms $f(\vec{x})$ satisfy the Fully RB cool condition.*

As before, we inductively construct $\mathcal{R}(t, \alpha)$ and $t^\star$ for all terms t, and show that they work properly.

Theorem 5.6 *Let $\mathcal{L}$ be a RB cool language. Then $\mathcal{L}$ is fully RB cool.*

Proof: Let

$$\mathcal{R}(x,\alpha) = \left\{ \frac{x \xrightarrow{\alpha} y}{x \xrightarrow{\alpha} y} \right\}.$$

Suppose, then, that $t = f(\vec{t})$ is a univariate term. Consider a rule ρ for f:

$$\rho = \frac{x_i \xrightarrow{\alpha_{ij}} y_{ij}}{f(\vec{x}) \xrightarrow{b} u}$$

Choose ruloids $\rho_{ij} \in \mathcal{R}(t_i, \alpha_{ij})$. These ruloids give conditions under which $t_i \xrightarrow{\alpha_{ij}}$. They have the form:

$$\rho_{ij} = \frac{\left\{z \xrightarrow{\alpha} w | z \xrightarrow{\alpha} w \in \mathsf{ante}(\rho_{ij})\right\}}{t_i \xrightarrow{\alpha_{ij}} u_{ij}}$$

Rename the variables as necessary in some fixed way so that $\mathsf{Vars}(u_{ij})$ are all disjoint, and the variables in ρ_{ij}'s are disjoint from those in ρ Let the ruloid $\bar{\rho}$ be:

$$\bar{\rho} = \frac{\bigcup_{ij} \mathsf{ante}(\rho_{ij})}{t \xrightarrow{c} u[x_i := t_i, y_{ij} := u_{ij}]}$$

Note that, by induction, the ruloids for $(f(\vec{x}))^\star$ are in the right correspondence with those for $f(\vec{x})$. That is, corresponding to ρ there is a ruloid:

$$\rho^\star = \frac{\left\{x'_{ij} \xrightarrow{\alpha_{ij}} y_{ij} | \tfrac{\mathsf{X}}{\mathsf{X}'}\left(x'_{ij}\right) = x_i, \quad x_i \xrightarrow{\alpha_{ij} y_{ij} \in \mathsf{ante}(\rho)} \right\}}{(f(\vec{x}))^\star \xrightarrow{c} \ddot{u}}$$

with $\tfrac{\mathsf{X}}{\mathsf{X}'}\Big|_{f(\vec{x})}(\ddot{u}) = u$ and to each ρ_{ij} there is a ruloid $\rho^\star_{ij}$:

$$\rho_{ij} = \frac{\left\{z' \xrightarrow{\alpha} w | \exists z \xrightarrow{\alpha} w \in \mathsf{ante}(\rho_{ij}). \tfrac{\mathsf{X}}{\mathsf{X}'}(z') = z\right\}}{t^\star_i \xrightarrow{\alpha_{ij}} \ddot{u}_{ij}}$$

with $\tfrac{\mathsf{X}}{\mathsf{X}'}(\ddot{u}_{ij}) = u_{ij}$. Let

$$t^\star = (f(\vec{x}))^\star[x'_i := t^\star_i]$$

Thus, there is a ruloid

$$\bar{\rho^\star} = \frac{\bigcup_{ij} \mathsf{ante}\left(\rho^\star{}_{ij}\right)}{t^\star \xrightarrow{b} \ddot{u}[x'_i := t^\star_i, y_{ij} := \ddot{u}_{ij}]}$$

Furthermore, we define $\tfrac{\mathsf{X}}{\mathsf{X}'}\Big|_t(z)$ to be the union of the $\tfrac{\mathsf{X}}{\mathsf{X}'}\Big|_{t_i}(\cdot)$ functions. That is, if $z \in \mathsf{Vars}(t)$, then $z \in \mathsf{Vars}(t_i)$ for precisely one i; we define $\tfrac{\mathsf{X}}{\mathsf{X}'}\Big|_t(z)$ for t to be $\tfrac{\mathsf{X}}{\mathsf{X}'}\Big|_{t_i}(z)$.

1. If t is straight, then so are all the t_i's, and f. Hence $\frac{\mathsf{X}}{\mathsf{X}'}(t_i^\star) = t_i$, and $\frac{\mathsf{X}}{\mathsf{X}'}(f(\vec{x})^\star) = f(\vec{x})$. That is, $f(\vec{x})^\star = f(\vec{x}')$. Hence $t^\star = f(\vec{t^\star})$, whence $\frac{\mathsf{X}}{\mathsf{X}'}(t^\star) = f(\frac{\mathsf{X}}{\mathsf{X}'}(\vec{t^\star})) = f(\vec{t})$ as desired.
2. Just as for this case of Theorem 3.12.
3. Just as for this case of Theorem 3.12.
4. We have $\mathsf{ante}(\rho^\star) = \bigcup \mathsf{ante}(\rho_{ij}^\star) = \bigcup \frac{\mathsf{X}}{\mathsf{X}'}(\mathsf{ante}(\rho_{ij})) = \frac{\mathsf{X}}{\mathsf{X}'}(\mathsf{ante}(\rho))$. Furthermore, $\mathsf{dom}\left(\frac{\mathsf{X}}{\mathsf{X}'}\Big|_t(\cdot)\right) \pitchfork \vec{x}$ by construction So,

$$\begin{aligned}
\tfrac{\mathsf{X}}{\mathsf{X}'}(\mathsf{target}(\rho^\star)) &= \tfrac{\mathsf{X}}{\mathsf{X}'}\left(\ddot{u}[x'_{ij} := t_i^\star, y_{ij} := \ddot{u}_{ij}]\right) \\
&= u[x'_{ij} := \tfrac{\mathsf{X}}{\mathsf{X}'}(t_i^\star), y_{ij} := \tfrac{\mathsf{X}}{\mathsf{X}'}(\ddot{u}_{ij})] \\
&= u[x_i := t_i, y_{ij} := u_{ij}]
\end{aligned}$$

as desired.

♆

6 Rooted Branching Bisimulation

Branching bisimulation suffers from the same kind of problem as weak bisimulation: some operations, + in particular, do not respect it. The solution is to use *rooted branching bisimulation*, Definition 1.6, which is adequate for all of CCS and most related languages. As with strongly rooted weak bisimulation, a rooted weak bisimulation relation gives us the full power of bisimulation on the first step of computation, and hence we may use the full power of GSOS rules.

Definition 6.1 *A GSOS language $\mathcal{L}$ is RBB cool iff the function symbols can be partitioned into* tame *and* wild *operations, such that*

1. *The sublanguage of $\mathcal{L}$ given by just the tame operations and all their rules is RB cool.*
2. *The targets of all rules in $\mathcal{L}$ are univariate terms mention only tame operations.*

Theorem 6.2 *Let $\mathcal{L}$ be a RBB cool language. Then $\leftrightarrow_{rb}$ is a congruence for $\mathcal{L}$.*

Proof: Just like Theorem 4.3. ♆

7 Equational Theories

In [ABV92a], we gave a general procedure for taking a GSOS language $\mathcal{L}$, and building an extension $\mathcal{L}'$ of $\mathcal{L}$ and an equational axiom system which, with the addition of one induction principle, is complete for proving strong bisimulations of processes.

[ABV92a] comprises a collection of methods for computing equations and, where necessary, auxiliary operations from the rules of $\mathcal{L}$. The central example is interleaving composition, (1). Interleaving is *smooth* but not *distinctive* — the full definitions are not relevant yet — and the method of [ABV92a] introduces the auxiliary operation of *leftmerge* [BK82], with rules

$$\frac{x \xrightarrow{\alpha} x'}{x \mathbin{\|\!\!\lfloor} y \xrightarrow{\alpha} x' \mathbin{|\!|\!|} y} \tag{35}$$

and the equation

$$x \mathbin{|\!|\!|} y = x \mathbin{\|\!\!\lfloor} y + y \mathbin{\|\!\!\lfloor} x \tag{36}$$

Leftmerge is a very tractable operation, being both smooth and distinctive. The method of [ABV92a] gives us essentially the axiom system of [BK82]:

$$\begin{aligned}
(x+y) \mathbin{\|\!\!\lfloor} z &= (x \mathbin{\|\!\!\lfloor} z) + (y \mathbin{\|\!\!\lfloor} z) && (37)\\
(ax) \mathbin{\|\!\!\lfloor} y &= a(x \mathbin{|\!|\!|} y) && (38)\\
\mathbf{0} \mathbin{\|\!\!\lfloor} x &= \mathbf{0} && (39)
\end{aligned}$$

which is complete for strong bisimulation.

7.1 Equations Fail for Unrooted Weak Bisimulations

However, $\mathbin{\|\!\!\lfloor}$ is not WB cool or RB cool, as it doesn't have a patience rule. Not surprisingly, it does not respect either weak or branching bisimulation. Letting $\sim$ stand for either weak or branching bisimulation, we have

$$a \sim \tau a$$

However, $a \mathbin{\|\!\!\lfloor} b \not\sim (\tau a) \mathbin{\|\!\!\lfloor} b$; they're not even weakly trace equivalent:

$$\begin{tikzcd}
a \mathbin{\|\!\!\lfloor} b \arrow[d, "a"] & & & \tau a \mathbin{\|\!\!\lfloor} b \arrow[d, "\tau"] & \\
\mathbf{0}\|b \arrow[d, "b"] & & & a\|b \arrow[dl, "a"'] \arrow[dr, "b"] & \\
\mathbf{0} & & \mathbf{0}\|b \arrow[dr, "b"] & & a\|\mathbf{0} \arrow[dl, "a"'] \\
& & & \mathbf{0} &
\end{tikzcd}$$

Making $\mathbin{\lfloor\!\lfloor}$ patient, by adding the rule

$$\textbf{(mistake)} \qquad \frac{x \xrightarrow{\tau} x'}{x \mathbin{\lfloor\!\lfloor} y \xrightarrow{\tau} x' \mathbin{\lfloor\!\lfloor} y}$$

will, of course, make it respect both weak and branching bisimulation. This doesn't help. While the patient $\mathbin{\lfloor\!\lfloor}$ still enjoys (37), (39), and a suitably modified version of (38), we have lost the all-important (36). Indeed, consider $\tau a \parallel \tau b$ and $(\tau a \mathbin{\lfloor\!\lfloor} \tau b) + (\tau b \mathbin{\lfloor\!\lfloor} \tau a)$. The τ-children of the former are:

$$\begin{tikzcd} & \tau a \parallel \tau b \arrow[dl, "\tau"'] \arrow[dr, "\tau"] & \\ a \parallel \tau b \arrow[dr, "\tau"] & & \tau a \parallel b \arrow[dl, "\tau"'] \\ & a \parallel b & \end{tikzcd}$$

However,

$$\textbf{(mistake)} \qquad (\tau a \mathbin{\lfloor\!\lfloor} \tau b) + (\tau b \mathbin{\lfloor\!\lfloor} \tau a) \xrightarrow{\tau} a \mathbin{\lfloor\!\lfloor} \tau b$$

which is committed to doing the a first; and $\tau a \parallel \tau b$ cannot evolve via τ steps to any state which is committed to doing a first. Thus the two are not even weakly bisimilar.

Conjecture 7.1 *We conjecture that this is not fixable in general; that no non-distinctive operation can be given a finite axiomization, even with auxiliary operators, with respect to weak or branching bisimulation.*

7.2 Equations for Strongly Rooted Equivalences

The situation for strongly rooted weak bisimulation and rooted branching bisimulation is much better: the auxiliary operations created by [ABV92a] can be added as wild operations, and hence those methods give a sound axiomatization.

Suppose that $\mathcal{L}$ is a SRWB cool or RBB cool language. Recall that each rule ρ of $\mathcal{L}$ has the form:

$$\frac{x_i \xrightarrow{\alpha_{ij}} y_{ij}}{f(\vec{x}) \xrightarrow{\beta} t_\rho}$$

where t_ρ is a term mentioning only tame operations.

If $\mathcal{L}'$ is the [ABV92a] extension of $\mathcal{L}$, then all the targets of new rules are (up to renaming of variables) equal to some targets of rules of $\mathcal{L}$. For example, the target of (35) is the same as that of (1). Hence, the targets of $\mathcal{L}'$ are all tame terms. Let us declare that all the new operations are wild. The tame sublanguage of $\mathcal{L}'$ is the same as that of $\mathcal{L}'$, and hence WB cool or RB cool as appropriate. Thus, $\mathcal{L}'$ is SRWB cool or RBB cool.

In particular, the new operations all respect $\underline{\Leftrightarrow}_{sr}$ or $\leftrightarrows_{rb}$. The equations generated by [ABV92a] are valid under strong bisimulation, and remain valid under coarser relations, including $\underline{\Leftrightarrow}_{sr}$ and $\leftrightarrows_{rb}$. Thus, the axiom system is sound.

Completeness is more of a problem, for familiar reasons. We must add axioms describing the equivalence as a quotient of strong bisimulation. This is a (probably easy) open problem for strongly rooted weak bisimulation. For rooted branching bisimulation, this is the equation [BW90]:

$$a(\tau(x+y)+x) = a(x+y).$$

However, for general processes, this will not be complete. [ABV92a] achieve completeness by using the facts that (1) two finitely-branching synchronization trees are strongly bisimilar iff they are bisimilar when truncated to all finite depths, and (2) GSOS languages only generate finitely-branching trees. Any decent process algebra can define a process p which is infinitely branching up to $\xrightarrow{\tau^*}$: that is, there are an infinite number of distinct processes p_n with $p \xrightarrow{\tau^*} p_n$. The best approach we are aware of is the use of auxiliary predicates to test for boundedness, and take a bounded version of the Approximation Induction Principle; for details, see [BW90].

7.3 Equations for Rooted Weak Bisimulation

As always, we need to have equations characterizing rooted weak bisimulation as a quotient of strong bisimulation. For rooted weak bisimulation, these are the familiar

$$\begin{aligned} \alpha.\tau.x &= \alpha x \\ \alpha(\tau(y+z)+y) &= \alpha(y+z) \end{aligned}$$

We need to extend [ABV92a] slightly; operations in RWB cool languages are frequently non-distinctive. The intuition behind distinctive operations is that at most one rule can apply to any term of the form $f(a_1p_1,\ldots,a_np_n)$. However, this is frequently not true in RWB cool languages. For example, in Definition 4.6, if t^2 and t are different, the rules (21) and (22) give t^1 two transitions with the same cause. However, the full force of distinctiveness isn't required for [ABV92a], nor is it here.

Definition 7.2 *Let f be a straight operation in a positive* GSOS(ε) *language, and ρ a rule for f. The* trigger *of f by ρ is the vector $\langle\gamma_1,\ldots,\gamma_n\rangle$, where γ_i is the action on the antecedent $x_i \xrightarrow{\gamma_i} y_i$.*

The trigger of a rule describes when it can fire. We are interested in when several rules can apply to a term. Clearly, two rules with the same trigger fire or don't fire together. However, rules with different triggers can also apply; *e.g.*, both rules of (1) both apply to $\alpha\|\alpha$, with triggers $\langle\alpha,\varepsilon\rangle$ and $\langle\varepsilon,\alpha\rangle$. The two rules *overlap*:

Definition 7.3 *Two triggers $\vec{\gamma}$ and $\vec{\delta}$* overlap *if, for all i, if $\gamma_i \neq \delta_i$, then either γ_i or δ_i is ε. Two rules overlap if their triggers do.*

It is impossible in general to axiomatize operations with overlapping rules directly; *e.g.*, there is no finite axiomatization of $\|\!|$ [Mol90]. The definition of distinctiveness in [ABV92a] simply forbade overlapping triggers. For our setting, we need a looser notion:

Definition 7.4 *A straight positive* GSOS(ε) *operation f is* instinctive *if,*

1. *whenever two rules have overlapping triggers, the triggers are equal; and*
2. *the rules for f use the same target variables; that is, if $x \xrightarrow{\alpha} y$ is an antecedent of one rule for ρ, then $x \xrightarrow{\alpha'_\rho} y$ is an antecedent of every rule ρ for some α'_ρ; and*
3. *For any argument x_i, either all rules have a non-ε antecedent for x_i or none do.*

A set of rules is instinctive if the operation defined by only those rules is instinctive. (Thus, we may speak of a subset of an operation's rules being instinctive.)

Lemma 7.5 *Let f be an instinctive operation, and let $\vec{\gamma}$ be a trigger of the right length for f. Let R be the set of all rules for f with $\vec{\gamma}$ as trigger. Let α^ρ be the action and t^ρ be the target of rule ρ. Let $X_i = \begin{cases} \gamma_i . y_i & \gamma_i \neq \varepsilon \\ y_i & \gamma_i = \varepsilon \end{cases}$. Then $\models f(\vec{X}) = \sum_{\rho \in R} \alpha^\rho . t^\rho$.*

Proof: Each of the rules in R applies to $f(\vec{X})$; each $\rho \in R$ causes an α^ρ transition to t^ρ. As the triggers of rules of f which are not in R do not overlap γ, no such rule applies to $f(\vec{X})$. ♀

Note that if there are no rules with $\vec{\gamma}$ as their trigger, this gives an equation of the form $f(\vec{X}) = \mathbf{0}$. The language is positive, so the tricks of [ABV92a] for negative rules are not necessary.

Lemma 7.6 *Let f be an instinctive operation. Then f is distributive over its active arguments. Specifically, suppose that x_i is active in $f(\vec{x})$. For convenience, let $C[v] = (f(\vec{x}))[x_i := v]$. Then*

$$\models C[x+y] \leftrightarrows C[x] + C[y].$$

Proof: This is a consequence of [ABV92b]'s Lemma 4.3, which essentially says that the third part of the definition of "instinctive" guarantees distributivity. ♀

Definition 7.7 *A RWB cool language is* classy *if, for each rule ρ for a wild operation, the rule ρ together with all instances of (18) and (19) for it are instinctive.*

An equational axiom system E is said to be *head-normalizing* if, whenever p is a process, there is a theorem of the form $E \vdash p = \sum_i a_i q_i$. Head normalization is the essential property of [ABV92a]-like theories.

Theorem 7.8 *Let $\mathcal{L}$ be a classy RWB cool language. Then there is a RWB cool extension $\mathcal{L}'$ of $\mathcal{L}$ with a sound and head-normalizing axiom system for* strong *bisimulation.*

Proof: The method of [ABV92a] proceeds in steps. The first step introduces straight versions of all non-straight operations in $\mathcal{L}$. If f is not straight, its straightened version f^+ has the same rules as $f^\star$; the main difference is that [ABV92a] does not introduce bifurcation rules taking f to f^+.

Thus, if f is non-straight but tame, then $\mathcal{L}$ already has a straightened version of $f(\vec{x})$. Indeed, the equation

$$f(\vec{x}) \leftrightarrow \tfrac{\mathsf{X}}{\mathsf{X}'}(f(\vec{x})^\star)$$

is valid, as the rules for the two sides are in such close correspondence. So, we need not introduce f^+ in this case.

If f is non-straight and wild, then we must introduce f^+. The targets of the rules for f^+ are the same as the targets of the rules for f:

$$\frac{x \xrightarrow{a} y_1, \quad x \xrightarrow{b} y_2}{f(x) \xrightarrow{c} g(y_1, y_2)} \quad \mapsto \quad \frac{x_1 \xrightarrow{a} y_1, \quad x_2 \xrightarrow{b} y_2}{f^+(x_1, x_2) \xrightarrow{c} g(y_1, y_2)}$$

In particular, f has rules (18) and (19), giving f τ-transitions to t^1 and t^2. f^+ has corresponding rules, giving f^+ transitions to t^1 and t^2. Of course, t^1 and t^2 are the same terms for f^+ as they were for f, so the remaining rules (which govern t^1 and t^2 transitions and do not mention f) show that f^+ has the required auxiliary ruloids. Hence, by declaring the new f^+ operations to be wild, the $\mathcal{L}$ extended by the straightening auxiliary operations is RWB cool.

The remaining steps of the [ABV92a] method are straightforward given our assumptions. Straight, instinctive operations can be axiomatized directly as described above. Suppose that f is straight and not instinctive. We will introduce some auxiliary, wild, instinctive operations which jointly describe f. For each rule ρ for f, we define the operation f_ρ, which just has rules corresponding to ρ, and the rules (18) and (19): the rules for f_ρ are identical to those for f, except that f_ρ replaces f in the sources. The condition of classiness ensures that f_ρ is instinctive, and hence can be axiomatized. Using the f_ρ, we can axiomatize f:

$$f(\vec{x}) \leftrightarrow \sum_{\rho} f_\rho(\vec{x})$$

♀

8 Conclusion

We have given fairly general rule formats for four popular notions of silent bisimulation, and one new formal as well. In many cases, the rule formats for the rooted equivalences admit the extension giving equational axiom systems of [ABV92a]. These axiom systems are as close to complete as can be expected, given the degree of undecidability of the silent bisimulations.

8.1 Open Problems

A number of questions remain to be answered.

1. Conjecture 7.1, that nondistinctive operations cannot be finitely axiomatized with respect to weak or branching bisimulation.

2. Rooted weak bisimulation might able tolerate negative rules to some (probably very small) extent, under some interpretation. Can this actually be done? Is the resulting extension useful?

3. Almost all of the rules used in this study have been positive GSOS. It seems likely that similar constructions would work for the tyft rules of [GV92].

4. What is the theory and practice of strongly rooted weak bisimulation? Is it actually good for anything, or does it just have a clean rule format?

5. Are the conditions given in this paper sufficient? That is, are there GSOS operations which do respect, say, weak bisimulation, but are not WB cool? We exclude *junk* operations; for example, if f has only one rule

$$\frac{x \xrightarrow{a} y, x \stackrel{a}{\not\longrightarrow}}{f(x) \xrightarrow{a} \mathbf{0}}$$

then f does respect weak bisimulation — indeed, f is semantically constant, with $f(x) \leftrightarrow \mathbf{0}$ — but this is uninteresting.

6. As with GSOS languages, we may define the congruence generated by a rule format: two processes are, say, RB cool congruent iff they have the same set of completed weak traces (*viz.*, without τ) in all RB cool languages. (Actually, there are a number of choices to be made here: *e.g.*, one may use weak partial traces instead of weak completed ones, or take divergence into account.) Do the congruences generated by these rule formats have any good characterization or properties?

9 Acknowledgements

Thanks are due to:

- Frits Vaandrager and Rob van Glabbeek for technical and historical advice.
- Vicki Borah, F. Marten, and the summer heat for terminological assistance.
- Paul Taylor for his diagrams package, which let me produce the trickiest diagrams in this document in half of no time.

References

[Abr87] Samson Abramsky. Observation equivalence as a testing equivalence. *Theoretical Computer Sci.*, 53(2/3):225–241, 1987.

[ABV92a] Luca Aceto, Bard Bloom, and Frits Vaandrager. Turning SOS rules into equations. In *Proceedings of LICS '92*, pages 113–124. IEEE Computer Society Press, 1992.

[ABV92b] Luca Aceto, Bard Bloom, and Frits Vaandrager. Turning SOS rules into equations. Technical Report CS-R9218, CWI, 1992.

[BC90] Kenneth Birman and Robert Cooper. The Isis project: Real experience with a fault tolerant programming system. Technical Report 90-1138, Department of Computer Science, Cornell University, July 1990.

[BCG91] Kenneth P. Birman, Robert Cooper, and Barry Gleeson. Programming with process groups: Group and multicast semantics. Technical Report 91-1185, Department of Computer Science, Cornell University, January 1991.

[BHR84] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984.

[BIM88] Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced (preliminary report). In *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages*, pages 229–239, 1988. Also appears as MIT Technical Memo MIT/LCS/TM-345.

[BIM90] Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. Technical Report TR 90-1150, Cornell, August 1990. (To appear in JACM).

[BK82] J. A. Bergstra and J. W. Klop. Fixed point semantics in process algebras. Technical Report IW 206/82, Department of Computer Science, Mathematisch Centrum, Amsterdam, The Netherlands, 1982.

[Blo89] Bard Bloom. *Ready Simulation, Bisimulation, and the Semantics of CCS-Like Languages.* PhD thesis, Massachusetts Institute of Technology, August 1989.

[Blo90] Bard Bloom. Strong process equivalence in the presence of hidden moves (preliminary report). (Unpublished), July 1990.

[Blo93] Bard Bloom. Ready, set, go: Structural operational semantics for linear-time process algebras. Technical Report 00000, Cornell, 1993. (to appear).

[BM90] Bard Bloom and Albert R. Meyer. Experimenting with process equivalence. In Dr. Marta Kwiatkowska, editor, *Proceedings of the International BCS-FACS Workshop on Semantics for Concurrency*, pages 189–201, Leicester, U.K., July 1990.

[BW90] J. C. M. Baeten and W. P. Weijland. *Process Algebra.* Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990.

[CPS89] Rance Cleaveland, Joachim Parrow, and B. Steffen. The Concurrency Workbench. In *Proceedings of the Workshop on Automated Verification Methods for Finite State Systems.* Springer-Verlag, 1989. LNCS 407; to appear in ACM TOPLAS.

[Gla90] R.J. van Glabbeek. The linear time – branching time spectrum. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings CONCUR 90,* Amsterdam, volume 458 of *Lect. Notes in Computer Sci.*, pages 278–297. Springer-Verlag, 1990.

[Gro90] Jan Friso Groote. Transition system specifications with negative premises. Technical Report CWI report R-CS8950, CWI, 1990.

[GV92] Jan Friso Groote and Frits Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, October 1992.

[Hen88] Matthew Hennessy. *Algebraic Theory of Processes.* MIT Press Series in the Foundations of Computing. MIT Press, Cambridge, Massachusetts, 1988.

[HM85] Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137-161, 1985.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lect. Notes in Computer Sci.* Springer-Verlag, 1980.

[Mil81] Robin Milner. A modal characterisation of observable machine-behaviour. In E. Astesiano and C. Böhm, editors, *CAAP '81: Trees in Algebra and Programming, 6th Colloquium*, volume 112 of *Lect. Notes in Computer Sci.*, pages 25–34. Springer-Verlag, 1981.

[Mil83] Robin Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Sci.*, 25(3):267–310, 1983.

[Mil84] Robin Milner. Lectures on a calculus for communicating systems. In S. D. Brookes, A. W. Roscoe, and G. Winskel, editors, *Seminar on Concurrency: CMU, July 9-11, 1984*, volume 197 of *Lect. Notes in Computer Sci.*, pages 197–220. Springer-Verlag, 1984.

[Mil89a] Robin Milner. *Communication and Concurrency.* Prentice Hall International Series in Computer Science. Prentice Hall, New York, 1989.

[Mil89b] Robin Milner. A complete axiomatisation for observational congruence of finite-state behaviours. *Information and Computation*, 81(2):227–247, May 1989.

[Mol90] Faron Moller. The importance of the left merge operator in process algebras. In M. Paterson, editor, *Automata, Languages and Programming: 17^{th} International Colloqium*, volume 443 of *Lect. Notes in Computer Sci.*, pages 752–764. Springer-Verlag, July 1990.

[Par81] D. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Theoretical Computer Science*, Lect. Notes in Computer Sci., page 261. Springer-Verlag, 1981.

[Plo81] Gordon Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, Computer Science Department, Denmark, 1981.

[Uli92] Irek Ulidowski. Equivalences on observable processes. In *Proceedings of the Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 148–161. IEEE, 1992.

[Vaa91] Frits Vaandrager. private communication. (email discussions), June 1991.

[vG93] Rob van Glabbeek. The linear time - branching time spectrum ii: The semantics of sequential processes with silent moves (preliminary version). Available by anonymous ftp from `Boole.stanford.edu`, April 1993.

[vGW89] Rob van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics. In G. X. Ritter, editor, *Information Processing 89: Proceedings of the IFIP 11th World Computer Congress*, pages 613–618. North-Holland, August 1989. TUM report I9052 (= SFB-Bericht Nr. 342/29/90 A) and CWI report CS-R9120.

[WBB92] Sam Weber, Bard Bloom, and Geoffrey Brown. Compiling Joy to silicon: A verified silicon compilation scheme. In Tom Knight and John Savage, editors, *Proceedings of the Advanced Research in VLSI and Parallel Systems Conference*, pages 79–98. 1992.