Partial Traces and the Semantics and Logic of CCS-Like Languages

Bard Bloom*

TR 89-1066
December 1989

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

*This research was largely done at MIT under NSF Grant No. 8511190-DCR and ONR Grant No. N00014-83-K-0125.

Partial Traces and the Semantics and Logic of CCS-Like Languages

Bard Bloom*
Cornell University

December 12, 1989

Abstract

We consider the theory of CCS-like languages when partial traces (simply finite sequences of actions that the process may perform) are the only observation. We characterize process equivalence, giving relational, logical, and operational definitions and showing that they coincide. This relation is adequate for all languages defined by a class of CCS-like rules; it is fully abstract for any language including process copying and controlled communication poerations. We also give a complete inequational axiom system for this notion of process equivalence for finite processes.

1 Introduction

Two similar and widely-used theories of concurrency are Milner's CCS [Mil83] and Hoare's CSP [Hoa78]. One of the most basic things that a theory of parallel processes should do is define a notion of *process equivalence*. The most straightforward definition, usable in most parts of computer science, is *interchangeability* or *congruence*: P and Q are equivalent in this sense iff either one can be substituted for the other in any context, and no difference can be observed. This definition depends on two parameters: the programming language over which the set of contexts is taken, and the set of observations which can be made about processes.

The core theories of CCS and CSP have fairly similar notions of what a process may be. In both, we have a set Act of *actions*; a process is a device which changes state nondeterminstically; on each step, it performs some action. In this study, we

*This research was largely done at MIT under NSF Grant No. 8511190–DCR and ONR grant No. N00014–83–K–0125.

consider only the so-called synchronous case; all actions that a process performs are visible.

There are a number of things which one could consider observing about such a process. The simplest useful one, as observed by CSP, is a *finite partial trace*: simply a sequence of actions which the process can perform. Two processes are considered equivalent in CSP iff they have the same partial traces in all CSP contexts. A more powerful notion is a *finite completed trace*, a finite sequence of actions which the process performs before halting; finite completed traces add the ability to detect termination. More powerful observations, such as infinite traces, are mathematically possible; we do not consider them here.

In general, most mathematical notions of semantics of processes will include a notion of process equivalence, telling when two processes are the same. We say that the semantics is *adequate* (with respect to a language and notion of observation) if semantic equivalence implies indistinguishability; it is *fully abstract* if the two are equivalent. Adequacy is generally considered essential for a semantics to be useful; full abstraction is pleasant but not as vital.

In previous papers [BIM88,BM89], we have discussed *bisimulation*, the CCS notion of process equivalence introduced in [Mil80,Mil81]. We have argued that it is too fine, and that it cannot be expressed as interchangeability with respect to any sufficiently CCS-like language for any known reasonable notion of observation. We defined the class of GSOS operations, and argued that it was a maximal extension of the form of rules used in CCS which preserved the basic properties of CCS; this is our definition of CCS-like.

We introduced the notion of *ready simulation*, a weakened version of bisimulation, and showed that (1) ready simulation is an adequate semantics for every GSOS language, and (2) ready simulation is fully abstract for a language including process copying and controlled communication operators. Both of these results used completed traces.

In this paper, we consider the case of partial traces. Partial traces can detect less about a process than completed traces, and not surprisingly completed-trace equivalence implies partial-trace equivalence in the setting of CCS and CSP. GSOS languages are quite powerful, and it is straightforward to define a true sequencing operation: the process $(P;Q)$ runs P until it halts, then runs Q.[1] Using the sequencing operation, we can detect termination, and thus conclude that ready simulation matches partial trace equivalence for GSOS languages in the same way it does completed trace

[1]CSP has a variant of this, which runs P until it *signals* that it has halted; this is weaker, because a process may deadlock and be unable to signal. CCS uses a restricted form of sequencing, only allowing P to be a single action.

equivalence.

However, GSOS languages allow the questionable feature of *negative rules*. If α is an operation on processes, $\alpha(P)$ may branch on the inability of P to perform some particular action. CCS and CSP do not use negative rules. It may be difficult to implement operations defined in terms of negative rules. Finally, we have decided that we should not observe partial traces; it is somewhat strange to allow operations in the language to observe them.

So, we consider the case of *positive GSOS languages*, *viz.*, those without negative rules. Ready simulation turns out to be too fine, and cannot be fully abstract with respect to any positive GSOS language with respect to observing partial traces.[2] We show that a fairly common notion of process equivalence, which we call *mutual monosimulation*, is adequate with respect to partial traces and any positive GSOS language, and fully abstract with respect to any positive GSOS language containing the process copying and controlled communication operations above.

Both bisimulation and ready simulation can be characterized by equivalence with respect to classes of modal formulas; bisimulation by the *Hennessy-Milner formulas*, and ready simulation by the *denial formulas*. Mutual monosimulation has a similar characterization in terms of $\langle\rangle$ -formulas. We use this characterization to obtain the full abstraction result. So, we have four equivalent characterizations of partial-trace congruence, in three quite different domains:

1. Partial-trace congruence with respect to all positive GSOS languages.

2. Partial-trace congruence with respect to a particular, fairly reasonable language.

3. Mutual monosimulation, a bisimulation-like relation.

4. $\langle\rangle$ -formula equivalence, a modal logic.

Finally, we give an inequational logic for mutual monosimulation. This logic is sound for all processes, and complete for finite processes.

The relation of monosimulation has appeared in several settings before, although the logical, observational, and equational characterization presented in this paper is (as far as we know) original. In the setting of finite automata [HS66], it appears mutatis mutandis as homomorphism of automata.

More recently, Lynch and Tuttle [LT88] have introduced *possibilities maps*, and Abadi and Lamport [AL88] have introduced *refinement maps*. All of these map or relate states in a program to states in its specification, such that transitions in the

[2] It is worth noting that ready simulation is fully abstract with respect to positive GSOS languages observing completed traces.

process correspond to transitions in the specification; they are effectively the concept of monosimulation as expressed in the researchers' models. If the experience of the verification community is any guide, monosimulation is a powerful proof method for processes. The languages used in verification are rather sparse (Lynch and Tuttle have two operations, Abadi and Lamport have none), and so there is no a priori reason to suspect that proofs carried out in one of these models might apply to the far richer settings such as CCS.

We suspect that the results of this paper can be adapted to fit the Lynch-Tuttle and Abadi-Lamport settings, and thus imply that programs proved correct by possibilities or refinement mappings are in fact correct in CCS, or indeed in any suitably extension of CCS.

2 CCS and GSOS languages

We briefly describe the core language of CCS; it is described more fully in [Mil80, Mil84]. We are working in the setting of *strong equivalence*; all actions taken by processes are visible. We fix a set Act of actions, which are uninterpreted; a, b, and c range over actions. CCS processes are defined by the grammar

$$P ::== \mathbf{0} \mid P + P \mid aP \mid P|P \mid P\backslash a \mid P[\varphi]$$

where $\varphi : \mathsf{Act} \to \mathsf{Act}$. These operations are given behavior by *structured operational rules*. The basic judgement is $P \xrightarrow{a} Q$, that the process P has the ability to perform the action a and thereafter behave like Q. $\mathbf{0}$ is the stopped process; aP is a process which performs the action a and then behaves like P; $P + Q$ nondeterministically chooses whether to behave like P or like Q; $P|Q$ runs P and Q in a parallel, with the possibility of communication. The rules include the following.

$$aP \xrightarrow{a} P \quad \frac{P \xrightarrow{a} P'}{P + Q \xrightarrow{a} P'} \frac{Q \xrightarrow{a} Q'}{P + Q \xrightarrow{a} Q'} \frac{P \xrightarrow{a} P'}{P|Q \xrightarrow{a} P'|Q} \frac{Q \xrightarrow{a} Q'}{P|Q \xrightarrow{a} P|Q'}$$

In [BIM88], we defined the *GSOS rules*, a class of rules extending the form of the CCS rules; we briefly remind the reader of this definition. We exclude guarded recursion from GSOS systems for the purposes of this paper; it can be included without modification of the theorems. Pick a finite set of operators with fixed arities; processes are terms generated by those operators.

Definition 2.1 *A* GSOS rule ρ *is a rule of the form:*

$$\frac{\left\{X_i \xrightarrow{a_{ij}} Y_{ij} \mid 1 \leq j \leq m_i\right\}_{i=1}^{l}, \quad \left\{X_i \stackrel{b_{ik}}{\not\rightarrow} \mid 1 \leq k \leq n_i\right\}_{i=1}^{l}}{\mathrm{op}(X_1, \ldots, X_l) \xrightarrow{c} C[\vec{X}, \vec{Y}]}$$

where all the variables are distinct, $l \geq 0$ is the arity of op*, $m_i \geq 0$, and $C[\vec{X}, \vec{Y}]$ is a context with free variables including at most the X's and Y's. (It need not contain all these variables.)*

The behaviors of processes are given in the natural way; $P \xrightarrow{a} Q$ in a GSOS language $\mathcal{L}$ iff there is a proof of $P \xrightarrow{a} Q$ from the rules for $\mathcal{L}$; otherwise $P \stackrel{a}{\not\rightarrow}$. For example, in CCS, we can show that $a(b\mathbf{0}) + a(c\mathbf{0}) \xrightarrow{a} b\mathbf{0}$. A typical non-CCS operator is $\alpha(P)$, defined by

$$\frac{X \stackrel{a}{\not\rightarrow}}{\alpha(X) \xrightarrow{a} X}$$

For example, we have $\alpha(\mathbf{0}) \xrightarrow{a} \mathbf{0}$, but $\alpha(a\mathbf{0})$ is unable to move.

The technical decisions in the definition of GSOS rules are motivated in [BIM88, Blo89]. Briefly, the GSOS rules are a maximal class of rules which guarantee the basic properties of CCS. For example, any language defined by GSOS rules has a unique, finitely-branching, computable transition relation, and the synchronization tree semantics below are adequate. Furthermore, any slight extension to the language seems to break one or more of these properties.

We extend this notation to $P \xrightarrow{s} Q$, where s is any string of actions; $P \xrightarrow{s} Q$ iff there are $P_1, \ldots, P_n$ such that $P \xrightarrow{s_1} P_1 \xrightarrow{s_2} \cdots \xrightarrow{s_n} P_n = Q$, where s_i is the i^{th} character of s.

The behavior of a process may be unwound indefinitely, giving a (generally infinitely deep, but finite-branching) tree edge-labeled by actions and node-labeled by process terms, which completely describes the behavior of the process. Abstracting slightly, we arrive at the notion of a *synchronization tree* [Kel76,Mil81], an unordered tree edge-labeled by actions with unlabeled nodes. Synchronization trees are an adequate semantics for CCS, and indeed for all GSOS languages.

(For technical reasons, we include synchronization trees as constants in GSOS languages. These constants have the same meaning in all languages. Our results compare synchronization trees, not process terms; as the meanings of operations in process terms depend on the language, it makes little sense to consider the problem of when two terms are equivalent in all languages.)

However, synchronization trees are generally agreed to give a too fine a notion of process equivalence. For example, the synchronization tree of $a + a$, with two a-edges leading to leaves, is different from that of a; however, $a + a$ and a cannot be distinguished. Intuitively, there is no difference between having one way of doing a and then stopping, and having two ways. Milner [Mil83] introduces *bisimulation*, an equivalence relation on synchronization trees, as a more useful notion of process equivalence. Bisimulation has many virtues which we will not consider here [Mil88].

In [Blo89,BIM88] we show that, for completed traces, bisimulation is adequate but not fully abstract for any GSOS language. We also introduced the notion of *ready simulation.*

Definition 2.2 *A relation* $\sqsubseteq$ *between processes is a* ready simulation relation *if, whenever* $P \sqsubseteq Q$*, then:*

1. *If* $P \xrightarrow{a} P'$*, then there is a process* Q' *such that* $Q \xrightarrow{a} Q'$ *and* $P' \sqsubseteq Q'$.
2. *If* $P \not\xrightarrow{a}$*, then* $Q \not\xrightarrow{a}$.

P is a ready simulation approximation of Q, $P \underset{\rightarrow}{\sqsubseteq} Q$*, if there is such a* $\sqsubseteq$*. P is ready similar to Q,* $P \rightleftharpoons Q$*, if* $P \underset{\rightarrow}{\sqsubseteq} Q \underset{\rightarrow}{\sqsubseteq} P$.

In [BIM88] we showed that ready simulation was precisely interchangeability with respect to completed traces in all GSOS languages.

3 Partial Traces

As we have said, we are interested in the case of partial traces. Rather than observing the process until it stops, we merely observe it for some finite time.

Definition 3.1 *The* partial trace set $\mathrm{ptr}(P)$ *of a process P is* $\left\{s : \exists Q.\, P \xrightarrow{s} Q\right\}$. *Processes P and Q are* partial trace equivalent, $P \equiv_{\mathrm{ptr}} Q$, *iff* $\mathrm{ptr}(P) = \mathrm{ptr}(Q)$. *They are* partial trace congruent *with respect to a class of languages* $\mathcal{L}$*, written* $P \equiv_{\mathrm{ptr}}{}^{\mathcal{L}} Q$*, iff, for each* $L \in \mathcal{L}$ *and each L-context* $C[X]$*, we have* $C[P] \equiv_{\mathrm{ptr}} C[Q]$.

Theorem 3.2 *Partial trace GSOS congruence coincides with ready simulation.*

Proof: It is routine by the methods of [BIM88,Blo89] to show that ready simulation implies partial trace GSOS congruence. The converse uses the fact that sequencing, $P;Q$, is definable in GSOS languages. Briefly, we use sequencing to attach end-markers to traces, and thus we can tell what the finished traces of a proces are by looking at those partial traces which contain end-markers.

$$\frac{P \xrightarrow{a} P'}{P\,;Q \xrightarrow{a} P'\,;Q} \qquad \frac{P \not\xrightarrow{a} \text{ for all } a \in \mathsf{Act}, \quad Q \xrightarrow{b} Q'}{P\,;Q \xrightarrow{b} Q'}$$

$\square$

4 Positive GSOS languages

Clearly, the negative rules in the language give contexts the ability to detect termination. If we take the position that termination should be undetectable inside the language as well as outside — or, equally plausibly, we are not willing to implement the potentially hard-to-implement operations which rely on negative information — we should not allow negative rules. From now on, we consider positive GSOS languages.

Definition 4.1 *A* positive GSOS rule ρ *is a rule of the form:*

$$\frac{\left\{X_i \xrightarrow{a_{ij}} Y_{ij} \mid 1 \leq j \leq m_i\right\}_{i=1}^{l}}{\mathrm{op}(X_1, \ldots, X_l) \xrightarrow{c} C[\vec{X}, \vec{Y}]}$$

where all the variables are distinct, $l \geq 0$ is the arity of op, *$m_i \geq 0$, and $C[\vec{X}, \vec{Y}]$ is a context with free variables including at most the X's and Y's. (It need not contain all these variables.)*

Most rules proposed in the literature [dS85,Mil80] are positive GSOS rules. In fact, positive GSOS rules include one feature frequently excluded from languages: the ability to dynamically make copies of processes. For example, the operator β defined by:

$$\frac{X \xrightarrow{a} X'}{\beta(X) \xrightarrow{a} X \parallel X'}$$

cannot be defined in CCS.

We are interested in seeing which processes can be distinguished by partial traces in positive GSOS languages; *viz.*, which processes are partial-trace positive GSOS language congruent. We present three equivalent characterizations of this relation: relational, logical, and as congruence with respect to a particular language.

5 Mutual Monosimulation

Definition 5.1 *A relation $\leq_0$ between synchronization trees is a* monosimulation relation *iff, whenever*

$$P \leq_0 Q \quad \textit{and} \quad P \xrightarrow{a} P'$$

then

$$\exists Q'. Q \xrightarrow{a} Q' \textit{ and } P' \leq_0 Q'.$$

We say that P is monosimulated by Q, $P \leq Q$, if there is some monosimulation relation $\leq_0$ such that $P \leq_0 Q$. P and Q are mutually monosimilar *(or* monosimulate each other*), $P \lesseqgtr Q$ if $P \leq Q \leq P$.*

For example, it is easy to see that $P \lesseqgtr P$, and that $\mathbf{0} \leq P$ for every P; if $P \neq \mathbf{0}$ then $P \not\leq \mathbf{0}$. It is worth remarking that mutual monosimulation is strictly weaker than either ready simulation or bisimulation. The processes aa and $aa + a$ monosimulate each other, but are neither ready similar nor bisimilar (nor CCS completed-trace congruent, for that matter); see Figure 1. This fact may be undesirable, as it says that a process which may die partway through its computation (or may succeed) is equivalent to one which will always succeed. To formalize this undesirability, one must include termination as a possible observation, and one returns to the setting of ready simulation of [BIM88].

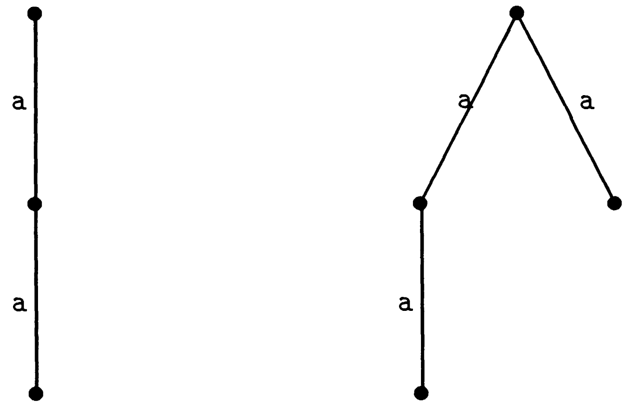

Figure 1: Mutually Similar but not Ready Similar, Bisimilar, or CCS Completed-Trace Congruent

Theorem 5.2 *If $P \lesseqgtr Q$ then P and Q are partial-trace congruent with respect to all positive GSOS languages.*

Proof: (Sketch) Show that all contexts of any number of arguments can be defined by positive GSOS like rules (mutatis mutandis), and using that fact show that the relation $\leq_1$, defined by $C[\vec{R}] \leq_1 C[\vec{S}]$ whenever each $R_i \leq S_i$, is a monosimulation relation. We then have $P \lesseqgtr Q$ implies $C[P] \lesseqgtr C[Q]$ for all positive GSOS contexts

$C[X]$. It is straightforward to show that $P \lessgtr Q$ implies $\mathrm{ptr}(P) = \mathrm{ptr}(Q)$; hence P and Q are partial trace congruent as desired. □

6 Logical Characterization

Recall that bisimulation and ready simulation each correspond to a suitable modal logic. Bisimulation matches Hennessy-Milner logic [HM80], with formulas defined by

$$\varphi ::== \mathrm{tt} \mid \mathrm{ff} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle a \rangle \varphi \mid [a]\varphi$$

and ready simulation matches denial logic [LS88,Blo89], given by

$$\varphi ::== \mathrm{tt} \mid \mathrm{ff} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle a \rangle \varphi \mid \mathrm{Can't}(a)$$

where a ranges over Act.

Definition 6.1 *$P \models \varphi$ iff*

- *$P \models \mathrm{tt}$ always; $P \models \mathrm{ff}$ never.*
- *$P \models \varphi \wedge \varphi'$ iff $P \models \varphi$ and $P \models \varphi'$;*
- *$P \models \varphi \vee \varphi'$ iff $P \models \varphi$ or $P \models \varphi'$;*
- *$P \models \langle a \rangle \varphi$ iff for some P', $P \xrightarrow{a} P'$ and $P' \models \varphi$.*
- *$P \models [a]\varphi$ iff for every P' such that $P \xrightarrow{a} P'$, $P' \models \varphi$.*
- *$P \models \mathrm{Can't}(a)$ iff $P \stackrel{a}{\not\rightarrow}$.*

Note that $P \models \mathrm{Can't}(a)$ iff $P \models [a]\mathrm{ff}$.

Fact 6.2 *([HM80,Blo89,LS88])*

1. *P and Q are bisimilar iff for all Hennessy-Milner formulas φ, $P \models \varphi$ iff $Q \models \varphi$.*
2. *P and Q are ready similar iff for all denial formulas φ, $P \models \varphi$ iff $Q \models \varphi$.*

A similar fact holds for mutual monosimulation. The appropriate class of formulas is the *$\langle\rangle$-formulas*:

$$\varphi ::== \mathrm{tt} \mid \mathrm{ff} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle a \rangle \varphi$$

Recall from modal logic that $\langle a \rangle(\varphi \vee \psi)$ and $(\langle a \rangle \varphi) \vee (\langle a \rangle \psi)$ are equivalent. Disjunction does not give any extra distinguishing power to $\langle\rangle$ -formulas: if P and Q disagree on some $\langle\rangle$-formula, then they disagree on a $\langle\rangle$-formula not involving $\vee$: a *conjunctive* $\langle\rangle$-formula. We use this fact later.

Theorem 6.3 *Processes P and Q are mutually similar iff for all $\langle\rangle$-formulas φ, $P \models \varphi$ iff $Q \models \varphi$.*

Proof: ($\Rightarrow$) Show that if $P \leq Q$, then $P \models \varphi$ implies $Q \models \varphi$ by induction on φ simultaneously for each P and Q. For example, suppose that it holds for ψ; we will show it for $\langle a\rangle\psi$. Suppose that $P \models \langle a\rangle\psi$. Then there is some P' such that $P \xrightarrow{a} P'$ and $P' \models \psi$. By definition of monosimulation, there is some Q' such that $Q \xrightarrow{a} Q'$ and $P' \leq Q'$; by induction, $Q' \models \psi$ and thus $Q \models \langle a\rangle\psi$.

($\Leftarrow$) Show that the relation $P \leq_2 Q$ is a monosimulation relation, where $P \leq_2 Q$ iff for each $\langle\rangle$-formula φ, $P \models \varphi$ implies $Q \models \varphi$. Suppose that $P \leq_2 Q$ and $P \xrightarrow{a} P'$, but that for no Q' such that $Q \xrightarrow{a} Q'$ and $P' \leq_2 Q'$. Let $Q'_1, \ldots, Q'_n$ be the list of a-children of Q; this list is finite because Q is finitely branching.[3] Then for each Q'_i, there is some formula φ_i such that $P' \models \varphi_i$ but $Q'_i \not\models \varphi_i$. Let φ be the conjunction of the φ_i, or tt if $n = 0$. Clearly $P' \models \varphi$, but no $Q'_i \models \varphi$. Thus $P \models \langle a\rangle\varphi$ but $Q \not\models \langle a\rangle\varphi$, contradicting the assumption that $P \leq_2 Q$. □

7 Mutual Monosimulation Can Be Traced

In this section, we present a language CCSSS capturing mutual monosimulation: mutual monosimulation is precisely CCSSS partial-trace congruence. (Incidentally, CCSSS also captures ready simulation as completed trace congruence.) We add two operations (plus a third auxiliary operation). $\langle\!\langle\rangle\!\rangle P$ is a copying operator: when P signals that it wants to fork, $\langle\!\langle\rangle\!\rangle P$ forks. $S \triangleright P$ is a sort of controlled communication: S runs alone, except that it occasionally allows P the ability to take a step and communicate with it.

Using these operations, we will code $\langle\rangle$-formulas into contexts and traces, and so understand monosimulation in CCSSS. $C_\varphi[P]$ tests the process P to see if it satisfies φ, producing a characteristic kind of partial trace, a *φ-happy trace*, if it does and not if it does not. So, if $P \not\gtrless Q$, then there is some φ for which (say) $P \models \varphi$ but $Q \not\models \varphi$. Then $C_\varphi[P]$ has a φ-happy trace, but $C_\varphi[Q]$ does not; thus, $\mathrm{ptr}(C_\varphi[P]) \neq \mathrm{ptr}(C_\varphi[Q])$.

We fix several distinct actions. We will use o as a sort of "visible silent action;" processes will emit o's while they are operating, unless they have something useful to emit. The actions c_1 and c_2 are used by processes to signal to the $\langle\!\langle\rangle\!\rangle$ operator that they wish to fork. In $S \triangleright P$, S uses the d action to signal that it wishes to communicate with P. There is an auxiliary operator $\blacktriangleright$ used by $\triangleright$.

[3] A similar theorem holds for arbitrarily branching processes and a language with infinitary conjunctions and disjunctions.

$\langle\!\langle\rangle\!\rangle(P)$ usually does just what P does. However, when P signals that it wants to be forked (by the c_1 and c_2 actions), $\langle\!\langle\rangle\!\rangle(P)$ forks it. $P \parallel Q$ is simple interleaving parallelism without communication.

$$\frac{X \xrightarrow{a} X' \quad (a \notin \{c_1, c_2\})}{\langle\!\langle\rangle\!\rangle X \xrightarrow{a} \langle\!\langle\rangle\!\rangle X'} \qquad \frac{X \xrightarrow{c_1} X_1, X \xrightarrow{c_2} X_2}{\langle\!\langle\rangle\!\rangle X \xrightarrow{a} (\langle\!\langle\rangle\!\rangle X_1) \parallel (\langle\!\langle\rangle\!\rangle X_2)}$$

$S \triangleright P$ usually does just what S does; P is frozen. However, when S signals that it wants to communicate with P (by performing a d-step), $S \triangleright P$ unfreezes P and lets it take a step in cooperation with S. This operation needs a bit of control state, telling whether or not P is frozen; we use the $\triangleright$ operator when P is frozen, and the $\blacktriangleright$ operator when P is unfrozen.

$$\frac{S \xrightarrow{a} S' \quad (a \neq d)}{S \triangleright P \xrightarrow{a} S' \triangleright P} \qquad \frac{S \xrightarrow{d} S'}{S \triangleright P \xrightarrow{o} S' \blacktriangleright P}$$

The operation $\blacktriangleright$ does one step of communication and then behaves like $\triangleright$.

$$\frac{S \xrightarrow{a} S', \quad P \xrightarrow{a} P'}{S \blacktriangleright P \xrightarrow{o} S' \triangleright P'}$$

We now define the coding of formulas. To make strings of actions easier to read, we write prefixing with a dot: "$d.a.t.S$" instead of "$datS$." Fix an action t, distinct from the previously-mentioned actions, which we use for partial success.

$$\begin{aligned} S_{tt} &= \mathbf{0} \\ S_{\varphi \wedge \psi} &= c_1 S_\varphi + c_2 S_\psi \\ S_{\langle a \rangle \varphi} &= d.a.t.S_\varphi \end{aligned}$$

The context $C_\varphi[X]$ is defined to be $\langle\!\langle\rangle\!\rangle\,(S_\varphi \triangleright X)$. $C_\varphi[P]$ will compute, emitting o's while it is working. Each time it processes an $\langle a \rangle$ correctly, it will emit a t. We will show that $P \models \varphi$ iff $C_\varphi[P]$ produces a trace with as many t's as φ has $\langle\cdot\rangle$'s.

For example, we present one computation path of $C_{\langle a \rangle tt \wedge \langle b \rangle tt}[a + b]$:

$$\begin{aligned} C_{\langle a \rangle tt \wedge \langle b \rangle tt}[a + b] &= \langle\!\langle\rangle\!\rangle\,((c_1.d.a.t + c_2.d.b.t) \triangleright (a + b)) \\ &\xrightarrow{o} \langle\!\langle\rangle\!\rangle\,(d.a.t \triangleright (a + b)) \parallel \langle\!\langle\rangle\!\rangle\,(d.b.t \triangleright (a + b)) \\ &\xrightarrow{o} \langle\!\langle\rangle\!\rangle\,(a.t \blacktriangleright (a + b)) \parallel \langle\!\langle\rangle\!\rangle\,(d.b.t \triangleright (a + b)) \\ &\xrightarrow{o} \langle\!\langle\rangle\!\rangle\,(t \triangleright \mathbf{0}) \parallel \langle\!\langle\rangle\!\rangle\,(d.b.t \triangleright (a + b)) \\ &\xrightarrow{t} \langle\!\langle\rangle\!\rangle\,(\mathbf{0} \triangleright \mathbf{0}) \parallel \langle\!\langle\rangle\!\rangle\,(d.b.t. \triangleright (a + b)) \end{aligned}$$

$$
\begin{aligned}
&\xrightarrow{o} \Diamond\!\!\Diamond\,(\mathbf{0} \triangleright \mathbf{0}) \parallel \Diamond\!\!\Diamond\,(b.t \triangleright (a+b)) \\
&\xrightarrow{o} \Diamond\!\!\Diamond\,(\mathbf{0} \triangleright \mathbf{0}) \parallel \Diamond\!\!\Diamond\,(t \triangleright \mathbf{0}) \\
&\xrightarrow{t} \Diamond\!\!\Diamond\,(\mathbf{0} \triangleright \mathbf{0}) \parallel \Diamond\!\!\Diamond\,(\mathbf{0} \triangleright \mathbf{0})
\end{aligned}
$$

Define $\lfloor \varphi \rfloor$ to be the number of $\langle a \rangle$'s occurring in φ. We say that a trace is *φ-happy* if it has $\lfloor \varphi \rfloor$ t's; otherwise, it is φ-sad. The following lemma is proved by induction on φ.

Lemma 7.1 *If $P \models \varphi$ then some trace of $C_\varphi[P]$ is φ-happy. If $P \not\models \varphi$, then all traces of $C_\varphi[P]$ are φ-sad.*

From this, we conclude that CCSSS partial-trace congruence implies $\langle\rangle$-formula equivalence. Combining the main theorems of this paper so far, we have:

Theorem 7.2 *The following are equivalent for all synchronization trees P and Q:*

1. *P and Q are partial-trace positive GSOS language congruent.*
2. *P and Q are CCSSS partial-trace congruent.*
3. *$P \lesseqgtr Q$*
4. *P and Q agree on all $\langle\rangle$-formulas.*

In fact, each of these equivalences has an associated preorder, such that two trees are equivalent iff each approximates the other. Mutual monosimulation is defined via a preorder. P is a partial-trace approximation of Q with respect to a class $\mathcal{L}$ of languages iff, for each context $C[X]$ over a language $L \in \mathcal{L}$, $\mathrm{ptr}(C[P]) \subseteq \mathrm{ptr}(C[Q])$. P is a $\langle\rangle$-formula approximation of Q iff, for each $\langle\rangle$-formula φ, $P \models \varphi$ implies $Q \models \varphi$. A stronger form of Theorem 7.2 holds: the preorders associated with the four equivalences also coincide.

Corollary 7.3 *True sequencing cannot be defined in a positive GSOS language.*

Proof: By the proof of Theorem 3.2, we see that if true sequencing were definable in a positive GSOS language, then positive GSOS congruence would be at least as fine as ready simulation. It is strictly coarser (being equal to mutual monosimulation); therefore sequencing is not definable.[4] $\square$

[4]CSP-style sequencing can be defined by positive GSOS rules; however, unlike true sequencing, it can be confused by deadlock. CSP-style sequencing is effective in practice.

8 Equational Logic of Mutual Monosimulation

In this section, we give an equational logic for mutual monosimulation. As with any worthwhile notion of process equivalence, mutual monosimulation is not decidable; it is in fact co-r.e. complete, and hence not axiomatizable either. However, CCS contains a class of *finite processes*, *viz.* processes with finite synchronization trees. All such processes are given as CCS terms, involving only the operations $\mathbf{0}$, $P + Q$, and aP. Mutual monosimulation is decidable for these processes (in fact, it is decidable for *regular processes*, those definable by finite-state automata). In this section, we present an inequational axiom system for it. The axioms are based on Milner's axioms for bisimulation [Mil88]. In fact, we need only add a single inequation to make the system complete for mutual monosimulation. The resulting axiom system is sound for all processes, and complete for finite ones.

It is convenient to axiomatize equality as well as approximation. We have the usual rules, stating that approximation is reflexive, antisymmetric, transitive, and respected by all operations (*e.g.*, if $P \sqsubseteq Q$ then $aP \sqsubseteq aQ$), and that equality is precisely mutual approximation. The axioms of $\mathbf{P}$ are:

$$\begin{aligned} P + \mathbf{0} &= P \\ P + Q &= Q + P \\ (P + Q) + R &= P + (Q + R) \\ P + P &= P \\ \mathbf{0} &\sqsubseteq P \end{aligned}$$

Theorem 8.1 $\mathbf{P}$ *is sound and complete for mutual monosimulation of finite processes; that is,* $P \leq Q$ *iff* $\mathbf{P} \vdash P \sqsubseteq Q$*, and so* $P \lesseqgtr Q$ *iff* $\mathbf{P} \vdash P = Q$.

9 Conclusion and Open Problems

We have analyzed the notion of trace equivalence in the setting in which no one, neither operators in the language nor the observer, is capable of the inability of a process to perform an action. In this case, we have three rather different but equivalent characterizations of process equivalence: mutual monosimulation, a relation between synchronization trees; $\langle\rangle$ -formula equivalence, a logical notion; and GSOS partial-trace congruence, an operational notion. We have also given a sound equational logic, which is complete for reasoning about finite processes.

This sort of close correspondance between bisimulation-like relations, modal logic equivalence, congruence with respect to a class of languages generally seems quite powerful. In some cases [GV89,Blo89] it leads to full abstraction results, complete axomatizations for finite processes, and similar insights into processes. This kind of close match thus seems reasonable as one characteristic of a good semantic model of concurrency.

This result depends on the fact that $P \overset{a}{\not\to}$ is not detectable. If either the observer or the language can see that a process P cannot perform an a, then the notion of equivalence becomes ready simulation.

The choice of partial traces is not obviously the right one, especially in the setting without silent moves. The processes aa and $a+aa$ are mutually monosimilar; the latter can die halfway through its computation, while the former cannot. The verification community generally separates problems into safety and liveness conditions; mutual monosimulation is a safety property, and liveness must be treated separately.

This work has all been done in the setting of strong equivalence; there is no silent action. In practice, we generally like to have the notion of weak equivalence, in which one action τ is designated as being "silent," and observations are forbidden to take τ's into account. The extension of this theory to the case of silent actions remains to be investigated.

10 Acknowledgements

I would like to thank Albert Meyer, Juris Hartmanis, and Fred Schneider for several helpful conversations, and Vicki Borah for support.

References

[AL88] Martín Abadi and Leslie Lamport. The existence of refinement mappings. In 3^d *Symp. Logic in Computer Science*, pages 165–177, 1988.

[BIM88] Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced (preliminary report). In *15^{th} Symp. Principles of Programming Languages*, pages 229–239, 1988. (also appears as a MIT Tech Report.).

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

[BM89] Bard Bloom and Albert R. Meyer. A remark on bisimulation between probabilistic processes. In A. R. Meyer and M. A. Taitslin, editors, *Logic at Botik '89*, volume 363 of *Lect. Notes in Computer Sci.*, pages 26–40, 1989.

[dS85] Robert de Simone. Higher-level synchronising devices in MIEJE-SCCS. *Theoretical Computer Sci.*, 37:245–267, 1985.

[GV89] Jan Friso Groote and Frits Vaandrager. Structured operational semantics and bisimulation as a congruence (extended abstract). In 16^{th} *ICALP*, 1989.

[HM80] Matthew Hennessy and Robin Milner. On observing nondeterminism and concurrency. In J. De Bakker and J. van Leeuwen, editors, 7^{th} *ICALP*, volume 85 of *Lect. Notes in Computer Sci.*, pages 299–309. Springer-Verlag, 1980.

[Hoa78] C.A.R. Hoare. Communicating sequential processes. *Comm. ACM*, 21:666–677, 1978.

[HS66] Juris Harmanis and R. Stearns. *Algebraic Structure Theory of Sequential Machines.* Prentice-Hall, 1966.

[Kel76] R. Keller. Formal verification of parallel programs. *CACM*, 19(7):561–572, 1976.

[LS88] Kim Larsen and Arne Skou. Bisimulation through probabilistic testing (preliminary report). Technical Report R 88-16, Institut for Elektroniske Systemer, Aalborg Universitetscenter, June 1988.

[LT88] Nancy Lynch and Mark Tuttle. An introduction to I/O automata. Technical Report MIT/LCF/TM-373, MIT, 1988. (TM-351 revised).

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lect. Notes in Computer Sci.* Springer-Verlag, 1980. 171 pp.

[Mil81] Robin Milner. A modal characterisation of observable machine-behaviour. In *Proceedings of CAAP 81, LNCS 112.* Springer-Verlag, 1981.

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

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

[Mil88] Robin Milner. Operational and algebraic semantics of concurrent processes. Technical Report ECS-LFCS-88-46, LCFS, 1988.