Turning SOS Rules into Equations

Luca Aceto* Bard Bloom† Frits Vaandrager‡

April 3, 1992

Abstract

Many process algebras are defined by structural operational semantics (SOS). Indeed, most such definitions are nicely structured and fit the GSOS format of [7]. We give a procedure for converting any GSOS language definition to a finite complete equational axiom system (except for possibly one infinitary induction principle) which precisely characterizes strong bisimulation of processes.

1 Introduction

One of the main insights of the last decade in semantics was that operational semantics for programming languages are best presented in terms of a *structural operational semantics* (SOS). In such a semantics, the behavior of a composite program is given in terms of the behaviors of its components. Some examples of languages specified by SOSses include [17, 18, 5, 14].

In SOS semantics, induction on terms and on the proofs of transitions are viable proof methods. SOSses are thus a fruitful area for proving properties of programming languages as a whole [21], compilation techniques [20], hardware implementations [5], and so forth.

However, it is often necessary to prove properties of individual programs. While it is in principle possible to work directly with the semantics of the language to verify a program, this can be quite difficult. For example, to verify a concurrent program directly, one might have to effectively calculate its entire transition graph. It is thus helpful to have some more abstract reasoning principles.

One fairly successful method is to give the specification as a (not necessarily implementable) process in the process algebra that the program is written in. One then verifies the program by showing that it is equivalent to (or a suitable approximation of) the specification. Indeed, one of the major schools of theoretical concurrency, that of ACP [2, 4], takes the notion of process equivalence as primary, and defines operational semantics to fit their algebraic laws.

A logic of programs is *complete* (relative to a programming language) if all true formulas of the language are provable in the logic. As properties of interest are generally highly nonrecursive, we are often obliged to have infinitary or other non-recursive rules in our logics to achieve completeness.

1.1 Results

We give an algorithm which yields a finite complete equational axiom system (with possibly one conditional equation) for any language specified by a fairly general form of structural operational semantic rules. We present the algorithm for *strong bisimulation*, the finest useful notion of process equivalence in this setting.

We work in the setting of process algebras, languages such as CCS, CSP, ACP, and MEIJE. A process P is an entity capable of repeatedly performing uninterpreted atomic actions a. The basic operational notion in such languages is

*Hewlett-Packard Labs, Pisa Science Center, Corso Italia 115, 56125 Pisa, Italy; luca@pisa5.italy.hp.com

†Dept. of Computer Science, Cornell University, Ithaca, New York, USA; bard@cs.cornell.edu. Supported by NSF grant (CCR-9003441).

‡Ecole des Mines CMA, Sophia-Antipolis, 06565 Valbonne Cedex, France; frits@cma.cma.fr

0-8186-2735-2/92 $3.00 © 1992 IEEE

$P \xrightarrow{a} P'$, indicating that P is capable of performing the action a and thereafter behaving like P'. In general, processes are nondeterministic; P may have several different possible behaviors after performing an a.

Most such languages have some basic operations (described in detail in Section 2) which allow one to construct the stopped process $\mathbf{0}$, the nondeterministic choice between two processes $P + Q$, and a process which does the action a and thereafter behaves like P, aP.

A typical operation found in many such languages (*e.g.*, [12]) is interleaving parallel composition without communication, which is defined by the rules (one pair of rules for every action a):

$$\frac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} \qquad \frac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'} \tag{1}$$

This is an intuitively reasonable definition of simple parallel composition, and the operational rule is easy to explain. It is somewhat harder to see how to describe it equationally. Some equations are clear enough – it is commutative and associative, and the stopped process is the identity – but the first finite equational description did not appear until [3]. This equational characterization required an additional operation, "left merge" $\between$.[1] For each a, left merge has a rule:

$$\frac{x \xrightarrow{a} x'}{x \between y \xrightarrow{a} x' \parallel y} \tag{2}$$

The equations for $\between$ are:

$$(x_1 + x_2) \between y = (x_1 \between y) + (x_2 \between y) \tag{3}$$

$$(ax) \between y = a(x \parallel y) \tag{4}$$

$$\mathbf{0} \between y = \mathbf{0} \tag{5}$$

$$x \parallel y = (x \between y) + (y \between x) \tag{6}$$

These equations for $\parallel$ and $\between$, together with appropriate axioms for $+$, $a(\cdot)$, and $\mathbf{0}$, form a finite complete equational axiom system for bisimulation of processes defined in terms of these operations.

[1] [15] showed that additional operations, such as $\between$, are indeed required.

In this paper, we give a procedure for extracting from a GSOS specification of an arbitrary process algebra a complete axiom system for bisimulation equivalence (equational, except for possibly one conditional equation). Our procedure introduces new operations as needed, such as $\between$ above. Our methods apply to almost all SOSses for process algebras that have appeared in the literature, and our axiomatizations compare reasonably well with most axioms that have been presented. In particular, they discover the $\between$ characterization of parallel composition. Completeness results for equational axiomatizations are tedious and have become rather standard in many cases. Our generalization of extant completeness results shows that in principle this burden can be completely removed if one gives a GSOS description of a process algebra. Of course, this does not mean that there is nothing to do on specific process algebras. For instance, sometimes it may be possible to eliminate some of the auxiliary operations, or the infinitary conditional equation.

1.2 Outline

Our algorithm generalizes the $\parallel$-$\between$ construction in several ways. Some operations – characterized in Section 4 as "smooth and distinctive" – can be completely axiomatized by *distributive* laws like (3), *action* laws like (4), and *inaction* laws like (5).

Many operations – $\parallel$ is a canonical example – that occur in practice are smooth but not distinctive, and thus cannot be completely described by equations like (3)-(5). In Section 4.5 we show how to express an arbitrary smooth operation as a sum of distinctive smooth operations, as in (6).

The smooth operations were introduced as a technical convenience. They are a restricted form of the (maximally general) class of GSOS operations, forbidding some forms of process copying. In Section 5 we show how to introduce auxiliary smooth operations which do the copying in the equation rather than in the operation. For example, if $g(x)$ makes three copies of x, we introduce a ternary operation $g'(x, y, z)$

such that $g(x) = g'(x, x, x)$. These methods are summarized in Figure 1.

Finally, in Section 6, we discuss completeness. There are two cases: a common simple case, and the fully general one. The equations so far allow us to reduce all processes to *head-normal form*, $\sum a_i P_i$. In a setting in which all processes terminate, this (together with the standard axiomatization of $+$ and $a(\cdot)$) gives a complete proof system between closed terms. In the more general setting (*e.g.*, GSOS languages with some form of recursion), head normalization does not imply general normalization and indeed no finite, purely equational axiom system can be complete; however, the *Approximation Induction Principle* of [2] is sound in our setting, can be expressed in GSOS terms, and makes the equations of Sections 4 and 5 complete.

2 Preliminaries

We assume familiarity with the basic notation of process algebra and structural operational semantics; see *e.g.* [12, 10, 13, 2, 9, 7, 6] for more details.

We use the standard notions of *variable* x and *signature* Σ. We write $\mathbb{T}(\Sigma)$ for the set of all terms over Σ and use $P, Q, \ldots$ to range over terms. The symbol $\equiv$ denotes the relation of syntactic equality on terms. We denote by $\mathrm{T}(\Sigma)$ the set of *closed* terms over Σ, *i.e.*, terms that do not contain variables. An operation symbol f of arity 0 will be often called a *constant* symbol, and the term $f()$ will be abbreviated as f. [2]

A *(closed)* Σ*-substitution* is a function σ from variables to (closed) terms over the signature Σ. For P a term, we write $P\sigma$ for the result of substituting $\sigma(x)$ for each x occurring in P. A Σ*-context* $C[\vec{x}]$ is a term in which at most the variables $\vec{x}$ appear. $C[\vec{P}]$ is $C[\vec{x}]$ with x_i replaced by P_i wherever it occurs. As there are no binding operations, this simple definition suffices.

[2] Most actual process algebras have a notion of *guardedly recursive process definition*. Our methods handle this in the obvious way, with an unwinding equation $\mathrm{rec}[x \Leftarrow P] = P[x := \mathrm{rec}[x \Leftarrow P]]$. We omit this from this study, as it adds far more complexity than insight.

Besides terms we have *actions*, elements of some given finite, nonempty set Act, which is ranged over by a, b, c, d. A *positive transition formula* is a triple of two terms and an action, written $P \xrightarrow{a} P'$. A *negative transition formula* is a pair of a term and an action, written $P \xrightarrow{a} \not\rightarrow$. In general, the terms in the transition formula will contain variables.

Definition 2.1 *Suppose Σ is a signature. A* GSOS rule *ρ over Σ is a rule of the form:*

$$\frac{\bigcup_{i=1}^{l} \left\{ x_i \xrightarrow{a_{ij}} y_{ij} \middle| 1 \le j \le m_i \right\} \cup \bigcup_{i=1}^{l} \left\{ x_i \xrightarrow{b_{ik}} \not\rightarrow \middle| 1 \le k \le n_i \right\}}{f(x_1, \ldots, x_l) \xrightarrow{c} C[\vec{x}, \vec{y}]}$$

where all the variables are distinct, $m_i, n_i \ge 0$, f is an operation symbol from Σ with arity l, and $C[\vec{x}, \vec{y}]$ is a Σ-context with variables including at most the x_i's and y_{ij}'s. (It need not contain all these variables.) Note that the a_{ij}, b_{ik}, and c are actions, and not, as for instance in [19], variables ranging over actions.

It is useful to name components of rules. The operation symbol f is the principal operation *of the rule, and the term $f(\vec{x})$ is the* source. *$C[\vec{x}, \vec{y}]$ is the* target*; c is the* action*; the formulas above the line are the* antecedents*; and the formula below the line is the* consequent. *If, for some i, $m_i > 0$ then we say that ρ* tests its *i-th argument positively. Similarly if $n_i > 0$ then we say that ρ* tests its *i-th argument negatively.*

All rules in this paper (and almost all rules appearing in the literature on process algebra) are examples of GSOS rules.

The intent of a GSOS rule is as follows. Suppose that we are wondering whether $f(\vec{P})$ is capable of taking a c-step. We look at each rule with principal operation f and action c in turn. We inspect each positive antecedent $x_i \xrightarrow{a_{ij}} y_{ij}$, checking if P_i is capable of taking an a_{ij}-step for each j and if so calling the a_{ij}-children Q_{ij}. We also check the negative antecedents; if P_i is *in*capable of taking a b_{ik}-step for each k. If so, then the rule *fires* and $f(\vec{P}) \xrightarrow{c} C[\vec{P}, \vec{Q}]$.

Properties of f	Equations
Smooth + distinctive	Distributive, action, and inaction equations.
Smooth + not distinctive	Introduce distinctive smooth operations f_i, at most one per rule for f, and the equation $f(\vec{x}) = \sum_i f_i(\vec{x})$.
Not smooth	f does more copying than is possible for a smooth operation. Introduce one smooth operation f' with possibly more arguments than f, such that $f(\vec{x})$ is equal to f' applied to a vector of variables consisting of the x_i's suitably repeated, *e.g.* $f(x,y) = f'(x,x,x,y,y)$.

Figure 1: Kinds of equations

Definition 2.2 *A* GSOS system *is a pair* $G = (\Sigma_G, R_G)$ *where* Σ_G *is a finite signature and* R_G *is a finite set of GSOS rules over* Σ_G.

The GSOS discipline is advocated in [6, 7]. Briefly, GSOS rules seem to be a maximal class of rules such that:

1. Every GSOS language has some basic sanity properties; *e.g.*, the transition relation defined informally above can be defined formally; it always exists and is unique (neither of which should be taken for granted, given negative antecedents), and indeed is computable and finitely branching; and it respects many of the stronger notions of process equivalence, in particular bisimulation and ready simulation.

2. It seems impossible to extend the format of the rules in any systematic way which preserves the basic sanity properties. Unlike 1, this is informal; [6] gives a series of examples showing that the most natural extensions violate the basic sanity properties of 1. There are other consistent rule formats, such as the *tyft/tyxt* format of [9] and the *ntyft/ntyxt* format of [8]. These two formats respect strong bisimulation, but induce transition systems that are in general neither computable nor finitely branching.

Our study takes *(strong) bisimulation* as the primitive notion. Briefly, two processes are strongly bisimilar iff whenever one can perform an action, the other can as well and the resulting processes are still strongly bisimilar; for a full definition see [13]. We use the notation $=_G$, or simply $=$, for strong bisimulation of terms in the GSOS system G.

Finally, we say that the GSOS system G' *disjointly extends* G, notation $G \sqsubseteq G'$, if the signature and rules of G' include those of G, and G' introduces no new rules for operations of G. In this case, $=_G$ and $=_{G'}$ coincide on terms of G.

3 The Problem

For a GSOS system G, let $\mathsf{Bisim}(G)$ denote the quotient algebra of closed Σ_G-terms modulo bisimulation. That is, for $P, Q \in \mathbb{T}(\Sigma_G)$,

$$\mathsf{Bisim}(G) \models P = Q \quad \Leftrightarrow$$

$$(\forall \text{ closed } \Sigma_G\text{-substitutions } \sigma : P\sigma =_G Q\sigma).$$

The main problem addressed in this paper is to find a *complete* axiomatization of bisimulation on closed terms – that is, equality in $\mathsf{Bisim}(G)$ – for an arbitrary GSOS system specification G. That is, we want to find a *finite* (conditional) equational theory T such that for all *closed* terms $P, Q \in \mathrm{T}(\Sigma_G)$,

$$T \vdash P = Q \quad \Leftrightarrow \quad \mathsf{Bisim}(G) \models P = Q.$$

Moller [15] has shown that bisimulation congruence over a subset of the usual CCS algebra with the interleaving operation $\|$ cannot be completely characterized by any finite set of equational axioms over that language. Thus, our program requires the addition of auxiliary operations to G.

We first define FINTREE, a simple fragment of CCS suitable for expressing finite trees. (Most

process algebras already contain the FINTREE operations, either directly or as derived operations.) FINTREE has a constant symbol $\mathbf{0}$ denoting the null process; unary symbols $a(\cdot)$, one for each action in Act, denoting action prefixing; and a binary symbol $+$ for nondeterministic choice. The null process is incapable of taking any action, and consequently has no rules. For each action a there is a rule $ax \xrightarrow{a} x$. The operational semantics of $P + Q$ is defined by the rules

$$\frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'} \tag{7}$$

The following completeness result is well-known [11, 13]:

Lemma 3.1 *Let T_{FINTREE} be the theory consisting of the equations*

$$x + y = y + x \tag{8}$$
$$(x + y) + z = x + (y + z) \tag{9}$$
$$x + x = x \tag{10}$$
$$x + \mathbf{0} = x \tag{11}$$

Then T_{FINTREE} is complete for equality in Bisim(FINTREE).

As a typical example of the way in which the above completeness result is used, consider the GSOS system $G_{\not\parallel}$, which extends FINTREE with a family of operations $\not\parallel B$, where B is a finite set of actions, with rules

$$\frac{x \xrightarrow{a} y}{x \not\parallel B \xrightarrow{a} y} \quad a \notin B$$

The process $P \not\parallel B$ behaves like P, except that it cannot do any actions from B in its first move. We use $\not\parallel$ to axiomatize negative premises.

Lemma 3.2 *Let $T_{\not\parallel}$ be the theory that extends T_{FINTREE} with the equations*

$$(x + y) \not\parallel B = x \not\parallel B + y \not\parallel B \tag{12}$$
$$ax \not\parallel B = ax \quad \text{if } a \notin B \tag{13}$$
$$ax \not\parallel B = \mathbf{0} \quad \text{if } a \in B \tag{14}$$
$$\mathbf{0} \not\parallel B = \mathbf{0} \tag{15}$$

Then $T_{\not\parallel}$ is complete for equality in $\mathsf{Bisim}\left(G_{\not\parallel}\right)$.

Proof: (Sketch) Using (12)-(15), it is possible to eliminate all occurrences of $\not\parallel$'s from terms. By Lemma 3.1, the axioms of FINTREE now suffice to prove bisimulation of $\not\parallel$-free terms. $\triangle\!\!\!+$

We like to generalize the idea of Lemma 3.2 to obtain complete axiomatizations of bisimulation equivalence for arbitrary GSOS systems. But first we have to discuss a subtlety. For the rest of this paper, we are not so much interested in the fact that the axioms of T_{FINTREE} are valid in the 'small' algebra Bisim(FINTREE). We would rather like to know that the axioms are valid in any disjoint extension G of FINTREE, because this will then allow us to use the T_{FINTREE} axioms to reason in the 'large' algebra $\mathsf{Bisim}(G)$. In general it is not the case that validity of equations is preserved by taking disjoint extensions. For instance, consider the trivial GSOS system NIL consisting of the single constant symbol $\mathbf{0}$ and with no rules. The law $x = y$ is valid in Bisim(NIL), but clearly does not hold in Bisim(FINTREE), even though FINTREE is a disjoint extension of NIL.

Fortunately, the T_{FINTREE} laws (and also all the other laws that we will discuss in this paper) are preserved by taking disjoint extensions. To formalize this observation we introduce, for G a GSOS system, the class $\mathsf{BISIM}(G)$ of all algebras $\mathsf{Bisim}(G')$, for G' a disjoint extension of G. Thus we have, for $P, Q \in \mathbb{T}(\Sigma_G)$,

$$\mathsf{BISIM}(G) \models P = Q \quad \Leftrightarrow$$
$$(\forall G' : G \sqsubseteq G' \;\Rightarrow\; \mathsf{Bisim}(G') \models P = Q).$$

Almost without any additional difficulty we can prove the following generalizations of the soundness results for FINTREE and $G_{\not\parallel}$:

Lemma 3.3 $\mathsf{BISIM}(\text{FINTREE}) \models T_{\text{FINTREE}}$ *and* $\mathsf{BISIM}\left(G_{\not\parallel}\right) \models T_{\not\parallel}$.

4 Smooth Operations

In this section, we show how to axiomatize a substantial subclass of GSOS operations. Distributive laws, like (12), are essential for our completeness result. However, in general we cannot hope

to get distributivity laws for arbitrary GSOS operations. The situation is particularly hopeless in the case of what we will call *non-smooth* operations. In this section we give axioms for the simpler *smooth* operations, using the still simpler *distinctive* operations as a base case. Full GSOS operations are deferred to Section 5.

Our goal in this section is a *head normal form* theorem for smooth systems. The distributivity, action, and inaction laws for distinctive smooth operations developed in Sections 4.1 - 4.3 suffice for obtaining head normal forms for such operations. In Section 4.5 we extend this to all smooth systems.

Definition 4.1 *A GSOS rule is* smooth *if it takes the form*

$$\frac{\left\{x_i \xrightarrow{a_i} y_i | i \in I\right\} \cup \left\{x_i \not\xrightarrow{b_{ij}} | i \in K, 1 \leq j \leq n_i\right\}}{f(x_1, \ldots, x_l) \xrightarrow{c} C[\vec{x}, \vec{y}]} \tag{16}$$

where I, K partition $\{1, \ldots, l\}$, and no x_i with $i \in I$ appears in $C[\vec{x}, \vec{y}]$. An operation from a GSOS system G is smooth *if all the rules for this operation are smooth. G is* smooth *if it contains smooth rules only.*

The format of smooth rules generalizes the format of De Simone [19, 9] since it allows restricted forms of negative hypotheses and copying. We will not motivate smoothness in this paper; it is a technical condition chosen to get proofs to work. (1), (2), and (7) are smooth rules. An excellent example of a non-smooth operation is the priority operation θ of Baeten, Bergstra and Klop [1]. Fix a partial-ordering relation $>$ on Act. For each a the operation θ has a rule

$$\frac{x \xrightarrow{a} x', \quad x \not\xrightarrow{b} \text{ (for all } b > a)}{\theta(x) \xrightarrow{a} \theta(x')} \tag{17}$$

For nontrivial $>$, θ is non-smooth since it tests its argument with both positive and negative antecedents.

4.1 Distributivity Laws

In general, smooth operations do not distribute over $+$ in all their arguments; for example, $\|$ defined by (1) is not distributive:

$$(a + b) \| c + (a + b) \| d \neq (a + b) \| (c + d)$$

as the left side must choose between c and d on its first action, while the right side may delay that decision. Other smooth operations distribute over $+$ in some arguments, and depend parametrically on the remaining arguments.

Lemma 4.2 *Let f be an l-ary smooth operation of a GSOS system G that disjointly extends* FIN-TREE, *and suppose i is an argument of f for which* each *rule for f has a positive antecedent. Then f distributes over $+$ in its i-th argument,* i.e.,

$$\mathsf{BISIM}(G) \models f(x_1, \ldots, x_i + y_i, \ldots, x_l) = f(x_1, \ldots, x_i, \ldots, x_l) + f(x_1, \ldots, y_i, \ldots, x_l)$$

The laws (3) and (12) are instances of Lemma 4.2.

4.2 Action Laws

We now derive *action laws*, which tell when a process can take an action. The a-rule for $\between$ fires if x can take an a-step. Phrased as an equation, this reads:

$$(ax) \between y = a(x \| y) \tag{18}$$

Next consider an operation whose definition involves negative hypotheses. For illustration, we choose the (useless) operation $\star$, defined by the single rule:

$$\frac{y \not\xrightarrow{a}, y \not\xrightarrow{b}}{x \star y \xrightarrow{c} x + y}$$

For any process Q such that $Q \not\xrightarrow{a}$ and $Q \not\xrightarrow{b}$, we know that $P \star Q = c(P + Q)$. We code this negative information into equations using the $\between\!\!\!/$ operation. Note that for any process S, $(S \between\!\!\!/ \{a, b\}) \not\xrightarrow{a}$ and $(S \between\!\!\!/ \{a, b\}) \not\xrightarrow{b}$. That is, $(y \between\!\!\!/ \{a, b\})$ ranges over all processes which cannot take either a or b steps on their first move. Hence the following law holds:

$$x \star (y \between\!\!\!/ \{a, b\}) = c(x + (y \between\!\!\!/ \{a, b\})) \tag{19}$$

The trick used to derive equations (18) and (19) cannot be used for smooth operations in general. But it does work if we assume the additional technical condition of *distinctiveness.*

Definition 4.3 *A smooth operation f from a GSOS system G is* distinctive *if, for each argument i, either all rules for f test i positively or none of them does, and moreover for each pair of different rules for f there is an argument for which both rules have a positive antecedent, but with a different action.*

For example, $\mathbf{0}$, $a(\cdot)$, $\not\parallel B$, and $\mathbb{L}$ are distinctive whereas $+$ and $\|$ are not. The relabelling and restriction operations from CCS are both distinctive.

Lemma 4.4 *Suppose f is a distinctive smooth operation of a disjoint extension G of $G_{\not\parallel}$, with a rule*

$$\frac{\left\{x_i \xrightarrow{a_i} y_i | i \in I\right\} \cup \left\{x_i \not\xrightarrow{b_{ij}} | i \in K, 1 \le j \le n_i\right\}}{f(\vec{x}) \xrightarrow{c} C[\vec{x}, \vec{y}]}$$

Let $B_i = \{b_{ij} | 1 \le j \le n_i\}$ and

$$P_i \equiv \begin{cases} a_i y_i & i \in I \\ x_i \not\parallel B_i & i \in K \wedge \varnothing \subsetneq B_i \subsetneq \mathsf{Act} \\ x_i & i \in K \wedge B_i = \varnothing \\ \mathbf{0} & i \in K \wedge B_i = \mathsf{Act} \end{cases}$$

Then

$$\mathsf{BISIM}(G) \models f(\vec{P}) = c \,.\, C[\vec{P}, \vec{y}] \qquad (20)$$

The laws (4), (13), and (19) are instances of Lemma 4.4. The $B_i = \mathsf{Act}$ and $B_i = \varnothing$ cases are formally unnecessary. However, they make the resulting rules much simpler; *e.g.*, we have terms $f(x, y, \mathbf{0})$ instead of $f(x \not\parallel \varnothing, y \not\parallel \varnothing, z \not\parallel \mathsf{Act})$. In our experience they are the most common cases appearing in practice.

4.3 Inaction Laws

We also need to know when $f(\vec{P}) = \mathbf{0}$. The term $f(\vec{P})$ is bisimilar to $\mathbf{0}$ iff it has no outgoing transitions; that is, iff for each rule ρ there is a reason why it cannot fire. The reason could be an argument i such that either ρ requires P_i to do some action that it can't do, or ρ requires P_i not to do an action that it does. The following lemma covers enough of these cases for our purposes.

Lemma 4.5 *Suppose f is an l-ary smooth operation of a GSOS system G that disjointly extends* FINTREE, *and suppose that, for $1 \le i \le l$, term P_i is of the form $\mathbf{0}$, x_i, ax_i or $ax_i + y_i$. Suppose further that for each rule for f of the form (16) there is an index i such that either (1) $i \in I$ and $P_i \equiv \mathbf{0}$ or $P_i \equiv ax_i$ for some $a \neq a_i$, or (2) $i \in K$ and $P_i \equiv b_{ij}x_i + y_i$ for some $1 \le j \le n_i$. Then*

$$\mathsf{BISIM}(G) \models f(\vec{P}) = \mathbf{0} \qquad (21)$$

The laws (5), (14) and (15) are instances of Lemma 4.5.

4.4 Head Normal Forms

The purpose of distributivity, action and inaction laws is to rewrite process expressions into *head normal forms.* Head normalization is the heart of the completeness proof in Section 6.

Definition 4.6 *A term P over a signature $\Sigma \supseteq \Sigma_{\text{FINTREE}}$ is in* head normal form *if it is of the form $\sum a_i P_i$. A theory T over Σ is* head normalizing *for P if there exists a Σ-term Q in head normal form such that $T \vdash P = Q$.*

The following theorem generalizes the elimination result in the proof of Lemma 3.2, and is proved in much the same way.

Theorem 4.7 (Informal) *The distributivity, action, and inaction laws suffice to reduce any closed term consisting of distinctive smooth operations to head normal form.*

4.5 General Smooth Operations

Many operations occurring in practice are smooth but not distinctive. We show how to axiomatize smooth operations via sums of distinctive smooth operations. Consider the true

sequencing operation ";", defined by the rules (one pair of rules for each action a):

$$\frac{x \xrightarrow{a} x'}{x; y \xrightarrow{a} x'; y} \qquad \frac{x \xrightarrow{b} \text{(for all } b\text{)}, \quad y \xrightarrow{a} y'}{x; y \xrightarrow{a} y'}$$

This operation is smooth but not distinctive. Now consider the operations "$;_1$" and "$;_2$" that one obtains by partitioning the rules for ";":

$$\frac{x \xrightarrow{a} x'}{x ;_1 y \xrightarrow{a} x'; y} \qquad \frac{x \xrightarrow{b} \text{(for all } b\text{)}, \quad y \xrightarrow{a} y'}{x ;_2 y \xrightarrow{a} y'}$$

If $P; Q \xrightarrow{a} S$, then this transition must be enabled by one of the sequencing rules for a. Hence, either $P ;_1 Q \xrightarrow{a} S$ or $P ;_2 Q \xrightarrow{a} S$. That is, the following law is sound:

$$x; y = (x ;_1 y) + (x ;_2 y)$$

This trick generalizes to all smooth operations.

Lemma 4.8 *Suppose G is a GSOS system with* FINTREE $\sqsubseteq G$, *and suppose f is an l-ary smooth operation of G. Then there exists a disjoint extension G' of G with l-ary distinctive smooth operations $f_1, \ldots, f_n$ such that, for all $\vec{x}$ of length l,*

$$\mathsf{BISIM}(G') \models f(\vec{x}) = f_1(\vec{x}) + \cdots + f_n(\vec{x}) \quad (22)$$

Proof: (Sketch) Let $R_1, \ldots, R_n$ be a partitioning of the set R of rules for f such that, for all i, f is distinctive in the GSOS system obtained from G by removing all the rules in $R - R_i$. Such a partition always exists because if one introduces one set R_i for each rule for f, the restriction of f to the single rule in R_i trivially yields a distinctive operation. Define $\Sigma_{G'}$ to be the signature obtained by extending Σ_G with fresh l-ary operation symbols $f_1, \ldots, f_n$. Next define R'_G to be the set of rules obtained by extending R_G, for each i, with rules derived from the rules of R_i by replacing the operation symbol in the source by f_i. $\triangle$

This lemma allows a clever (or exhaustive-search) algorithm to produce only a small number of auxiliary operations, by choosing a small partition. For example, one good partition for ; produces $;_1$ and $;_2$. The singleton partition, with each rule in a separate set, always works: for ;, this would produce operations $;_{1a}$ and $;_{2a}$ for each a.

Lemma 4.9 *Suppose G is a GSOS system with $G_I \sqsubseteq G$. Let $\Sigma \subseteq \Sigma_G - \Sigma_I$ be a collection of smooth operations of G. Then there exist*

1. *a disjoint extension G' of G with a finite collection Σ' of distinctive smooth operations, and*
2. *a finite equational theory T that extends T_I,*

such that $\mathsf{BISIM}(G') \models T$ *and T is head normalizing for all terms in* $\mathrm{T}(\Sigma' \cup \Sigma \cup \Sigma_I)$.

FINTREE rather than G_I suffices if G has no negative rules.

5 General GSOS operations

The results of the previous section give us head normalization for all smooth operations. In this section we show how to axiomatize non-smooth operations. An operation can fail to be smooth by using an argument in too many different ways: having more than one positive rule concerning an argument; having both positive and negative rules concerning the same argument; or having an antecedent $x_i \xrightarrow{a_{ij}} y_{ij}$ and having x_i appear in the target. The following operation illustrates all of these problems:

$$\frac{x \xrightarrow{a} y_1, \quad x \xrightarrow{b} y_2, \quad x \not\xrightarrow{c}}{g(x) \xrightarrow{c} x + y_1}$$

In this case, we introduce an auxiliary smooth operation with one argument for each distinct kind of use of x:

$$\frac{x_1 \xrightarrow{a} y_1, \quad x_2 \xrightarrow{a} y_2, \quad x_0 \not\xrightarrow{c}}{g'(x_0, x_1, x_2) \xrightarrow{c} x_0 + y_1}$$

The main use of g' is as a smoothed version of g, by setting $x_0 = x_1 = x_2 = x$. It is clear that, for all z, we have:

$$g(z) = g'(z, z, z)$$

This trick generalizes to all GSOS operations.

Lemma 5.1 *Suppose G is a GSOS system containing a non-smooth operation f with arity l. Then there exists a disjoint extension G' of G with a smooth operation f' with arity l' (possibly different from l), and there exist vectors $\vec{z}$ of l distinct variables, and $\vec{v}$ of l' variables in $\vec{z}$ (possibly repeated), such that*

$$\mathsf{BISIM}(G') \models f(\vec{z}) = f'(\vec{v}) \qquad (23)$$

Theorem 5.2 *Let G be a GSOS system. Then there exist a disjoint extension G' of G, and a finite equational theory T, such that* $\mathsf{BISIM}(G') \models T$ *and T is head normalizing for all terms in* $\mathrm{T}(\Sigma_{G'})$.

5.1 Example: the Priority Operation

As an example of application of the strategy presented in Sections 4-5, we will now present an axiomatization of the priority operation θ of Baeten, Bergstra and Klop defined in (17). We do not have a distributivity law for θ. For instance, if $b > a$, then clearly $b \rightarrow \theta(a+b) \not\rightarrow \theta(a) + \theta(b) \rightarrow a + b$. Following Lemma 5.1, we therefore define a smoothed version of θ in the form of a fresh binary operation $\triangle$ with rules (one for each $a \in \mathsf{Act}$):

$$\frac{x \xrightarrow{a} x', \quad y \not\xrightarrow{b} \text{ (for all } b > a)}{x \triangle y \xrightarrow{a} \theta(x')} \qquad (24)$$

Note that $\triangle$ is a distinctive smooth operation. The relationships between θ and $\triangle$ are expressed by the following instance of law (23):

$$\theta(x) \;=\; x \triangle x$$

The distinctive smooth operation $\triangle$ can be axiomatized using the strategy of Theorem 4.7. Let $V_a = \{b \in \mathsf{Act} | b > a\}$. We then have:

$$\begin{aligned}
(x+y)\triangle z &= x\triangle z + y\triangle z \\
ax\triangle y &= a.\theta(x) \\
&\qquad \text{if } a \text{ is maximal} \\
ax\triangle(y \between V_a) &= a.\theta(x) \\
&\qquad \text{if } a \text{ is not maximal} \\
\mathbf{0}\triangle x &= \mathbf{0} \\
ax\triangle(by+z) &= \mathbf{0} \qquad \text{if } b > a
\end{aligned}$$

The axiomatization of the priority operation obtained by applying our general strategy compares rather well with the one given in [1]. The axiomatization given there also relies on the introduction of an auxiliary operation, the *unless* operation $\lhd$. Ignoring termination issues, this operation may be specified by the following rules (one for each a):

$$\frac{x \xrightarrow{a} x', \quad y \not\xrightarrow{b} \text{ (for all } b > a)}{x \lhd y \xrightarrow{a} x'}$$

which are quite similar to (24).

6 Completeness

For any GSOS system G, the algorithm presented in Sections 4-5 allows us to generate a disjoint GSOS extension G' with a finite head-normalizing equational theory. In the case of $\|$ and $\between$ in Section 1, this equational theory is strong enough to eliminate these operations from all terms, so that we can use Lemma 3.1 to obtain completeness.

However, this will not work in general, as head normalization does not imply general normalization. Consider, for example, a constant ω with rule $\omega \xrightarrow{a} \omega$. The instance of action law (20) for this operation is

$$\omega \;=\; a.\omega$$

and, obviously, the process of elimination of the constant symbol ω is not going to terminate.

In Section 6.1 we consider the case of terminating processes: these terms can be reduced to FINTREE terms and the completeness of FINTREE applies. In Section 6.2 we consider the general case: the reduction to FINTREE does not apply, but a suitable infinitary rule gives completeness.

6.1 Completeness for Well-Founded GSOS Systems

Definition 6.1 *Let G be a GSOS system. A term $P \in \mathrm{T}(\Sigma_G)$ is* well-founded *iff there exists no infinite sequence $P_0, a_0, P_1, a_1, P_2, \ldots$ of*

terms in $\mathrm{T}(\Sigma_G)$ *and actions in* Act *with* $P \equiv P_0$ *and* $P_i \xrightarrow{a_i}_G P_{i+1}$ *for all* $i \geq 0$. G *is* well-founded *iff all terms in* $\mathrm{T}(\Sigma_G)$ *are well-founded.*

It is immediate to see that any GSOS system which includes the constant ω given above is not well-founded. On the other hand, the class of well-founded GSOS systems contains the recursion-free finite-alphabet sublanguages of most of the standard process algebras, and is thus of some interest.

For well-founded GSOS systems G, it is possible to iterate the reduction of terms to head normal forms a finite number of times to eliminate all non-FINTREE operations. As in the proof of Lemma 3.2, this reduces completeness to head normalization.

Theorem 6.2 *Suppose that G is a GSOS system. Let G' and T denote the disjoint extension of G, and the finite head normalizing equational theory constructed by the methods of Sections 4-5, respectively. Suppose P and Q are well-founded terms in $\mathrm{T}(\Sigma_{G'})$. Then*

$$\mathsf{Bisim}(G') \models P = Q \quad \Leftrightarrow \quad T \vdash P = Q.$$

The definition of well-foundedness for a GSOS system G given in Definition 6.1 relies upon properties of the transition relation $\rightarrow_G$. In the full paper, we will describe sufficient syntactic conditions under which a GSOS system will be well-founded.

6.2 Completeness for General GSOS Systems

As bisimulation of finitely branching processes is easily seen to be Π_1-hard and provable equality from a finite set of equations is Σ_1-complete, the extension of the completeness result given in Theorem 6.2 to general GSOS systems requires some reasoning principles beyond purely equational logic. However, it is possible to extend our results to the whole class of GSOS systems in a rather standard way. Bisimulation equivalence over finitely branching labelled transition systems supports a powerful induction principle, known as the *Approximation Induction Principle* (AIP), see [2]. All GSOS processes are finitely branching [6], so the AIP applies.

We introduce a family of operations $\pi_n(\cdot), n \in \mathbf{N}$. These operations are known as *projection* operations in the literature on ACP [2]. Intuitively, $\pi_n(P)$ allows P to perform n moves freely, and then stops it. It is straightforward to define $\pi_n(\cdot)$ with distinctive smooth rules. The Approximation Induction Principle is the following infinitary conditional equation:

$$\frac{\pi_n(x) = \pi_n(y) \quad (\text{for all } n)}{x = y}$$

Intuitively, AIP states a "continuity" property of bisimulation, namely that if two processes are equivalent at any finite depth then they are equivalent.

The projection operations themselves are somewhat heavy-handed, as there are infinitely many of them, and GSOS systems are defined to be finite. Fortunately, it is possible to mimic the projection operations by means of a single binary operation, denoted by $\cdot/\cdot$. Intuitively, P/H runs the process P until the "hourglass" process H runs out and stops taking steps. That is, for all actions $a, b \in$ Act, we have the following rule for $\cdot/\cdot$:

$$\frac{x \xrightarrow{a} x', \quad y \xrightarrow{b} y'}{x/y \xrightarrow{a} x'/y'}$$

(In particular, when $H \not\rightarrow$, then $P/H \not\rightarrow$.) For each $n \in \mathbf{N}$, $\pi_n(P) = P/b^n$, where

$$b^n \triangleq \underbrace{b.\ldots.b}_{n\text{-times}}.\mathbf{0}$$

In this formulation, we may rephrase the Approximation Induction Principle as follows:

$$\frac{x/b^n = y/b^n \quad (\text{for all } n)}{x = y}$$

Note that $\cdot/\cdot$ is smooth and distinctive. Applying the strategy presented in Section 4, we automatically derive the following equations for it:

$$(x + y)/z = x/z + y/z \qquad (25)$$

$$x/(y+z) = x/y + x/z \quad (26)$$
$$ax/by = a(x/y) \quad (27)$$
$$\mathbf{0}/y = \mathbf{0} \quad (28)$$
$$x/\mathbf{0} = \mathbf{0} \quad (29)$$

For any GSOS system G, $G_/$ will be used to denote the disjoint extension of G with the operation $\cdot/\cdot$.

Proposition 6.3 $\mathsf{BISIM}\big(\mathrm{FINTREE}_/\big) \models AIP$.

Theorem 5.2 shows that the equational rules presented so far suffice to give the one-step behaviour of all closed terms, and hence their n-step behaviour for all $n \in \mathbf{N}$. For any closed term P and integer n, equations (25)-(29) for the operation $\cdot/\cdot$ can then be used to obtain FINTREE-terms which exhibit the same n-step behaviour as P.

Theorem 6.4 *Suppose G is a GSOS system. Let G' and T denote the disjoint extension of $G_/$ and the finite equational theory for it given in Theorem 5.2, respectively. Then T and AIP together are complete for equality in* $\mathsf{Bisim}(G')$.

7 Further Research

Thus far the discussion of this paper took place in a setting with strong bisimulation equivalence. However, most of our results easily extend to other process equivalences as well, at least in a setting without internal actions. We have chosen bisimulation equivalence because it is widely viewed to be the finest acceptable process equivalence.[3] Thus other process equivalences will be coarser, and the corresponding process algebras can be obtained as homomorphic images of algebras based on strong bisimulation. Since validity of equations is preserved by taking homomorphic images, this implies that the soundness part of our method extends trivially; completeness, of course, will require extra equations.

[3] In fact, at the price of some minor complications it is possible to redo the work of this paper using the even finer equivalence notion of *synchronization tree isomorphism*.

However, conditional equations are not in general preserved by homomorphisms, and the soundness (much less the completeness) of the AIP is nontrivial. Furthermore, the auxiliary operations we introduce may not respect all equivalences weaker than bisimulation; *e.g.*, $\mathbb{L}$ does not respect weak failure equivalence. Thus extending the results of this paper to weak process equivalences is a delicate matter.

Of course, several variations are possible on our method to obtain complete axiom systems for GSOS languages. In the full paper we will study one such variation, which is interesting because it does not use the $\between$ operations, and also because of its nice term rewriting properties.

It would also be desirable to axiomatize bisimilarity between open terms, proving for example equations such as $x \| (y \| z) = (x \| y) \| z$. Methods for proving such equations will appear in later work.

Other forms of SOS specification remain to be investigated. It is not clear how to handle SOS methods which, like the *tyxt/tyft* rules of [9] and infinitary GSOS rules, can produce infinitely-branching processes.

References

[1] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Syntax and defining equations for an interrupt mechanism in process algebra. *Fundamenta Informaticae*, IX(2):127–168, 1986.

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

[3] J.A. Bergstra and J.W. Klop. Fixed point semantics in process algebras. Report IW 206, Mathematisch Centrum, Amsterdam, 1982.

[4] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Computation*, 60(1/3):109–137, 1984.

[5] G. Berry. A hardware implementation of Pure Esterel. Rapport de Recherche 06/91, Ecole des Mines, CMA, Sophia-Antipolis, France, 1991.

[6] B. Bloom. *Ready Simulation, Bisimulation, and the Semantics of CCS-like Languages.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, August 1989.

[7] B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced: preliminary report. In *Conference Record of the* 15^{th} *ACM Symposium on Principles of Programming Languages,* San Diego, California, pages 229–239, 1988. Full version available as Technical Report 90-1150, Department of Computer Science, Cornell University, Ithaca, New York, August 1990.

[8] R.N. Bol and J.F. Groote. The meaning of negative premises in transition system specifications (extended abstract). In J. Leach Albert, B. Monien, and M. Rodríguez, editors, *Proceedings* 18^{th} *ICALP,* Madrid, LNCS 510, pages 481–494. Springer-Verlag, 1991. Full version available as Report CS-R9054, CWI, Amsterdam, 1990.

[9] J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence (extended abstract). In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors, *Proceedings* 16^{th} *ICALP,* Stresa, LNCS 372, pages 423–438. Springer-Verlag, 1989. Full version to appear in *Information and Computation.*

[10] M. Hennessy. *Algebraic Theory of Processes.* MIT Press, Cambridge, Massachusetts, 1988.

[11] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM,* 32(1):137–161, 1985.

[12] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, Englewood Cliffs, 1985.

[13] R. Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs, 1989.

[14] R. Milner. Functions as processes. In Paterson [16], pages 167–180.

[15] F. Moller. The importance of the left merge operator in process algebras. In Paterson [16], pages 752–764.

[16] M. Paterson, editor. *Proceedings* 17^{th} *ICALP,* Warwick, LNCS 443, Springer-Verlag, July 1990.

[17] G.D. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[18] G.D. Plotkin. An operational semantics for CSP. In D. Bjørner, editor, *Proceedings IFIP TC2 Working Conference on Formal Description of Programming Concepts – II,* Garmisch, pages 199–225, Amsterdam, 1983. North-Holland.

[19] R. de Simone. Higher-level synchronising devices in MEIJE–SCCS. *Theoretical Computer Science,* 37:245–267, 1985.

[20] S. Weber, B. Bloom, and G. Brown. Compiling Joy to silicon: A verified silicon compilation scheme. To appear in the proceedings of the Brown/MIT Conference on VLSI and Parallel Systems, November 1991.

[21] A.K. Wright and M. Felleisen. A syntactic approach to type soundness. Technical Report TR91-160, Rice University, 1991.