Turning SOS Rules into Equations

Luca Aceto
COGS,
University of Sussex,
Brighton, United Kingdom;
luca@cogs.sussex.ac.uk

Bard Bloom
Dept. of Computer Science,
Cornell University,
Ithaca, New York, USA;
bard@cs.cornell.edu

Frits Vaandrager
Department of Software Technology, CWI
P.O. Box 4079,
1009 AB Amsterdam,
The Netherlands;
Programming Research Group,
University of Amsterdam,
Kruislaan 403,
1098 SJ Amsterdam,
The Netherlands;
fritsv@cwi.nl

Abstract

Many process algebras are defined by structural operational semantics (SOS). Indeed, most such definitions are nicely structured and fit the GSOS format of [19]. We give a procedure for converting any GSOS language definition to a finite complete equational axiom system (possibly with one infinitary induction principle) which precisely characterizes strong bisimulation of processes.

1991 Mathematics Subject Classification: 68Q05, 68Q55.
1991 CR Categories: D.1.3, D.3.1, F.1.1, F.3.1, F.3.2, F.3.3.
Key Words and phrases: Structural Operational Semantics (SOS), GSOS format, bisimulation equivalence, process algebra, equational logic, complete axiomatizations, approximation induction principle.
Note: The research of the second author was supported by NSF grant (CCR-9003441). Most of the work on this paper was done while the first author was employed by Hewlett-Packard Laboratories, Pisa Science Center, and the third author was employed by the Ecole des Mines, CMA, at Sophia Antipolis, France.

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 process is given in terms of the behaviors of its components. Some examples of languages specified by SOSses include [43, 44, 3, 4, 36, 15].

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 [52], compilation techniques [51], hardware implementations [15], and logics of programming [46].

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 calculate its entire transition graph. It is thus helpful to have some more abstract reasoning principles.

There are several methods of specifying processes, *e.g.* modal formulas [49] and variants of Hoare logic [39, 48]. One fairly successful verification technique 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 [10, 13], 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 nonrecursive, we are often obliged to have infinitary or other nonrecursive 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 infinitary 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 [35], CSP [31], ACP, and MEIJE [5]. A process P is an entity capable of repeatedly performing uninterpreted atomic actions a. The basic operational notion in such languages is $P \xrightarrow{a} P'$, 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 action a.

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

A typical operation found in many such languages (*e.g.*, [31]) 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 [12]. This equational characterization required an additional operation, "left merge" $\mathbin{\|\!\!\_}$.[1] Intuitively, $P \mathbin{\|\!\!\_} Q$ is like $P \| Q$ except that the first move must be taken by P. For each a, left merge has a rule:

$$\frac{x \xrightarrow{a} x'}{x \mathbin{\|\!\!\_}\, y \xrightarrow{a} x' \parallel y} \tag{2}$$

The equations for $\|$ and $\mathbin{\|\!\!\_}$ are:

$$\begin{aligned}
(x+y) \mathbin{\|\!\!_}\, z &= (x \mathbin{\|\!\!_}\, z) + (y \mathbin{\|\!\!_}\, z) && (3)\\
(ax) \mathbin{\|\!\!_}\, y &= a(x \parallel y) && (4)\\
\mathbf{0} \mathbin{\|\!\!_}\, x &= \mathbf{0} && (5)\\
x \parallel y &= (x \mathbin{\|\!\!_}\, y) + (y \mathbin{\|\!\!_}\, x) && (6)
\end{aligned}$$

These equations for $\|$ and $\mathbin{\|\!\!\_}$, 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.

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 $\mathbin{\|\!\!\_}$ 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 $\mathbin{\|\!\!\_}$ 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 $\|$-$\mathbin{\|\!\!\_}$ construction in several ways. Some operations – characterized in Section 4.1 as "smooth and distinctive" – can be completely axiomatized by *distributive* laws like (3), *action* laws like (4), and *inaction* laws like (5).

[1] [38] showed that additional operations, such as $\mathbin{\|\!\!\_}$, are indeed required.

Properties of f	**Equations**
Smooth + distinctive	Distributive, action, and inaction equations.
Smooth + not distinctive	Introduce smooth, distinctive 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

Many operations – $\|$ is a canonical example – that occur in practice are smooth in the sense defined in Section 4.1 but not distinctive, and thus cannot be completely described by equations like (3)-(5). In Section 4.1.5 we show how to express an arbitrary smooth operation as a sum of smooth distinctive operations, as in (6).

The smooth operations were introduced as a technical convenience. They are a restricted form of the (very general) class of GSOS operations, forbidding some forms of process copying. In Section 4.2 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.

In Section 5, we discuss completeness. There are two cases: a common simple case, and the fully general one. The equations generated 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 $+$, $a(\cdot)$ and $\mathbf{0}$) gives a complete proof system between closed terms. We describe sufficient syntactic conditions under which all processes terminate for a given GSOS system. In the more general setting (*e.g.*, GSOS languages with recursive process definition), head normalization does not imply general normalization and indeed no finite, purely equational axiom system can be complete; however, the *Approximation Induction Principle* of [14, 10] is sound in our setting, can be expressed in GSOS terms, and makes the equations of Sections 4.1 and 4.2 complete.

Of course, variations are possible on our method to obtain complete axiom systems for GSOS languages. In Section 6, we study one such variation, which uses fewer auxiliary operations. Finally, in Section 7, we discuss some topics for future research.

2 Preliminaries

We assume familiarity with the basic notation of process algebra and structural operational semantics; see *e.g.* [31, 29, 35, 10, 27, 19, 16] for more details.

We start from a countably infinite set Var of *process variables* with typical elements x, y. A *signature* Σ consists of a set of *operation symbols*, disjoint from Var, together with a function *arity* that assigns a natural number to each operation symbol. The set of *terms* over Σ is the least set such that[2]

- Each $x \in \mathsf{Var}$ is a term.
- If f is an operation symbol of arity l, and $P_1, \dots, P_l$ are terms, then $f(P_1, \dots, P_l)$ is a term.

We write $\mathbb{T}(\Sigma)$ for the set of all terms over Σ and use $P, Q, \dots$ 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. By convention, whenever we write a term-like phrase (*e.g.*, $f(P, Q)$), we intend it to be a term (*e.g.*, f is binary).

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.

Besides terms we have *actions*, elements of some given nonempty, finite 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 $P \stackrel{a}{\to} P'$. A *negative transition formula* is a pair of a term and an action, written $P \stackrel{a}{\not\to}$. 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 pair containing a set of transition formulas and a single transition formula, written*

$$\frac{\bigcup_{i=1}^{l} \left\{ x_i \stackrel{a_{ij}}{\to} y_{ij} | 1 \le j \le m_i \right\} \quad \cup \quad \bigcup_{i=1}^{l} \left\{ x_i \stackrel{b_{ik}}{\not\to} | 1 \le k \le n_i \right\}}{f(x_1, \dots, x_l) \stackrel{c}{\to} C[\vec{x}, \vec{y}]} \tag{7}$$

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 [47], 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*. An operation f* tests its i-th argument positively (resp. negatively) *if it occurs as principal operation of a rule that tests its i-th argument positively (resp. negatively).*

All rules in this paper (and almost all rules appearing in the literature on process algebra) are examples of GSOS rules.[3] Informally, the intent of a GSOS rule is as follows. Suppose

[2] Most actual process algebras have a notion of *guardedly recursive process definition*; *e.g.*, a constant x satisfying the equation $x = P$ where P is a term containing x used safely. Our methods handle this in the obvious way: adding the axiom scheme $\mathrm{rec}\,[x \Leftarrow P] = P[x := \mathrm{rec}\,[x \Leftarrow P]]$. We omit this from this study, as it adds far more complexity than insight. It is worth noting that any finite set of recursively-defined processes may be added to any GSOS language as fresh constants (nullary operators) defined by GSOS rules.

[3] GSOS may as well stand for "Grand SOS."

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 $f(\vec{P}) \xrightarrow{c} C[\vec{P}, \vec{Q}]$.

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 [16, 19]. Briefly, GSOS rules seem to be a maximal class of rules such that:

1. Every GSOS system 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. Moreover, every operation in a GSOS system respects many of the stronger notions of process equivalence, in particular bisimulation [41, 34] and ready simulation [19].

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; [16] 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 [27] and the *ntyft/ntyxt* format of [20]. These two formats respect strong bisimulation, but induce transition systems that are in general neither computable nor finitely branching.

We will now formally define the transition relation induced by a GSOS system, and state precisely some of the basic sanity properties of the format.

Definition 2.3 *A* transition relation *over a signature Σ is a relation $\rightsquigarrow \subseteq \mathrm{T}(\Sigma)\times\mathsf{Act}\times\mathrm{T}(\Sigma)$. We write $P \overset{a}{\rightsquigarrow} Q$ as an abbreviation for $(P, a, Q) \in \rightsquigarrow$.*

Definition 2.4 *A* [closed] Σ-substitution *is a function σ from variables to [closed] terms over the signature Σ. For t a term, transition formula, GSOS rule, etc., we write $t\sigma$ for the result of substituting $\sigma(x)$ for each x occurring in t. For $\vec{P}$ and $\vec{x}$ two vectors with the same length of, respectively, terms and distinct variables, $\langle\vec{P}/\vec{x}\rangle$ denotes the substitution that replaces the i-th variable of $\vec{x}$ by the i-th term of $\vec{P}$, and leaves all other variables unchanged.*

Definition 2.5 *Suppose $\rightsquigarrow$ is a transition relation and σ a closed substitution. For each transition formula φ, the predicate $\rightsquigarrow, \sigma \models \varphi$ is defined by*

$$\begin{aligned} \rightsquigarrow, \sigma \models P \xrightarrow{a} Q &\triangleq P\sigma \overset{a}{\rightsquigarrow} Q\sigma \\ \rightsquigarrow, \sigma \models P \overset{a}{\nrightarrow} &\triangleq \nexists Q : P\sigma \overset{a}{\rightsquigarrow} Q \end{aligned}$$

For H a set of transition formulas, we define

$$\rightsquigarrow, \sigma \models H \;\triangleq\; \forall \varphi \in H : \rightsquigarrow, \sigma \models \varphi$$

and for $\dfrac{H}{\varphi}$ *a GSOS rule,*

$$\rightsquigarrow, \sigma \models \frac{H}{\varphi} \;\triangleq\; \Big(\rightsquigarrow, \sigma \models H \;\;\Rightarrow\;\; \rightsquigarrow, \sigma \models \varphi\Big).$$

Definition 2.6 *Suppose G is a GSOS system and $\rightsquigarrow$ is a transition relation over Σ_G. Then $\rightsquigarrow$ is* sound *for G iff for every rule $\rho \in R_G$ and every closed Σ_G-substitution σ, we have $\rightsquigarrow, \sigma \models \rho$. A transition $P \overset{a}{\rightsquigarrow} Q$ is* supported *by some rule $\dfrac{H}{\varphi} \in R_G$ iff there exists a substitution σ such that $\rightsquigarrow, \sigma \models H$ and $\varphi\sigma = \Big(P \xrightarrow{a} Q\Big)$. The relation $\rightsquigarrow$ is* supported *by G iff each transition in $\rightsquigarrow$ is supported by a rule in R_G.*

Lemma 2.7 ([19]) *For each GSOS system G there is a unique sound and supported transition relation.*

We write $\rightarrow_G$ for the unique sound and supported transition relation for G. Note that the definition of a GSOS system does not exclude *junk* rules, *i.e.*, rules that support no transition in $\rightarrow_G$. For example, the rule

$$\frac{x \xrightarrow{a} y, \qquad x \overset{a}{\not\rightarrow}}{f(x) \xrightarrow{a} f(y)}$$

has contradictory antecedents and can never fire. Also it can be the case that a rule like

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

does not support any transition if $\rightarrow_G$ contains no a-transitions. In general it is decidable whether a given rule is junk, but the decision procedure is nontrivial. As none of the results in this paper requires the absence of junk rules, we saw no reason to complicate definitions in order to exclude them.

Lemma 2.8 ([19]) *Suppose G is a GSOS system. Then, for each closed Σ_G-term P, the set $\Big\{(a, Q) | P \xrightarrow{a}_G Q\Big\}$ is finite and can be effectively computed in canonical index.*

The basic notion of equivalence among terms of a GSOS system we will consider in this paper is that of *bisimulation.*

Definition 2.9 *Suppose G is a GSOS system. A binary relation $\sim \;\subseteq \mathrm{T}(\Sigma_G) \times \mathrm{T}(\Sigma_G)$ over closed terms is a* bisimulation *if $P \sim Q$ implies, for all $a \in \mathsf{Act}$,*

1. *If $P \xrightarrow{a}_G P'$ then, for some Q', $Q \xrightarrow{a}_G Q'$ and $P' \sim Q'$.*
2. *If $Q \xrightarrow{a}_G Q'$ then, for some P', $P \xrightarrow{a}_G P'$ and $P' \sim Q'$.*

We write $P \leftrightarroweq_G Q$ if there exists a bisimulation $\sim$ relating P and Q. The subscript G is omitted when it is clear from context.

Lemma 2.10 ([19]) *Suppose G is a GSOS system. Then $\leftrightarroweq_G$ is an equivalence relation and a congruence for all operation symbols f of G,* i.e., $(\forall i : P_i \leftrightarroweq_G Q_i) \Rightarrow f(\vec{P}) \leftrightarroweq_G f(\vec{Q})$.

Definition 2.11 *A GSOS system H is a* disjoint extension *of a GSOS system G, notation $G \sqsubseteq H$, if the signature and rules of H include those of G, and H introduces no new rules for operations of G.* [4]

If H disjointly extends G then H introduces no new outgoing transitions for terms of G. This means in particular that, for $P, Q \in \mathrm{T}(\Sigma_G)$, $P \leftrightarroweq_G Q \Leftrightarrow P \leftrightarroweq_H Q$. Note that $\sqsubseteq$ is a partial order.

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 $P, Q \in \mathbb{T}(\Sigma_G)$,

$$\mathsf{Bisim}(G) \models P = Q \quad \Leftrightarrow \quad (\forall \text{ closed } \Sigma_G\text{-substitutions } \sigma : P\sigma \leftrightarroweq_G Q\sigma).$$

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 [38] 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 $\mathsf{FINTREE}$, a simple fragment of CCS suitable for expressing finite trees. (Most process algebras already contain the $\mathsf{FINTREE}$ operations, either directly or as derived operations.) $\mathsf{FINTREE}$ 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 non-deterministic 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 (for each a)

$$\frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'} \tag{8}$$

[4] Our notion of disjoint extension is more restrictive than the semantic one of "conservative extension" studied by Groote and Vaandrager [27], and Bol and Groote [20]. The restriction is essential in the technical development of the paper; it is needed for instance for Lemma 4.12.

The intent of these rules is that, whenever a transition is known to be possible for P or Q, that transition is possible for $P+Q$. For example, we have $a(b+c) \xrightarrow{a} (b+c) \begin{array}{l}\xrightarrow{b} \mathbf{0}\\ \xrightarrow{c} \mathbf{0}\end{array}$, where we write a for $a\mathbf{0}$ in process terms. The following completeness result is well-known [30, 35]:

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

$$
\begin{aligned}
x + y &= y + x &(9)\\
(x+y)+z &= x+(y+z) &(10)\\
x + x &= x &(11)\\
x + \mathbf{0} &= x &(12)
\end{aligned}
$$

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

We will use the standard process algebra conventions for the $\mathsf{FINTREE}$ language. Thus $a(\cdot)$ binds stronger, and $+$ binds weaker than all other operations, and we write $a\,.\,P$ or just aP for $a(P)$. Also, for $I = \{i_1, \ldots, i_n\}$ a finite index set, we write $\sum_{i\in I} P_i$ for $P_{i_1} + \cdots + P_{i_n}$. By convention, $\sum_{i\in\varnothing} P_i$ stands for $\mathbf{0}$.

As a typical example of the way in which the above completeness result is used, consider the GSOS system $G_\dagger$, which extends $\mathsf{FINTREE}$ with a family of operations $\dagger B$ (called ∂^1_B in [6]), where B is a finite set of actions, with rules

$$\frac{x \xrightarrow{a} y}{x \dagger B \xrightarrow{a} y}\, a \notin B$$

The process $P \dagger B$ behaves like P, except that it cannot do any actions from B in its first move. In later sections, we will use $\dagger$ to axiomatize negative premises. Note that $\dagger$ is different from the CCS restriction operation, as CCS restriction is persistent (static) and $\dagger$ goes away after one move.

Lemma 3.2 *Let $T_\dagger$ be the theory that extends T_{FINTREE} with the equations*

$$
\begin{aligned}
(x+y) \dagger B &= (x \dagger B) + (y \dagger B) & &(13)\\
ax \dagger B &= ax & \textit{if } a \notin B\quad &(14)\\
ax \dagger B &= \mathbf{0} & \textit{if } a \in B\quad &(15)\\
\mathbf{0} \dagger B &= \mathbf{0} & &(16)
\end{aligned}
$$

Then $T_\dagger$ is complete for equality in $\mathsf{Bisim}(G_\dagger)$.

Proof: Let $P, Q \in \mathrm{T}(\Sigma_\dagger)$. We must show that $T_\dagger \vdash P = Q \;\Leftrightarrow\; \mathsf{Bisim}(G_\dagger) \models P = Q$.

The implication '$\Rightarrow$' (soundness) is straightforward and in fact subsumed by the later results of this paper. We sketch the proof of implication '$\Leftarrow$' (completeness). Suppose $\mathsf{Bisim}(G_\dagger) \models P = Q$. We have to show $T_\dagger \vdash P = Q$. By induction on the structure of terms one can easily establish that for each term $S \in \mathrm{T}(\Sigma_{\mathsf{FINTREE}})$ and for each B, there exists a term $S' \in \mathrm{T}(\Sigma_{\mathsf{FINTREE}})$ such that $T_\dagger \vdash S \dagger B = S'$. Using this fact, a subsequent trivial structural induction shows that all occurrences of $\dagger$ can be eliminated from terms, *i.e.*, for each term $S \in \mathrm{T}(\Sigma_\dagger)$ there exists a term $S' \in \mathrm{T}(\Sigma_{\mathsf{FINTREE}})$ such that $T_\dagger \vdash S = S'$. So let $P', Q' \in \mathrm{T}(\Sigma_{\mathsf{FINTREE}})$ with $T_\dagger \vdash P = P'$ and $T_\dagger \vdash Q = Q'$. By the soundness of the axioms and using the fact that $\leftrightarrows$ is an equivalence, we obtain $\mathsf{Bisim}(G_\dagger) \models P' = Q'$. P' and Q' only mention the $\mathsf{FINTREE}$ operations. Since $G_\dagger$ is a disjoint extension of $\mathsf{FINTREE}$, we infer $\mathsf{Bisim}(\mathsf{FINTREE}) \models P' = Q'$. Thus, by Lemma 3.1, $T_{\mathsf{FINTREE}} \vdash P' = Q'$. Then certainly $T_\dagger \vdash P' = Q'$. Combination of a proof of $P' = Q'$ with proofs of $P = P'$ and $Q = Q'$ yields a proof of $P = Q$ from $T_\dagger$. ♧

We like to generalize the idea of Lemma 3.2 to obtain complete axiomatizations of bisimulation equivalence for arbitrary GSOS systems, *i.e.*, we want to find laws, on top of the T_{FINTREE} axioms, that allow us to eliminate all the other operations from terms, thus reducing the completeness result for the full language to proving equality in $\mathsf{FINTREE}$, which is solved. This requires a variety of methods, which will appear over the next three sections.

Before presenting these methods, however, 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 $\mathsf{Bisim}(\mathsf{FINTREE})$. We would rather like to know that the axioms are valid in any disjoint extension $G \sqsupseteq \mathsf{FINTREE}$, 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 $\mathsf{Bisim}(\mathrm{NIL})$, but clearly does not hold in $\mathsf{Bisim}(\mathsf{FINTREE})$, even though $\mathsf{FINTREE}$ 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 $P, Q \in \mathbb{T}(\Sigma_G)$,

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

Almost without any additional difficulty we can prove the following generalizations of the soundness results for $\mathsf{FINTREE}$ and $G_\dagger$:

Lemma 3.3 $\mathsf{BISIM}(\mathsf{FINTREE}) \models T_{\mathsf{FINTREE}}$ *and* $\mathsf{BISIM}(G_\dagger) \models T_\dagger$.

For ease of reference, the notions of satisfaction and provability that we will use throughout the paper are summarized in Figure 2.

4 The Axiomatization of GSOS Operations

We now present the core of our strategy for axiomatizing GSOS operations. We proceed in two steps: first, we shall show how to axiomatize a class of particularly well-behaved operations, the smooth operations introduced in Section 4.1. Secondly, we shall extend our results to arbitrary GSOS operations in Section 4.2.

Notation	Definition
$T \vdash P = Q$	The equation $P = Q$ is provable from the set of axioms T.
$\mathsf{Bisim}(G) \models P = Q$	The equation $P = Q$ holds in the algebra $\mathsf{Bisim}(G)$, *i.e.*, $P\sigma \leftrightarrows_G Q\sigma$ for all closed Σ_G-substitutions σ.
$\mathsf{BISIM}(G) \models P = Q$	For all GSOS rule systems G' which disjointly extend G, $\mathsf{Bisim}(G') \models P = Q$.

Figure 2: Notions of Satisfaction

4.1 Smooth Operations

In this subsection, we show how to axiomatize a substantial subclass of GSOS operations. Distributive laws, like (13), 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 subsection we give axioms for the simpler *smooth* operations[5], using the still simpler *distinctive* operations as a base case.

Our goal in this subsection is a *head normal form* theorem for smooth systems. The distributivity, action, and inaction laws for distinctive smooth operations developed in Sections 4.1.1 - 4.1.3 suffice for obtaining head normal forms for such operations. In Section 4.1.5 we extend this to all smooth systems.

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

$$\frac{\left\{x_i \xrightarrow{a_i} y_i | i \in I\right\} \;\cup\; \left\{x_i \overset{b_{ij}}{\not\to} | i \in K, 1 \le j \le n_i\right\}}{f(x_1, \ldots, x_l) \xrightarrow{c} C[\vec{x}, \vec{y}]} \tag{17}$$

where I, K are disjoint sets such that $I \cup K = \{1, \ldots, l\}$, 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.*

We will not motivate smoothness in this paper; it is a technical condition chosen to get proofs to work. The name "smooth" comes from the lack of "barbs"; that is, positive and negative tests on the same term.

The format of smooth rules generalizes the format of De Simone [47, 27] since it allows restricted forms of negative hypotheses and copying. If there is a positive hypothesis $x_i \xrightarrow{a_i} y_i$, then y_i may be copied; however, x_i must not appear *at all*, even if y_i is completely destroyed. If there are no positive antecedents about x_j, then x_j can be copied. (1), (2), and (8) are smooth rules.

An excellent example of a non-smooth operation is the priority operation θ of Baeten, Bergstra and Klop [7]. Fix a partial ordering relation $>$ on Act. For each a the operation θ has a rule

$$\frac{x \xrightarrow{a} x', \quad x \overset{b}{\not\to} \text{ (for all } b > a)}{\theta(x) \xrightarrow{a} \theta(x')} \tag{18}$$

[5] We use the term in an utterly different setting than [45], and no confusion should arise.

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

As a technical notion that will be useful later on, we define the notion of a *trigger* of an l-ary smooth operation f: an l-tuple over $\mathsf{Act} \cup 2^{\mathsf{Act}}$ (we assume disjointness of Act and 2^{Act}) which characterizes when some rule for f will fire.

Definition 4.2 *The* trigger *of rule (17) is the l-tuple $\langle e_1, \ldots, e_l \rangle$, where*

$$e_i = \begin{cases} a_i & \text{if } i \in I \\ \{b_{ij} | 1 \leq j \leq n_i\} & \text{otherwise} \end{cases}$$

For example, the triggers for $\|$ are $\langle a, \varnothing \rangle$ and $\langle \varnothing, a \rangle$ for every a.

4.1.1 Distributivity Laws

In general also smooth operations do not distribute over $+$ in all their arguments; for example, $\|$ defined by (1) is distributive in neither of its arguments. For instance,

$$(a \| b) + (a \| c) \not\leftrightarrow a \| (b + c)$$

as the left side must choose between b and c on its first action, while the right side may delay that decision. A symmetric argument shows that $\|$ is also not distributive in its first argument. Other smooth operations distribute over $+$ in some arguments, and depend parametrically on the remaining ones, as in (3).

Lemma 4.3 *Let f be an l-ary smooth operation of a GSOS system G that disjointly extends* FINTREE*, 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.,

$$\mathsf{BISIM}(G) \models f(x_1, \ldots, x_i + y_i, \ldots, x_l) = f(x_1, \ldots, x_i, \ldots, x_l) + f(x_1, \ldots, y_i, \ldots, x_l) \quad (19)$$

The laws (3) and (13) are instances of Lemma 4.3. However, this lemma does not help for $\|$, as neither argument appears positively in every rule.

4.1.2 Action Laws

We now derive *action laws*, which tell when a process can take an action. The a-rule for $\llfloor$ given in (2) fires if x can take an a-step. Phrased as an equation, this reads:

$$(ax) \llfloor y = a(x \| y) \quad (20)$$

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

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

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

$$x \star (y \dagger \{a,b\}) = c\Big(x + (y \dagger \{a,b\})\Big) \tag{21}$$

The trick used to derive equations (20) and (21) cannot be used for smooth operations in general. But it does work if we assume the additional technical condition of *distinctiveness.* Informally, an operation is distinctive if at most one rule can apply (and we can tell which one) under certain restricted circumstances.

Definition 4.4 *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, $\mathbf{0}$, $a(\cdot)$, $\dagger B$ and $\Bbbk$ are distinctive whereas $+$ and $\|$ are not. The relabelling and restriction operations from CCS are both distinctive.

Lemma 4.5 *Suppose f is a distinctive smooth operation of a disjoint extension G of $G_\dagger$, with a rule*

$$\frac{\Big\{x_i \stackrel{a_i}{\rightarrow} y_i | i \in I\Big\} \;\cup\; \Big\{x_i \stackrel{b_{ij}}{\nrightarrow} | i \in K, 1 \leq j \leq n_i\Big\}}{f(\vec{x}) \stackrel{c}{\rightarrow} C[\vec{x},\vec{y}]} \tag{22}$$

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

$$P_i \equiv \begin{cases} a_i y_i & i \in I \\ x_i \dagger B_i & i \in K \wedge \varnothing \subsetneq B_i \subsetneq \mathsf{Act} \\ x_i & i \in K \wedge B_i = \varnothing \\ \mathbf{0} & i \in K \wedge B_i = \mathsf{Act} \end{cases}$$

Then

$$\mathsf{BISIM}(G) \models f(\vec{P}) = c\,.\,C[\vec{P},\vec{y}] \tag{23}$$

Proof: For any closed substitution σ, rule (22) fires from $(f(\vec{P}))\sigma$. Because f is distinctive, it is in fact the only rule that fires from this term. It is easy to check that the resulting transition matches the single outgoing transition of $(c\,.\,C[\vec{P},\vec{y}])\sigma$. $\clubsuit$

The laws (4), (14), and (21) are instances of Lemma 4.5. The $B_i = \mathsf{Act}$ and $B_i = \varnothing$ cases are formally unnecessary. However, they make the resulting equations much simpler; *e.g.*, we have term $f(x,y,\mathbf{0})$ instead of $f(x \dagger \varnothing, y \dagger \varnothing, z \dagger \mathsf{Act})$. In our experience they are the most common cases appearing in practice.

4.1.3 Inaction Laws

Distributivity laws describe the interaction between arbitrary GSOS operations and the $+$ operation. Similarly action laws describe the interaction with the prefixing operations $a(\cdot)$. In this section we present *inaction laws* to describe the interaction between arbitrary operations and the $\mathsf{FINTREE}$ constant $\mathbf{0}$; that is, laws which say that $f(\vec{P}) = \mathbf{0}$.

A term of the form $f(\vec{P})$ is bisimilar to $\mathbf{0}$ iff it has no outgoing transitions; that is, iff for each rule ρ for f 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 4.6 *Suppose f is an l-ary smooth operation of a GSOS system G that disjointly extends $\mathsf{FINTREE}$, and suppose that, for $1 \leq i \leq l$, 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 (17) 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 $P_i \equiv b_{ik}x_i + y_i$ for some $1 \leq k \leq n_i$. Then*

$$\mathsf{BISIM}(G) \models f(\vec{P}) = \mathbf{0} \tag{24}$$

The laws (5), (15) and (16) are instances of Lemma 4.6. As an example of application of Lemma 4.6 to obtain inaction equations, consider the (useless) operation f given by the rules

$$\frac{x \xrightarrow{a} x', \quad y \stackrel{a}{\not\rightarrow}}{f(x,y) \xrightarrow{a} \mathbf{0}} \qquad \frac{x \xrightarrow{b} x', \quad y \stackrel{b}{\not\rightarrow}}{f(x,y) \xrightarrow{a} \mathbf{0}}$$

Calculating from Lemma 4.6, we obtain the following. Consider $f(P,Q)$ where P and Q are as given in the lemma. If $f(P,Q)$ is to exclude the first rule, then either $P = \mathbf{0}$, $P = bx$, or $Q = ay + y'$. If $f(P,Q)$ is to exclude the second rule, then $P = \mathbf{0}$, $P = ax$, or $Q = by + y'$. Fitting these together in all possible ways, we have the following equations:

1. If $P = \mathbf{0}$, then both rules are excluded by form (1) of the lemma, so Q can be anything:

$$\begin{array}{rcl} f(\mathbf{0},\mathbf{0}) & = & \mathbf{0} \\ f(\mathbf{0},y) & = & \mathbf{0} \\ f(\mathbf{0},ay) & = & \mathbf{0} \\ f(\mathbf{0},by) & = & \mathbf{0} \\ f(\mathbf{0},ay+y') & = & \mathbf{0} \\ f(\mathbf{0},by+y') & = & \mathbf{0} \end{array} \tag{25}$$

2. If $P = bx$, then the first rule is excluded by form (1); however, we must exclude the second rule as well. Clearly if $P = bx$, we cannot take $P = \mathbf{0}$ or $P = ax$; we thus have only

$$f(bx, by + y') = \mathbf{0} \tag{26}$$

3. Finally, if the first rule is excluded by $Q = ay + y'$, then either restriction on P can exclude the second rule. We have already given the rule for $P = \mathbf{0}$; so this leaves only

$$f(ax, ay + y') = \mathbf{0} \tag{27}$$

Note that most of the equations for $P = \mathbf{0}$ are redundant; they all follow from $f(\mathbf{0}, y) = \mathbf{0}$. We discuss this further in Section 4.1.4.

Lemma 4.6 also gives equations of the form

$$f(\vec{x}) = \mathbf{0}$$

for each operation symbol f with no rules having it as principal operation. For example, we get the equation

$$x \dagger \mathsf{Act} = \mathbf{0}$$

4.1.4 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 5.

Definition 4.7 *A term P over a signature $\Sigma \supseteq \Sigma_{\mathsf{FINTREE}}$ 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 small lemma about $\dagger$ will be needed below.

Lemma 4.8 *Suppose $P \equiv \sum a_i P_i$, $B \subseteq \mathsf{Act}$ and, for all i, $a_i \notin B$. Then $T_\dagger \vdash P = P \dagger B$.*

Proof: If $P \equiv \mathbf{0}$ then the claim follows by (16). Otherwise,

$$\begin{array}{rcll} P & = & \sum a_i P_i & \\ & = & \sum (a_i P_i \dagger B) & \quad (\text{as } a_i \notin B \text{ by } (14)) \\ & = & (\sum a_i P_i) \dagger B & \quad (\text{by } (13)) \\ & = & P \dagger B & \end{array}$$

Theorem 4.9 *Suppose G is a GSOS system with $G_\dagger \sqsubseteq G$. Let $\Sigma \subseteq \Sigma_G - \Sigma_\dagger$ be a collection of distinctive smooth operations of G. Let T be the finite equational theory that extends $T_\dagger$ with the following axioms, for each operation f from Σ,*

1. *for each argument i of f that is tested positively, a distributivity axiom (19),*
2. *for each rule for f an action law (23),*
3. *all the inaction laws (24) for f.*

Then $\mathsf{BISIM}(G) \models T$, and T is head normalizing for all terms in $\mathrm{T}(\Sigma \cup \Sigma_\dagger)$.

Proof: The fact that $\mathsf{BISIM}(G) \models T$ follows immediately from Lemmas 4.3–4.6. We are thus left to show that T is head normalizing for all terms in $\mathrm{T}(\Sigma \cup \Sigma_\dagger)$. This will follow via a trivial induction from the following claim.

Claim *Suppose f is an l-ary operation symbol from Σ, and $P_1, \ldots, P_l \in \mathbb{T}(\Sigma_G)$ are all in head normal form. Then there exists a term $P \in \mathbb{T}(\Sigma_G)$ in head normal form such that $T \vdash f(P_1, \ldots, P_l) = P$.*

The proof of the claim proceeds by induction on the combined size of $P_1, \ldots, P_l$. There are three cases.

Case 1 There is an argument i that is tested positively by f and for which P_i is of the form $P_i' + P_i''$. In this case we can apply one of the distributivity laws (19) to infer

$$T \vdash f(P_1, \ldots, P_i' + P_i'', \ldots, P_l) = f(P_1, \ldots, P_i', \ldots, P_l) + f(P_1, \ldots, P_i'', \ldots, P_l)$$

Next the induction hypothesis gives that there exist head normal forms P' and P'' such that $T \vdash f(P_1, \ldots, P_i', \ldots, P_l) = P'$ and $T \vdash f(P_1, \ldots, P_i'', \ldots, P_l) = P''$. Hence $T \vdash f(P_1, \ldots, P_l) = P' + P''$, and the induction step follows.

Case 2 There is an argument i that is tested positively by f and for which $P_i \equiv \mathbf{0}$. Since f is distinctive, all rules for f test i positively. Thus T contains an inaction law $f(x_1, \ldots, x_{i-1}, \mathbf{0}, x_{i+1}, \ldots, x_l) = \mathbf{0}$. Instantiation of this law gives $T \vdash f(\vec{P}) = \mathbf{0}$, and the induction step follows.

Case 3 For all arguments k that are tested positively by f, P_k is of the form $a_k P_k'$. We consider two subcases.

Case 3.1 For each rule for f with trigger $\langle e_1, \ldots, e_l \rangle$, there is an i that is tested positively such that $e_i \neq a_i$. Then T contains an inaction law $f(\vec{Q}) = \mathbf{0}$, where $Q_k \equiv a_k x_k$ if k is tested positively, and $Q_k \equiv x_k$ otherwise. Instantiation of this law gives $T \vdash f(\vec{P}) = \mathbf{0}$, and the induction step follows.

Case 3.2 There exists a rule ρ for f with trigger $\langle e_1, \ldots, e_l \rangle$ such that $e_k = a_k$ for all k that are tested positively. Since f is distinctive, ρ is in fact the unique rule with this property. Again there are two subcases (the last ones).

Case 3.2.1 There is an index j that is not tested positively, and there is an action $b \in e_j$ such that T proves an equation of the form $P_j = bP_j' + P_j''$. Now we note that T contains an inaction law $f(\vec{Q}) = \mathbf{0}$, where $Q_k \equiv a_k x_k$ if k is tested positively, $Q_k \equiv b x_k + y_k$ if $k = j$, and $Q_k \equiv x_k$ otherwise. Application of this law gives $T \vdash f(P_1, \ldots, bP_j' + P_j'', \ldots, P_l) = \mathbf{0}$, and the induction step follows.

Case 3.2.2 For each index n that is not tested positively, P_n is of the form $\sum a_{ij} P_{ij}$ with, for all j, $a_{ij} \notin e_n$. In this case we can apply Lemma 4.8 to replace, for all n with $\varnothing \neq e_n \subsetneq \mathsf{Act}$, P_n by $P_n \dagger e_n$. Next an application of the action law (23) corresponding to ρ suffices to obtain the required head normal form. $\clubsuit$

Extraneous inaction laws: In Theorem 4.9 we included, for simplicity, *all* the inaction laws (24) in the theory T. Inspection of the above proof shows, however, that we can strengthen the theorem by restricting T to those equations $P = \mathbf{0}$ that satisfy the following conditions:

1. On the positions that are tested positively, subterms take the form $\mathbf{0}$, x or ax, and on the other positions subterms take the form x or $bx + y$.

2. P has at most one subterm of the form $bx + y$.

3. $P = \mathbf{0}$ cannot be obtained as a substitution instance of another equation in T.

Condition (3) can be applied, for instance, if T contains an inaction law of the form $g(ax_1, x_2) = \mathbf{0}$ for a binary operation g with two arguments that are tested positively. In this case it is not necessary to have an additional inaction law $g(ax_1, bx_2) = \mathbf{0}$.

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

$$\frac{x \xrightarrow{a} x'}{x;y \xrightarrow{a} x';y} \qquad \frac{x \stackrel{b}{\not\rightarrow} \text{(for all } b\text{)}, \quad y \xrightarrow{a} y'}{x;y \xrightarrow{a} y'} \tag{28}$$

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

$$\frac{x \xrightarrow{a} x'}{x;_1 y \xrightarrow{a} x';y} \qquad \frac{x \stackrel{b}{\not\rightarrow} \text{(for all } b\text{)}, \quad y \xrightarrow{a} y'}{x;_2 y \xrightarrow{a} y'} \tag{29}$$

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:

$$x;y = (x;_1 y) + (x;_2 y) \tag{30}$$

This trick generalizes to all smooth operations.

Lemma 4.10 *Suppose G is a GSOS system with $\mathsf{FINTREE} \sqsubseteq G$, 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,*

$$\mathsf{BISIM}(G') \models f(\vec{x}) = f_1(\vec{x}) + \cdots + f_n(\vec{x}) \tag{31}$$

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. It is routine to check that for all disjoint extensions G'' of G', and for all $P_1, \ldots, P_l \in \mathrm{T}(\Sigma_{G''})$, $f(\vec{P}) \leftrightarrows_{G''} f_1(\vec{P}) + \cdots + f_n(\vec{P})$. $\triangle$

Of course one always likes to minimize the number of auxiliary operations in a complete axiomatization. In most cases it is not needed to introduce an auxiliary operation for each rule for f, which corresponds to taking the singleton partition of R, and one can do much better by following a more "optimistic" approach. In this approach one starts with the partition $R_1, \ldots, R_n$ consisting of the subsets of rules that test the same arguments positively. We are not aware of any operation occurring in the literature on process algebras for which this procedure leads to operations that are not distinctive, but if one encounters a non-distinctive operation, one just continues to refine the corresponding block in the partition until one has distinctive operations only.

As a corollary of Theorem 4.9 and Lemma 4.10, we obtain the following result.

Lemma 4.11 *Suppose G is a GSOS system with $G_\dagger \sqsubseteq G$. Let $\Sigma \subseteq \Sigma_G - \Sigma_\dagger$ be a collection of smooth operations of G. Then there exist*

1. *a disjoint extension G' of G with a finite collection $\Sigma' = \Sigma_{G'} - \Sigma_G$ of distinctive smooth operations, and*

2. *a finite equational theory T that extends $T_\dagger$,*

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

Proof: Straightforward from Lemma 4.10 and Theorem 4.9.

4.1.6 Examples

We demonstrate our method on the parallel composition operation $\|$ of ACP [13, 10]. Fix a partial, commutative and associative function $\gamma : \mathsf{Act} \times \mathsf{Act} \rightharpoonup \mathsf{Act}$, which describes the synchronization between actions. If, for simplicity, we do not consider the issue of successful termination, the $\|$ operation can be described by the rules (for all $a, b, c \in \mathsf{Act}$):

$$\frac{x \xrightarrow{a} x'}{x \| y \xrightarrow{a} x' \| y} \qquad \frac{y \xrightarrow{a} y'}{x \| y \xrightarrow{a} x \| y'} \qquad \frac{x \xrightarrow{a} x',\ y \xrightarrow{b} y'}{x \| y \xrightarrow{c} x' \| y'}\, \gamma(a,b) = c$$

Note that in the special case where γ is always undefined, these rules coincide with the rules in (1). Following the method of the previous subsection, we introduce a new operation for each nonempty set of positively tested arguments. Call them $\mathbin{|\!\!\lfloor}$, $\mathbin{\rfloor\!\!|}$, and $|$. $\mathbin{|\!\!\lfloor}$ is defined as in (2); $\mathbin{\rfloor\!\!|}$ is similar. The rules for $|$ are:

$$\frac{x \xrightarrow{a} x',\ y \xrightarrow{b} y'}{x | y \xrightarrow{c} x' \| y'}\, \gamma(a,b) = c$$

The operations $\mathbin{|\!\!\lfloor}$, $\mathbin{\rfloor\!\!|}$ and $|$ are all smooth and distinctive. We've seen the $\mathbin{|\!\!\lfloor}$ equations (they are independent of γ); the $\mathbin{\rfloor\!\!|}$ equations are similar. The $|$ equations, as produced by Theorem 4.9, are:

$$
\begin{array}{rcll}
(x+y)|z & = & x|z+y|z & \\
x|(y+z) & = & x|y+x|z & \\
ax|by & = & c(x\|y) & \text{if } \gamma(a,b)=c \\
ax|by & = & \mathbf{0} & \text{if } \gamma(a,b) \text{ is undefined} \\
\mathbf{0}|x & = & \mathbf{0} & \\
x|\mathbf{0} & = & \mathbf{0} &
\end{array}
$$

Finally, as an instance of (31), we have the equation for $\|$ itself:

$$x\|y = x \mathbin{⌊\!⌊} y + x \mathbin{⌋\!⌋} y + x|y$$

This axiomatization is quite similar to the axiomatization of the parallel operation of ACP. Besides termination issues, our $⌊\!⌊$ is identical to the left merge operation used in the ACP axiomatization of $\|$, and our $|$ is the same as the ACP's communication merge. In ACP there is no right merge operation $⌋\!⌋$, since the role of this operation can be played by the left merge ($x \mathbin{⌋\!⌋} y = y \mathbin{⌊\!⌊} x$).

Turning the crank, we also obtain an axiomatization for the sequencing operation described in (28). Besides the law (30) it consists of:

$$
\begin{array}{rcl|rcl}
(x+y);_1 z & = & x;_1 z + y;_1 z & x;_2 (y+z) & = & x;_2 y + x;_2 z \\
ax;_1 y & = & a(x;y) & \mathbf{0};_2 ay & = & ay \\
 & & & (ax+y);_2 z & = & \mathbf{0} \\
\mathbf{0};_1 x & = & \mathbf{0} & x;_2 \mathbf{0} & = & \mathbf{0}
\end{array}
$$

This is somewhat inferior to the axiomatization of [9], which does not require auxiliary operations. However, finding the latter axiomatization required a lot of thinking whereas the one presented here is produced automatically.

4.2 General GSOS Operations

In this subsection 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 antecedent concerning an argument; having both positive and negative antecedents 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:

$$\frac{x \xrightarrow{a} y_1, \quad x \xrightarrow{b} y_2, \quad x \stackrel{c}{\nrightarrow}}{g(x) \xrightarrow{c} x + y_1}$$

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

$$\frac{x_1 \xrightarrow{a} y_1, \quad x_2 \xrightarrow{b} y_2, \quad x_0 \stackrel{c}{\nrightarrow}}{g'(x_0, x_1, x_2) \xrightarrow{c} x_0 + y_1}$$

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. First, we need a technical lemma, that says that two operations defined by essentially identical rules behave identically.

Lemma 4.12 *Suppose G is a GSOS system and $P = f(\vec{z})$ and $Q = f'(\vec{v})$ are terms over Σ_G with variables that do not occur in R_G. Suppose that there exists a 1-1 correspondence between rules for f and rules for f' such that, whenever a rule ρ for f with source $f(\vec{x})$ is related to a rule ρ' for f' with source $f'(\vec{y})$, we have that, with exception of their sources, $\rho\langle\vec{z}/\vec{x}\rangle$ and $\rho'\langle\vec{v}/\vec{y}\rangle$ are identical.*[6] *Then $\mathsf{BISIM}(G) \models P = Q$.*

Proof: Suppose G' is a disjoint extension of G and σ is a closed $\Sigma_{G'}$-substitution. We have to prove $P\sigma \leftrightarroweq_{G'} Q\sigma$. For this it suffices to show that, for all $a \in \mathsf{Act}$ and $S \in \mathrm{T}(\Sigma_{G'})$,

$$P\sigma \xrightarrow{a}_{G'} S \;\Leftrightarrow\; Q\sigma \xrightarrow{a}_{G'} S.$$

In fact, it is already sufficient to prove the implication '$\Rightarrow$', since the reverse implication is symmetric. So suppose $P\sigma \xrightarrow{a}_{G'} S$. We will prove $Q\sigma \xrightarrow{a}_{G'} S$.

Since $\rightarrow_{G'}$ is supported by G', $R_{G'}$ contains a rule ρ of the form

$$\frac{H}{f(\vec{x}) \xrightarrow{a} T}$$

and there exists a $\Sigma_{G'}$-substitution τ such that

$$\rightarrow_{G'}, \tau \models H \tag{32}$$

$$f(\vec{x})\tau \equiv P\sigma \tag{33}$$

$$T\tau \equiv S \tag{34}$$

Since G' disjointly extends G, we know that ρ is a rule of G. Thus there exists a rule ρ' in R_G (and hence in $R_{G'}$) of the form

$$\frac{H'}{f'(\vec{y}) \xrightarrow{a} T'}$$

such that

$$H\langle\vec{z}/\vec{x}\rangle = H'\langle\vec{v}/\vec{y}\rangle \tag{35}$$

$$T\langle\vec{z}/\vec{x}\rangle \equiv T'\langle\vec{v}/\vec{y}\rangle \tag{36}$$

Let τ' be the $\Sigma_{G'}$-substitution defined by

$$\tau'(w) \equiv \begin{cases} \sigma(w) & \text{if } w \text{ occurs in } \vec{z} \text{ or } \vec{v} \\ \tau(w) & \text{otherwise} \end{cases}$$

[6]This condition relates the variables used as arguments for f and f', which are otherwise unconstrained.

We claim that for all variables w that do not occur in $\vec{z}$ or $\vec{v}$, $\tau' \circ \langle \vec{z}/\vec{x} \rangle(w) \equiv \tau(w)$. Because either w does not occur in $\vec{x}$ and we have $\tau' \circ \langle \vec{z}/\vec{x} \rangle(w) \equiv \tau'(w) \equiv \tau(w)$, or w does occur in $\vec{x}$, in which case the claim follows since

$$\begin{array}{rcll} f(\vec{x})\tau & \equiv & f(\vec{z})\sigma & \text{(by (33))} \\ & \equiv & f(\vec{z})\tau' & \text{(since } \sigma = \tau' \text{ on } \vec{z}\text{)} \\ & \equiv & f(\vec{x})\langle \vec{z}/\vec{x} \rangle \tau' & \end{array}$$

Now we infer

$$\begin{array}{rcl} \mathsf{true} & \Rightarrow & \text{(by (32))} \\ \rightarrow_{G'}, \tau \models H & \Rightarrow & \text{(variables of } H \text{ do not occur in } \vec{z} \text{ or } \vec{v}\text{)} \\ \rightarrow_{G'}, \tau' \circ \langle \vec{z}/\vec{x} \rangle \models H & \Rightarrow & \text{(general property of } \models\text{)} \\ \rightarrow_{G'}, \tau' \models H\langle \vec{z}/\vec{x} \rangle & \Rightarrow & \text{(by (35))} \\ \rightarrow_{G'}, \tau' \models H'\langle \vec{v}/\vec{y} \rangle & \Rightarrow & \text{(general property of } \models\text{)} \\ \rightarrow_{G'}, \tau' \circ \langle \vec{v}/\vec{y} \rangle \models H' & \Rightarrow & \text{(soundness of } \rightarrow_{G'} \text{ for } \rho'\text{)} \\ \rightarrow_{G'}, \tau' \circ \langle \vec{v}/\vec{y} \rangle \models f'(\vec{y}) \xrightarrow{a} T' & \Rightarrow & \text{(by definition of } \models\text{)} \\ f'(\vec{y})\langle \vec{v}/\vec{y} \rangle \tau' \xrightarrow{a}_{G'} T'\langle \vec{v}/\vec{y} \rangle \tau' & \Rightarrow & \text{(by (36))} \\ f'(\vec{v})\tau' \xrightarrow{a}_{G'} T\langle \vec{z}/\vec{x} \rangle \tau' & \Rightarrow & (\tau' = \sigma \text{ on } \vec{v}, \text{ and } \tau' \circ \langle \vec{z}/\vec{x} \rangle = \tau \text{ on variables in } T) \\ f'(\vec{v})\sigma \xrightarrow{a}_{G'} T\tau & \Rightarrow & \text{(by (34))} \\ Q\sigma \xrightarrow{a}_{G'} S & & \end{array}$$

♀

Lemma 4.13 *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*

$$\mathsf{BISIM}(G') \models f(\vec{z}) = f'(\vec{v}) \tag{37}$$

Proof: In order to determine the arity of f' we first quantify the degree in which f is non-smooth. For ρ a general GSOS rule of the form (7), and $1 \leq i \leq l$, the *barb factor* of ρ and i is defined as m_i, if $n_i = 0$ and x_i does not occur in the target, and $m_i + 1$ otherwise. The *barb factor* of f and i, notation $B(f, i)$, is defined as the maximum over all rules ρ for f of the barb factor of ρ and i. Smooth operations are characterized by the property that for all their arguments the barb factor is ≤ 1, and non-smooth operations have a barb factor > 1 for at least one argument. For instance, the priority operation θ has barb factor 2 for its one argument. Let $l' = \sum_{i=1}^{l} B(f, i)$ and let f' be a fresh operation symbol. Then $\Sigma_{G'}$ is defined as the signature that extends Σ_G with an l'-ary operation symbol f'. Let $\vec{w} = w_{11}, \ldots, w_{1B(f,1)}, \ldots, w_{l1}, \ldots, w_{lB(f,l)}$ be a vector of l' different variables. Suppose ρ is a rule for f as in (7) and suppose τ is the substitution that maps each variable w_{ij} to x_i and leaves all the other variables unchanged. Let ρ' be a smooth GSOS rule with source $f'(\vec{w})$,

Input	A GSOS system G.
Output	A GSOS system G' with $G \sqsubseteq G'$ and a finite equational theory T, such that $\mathsf{BISIM}(G') \models T$ and T is head normalizing for all terms of G'.

Step 1. If G does not disjointly extend $G_\dagger$ then add to it a disjoint copy of $G_\dagger$.

Step 2. For each operation f that is non-smooth, apply the construction of Lemma 4.13 to extend the system with a smoothed version of f, f'. Add all the resulting instances of law (37) to $T_\dagger$.

Step 3. For each smooth, non-distinctive operation $f \notin \Sigma_\dagger$ in the resulting system, apply the construction of Lemma 4.10 to generate smooth, distinctive operations $f_1, \ldots, f_n$. The system so-obtained is the G' to be constructed. Add to the equational theory all the resulting instances of law (31).

Step 4. Add to the equational theory obtained in Step 3 the equations given by applying Theorem 4.9 to all the smooth, distinctive operations in $\Sigma_{G'} - \Sigma_\dagger$. The result is the theory T to be constructed.

Figure 3: The algorithm for the proof of Theorem 4.14

and such that, with exception of their sources $\rho'\tau$ and ρ are identical. In fact, such a ρ' can be obtained from ρ by replacing the source of ρ by $f'(\vec{w})$, and by replacing variables x_i in the antecedents and the target by variables w_{ij} in such a way that no barbs arise. This can be done as follows: starting with the positive antecedents, one replaces each occurrence of x_i with a different variable w_{ij}; after that one more w_{ij} is available for the occurrences of x_i in the negative antecedents and the target in case there are such occurrences. Define $R_{G'}$ to be a set of rules that extends R_G with a rule ρ', defined as above, for each rule ρ for f.

Let $\vec{z} = z_1, \ldots, z_l$ be a vector of different variables, all of them not occurring in R_G, and let $\vec{v} = v_{11}, \ldots, v_{1B(f,1)}, \ldots, v_{l1}, \ldots, v_{lB(f,l)}$ be the vector of length l' given by $v_{ij} = z_i$. It is easy to see that, for each pair ρ, ρ' of corresponding rules, $\rho\langle \vec{z}/\vec{x}\rangle$ and $\rho'\langle \vec{v}/\vec{w}\rangle$ are identical, with exception of their sources. Thus we can apply Lemma 4.12 to obtain $\mathsf{BISIM}(G') \models f(\vec{z}) = f'(\vec{v})$, as required. ♧

Theorem 4.14 *Let G be a GSOS system. Then the disjoint extension G' of G and finite equational theory T produced by the algorithm of Figure 3 have the property that* $\mathsf{BISIM}(G') \models$ *T and T is head normalizing for all terms in $\mathrm{T}(\Sigma_{G'})$.*

Proof: A straightforward combination of the previous theory. ♧

The algorithm used in the proof of the above theorem is summarized in Figure 3.

4.3 Example: the Priority Operation

As an example of application of the strategy presented in Sections 4.1-4.2, we will now present an axiomatization of Baeten, Bergstra and Klop's priority operation θ. For the sake of clarity, we recall here that the priority operation assumes a partial ordering relation $>$ on Act and that, for each a, it has a rule

$$\frac{x \xrightarrow{a} y, \quad x \stackrel{b}{\not\rightarrow} \text{ (for all } b > a)}{\theta(x) \xrightarrow{a} \theta(y)}$$

θ is a non-smooth operation with, as previously noted, barb factor 2. We do not have a distributivity law for θ. For instance, if $b > a$, then clearly $b \leftrightarrows \theta(a+b) \underline{\not\leftrightarrow} \theta(a)+\theta(b) \leftrightarrows a+b$. Following Lemma 4.13, we therefore define a smoothed version of θ in the form of a fresh binary operation $\triangle$ with rules (one for each $a \in \mathsf{Act}$):

$$\frac{x \xrightarrow{a} x', \quad y \stackrel{b}{\not\rightarrow} \text{ (for all } b > a)}{x \triangle y \xrightarrow{a} \theta(x')} \tag{38}$$

Note that $\triangle$ is a distinctive smooth operation. The relationships between θ and $\triangle$ are expressed by the following instance of law (37):

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

The smooth distinctive operation $\triangle$ can be axiomatized using the strategy of Theorem 4.9:

$$\begin{aligned}
(x+y)\triangle z &= x\triangle z + y \triangle z && && (39)\\
ax \triangle y &= a.\theta(x) && \text{if } a \text{ is maximal} && (40)\\
ax \triangle (y \dagger \{b \in \mathsf{Act} | b > a\}) &= a.\theta(x) && \text{if } a \text{ is not maximal} && (41)\\
\mathbf{0} \triangle x &= \mathbf{0} && && (42)\\
ax \triangle (by+z) &= \mathbf{0} && \text{if } b > a && (43)
\end{aligned}$$

The axiomatization of the priority operation obtained by applying our general strategy compares rather well with the one given in [7]. 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):

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

which are quite similar to (38). Bergstra [11] gives a finite axiomatization of θ without auxiliary operations. However, in that axiomatization the total number of axioms grows exponentially with the size of the action alphabet.

5 Completeness

For any GSOS system G, the algorithm presented in Figure 3 allows us to generate a disjoint GSOS extension G' with a finite head-normalizing equational theory. In the case of $\|$ and $\|\!\!\_$, 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

$$\omega \xrightarrow{a} \omega \tag{44}$$

The instance of action law (23) for this operation is

$$\omega \quad = \quad a.\omega$$

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

In Section 5.1 we consider the case in which all processes terminate: all terms can be reduced to FINTREE terms and the completeness of FINTREE applies. In Section 5.2 we consider the general case: the reduction to FINTREE does not apply, but a suitable infinitary rule gives completeness.

5.1 Completeness for Well-Founded GSOS Systems

Definition 5.1 *Let G be a GSOS system. A term $P \in \mathrm{T}(\Sigma_G)$ is* (semantically) 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* (semantically) well-founded *iff all terms in $\mathrm{T}(\Sigma_G)$ are well-founded.*

It is immediate to see that the constant ω defined in (44) 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 5.2 *Suppose that G is a GSOS system with $\mathsf{FINTREE} \sqsubseteq G$. Assume that T is an equational theory that extends T_{FINTREE} such that $\mathsf{Bisim}(G) \models T$ and T is head normalizing for all terms in $\mathrm{T}(\Sigma_G)$. Suppose P and Q are well-founded terms in $\mathrm{T}(\Sigma_G)$. Then*

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

Proof: Since $\rightarrow_G$ is finitely branching, we can associate to each well-founded term S a natural number $depth(S)$, denoting the maximum number of consecutive transitions possible from S.

Let now P and Q be well-founded terms in $\mathrm{T}(\Sigma_G)$ such that $\mathsf{Bisim}(G) \models P = Q$. By induction on $depth(P)$ we show that P is provably equal to a FINTREE term. Since T is head normalizing for P, there is a term $P' \equiv \sum a_i P_i$ such that $T \vdash P = P'$. A simple argument gives that $\leftrightarrows$ preserves depth, and thus, since T is sound, $depth(P) = depth(P')$. Since, for all i, $P' \xrightarrow{a_i}_G P_i$, it follows that $depth(P_i) < depth(P') = depth(P)$. Thus we can apply the induction hypothesis to infer that, for each i, there exists a FINTREE term P_i' such that $T \vdash P_i = P_i'$. Let $P'' \equiv \sum a_i P_i'$. Then $T \vdash P = P''$, and the induction step follows.

In a similar way we can find a FINTREE term Q'' such that $T \vdash Q = Q''$. Now $T \vdash P = Q$ follows by completeness of T_{FINTREE}, as in the proof of Lemma 3.2. ♧

In view of Theorems 4.14 and 5.2, the strategy presented in Figure 3 automatically produces finite complete equational axiomatizations for bisimulation equivalence over well-founded GSOS systems. In particular, turning the crank, it can be used to give finite equational characterizations of bisimulation equivalence over the recursion-free, finite-alphabet sublanguages of, *e.g.*, CCS, ACP and MEIJE.

There is one subtlety though: it can happen that we start from a well-founded GSOS system G, but then, after adding the auxiliary operations, end up with a disjoint extension G' that is no longer well-founded. In this case, the axioms produced by our methods are complete for equality on closed Σ_G-terms by Theorem 5.2 (and this is of course the important thing in which we are essentially interested), but need not be complete for equality on closed $\Sigma_{G'}$-terms. As a trivial example, consider the GSOS system with a single unary operation symbol f, and a single rule

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

This system is well-founded for the simple reason that the set of closed terms over its signature is empty. However, after adding the $\mathsf{FINTREE}$ operations to it, we lose well-foundedness as $f(a) \xrightarrow{a} f(a)$. The following lemma identifies $\mathsf{FINTREE}$ as the only potential source of trouble, by showing that all other auxiliary operations added by our method preserve well-foundedness.

Lemma 5.3 *Let G be a well-founded GSOS system with $\mathsf{FINTREE} \sqsubseteq G$. Then the disjoint extension G' of G constructed by the algorithm in Figure 3 is well-founded.*

Proof: We only prove that, for any non-smooth operation f in a well-founded GSOS system G which disjointly extends $\mathsf{FINTREE}$, the system G' that disjointly extends G with a fresh operation f', as in in Lemma 4.13, is well-founded. The fact that the construction in Lemma 4.10 preserves well-foundedness can be shown by a similar, but simpler, argument.

Let f, f' and G' be as above. By structural induction we will show that each term $P \in \mathrm{T}(\Sigma_{G'})$ is well-founded. We proceed by distinguishing two cases, depending on whether the head operation of P is f' or not.

Case 1. Assume that $P \equiv g(P_1, \ldots, P_l)$ for some $g \neq f'$. By the inductive hypothesis, all terms P_i are well-founded. As the transition systems specified by GSOS systems are finite branching and G' itself is a disjoint extension of $\mathsf{FINTREE}$, it is easy to see that for all $i \in \{1, \ldots, l\}$, there exists a $\mathsf{FINTREE}$ term Q_i such that $P_i \leftrightarroweq_{G'} Q_i$. As $\leftrightarroweq_{G'}$ is a congruence, we then have that

$$g(P_1, \ldots, P_l) \leftrightarroweq_{G'} g(Q_1, \ldots, Q_l).$$

Now, $g(Q_1, \ldots, Q_l)$ is a Σ_G-term and well-founded since G is well-founded. Hence, as $\leftrightarroweq$ preserves well-foundedness, $g(P_1, \ldots, P_l)$ is also well-founded.

Case 2. Assume that $P \equiv f'(P_1, \ldots, P_l)$. In order to see that P is well-founded, assume that $P \xrightarrow{a}_{G'} Q$, for some Q. We claim that Q is well-founded. To see that this indeed the case recall that, as $\rightarrow_{G'}$ is supported by G', there exist a rule ρ of the form

$$\frac{H}{f'(x_1 \ldots, x_l) \xrightarrow{a} C[\vec{x}, \vec{y}]}$$

and a substitution σ such that $\rightarrow_{G'}, \sigma \models H$ and

$$\begin{aligned} f'(x_1, \ldots, x_l)\sigma &\equiv P \\ C[\vec{x}, \vec{y}]\sigma &\equiv Q \end{aligned}$$

By the construction of the set of rules for f' given in the proof of Lemma 4.13, we have that $C[\vec{x}, \vec{y}]$ is a Σ_G-context. By the inductive hypothesis, all the P_i are well-founded. Thus, for all i, j, $\sigma(x_i)$ and $\sigma(y_{ij})$ are well-founded. By the same argument as used in Case 1, we can therefore find FINTREE terms Q_i, S_{ij} such that $\sigma(x_i) \leftrightarrow_{G'} Q_i$, and $\sigma(y_{ij}) \leftrightarrow_{G'} S_{ij}$. Again using that $\leftrightarrow_{G'}$ is a congruence, we infer $Q \leftrightarrow_{G'} C[\vec{Q}, \vec{S}]$. Since $C[\vec{Q}, \vec{S}]$ is a Σ_G-term, it is well-founded. Thus, as $\leftrightarrow$ preserves well-foundedness, Q is also well-founded.

Hence we have shown that, for all Q, $P \equiv f'(P_1, \ldots, P_l) \xrightarrow{a}_{G'} Q$ implies that Q is well-founded. This entails that P itself is well-founded. $\clubsuit$

Theorem 5.4 *Suppose that G is a well-founded GSOS system with* FINTREE $\sqsubseteq G$. *Let G' and T denote the disjoint extension of G, and the finite head normalizing equational theory constructed by the algorithm in Figure 3, respectively. Then T is complete for equality in* Bisim(G').

Proof: Since G is well-founded and disjointly extends FINTREE, we know, by Lemma 5.3, that G' is also well-founded. By Theorem 4.14, the equations in T are sound over Bisim(G'). The claim now follows immediately by Theorem 5.2. $\clubsuit$

5.1.1 Syntactic Well-Foundedness

The definition of semantic well-foundedness for a GSOS system G given in Definition 5.1 relies upon properties of the transition relation $\rightarrow_G$. It is in general not decidable whether a GSOS system is semantically well-founded. We will now show that, for an interesting subclass of GSOS systems, there exists a rather natural, effective constraint on the rules of GSOS systems that ensures semantic well-foundedness. (The following definitions and results are slight generalizations of those presented in the report version of this paper [2]).

Definition 5.5 *A GSOS rule of the general form (7) is* linear *if each variable occurs at most once in the target and, for each argument i that is tested positively, at most one of the following conditions holds:*

1. *x_i occurs in the target;*
2. *at most one of the y_{ij}'s does.*

An operation from a GSOS system G is linear *iff all rules for it are linear. Finally, G itself is* linear *iff it only contains linear rules.*

As far as we know, all the operations occurring in the standard process algebras are linear.

Definition 5.6 *A GSOS system G is* syntactically well-founded *iff there exists a function w from operation symbols in Σ_G to natural numbers such that, for each rule $\rho \in R_G$ with principal operation symbol f and target $C[\vec{x},\vec{y}]$ the following conditions hold:*

- *if for every positive antecedent $x_i \stackrel{a_{ij}}{\rightarrow} y_{ij}$ of ρ, the variable x_i occurs in $C[\vec{x},\vec{y}]$ then $W(C[\vec{x},\vec{y}]) < w(f)$, and*
- *$W(C[\vec{x},\vec{y}]) \leq w(f)$ otherwise,*

where $W : \mathbb{T}(\Sigma_G) \rightarrow \mathbf{N}$ is given by

$$\begin{array}{rcl} W(x) & \triangleq & 0 \\ W(f(P_1,\ldots,P_l)) & \triangleq & w(f) + W(P_1) + \cdots + W(P_l). \end{array}$$

Once a weight function w has been defined over its signature, checking that a GSOS system is syntactically well-founded involves a simple verification of the validity of the conditions given in the above definition for each of its rules. For example, the syntactic well-foundedness of the GSOS system $G_\dagger$ can be shown by assigning weight 1 to the action prefixing operations and weight 0 to all the other function symbols. We will now prove that in order to show the semantic well-foundedness of a linear GSOS system, it suffices to establish its syntactic well-foundedness.

Proposition 5.7 *Let G be a syntactically well-founded linear GSOS system. Then G is well-founded.*

Proof: Since G is syntactically well-founded, there exist functions w and W that satisfy the conditions of Definition 5.6. In order to show that G is semantically well-founded, we must prove that for no $P \in \mathrm{T}(\Sigma_G)$ there exist $P_i \in \mathrm{T}(\Sigma_G)$ and $a_i \in \mathsf{Act}$, $i \geq 0$, such that $P_0 \equiv P$ and $P_i \stackrel{a_i}{\rightarrow}_G P_{i+1}$ for all $i \geq 0$. This will follow if we prove that, for all $P, Q \in \mathrm{T}(\Sigma_G)$,

$$P \stackrel{a}{\rightarrow}_G Q \quad \text{implies} \quad W(Q) < W(P). \tag{45}$$

Claim (45) is proved by induction on the structure of P. Assume then that $P \stackrel{a}{\rightarrow}_G Q$. As $\rightarrow_G$ is supported by G, R_G contains a rule ρ of the form

$$\frac{H}{f(x_1,\ldots,x_l) \stackrel{a}{\rightarrow} C[\vec{x},\vec{y}]}$$

and there exists a closed substitution σ such that $\rightarrow_G, \sigma \models H$, $f(x_1,\ldots,x_l)\sigma \equiv P$ and $C[\vec{x},\vec{y}]\sigma \equiv Q$. Since G is linear, we moreover know that ρ is linear. We proceed by distinguishing two cases depending on whether for every positive antecedent $x_i \stackrel{a_{ij}}{\rightarrow} y_{ij}$ of ρ the variable x_i occurs in $C[\vec{x},\vec{y}]$ or not.

Case 1. Assume that for every positive antecedent $x_i \stackrel{a_{ij}}{\rightarrow} y_{ij}$ of ρ the variable x_i occurs in $C[\vec{x},\vec{y}]$. As G is syntactically well-founded, we have that $W(C[\vec{x},\vec{y}]) < w(f)$. By the linearity of ρ, $C[\vec{x},\vec{y}]$ is a context over $\{x_1,\ldots,x_l\}$ in which each x_i occurs at most once. It is then easy to see that

$$\begin{aligned} W(Q) &= W(C[\vec{x},\vec{y}]) + \sum_{\{x \mid x \text{ occurs in } C[\vec{x},\vec{y}]\}} W(\sigma(x)) \\ &< w(f) + \sum_{i=1}^{l} W(\sigma(x_i)) \\ &= W(P). \end{aligned}$$

Case 2. Assume that there exists a positive antecedent $x_h \xrightarrow{a_{hj}} y_{hj}$ of ρ with x_h not occurring in $C[\vec{x},\vec{y}]$. As $\rightarrow_G, \sigma \models H$, we have that, for each positive antecedent $x_i \xrightarrow{a_{ij}} y_{ij}$ of ρ, $\sigma(x_i) \xrightarrow{a_{ij}}_G \sigma(y_{ij})$. By induction hypothesis, this implies for all i,j,

$$W(\sigma(y_{ij})) < W(\sigma(x_i)). \tag{46}$$

Now we derive:

$$\begin{aligned} W(Q) &= W(C[\vec{x},\vec{y}]) + \textstyle\sum_{\{z \mid z \text{ occurs in } C[\vec{x},\vec{y}]\}} W(\sigma(z)) \\ &\qquad \text{(by linearity of } \rho) \\ &< W(C[\vec{x},\vec{y}]) + \textstyle\sum_{i=1}^{l} W(\sigma(x_i)) \\ &\qquad \text{(by (46), using the fact that } x_h \text{ does not occur in } C[\vec{x},\vec{y}]) \\ &\leq w(f) + \textstyle\sum_{i=1}^{l} W(\sigma(x_i)) \\ &\qquad \text{(by syntactic well-foundedness of } G) \\ &= W(P). \end{aligned}$$

♀

Theorem 5.8 *Syntactic well-foundedness is decidable.*

Proof: The problem of deciding whether a given GSOS system is syntactically well-founded, easily reduces to the problem of deciding whether a linear system of diophantine equations has an integer solution: $Ax \leq b$, where A is an integer matrix with m rows and n columns, b a vector in $\mathbf{Z}^m$ and x a vector of n unknowns. For instance, in order to determine whether a GSOS system with rules

$$\frac{x \xrightarrow{a} x'}{f(x) \xrightarrow{a} g(x')} \qquad \frac{}{g(x) \xrightarrow{a} i(x)}$$

$$\frac{}{h(x) \xrightarrow{a} f(f(f(g(x))))} \qquad \frac{x \xrightarrow{a} x'}{h(x) \xrightarrow{a} g(g(x'))}$$

is well-founded, we must find out whether the system

$$\begin{aligned} g &\leq f \\ i &< g \\ 3f + g &< h \\ 2g &\leq h \end{aligned}$$

has a solution in nonnegative integers. But this problem is equivalent to the problem of finding an integer solution to

$$
\begin{array}{rcl}
-f & \leq & 0 \\
-g & \leq & 0 \\
-h & \leq & 0 \\
-i & \leq & 0 \\
-f+g & \leq & 0 \\
-g+i & \leq & -1 \\
3f+g-h & \leq & -1 \\
2g-h & \leq & 0
\end{array}
$$

The general problem of deciding whether a linear system of diophantine equations has a solution is known to be NP-complete [40]. (In this case, $i=0$, $f=g=1$, and $h=5$ works.)

The import of the above proposition is that in some cases one can check whether a linear GSOS system is well-founded by proving its syntactic well-foundedness. We have not been able to find such a syntactic check for non-linear GSOS systems. To illustrate the problems that arise in this case, we will now give an example showing that the result does not hold if we allow rules in whose target more than one of the y_{ij}'s associated with an argument x_i occurs.

Example: Consider the GSOS system obtained by adding to $\mathsf{FINTREE}$ the parallel operation (without synchronization) $\|$ from (1), and a binary operation f with rule

$$\frac{x \xrightarrow{a} x_1, \quad x \xrightarrow{a} x_2}{f(x,y) \xrightarrow{a} f(y, x_1 \| x_2)}$$

The resulting GSOS system is syntactically well-founded. For example, one can assign weight 1 to the action prefixing operations and weight 0 to all the other function symbols. However,

$$f(a\|a, a\|a) \xrightarrow{a} f(a\|a, \mathbf{0}\|a\|a\|\mathbf{0}) \underline{\leftrightarrow} f(a\|a, a\|a)$$

Hence the GSOS system is *not* semantically well-founded.

It is easy to modify the above example to show that copying of arguments in the target, or allowing arguments which are tested positively to occur in the target, would also invalidate Prop. 5.7.

Below we give a simple example of a semantically well-founded linear GSOS system that is not syntactically well-founded.

Example: Consider the trivial GSOS system consisting of constant symbols f and g, a unary symbol h, and axioms

$$f \xrightarrow{a} g \qquad\qquad g \xrightarrow{a} h(f)$$

This system is semantically well-founded. In fact, terms of the form $h(P)$ can perform no transition at all, term g just one transition, and term f two consecutive transitions.

However, there is no weight function over the above signature that satisfies the conditions of Definition 5.6 since the first rule requires $w(g) < w(f)$ and the second rule requires $w(f) \leq w(f) + w(h) < w(g)$. Thus this GSOS system is not syntactically well-founded.

The above example is, however, rather contrived and in many practical applications the notion of weight function provides a simple way of checking the semantic well-foundedness of a linear GSOS system.

5.2 Completeness for General GSOS Systems

It follows from some simple recursion theoretic considerations, to be discussed in detail in Section 5.2.1, that the extension of the completeness result given in Theorem 5.4 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 [14, 10]. By Lemma 2.8, all GSOS processes are finitely branching, so the AIP applies.

We introduce a family of unary operations $\pi_n(\cdot)$, $n \in \mathbf{N}$, with rules (one for each $a \in \mathsf{Act}$):

$$\frac{x \xrightarrow{a} y}{\pi_{n+1}(x) \xrightarrow{a} \pi_n(y)}$$

These operations are known as *projection* operations in the literature on ACP [10]. Intuitively, $\pi_n(P)$ allows P to perform n moves freely, and then stops it. Since the $\pi_n(\cdot)$ operations are smooth and distinctive, we may thus apply the strategy presented in Section 4.1 (or consult [10]) to automatically derive the following equations for them:

$$\begin{aligned}
\pi_n(x+y) &= \pi_n(x) + \pi_n(y) \\
\pi_{n+1}(ax) &= a \,.\, \pi_n(x) \\
\pi_n(\mathbf{0}) &= \mathbf{0} \\
\pi_0(x) &= \mathbf{0}
\end{aligned}$$

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 $a, b \in \mathsf{Act}$, 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 $n \in \mathbf{N}$, $\pi_n(P) \leftrightarrow P/b^n$, where b is some arbitrarily-chosen action and

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

In this formulation, we may rephrase the Approximation Induction Principle as follows:[7]

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

Note that $\cdot/\cdot$ is smooth and distinctive. Applying the strategy presented in Section 4.1, we automatically derive the following equations for it:

$$(x+y)/z \;=\; x/z + y/z \tag{47}$$
$$x/(y+z) \;=\; x/y + x/z \tag{48}$$
$$ax/by \;=\; a(x/y) \tag{49}$$
$$\mathbf{0}/y \;=\; \mathbf{0} \tag{50}$$
$$x/\mathbf{0} \;=\; \mathbf{0} \tag{51}$$

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

Proposition 5.9 $\mathsf{BISIM}\big(\mathsf{FINTREE}_/\big) \models AIP$.

Proof: Standard. The proof closely follows the proof of Theorem 2.5.8 in [10], and essentially uses the result of Lemma 2.8 that all GSOS processes are finitely branching, *i.e. boundedly non-deterministic* in the terminology of [10]. $\triangle$

Theorem 4.14 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 (47)-(51) for the operation $\cdot/\cdot$ can then be used to obtain $\mathsf{FINTREE}$-terms which exhibit the same n-step behaviour as P.

Applying these ideas, we can prove the following lemma.

Lemma 5.10 *Suppose G is a GSOS system. Let G' and T denote the disjoint extension of $G_/$ and the finite equational theory for it given by the algorithm in Figure 3, respectively. Then, for all $P \in \mathrm{T}(\Sigma_{G'})$ and $n \in \mathbf{N}$, there exists a closed* $\mathsf{FINTREE}$*-term Q such that $T \vdash P/b^n = Q$.*

Proof: Note that, by construction, T contains the equations (47)-(51). The proof of the lemma proceeds by a trivial induction on n, using (47)-(51) and the fact that, by Theorem 4.14, T is head normalizing for all terms in $\mathrm{T}(\Sigma_{G'})$. $\triangle$

Collecting the results presented so far, we can now prove the completeness theorem for general GSOS systems.

---

[7] Here we use nonemptiness of the set Act of actions. In fact, there is no fundamental reason to exclude the empty set of actions; it is just convenient to do so. If Act $= \varnothing$ then any GSOS system is trivially syntactically well-founded as it is without rules. Thus the results of the previous subsection give us a complete proof system, albeit an overly complex one since the single law $x = y$ already does the job in this degenerate case.

Theorem 5.11 *Suppose G is a GSOS system. Let G' and T denote the disjoint extension of $G_/$ and the finite equational theory for it given by the algorithm in Figure 3, respectively. Then T and AIP together are complete for equality in* $\mathsf{Bisim}(G')$.

Proof: We will prove that, for all $P, Q \in \mathrm{T}(\Sigma_{G'})$,

$$\mathsf{Bisim}(G') \models P = Q \;\Leftrightarrow\; T, \mathrm{AIP} \vdash P = Q. \tag{52}$$

Implication '$\Leftarrow$' (soundness) in (52) follows immediately by Theorem 4.14 and Prop. 5.9.

For the '$\Rightarrow$' implication (completeness) in (52), assume that $\mathsf{Bisim}(G') \models P = Q$. This implies that $\mathsf{Bisim}(G') \models P/b^n = Q/b^n$, for all $n \in \mathbf{N}$. It is now sufficient to show that, for all $n \in \mathbf{N}$, $T \vdash P/b^n = Q/b^n$ as the claim will then follow by AIP. Fix $n \in \mathbf{N}$. By Lemma 5.10, there exist closed $\mathsf{FINTREE}$-terms P' and Q' such that $T \vdash P/b^n = P'$ and $T \vdash Q/b^n = Q'$. By the soundness of the axioms and using the fact that $\leftrightarrows_{G'}$ is an equivalence, we obtain $\mathsf{Bisim}(G') \models P' = Q'$. By construction, G' is a disjoint extension of $G_\dagger$, and thus of $\mathsf{FINTREE}$. Hence we infer $\mathsf{Bisim}(\mathsf{FINTREE}) \models P' = Q'$. By the completeness result for $\mathsf{FINTREE}$ (Lemma 3.1) and by the fact that $T_{\mathsf{FINTREE}} \subseteq T$, we obtain that $T \vdash P' = Q'$. Hence, $T \vdash P/b^n = Q/b^n$. Since n was chosen arbitrarily, this statement holds for all $n \in \mathbf{N}$. ♀

5.2.1 Why an Induction Principle is Needed

It is well-known that provable equality from a finite set of equations is recursively enumerable, and in fact Σ_1 complete. In order to show that, for general GSOS systems, bisimulation equivalence cannot be axiomatized completely by a finite set of equations, it therefore suffices to show that bisimulation equivalence is not r.e. Let G be any GSOS system which disjointly extends $\mathsf{FINTREE}$. By Prop. 5.9, we have that, for all closed Σ_G-terms P, Q,

$$P \leftrightarrows_G Q \qquad \Leftrightarrow \qquad \forall n : P/b^n \leftrightarrows_{G_/} Q/b^n$$

for some, arbitrarily chosen, $b \in \mathsf{Act}$. Since, for each n, the transition graphs for P/b^n and Q/b^n are finite and can be effectively computed (cf. Lemma 2.8), and since bisimulation on finite graphs is decidable, it follows that the predicate $R(n, P, Q) \triangleq P/b^n \leftrightarrows_{G_/} Q/b^n$ is recursive. Thus, by definition of the arithmetic hierarchy, $\leftrightarrows_G$ is Π_1. Π_1 predicates are r.e. iff they are recursive, so in order to show that bisimulation equivalence for general GSOS systems is not r.e., it suffices to prove that an oracle for bisimulation equivalence would allow us to solve the halting problem.

We exhibit a GSOS system with, for each n, a term $\mathrm{U2CM}_n$ which behaves like a universal 2-counter machine on input n. Then $\mathrm{U2CM}_n \leftrightarrows \omega$ (where ω is given by $\omega \xrightarrow{a} \omega$) iff the 2-counter machine diverges, a nonrecursive problem.

Suppose that the 2-counter machine has code of the form:

l_1: `if I=0 goto` l_5
l_2: `inc I`
l_3: `dec J`
l_4: `goto` l_7
$\vdots$
l_k: `halt`

We build a GSOS system on top of $\mathsf{FINTREE}$ and the system with constant ω and rule (44). Besides action a, we assume the presence of actions *succ* and *zero*, which are successor and zero for the counters: we code a natural number n by the $\mathsf{FINTREE}$ term $succ^n.zero$. Thus, if P codes n and $P \overset{succ}{\rightarrow} P'$, then $n > 0$ and P' codes $n-1$. Also, if P codes n, then $P \overset{zero}{\rightarrow}$ iff $n = 0$.[8] The a is a "tick" action which the process emits as it computes. We also need binary function symbols $l_1, \ldots, l_k$ to code the states of the 2-counter machine: $l_i(succ^m.zero, succ^n.zero)$ codes the machine at label l_i, with the two counters $\mathtt{I} = m$ and $\mathtt{J} = n$.

If the i-th instruction is of the form `if I=0 goto` l_j, then we have the rules:

$$\frac{x \overset{zero}{\rightarrow} x'}{l_i(x,y) \overset{a}{\rightarrow} l_j(zero, y)} \qquad \frac{x \overset{succ}{\rightarrow} x'}{l_i(x,y) \overset{a}{\rightarrow} l_{i+1}(succ.x', y)}$$

If the i-th instruction is `inc I` (resp. `goto` l_j):

$$\frac{}{l_i(x,y) \overset{a}{\rightarrow} l_{i+1}(succ.x, y)} \qquad \frac{}{l_i(x,y) \overset{a}{\rightarrow} l_j(x, y)}$$

If the i-th instruction is `dec I`:

$$\frac{x \overset{zero}{\rightarrow} x'}{l_i(x,y) \overset{a}{\rightarrow} l_{i+1}(zero, y)} \qquad \frac{x \overset{succ}{\rightarrow} x'}{l_i(x,y) \overset{a}{\rightarrow} l_{i+1}(x', y)}$$

Commands which deal with the other counter `J` are similar. There are no rules for labels of `halt` commands, as these cause the automaton to halt. We define $\mathrm{U2CM}_n = l_1(succ^n.zero, zero)$; then $\mathrm{U2CM}_n \leftrightarroweq \omega$ iff the universal machine diverges on input n. Thus, testing bisimulation is co-r.e. hard; it is easily seen to be co-r.e. complete.

This system could scarcely be simpler in format; it is smooth and distinctive, and indeed is De Simone format. Undecidability of bisimulation is not due to eccentric language constructs permitted by copying and negative tests; it is a fundamental computational phenomenon.

Even though an induction principle like AIP is needed to deal with general non-well founded GSOS systems, there are several examples in the literature of non-well-founded GSOS systems for which finite equational axiomatizations exist. An example is the *delay* operation δ of SCCS [34], which is defined by the rules

$$\frac{}{\delta x \overset{1}{\rightarrow} \delta x} \qquad \frac{x \overset{a}{\rightarrow} y}{\delta x \overset{a}{\rightarrow} y}$$

Hennessy [28] axiomatized this operation relative to a kind of bisimulation preorder. If G_δ is the GSOS system obtained by adding the above rules to $\mathsf{FINTREE}$, then equality in $\mathsf{Bisim}(G_\delta)$ can be axiomatized by simply adding to T_{FINTREE} the two axioms

$$\begin{aligned} \delta x &= 1.\delta x + x \\ \delta\delta x &= \delta x \end{aligned}$$

[8] The interpretation of *succ* as successor and *zero* as zero is clearly not correct for arbitrary terms – *e.g.*, *succ* + *zero* looks both positive and zero – but it works for our coding of the naturals.

Another example is the complete equational axiomatization for an I/O automata calculus due to De Nicola and Segala [22]. An interesting open problem is to find a class of non-well-founded GSOS languages for which finite complete equational axiomatizations exist.

6 An Alternative Strategy

Of course, several variations are possible on our method to obtain complete axiom systems for GSOS languages. In this section we will study one such variation, which is interesting because it does not use the † operations. We used the † operations in our axiomatizations because they allow for a simple and one-to-one correspondence between SOS rules and the action laws (*cf.* Lemma 4.5). But since one generally likes to have as few auxiliary operations as possible, a natural question to ask is whether the †'s are really needed. In this section we will see that the answer is no.

To illustrate our basic idea of how to eliminate the †'s from axiomatizations, we consider of the θ and $\triangle$ operations from Section 4.3. The axiomatization of the $\triangle$ operation generated by our methods, and consequently the one of the priority operation θ, can be further simplified by replacing the action laws (40) and (41) by

$$\begin{aligned} ax\triangle(by+z) &= ax\triangle z \qquad \text{if } \neg(b>a) & (53)\\ ax\triangle\mathbf{0} &= a.\theta(x) & (54) \end{aligned}$$

Equation (54) can be viewed the instance of action laws (40) and (41), obtained by setting y to $\mathbf{0}$, and, in the case of (41), using law (16) to replace $\mathbf{0}\dagger B$ by $\mathbf{0}$. Equation (53) is what we call a "peeling" law. It allows for the stepwise reduction of the negatively tested argument to a form in which either action law (54) or inaction law (43) can be applied. It is an easy exercise to show that also this new set of laws is head normalizing for closed terms built from θ, $\triangle$ and the $\mathsf{FINTREE}$ operations.

We will now generalize the idea underlying laws (53) and (54) and show how it can be incorporated in our strategy. To this end, we first give general formulations of (53) and (54).

Lemma 6.1 *Suppose f is a distinctive smooth operation of a disjoint extension G of* $\mathsf{FINTREE}$*, with a rule ρ of the form*

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

Then the following statements hold:

- *Let $k\in K$ be such that x_k does not occur in $C[\vec{x},\vec{y}]$, and $b\notin\{b_{kj}|1\leq j\leq n_k\}$. Take*

$$P_i \equiv \begin{cases} a_i y_i & i\in I\\ bx'_k + x''_k & i=k\\ x_i & i\in K\wedge i\neq k\end{cases} \quad \text{and} \quad Q_i \equiv \begin{cases} a_i y_i & i\in I\\ x''_k & i=k\\ x_i & i\in K\wedge i\neq k\end{cases}$$

  Then

$$\mathsf{BISIM}(G)\models f(\vec{P}) = f(\vec{Q}) \qquad (55)$$

- *Let*

$$P_i \equiv \begin{cases} a_i y_i & i \in I \\ \mathbf{0} & i \in K \wedge n_i > 0 \\ x_i & otherwise \end{cases}$$

Then

$$\mathsf{BISIM}(G) \models f(\vec{P}) = c.C[\vec{P}, \vec{y}] \tag{56}$$

Proof: The validity of equation (56) follows immediately from Lemma 4.5. To see that (55) holds, it is sufficient to note that, for any closed substitution σ, rule ρ fires from $f(\vec{P})\sigma$ iff it fires from $f(\vec{Q})\sigma$. By the distinctiveness of f, ρ is the only rule that can possibly fire from these terms. Moreover, as x_k does not occur in $C[\vec{x}, \vec{y}]$, it is easy to check that if ρ fires, then the targets of the matching transitions from $f(\vec{P})\sigma$ and $f(\vec{Q})\sigma$ are syntactically equal. $\clubsuit$

It is easy to see that (53) and (54) can be obtained as instances of (55) and (56), respectively.

The combination of peeling laws (55), instantiated action laws (56), distributivity laws (19), and inaction laws (24), gives a theory that is head normalizing for terms built from distinctive smooth operations that are *discarding.*

Definition 6.2 *A smooth GSOS rule of the form (17) is* discarding *if for no argument i that is tested negatively, x_i occurs in the target. A smooth operation is* discarding *if all the rules for this operation are.*

For example, the $\triangle$ operation is discarding, and so are all the operations whose rules have only positive antecedents. A distinctive smooth operation that is not discarding is the unary operation f given by the rule

$$\frac{x \stackrel{a}{\not\rightarrow}}{f(x) \stackrel{a}{\rightarrow} x} \tag{57}$$

It is easy to see that the peeling law (55) does not hold in the system obtained by extending $\mathsf{FINTREE}$ with this f. One has, for example, that, $a.(b+c) \leftrightarrow f(b+c) \not\leftrightarrow f(c) \leftrightarrow a.c.$

The following theorem formally states the head normalization property, and serves as an alternative for Theorem 4.9 in the approach of this section.

Theorem 6.3 *Suppose G is a GSOS system with $\mathsf{FINTREE} \sqsubseteq G$. Let $\Sigma \subseteq \Sigma_G - \Sigma_{\mathsf{FINTREE}}$ be a collection of distinctive, smooth and discarding operations of G. Let T be the finite equational theory that extends T_{FINTREE} with the following axioms, for each operation f from Σ,*

1. *for each argument i of f that is tested positively, a distributivity axiom (19),*
2. *for each rule for f of the form (17), for each argument i that is tested negatively, and for each action $b \notin \{b_{ij} | 1 \leq j \leq n_i\}$, a peeling law (55),*
3. *for each rule for f, an action law (56),*
4. *all the inaction laws (24) for f.*

Then $\mathsf{BISIM}(G) \models T$*, and* T *is head normalizing for all terms in* $\mathrm{T}(\Sigma \cup \Sigma_{\mathsf{FINTREE}})$.

Proof: Very similar to the proof of Theorem 4.9. The only difference arises in the analysis of Case 3.2.2, where the relevant instances of laws (55) and (56) are now used. The details are left to the reader.

The next lemma shows how to handle general smooth and discarding operations.

Lemma 6.4 *Suppose* G *is a GSOS system with* $\mathsf{FINTREE} \sqsubseteq G$*, and suppose* f *is an* l*-ary smooth and discarding operation of* G*. Then there exists a disjoint extension* G' *of* G *with* l*-ary distinctive, smooth and discarding operations* $f_1, \ldots, f_n$ *such that, for all* $\vec{x}$ *of length* l*,*

$$\mathsf{BISIM}(G') \models f(\vec{x}) = f_1(\vec{x}) + \cdots + f_n(\vec{x}) \tag{58}$$

Proof: Identical to the proof of Lemma 4.10, since the construction of that lemma guarantees that if f is discarding, the new operations f_i are discarding as well.

The final lemma of this section gives head normalization for general GSOS operations.

Lemma 6.5 *Suppose* G *is a GSOS system containing an operation* f *with arity* l *that is not both smooth and discarding. Then there exists a disjoint extension* G' *of* G *with a smooth and discarding 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*

$$\mathsf{BISIM}(G') \models f(\vec{z}) = f'(\vec{v}) \tag{59}$$

Proof: The proof closely follows the proof of Lemma 4.13. The only difference is that we make an additional copy in f' of each argument of f that is both tested negatively and occurs in the target. The rules for operation f' then use one copy of such an argument for the negative testing, and another inside the target.

For example, the discarding version of the operation f given by the rule (57) is the binary operation f' with rule

$$\frac{x \stackrel{a}{\not\rightarrow}}{f'(x,y) \stackrel{a}{\rightarrow} y}$$

The relationship between the two operations is then given by the instance of (59) for f, *i.e.*, $f(x) = f'(x,x)$.

The reader will have no trouble in convincing himself/herself that, by combination of the above lemmas, we obtain an alternative way to build complete axiomatizations. For the sake of clarity, we give in Figure 4 the steps of the revised strategy to obtain a complete axiom system for an arbitrary GSOS system G.

The original algorithm of Figure 3 has the advantage that it produces axioms (the action laws) that are in a simple and direct correspondence with the GSOS rules from the input. However, it may be argued that the new strategy of Figure 4 improves upon this algorithm as it does not use the † operations. Also, preliminary results of [21] indicate that the axiomatizations produced by the alternative strategy have better term rewriting properties.

Step 1. Add to G a disjoint copy of FINTREE, if it is not the case that FINTREE $\sqsubseteq G$.

Step 2. For each operation f that is not both smooth and discarding, apply the construction of Lemma 6.5 to extend the system with a smooth and discarding version f', in such a way that law (59) holds.

Step 3. For each smooth, discarding and non-distinctive operation $f \notin \Sigma_{\mathsf{FINTREE}}$ in the resulting system, apply the construction of Lemma 6.4 to generate smooth, discarding and distinctive operations $f_1, \ldots, f_n$ in such a way that law (58) is valid.

Step 4. Apply Lemmas 4.3, 4.6 and 6.1 to obtain head-normalization for the distinctive, discarding and smooth operations in the GSOS system obtained in Step 3.

Step 5. Try to prove that the GSOS system obtained in Step 3 is well-founded (for instance, by establishing syntactic well-foundedness). If this succeeds then stop, otherwise add AIP to the axiomatization.

Figure 4: The revised strategy

7 Further Research

7.1 Other Equivalences

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. We have chosen bisimulation equivalence because it is widely viewed to be the finest acceptable process equivalence.[9] 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 equational part of our method extends trivially: one simply adds equations capturing the equivalence as a quotient of bisimulation. Many of the weaker equivalences are best described as the kernels of preorders; the methods of this paper extend straightforwardly to preorders as well.

However, in general conditional equations are not preserved by homomorphisms, and one has to be careful in the case of AIP. This proof rule does hold for most 'strong' equivalences on finitely branching labelled transition systems occurring in the literature, in particular for the 11 equivalences on this domain that are discussed in [24]. AIP is not valid though for most of the so-called 'weak' process equivalences, which declare a special internal action τ to be invisible. A weaker rule AIP$^-$ holds in several of these cases (see [23, 10]) but unless additional proof principles like Koomen's Fair Abstraction Rule [10] are added, one will often lose completeness in the case of non-well-founded operations.

Strong bisimulation is a congruence for all GSOS operations, but it is well-known that the

[9] 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*.

same is not true in general for the coarser equivalences. By now it is quite well understood which equivalences are congruences for which GSOS systems (see [19, 27, 50, 17, 18]), but some work remains to be done to extend these results to the new formats presented in this paper. In [19], *ready simulation* is introduced as an equivalence that is a congruence for all operations defined via GSOS rules. In addition a GSOS language is presented for which ready simulation is in fact fully abstract (relative to completed trace equivalence). It is not too hard to define a language that uses only the smooth GSOS rules introduced in this paper such that ready simulation is fully abstract. [17] gives subclasses of GSOS languages which respect ready equivalence [ready trace equivalence], and for which our equational construction stays within the scope of the class.

A subtle problem that may arise in our approach is that, in order to axiomatize the model induced by a given GSOS system G and behavioral equivalence $\approx$ that is a congruence for the operations of G, we may have to introduce auxiliary operations for which $\approx$ is no longer a congruence. We are not aware of any occurrence of this situation in the case of 'strong' equivalences, but there are several examples with 'weak' equivalences. For instance, weak failure equivalence is a congruence for all the operations of the CSP language, but not for the auxiliary operations $+$, $\parallel\!\!\!\_$, $\_\!\!\!\parallel$ and $\mid$ generated by our method. The problem is analyzed in [25], where also a solution is proposed: the use of *module logic* instead of the standard (conditional) equational logic. Reference [18] gives a subclass of GSOS languages which respect various weak bisimulations; there it is shown how the methods of this paper apply to rooted branching bisimulation, and to a large extent to rooted weak bisimulation (Milner's "observational congruence" [35]).

7.2 Other Formats

In this paper we assume that the set Act of actions is finite. Consequently our methods can not be applied directly to the full versions of calculi like CCS [35] and MEIJE [5], since these calculi postulate an infinite action set. If the action set is infinite it is natural to use, both in the inference rules and in the equations, variables ranging over actions instead of just actions. A GSOS system will then contain for instance a finite number of rules:

$$\frac{\bigcup_{i=1}^{l}\left\{x_i \xrightarrow{\alpha_{ij}} y_{ij} | 1 \leq j \leq m_i\right\} \ \cup \ \bigcup_{i=1}^{l}\left\{x_i \stackrel{\beta_{ik}}{\not\rightarrow} | 1 \leq k \leq n_i\right\}}{f(x_1, \ldots, x_l) \xrightarrow{\gamma} C[\vec{x}, \vec{y}]} \ Pr(\vec{\alpha}, \vec{\beta}, \gamma)$$

where the $\alpha_{ij}, \beta_{ik}, \gamma$ are now *variables* ranging over Act and $Pr(\vec{\alpha}, \vec{\beta}, \gamma)$ is a predicate on actions. We expect that our approach will extend to this setting with appropriate extensions, at least under some restriction on the allowed predicates on actions.

It is not clear to us how to handle more general rule formats with *lookahead* and function symbols in the antecedents, as the tyft/tyxt format of [27] and the ntyft/ntyxt format of [20].

7.3 The Other Direction

Now that we have shown that it is possible to generate complete axiom systems for arbitrary SOS specifications in GSOS format, a natural question to ask is whether it is possible to go in the other direction as well, that is, to generate an SOS-style semantics starting from an arbitrary equational process theory. A problem with this question is that, unlike in the case of SOS, no structural properties for equations have been proposed in the literature on process algebras. Thus any single-sorted equational theory would qualify as a 'process' theory, which does not sound very meaningful. In the literature on process algebras equational theories often define operations in terms of the FINTREE combinators, via an induction scheme. In these cases it is, almost without exception, easy to find an SOS semantics. In particular this is always possible in the case of the axiomatizations generated by our method, since the original rules can be retrieved from the action laws.

He Jifeng and Hoare [32] derive the standard SOS semantics of a large subset of CSP and a notational variant of Dijkstra's language from the algebraic semantics of these languages. Translated to the setting of this paper, their idea is to define a transition relation in terms of algebraic laws by

$$
\begin{aligned}
P \xrightarrow{a} R \quad &\Leftrightarrow \quad T \vdash P = aR + P, \\
P \stackrel{a}{\nrightarrow} \quad &\Leftrightarrow \quad T \vdash P = P \dagger \{a\}.
\end{aligned}
$$

With this definition one can then try to derive SOS rules. For instance, in the case of the ACP fragment of Example 4.1.6 we can derive the first rule for parallel as follows. Suppose $P \xrightarrow{a} R$. Then $\vdash P = aR + P$. Hence

$$\vdash \;\; P \| Q = P \mathbin{⫦} Q + P | Q = (aR + P) \mathbin{⫦} Q + P | Q = aR \mathbin{⫦} Q + P \mathbin{⫦} Q + P | Q = a(R \| Q) + P \| Q,$$

and thus $P \| Q \xrightarrow{a} R \| Q$.

In fact, using the above strategy it is possible to derive all the rules of an arbitrary GSOS system from a slightly strengthened version of the theory generated by the algorithm we presented in Section 4. This is obtained by adding the valid equations

$$
\begin{aligned}
x \dagger \varnothing \;\; &= \;\; x \\
(x \dagger B) \dagger C \;\; &= \;\; x \dagger (B \cup C)
\end{aligned}
$$

to the theory $T_{\dagger}$.

7.4 ω-Completeness

In general, the axiom systems that we end up with are complete for closed terms but not for open terms. For instance, if we take our axiomatization of the ACP parallel composition, then we can prove from these axioms the equations $P \| Q = Q \| P$ and $P \| (Q \| R) = (P \| Q) \| R$ for all closed terms P, Q, R, but *not* the valid laws $x \| y = y \| x$ and $x \| (y \| z) = (x \| y) \| z$. The proof of this fact is both simple and instructive. Consider the algebra $\mathcal{A}$ with the set of positive natural numbers as domain, and mapping each operation symbol f to an operation $f_{\mathcal{A}}$ defined by

$$
\begin{aligned}
\mathbf{0}_{\mathcal{A}} \;\; &\triangleq \;\; 1 \\
a_{\mathcal{A}}(m) \;\; &\triangleq \;\; 1
\end{aligned}
$$

$$
\begin{aligned}
m +_{\mathcal{A}} n &\triangleq \max(m, n) \\
m \|_{\mathcal{A}} n &\triangleq m^n \\
m \mathbin{\|\!\_}_{\mathcal{A}}\, n &\triangleq m^n \\
m \mathbin{\_\!\|}_{\mathcal{A}}\, n &\triangleq 1 \\
m |_{\mathcal{A}} n &\triangleq 1
\end{aligned}
$$

One can easily check that all the axioms that we give for the ACP fragment are valid in this model. However, since exponentiation is neither commutative nor associative, the laws $x \| y = y \| x$ and $x \| (y \| z) = (x \| y) \| z$ are not valid in $\mathcal{A}$. This means in particular that they cannot be proven from the other laws.[10]

An equivalent way of phrasing the problem is that our axiomatizations are complete, but not *ω-complete*. We recall that an equational theory T over a signature Σ is *ω-complete* if for all open terms P, Q,

$$T \vdash P = Q \quad \Leftrightarrow \quad T \vdash P\sigma = Q\sigma \text{ for all closed substitutions } \sigma.$$

Although some results have been obtained recently ([37, 26]), it appears to be extremely difficult to obtain ω-complete axiomatizations for nontrivial process algebras. For instance, the problem of finding a finite ω-complete axiomatization for finite CCS (possibly involving auxiliary operations) is, as far as we know, still wide open. Even if one is willing to live with axiomatizations that are not ω-complete, one can still argue that laws like the commutativity and associativity of parallel composition express fundamental properties of this operation, and that therefore they should at least be derivable from the other axioms. The question arises whether there exists a general, systematic way of finding more powerful axiomatizations. In [47], De Simone introduced the notion of *FH-bisimulation* as a viable, though not complete, proof technique for open equations. Using FH-bisimulations it is not too hard to come up with an algorithm that can at least *check* the validity of the above commutativity and associativity laws, as well as many other important open equations. FH-bisimulations are defined in [47] for a format of linear, smooth rules with positive antecedents only, but we think that it will be possible to generalize this to general GSOS rules.

7.5 Improved Equations

The equations generated by these methods are by no means optimal for all operations. The precise definition of optimality is not clear, but some axiomatizations certainly aren't; *e.g.*, there are axiomatizations for sequencing which do not involve generating auxiliary operations. It is presumably possible to generate better equations by various measures, for some subclasses of operations at least. For example, our algorithm performs awkwardly on constants. Consider a constant κ defined by

$$\kappa \xrightarrow{a} a.\kappa \qquad\qquad \kappa \xrightarrow{b} b.\kappa. \tag{60}$$

[10] Because in ACP one writes $y \mathbin{\|\!\_}\, x$ instead of $x \mathbin{\_\!\|}\, y$, the algebra $\mathcal{A}$ is not a model of the basic ACP axioms for $\|$. However, it follows from an example in [37] that also the ACP axioms are not complete for open terms.

κ is not distinctive; indeed, a constant is distinctive iff it has exactly one rule. Our algorithm generates two auxiliary constants

$$\kappa_a \xrightarrow{a} a.\kappa \qquad\qquad \kappa_b \xrightarrow{b} b.\kappa \tag{61}$$

and axioms:

$$\begin{aligned}\kappa &= \kappa_a + \kappa_b \\ \kappa_a &= a.a.\kappa \\ \kappa_b &= b.b.\kappa\end{aligned}$$

However, there is a much better axiomatization:

$$\kappa = a.a.\kappa + b.b.\kappa. \tag{62}$$

While this can be derived easily from the generated axioms, it would be better to construct it directly and skip κ_a and κ_b. The same trick applies to all constants. There are doubtless any number of other clever tricks axiomatizing other kinds of operations. (See, *e.g.*, [18, 1] for a discussion of alternative axiomatizations of operators specified by rules without negative antecedents).

The issue of improving the equations, and in particular reducing the number of auxiliary operations in equational characterizations of process equivalences, is of more than purely theoretical interest. General tools which support algebraic style proofs of process equalities have recently been developed, *e.g.*, Huimin Lin's PAM [33]. We envisage that suitably optimized versions of our methods might be the theoretical foundations for the development of front-ends to these tools, which, when fed with a GSOS system, would generate tractable equational axiomatizations of bisimulation equivalence. Generating equations tuned to automatic proof — *e.g.*, with fewer auxiliary constants — will thus be of value in verifying programs.

8 Acknowledgements

We would like to thank Ilaria Castellani and Sam Weber for comments on this paper, Robert de Simone and Loïc Pottier for telling us about [40], and the participants to the Process Algebra Meetings at CWI for the reference to [45]. The anonymous referees provided useful comments that led to improvements in the paper. In particular, one of them suggested the slightly more general versions of Definitions 5.5 and 5.6 that appear in this paper.

References

[1] L. Aceto. Deriving complete inference systems for a class of GSOS languages generating regular behaviours. Report IR 93–2009, Institute for Electronic Systems, Department of Mathematics and Computer Science, Aalborg University, Aalborg, November 1993.

[2] L. Aceto, B. Bloom, and F.W. Vaandrager. Turning SOS rules into equations. Report CS-R9218, CWI, Amsterdam, June 1992.

[3] P. America, J.W. de Bakker, J.N. Kok, and J.J.M.M. Rutten. Operational semantics of a parallel object-oriented language. In *Conference Record of the* 13^{th} *Annual ACM Symposium on Principles of Programming Languages,* St. Petersburg, Florida, pages 194–208, 1986.

[4] E. Astesiano, A. Giovini, F. Mazzanti, G. Reggio, and E. Zucca. The Ada challenge for new formal semantic techniques. In P.L. Wallis, editor, *Ada: Managing the Transition – Proceedings of the 1986 Ada-Europe International Conference*, Cambridge, U.K., 1986. Cambridge University Press.

[5] D. Austry and G. Boudol. Algèbre de processus et synchronisations. *Theoretical Computer Science*, 30(1):91–131, 1984.

[6] J.C.M. Baeten and J.A. Bergstra. A survey of axiom systems for process algebras. Report P9111, University of Amsterdam, Amsterdam, October 1991.

[7] 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.

[8] J.C.M. Baeten and J.W. Klop, editors. *Proceedings CONCUR 90,* Amsterdam, volume 458 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[9] J.C.M. Baeten and F.W. Vaandrager. An algebra for process creation. *Acta Informatica*, 29(4):303–334, 1992.

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

[11] J.A. Bergstra. Put and get, primitives for synchronous unreliable message passing. Logic Group Preprint Series Nr. 3, CIF, State University of Utrecht, 1985.

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

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

[14] J.A. Bergstra and J.W. Klop. Verification of an alternating bit protocol by means of process algebra. In W. Bibel and K.P. Jantke, editors, *Math. Methods of Spec. and Synthesis of Software Systems '85, Math. Research 31*, pages 9–23, Berlin, 1986. Akademie-Verlag. First appeared as: Report CS-R8404, CWI, Amsterdam, 1984.

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

[16] 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.

[17] B. Bloom. Ready, set, go: Structural operational semantics for linear-time process algebras. Technical Report TR 93-1372, Cornell, August 1993.

[18] B. Bloom. Structural operational semantics for weak bisimulations. Technical Report TR 93-1373, Cornell, August 1993.

[19] B. Bloom, S. Istrail, and A.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. Full version available as Technical Report 90-1150, Department of Computer Science, Cornell University, Ithaca, New York, August 1990. To appear in the *Journal of the ACM.*

[20] 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, volume 510 of *Lecture Notes in Computer Science*, pages 481–494. Springer-Verlag, 1991. Full version available as Report CS-R9054, CWI, Amsterdam, 1990.

[21] D.J.B. Bosscher. Term rewriting properties of SOS axiomatizations, 1993. To appear as CWI technical report. An extended abstract will appear in the *Proceedings of TACS '94*.

[22] R. De Nicola and R. Segala. A process algebraic view of input/output automata. Rapporto di Ricerca SI-92/07, Dipartimento di Scienze dell'Informazione, Università degli Studi di Roma "La Sapienza", September 1992.

[23] R.J. van Glabbeek. Bounded nondeterminism and the approximation induction principle in process algebra. In F.J. Brandenburg, G. Vidal-Naquet, and M. Wirsing, editors, *Proceedings STACS 87*, volume 247 of *Lecture Notes in Computer Science*, pages 336–347. Springer-Verlag, 1987.

[24] R.J. van Glabbeek. The linear time – branching time spectrum. In Baeten and Klop [8], pages 278–297.

[25] R.J. van Glabbeek and F.W. Vaandrager. Modular specification of process algebras. *Theoretical Computer Science*, 113(2):293–348, 1993.

[26] J.F. Groote. A new strategy for proving ω–completeness with applications in process algebra. In Baeten and Klop [8], pages 314–331.

[27] J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, October 1992.

[28] M. Hennessy. A term model for synchronous processes. *Information and Control*, 51(1):58–75, 1981.

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

[30] M. Hennessy and R. Milner. On observing nondeterminism and concurrency. In J.W. de Bakker and J. van Leeuwen, editors, *Proceedings* 7^{th} *ICALP,* Noorwijkerhout, volume 85 of *Lecture Notes in Computer Science*, pages 299–309. Springer-Verlag, July 1980. This is a preliminary version of: Algebraic laws for nondeterminism and concurrency. *JACM*, 32(1):137–161, 1985.

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

[32] He Jifeng and C.A.R. Hoare. From algebra to operational semantics. *Information Processing Letters*, 45(2):75–80, 1993.

[33] Huimin Lin. PAM: A Process Algebra Manipulator (Version 1.0). Computer Science Report 4/93, University of Sussex, February 1993.

[34] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

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

[36] R. Milner. Functions as processes. In Paterson [42], pages 167–180.

[37] F. Moller. *Axioms for concurrency.* PhD thesis, Report CST-59-89, Department of Computer Science, University of Edinburgh, 1989.

[38] F. Moller. The importance of the left merge operator in process algebras. In Paterson [42], pages 752–764.

[39] S. Owicki and D. Gries. An axiomatic proof technique for parallel programs. *Acta Informatica*, 6(4):319–340, 1976.

[40] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization – Algorithms and Complexity.* Prentice-Hall International, Englewood Cliffs, 1982.

[41] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, 5^{th} *GI Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.

[42] M. Paterson, editor. *Proceedings* 17^{th} *ICALP,* Warwick, volume 443 of *Lecture Notes in Computer Science.* Springer-Verlag, July 1990.

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

[44] 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.

[45] Sade. Smooth operator. In *Diamond Life*. Epic Records, 1984.

[46] F.B. Schneider, B. Bloom, and K. Marzullo. Putting time into proof outlines. In *Proceedings REX Workshop "Real-Time: Theory in Practice",* Mook, The Netherlands, June 1991, Lecture Notes in Computer Science, pages 618–639. Springer-Verlag, to appear.

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

[48] C. Stirling. A generalization of Owicki-Gries's Hoare logic for a concurrent while-language. *Theoretical Computer Science*, 58:34–359, 1988.

[49] C. Stirling. Modal and temporal logics. In S. Abramsky, D. Gabbay, and T.Maibaum, editors, *Handbook of Logic in Computer Science, Vol I.* Oxford University Press, 1990. To appear.

[50] F.W. Vaandrager. On the relationship between process algebra and input/output automata (extended abstract). In *Proceedings* 6^{th} *Annual Symposium on Logic in Computer Science,* Amsterdam, pages 387–398. IEEE Computer Society Press, 1991.

[51] S. Weber, B. Bloom, and G. Brown. Compiling Joy to silicon: A verified silicon compilation scheme. In *Proceedings of the Advanced Research in VLSI and VLSI and Parallel Systems Conference*, Providence, RI, 1992.

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