GSOS and Finite Labelled Transition Systems

Luca Aceto
School of Cognitive and Computing Studies,
University of Sussex,
Falmer, Brighton BN1 9QH, England
email: luca@cogs.sussex.ac.uk

Abstract

Recently there has been considerable interest in studying formats of Plotkin style inference rules which ensure that the induced labelled transition system semantics have certain properties. In this note, I shall give a contribution to this line of research by giving a restricted version of Bloom, Istrail and Meyer's GSOS format [7, 6] which induces *finite* labelled transition systems.

Keywords: Concurrency, formal semantics, structural operational semantics, GSOS format, labelled transition systems, process graphs.

1 Introduction

Labelled transition systems [18] are a widely used model of program behaviour, and form the basis of Plotkin's structural approach to giving operational semantics to programming languages [24]. The states of the transition system are usually programs of the language one wants to give an operational semantics to, and the transitions between states are defined by means of a set of inference rules over the syntax of the language. These rules allow one to infer the semantics of a program from that of its subparts.

Recently there has been considerable interest in studying formats of Plotkin style inference rules which ensure that the induced labelled transition system semantics have certain properties. Contributions to this line of research may be found in, *e.g.*, [25, 7, 6, 8, 27, 13, 28]. In this note, I shall give a contribution to this line of research by giving a restricted version of Bloom, Istrail and Meyer's GSOS format [7, 6] which induces *finite* labelled transition systems.

Finite labelled transition systems may be used to describe many interesting concurrent systems, *e.g.* several communication protocols and mutual exclusion algorithms [29], and form the basis of all the semantic-based automated verification tools which have been developed. See, *e.g.*, [9, 11, 12, 26]. As (subsets of) programming languages which can be given semantics in terms of finite labelled transition systems are, at least in principle, amenable to automated verification techniques, it is important to develop techniques to check whether languages give rise to finite labelled transition systems. In particular, as this property is in general undecidable, it is interesting to develop sufficient syntactic conditions on the rules giving the operational semantics of programs which ensure finiteness of the defined labelled transition systems. The contribution of this note is one such syntactic condition over the GSOS format of operational rules.

I now give a brief outline of the contents of this note. Section 2 is devoted to preliminaries on GSOS systems and labelled transition systems. The format of *simple GSOS rules* is presented in

Section 3, where it is also shown that simple GSOS systems associate finite process graphs with each term. Section 4 is devoted to a possible generalization of this result to simple GSOS systems with recursive definitions. The note ends with some remarks on an infinitary version of GSOS systems and a discussion of related literature.

2 Preliminaries

Let Var be a denumerable set of *variables* ranged over by 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 $\mathbb{T}(\Sigma)$ of *terms* over Σ is the least set such that

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

I shall use $P, Q, \ldots$ to range over terms and the symbol $\equiv$ for the relation of syntactic equality on terms. $\mathrm{T}(\Sigma)$ is the set of *closed* terms over Σ, *i.e.*, terms that do not contain variables. Constants, *i.e.* terms of the form $f()$, will be abbreviated as f.

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.

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

Definition 2.1 (GSOS Rules and GSOS Systems [7]) *Suppose Σ is a signature. A* GSOS rule *ρ over Σ is an inference rule of the form:*

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

where all the variables are distinct, $m_i, n_i \ge 0$, f is an operation symbol from Σ with arity l, $C[\vec{x}, \vec{y}]$ is a Σ-context, and the a_{ij}, b_{ik}, and c are actions in Act. *In the above rule, f is the* principal operation *of the rule and $C[\vec{x}, \vec{y}]$ is its* target.
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.*

GSOS systems have been introduced and studied in depth in [7, 6]. The interested reader is referred to those references for much more on them. Intuitively, a GSOS system gives a language, whose constructs are the operations in the signature Σ_G, together with a Plotkin-style operational semantics [24] for it defined by the set of conditional rules R_G. As usual, the operational semantics for the closed terms over Σ_G will be given in terms of the notion of labelled transition system.

Definition 2.2 (Labelled Transition Systems) *Let A be a set of labels. A* labelled transition system (lts) *is a pair $(S, \rightarrow)$, where S is a set of* states *and $\rightarrow \subseteq S \times A \times S$ is the* transition relation. *As usual, I shall write $s \xrightarrow{a} t$ in lieu of $(s, a, t) \in \rightarrow$, and $s \rightarrow t$ when the label associated with the transition is immaterial. A state t is* reachable *from state s if there exist states $s_0, \ldots, s_n$ and labels $a_1, \ldots, a_n$ such that*

$$s = s_0 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n = t$$

The set of states which are reachable from s, also known as the set of derivatives *of s, will be denoted by* $\mathsf{der}(s)$.
A process graph *is a triple $(r, S, \rightarrow)$, where $(S, \rightarrow)$ is an LTS, $r \in S$ is the* root, *and each state in S is reachable from r. If $(S, \rightarrow)$ is an lts and $s \in S$ then $graph(s, (S, \rightarrow))$ is the process graph obtained by taking s as the root and restricting $(S, \rightarrow)$ to the part reachable from s. I shall write $graph(s)$ for $graph(s, (S, \rightarrow))$ whenever the underlying lts $(S, \rightarrow)$ is understood from the context.*
An lts $(S, \rightarrow)$ is finite *iff S and $\rightarrow$ are finite sets. A process graph $graph(s, (S, \rightarrow))$ is finite if the restriction of $(S, \rightarrow)$ to the part reachable from s is.*

For the sake of completeness, I shall now formally define the lts induced by a GSOS system following [7, 6].

Definition 2.3 *A* closed Σ-substitution *is a function σ from variables to closed terms over the signature Σ. For each term P, $P\sigma$ will denote the result of substituting $\sigma(x)$ for each x occurring in P.*

Definition 2.4 *A* transition relation *over a signature Σ is a relation $\rightsquigarrow \subseteq \mathrm{T}(\Sigma) \times \mathsf{Act} \times \mathrm{T}(\Sigma)$.*

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

$$\begin{aligned} \rightsquigarrow, \sigma \models P \xrightarrow{a} Q &\triangleq P\sigma \overset{a}{\rightsquigarrow} Q\sigma \\ \rightsquigarrow, \sigma \models P \overset{a}{\not\rightarrow} &\triangleq \nexists Q : P\sigma \overset{a}{\rightsquigarrow} Q \end{aligned}$$

For H a set of transition formulas, I define

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

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

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

Definition 2.5 *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 $\varphi\sigma = \left(P \xrightarrow{a} Q\right)$. The relation $\rightsquigarrow$ is* supported *by G iff each transition in $\rightsquigarrow$ is supported by a rule in R_G.*

The requirements of soundness and supportedness are sufficient to associate a unique transition relation with each GSOS system.

Lemma 2.6 ([7]) *For each GSOS system G there is a unique sound and supported transition relation.*

I write $\rightarrow_G$ for the unique sound and supported transition relation for G. The lts specified by a GSOS system G is then given by $lts(G) = (\mathrm{T}(\Sigma_G), \rightarrow_G)$ and the process graph defining the operational semantics of a closed term P is $graph(P, lts(G))$ (abbreviated to $graph(P)$ throughout the remainder of this note).

3 Finite Labelled Transition Systems from GSOS Rules

In this section, I shall show how to impose syntactic restrictions on the format of rules in a GSOS system G which ensure that $graph(P)$ is a finite process graph for each $P \in \mathrm{T}(\Sigma_G)$.

Definition 3.1 *A GSOS rule of the form (1) is* simple *iff $C[\vec{x},\vec{y}]$ is either a variable in $\vec{x},\vec{y}$ or it is of the form $g(z_1,\ldots,z_n)$ where each z_i is a variable in $\vec{x},\vec{y}$. A GSOS system $G=(\Sigma_G,R_G)$ is* simple *iff each rule in R_G is.*

I shall now proceed to show that if G is a simple GSOS system, then $graph(P)$ is a finite process graph for all $P \in \mathrm{T}(\Sigma_G)$. The following definition will be useful in the remainder of this note.

Definition 3.2 *Let $G=(\Sigma_G,R_G)$ be a simple GSOS system. The* operator dependency graph *associated with G is the directed graph with*

- *Σ_G as set of nodes, and*
- *set of edges E given by: $(f,g)\in E$ iff there exists a rule $\rho \in R_G$ with f as principal operation and target $g(z_1,\ldots,z_n)$, for some $z_1,\ldots,z_n \in \mathsf{Var}$.*

I shall write $f \prec_G g$ iff $fE^\star q$ in the operator dependency graph for G.

The following theorem, which gives a characterization of the set of derivatives of a term P in terms of those of its subterms, will be the key to the proof of the main result of this note.

Theorem 3.3 *Let $G=(\Sigma_G,R_G)$ be a simple GSOS system and $P\equiv f(P_1,\ldots,P_l)\in \mathrm{T}(\Sigma_G)$. Then*

$$\mathsf{der}(P) \subseteq \{g(R_1,\ldots,R_n) \mid f \prec_G g \wedge \forall i\in\{1,\ldots,n\}\exists j\in\{1,\ldots,l\}:\ R_i\in\mathsf{der}(P_j)\} \cup \bigcup_{i=1}^{l}\mathsf{der}(P_i)$$

Proof: Let $Q\in\mathsf{der}(P)$. By the definition of the set $\mathsf{der}(P)$, this means that $P\rightarrow_G^\star Q$. I shall now show that

$$Q \in \{g(R_1,\ldots,R_n) \mid f \prec_G g \wedge \forall i\in\{1,\ldots,n\}\exists j\in\{1,\ldots,l\}:\ R_i\in\mathsf{der}(P_j)\} \cup \bigcup_{i=1}^{l}\mathsf{der}(P_i)$$

by induction on the length of the derivation $P\rightarrow_G^\star Q$.

Base Case: $P\equiv Q$. The claim follows immediately as $\prec_G$ is reflexive by definition and $R\in\mathsf{der}(R)$ for all $R\in\mathrm{T}(\Sigma_G)$.

Inductive Step: $P\rightarrow_G R\rightarrow_G^\star Q$ for some $R\in\mathrm{T}(\Sigma_G)$. As $\rightarrow_G$ is supported by G, $P\rightarrow_G R$ because there exist a simple rule $\rho\in R_G$, with f as principal operation, of the form (1) and a substitution σ such that $P\equiv f(x_1,\ldots,x_l)\sigma$, $R\equiv C[\vec{x},\vec{y}]\sigma$ and $\rightarrow_G,\sigma\models H$, where H stands for the set of hypotheses of ρ. As ρ is a simple, there are two forms that the target context $C[\vec{x},\vec{y}]$ may take. I shall examine them in turn:

1. $C[\vec{x},\vec{y}]$ is either x_i or y_{ij}, for some i,j. In this case, R is syntactically equal to either $\sigma(x_i)$ or to $\sigma(y_{ij})$ for some i,j. Then surely $R\in\mathsf{der}(P_i)$ for some $i\in\{1,\ldots,l\}$. As $R\rightarrow_G^\star Q$, it follows that $Q\in\mathsf{der}(P_i)$ for some $i\in\{1,\ldots,l\}$. The proof for this case is then complete.

2. $C[\vec{x},\vec{y}] \equiv g(z_1,\ldots,z_n)$ for some $g \in \Sigma_G$ and $z_1,\ldots,z_n$ in $\vec{x},\vec{y}$. In this case, $R \equiv g(z_1,\ldots,z_n)\sigma$ and, as $\rightarrow_G, \sigma \models H$, it follows that

$$\forall h \in \{1,\ldots,n\} \exists j \in \{1,\ldots,l\} : \ \sigma(z_h) \in \mathsf{der}(P_j) \tag{2}$$

Let $\sigma(z_h) \equiv R_h$ for all $h \in \{1,\ldots,n\}$. Then $R \equiv g(R_1,\ldots,R_n) \rightarrow_G^\star Q$ by a shorter derivation. Applying the inductive hypothesis to $R \equiv g(R_1,\ldots,R_n) \rightarrow_G^\star Q$, it follows that

(a) $Q \in \mathsf{der}(R_k)$ for some $k \in \{1,\ldots,n\}$, or

(b) $Q \equiv g'(Q_1,\ldots,Q_s)$ for some $g' \in \Sigma_G$ and $Q_1,\ldots,Q_s \in \mathrm{T}(\Sigma_G)$ such that $g \prec_G g'$ and

$$\forall k \in \{1,\ldots,s\} \exists h \in \{1,\ldots,n\} : \ Q_k \in \mathsf{der}(R_h) \tag{3}$$

I shall proceed by examining these two possibilities in turn.

(a) Assume that $Q \in \mathsf{der}(R_k)$ for some $k \in \{1,\ldots,n\}$. In this case, as $R_k \in \mathsf{der}(P_j)$ for some $j \in \{1,\ldots,l\}$ by (2), by transitivity it follows that $Q \in \mathsf{der}(P_j)$ for some $j \in \{1,\ldots,l\}$.

(b) Assume that $Q \equiv g'(Q_1,\ldots,Q_s)$ for some $g' \in \Sigma_G$ and $Q_1,\ldots,Q_s \in \mathrm{T}(\Sigma_G)$ such that $g \prec_G g'$ and

$$\forall k \in \{1,\ldots,s\} \exists h \in \{1,\ldots,n\} : \ Q_k \in \mathsf{der}(R_h)$$

As $f \prec_G g$, by the transitivity of $\prec_G$ it follows that $f \prec_G g'$. Moreover, by (2) and (3), I immediately have that

$$\forall k \in \{1,\ldots,s\} \exists j \in \{1,\ldots,l\} : \ Q_k \in \mathsf{der}(P_j)$$

Hence, in this case, Q is an element of the set

$$\{g(R_1,\ldots,R_n) \mid f \prec_G g \wedge \forall i \in \{1,\ldots,n\} \exists j \in \{1,\ldots,l\} : \ R_i \in \mathsf{der}(P_j)\}$$

This completes the inductive argument and the proof of the theorem.

□

Theorem 3.4 *Let $G = (\Sigma_G, R_G)$ be a simple GSOS system. Then, for all $P \in \mathrm{T}(\Sigma_G)$, graph$(P)$ is a finite process graph.*

Proof: It is sufficient to show that $\mathsf{der}(P)$ is finite for all $P \in \mathrm{T}(\Sigma_G)$. This I prove by induction on the structure of P.

Assume then that $P \equiv f(P_1,\ldots,P_l)$. By the inductive hypothesis, $\mathsf{der}(P_i)$ is finite for each $i \in \{1,\ldots,l\}$. Using the finiteness of each $\mathsf{der}(P_i)$, I can now show that $\mathsf{der}(P)$ is itself finite. Indeed this follows easily from the above theorem as $\mathsf{der}(P)$ is contained in the set

$$\{g(R_1,\ldots,R_n) \mid f \prec_G g \wedge \forall i \in \{1,\ldots,n\} \exists j \in \{1,\ldots,l\} : \ R_i \in \mathsf{der}(P_j)\} \cup \bigcup_{i=1}^{l} \mathsf{der}(P_i)$$

which is finite as Σ_G and each $\mathsf{der}(P_i)$ are. □

The above theorem gives a purely syntactic way of checking whether the process graphs giving semantics to programs in a GSOS system are finite. To this end, it is sufficient to check that all the rules are simple. The reader familiar with the literature on process algebras, see *e.g.* [22, 17, 15, 5], will have already noticed that most of the standard operations used in process algebras are given operational semantics in terms of simple rules. Two exceptions known to me are the "desynchronizing" Δ operation present in the early versions of Milner's SCCS [21] studied in [20, 14], and the parallel composition operation in Milner, Parrow and Walker's π-calculus [23]. The Δ operation has rules (one such rule for each a):

$$\frac{x \stackrel{a}{\rightarrow} x'}{\Delta x \stackrel{a}{\rightarrow} \delta \Delta x'}$$

where δ is the *delay* operation of SCCS. The rules for the parallel composition operation in the π-calculus which are not simple are those dealing with the so-called *scope extrusions*. (See [23, Part II].) These take the form

$$\frac{P \stackrel{\bar{x}(w)}{\rightarrow} P',\ Q \stackrel{x(w)}{\rightarrow} Q'}{P \mid Q \stackrel{\tau}{\rightarrow} (w)(P' \mid Q')}$$

where (w) denotes the restriction operation of the π-calculus.

An example of an interesting operation whose operational rules are simple and use negative premises is the priority operation θ of Baeten, Bergstra and Klop [4]. Fix a partial ordering relation $>$ on Act. For each a the operation θ has a rule

$$\frac{x \stackrel{a}{\rightarrow} x', \quad x \stackrel{b}{\not\rightarrow} \ (\text{for all } b > a)}{\theta(x) \stackrel{a}{\rightarrow} \theta(x')} \tag{4}$$

which is simple. An example of an operation definable in terms of simple rules, but not definable in process algebras like CCS and ACP up to strong bisimulation equivalence is the operation $\mathsf{a\text{-}if\text{-}b}(\cdot)$ from [6]. This is given by the rule

$$\frac{x \stackrel{a}{\rightarrow} y_1, x \stackrel{b}{\rightarrow} y_2}{\mathsf{a\text{-}if\text{-}b}(x) \stackrel{a}{\rightarrow} \mathsf{a\text{-}if\text{-}b}(y_1)}$$

In addition, the format of simple GSOS rules allows for copying of arguments of operations. For example, the unary operation double with rule

$$\mathsf{double}(x) \stackrel{a}{\rightarrow} x \| x$$

where $\|$ denotes the parallel composition operator of Milner's CCS [22], is simple.

Theorem 3.4 would, however, not hold if I allowed for GSOS rules with more than one function symbol in their target, as the following example shows.

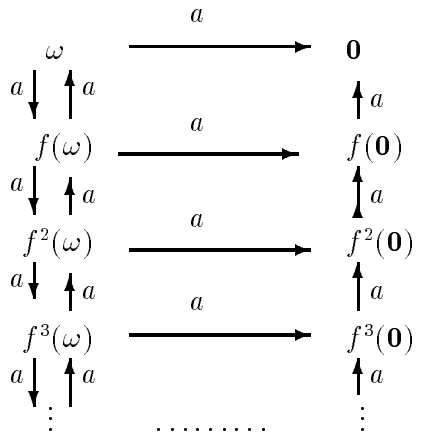
Example: Consider a GSOS system with a constant ω given by the rules

$$\omega \stackrel{a}{\rightarrow} \mathbf{0} \qquad \omega \stackrel{a}{\rightarrow} f(\omega)$$

where the unary function symbol f is specified by the rules

$$\frac{}{f(x) \stackrel{a}{\rightarrow} x} \qquad \frac{x \stackrel{a}{\rightarrow} y}{f(x) \stackrel{a}{\rightarrow} f(y)}$$

It is easy to see that $graph(\omega)$ is the following infinite labelled transition system:



Notice that this labelled transition system is infinite-state even modulo bisimulation equivalence [22], $\leftrightarrows$. In fact, it is immediate to see that, for all $n \neq m$, $f^n(\mathbf{0}) \leftrightarrows a^n \not\leftrightarrows a^m \leftrightarrows f^m(\mathbf{0})$. □

The example above shows that the condition on the contexts allowed as targets of simple GSOS rules cannot be relaxed in any obvious way. In fact, already admitting two function symbols in the targets of GSOS rules invalidates Theorem 3.4.

4 Adding Explicit Recursion

As shown by the previous example, GSOS processes can exhibit infinite behaviour even in the absence of a facility for recursive definitions of processes. Indeed, as stated in [7, 6], one can add guarded recursive processes as constants to GSOS systems. However, most process algebras which have been presented in the literature include a facility for recursive definitions. It is thus interesting to see how the result I have presented in the previous section can be extended to deal with languages which include explicit recursion. In this section I shall present one possible generalization of Theorem 3.4 to a class of these languages.

Definition 4.1 (Guarding Operations) *Let $G = (\Sigma_G, R_G)$ be a simple GSOS system. An operation $f \in \Sigma_G$ is* guarding *iff every rule in R_G with f as principal operation has an empty set of hypotheses,* i.e. *it is of the form*

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

An operation $f \in \Sigma_G$ is said to be hereditarily guarding *iff every $g \in \Sigma_G$ such that $f \prec_G g$ is guarding.*

The notion of guarding operation is closely related to the more general one of *guarded term* introduced by F. Vaandrager for de Simone systems in [28, Definition 3.1]. Indeed, an operation f is guarding in the sense of Definition 4.1 iff the term $f(X, \ldots, X)$, where X is a process name (see below), is guarded in the sense of [28, Definition 3.1].

The reader familiar with the literature on CCS will have noticed that the only guarding operations in CCS are the action-prefixing operations. These operations are also hereditarily guarding. As an example of an operation which is guarding, but not hereditarily guarding, consider the unary operation given by the rules:

$$\frac{}{f(x) \stackrel{a}{\rightarrow} g(x)} \qquad \frac{}{g(x) \stackrel{a}{\rightarrow} \mathbf{0}} \qquad \frac{x \stackrel{a}{\rightarrow} y}{g(x) \stackrel{a}{\rightarrow} g(y)} \tag{5}$$

where **0** denotes a stopped process. The operation f is guarding, but not hereditarily so, as g is not.

In order to add a facility for recursive definitions to simple GSOS systems, I shall assume a given, finite set of constant function symbols $\mathcal{N}$, whose elements will be referred to as *process names*. I shall use $X, Y, \ldots$ to range over $\mathcal{N}$. Without loss of generality, I shall assume that the constant symbols in $\mathcal{N}$ are *fresh*, in the sense that they do not appear in the signature of any simple GSOS system G.

The intended interpretation of process names will be given in terms of a declaration function. This is made precise in the following definition.

Definition 4.2 (Simple GSOS Systems with Explicit Recursion) *Let $G = (\Sigma_G, R_G)$ be a simple GSOS system. Let $\Delta : \mathcal{N} \to \mathrm{T}(\Sigma_G \cup \mathcal{N})$ be such that, for all $X \in \mathcal{N}$, $\Delta(X) \equiv f(X_1, \ldots, X_l)$ for some hereditarily guarding $f \in \Sigma_G$ and $X_1, \ldots, X_l \in \mathcal{N}$. The extension of G with recursive definitions G_Δ is the pair $(\Sigma_\Delta, R_\Delta)$ such that:*

- $\Sigma_\Delta = \Sigma_G \cup \mathcal{N}$ *and*
- R_Δ *is obtained by extending R_G with the rules (one such rule for each $X \in \mathcal{N}$ and $a \in \mathsf{Act}$)*

$$\frac{\Delta(X) \stackrel{a}{\to} y}{X \stackrel{a}{\to} y}$$

By structural induction on closed Σ_Δ-terms, it is easy to see that there is a unique transition relation $\to_{G_\Delta}$ that is sound and supported for G_Δ. In particular, this transition relation has the property that, for all $X \in \mathcal{N}$,

$$\begin{aligned} X \stackrel{a}{\to} P \quad &\Leftrightarrow \quad \Delta(X) \equiv f(X_1, \ldots, X_l) \stackrel{a}{\to} P \\ &\Leftrightarrow \quad \exists \rho = \left(f(x_1, \ldots, x_l) \stackrel{a}{\to} C[\vec{x}]\right) \in R_G : \quad C[\vec{X}] \equiv P \end{aligned}$$

With abuse of notation, I shall use $\mathit{graph}(P)$ to denote the process graph defining the operational semantics of a closed Σ_Δ-term P. I shall now show that $\mathit{graph}(P)$ is finite for all $P \in \mathrm{T}(\Sigma_\Delta)$.

By inspecting the proof of Theorem 3.3, it is immediate to see that the statement also holds over G_Δ. In fact, only properties of simple rules were used in the proof of that result.

Lemma 4.3 *Let $G = (\Sigma_G, R_G)$ be a simple GSOS system, and G_Δ be as in Definition 4.2. Then, for all $P \equiv f(P_1, \ldots, P_l) \in \mathrm{T}(\Sigma_\Delta)$,*

$$\mathsf{der}(P) \subseteq \{g(R_1, \ldots, R_n) \mid f \prec_G g \wedge \forall i \in \{1, \ldots, n\} \exists j \in \{1, \ldots, l\} : \ R_i \in \mathsf{der}(P_j)\} \cup \bigcup_{i=1}^{l} \mathsf{der}(P_i)$$

In order to prove that simple GSOS systems with explicit recursion give rise to finite process graphs, I shall need a sharpened version of the above result for process names.

Theorem 4.4 *Let $G = (\Sigma_G, R_G)$ be a simple GSOS system, and G_Δ be as in Definition 4.2. Then, for all hereditarily guarding $f \in \Sigma_G$ and $X_1, \ldots, X_l \in \mathcal{N}$,*

$$\mathsf{der}(f(X_1, \ldots, X_l)) \subseteq \{g(Y_1, \ldots, Y_n) \mid g \in \Sigma_G \wedge Y_1, \ldots, Y_n \in \mathcal{N}\} \cup \mathcal{N}$$

Proof: Let $Q \in \mathsf{der}(f(X_1, \ldots, X_l))$. This means that $f(X_1, \ldots, X_l) \rightarrow^{\star}_{G_\Delta} Q$. I shall now show that

$$Q \in \{g(Y_1, \ldots, Y_n) \mid g \in \Sigma_G \wedge Y_1, \ldots, Y_n \in \mathcal{N}\} \cup \mathcal{N}$$

by induction on the length of the derivation $f(X_1, \ldots, X_l) \rightarrow^{\star}_{G_\Delta} Q$. The base case of the induction is trivially seen to hold.

For the inductive step, assume that $f(X_1, \ldots, X_l) \rightarrow_{G_\Delta} P \rightarrow^{\star}_{G_\Delta} Q$, for some $P \in \mathrm{T}(\Sigma_\Delta)$. As $\rightarrow_{G_\Delta}$ is supported by G_Δ and f is hereditarily guarding, $f(X_1, \ldots, X_l) \rightarrow_{G_\Delta} P$ because there exists a simple rule $\rho \in R_G$ such that

$$\rho = f(x_1, \ldots, x_l) \stackrel{a}{\rightarrow} C[\vec{x}] \text{ and } C[\vec{X}] \equiv P$$

As f is simple, there are two possible forms $C[\vec{x}]$ may take; namely, $C[\vec{x}] \equiv g(z_1, \ldots, z_m)$, where each z_i is a variable in the set $\{x_1, \ldots, x_l\}$, or $C[\vec{x}] \equiv x_i$ for some $i \in \{1, \ldots, l\}$.

If $C[\vec{x}] \equiv g(z_1, \ldots, z_m)$, then $P \equiv g(\vec{Z})$, where each Z_i is in $\vec{X}$. As f is hereditarily guarding and $f \prec_G g$, so is g. The claim then follows immediately by using the inductive hypothesis.

Otherwise, $P \equiv X_i$ for some $i \in \{1, \ldots, l\}$. Now, $P \equiv X_i \rightarrow^{\star}_{G_\Delta} Q$ iff either $Q \equiv X_i$ or $\Delta(X_i) \equiv g(Y_1, \ldots, Y_m) \rightarrow^{+}_{G_\Delta} Q$. If $Q \equiv X_i$ then the claim follows trivially. Otherwise, by the construction of G_Δ, I have that g is itself hereditarily guarding. The claim then follows by applying the inductive hypothesis to the derivation $g(Y_1, \ldots, Y_m) \rightarrow^{+}_{G_\Delta} Q$. □

The following result generalizes Theorem 3.4 to simple GSOS systems with explicit recursive definitions.

Theorem 4.5 *Let $G = (\Sigma_G, R_G)$ be a simple GSOS system, and G_Δ be as in Definition 4.2. Then, for all $P \in \mathrm{T}(\Sigma_\Delta)$, $graph(P)$ is a finite process graph.*

Proof: It is sufficient to show that $\mathsf{der}(P)$ is finite for all $P \in \mathrm{T}(\Sigma_\Delta)$. This I prove by induction on the structure of P. The proof follows that of Theorem 3.4, using Theorem 4.4 for process names, and Lemma 4.3 for the inductive step. □

Theorem 4.5 would, however, not hold if I allowed for extensions of simple GSOS systems with recursive definitions involving operations which are not hereditarily guarding, as the following example shows.

Example: Consider a simple GSOS system with constant $\mathbf{0}$ and unary operations f, g specified by the rules given in (5). As previously noted, f is guarding, but not hereditarily guarding. Let X be a process name in $\mathcal{N}$, and take $\Delta(X) \equiv f(X)$. Then it is easy to see that $graph(X)$ is an infinite state process graph. In fact, $X \rightarrow^{+}_{G_\Delta} g^n(X)$, for all n.

Note, moreover, that $graph(X)$ is infinite-state even modulo bisimulation equivalence. In fact, it can be seen that each term of the form $g^n(X)$ can perform n a-actions in a row and become $\mathbf{0}$ in doing so, while no $g^m(X)$ with $m < n$ can. □

5 Concluding Remarks

5.1 Infinitary GSOS Systems

In keeping with the standard treatment of GSOS languages [7, 6], I have only considered languages of a finitary nature, *i.e.* languages over a finite set of combinators and finite sets of actions and GSOS rules. Process algebras like CCS [22] and Meije [3], however, postulate an infinite action

set. Consequently, the results presented in this note cannot be applied directly to the full versions of these calculi. I shall now briefly sketch a possible extension of the results presented in Section 3 to a class of "infinitary" GSOS systems. For the purpose of this section, I assume that the set of actions Act is countable[1].

Definition 5.1 *An* infinitary GSOS system *is a pair $G = (\Sigma_G, R_G)$, where Σ_G is a countable signature and R_G is a countable set of GSOS rules over Σ_G.*

In the presence of a possibly infinite action set and signature, care must be taken to preserve the basic sanity properties of GSOS systems [7, 6] which have bearing on the aim of this note. For instance, processes which give rise to infinitely branching process graphs can now be easily specified, and should be ruled out. An example of such a process is the constant $\mathsf{all\text{-}actions}$ with rules (one such rule for each $a \in \mathsf{Act}$):

$$\mathsf{all\text{-}actions} \stackrel{a}{\rightarrow} \mathsf{all\text{-}actions}$$

The process graph associated with $\mathsf{all\text{-}actions}$ is infinitely branching, if Act is infinite. As a technical notion that will be useful in identifying an interesting class of "well-behaved" infinitary GSOS systems, I define the notion of a *positive trigger* of an l-ary operation f: an l-tuple over Act which gives the positive constraints under which some rule for f might fire.

Definition 5.2 *The* positive trigger *of rule (1) is the l-tuple $\langle e_1, \ldots, e_l \rangle$, where*

$$e_i = \{a_{ij} \,|\, 1 \leq j \leq m_i\}$$

For example, the positive trigger of the operation $\mathsf{a\text{-}if\text{-}b}(\cdot)$ is the tuple $\langle \{a, b\} \rangle$.

The following definition presents an adaptation of the notion of bounded de Simone system, due to F. Vaandrager [28, Definition 3.2], to infinitary GSOS systems. The interested reader is referred to [28] for more information on the notion of boundedness.

Definition 5.3 (Boundedness) *An infinitary GSOS system is* bounded *iff for each operation and for each positive trigger, the corresponding set of rules is finite.*

All the standard operations used in the literature on process algebras satisfy the boundedness condition. An operation which does not is the constant $\mathsf{all\text{-}actions}$ given above.

A bounded infinitary GSOS system associates a finitely branching process graph with each term. (See [28, Theorem 3.3] for a similar result over de Simone systems.)

Proposition 5.4 *For each infinitary GSOS system G there is a unique sound and supported transition relation, $\rightarrow_G$. If G is bounded, then $\rightarrow_G$ is finitely branching,* i.e. *for all $P \in \mathrm{T}(\Sigma_G)$, the set*

$$\left\{ Q \mid \exists a \in \mathsf{Act} : \; P \stackrel{a}{\rightarrow} Q \right\}$$

is finite.

[1] A set X is countable if it is empty or if there exists an enumeration of X, that is a surjective mapping from the set of positive integers onto X.

Proof: The proof of the first part of this proposition follows the standard lines of that of Lemma 2.6. To prove the second statement, it is sufficient to show that, for bounded infinitary GSOS systems, the sets $\left\{a \in \mathsf{Act} \mid \exists Q \in \mathrm{T}(\Sigma_G) : \ P \stackrel{a}{\rightarrow} Q\right\}$ and $\left\{Q \mid P \stackrel{a}{\rightarrow} Q\right\}$ are finite, for all $P \in \mathrm{T}(\Sigma_G)$ and $a \in \mathsf{Act}$. This can be easily shown by structural induction on P. □

In general, the condition of boundedness is not enough to ensure that the process graph associated with each term in a simple infinitary GSOS system is finite. Consider, for example, a simple infinitary GSOS system with constants c_i, $i \in \omega$, and rules

$$c_i \stackrel{a}{\rightarrow} c_{i+1}$$

Such a GSOS system is obviously bounded, but $\mathsf{der}(c_i)$ is infinite for all $i \in \omega$. This pathological behaviour is due to the fact that the operator dependency relation $\prec_G$ associated with such an infinitary GSOS system is not *image-finite* [16]. For the sake of completeness, I recall that a binary relation $\mathcal{R}$ over a set E is image-finite iff for all $e \in E$ the set $\{e' \mid e \,\mathcal{R}\, e'\}$ is finite.

Theorem 5.5 *Let $G = (\Sigma_G, R_G)$ be a simple, bounded infinitary GSOS system such that $\prec_G$ is image-finite. Then, for all $P \in \mathrm{T}(\Sigma_G)$, $graph(P)$ is a finite process graph and the* sort *of P*

$$\mathsf{sort}(P) = \left\{a \in \mathsf{Act} \mid \exists Q, R \in \mathsf{der}(P) : \ Q \stackrel{a}{\rightarrow} R\right\}$$

is finite.

Proof: By structural induction on P, one proves that $\mathsf{der}(P)$ is finite using Theorem 3.3 and the fact that $\prec_G$ is image-finite. Next, by Lemma 5.4, I obtain that $graph(P)$ is finite branching. These two facts imply that $graph(P)$ is indeed finite, and $\mathsf{sort}(P)$ is a finite set. □

The operator dependency graph associated with the recursion-free sublanguages of all the process algebras I am aware of is image-finite. Indeed, $\prec_G$ is the identity in CCS, CSP, MEIJE and ACP.

5.2 Related Work

After the technical part of this note was written, I. Castellani and F. Vaandrager pointed out to me the important reference [19]. In that paper, E. Madelaine and D. Vergamini study some syntactic conditions on operational rules in de Simone's format [25] which ensure that the process graphs giving the operational semantics of terms are finite. This they do by identifying two classes of well-behaved operations, which they call *non-growing operations* and *sieves*. Intuitively, *non-growing operations* are operations which, when fed with (terms denoting) finite process graphs, build finite process graphs. *Sieves* are a special class of unary non-growing operations whose operational rules have the form

$$\frac{x \stackrel{a}{\rightarrow} x'}{f(x) \stackrel{a}{\rightarrow} f(x')}$$

The reader familiar with standard process algebras will have noticed that operations like CCS restriction and renaming [22], and hiding [17] are sieves.

In view of Theorem 3.4, all GSOS operations given in terms of simple rules are non-growing in the sense of Madelaine and Vergamini. Moreover, the rule for sieves are all simple. The syntactic condition used by Madelaine and Vergamini to establish the fact that some operations are non-growing is based on term rewriting techniques; namely, on finding a *simplification ordering* over

terms (see [19, Definition 4]). This is similar in spirit to the technique proposed in [2, Section 6] to show that linear GSOS systems, which are a generalization of de Simone systems, are *syntactically well-founded*. The notion of simple rule, albeit less powerful than term-rewriting techniques based on simplification orderings, offers a much simpler syntactic criteria which guarantees the finiteness of the semantics of terms. It is also a criteria which applies well to general GSOS rules; for instance, it can be used to show that some operations which use negative premises, like the priority operation specified by (4), generate finite process graphs from finite ones.

Specialized techniques which can be used to show that certain processes give rise to finite process graphs have been proposed for CCS and related languages. The interested reader is invited to consult [10] and the references therein. Not surprisingly, these specialized methods tend to be more powerful than general syntactic ones as they rely on language-dependent semantic information. For instance, a method to check the finiteness of a large set of CCS processes based on abstract interpretation techniques [1] has been proposed in [10]. However, the language dependency of these techniques, which is the source of their power, makes it difficult to generalize them to classes of languages.

Acknowledgements: Many thanks to Bard Bloom for his useful comments on this note, and to Ilaria Castellani and Frits Vaandrager for pointing out the reference [19].

References

[1] S. Abramsky and C. Hankin. *Abstract interpretation of declarative languages.* Ellis Horwood, 1987.

[2] L. Aceto, B. Bloom, and F.W. Vaandrager. Turning SOS rules into equations. Report CS-R9218, CWI, Amsterdam, June 1992. Submitted for publication to *Information and Computation.*

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

[4] 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.

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

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

[7] B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced: preliminary report. In *Conference Record of the 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.*

[8] R.N. Bol and J.F. Groote. The meaning of negative premises in transition system specifications (extended abstract). In J. Leach Albert, B. Monien, and M. Rodríguez, editors,

Proceedings 18^{th} *ICALP,* Madrid, volume 510 of *Lecture Notes in Computer Science*, pages 481–494. Springer-Verlag, 1991. Full version available as Report CS-R9054, CWI, Amsterdam, 1990.

[9] R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench: a semantics-based verification tool for finite-state systems. Report ECS–LFCS–89–83, University of Edinburgh, Edinburgh, 1989.

[10] N. De Francesco and P. Inverardi. Proving finiteness of CCS processes by non-standard semantics, 1992. Nota interna IEI–CNR, Pisa. A preliminary version of this paper appeared in the *Proceedings of CAV '92.*

[11] J.-C. Fernandez. Aldébaran: a tool for verification of communicating processes. Technical report SPECTRE c14, LGI–IMAG, Grenoble, 1989.

[12] J.C Godskesen, K.G. Larsen, and M. Zeeberg. TAV (Tools for Automatic Verification): users manual, 1988.

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

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

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

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

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

[18] R.M. Keller. Formal verification of parallel programs. *Communications of the ACM*, 19(7):371–384, 1976.

[19] E. Madelaine and D. Vergamini. Finiteness conditions and structural construction of automata for all process algebras. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 3:275–292, 1991.

[20] R. Milner. On relating synchrony and asynchrony. Technical Report CSR–75–80, Department of Computer Science, University of Edinburgh, 1981.

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

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

[23] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Part I + II. *Information and Control*, 100(1):1–77, 1992.

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

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

[26] R. de Simone and D. Vergamini. Aboard AUTO. Technical Report 111, INRIA, Centre Sophia-Antipolis, Valbonne Cedex, 1989.

[27] 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.

[28] F.W. Vaandrager. Expressiveness results for process algebras. To appear in the *Proceedings of REX '92*, 1992.

[29] D. Walker. Analysing mutual exclusion algorithms using CCS. Technical Report ECS-LFCS-88-45, Department of Computer Science, University of Edinburgh, 1988.