GSOS and finite labelled transition systems

Luca Aceto

School of Cognitive and Computing Sciences, University of Sussex, Falmer, Brighton, BN1 9QH, UK

Communicated by A.R. Meyer
Received April 1993
Revised July 1993

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 Bloom et al. (1988), Bloom (1989) which induces finite labelled transition systems.

1. Introduction

Labelled transition systems [21] are a widely used model of program behaviour, and form the basis of Plotkin's structural approach to giving operational semantics to programming languages [28]. 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., [8–10, 16, 29, 31, 32]. In this note, I shall give a contribution to this line of research by

Correspondence to: L. Aceto, School of Cognitive and Computing Sciences, University of Sussex, Falmer, Brighton, BN1 9QH United Kingdom. Email: luca@cogs.sussex.ac.uk.
giving a restricted version of Bloom, Istrail and Meyer's GSOS format [8, 9] 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 [33], and form the basis of all the semantic-based automated verification tools which have been developed. See, e.g., [11, 14, 15, 30]. As (subsets of) 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 T(Σ) of terms over Σ is the least set such that

- Each x ∈ Var is a term.
- If f is an operation symbol of arity l, and P₁, ..., P₁ are terms, then f(P₁, ..., P₁) is a term.

I shall use P, Q, ... to range over terms and the symbol = for the relation of syntactic equality on terms. T(Σ) 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[ x ] is a term in which at most the variables x appear. C[P ] is C[ x ] with x₁ replaced by P₁ 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 ⇝ P'. A negative transition formula is a pair of a term and an action, written P ⇞. In general, the terms in the transition formula will contain variables.
Definition 2.1 (GSOS rules and GSOS systems [9]). Suppose $\Sigma$ is a signature. A GSOS rule $\rho$ over $\Sigma$ is an inference rule of the form:

$$\left( \bigcup_{i=1}^f \{ x_i \rightarrow y_{ij} \} \bigg| 1 \leq j \leq m_i \right) \cup \left( \bigcup_{l=1}^g \{ x_i \rightarrow b_{lk} \} \bigg| 1 \leq k \leq n_l \right)$$

$$f(x_1, \ldots, x_t) \rightarrow C[x, y]$$

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

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

GSOS systems have been introduced and studied in depth in [8, 9]. The interested reader will find much more on them in the aforementioned references. Intuitively, a GSOS system gives a language, whose constructs are the operations in the signature $\Sigma_G$, together with a Plotkin-style operational semantics [28] for it defined by the set of conditional rules $R_G$. As usual, the operational semantics for the closed terms over $\Sigma_G$ 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 (Its) 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 $s \rightarrow s'$ 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

$$s = s_0 \rightarrow a_1 \rightarrow s_1 \rightarrow \ldots \rightarrow a_n \rightarrow s_n = t$$

The set of states which are reachable from $s$, also known as the set of derivatives of $s$, will be denoted by $\text{der}(s)$.

A process graph is a triple $(r, S, \rightarrow)$, where $(S, \rightarrow)$ is an Its, $r \in S$ is the root, and each state in $S$ is reachable from $r$. If $(S, \rightarrow)$ is an Its 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 Its $(S, \rightarrow)$ is understood from the context. An Its $(S, \rightarrow)$ 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 Its induced by a GSOS system following [9, 8].

Definition 2.3. A closed $\Sigma$-substitution is a function $\sigma$ from variables to closed terms over the signature $\Sigma$. 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 $\Sigma$ is a relation $\rightarrow \subseteq T(\Sigma) \times \text{Act} \times T(\Sigma)$. 
Let $\rightarrow$ be a transition relation and $\sigma$ a closed substitution. For each transition formula $\varphi$, the predicate $\rightarrow, \sigma \models \varphi$ is defined by

$$\rightarrow, \sigma \models P \Rightarrow Q \triangleq \sigma \Rightarrow Q \sigma$$
$$\rightarrow, \sigma \models P \Rightarrow \exists Q : \sigma \Rightarrow Q$$

For $\mathcal{H}$ a set of transition formulas, I define

$$\rightarrow, \sigma \models \mathcal{H} \triangleq \forall \varphi \in \mathcal{H} : \rightarrow, \sigma \models \varphi$$

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

$$\rightarrow, \sigma \models \frac{H}{\varphi} \triangleq \left( \rightarrow, \sigma \models H \Rightarrow \rightarrow, \sigma \models \varphi \right).$$

**Definition 2.5.** Suppose $G$ is a GSOS system and $\rightarrow$ is a transition relation over $\Sigma_G$. Then $\rightarrow$ is sound for $G$ iff for every rule $\rho \in R_G$ and every closed $\Sigma_G$-substitution $\sigma$, we have $\rightarrow, \sigma \models \rho$. A transition $P \Rightarrow Q$ is supported by some rule $\frac{H}{\varphi} \in R_G$ iff there exists a substitution $\sigma$ such that $\rightarrow, \sigma \models H$ and $\varphi \sigma = (P \Rightarrow Q)$. The relation $\rightarrow$ is supported by $G$ iff each transition in $\rightarrow$ 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** ([9]). *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 its specified by a GSOS system $G$ is then given by $\text{its}(G) = (T(\Sigma_G), \rightarrow_G)$ and the process graph defining the operational semantics of a closed term $P$ is $\text{graph}(P, \text{its}(G))$ (abbreviated to $\text{graph}(P)$ throughout the remainder of this paper).

### 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 $\text{graph}(P)$ is a finite process graph for each $P \in T(\Sigma_G)$.

**Definition 3.1.** A GSOS rule of the form (1) is simple iff $C[\bar{x}, \bar{y}]$ is either a variable in $\bar{x}, \bar{y}$ or it is of the form $g(z_1, \ldots, z_n)$ where each $z_i$ is a variable in $\bar{x}, \bar{y}$. 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 $\text{graph}(P)$ is a finite process graph for all $P \in T(\Sigma_G)$. 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

- $\Sigma_G$ 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 $z_1, \ldots, z_n \in \text{Var}$.

I shall write $f \prec_G g$ iff $f E^* g$ in the operator dependency graph for $G$, where $E^*$ denotes the reflexive and transitive closure of $E$.

The following proposition, 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.

Proposition 3.3. Let $G = (\Sigma_G, R_G)$ be a simple GSOS system and $P \equiv f(P_1, \ldots, P_l) \in T(\Sigma_G)$. Let reach$(P)$ denote the set

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

Then $\text{der}(P) \subseteq \text{reach}(P)$.

Proof. Let $Q \in \text{der}(P)$. By the definition of the set $\text{der}(P)$, this means that $P \rightarrow^*_G Q$. I shall now show that $Q \in \text{reach}(P)$ by induction on the length of the derivation $P \rightarrow^*_G Q$.

Base case. $P \equiv Q$. The claim follows immediately as $\prec_G$ is reflexive by definition and $K \in \text{der}(K)$ for all $K \in T(\Sigma_G)$.

Inductive step. $P \rightarrow^*_G R \rightarrow^*_G Q$ for some $R \in T(\Sigma_G)$. 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 $\sigma$ such that $P \equiv f(x_1, \ldots, x_l)\sigma, R = C[\hat{x}, \hat{y}]\sigma$ and $\rightarrow^*_G, \sigma \models H$, where $H$ stands for the set of hypotheses of $\rho$. As $\rho$ is simple, there are two forms that the target context $C[\hat{x}, \hat{y}]$ may take. I shall examine them in turn:

1. $C[\hat{x}, \hat{y}]$ is either $x_i$ or $y_j$ for some $i, j$. In this case, $R$ is syntactically equal to either $\sigma(x_i)$ or $\sigma(y_j)$ for some $i, j$. Then surely $R \in \text{der}(P_j)$ for some $i \in \{1, \ldots, l\}$. As $R \rightarrow^*_G Q$, it follows that $Q \in \text{der}(P_i)$ for some $i \in \{1, \ldots, l\}$. The proof for this case is then complete.

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

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

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

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

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

$$\forall k \in \{1, \ldots, s\} \exists h \in \{1, \ldots, n\}: Q_k \in \text{der}(R_h).$$
I shall proceed by examining these two possibilities in turn.
(a) Assume that \( Q \in \text{d}(R_k) \) for some \( k \in \{1, \ldots, n\} \). In this case, as \( R_j \in \text{d}(P_j) \) for some \( j \in \{1, \ldots, l\} \) by (2), by transitivity it follows that \( Q \in \text{d}(P_j) \) for some \( j \in \{1, \ldots, l\} \).
(b) Assume that \( Q \equiv g'(Q_1, \ldots, Q_s) \) for some \( g' \in \Sigma \) and \( Q_1, \ldots, Q_s \in T(\Sigma) \) such that \( g <_\sigma g' \) and
\[
\forall k \in \{1, \ldots, s\} \exists h \in \{1, \ldots, n\} : Q_k \in \text{d}(R_h).
\]
As \( f <_\sigma g \), by the transitivity of \( <_\sigma \) it follows that \( f <_\sigma g' \). Moreover, by (2) and (3), I immediately have that
\[
\forall k \in \{1, \ldots, s\} \exists j \in \{1, \ldots, l\} : Q_k \in \text{d}(P_j).
\]
Hence, in this case, \( Q \) is an element of the set
\[
\{ g(R_1, \ldots, R_n) | f <_\sigma g \land \forall i \in \{1, \ldots, n\} \exists j \in \{1, \ldots, l\} : R_i \in \text{d}(P_j) \}.
\]
This completes the inductive argument and the proof of the proposition. \( \square \)

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

**Proof.** It is sufficient to show that \( \text{d}(P) \) is finite for all \( P \in T(\Sigma) \). This I prove by induction on the structure of \( P \).

Assume then that \( P = f(P_1, \ldots, P_l) \). By the inductive hypothesis, \( \text{d}(P_i) \) is finite for each \( i \in \{1, \ldots, l\} \). Using the finiteness of each \( \text{d}(P_i) \), I can now show that \( \text{d}(P) \) is itself finite. Indeed this follows easily from the above proposition as \( \text{d}(P) \) is contained in the set \( \text{reach}(P) \), which is finite as \( \Sigma \) and each \( \text{d}(P_i) \) are. \( \square \)

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. [25, 20, 18, 7], 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 are the “desynchronizing” \( \Delta \) operation present in the early versions of Milner’s SCCS [24] studied in [23, 17], and the parallel composition operation in Milner et al. \( \pi \)-calculus [26]. The \( \Delta \) operation has rules (one such rule for each \( a \)):

\[
\begin{align*}
\Delta x &\rightarrow x' \\
\delta &\rightarrow \Delta x'
\end{align*}
\]

where \( \delta \) is the \textit{delay} operation of SCCS. The rules for the parallel composition operation in the \( \pi \)-calculus which are not simple are those dealing with the so-called
GSOS and finite labelled transition systems

Scope extrusions (see [26, Part II]). These take the form

\[
P \xrightarrow{x(w)} P', \quad Q \xrightarrow{x(w)} Q' \quad \frac{P \parallel (w) (P' \parallel Q')}{P \parallel Q}
\]

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

An example of an interesting operation whose operational rules are simple and use negative premises is the priority operation \( \theta \) of Baeten et al. [6]. Fix a partial ordering relation \( > \) on \( \text{Act} \). For each \( a \) the operation \( \theta \) has a rule

\[
x \xrightarrow{a} x', \quad x \xrightarrow{b} (\text{for all } b > a) \quad \frac{\theta(x) \xrightarrow{a} \theta(x')}{(4)}
\]

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 \( \text{a-while-b}(\cdot) \) from [8]. This is given by the rule

\[
x \xrightarrow{a} y_1, \quad x \xrightarrow{b} y_2 \quad \frac{\text{a-while-b}(x) \xrightarrow{a} \text{a-while-b}(y_1)}{}
\]

In addition, the format of simple GSOS rules allows for copying of arguments of operations. For example, the unary operation \( \text{double} \) with rule

\[
\text{double}(x) \xrightarrow{a} x \parallel x
\]

where \( \parallel \) denotes the parallel composition operator of Milner's CCS [25], 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 \( \omega \) given by the rules

\[
\omega \xrightarrow{a} 0 \quad \omega \xrightarrow{f(\omega)}
\]

where the unary function symbol \( f \) is specified by the rules

\[
x \xrightarrow{a} y \quad \frac{f(x) \xrightarrow{a} x}{f(x) \xrightarrow{a} f(y)}
\]

Note that the second rule for \( \omega \) is not simple as its target has two function symbols. It is easy to see that \( \text{graph}(\omega) \) is the infinite labelled transition system shown in Fig. 1. Notice that this labelled transition system is infinite-state even modulo bisimulation equivalence [25], \( \leftrightarrow \). In fact, it is immediate to see that, for all \( n \neq m, f^{-n}(0) \leftrightarrow a^n \leftrightarrow a^m \leftrightarrow f^m(0) \). \( \square \)

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 [9, 8], 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 generalization of Theorem 3.4 to a class of these languages.

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

$$f(x_1, ..., x_l) \rightarrow C[\overline{x}]$$

An operation $f \in C, R_C$ is said to be hereditarily guarding iff every $g \in C, R_C$ such that $f < G g$ is guarding.

The notion of guarding operation is similar to the definition of guardedness given in [4]. It is also closely related to the more general one of guarded term introduced by F. Vaandrager for de Simone systems in [32, Definition 3.11]. Indeed, an operation $f$ is guarding in the sense of Definition 4.1 iff the term $f(X, ..., X)$, where $X$ is a process name (see below), is guarded in the sense of [32, 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:

$$f(x) \rightarrow g(x) \quad \frac{g(x) \rightarrow 0}{x \rightarrow y} \quad \frac{x \rightarrow y}{g(x) \rightarrow g(y)}$$

Fig. 1.
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 $G=(\Sigma, R)$ be a simple GSOS system. Let $d: \mathcal{N} \rightarrow T(\Sigma \cup \mathcal{N})$ be such that, for all $X \in \mathcal{N}$, $d(X) = f(X_1, \ldots, X_i)$ for some hereditarily guarding $f \in \Sigma$ and $X_1, \ldots, X_i \in \mathcal{N}$. The extension of $G$ with recursive definitions $G_d$ is the pair $(\Sigma_d, R_d)$ such that:

- $\Sigma_d = \Sigma \cup \mathcal{N}$ and
- $R_d$ is obtained by extending $R$ with the rules (one such rule for each $X \in \mathcal{N}$ and $a \in \text{Act}$)

$$
\begin{align*}
\Gamma(X) \Rightarrow y \\
X \Rightarrow y
\end{align*}
$$

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

$$
X \Rightarrow P \iff \Gamma(X) \equiv f(X_1, \ldots, X_i) \Rightarrow P
$$

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

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

**Lemma 4.3.** Let $G=(\Sigma, R)$ be a simple GSOS system, and $G_d$ be as in Definition 4.2. Then, for all $P \equiv f(P_1, \ldots, P_i) \in T(\Sigma_d)$, $\text{der}(P) \subseteq \text{reach}(P)$.

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 $G_3$ be as in Definition 4.2. Then, for all hereditarily guarding $f \in \Sigma_G$ and $X_1, \ldots, X_l \in \mathcal{N}$,
\[
\text{der}(f(X_1, \ldots, X_l)) \subseteq \{g(Y_1, \ldots, Y_n) | g \in \Sigma_G \land Y_1, \ldots, Y_n \in \mathcal{N}\} \cup \mathcal{N}.
\]

Proof. Let $Q \in \text{der}(f(X_1, \ldots, X_l))$. This means that $f(X_1, \ldots, X_l) \rightarrow^{*}_{G_3} Q$. I shall now show that
\[
Q \in \{g(Y_1, \ldots, Y_n) | g \in \Sigma_G \land Y_1, \ldots, Y_n \in \mathcal{N}\} \cup \mathcal{N}
\]
by induction on the length of the derivation $f(X_1, \ldots, X_l) \rightarrow^{*}_{G_3} Q$. The base case of the induction is trivially seen to hold.

For the inductive step, assume that $f(X_1, \ldots, X_l) \rightarrow^{G_3} P \rightarrow^{*}_{G_3} Q$, for some $P \in T(\Sigma_3)$. As $\rightarrow^{G_3}$ is supported by $G_3$ and $f$ is hereditarily guarding $f(X_1, \ldots, X_l) \rightarrow^{G_3} P$ because there exists a simple rule $\rho \in R_G$ such that
\[
\rho = f(x_1, \ldots, x_l) \rightarrow^{G_3} C[\tilde{x}] \text{ and } C[\tilde{x}] \equiv P.
\]
As $f$ is simple, there are two possible forms $C[\tilde{x}]$ may take; namely, $C[\tilde{x}] \equiv g(z_1, \ldots, z_m)$, where each $z_i$ is a variable in the set $\{x_1, \ldots, x_l\}$, or $C[\tilde{x}] \equiv x_i$ for some $i \in \{1, \ldots, l\}$.

If $C[\tilde{x}] \equiv g(z_1, \ldots, z_m)$, then $P \equiv g(\tilde{Z})$, where each $Z_i$ is in $\tilde{X}$. As $f$ is hereditarily guarding and $f <_{G_3} g$, 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, $P \equiv X_i \rightarrow^{G_3} Q$ iff either $Q \equiv X_i$ or $A(X_i) \equiv g(Y_1, \ldots, Y_m) \rightarrow^{G_3} Q$. If $Q \equiv X_i$ then the claim follows trivially. Otherwise, by the construction of $G_3$, I have that $g$ is itself hereditarily guarding. The claim then follows by applying the inductive hypothesis to the derivation $g(Y_1, \ldots, Y_m) \rightarrow^{*}_{G_3} Q$.

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 $G_3$ be as in Definition 4.2. Then, for all $P \in T(\Sigma_3)$, $\text{graph}(P)$ is a finite process graph.

Proof. It is sufficient to show that $\text{der}(P)$ is finite for all $P \in T(\Sigma_3)$. This 1 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 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 $\Delta(X) = f(X)$. Then it is easy to see that $\text{graph}(X)$ is an infinite state process graph. In fact, $X \rightarrow \delta_n g^n(X)$, for all $n$.

Note, moreover, that $\text{graph}(X)$ 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 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 [9, 8], I have only considered languages of a finitary nature, i.e. languages over a finite set of combinator and finite sets of actions and GSOS rules. Process algebras like CCS [25] 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 $G = (\Sigma_0, R_0)$, where $\Sigma_0$ is a countable signature and $R_0$ is a countable set of GSOS rules over $\Sigma_0$.

In the presence of a possibly infinite action set and signature, care must be taken to preserve the basic sanity properties of GSOS systems [9, 8] 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 all-actions with rules (one such rule for each $a \in \text{Act}$):

$$\text{all-actions} \rightarrow \text{all-actions}$$

The process graph associated with all-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 $f$. This is an $l$-tuple over $2^\text{Act}$ associated with a rule $\rho$ for $f$, which gives the sets of actions that the arguments of $f$ must be able to perform in order for $\rho$ to 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 $a$-while-$b(\cdot)$ is the tuple $\langle \{ a, b \} \rangle$.

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 $X$. 
The following definition presents an adaptation of the notion of bounded de Simone system, due to F. Vaandrager [32, Definition 3.2], to infinitary GSOS systems. The interested reader is referred to [32] 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 all-actions given above.

A bounded infinitary GSOS system associates a finitely branching process graph with each term. (See [32, 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 \( P \in T(\Sigma_G) \), the set

\[
\{ Q \mid \exists a \in \text{Act}: P \xrightarrow{a} Q \}
\]

is finite.

**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 \( \{ a \in \text{Act} \mid \exists Q \in T(\Sigma_G): P \xrightarrow{a} Q \} \) and \( \{ Q \mid P \xrightarrow{a} Q \} \) are finite, for all \( P \in T(\Sigma_G) \) and \( a \in \text{Act} \). 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

\[
c_i \xrightarrow{a} c_{i+1}
\]

Such a GSOS system is obviously bounded, but \( \text{der}(c_i) \) 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* [19]. 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 \( P \in T(\Sigma_G) \), \( \text{graph}(P) \) is a finite process graph and the sort of \( P \)

\[
\text{sort}(P) = \{ a \in \text{Act} \mid \exists Q, R \in \text{der}(P): Q \xrightarrow{a} R \}
\]

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

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

### 5.2. Related work

After the technical part of this note was written, Castellani and Vaandrager pointed out to me the important reference [22]. In that paper, Madelaine and Vergamini study some syntactic conditions on operational rules in de Simone's format [29] 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

$$x \xrightarrow{a} x'$$

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

The reader familiar with standard process algebras will have noticed that operations like CCS restriction and renaming [25], and hiding [20] 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 [22, 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 criterion which guarantees the finiteness of the semantics of terms. It is also a criterion 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. Moreover, whereas the existence of a simplification ordering compatible with a set of rewrite rules is not decidable, it is immediate to check whether the rules in a GSOS system are simple.

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 [13] 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 [13]. However, the language dependency of these techniques, which is the source of their power, makes it difficult to generalize them to classes of languages.

Formats of structural operational rules similar to the simple GSOS rules studied in this note have emerged in work by other researchers. See, e.g., [27, Definition 13], [12, page 230] and [5]. The convergence on similar formats for operational rules in investigations underlied by different motivations has probably good reasons to exist, and should be explained. This I leave as an interesting topic for further research.

Acknowledgement

Many thanks to Bard Bloom for his useful comments on this note, to Ilaria Castellani and Frits Vaandrager for pointing out the reference [22], and to Philippe Darondeau for bringing [4, 5, 12] to my attention. The anonymous referees provided useful comments.

References


