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*

3 Finite Labelled ransition 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 $P \in \mathrm{T}(\Sigma$

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. 4ll $R_1,\ldots,R_n)$ $1,\ldots,l\}$.

The above theorem gives a purely syntactic way of checking whether the process graphs giving semantics to programs

where $\mathbf{0}$ 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*

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 $P \in \mathrm{T}(\Sigma_\Delta$

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 $a \in \mathsf{Act}$):

$$\mathsf{all\text{-}actions} \xrightarrow{a} \mathsf{all\text{-}actions}$$

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

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

[25] R. de Simone. Higher-level synchronising devices