A Note on Expressiveness of Process Algebra

Iain Phillips
iccp@doc.ic.ac.uk
Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ, United Kingdom

Abstract

We extend the work of Baeten, Bergstra & Klop by showing that every recursively enumerable transition system can be expressed in process algebra up to weak bisimulation.

The various forms of process algebra (CCS [5], CSP [4], ACP [2]) are known to have universal computing power, in the sense that they can compute any recursive function on e.g. the natural numbers (see for instance [5], page 135). It is common to give them an operational semantics in terms of (labelled) transition systems. We can then ask to what extent process algebra can express transition systems. To make the question more precise, given an equivalence $\sim$ on transition systems we can ask:

> For any transition system T within a certain class, is there a process algebra term p such that $O(p) \sim T$?

Here $O(p)$ is the operational semantics of p. Not every transition system can arise as the semantics of a process. A necessary condition is that it should be a *transition graph*, viz. it should have a starting state (the root), and every state must be reachable from the root.

The question first received attention from de Simone [3]. He showed the positive result that any recursively enumerable (r.e.) transition graph can be expressed by a (finite) MEIJE-SCCS term up to isomorphism (Theorem 3.2). Baeten, Bergstra & Klop [1] showed two expressiveness results, one positive and one negative:

- Every computable transition graph (with a finite set of labels) is expressible by a finite guarded specification in ACP_τ, but in this case up to weak bisimulation (Theorem 6.11).
- There is a computable transition graph which cannot be expressed by any finite guarded specification in ACP up to strong bisimulation (Theorem 8.2).

Vaandrager [6] points out that de Simone allows unguarded recursion. Vaandrager proves a strengthened version of Baeten, Bergstra & Klop's second result, namely, for any r.e. language L with an effective operational semantics there is a computable transition graph which cannot be expressed by L up to trace equivalence. It follows, of course, that de Simone's language does not have an effective operational semantics in Vaandrager's sense.

Baeten, Bergstra & Klop's first result leaves open the question of whether finite guarded process algebra can express r.e. transition graphs up to weak bisimulation. This note answers this in the affirmative. Our method is simply to show that any r.e. transition graph can be converted to a computable transition graph which is equivalent under weak bisimulation.

We start by giving precise definitions of the terms we have used already:

Definition 1 A *labelled transition system* is a triple $(P, A, \rightarrow)$, where P is a set of states, A is a set of labels and $\rightarrow$ is a relation on $P \times A \times P$. A *transition graph* (TG) is a transition system together with a root $r \in P$ such that every $p \in P$ is reachable from r. We let g range over TGs. A TG is *r.e.* if there is a coding of P, A into the natural numbers such that P, A are r.e. and $\rightarrow$ is r.e. A TG is *computable* if it is finite branching, and given any $p \in P$ the set of immediate derivatives $\{(a, q) | p \xrightarrow{a} q\}$ can be computed effectively in p.

Notice that with a computable TG, given any n we can compute its unfolding to depth n. Clearly any finite TG is computable. To clarify the distinction between the two types of TG, note the following: it might be that in an r.e. TG the set of derivatives of a given p was finite. We could effectively enumerate the whole of this set in a finite time. However we would not then know that we had finished the enumeration. With a computable TG we have this further information. We give an example of an r.e. TG which is finite branching and has a finite label set, but is not computable:

Example 2 Let K be an r.e. set which is not recursive. For instance, K might be the set of all n such that the nth Turing machine halts when given input n. Our TG will have states p_n, q_n $(n \in \mathbb{N})$, root p_0, labels a, b, and transitions $p_n \xrightarrow{a} p_{n+1}$ (all n), and

$$p_n \xrightarrow{b} q_n \text{ iff } n \in K.$$

See Figure 1. For purposes of illustration we have taken $0 \notin K$, $1 \in K$, $2 \notin K$, $3 \in K$.

We refer the reader to [5] for the definitions of weak and strong bisimulation.

Theorem 3 Let g be an r.e. TG. Then there exists a computable TG g' such that g, g' are weakly bisimilar.

We use the following lemma in the proof:

Lemma 4 Let $X \subseteq \mathbb{N}$ be r.e. Then there exists a total recursive function $f : \mathbb{N} \rightarrow \mathbb{N} \cup \{\uparrow\}$ such that

- $rge(f) \cap \mathbb{N} = X$
- if $n \in X$ then there exist infinitely many $i \in \mathbb{N}$ such that $f(i) = n$.

Proof of lemma. We proceed informally. Let some machine M be given which enumerates X. This machine computes by stages. Define $f(i) = \uparrow$ for any stage i at which M has output nothing. If and when a stage i_0 is reached at which M first outputs a number n_0, let $f(i_0) = n_0$. Values of f for $i > i_0$ are then obtained by enumerating the values output by M in rotation following the plan

$$0, 1, 0, 1, 2, 0, 1, 2, 3, \ldots$$

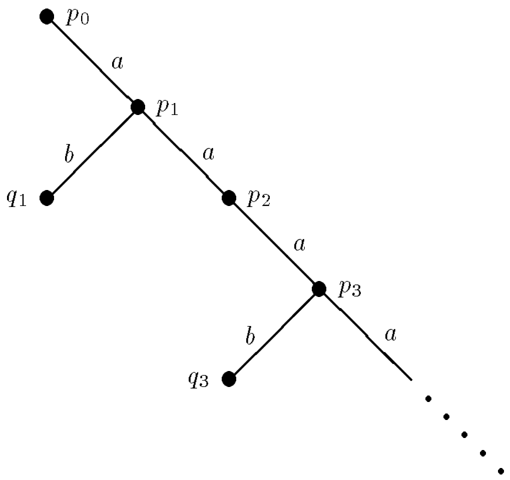

Figure 1

and marking time by further rotation if the output from M fails to keep up (and in particular, of course, if X is in fact finite). □

REMARK. f has the further property that if $f(i) = \uparrow$ then $f(j) = \uparrow$ for all $j < i$. We do not need this in what follows.

PROOF OF THEOREM. Let g be an r.e. TG $(P, A, \rightarrow, r)$. We define a new TG g' with states $P \times \mathbb{N}$. We write (p, n) as $p.n$. The new root will be $r.0$ and the new set of labels will be $A \cup \{\tau\}$. We may or may not have $\tau \in A$. It remains to define the new $\rightarrow$. Consider any state p of g. Let $p.i \xrightarrow{\tau} p.(i+1)$ for each i, so that we create an infinite chain of τ's in g', starting at $p.0$. Let $(a_0, p_0), (a_1, p_1), \ldots$ be an effective enumeration of the derivatives of p. Let f be the corresponding function given by the lemma. At the ith node $p.i$ of the chain we create an extra branch $p.i \xrightarrow{a} q.0$ if $f(i) = (a, q)$. If $f(i) = \uparrow$ then we do not branch at that node. See Figure 2.

In the example shown, $f(0) = \uparrow$, $f(1) = (a, q)$, $f(2) = (a', q')$, $f(3) = (a, q)$, etc.

Carry out this process uniformly for all $p \in P$. We now have a new TG g' which is weakly bisimilar to g, with weak bisimulation $\{(p, p.n) | p \in P, n \in \mathbb{N}\}$. It is plainly computable. □

REMARKS. The new TG is not only finitely branching; each node has out-degree either 1 or 2. Notice that it is everywhere divergent in the sense that from every state there is an infinite sequence of τ's. However, it may be argued that we have only really introduced divergence in the case where p has no derivatives (which, of course, we cannot in general discover in finite time). In the other cases the divergence can be eliminated by an assumption of fairness of choice. The above proof was inspired by Lemma 6.10 of [1] which states that a computable TG is weakly bisimilar to a TG with branching at most 2, and where τ-cycles are introduced.

Corollary 5 Every r.e. TG $(P, A, \rightarrow, r)$ where A is finite can be expressed by a finite guarded ACP_τ specification up to weak bisimulation. □

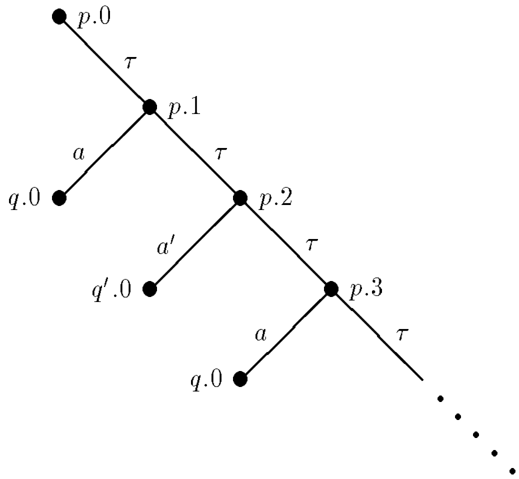

Figure 2

REMARK. Baeten, Bergstra & Klop restrict themselves to finite A in their first result. This may not be necessary, in view of the following:

Proposition 6 (cf [1] Theorem 6.8, Corollary 6.9) Let g be a computable TG where

- ♦ each state has one or two successors
- ♦ a τ is always immediately possible

Then there is a process S such that $O(S)$ and g are weakly bisimilar.

IDEA OF PROOF. Apart from using techniques such as in [1] to express in process algebra both the storage of numbers in counter processes and the computable functions which modify these counters, we need to deal with a possibly infinite A. Let A be enumerated as $a_0, a_1, \ldots$ with $\tau = a_0$. To correspond to a given state of g, we need S to offer either τ alone, or τ together with a_n, some n. Let us assume that we have calculated n and stored it in a counter. We actually store $n+1$, so that $n = 0$ corresponds to the case of τ alone. We now indicate how S can produce the required choice of actions. We use CCS-style notation. The counter is specified by

$$C(n) = d^n e$$

Define the relabelling R by

$$R(a_i) = a_{i+1}$$

(all $i \in \mathbb{N}$). Let a process B be given by

$$\begin{aligned} B &= eb' + dB' \\ B' &= e(a_0 b + b') + dB'[R] \end{aligned}$$

Let

$$B_n = (B|C(n))\backslash\{d, e\}$$

Then

$$\begin{aligned} B_0 &\approx b' \\ B_{n+1} &\approx a_n b + b' \end{aligned}$$

(where $\approx$ denotes weak bisimulation). The point of the actions b, b' is to trigger the processes which carry on the computation after either a_n or τ (respectively) has taken place. The special case $n = 0$ is to cope with the possibility that only a τ is offered.

Of course, we have neglected many of the complications, such as the need to regenerate B for the next stage of the computation. □

Acknowledgments

Supported by SERC grant GR/F72475. This note arose out of discussions with Marta Kwiatkowska, and was assisted by a discussion with Mike Smyth.

References

[1] J.C.M. Baeten, J. Bergstra, and J.-W. Klop. On the consistency of Koomen's fair abstraction rule. *Theoretical Computer Science*, 51:129–176, 1987.

[2] J.C.M. Baeten and J.-W. Klop. Process algebra for synchronous communication. *Information and Computation*, 60:109–137, 1984.

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

[4] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[5] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[6] F.W. Vaandrager. Expressiveness results for process algebras. In J. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Semantics: foundations and applications*, 1993. Lecture Notes in Computer Science Vol. 666.