A generic process algebra

Jos C.M. Baeten

Formal Methods Group, Division of Computer Science,
Department of Mathematics and Computer Science,
Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands.
E-mail: josb@win.tue.nl

Mario Bravetti

Dipartimento di Scienze dell'Informazione, Università di Bologna,
Mura Anteo Zamboni 7, I-40127 Bologna, Italy.
E-mail: bravetti@cs.unibo.it

Abstract

The three classical process algebra CCS, CSP and ACP present several differences in their respective technical machinery. This is due, not only to the difference in their operators, but also to the terminology and "way of thinking" of the community which has been (and still is) working with them. In this paper we will first discuss such differences and try to clarify the different usage of terminology and concepts. Then, as a result of this discussion, we define a generic process algebra where each basic mechanism of the three process algebras is expressed by an operator and which can be used as an underlying common language. We show an example of the advantages of adopting such a language instead of one of the three more specialized algebras: producing a complete axiomatization of finite-state behaviours.

1 Introduction

The huge amount of research work on process algebra carried out in the last 25 years started from the introduction of the theory of the process algebras CCS [13], CSP [11] and ACP [7]. In spite of conceptual similarities those process algebras where developed starting from quite different viewpoints and give rise to different approaches: CCS is heavily based on having an observational bisimulation-based theory for communication over processes starting from an operational viewpoint; CSP is born as a theoretical version of a practical language for concurrency and is still based on operational semantics which, however, is interpreted w.r.t. a simpler theory based more on traces than on bisimulation; finally ACP starts from a completely different viewpoint where concurrent systems are seen, according to a

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

purely mathematical algebraic view, as the solutions of systems of equations (axioms) over the signature of the algebra considered, and operational semantics and bisimulation (in this case a different notion of branching bisimulation is considered) are seen as just one of the possible models over which the algebra can be defined and the axioms can be applied. Such differences reflect the different "way of thinking" of the different communities which started working (and often keep working) with them. This paper aims at pointing out such differences, which often reflect in the usage of different terminology within the different communities, and at creating a means for a unified view of process algebras. The impact of such differences can be easily underestimated at a first glance. However when it comes to dealing with related machinery concerning recursion and treatment of process variables in the three different contexts the need for clarification and comparison comes out. Our study concretizes in the development of a common theory of process algebra. In particular we make use of a process algebra called TCP+REC, which is defined in such a way that each basic mechanism involved in the operators of the three process algebras is directly expressed by a different operator. The idea is that TCP+REC: (i) is an underlying common language which can be used to express processes of any of the three process algebras; (ii) can be used as a means for formal comparison of the three respective approaches; and (iii) can be used to produce new results in the context of process algebra theory due to its generality (e.g. to produce an axiomatization which is complete over finite-state processes).

The remainder of the paper is organized as follows. In Sect. 2 we focus on presentation of differences concerning recursion and treatment of process variables in CCS, CSP and ACP. In Sect. 3 we present TCP+REC. In Sect. 4 we present the result of axiomatization over finite-state processes which is based on TCP+REC.

2 Process variables and recursion

The different viewpoint assumed in the ACP process algebra with respect to, e.g., the CCS process algebra gives rise to a different technical treatment of process variables in axiomatizations.

In CCS axioms are considered as equations between terms which can be expressed by using meta-variables P (as, e.g., in $P+P=P$) standing for any term. The meaning is that the model generated by the term in the left of "$=$" is equivalent to the term to the right of "$=$" according to the considered notion of equivalence (e.g. observational congruence for CCS). Terms to the left and to the right of $=$ may include free variables X (they may be so called open terms). The meaning in this case is the following: for any substitution of free variables the term on the left is equivalent to the term on the right. Often a different meta-variable E is used to range over open terms, while P just ranges over closed terms: i.e. terms where free variables X do not occur (or if they occur they are bound by, e.g. a recursion operator like $recX.E$). Note that in this context the word "process" (recalling the meta-variable P) is used as synonymous for "closed term".

In ACP axioms are instead considered as equations over *process variables* "x"

(representing any *process* in the model that is assumed for the algebra) combined by means of operators in the signature of the algebra (as, e.g., in $x+x=x$). Note that here, differently from the case of CCS, the word process is used to denote any element in the model which is considered (e.g. transition systems modulo branching bisimulation). Such process variables act similarly to meta-variables P of CCS only if the so-called *term model* is assumed: the model in which each element is generated/represented by terms made up of operators of the signature of the considered process algebra. In ACP free variables X of CCS are not considered (term models never include free variables): this is mainly due to the fact that in ACP a binding operator (like "$recX.P$" in CCS) is not considered. Note however that this does not prevent the possibility of "reasoning" with open terms: this is done in ACP axiomatizations by replacing axioms in the body of other axioms. Related to this difference between ACP and CCS, is the usage of the word "calculus" to denote a process algebra. Differently from CCS, in the ACP context the word calculus is only used if binding operators are introduced, in order to emphasize that we leave the purely algebraic domain in the presence of such operators. Finally we would like to observe that the notion of "complete axiomatization" in the context of CCS corresponds to what in ACP is said to be "ground complete": i.e. the axiomatization is complete with respect to identities between closed terms (so, for the term model).

Once these basic differences are explained, in the following we will focus on the different ways of expressing recursion in the three process algebras. Let V be a set of variables ranging over processes, ranged over by X,Y. According to a terminology which is usual in the ACP setting, a *recursive specification* $E=E(V)$ is a set of equations $E=\{X=t_X \mid X\in V\}$ where each t_X is a term over the signature in question and variables from V. A *solution* of a recursive specification $E(V)$ is a set of elements $\{y_X \mid X\in V\}$ of some model of the theory such that the equations of $E(V)$ correspond to equivalent elements, if for all $X\in V$, y_X is substituted for X. Mostly, we are interested in one particular variable $X\in V$, called the *initial* variable. The *guardedness* criterion for such recursive specifications ensures unique solutions in preferred models of the theory, and unguarded specifications will have several solutions. For example the unguarded specification $\{X=X\}$ will have every element as a solution and, e.g. if transition systems modulo observational congruence are considered, the unguarded specification $\{X=\tau.X\}$ will have multiple solutions, as any transition system with a τ-step as only initial step will satisfy this equation.

As far as guarded recursive specifications are concerned, while in CCS the unique solution can be represented by using the recursion operator "$recX.P$", in ACP, where there is no explicit recursion operator, this is not possible. As a consequence while in CCS the property of uniqueness of the solution is expressed by the two standard axioms

$$(Unfold)\ recX.t=t\{recX.t/X\}$$
$$(Fold)\ t'=t\{t'/X\}\ \Rightarrow\ t'=recX.t \ \text{ if } X \text{ is guarded in } t$$

which actually make it possible to derive the solution, in ACP this property is expressed by using so-called "principles". The *Recursive Definition Principle*, which corresponds to the Unfold axiom, states that each recursive specification has a solution (no matter if it is guarded). The *Recursive Specification Principle*, which corresponds to the Fold axiom, states that each guarded recursive specification has at most one solution.

As far as unguarded recursive specifications are concerned, the process algebras ACP, CCS and CSP handle them in different ways. In ACP, variables occurring in unguarded recursive specifications are treated as (constrained) variables, and not as processes. In CCS, where recursive specifications are made via so-called "*constants*", ranged over by $A, B, ..$, or equivalently by the $recX.t$ operator, where t is a term containing variable X, from the set of solutions the solution will be chosen that has the least transitions in the generated transition system. Thus, the solution chosen for the equation $\{X = X\}$ has no transitions (it is the deadlocked process δ in the ACP terminology), and the solution chosen for $\{X = \tau.X\}$ has only a τ-transition to itself, a process that is bisimilar to $\tau.\delta$ in observational congruence. In CCS such a behaviour is expressed by the three axioms for unguarded recursion

$$(FUng)\ recX.(X+t) = recX.t$$
$$(WUng1)\ recX.(\tau.X+t) = recX.\tau.t$$
$$(WUng2)\ recX.(\tau.(X+t)+s) = recX.(\tau.X+t+s)$$

that make it possible to turn each unguarded recursive specification into a guarded one (actually WUng1 and WUng2 can be expressed by a single axiom as we show in [5]). It is worth noting that, if unguardedness is caused just by τ actions (weak unguardedness), as in $\{X = \tau.X\}$, and not by variable being directly executable in right-hand side of equations (full unguardedness), as in $\{X = X\}$, in ACP it is possible to obtain the same effect as with $recX.t$ in CCS by means of the hiding operator: e.g. the CCS semantics of $\{X = \tau.X\}$ can be obtained in ACP by writing $\tau_{\{a\}}(X)$ where $X = a.X$ (in ACP "$\tau_I(t)$" is the hiding operator). This technique makes it possible to "reason" about weakly guarded recursion also in ACP, but in an undirect way, via the hiding operator. More precisely, in ACP it is possible to express an analogy of axioms WUng1 and WUng2 by adding a much more complex set of conditional equations called CFAR (Cluster Fair Abstraction Rule) introduced in [15]. CFAR is a generalisation of the KFAR (Koomen's Fair Abstraction Rule) introduced in [6]. Note however that CFAR and KFAR, differently from the axioms above, also work for branching bisimulation instead of Milner's observational congruence. Finally, in CSP the way of dealing with unguarded recursive specification is such that a solution will be chosen like in CCS, but a different one: the least deterministic one. Thus, both CCS and CSP use a least fixed point construction, but with respect to a different ordering relation. In CSP, the solution chosen for the equation $\{X = X\}$ is the *chaos* process $\perp$, a process that satisfies $x + \perp = \perp$ for all processes x (for an extension of ACP with such a process see [3]).

3 The generic process algebra TCP+REC

The algebra TCP+REC is an extension of the algebra TCP [1,2] which in turn extends ACP by including successful termination ε and prefixing à la CCS. The algebra TCP is parameterized on a set of actions A (which does not include the special internal action τ) and is endowed with sequencing "$t' \cdot t''$", hiding "$\tau_I(t)$", restriction "$\partial_H(t)$", relabeling "$\rho_f(t)$", and parallel composition "$t' \parallel t''$" à la ACP (where a communication function γ is assumed to compute the type of communicating actions). Moreover it includes the left-merge "$t' \lfloor\!\lfloor t''$" and synchronization merge "$t' \,|\, t''$" operators which are used for axiomatizing parallel composition. TCP+REC considers in addition to TCP a recursion operator $\langle X|E \rangle$ (where $E = E(V)$ is a recursive specification and X a variable in V which acts as the initial variable) which, similarly as in CCS, computes minimal fixpoint solutions of (non guarded) systems of equations and which extends the similar operator introduced in [9] with the possibility of nesting recursion operators inside recursion operators. $\langle X|E \rangle$ encompasses both the CCS $recX.t$ operator (which is obtained by taking $E = \{X = t\}$) and the standard way to express recursion in ACP (where usually only guarded recursion is considered via systems of equations E). Moreover, in the context of CCS, the operator $\langle X|E \rangle$ can be used to directly express the solution of the (Standard) Equation Sets considered by Milner in [12,14] (i.e. systems of equations E that contain free variables) and used to prove completeness of axiomatizations for strong bisimulation and observational congruence over terms with guarded recursion.

This process algebra is *generic*, in the sense that all features of commonly used process algebras can be embedded in it. In the following, we made use of [10,4] (the translation of CSP external choice is due to Pedro D'Argenio, a similar translation has also been developed by Rob van Glabbeek).

We consider a subtheory corresponding to CCS, see [13]. This is done by omitting the signature elements $\varepsilon, \cdot, \lfloor\!\lfloor, |$. Next, we specialize the parameter set A by separating it into three parts: a set of names $\mathscr{A}$, a set of co-names $\bar{\mathscr{A}}$ and a set of communications $\mathscr{A}_c$ such that for each $a \in \mathscr{A}$ there is exactly one $\bar{a} \in \bar{\mathscr{A}}$ and exactly one $a_c \in \mathscr{A}_c$. The communication function γ is specialized to having as the only defined communications $\gamma(a,\bar{a}) = \gamma(\bar{a},a) = a_c$, and then the CCS parallel composition operator $|_{CCS}$ can be defined by the formula

$$x \,|_{CCS}\, y = \tau_{\mathscr{A}_c}(x \parallel y).$$

We consider a subtheory corresponding to ACP$_\tau$, see [8]. This is done by defining, for each $a \in A$, a new constant a by $a = a.\varepsilon$, and then omitting the signature elements $\varepsilon, ., \rho_f$.

We consider a subtheory corresponding to CSP, see [11]. The *non-deterministic choice* operator $\sqcap$ can be defined by

$$x \sqcap y = \tau.x + \tau.y.$$

As far as the CSP *parallel composition* operator $\parallel_S$ is concerned, we specialize the parameter set A into two parts: a set of names $\mathscr{A}$ and a set of communications $\mathscr{A}_c$ such that for each $a \in \mathscr{A}$ there is exactly one $a_c \in \mathscr{A}_c$. The communication function γ is specialized to having as the only defined communications $\gamma(a,a) = a_c$, and further, we use the renaming function f that has $f(a_c) = a$. Then, $x \parallel_S y$, where x and y are processes using names over $\mathscr{A}$ only and $S \subseteq \mathscr{A}$, can be defined by the formula

$$x \parallel_S y = \rho_f(\partial_{S \cup (\mathscr{A}_c - S_c)}(x \parallel y))$$

where we use "S_c" to denote the set of names $\{a_c \mid a \in S\}$ and "$-$" to express difference of sets. As far as the CSP *external choice* operator $\Box$ is concerned, we further specialize the set of names $\mathscr{A}$ into three parts: a set of names $\mathscr{B}$, and two sets of names $\mathscr{B}_1$ and $\mathscr{B}_2$, such that for each $a \in \mathscr{B}$ there is exactly one name $a_1 \in \mathscr{B}_1$ and one name $a_2 \in \mathscr{B}_2$. The communication function γ is not changed (no further communication is added). Finally, we use the renaming functions f' and f'' that have $f'(a_1) = a$ and $f''(a_2) = a$. Then, $x \Box y$, where x and y are processes using names over $\mathscr{B}$ only, can be defined by the formula

$$x \Box y = \rho_{f' \cup f''}(\ \ (\rho_{f'^{-1}}(x) \parallel \rho_{f''^{-1}}(y)) \ \parallel_{\mathscr{B}_1 \cup \mathscr{B}_2} \ ({\mathscr{B}_1}^* \varepsilon + {\mathscr{B}_2}^* \varepsilon) \ \ )$$

where, given a set of names B and a process x, "$B^* x$" stands for:

$$recX.(x + \sum_{a \in B} a.X).$$

4 Example: Axiomatizing Finite-State Processes

As we show in [5], by using a restricted version of TCP+REC it is possible to solve the open problem of developing a ground-complete axiomatization for a process algebra with static operators (like, e.g., CCS parallel and restriction) over finite-state processes modulo observational congruence, thus extending Milner's result which holds for CCS without static operators. Note that if we consider the signature of full CCS, we have that Milner's axioms are no longer sufficient to get rid of unguarded recursion. In other words, even if two CCS terms are both finite-state it may be that they are not equated by an axiomatization including the standard CCS axioms (the axioms for CCS without the $recX.t$ recursion operator) plus the axioms for unguarded and guarded recursion. An example is the following:

$$(\,(recX.a.X) \mid (recX.a.X)\,) \backslash a$$

where "$|$" and "$\backslash$" denote CCS parallel composition and restriction, respectively. The model of such a term has just one state with a τ self-loop, but cannot be equated by Milner's axiomatization to the equivalent term $recX.\tau.X$ or to $\tau.\underline{0}$ because unguardedness cannot be removed in order to apply the folding axiom and get rid of static operators.

In particular we consider TCP+REC$_f$ where in the $\langle X|E\rangle$ operators, with $E = E(V)$, we require variables in V (which are bound by the operator) to be "serial" in E: i.e. we disallow such variables to occur inside E in the scope of static operators like hiding, restriction, relabeling and parallel composition operators or in the left-hand side of a sequencing operator. The axiomatization is obtained by considering the crucial axiom

$$\tau_I(\langle X|X = t\rangle) = \langle X|X = \tau_I(t)\rangle \ \text{ if } X \text{ is serial in } t$$

which allows the hiding operator (the only static operator which may generate unguarded recursion) to be exchanged with the recursion operator.

References

[1] J.C.M. Baeten. Embedding untimed into timed process algebra: The case for explicit termination. *Mathematical Structures in Computer Science*, 13(4):589–618, 2003.

[2] J.C.M. Baeten, T. Basten, and M.A. Reniers. *Algebra of Communicating Processes*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2005.

[3] J.C.M. Baeten and J.A. Bergstra. Process algebra with propositional signals. *Theoretical Computer Science*, 177(2):381–406, 1997.

[4] J.C.M. Baeten, J.A. Bergstra, C.A.R. Hoare, R. Milner, J. Parrow, and R. de Simone. The variety of process algebra. Deliverable ESPRIT Basic Research Action 3006, CONCUR, 1991.

[5] J.C.M. Baeten and M. Bravetti. A ground-complete axiomatization of finite state processes in process algebra. In Martín Abadi and Luca de Alfaro, editors, *Proceedings CONCUR'05*, volume 3653 of *Lecture Notes in Computer Science*, pages 248–262. Springer Verlag, 2005.

[6] J. A. Bergstra and J. W. Klop. Verification of an alternating bit protocol by means of process algebra. In Wolfgang Bibel and Klaus P. Jantke, editors, *Proc. Mathematical Methods of Specification and Synthesis of Software Systems*, volume 215 of *LNCS*, pages 9–23. Springer, 1986.

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

[8] J.A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985.

[9] J.A. Bergstra and J.W. Klop. A complete inference system for regular processes with silent moves. In F.R. Drake and J.K. Truss, editors, *Proc. Logic Colloquium'86*, pages 21–81. North-Holland, 1988.

[10] R.J. van Glabbeek. Notes on the methodology of CCS and CSP. *Theoretical Computer Science*, 177(6):329–349, 1997.

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

[12] R. Milner. A complete inference system for a class of regular behaviours. *Journal of Comput. System Sci.*, 28(3):439–466, 1984.

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

[14] R. Milner. A complete axiomatization for observational congruence of finite-state behaviours. *Information and Computation*, 81:227–247, 1989.

[15] F.W. Vaandrager. Verification of two communication protocols by means of process algebra. Technical Report report CS-R8608, CWI Amsterdam, 1986.