A Ground-Complete Axiomatization of Finite State Processes in Process Algebra

Jos C.M. Baeten[1] and Mario Bravetti[2]

[1] Division of Computer Science,Technische Universiteit Eindhoven, josb@win.tue.nl
[2] Department of Computer Science,Università di Bologna, bravetti@cs.unibo.it

Abstract. We consider a generic process algebra of which the standard process algebras ACP, CCS and CSP are subalgebras of reduced expressions. In particular such an algebra is endowed with a recursion operator which computes minimal fixpoint solutions of systems of equations over processes. As model for processes we consider finite-state transition systems modulo Milner's observational congruence and we define an operational semantics for the process algebra. Over such a generic algebra we show the following. We provide a syntactical characterization (allowing as many terms as possible) for the equations involved in recursion operators, which guarantees that transition systems generated by the operational semantics are indeed finite-state. Vice-versa we show that every process admits a specification in terms of such a restricted form of recursion. We then present an axiomatization which is ground-complete over such a restricted signature: an equation can be derived from the axioms between closed terms exactly when the corresponding finite-state transition systems are observationally congruent. Notably, in presenting such an axiomatization, we also show that the two standard axioms of Milner for weakly unguarded recursion can be expressed by using just a single axiom.

1 Introduction

The problem of developing a sound and complete axiomatization for a weak form of bisimulation (abstracting from internal τ activities) over a process algebra expressing finite-state processes with both guarded and (weakly and fully) unguarded recursion has been solved by Robin Milner [16]. His solution has been developed in the context of a basic process algebra (basic CCS) made up of visible prefix $a.t$, silent prefix $\tau.t$, summation $t' + t''$ and recursion $recX.t$ (based on least transition system solution), whose model is assumed to be finite-state transition systems modulo observational congruence (rooted weak bisimulation). Such a solution is crucially based on three axioms: one for fully unguarded recursion

$(FUng)$ $recX.(X + t) = recX.t$

and two for weakly unguarded recursion

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

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

The idea is that by means of the three axioms above we are able to turn each (weakly or fully) unguarded process algebraic term into an equivalent guarded one. Then the proof of completeness just works on normal forms where recursion is assumed to be guarded, i.e. it is shown that if two guarded terms are equivalent then they can be equated by the axiomatization. This is done by exploiting the two crucial axioms

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

However Milner's result is crucially based on the fact that the signature of the process algebra under consideration is very simple. For example if we extend the signature to full CCS (by e.g. considering parallel composition and restriction), we have that the axioms above 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 above. An example is the following:

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

where "$|$" and "$\setminus$" 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 the axiomatization to the equivalent term $recX.\tau.X$ or to $\tau.\underline{0}$. The problem is that, since the process above produces unguarded recursion (a loop with only τ transitions in the transition system), we cannot apply the folding axiom $(Fold)$. We should first remove unguarded recursion, but the three axioms $(FUng)$, $(WUng1)$, $(WUng2)$ only work with the restricted signature (which does not include the parallel and restriction operators).

In this paper we consider a generic process algebra of which the standard process algebras ACP, CCS and CSP are subalgebras of reduced expressions. More precisely such an algebra is an extension of the algebra TCP [1, 2] (which extends ACP by including successful termination ϵ and prefixing à la CCS) with a recursion operator $\langle X|E\rangle$ which computes minimal fixpoint solutions of systems of equations (denoted by $E = \{X = t_X, Y = t_Y, \dots\}$) over processes and consider an initial variable X among variables V defined by the system of equations E. Such an operator (which extends the similar operator introduced in [8] with the possibility of nesting recursion operators inside recursion operators) 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). As we will see, such an algebra, called TCP+REC, 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).

As model for processes we consider finite-state transition systems modulo Milner's observational congruence and we define an operational semantics for such a process algebra.

In order to guarantee that transition systems generated by the operational semantics are indeed finite-state, we provide a syntactical constraint for the systems of equations $E = E(V)$ involved in recursion operators $\langle X|E\rangle$. Such a constraint is similar to that considered in [9]: in essence we disallow variables in V occurring in the right-hand side of equations in E (that are bound by the $\langle X|E\rangle$ operator) to be in the scope of static operators like hiding, restriction, relabeling and parallel composition or in the left-hand side of a sequencing operator. For example $\langle X|\{X = \tau_I(a.X)\}\rangle$ for any hiding set I, which produces an infinite-state transition system, is a term rejected by the constraint that we consider. Note however that recursion can be included in the scope of static operators (or in the left-hand side of sequencing) as in the case of the CCS term $((recX.a.X)\,|\,(recX.a.X))\backslash a$ shown before (it is simple to express such a term in terms of our generic process algebra by using ACP parallel, hiding and restriction). We also show that the syntactical constraint that we propose is somehow the weakest: if a (reachable) variable which is bound by an outer recursion operator occurs in the scope of a static operator or in the lefthand-side of sequencing then it produces an infinite-state transition system. We call TCP+REC$_f$ the process algebra which extends TCP with the recursion operator $\langle X|E\rangle$, where E satisfies the constraint above.

Vice-versa we show that in the considered context of finite-state models every process admits a specification in terms of TCP+REC$_f$.

The main result of the paper is the introduction of an axiomatization that is ground-complete over the signature of TCP+REC$_f$: an equation can be derived from the axioms between closed terms exactly when the corresponding finite-state transition systems are observationally congruent.

This axiomatization is based on the introduction of the new axiom

$$\tau_I(\langle X|X = t\rangle) = \langle X|X = \tau_I(t)\rangle$$

which allows the hiding operator (the only static operator which may generate unguarded recursion) to be exchanged with the recursion operator. We will show that by using such a crucial axiom, that was previously considered also in [14] (where the author just showed it to be sound), it is possible to achieve completeness in the finite-state case when static operators are considered, thus extending Milner's result. The main idea is that, by means of this axiom, we can first move the hiding operator inside recursion and more generally outside-in traversing the whole syntactic structure of the term considered (so to get the effect of hiding on the actions syntactically occurring in the term), and then (by applying it in the reversed way) inside-out again. Supposing that we are turning the term into normal form (essentially basic CCS where recursion is guarded) by means of syntactical induction, once we have done the procedure above we can apply Milner's rule for unguarded recursion in the term inside the hiding operator, thus getting a term in normal form on which the hiding operator has no longer any effect. As a consequence we can get rid of it like we do with any other static operator by using the Fold axiom.

Notably, in the axiomatization that we present we also make use of the following result that we introduce here. The two axioms of Milner for getting rid of weakly unguarded recursion presented above (WUng1 and WUng2) can be equivalently expressed by means of the following single axiom:

$$\langle X|X = \tau.(X + t) + s\rangle = \langle X|X = \tau.(t + s)\rangle$$

The paper is structured as follows. In Sect. 2 we present the model of processes that we consider (finite state transition systems) and the notion of observational congruence. In Sect. 3 we present the process algebra TCP and its operational semantics. In Sect. 4 we introduce the recursion operator, its operational semantics, the considered syntactical constraint over sets of equations and the full syntax of TCP+REC$_f$. Moreover we prove that: (i) TCP+REC$_f$ terms produce finite-state transitions systems only, (ii) the constraint that we consider is the weakest and (iii) every finite-state transition system can be expressed in terms of a TCP+REC$_f$ term. In Sect. 5 we present the axiomatization and we show that it is sound and ground-complete for observational congruence over the TCP+REC$_f$ signature. Sect. 6 concludes the paper.

1.1 Acknowledgements

We thank Rob van Glabbeek (NICTAustralia) and the anonymous reviewers for their useful remarks and suggestions. The replacement of the two axioms of Milner for weakly guarded recursion by just one axiom was also found independently by Rob van Glabbeek, but never published.

2 Finite Behaviours

In this paper, we consider finite behaviours: the model of finite state transition systems modulo Milner's observational congruence.

Definition 1 (Transition-system space) A *transition-system space* over a set of labels L is a set S of *states*, equipped with one ternary relation $\rightarrow$ and one subset $\downarrow$:

1. $\rightarrow \subseteq S \times L \times S$ is the set of *transitions*;
2. $\downarrow \subseteq S$ is the set of *terminating* or *final* states.

The notation $s \overset{\alpha}{\rightarrow} t$ is used for $(s, \alpha, t) \in \rightarrow$ and $s \downarrow$ for $s \in \downarrow$.

Here, we will always assume the sets S and L are finite, and the set of labels will consist of a set of actions A and a special label $\tau \notin A$.

In the remainder, assume that $(S, L, \rightarrow, \downarrow)$ is a transition-system space. Each state $s \in S$ can be identified with a transition system that consists of all states and transitions reachable from s. The notion of reachability is defined as usual.

Definition 2 (Weak Bisimilarity) Define $s \Rightarrow t$ if there is a sequence of 0 or more τ-steps from s to t. A symmetric binary relation R on the set of states S of a transition-system space is a *weak bisimulation* relation if and only if the following so-called *transfer conditions* hold:

1. for all states $s, t, s' \in S$, whenever $(s,t) \in R$ and $s \xrightarrow{\alpha} s'$ for some $\alpha \in L$, then either $\alpha = \tau$ and $(s', t) \in R$ or there are states t^*, t'', t' such that $t \Rightarrow t^* \xrightarrow{a} t'' \Rightarrow t'$ and $(s', t') \in R$;
2. whenever $(s,t) \in R$ and $s \downarrow$ then there is a state t^* such that $t \Rightarrow t^* \downarrow$;

Two transition systems $s, t \in S$ are *weak bisimulation equivalent* or *weakly bisimilar*, notation $s \leftrightarrow_w t$, if and only if there is a weak bisimulation relation R on S with $(s,t) \in R$.

The pair (s,t) in a weak bisimulation R *satisfies the root condition* if whenever $s \xrightarrow{\tau} s'$ there are states t'', t' such that $t \xrightarrow{\tau} t'' \Rightarrow t'$ and $(s', t') \in R$. Two transition systems $s, t \in S$ are *rooted weak bisimulation equivalent*, *observationally congruent* or *rooted weakly bisimilar*, notation $s \leftrightarrow_{rw} t$, if and only there is a weak bisimulation relation in which the pair (s,t) satisfies the root condition.

3 Process Algebra

We consider the process algebra TCP (Theory of Communicating Processes), introduced in [1] and completely worked out in [2], of which the standard process algebras ACP, CCS and CSP are subalgebras of reduced expressions.

Our theory has two parameters: the set of actions A, and a *communication function* $\gamma : A \times A \to A$. The function γ is partial, commutative and associative. The signature elements are the following. Constant δ denotes *inaction* (or deadlock), and is the neutral element of alternative composition: process δ cannot execute any action, and cannot terminate. Constant ϵ denotes the *empty process* or *skip* and is the neutral element of sequential composition: process ϵ cannot execute any action, but terminates successfully. For each $a \in A$, there is the unary *prefix operator* $a.\_$: process $a.x$ executes action a and then proceeds as x. There is the additional prefix operator $\tau.\_$. Here, $\tau \notin A$ is the *silent step*, that cannot be observed directly. Binary operator $+$ denotes *alternative composition* or choice: process $x + y$ executes either x or y, but not both (the choice is resolved upon execution of the first action). Binary operator $\cdot$ denotes *sequential composition*: having sequential composition as a basic operator, makes it necessary to have a difference between successful termination (ϵ) and unsuccessful termination (δ). Sequential composition is more general than action prefixing. Binary operator $\|$ denotes *parallel composition*. In order to give a finite axiomatization of parallel composition, there are two variations on this operator, the auxiliary operators $\mathbin{|\!\lfloor}$ (left-merge) and $\mid$ (synchronization merge). In the parallel composition $x \parallel y$, the separate components may execute a step independently (denoted by $x \mathbin{|\!\lfloor} y$ resp. $y \mathbin{|\!\lfloor} x$), or they may synchronize in executing a communication action (when they can execute actions for which γ is defined), or they may terminate together (the last two possibilities given by $x \mid y$). Unary operator ∂_H denotes *encapsulation* or *restriction*, for each $H \subseteq A$: actions from H are blocked, cannot be executed. Unary operator τ_I denotes *abstraction* or *hiding*, for each $I \subseteq A$: actions from I are turned into τ, and are thus made unobservable. Unary operator ρ_f denotes *renaming* or *relabeling*, for each $f : A \to A$.

In the following we will use meta-variables x, y to range over processes of our process algebra, i.e. finite-state transition-systems possibly denoted via a term over the signature of the algebra, a, b, c to range over A and α to range over $A \cup \{\tau\}$.

We turn the set of closed terms (i.e. terms containing no variables) over the signature of the algebra into a transition-system space by providing so-called operational rules. See Table 1. States in the transition-system space are denoted by closed terms over the signature. These rules give rise to a finite transition system, without cycles, for each closed term.

$\epsilon\downarrow$	$\alpha.x \xrightarrow{\alpha} x$		
$\dfrac{x \xrightarrow{\alpha} x'}{x+y \xrightarrow{\alpha} x'}$	$\dfrac{y \xrightarrow{\alpha} y'}{x+y \xrightarrow{\alpha} y'}$	$\dfrac{x\downarrow}{x+y\downarrow}$	$\dfrac{y\downarrow}{x+y\downarrow}$
$\dfrac{x \xrightarrow{\alpha} x'}{x \cdot y \xrightarrow{\alpha} x' \cdot y}$	$\dfrac{x\downarrow, y \xrightarrow{\alpha} y'}{x \cdot y \xrightarrow{\alpha} y'}$	$\dfrac{x\downarrow, y\downarrow}{x \cdot y\downarrow}$	
$\dfrac{x \xrightarrow{a} x', y \xrightarrow{b} y', \gamma(a,b)=c}{x \parallel y \xrightarrow{c} x' \parallel y'}$	$\dfrac{x\downarrow, y\downarrow}{x \parallel y\downarrow}$	$\dfrac{x \xrightarrow{\alpha} x'}{x \parallel y \xrightarrow{\alpha} x' \parallel y}$	$\dfrac{y \xrightarrow{\alpha} y'}{x \parallel y \xrightarrow{\alpha} x \parallel y'}$
$\dfrac{x \xrightarrow{a} x', y \xrightarrow{b} y', \gamma(a,b)=c}{x \mid y \xrightarrow{c} x' \parallel y'}$	$\dfrac{x\downarrow, y\downarrow}{x \mid y\downarrow}$	$\dfrac{x \xrightarrow{\alpha} x'}{x \mathbin{\lfloor\!\lfloor} y \xrightarrow{\alpha} x' \parallel y}$	
$\dfrac{x \xrightarrow{\tau} x', x' \mid y \xrightarrow{\alpha} z}{x \mid y \xrightarrow{\alpha} z}$	$\dfrac{y \xrightarrow{\tau} y', x \mid y' \xrightarrow{\alpha} z}{x \mid y \xrightarrow{\alpha} z}$	$\dfrac{x \xrightarrow{\tau} x', x' \mid y\downarrow}{x \mid y\downarrow}$	$\dfrac{y \xrightarrow{\tau} y', x \mid y'\downarrow}{x \mid y\downarrow}$
$\dfrac{x \xrightarrow{\alpha} x', \alpha \notin H}{\partial_H(x) \xrightarrow{\alpha} \partial_H(x')}$	$\dfrac{x\downarrow}{\partial_H(x)\downarrow}$	$\dfrac{x \xrightarrow{\alpha} x', \alpha \notin I}{\tau_I(x) \xrightarrow{\alpha} \tau_I(x')}$	$\dfrac{x \xrightarrow{a} x', a \in I}{\tau_I(x) \xrightarrow{\tau} \tau_I(x')}$
$\dfrac{x\downarrow}{\tau_I(x)\downarrow}$	$\dfrac{x \xrightarrow{a} x'}{\rho_f(x) \xrightarrow{f(a)} \rho_f(x')}$	$\dfrac{x \xrightarrow{\tau} x'}{\rho_f(x) \xrightarrow{\tau} \rho_f(x')}$	$\dfrac{x\downarrow}{\rho_f(x)\downarrow}$

Table 1. Deduction rules for TCP.

We can provide an axiomatization that is *ground-complete*, i.e. an equation can be derived from the axioms between two closed terms exactly when the corresponding transition systems are observationally congruent. The basic set of axioms is presented in Table 2.

This process algebra is *generic*, in the sense that most features of commonly used process algebras can be embedded in it. In the following, we made use of [13, 14] and [4].

We consider a subtheory corresponding to CCS, see [17]. This is done by omitting the signature elements $\epsilon, \cdot, \mathbin{\lfloor\!\lfloor}, \mid$. Next, we specialize the parameter set A by separating it into three parts: a set of names $\mathcal{A}$, a set of co-names $\bar{\mathcal{A}}$ and

$x + y = y + x$	A1	$x \parallel y = x \mathbin{\|\!\_} y + y \mathbin{\|\!\_} x + x \mid y$	M
$(x + y) + z = x + (y + z)$	A2		
$x + x = x$	A3	$\delta \mathbin{\|\!\_} x = \delta$	LM1
$(x + y) \cdot z = x \cdot z + y \cdot z$	A4	$\epsilon \mathbin{\|\!\_} x = \delta$	LM2
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	A5	$\alpha.x \mathbin{\|\!\_} y = \alpha.(x \parallel y)$	LM3
$x + \delta = x$	A6	$(x + y) \mathbin{\|\!\_} z = x \mathbin{\|\!\_} z + y \mathbin{\|\!\_} z$	LM4
$\delta \cdot x = \delta$	A7		
$\epsilon \cdot x = x$	A8	$x \mid y = y \mid x$	SM1
$x \cdot \epsilon = x$	A9	$\delta \mid x = \delta$	SM2
$(\alpha.x) \cdot y = \alpha.(x \cdot y)$	A10	$\epsilon \mid \epsilon = \epsilon$	SM3
		$a.x \mid b.y = c.(x \parallel y)$ if $\gamma(a, b) = c$	SM4
$\partial_H(\delta) = \delta$	D1	$a.x \mid b.y = \delta$ otherwise	SM5
$\partial_H(\epsilon) = \epsilon$	D2	$a.x \mid \epsilon = \delta$	SM6
$\partial_H(a.x) = \delta$ if $a \in H$	D3	$(x + y) \mid z = x \mid z + y \mid z$	SM7
$\partial_H(\alpha.x) = \alpha.\partial_H(x)$ otherwise	D4		
$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$	D5	$\rho_f(\delta) = \delta$	RN1
		$\rho_f(\epsilon) = \epsilon$	RN2
$\tau_I(\delta) = \delta$	TI1	$\rho_f(a.x) = f(a).\rho_f(x)$	RN3
$\tau_I(\epsilon) = \epsilon$	TI2	$\rho_f(\tau.x) = \tau.\rho_f(x)$	RN4
$\tau_I(a.x) = \tau.\tau_I(x)$ if $a \in I$	TI3	$\rho_f(x + y) = \rho_f(x) + \rho_f(y)$	RN5
$\tau_I(\alpha.x) = \alpha.\tau_I(x)$ otherwise	TI4		
$\tau_I(x + y) = \tau_I(x) + \tau_I(y)$	TI5		
$\alpha.\tau.x = \alpha.x$	T1	$\tau.x + x = \tau.x$	T2
$\alpha.(\tau.x + y) = \alpha.(\tau.x + y) + \alpha.x$	T3	$\tau.x \mid y = x \mid y$	T4

Table 2. Axioms of TCP.

a set of communications $\mathcal{A}^*$ such that for each $a \in \mathcal{A}$ there is exactly one $\bar{a} \in \bar{\mathcal{A}}$ and exactly one $a^* \in \mathcal{A}^*$. The communication function γ is specialized to having as the only defined communications $\gamma(a, \bar{a}) = \gamma(\bar{a}, a) = a^*$, and then the CCS parallel composition operator $\mid_{CCS}$ can be defined by the formula

$$x \mid_{CCS} y = \tau_{\mathcal{A}^*}(x \parallel y).$$

We consider a subtheory corresponding to ACP_τ, see [7]. This is done by defining, for each $a \in A$, a new constant a by $a = a.\epsilon$, and then omitting the signature elements $\epsilon, ., \rho_f$.

We consider a subtheory corresponding to CSP, see [15]. The *non-deterministic choice* operator $\sqcap$ can be defined by

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

but the *external choice* operator $\Box$ cannot be defined directly, as possible non-determinism is removed at the start of the process. It can be axiomatized as shown by Brookes in [10]. The parameter set A is specialized into two parts: a set of names $\mathcal{A}$ and a set of communications $\mathcal{A}^*$ such that for each $a \in \mathcal{A}$ there is exactly one $a^* \in \mathcal{A}^*$. The communication function γ is specialized to having as the only defined communications $\gamma(a, a) = a^*$, and further, we use the renaming function f that has $f(a^*) = a$. Then, the CSP parallel composition operator $\|_S$, parametrized by a set of names $S \subseteq \mathcal{A}$, can be defined by the formula

$$x \|_S y = \rho_f(\partial_S(x \| y)).$$

4 Recursion

We proceed to define recursion in our setting, in order to obtain also finite-state transition systems with cycles.

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 transition systems $\{y_X \mid X \in V\}$ such that the equations of $E(V)$ correspond to equivalent transition systems, 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.

Let t be a term containing a variable X. We call an occurrence of X in t *guarded* if this occurrence of X is in the scope of an action prefix operator (not τ prefix) and *not* in the scope of an abstraction operator.

We call a recursive specification *guarded* if all occurrences of all its variables in the right-hand sides of all its equations are guarded or it can be rewritten to such a recursive specification using the axioms of the theory and the equations of the specification.

Now, in the models obtained by adding rules for recursion to the operational semantics given above, and dividing out one of the congruence relations strong bisimulation, or observational congruence, guarded recursive specifications have unique solutions, so we can talk about *the* process given by a guarded recursive specification. On the other hand, unguarded recursive specifications usually have several solutions. Thus, the specification $\{X = X\}$ will have every transition system as a solution, and the specification $\{X = \tau.X\}$ will have multiple solutions under observational congruence, as any transition system with a τ-step as only initial step will satisfy this equation.

The process algebras ACP, CCS and CSP handle this situation 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, is the process δ, 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. Finally, also in CSP a solution will be chosen, 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 TCP with such a process, see [2], based on [3]).

Here we will introduce in TCP the possibility of performing (not guarded) recursive specifications by means of an 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, yields the least transitions in the generated transition system. Note that our approach also encompasses recursive specifications in ACP which are usually assumed to be guarded. The extended signature gives rise to a process algebra that we call TCP+REC.

More precisely, the set of terms of TCP+REC is generated by the following syntax:

$$t ::= \delta \mid \epsilon \mid a.t \mid \tau.t \mid t+t \mid t \cdot t \mid t \parallel t \mid t \mathbin{\lfloor\!\lfloor} t \mid t \,|\, t \mid \partial_H(t) \mid \tau_I(t) \mid \rho_f(t) \mid X \mid \langle X|E\rangle$$

where $E = E(V)$ is a set of equations $E = \{X = t \mid X \in V\}$.

Note that terms t included in recursive specifications are again part of the same syntax, i.e. they may include again recursive specifications. In the following we will use t_X to denote the term defining variable X (i.e. $X = t_X$) in a given recursive specification.

As usual, in the following, we will use, as terms representing processes, closed terms over the syntax above. In the setting above a closed term is a term in which every variable X occurs in the scope of a binding recursive specification $E(V)$ such that $X \in V$. Note that the binding recursive specification may not be the one that directly includes the equation which contains the occurrence of X in the right-hand term, but X may be bound by an outer recursive specification, as e.g. in:

$$\langle\, X \mid \{\, X = a.\langle Y|\{Y = X + Y\}\rangle\,\}\,\rangle$$

Table 3 provides deduction rules for recursive specifications. Such rules are similar to those in [11], but we have the additional possibility of nesting recursion operators inside recursion operators. They come down to looking upon $\langle X|E\rangle$ as the process $\langle t_X|E\rangle$, which is defined as follows.

Definition 3 Given a recursive specification $\langle X|E\rangle$ with the syntax above, where $E = E(V)$, we define $\langle t_X|E\rangle$ to be t_X where, for all $Y \in V$, all free occurrences of Y in t_X are replaced by $\langle Y|E\rangle$.

Note that in $\langle t_X|E\rangle$ we replace not only variables $Y \in V$ occurring directly in t_X, but even Y occurring freely inside inner recursive specifications, e.g. in

$$\langle\ a.\langle Y|\{Y = X + Y\}\rangle \ \mid\ \{\, X = a.\langle Y|\{Y = X + Y\}\rangle\,\}\ \rangle$$

variable X of $a.\langle Y|\{Y = X+Y\}\rangle$ is replaced by $\langle X|\{\, X = a.\langle Y|\{Y = X+Y\}\rangle\,\}\rangle$ yielding:

$$a.\langle\ Y\ \mid\ \{Y = \langle X|\{\, X = a.\langle Y|\{Y = X + Y\}\rangle\,\}\rangle + Y\}\ \rangle$$

In order to remain in the setting of processes with a finite-state model we now consider a restricted syntax for constants $\langle X|E\rangle$ which guarantees that transition systems generated by the operational rules are indeed finite-state.

$$\frac{\langle t_X|E\rangle \xrightarrow{\alpha} y}{\langle X|E\rangle \xrightarrow{\alpha} y} \quad \frac{\langle t_X|E\rangle \downarrow}{\langle X|E\rangle \downarrow}$$

Table 3. Deduction rules for recursion.

Definition 4 Let E be a recursive specification over a set of variables V. We call E *essentially finite state* if E has only finitely many equations and all variables in all right-hand sides of all equations of E do not occur in the scope of one of the operators $\|, \lfloor\!\lfloor, \mid, \partial_H, \tau_I, \rho_f$ or on the left-hand side of the operator $\cdot$. We call E *regular* if E has only finitely many equations and each equation is of the form

$$X = \sum_{1 \leq i \leq n} \alpha_i.X_i + \{\epsilon\},$$

where an empty sum stands for δ and the ϵ summand is optional, for certain $n \in \mathbb{N}, \alpha_i \in A \cup \{\tau\}, X_i \in V$. It is immediate that every regular recursive specification is essentially finite-state.

Now it is a well-known fact that each finite state process allows a regular recursive specification. But also in the other direction, every process specified by a term including essentially finite state recursive specifications only has finitely many states in the transition system generated by the operational rules.

Proposition 5 Given a term t such that every recursive specification E included in t is essentially finite state. Then the transition system generated by the operational rules has only finitely many states.

Proof See [5] for a complete proof. □

The following proposition shows that the definition of essentially finite state does not disregard unnecessarily terms which generate finite-state transition systems. In the proposition we assume that an occurrence of a variable X is reachable, if, once such an occurrence of X is replaced by $a_X.\delta$, there is a path that leads to the execution of the action a_X.

Proposition 6 Let t be a term that includes a recursion $\langle X|E\rangle$ such that the recursive specification E has finitely many equations but is not essentially finite state. If one of the occurrences of variables in V which violate the condition, i.e. which are in the scope of one of the operators $\|, \lfloor\!\lfloor, \mid, \partial_H, \tau_I, \rho_f$ or on the left-hand side of the operator $\cdot$, is reachable from $\langle X|E\rangle$, then t has infinitely many states.

Proof It is just a matter of showing that, since there is a variable which violates the condition and is reachable, then there is a loop where every time a new copy of the static operator is produced (or of the righthand-side of the sequencing), hence the transition system produced is infinite. □

$\langle X \mid E \,\tilde{\cup}\, \{Y = t\}\rangle = \langle X \mid E\{\langle Y \mid Y = t\rangle / Y\}\rangle$ if $X \neq Y$	Dec
$\langle X \mid X = t\rangle = t\{\langle X \mid X = t\rangle / X\}$	Unf
$y = t\{y/X\} \quad \Rightarrow \quad y = \langle X \mid X = t\rangle$ if $X = t$ guarded	Fold
$\langle X \mid X = X + t\rangle = \langle X \mid X = t\rangle$	Ung
$\langle X \mid X = \tau.(X + t) + s\rangle = \langle X \mid X = \tau.(t + s)\rangle$	WUng
$\tau_I(\langle X \mid X = t\rangle) = \langle X \mid X = \tau_I(t)\rangle$	Hid

Table 4. Axioms for recursion.

In the rest of this paper, we will consider the process algebra TCP+REC$_f$ obtained by extending the signature of TCP with essentially finite state recursive specifications, i.e. we consider closed terms in the syntax above, where we additionally require that every recursive specification included is essentially finite-state. Together Table 1 and Table 3 provide a transition system space over the signature of TCP+REC$_f$.

5 Axiomatization

Now we will present a sound axiomatization which is ground-complete for the process algebra TCP+REC$_f$. The axioms in Table 2 together with the axioms in Table 4 form such an axiomatization. In the axioms of Table 4 we use the usual operation $\{t/X\}$ for expressing syntactical replacement of a closed term t for every free occurrence of variable X. Such an operation can be applied to a term t' or to the righthand-side of all equations in a recursive specification $E = E(V)$ such that $X \notin V$ by writing $t'\{t/X\}$ and $E\{t/X\}$, respectively. Moreover the symbol $\tilde{\cup}$ stands for disjoint union. Note that the axioms in Table 4 are *axiom schemes*: we have these axioms for each possible term t.

The axiom Dec is used to decompose recursive specifications E made up of multiple (finitely-many) equations into several recursive specifications made up of single equations. For example the process

$$\langle\ X \mid \{X = a.X + b.Y, Y = c.X + d.Y\}\ \rangle$$

is turned into

$$\langle\ X \mid \{X = a.X + b.\langle Y \mid \{Y = c.X + d.Y\}\rangle\}\ \rangle$$

The unfolding axiom (Unf) is Milner's standard one. In ACP, where usually there is no explicit recursion operator, it corresponds to the *Recursive Definition Principle*: it states that the constant $\langle X \mid E\rangle$ is a solution of the recursive specification E. Thus, each recursive specification has a solution. The folding axiom (Fold) is Milner's standard one: it states that if y is a solution for X in E, and E is guarded, then $y = \langle X \mid E\rangle$. In ACP, where usually there is no explicit recursion operator, it corresponds to the *Recursive Specification Principle*: it says that each guarded recursive specification has at most one solution.

Axioms Ung, WUng, Hid are used to deal with unguarded specifications. Ung, which is the same as in Milner's axiomatization, is the axiom that deals with variables not in the scope of any prefix operator (fully unguarded recursion). WUng and Hid are instead needed to get rid of weakly unguarded recursion. As far as WUng is concerned, it gets rid of weakly unguarded recursion arising from just prefixing and summation. It is easy to see that it replaces the two axioms of Milner:

$$\langle X|X = \tau.X + t\rangle = \langle X|X = \tau.t\rangle$$
$$\langle X|X = \tau.(X + t) + s\rangle = \langle X|X = \tau.X + t + s\rangle$$

The first one is obtained from (WUng) by just taking $t = \delta$. The second one is obtained from (WUng) as follows:

$$\langle X|X = \tau.(X + t) + s\rangle = \langle X|X = \tau.(t + s)\rangle$$

by directly applying (WUng) and then

$$\langle X|X = \tau.(t + s)\rangle = \langle X|X = \tau.X + t + s\rangle$$

by applying (WUng) where we take $s = t + s$ and $t = \delta$.

As explained in the introduction, the axiom (Hid) is used to get rid of weak unguardedness generated by the hiding operator. It allows to turn a term into such a form that the standard axioms for weak unguardedness can be used (see the proof of the following Proposition 8).

Note that if we want to derive a ground-complete axiomatization in a setting where no construct is added for recursion, as usually done in the context of the ACP process algebra (so we just have closed terms over the syntax of TCP and we just consider sets of recursion equations over this syntax), then in order to achieve the effect of our axiom Hid we have to add a much more complex set of conditional equations called CFAR (Cluster Fair Abstraction Rule) introduced in [18]. CFAR is a generalisation of the KFAR (Koomen's Fair Abstraction Rule) introduced in [6].

Proposition 7 The axiomatization formed by the axioms in Table 2 and by the axioms in Table 4 is sound for the model of transition systems modulo observational congruence generated by the rules in Tables 1 and 3.

Proof Most of the axioms are standard. See [14] for the axiom (Hid). □

Proposition 8 The axiomatization formed by the axioms in Table 2 and by the axioms in Table 4 is ground-complete for the model of transition systems modulo observational congruence generated by the rules in Tables 1 and 3.

Proof We show, by structural induction over the syntax of (possibly open) terms t of TCP+REC$_f$ such that free variables do not occur in the scope of one of the operators $\|, \mathbin{\|\!\_}, |, \partial_H, \tau_I, \rho_f$ or on the left-hand side of the operator $\cdot$, that t can be turned into normal form, where normal forms are defined as follows. A term is normal form if it is made up of only δ,ϵ,X,$a.t'$,$\tau.t'$,$t' + t''$ and $\langle X|E\rangle$, where E is guarded and contains one equation only. Proving this yields ground-completeness; this because normal forms are like terms of basic CCS (with the only difference that we have two non equivalent kinds of terminating processes δ and ϵ, instead of just one) and completeness over such terms has been proved by Milner (since we do not have $\cdot$ or $\|$ operators in normal forms the presence of the two ways of termination does not change the proof).

The base cases of the induction ($t \equiv \delta$ or $t \equiv \epsilon$ or $t \equiv X$) are trivial because they are in normal form already.

The inductive cases of the induction are the following ones:

- if $t \equiv a.t'$ or $t \equiv \tau.t'$ or $t \equiv t' + t''$ then t can be turned into normal form by directly exploiting the inductive argument over t' and t''.
- if $t \equiv t' \parallel t''$ or $t \equiv t' \lfloor\!\lfloor t''$ or $t \equiv t' \,|\, t''$ or $t \equiv \partial_H(t')$ or $t \equiv \rho_f(t')$, then we can turn t into normal form as follows. By exploiting the inductive argument over t' and t'', and by observing that t cannot include free variables, we know that t has a finite transition system. Let $t_1 \ldots t_n$ be the states of the transition system of t, $t_n \equiv t$. It can be easily seen that, for each $i \in \{1 \ldots n\}$, there exist m_i, $\{\alpha^i_j\}_{j \leq m_i}$ (denoting actions), $\{k^i_j\}_{j \leq m_i}$ (denoting natural numbers) s.t. we can derive $t_i = \sum_{j \leq m_i} \alpha^i_j.t_{k^i_j} + \{\epsilon\}$. Hence we can characterize the behavior of t by means of a set of equations similarly as in [16]. Moreover, similarly as for the unique solution of equations theorem of [16], we have that there is a term t''' in normal form such that we can derive $t''' = t_n \equiv t$. This can be shown as follows. For each i, from 1 to n, we do the following. If i is such that $\exists j \leq m_i : k^i_j = i$ we have, by applying $Fold$, that $t_i = \langle X | X = \sum_{j \leq m_i : k^i_j \neq i} \alpha^i_j.t_{k^i_j} + \sum_{j \leq m_i : k^i_j = i} \alpha^i_j.X + \{\epsilon\}\rangle$. Note that axiom $Fold$ is applicable because, by exploiting the inductive argument, t' and t'' are in normal form and contain guarded recursion only, hence (since the operators considered cannot turn visible actions into τ ones) every cycle in the derived transition system contains at least a visible action. Then we replace each subterm t_i occurring in the equations for $t_{i+1} \ldots t_n$ with its equivalent term. When, in the equation for $t_n \equiv t$, we have replaced t_{n-1}, we are done.
- if $t \equiv t' \cdot t''$ then t is turned into normal form similarly as in the previous item. The only difference is that t'' may include free variables. Supposing that $c(t'')$ denotes the closed term obtained from t'' by replacing each free occurrence of a variable X by $a_X.\delta$, the procedure for obtaining the normal form from $t \equiv t' \cdot t''$ is the same followed for $t' \cdot c(t'')$ with the procedure of the previous item (note that when a state is reached such that any actions a_X corresponding to free variables X in t'' are immediately executable, the $\cdot$ operator has disappeared already).
- if $t \equiv \langle X | E \rangle$ then t is turned into normal form by first exploiting the inductive argument over terms t_Y where $Y \in V$, assuming $E = E(V)$, and then by applying axioms Ung and $WUng$ to get rid of generated unguarded recursion as in the standard approach of Milner (after decomposing multi-variable recursion with axiom Dec).
- if $t \equiv \tau_I(t')$ then t is turned into normal form as follows. By exploiting the inductive argument over t', we consider term t'' which is obtained by turning t' into normal form. Observe that t' (hence t'') cannot include free variables and that it has a finite transition system.

  We first show, by structural induction on term t'', that $\tau_I(t'')$ can be turned into $\tau_I(t''')$, where t''' is obtained from t'' by syntactically replacing each occurrence of an action in I with τ.

The base cases of the induction ($t'' \equiv \delta$ or $t'' \equiv \epsilon$ or $t'' \equiv X$) are trivial because no action in I is included.
The inductive cases of the induction are the following ones:

- if $t'' \equiv a.t''_1$ then we have the following two cases:
  - if $a \in I$ then, since $\tau_I(a.t''_1)$ can be turned into $\tau.\tau_I(t''_1)$, which by induction hypothesis can be turned into $\tau.\tau_I(t'''_1)$, with t'''_1 such that each occurrence of an action in I is replaced with τ, we obtain term t''' by the final transformation into $\tau_I(\tau.t'''_1)$.
  - if $a \notin I$ then it is a repetition of the previous case where a is not turned into τ.
- if $t'' \equiv \tau.t''_1$ it is a repetition of the previous item where τ is not affected by the transformation.
- if $t'' \equiv t''_1 + t''_2$ then, since $\tau_I(t''_1 + t''_2)$ can be turned into $\tau_I(t''_1) + \tau_I(t''_2)$, which by induction hypothesis can be turned into $\tau_I(t'''_1) + \tau_I(t'''_2)$, with t'''_1 and t'''_2 such that each occurrence of an action in I is replaced with τ, we obtain term t''' by the final transformation into $\tau_I(t'''_1 + t'''_2)$.
- if $t'' \equiv \langle X|\{X = t''_1\}\rangle$ then, since $\tau_I(\langle X|\{X = t''_1\}\rangle)$ can be turned into $\langle X|\{X = \tau_I(t''_1)\}\rangle$ by means of axiom Hid, which by induction hypothesis can be turned into $\langle X|\{X = \tau_I(t'''_1)\}\rangle$, with t'''_1 such that each occurrence of an action in I is replaced with τ, we obtain term t''' by the final transformation into $\tau_I(\langle X|\{X = t'''_1\}\rangle)$ by means again of axiom Hid.

Then we use Ung and $WUng$ to get rid of generated unguarded recursion into t''' as in Milner's standard approach, thus getting a guarded t''''.
Finally we consider $\tau_I(t'''')$ and we apply the same technique as for, e.g., the $\|$ operator to turn it into normal form (exploiting the fact that t'''' is guarded, finite state and does not include free variables). In particular now we can do that because the application of the hiding operator has no effect on labels of transitions, hence it cannot generate cycles made up of only τ actions when the semantics is considered. $\square$

6 Conclusion

We just make some commentary about future work. First of all, we claim that the axiomatization that we presented is complete over all terms in the signature of TCP plus the recursion operator $\langle X|E\rangle$ (without syntactical restriction) which are finite state, i.e. we can include also terms with variables bound by an outer recursion operator that are in the scope of static operators (or in the lefthand-side of a sequence) provided that they are not reachable. Moreover, we plan to rebuild the whole machinery we showed here in the case of branching bisimulation instead of considering observational congruence. In particular we claim that we can find a ground-complete axiomatization for essentially finite state behaviours modulo branching bisimulation by taking the axiomatization of [12], extending the syntax as we have done, and adding as only extra axiom our axiom (Hid).

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. Technical Report CS Report 05-18, Technische Universiteit Eindhoven, Department of Mathematics and Computer Science, 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. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985.
8. 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.
9. M. Bravetti and R. Gorrieri. Deciding and axiomatizing weak st bisimulation for a process algebra with recursion and action refinement. *ACM Transactions on Computational Logic*, 3(4):465–520, 2002.
10. S.D. Brookes. On the relationship of CCS and CSP. In J. Diaz, editor, *Proceedings ICALP'83*, number 154 in LNCS, pages 83–96. Springer Verlag, 1983.
11. R.J. van Glabbeek. Bounded nondeterminism and the approximation induction principle in process algebra. In F.J. Brandenburg, G. Vidal-Naquet, and M. Wirsing, editors, *Proceedings STACS'87*, number 247 in Lecture Notes in Computer Science, pages 336–347. Springer Verlag, 1987.
12. R.J. van Glabbeek. A complete axiomatization for branching bisimulation congruence of finite-state behaviours. In A.M. Borzyszkowski and S. Sokolowski, editors, *Proc. MFCS'93*, volume 711 of *LNCS*, pages 473–484. Springer, 1993.
13. R.J. van Glabbeek. On the expressiveness of ACP (extended abstract). In A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, Proceedings First Workshop on the *Algebra of Communicating Processes,* ACP94, Utrecht, The Netherlands, May 1994, Workshops in Computing, pages 188–217, 1994. Available at `http://boole.stanford.edu/pub/acp.ps.gz`.
14. R.J. van Glabbeek. Notes on the methodology of CCS and CSP. *Theoretical Computer Science*, 177(6):329–349, 1997.
15. C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.
16. R. Milner. A complete inference system for a class of regular behaviours. *Journal of Comput. System Sci.*, 28(3):439–466, 1984.
17. R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.
18. F.W. Vaandrager. Verification of two communication protocols by means of process algebra. Technical Report report CS-R8608, CWI Amsterdam, 1986.