

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

JOS C. M. BAETEN¹ † and MARIO BRAVETTI¹

¹ *Division of Computer Science, Technische Universiteit Eindhoven, josb@win.tue.nl*

² *Department of Computer Science, Università di Bologna, bravetti@cs.unibo.it*

Received 18 July 2007; Revised 3 April 2008

The three classical process algebras 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 (including minimal fixpoint based unguarded recursion) 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 for Milner’s observational congruence in the presence of (unguarded) recursion and static operators. More precisely, 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 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. Notably, 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 large amount of research work on process algebra carried out in the last 25 years started with the introduction of the theory of the process algebras CCS (Milner 1989a), CSP (Hoare 1985) and ACP (Bergstra and Klop 1984). In spite of conceptual similarities those process algebras were 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

† Research partially funded by EU Integrated Project Sensoria, contract n. 016004.

and originally had a denotational semantics (Brookes *et al.* 1984) that, when interpreted operationally, is not based on bisimilarity but on decorated traces; finally ACP starts from a completely different viewpoint where concurrent systems are seen, according to a purely mathematical algebraic view, as the solutions of systems of equations (axioms) over the signature of the algebra considered, and operational semantics and bisimilarity (in this case a different notion of branching bisimilarity 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.

In this paper we initially aim at pointing out such differences, which are often reflected 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 the treatment of process variables in the three different contexts the need for clarification and comparison comes out. Our study concretizes, as a first contribution of the paper, the development of a common theory of process algebra: we introduce 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. More precisely such an algebra is an extension of the algebra TCP (Baeten 2003; Baeten *et al.* 2008) (which extends ACP by including successful termination $\underline{1}$ and prefixing à la CCS) with a recursion operator $\langle X|E \rangle$ that computes the least transition relation satisfying a system of recursive equations (denoted by $E = \{X = t_X, Y = t_Y, \dots\}$) over processes and considers an initial variable X among variables V defined by the system of equations E . Such an operator (which extends the similar operator introduced in (Bergstra and Klop 1988) 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). Note that, like in CCS, the $\langle X|E \rangle$ operator evaluates the fixpoint solution for X that is minimal with respect to inclusion of the transition relation, which may not be the minimal transition system in its equivalence class, in the case some notion of equivalence is considered. As we will see, the algebra 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). 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. As an example of the last item, we show how, by using TCP+REC, we can solve the problem of producing an axiomatization which is complete over finite-state behaviours in the presence of unguarded recursion and static operators as, i.e., parallel, hiding and restriction. Such an axiomatization and the related theorems are the second and main contribution of the paper.

The problem of developing a sound and complete axiomatization for a weak form of

bisimilarity (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 (Milner 1989b). His solution has been developed in the context of a basic process algebra (basic CCS) made up of visible prefix $l.t$, where l can be a typed input a or a typed output \bar{a} , silent prefix $\tau.t$, summation $t' + t''$ and recursion $recX.t$ (based on least transition relation solution), whose model is assumed to be finite-state transition systems modulo observational congruence (rooted weak bisimilarity). Such a solution is based on three axioms: one for fully unguarded recursion

$$(FUng) \text{ } recX.(X + t) = recX.t$$

and two for weakly unguarded recursion

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

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

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 axioms

$$(Unfold) \text{ } recX.t = t\{recX.t/X\}$$

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

that express existence and uniqueness of solutions in guarded recursion specifications. To be more precise, the obtained axiomatization is shown to be complete for open terms, i.e. also for terms including free occurrences of variables X .

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:

$$((recX.a.X) | (recX.\bar{a}.X)) \setminus \{a\}$$

where “|” and “\” 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 $\tau.0$. 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). As the main contribution of the paper we show that, by using TCP+REC and by introducing an additional axiom, we are able to extend Milner's result to encompass its full signature (for terms such that finite-stateness is guaranteed).

First, we consider as model for processes 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

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 (Bravetti and Gorrieri 2002): in essence we require 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 “*serial*”, i.e. not 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 (even if it becomes finite when observational congruence is divided out). 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 $((recX.a.X)|(recX.\bar{a}.X)) \setminus \{a\}$ 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 static operators or in the lefthand-side of sequencing (and reachability is preserved by the static operators) 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 $TCP+REC_f$.

Then, we produce, as a main result of the paper, an axiomatization for $TCP+REC$ 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

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

which allows one to exchange the hiding operator (the only static operator which may generate unguarded recursion) with the recursion operator. This axiom is considered also in (van Glabbeek 1997) without the seriality condition, that however is necessary to make it sound. We will show that, by using such a crucial axiom, 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 syntactical 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:

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

Finally, we would like to explicitly note that the procedure that we use to turn TCP+REC_f terms into normal forms, which is based on the finiteness of the underlying semantic model, can also be used as a technique to prove completeness when a reduced signature is considered (e.g. for TCP) in alternative to other techniques (e.g. that in (Bergstra and Klop 1985)).

The paper is structured 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 the model of processes that we consider: transition systems modulo observational congruence. In Sect. 4 we present the generic process algebra TCP+REC, its operational semantics, and the encoding of the operators of the other algebras CCS, CSP and ACP. In Sect. 5 we present the considered syntactical constraint over sets of equations and the process algebra TCP+REC_f: we prove that TCP+REC_f terms produce finite-state transition systems only (and, the other way around, every finite-state transition system can be expressed in terms of a TCP+REC_f term) and we give a formal argument supporting the claim that the syntactical constraint that we consider is the weakest that guarantees finite-stateness. In Sect. 6 we present the axiomatization and we show that it is sound and ground-complete for observational congruence over the TCP+REC_f signature. Sect. 7 concludes the paper.

This paper is an extended integrated version of the two papers (Baeten and Bravetti 2005; Baeten and Bravetti 2006) that includes proofs for all theorems.

1.1. Acknowledgements

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

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 also include free variables X (they may be so-called open terms): 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 as in $recX.E$). The meaning of “=” in this case is the following: for any substitution of free variables with closed terms the term on the left is equivalent to the term on the right.

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 bisimilarity). 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. Equivalence over elements of the term model can then be assumed, e.g., to be based on observational congruence like in CCS. In ACP syntactical 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.

As a consequence, while the CCS axiom $E + E = E$ allows us to derive $X + X = X$ (by instantiating E with the open term X), we cannot do this with the corresponding ACP axiom $x + x = x$. Note, however, that this does not prevent the possibility of “reasoning” with open terms in ACP: this is done in axiom systems by deriving, from the initial axioms, (possibly) open equations, i.e. identities between terms which use process variables like for such axioms. This capability of deriving (open) equations from (open) equations is obtained by exploiting the axiom system derivation rules that allow, e.g., to instantiate, in an equation, a process variable with a term that can include process variables and to replace equations in the body of other equations. For example, if we also consider the axiom $x + \underline{0} = x$, we can derive from $x + x = x$ the open equation $x + x + \underline{0} = x$. In this view, the capability of ACP to derive an open equation corresponds to the capability of CCS to derive an equation between two open terms, where syntactical free variables X are used instead of process variables. In the example above, the capability to derive $x + x + \underline{0} = x$ in ACP corresponds to the capability to derive $X + X + \underline{0} = X$ from $E + E = E$ and $E + \underline{0} = E$ in CCS. 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, if in ACP we consider the model of labeled transition systems (or optionally the term model) modulo observational congruence, the notion of “axiomatization complete over closed terms” in the context of CCS corresponds to what in ACP is said to be “*ground-complete*”: the axiomatization is complete with respect to closed equations, i.e. identities between closed terms. Moreover, if in ACP we consider the term model (in this case the usage of such a model is mandatory for the correspondence to hold) modulo observational congruence, the notion of “axiomatization complete over open terms” in the context of CCS corresponds to what in ACP is said to be “*complete*”: the axiomatization is complete with respect to (possibly) open equations, i.e. identities between terms that (possibly) include process variables. Alternatively, completeness over open terms in CCS can be expressed in ACP in terms of the ground-completeness requirement described above (to

express completeness over closed terms) plus “ ω -completeness”, which basically requires an open equation to be derivable if and only if all its closed instances are derivable.

Once these basic differences are explained, in the following we will focus on the different ways of expressing recursion in the three process algebras CCS, CSP and ACP. 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 (and that we used also in the introduction) 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 equational theory under consideration such that the equations of $E(V)$ correspond to equal 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 its 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 axioms we showed in the introduction

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

that 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 (whether it is guarded or not). 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, \dots , 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 $\underline{0}$ 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 $\tau.\underline{0}$ in observational congruence. As already observed in the introduction, in CCS such a behaviour is expressed by the three axioms for unguarded recursion

$$\begin{aligned} (FUng) \quad & recX.(X + t) = recX.t \\ (WUng1) \quad & recX.(\tau.X + t) = recX.\tau.t \\ (WUng2) \quad & recX.(\tau.(X + t) + s) = recX.(\tau.X + t + s) \end{aligned}$$

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 will see in Sect. 6). It is worth noting that, if unguardedness is caused just by τ actions (weak unguardedness), as in $\{X = \tau.X\}$, and not by a variable being directly executable on the 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 analogue of axioms WUng1 and WUng2 by adding a much more complex set of conditional equations called CFAR (Cluster Fair Abstraction Rule) introduced in (Vaandrager 1986). CFAR is a generalisation of the KFAR (Koomen’s Fair Abstraction Rule) introduced in (Bergstra and Klop 1986). Note, however, that CFAR and KFAR, differently from the axioms above, are also valid if we work with rooted branching bisimilarity 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 (Baeten and Bergstra 1997)).

3. Behaviours Modulo Observational Congruence

In this paper, we consider the model of transition systems modulo Milner’s observational congruence.

Definition 3.1 (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 $s \xrightarrow{\alpha} t$ is used for $(s, \alpha, t) \in \rightarrow$ and $s \downarrow$ for $s \in \downarrow$.

Here, we will always assume the set S is countable and the set L is finite. Moreover, 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.

The definition of weak bisimulation equivalence that we consider in the following is the usual extension the standard one that is adopted when successful termination is distinguished from unsuccessful termination. Such a distinction is technically needed to have the compatibility (congruence) with a sequential composition operator. It corresponds exactly to the standard one when successful termination is represented by means of an outgoing transition labeled with a special action, instead of a predicate \downarrow .

Definition 3.2 (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 $t \Rightarrow t^* \xrightarrow{\alpha} t'' \Rightarrow t'$ 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 if there is a weak bisimulation relation in which the pair (s, t) satisfies the root condition.

Note that the choice of adopting rooted weak bisimilarity is not a crucial assumption for the theory that we develop. For example a model based on rooted branching bisimilarity (van Glabbeek and Weijland 1996) could also be considered. As we discuss in the paper conclusions, the development of a corresponding theory for rooted branching bisimilarity is left for future work.

4. A Generic Process Algebra

4.1. Theory of Communicating Processes

We consider the process algebra TCP (Theory of Communicating Processes), introduced in (Baeten 2003) and completely worked out in (Baeten *et al.* 2008).

Our theory has two parameters: the set of actions A , and a *communication function* $\gamma : A \times A \rightarrow A$. The function γ is partial, commutative and associative. The signature elements are the following. Constant $\underline{0}$ denotes *inaction* (or deadlock), and is the neutral element of alternative composition: process $\underline{0}$ cannot execute any action, and cannot terminate. Constant $\underline{1}$ denotes the *empty process* or *skip* and is the neutral element of sequential composition: process $\underline{1}$ 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*, which 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 ($\underline{1}$) and unsuccessful termination ($\underline{0}$). Sequential composition is more general than action prefixing. Binary operator \parallel denotes *parallel composition*. In order to give a finite axiomatization of parallel composition, there are two variations on this operator, the auxiliary operators \ll (left-merge) and \mid (synchronization merge). In the parallel composition $x \parallel y$, the separate components may execute

a step independently (denoted by $x \parallel y$ resp. $y \parallel x$), 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 and 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 $f : A \rightarrow A$.

In the following we will use: meta-variables x, y to range over processes of our process algebra, i.e. 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 $A \cup \{\tau\}$. Moreover, by exploiting the commutativity and associativity of choice $+$, in the following we will use the sum notation $\sum_{i \in I} x_i$ to denote a choice among all processes x_i with $i \in I$, where we assume an empty sum (case $I = \emptyset$) to stand for $\underline{0}$.

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 Fig. 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.

Observational congruence is a congruence over TCP and an axiomatization can be provided 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 Fig. 2.

4.2. Theory of Communicating Processes with Recursive Specifications

Here we will introduce in TCP the possibility of performing recursive specifications $E = E(V)$, where $E = \{X = t_X \mid X \in V\}$.

Since as model for our theory we are considering transition systems modulo observational congruence, the guardedness criterion for recursive specifications (we spoke about in Sect. 2) is the following one. 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 bisimilarity, or observational congruence, guarded recursive specifications have unique solutions, so we can talk about *the* process given by a guarded recursive specification.

Our extension to TCP however will not be limited to guarded recursive specifications: we will include the possibility to include general (not necessarily 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 transition relation satisfying the recursive specification. Note that our approach also

$\underline{1} \downarrow$	$\alpha.x \xrightarrow{\alpha} x$		
$\frac{x \xrightarrow{\alpha} x'}{x + y \xrightarrow{\alpha} x'}$	$\frac{y \xrightarrow{\alpha} y'}{x + y \xrightarrow{\alpha} y'}$	$\frac{x \downarrow}{x + y \downarrow}$	$\frac{y \downarrow}{x + y \downarrow}$
$\frac{x \xrightarrow{\alpha} x'}{x \cdot y \xrightarrow{\alpha} x' \cdot y}$	$\frac{x \downarrow, y \xrightarrow{\alpha} y'}{x \cdot y \xrightarrow{\alpha} y'}$	$\frac{x \downarrow, y \downarrow}{x \cdot y \downarrow}$	
$\frac{x \xrightarrow{a} x', y \xrightarrow{b} y', \gamma(a, b) = c}{x \parallel y \xrightarrow{c} x' \parallel y'}$	$\frac{x \downarrow, y \downarrow}{x \parallel y \downarrow}$	$\frac{x \xrightarrow{\alpha} x'}{x \parallel y \xrightarrow{\alpha} x' \parallel y}$	$\frac{y \xrightarrow{\alpha} y'}{x \parallel y \xrightarrow{\alpha} x \parallel y'}$
$\frac{x \xrightarrow{a} x', y \xrightarrow{b} y', \gamma(a, b) = c}{x y \xrightarrow{c} x' y'}$	$\frac{x \downarrow, y \downarrow}{x y \downarrow}$	$\frac{x \xrightarrow{\alpha} x'}{x \ll y \xrightarrow{\alpha} x' \parallel y}$	
$\frac{x \xrightarrow{\tau} x', x' y \xrightarrow{\alpha} z}{x y \xrightarrow{\alpha} z}$	$\frac{y \xrightarrow{\tau} y', x y' \xrightarrow{\alpha} z}{x y \xrightarrow{\alpha} z}$	$\frac{x \xrightarrow{\tau} x', x' y \downarrow}{x y \downarrow}$	$\frac{y \xrightarrow{\tau} y', x y' \downarrow}{x y \downarrow}$
$\frac{x \xrightarrow{\alpha} x', \alpha \notin H}{\partial_H(x) \xrightarrow{\alpha} \partial_H(x')}$	$\frac{x \downarrow}{\partial_H(x) \downarrow}$	$\frac{x \xrightarrow{\alpha} x', \alpha \notin I}{\tau_I(x) \xrightarrow{\alpha} \tau_I(x')}$	$\frac{x \xrightarrow{a} x', a \in I}{\tau_I(x) \xrightarrow{\tau} \tau_I(x')}$
$\frac{x \downarrow}{\tau_I(x) \downarrow}$	$\frac{x \xrightarrow{a} x'}{\rho_f(x) \xrightarrow{f(a)} \rho_f(x')}$	$\frac{x \xrightarrow{\tau} x'}{\rho_f(x) \xrightarrow{\tau} \rho_f(x')}$	$\frac{x \downarrow}{\rho_f(x) \downarrow}$

Fig. 1. Deduction rules for TCP.

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:

$$t ::= \underline{0} \mid \underline{1} \mid a.t \mid \tau.t \mid t + t \mid t \cdot t \mid t \parallel t \mid t \ll t \mid t | t \mid \partial_H(t) \mid \tau_I(t) \mid \rho_f(t) \mid X \mid \langle X | E \rangle$$

where $E = E(V)$ is a set of equations $E = \{X = t \mid X \in V\}$. In the following we will use s, t, u, z to range over terms of TCP+REC.

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

$x + y = y + x$	A1	$x \parallel y = x \parallel y + y \parallel x + x \mid y$	M
$(x + y) + z = x + (y + z)$	A2		
$x + x = x$	A3	$\underline{0} \parallel x = \underline{0}$	LM1
$(x + y) \cdot z = x \cdot z + y \cdot z$	A4	$\underline{1} \parallel x = \underline{0}$	LM2
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	A5	$\alpha.x \parallel y = \alpha.(x \parallel y)$	LM3
$x + \underline{0} = x$	A6	$(x + y) \parallel z = x \parallel z + y \parallel z$	LM4
$\underline{0} \cdot x = \underline{0}$	A7		
$\underline{1} \cdot x = x$	A8	$x \mid y = y \mid x$	SM1
$x \cdot \underline{1} = x$	A9	$\underline{0} \mid x = \underline{0}$	SM2
$(\alpha.x) \cdot y = \alpha.(x \cdot y)$	A10	$\underline{1} \mid \underline{1} = \underline{1}$	SM3
		$a.x \mid b.y = c.(x \parallel y)$ if $\gamma(a, b) = c$	SM4
$\partial_H(\underline{0}) = \underline{0}$	D1	$a.x \mid b.y = \underline{0}$ otherwise	SM5
$\partial_H(\underline{1}) = \underline{1}$	D2	$a.x \mid \underline{1} = \underline{0}$	SM6
$\partial_H(a.x) = \underline{0}$ 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	$\rho_f(\underline{0}) = \underline{0}$	RN1
		$\rho_f(\underline{1}) = \underline{1}$	RN2
$\tau_I(\underline{0}) = \underline{0}$	TI1	$\rho_f(a.x) = f(a).\rho_f(x)$	RN3
$\tau_I(\underline{1}) = \underline{1}$	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

Fig. 2. Axioms of TCP.

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:

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

In the following we use the usual operation $t\{s/X\}$ for expressing syntactical replacement of a closed term s for every free occurrence of variable X : as usual not only variables X occurring directly in t are replaced, but also variables X occurring freely inside its inner recursive specifications.

Fig. 3 provides deduction rules for recursive specifications. Such rules are similar to those in (van Glabbeek 1987), but we have the additional possibility of nesting recursion operators inside recursion operators. They come down to looking upon $\langle X \mid E \rangle$ as the process $\langle t_X \mid E \rangle$, which is defined as follows.

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

Fig. 3. Deduction rules for recursion.

Definition 4.1 Given a set of equations $E = \{X = t_X \mid X \in V\}$ and a TCP+REC term t , we define $\langle t|E \rangle$ to be $t\{\langle X|E \rangle/X \mid X \in V\}$, i.e. t where, for all $X \in V$, all free occurrences of X in t are replaced by $\langle X|E \rangle$.

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

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

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

Together Fig. 1 and Fig. 3 provide a transition-system space, the minimal one (with respect to inclusion of \rightarrow relations and \downarrow sets) that satisfies the operational rules, over closed TCP+REC terms.

4.3. Encoding of Other Process Algebras

The TCP+REC process algebra is *generic*, in the sense that most features of commonly used process algebras can be embedded in it. In particular we will show that the standard process algebras ACP, CCS and CSP are subalgebras of reduced expressions of TCP+REC.

In the following, we made use of (van Glabbeek 1994; van Glabbeek 1997) and (Baeten *et al.* 1991) (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 (Milner 1989a). This is done by omitting the signature elements $\underline{1}, \cdot, \parallel, \mid$. 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 a set of communications \mathcal{A}_c such that for each $a \in \mathcal{A}$ there is exactly one $\bar{a} \in \bar{\mathcal{A}}$ and exactly one $a_c \in \mathcal{A}_c$. 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 \mid_{CCS} can be defined by the formula

$$x \mid_{CCS} y \stackrel{def}{=} \tau_{\mathcal{A}_c}(x \parallel y).$$

We consider a subtheory corresponding to ACP $_{\tau}$, see (Bergstra and Klop 1985). This is done by defining, for each $a \in A$, a new constant a by $a = a.\underline{1}$, and then omitting the signature elements $\underline{1}, \cdot, \rho_f$.

We consider a subtheory corresponding to CSP, see (Hoare 1985). The *non-deterministic choice* operator \sqcap can be defined by

$$x \sqcap y \stackrel{def}{=} \tau.x + \tau.y.$$

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 \mathcal{A} and a set of communications \mathcal{A}_c such that for each $a \in \mathcal{A}$ there is exactly one $a_c \in \mathcal{A}_c$. 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 \mathcal{A} only and $S \subseteq \mathcal{A}$, can be defined by the formula

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

where we use “ S_c ” to denote the set of names $\{a_c \mid a \in S\}$ and “ $-$ ” to express difference of sets. Notice that just adopting the naive communication function $\gamma(a, a) = a$ would not work because, e.g., if we try to translate $a.x \parallel_{\{a\}} a.y$ into $a.x \parallel a.y$ we can erroneously do independent a moves; if we instead consider $\partial_{\{a\}}(a.x \parallel a.y)$ the synchronization on a is erroneously blocked. As far as the CSP *external choice* operator \square is concerned, we further specialize the set of names \mathcal{A} into three parts: a set of names \mathcal{B} , and two sets of names \mathcal{B}_1 and \mathcal{B}_2 , such that for each $a \in \mathcal{B}$ there is exactly one name $a_1 \in \mathcal{B}_1$ and one name $a_2 \in \mathcal{B}_2$. 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, $x \square y$, where x and y are processes using names over \mathcal{B} only, can be defined by the formula

$$x \square y \stackrel{def}{=} \rho_{f' \cup f''}(\rho_{f'^{-1}}(x) \parallel \rho_{f''^{-1}}(y)) \parallel_{\mathcal{B}_1 \cup \mathcal{B}_2} (\mathcal{B}_1^* \underline{1} + \mathcal{B}_2^* \underline{1})$$

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

$$\langle X \mid \{X = x + \sum_{a \in B} a.X\} \rangle.$$

Note that the definition above does not work for versions of \square , as, e.g., that in (Baeten *et al.* 2008), that take the distinction between successful and unsuccessful termination into account.

5. A Generic Process Algebra for Finite Behaviours

In order to restrict to a setting of processes with a finite-state model only, we now consider a restricted syntax for constants $\langle X \mid E \rangle$ which guarantees that transition systems generated by the operational rules are indeed finite-state. The restricted syntax is based on the requirement that E is an *essentially finite-state* recursive specification according to the definition we present below.

We consider the process algebra $TCP+REC_f$ to be obtained by extending the signature of TCP with essentially finite-state recursive specifications; i.e. we consider closed terms in the $TCP+REC$ syntax, where we additionally require that every recursive specification is essentially finite-state.

Definition 5.1 A free variable X is *serial* in a term t of $TCP + REC$ if every free occurrence of X is in the scope of one of the operators $\parallel, \parallel, |, \partial_H, \tau_I, \rho_f$ or in the left-hand side of the operator \cdot .

Definition 5.2 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 are serial in right-hand sides of all equations of E . We call E *regular* if E has only finitely many equations and each equation is of the form

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

where an empty sum stands for $\underline{0}$ and the $\underline{1}$ summand is optional, for some $n \in \mathbb{N}$, $\alpha_i \in A \cup \{\tau\}$, $X_i \in V$. It is immediate that every regular recursive specification is essentially finite-state.

Now it is a well-known fact that each finite-state process can be described by a regular recursive specification. Conversely, in the following proposition we show that 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.

In the proof of the proposition we make use of the fact that, according to the Fresh Atom Principle (Baeten and van Glabbeek 1987), it is always possible to introduce a fresh action r by extending with r the parameters the of theory under consideration A and γ (the latter can, possibly, be extended to include communication with r): for terms over the signature with the previous parameters A and γ (i.e. where the new action r does not appear) we have unchanged transition systems/bisimilarities/equalities. In this paper we assume γ to remain unchanged when we introduce fresh actions.

Proposition 5.3 Let t be a closed term such that every recursive specification E included in t is essentially finite-state. The transition system for t generated by the operational rules has only finitely many states.

Proof. To start, define $c(t)$ to be the closed term obtained from any (possibly open) term t by replacing each free variable X occurring in t with $a_X.\underline{0}$, where a_X is a fresh action.

We now show, by structural induction over the syntax of (possibly open) terms t , that: if t is such that every recursive specification E included in t is essentially finite-state, then $c(t)$ generates a finite-state transition system.

The base cases of the induction are the following ones:

- if $t \equiv \underline{0}$, then $c(t) = \underline{0}$ is obviously finite-state.
- if $t \equiv \underline{1}$, then $c(t) = \underline{1}$ is obviously finite-state.
- if $t \equiv X$, then $c(t) = a_X.\underline{0}$ is obviously finite-state.

The inductive cases of the induction are the following ones:

- if $t \equiv a.t'$ or $t \equiv \tau.t'$ or $t \equiv t' + t''$ or $t \equiv t' \cdot t''$ or $t \equiv t' \parallel t''$ or $t \equiv t' \ll t''$ or $t \equiv t' | t''$ or $t \equiv \partial_H(t')$ or $t \equiv \tau_I(t')$ or $t \equiv \rho_f(t')$, then $c(t)$ is obviously finite-state by an inductive argument over t' and t'' .
- if $t \equiv \langle X|E' \rangle$ then $c(t)$ is proved to be finite-state as follows. Given $E = E(V)$ such that $\langle X|E \rangle \equiv c(\langle X|E' \rangle)$ and assuming that the set of states in the transition system

generated by a term t' is denoted by $S(t')$, we show that:

$$S(\langle X|E \rangle) \subseteq \{\langle X|E \rangle\} \cup \text{ren}\left(\bigcup_{Y \in V} S(c(t_Y))\right)$$

where $\text{ren}(t')$ is a renaming function for a term t' that, for any $Y \in V$, replaces every occurrence of $a_Y.\underline{0}$ with $\langle Y|E \rangle$ (here we use the obvious extension of function ren to set of terms where such a renaming is applied to every term in the set). Once proved that the above statement holds then $c(t)$ is obviously finite-state by inductive argument over terms t_Y , for every $Y \in V$.

In the following we prove that the inclusion above indeed holds. First of all we assume that in $\langle X|E \rangle$ bound variables inside E are α -renamed in such a way that there is no recursion operator binding a variable by using a name which is already bound by an outer operator. Then we show, by induction on the height of the inference tree by which any transition $s \xrightarrow{\alpha} s'$ is derived with the operational semantics it holds that:

- If there is no $Y \in V$ such that $\langle Y|E \rangle$ is included in s then there is no $Y \in V$ such that $\langle Y|E \rangle$ is included in s' .
- If there is no $Y \in V$ such that $\langle Y|E \rangle$ is included in s inside the scope of one of the operators $\|, \llbracket, |, \partial_H, \tau_I, \rho_f$ or on the left-hand side of the operator \cdot , then there is no $Y \in V$ such that $\langle Y|E \rangle$ is included in s' inside the scope of one of the operators $\|, \llbracket, |, \partial_H, \tau_I, \rho_f$ or on the left-hand side of the operator \cdot .
- If for every $Y \in V$ it holds that the occurrence of $\langle Y|E'' \rangle$ in s implies $E'' = E$, then for every $Y \in V$ it holds that the occurrence of $\langle Y|E'' \rangle$ in s' implies $E'' = E$.

This can be easily proved by analysis of each operational rule supposing that the above statement holds for the premise and observing that it holds for the transition derived in the conclusion.

Then, by induction on the length of a derivation sequence from $\langle X|E \rangle$ to any state u , we have that the following holds. $u \in S(\langle X|E \rangle)$ implies:

- There is no $Y \in V$ such that $\langle Y|E \rangle$ is included in u inside the scope of one of the operators $\|, \llbracket, |, \partial_H, \tau_I, \rho_f$ or on the left-hand side of the operator \cdot .
- For every $Y \in V$ it holds that the occurrence of $\langle Y|E'' \rangle$ in u implies $E'' = E$.

Now, given any transition $s \xrightarrow{\alpha} s'$, we say that $s \xrightarrow{\alpha} s'$ can be inferred without E iff $s \xrightarrow{\alpha} s'$ can be inferred by using no operational rule of any $\langle Y|E \rangle$, with $Y \in V$. Given any transition $s \xrightarrow{\alpha} s'$ that cannot be inferred without E , with $s \in S(\langle X|E \rangle)$, we say that $s \xrightarrow{\alpha} s'$ can be inferred by using $\langle Y|E \rangle$ iff $s \xrightarrow{\alpha} s'$ can be inferred in such a way that, the operator $\langle Z|E \rangle$, with $Z \in V$, whose operational rule is applied at the highest depth in the inference (distance from the derivation of $s \xrightarrow{\alpha} s'$), is such that $Z = Y$.

Note that, the latter is well-defined, because, for the properties above that characterize states in $S(\langle X|E \rangle)$, it is not possible to infer $s \xrightarrow{\alpha} s'$ by means of multiple operational rules for operators $\langle Y|E \rangle$, with $Y \in V$, that are applied in different branches of the inference.

We now conclude the proof by showing that, given $u \in S(\langle X|E \rangle)$, then either $u \equiv \langle X|E \rangle$, or there exists t' such that $c(t')$ is derivable from $c(t_Y)$ for some $Y \in V$ and

$u = \text{ren}(c(t'))$.

Supposed that $u \not\equiv \langle X|E \rangle$, then given the non-empty derivation sequence from $\langle X|E \rangle$ to u , we consider the last transition $s \xrightarrow{\alpha} s'$ in such a sequence such that $s \xrightarrow{\alpha} s'$ cannot be inferred without E (we are sure that such a transition exists because the first transition in the derivation sequence is of this kind). So let us consider a variable Y such that $s \xrightarrow{\alpha} s'$ can be inferred by using $\langle Y|E \rangle$. It is easy to see that there exists t' such that $c(t')$ is derivable from $c(t_Y)$ and $u = \text{ren}(c(t'))$. This is because:

- s' is such that $\text{ren}(c(t_Y)) \xrightarrow{\alpha} s'$ and $\text{ren}(c(t_Y)) \xrightarrow{\alpha} s'$ can be inferred without E ; from the fact that $s \xrightarrow{\alpha} s'$ can be inferred by using $\langle Y|E \rangle$ and for the properties above that characterize states in $S(\langle X|E \rangle)$: i.e. $\langle Y|E \rangle$ is allowed to occur inside s only in the scope of $+$ or recursion operators or in the right-hand side of \cdot operators.
- Given any t_1, v', α' such that $\text{var}(\text{ren}(c(t_1)) \xrightarrow{\alpha'} v')$ can be inferred without E , there exists t'_1 such that $v' = \text{ren}(c(t'_1))$ and $c(t_1) \xrightarrow{\alpha'} c(t'_1)$; by induction on the height of the inference tree of transitions $\text{var}(\text{ren}(c(t_1)) \xrightarrow{\alpha'} v')$ inspecting each operational rule.

□

The syntactical restriction that we propose on recursive specifications ensures that the operational rules generate only finitely many states. But, even if the operational rules generate infinitely many states, it can still be the case that there are only finitely many states modulo bisimilarity. For instance, for the recursive equation $X = \tau_{\{a\}}(a.X)$, each time a new abstraction operator is generated, but all the generated terms are bisimilar as the abstraction operator is idempotent. Of course, in other cases, as in $X = (a.\underline{1}) + (b.X \cdot X)$, we do obtain infinitely many terms that are not bisimilar. However, note that the axioms that we will present in the following Sect. 6 remain valid, also if recursive specifications that are not essentially finite-state are considered (obtaining a model of possibly infinite transition systems modulo bisimilarity).

In the following we present a proposition that shows that the definition of essentially finite-state does not disregard unnecessarily terms which generate finite-state transition systems. First, we need to introduce some machinery related to the representation of contexts and a technical lemma.

Definition 5.4 A context is a term $t_{\dot{X}}$ that includes a single occurrence of the free variable X (and possibly other free variables). A context $t_{\dot{X}}$ is a closed context if X is the only free variable in $t_{\dot{X}}$. A context $t_{\dot{X}}$ is unfolded if X does not occur in $t_{\dot{X}}$ in the scope of an operator $\langle Y|E \rangle$ for any Y, E . We use $t_{\dot{X}}(t')$ to stand for $t_{\dot{X}}\{t'/X\}$ and $t_{\dot{X}}^n$, with $n \geq 0$ to stand for the term inductively defined as follows. $t_{\dot{X}}^0 \equiv X$. $t_{\dot{X}}^n \equiv t_{\dot{X}}(t_{\dot{X}}^{n-1})$ for $n > 0$. A static context is a context such that X may only occur in the scope of $\|, \partial_H, \tau_I, \rho_f$ operators or in left-hand side of operators \cdot .

Note that a closed static context is obviously unfolded. In the following we use w to range over action sequences, i.e. non-empty strings of actions $\alpha \in A \cup \{\tau\}$. We also use

$w.t$ as a shorthand notation for a sequence of prefixes generating a path labelled with w that leads to t , i.e. $w.t \equiv \alpha.t$ if $w = \alpha$, $w.t \equiv \alpha.(w'.t)$ if $w = \alpha w'$. Finally, we implicitly assume that, when introducing some fresh action r , the renaming functions f occurring inside terms under consideration keep r unchanged, i.e. they are such that $f(r) = r$.

Definition 5.5 Let $t_{\dot{X}}, t'_{\dot{X}}$ be closed contexts, with $t_{\dot{X}}$ unfolded, w be an action sequence and $\alpha \in A \cup \{\tau\}$. We say that $t'_{\dot{X}}$ is an α -derivative of $t_{\dot{X}}$ for w , iff, considered a fresh action r , we have $t_{\dot{X}}(w.r.\underline{0}) \xrightarrow{\alpha} t'_{\dot{X}}(r.\underline{0})$ and $t'_{\dot{X}}(r.\underline{0}) \xrightarrow{r}$.

Definition 5.6 Let $t_{\dot{X}}$ be a static context. We define the static context $free(t_{\dot{X}})$ as the unique (up to renaming of non- X free variables) static context $t'_{\dot{X}}$ such that: all subterms of $t'_{\dot{X}}$ that do not include X are free variables; for every free variable Y of $t'_{\dot{X}}$ a single occurrence of Y is included; and for some substitution θ of the non- X free variables in $t'_{\dot{X}}$ we have $t'_{\dot{X}}\theta \equiv t_{\dot{X}}$.

Lemma 5.7 Let $t_{\dot{X}}$ be a closed unfolded context and v be a closed term. If $t_{\dot{X}}(v.\underline{0}) \xrightarrow{\alpha} t'$ for some $\alpha \in A \cup \{\tau\}$ and closed term t' then at least one of the following conditions holds true.

1. There exist a closed context $t'_{\dot{X}}$ and a closed term v' that is reachable from v via a path labeled by some sequence of actions w , with $t' \equiv t'_{\dot{X}}(v'.\underline{0})$ and $t'_{\dot{X}}$ is an α -derivative of $t_{\dot{X}}$ for w , such that:
 - a) $t'_{\dot{X}}$ is static and: if $t_{\dot{X}}$ has X in the scope of one of the operators $\parallel, \llbracket, \mid, \partial_H, \tau_I, \rho_f$ or in left-hand side of the operator \cdot then $t'_{\dot{X}} \not\equiv X$; if $t_{\dot{X}}$ is also static then $free(t'_{\dot{X}})$ coincides up to renaming of non- X free variables with $free(t_{\dot{X}})$.
 - b) For any closed terms s, s' such that there exists a path from s to s' labeled by w we have $t_{\dot{X}}(s) \xrightarrow{\alpha} t'_{\dot{X}}(s')$.
2. There exists a closed unfolded context $t'_{\dot{X}}$, with $t' \equiv t'_{\dot{X}}(v.\underline{0})$ and, considered a fresh action r , $t_{\dot{X}}(r.\underline{0}) \xrightarrow{\alpha} t'_{\dot{X}}(r.\underline{0})$. Moreover, the following two properties are satisfied. If $t_{\dot{X}}$ is static or, for some static context $s_{\dot{X}}$ and closed term u , $t_{\dot{X}} \equiv s_{\dot{X}} \mid u$ or $t_{\dot{X}} \equiv u \mid s_{\dot{X}}$, then $t'_{\dot{X}}$ is static. If X occurs in $t'_{\dot{X}}$ in the scope of a $+$ operator then X occurs in $t_{\dot{X}}$ in the scope of a $+$ operator.
3. X occurs in $t_{\dot{X}}$ in the scope of a $+$ operator and, considered a fresh action r , $t_{\dot{X}}(r.\underline{0}) \xrightarrow{\alpha} t'$.

Moreover, if $t_{\dot{X}}(v.\underline{0}) \downarrow$ then X occurs in $t_{\dot{X}}$ in the scope of a $+$ operator and, considered a fresh action r , $t_{\dot{X}}(r.\underline{0}) \downarrow$.

Proof. The proof is by induction on the height of the inference tree by which transitions $t_{\dot{X}}(v.\underline{0}) \xrightarrow{\alpha} t'$ of $t_{\dot{X}}(v.\underline{0})$, for any closed unfolded context $t_{\dot{X}}$ and closed term v , or its termination capability, are inferred.

As the base step of the induction we consider the cases $t_{\dot{X}} \equiv X$ and $t_{\dot{X}} \equiv \alpha.s_{\dot{X}}$, for some context $s_{\dot{X}}$, for which the lemma is obvious (condition 1 and 2, respectively, obviously hold).

The inductive step is divided in cases depending on the topmost operator in $t_{\dot{X}}$.

In the following we just develop the case $t_{\dot{X}} \equiv s_{\dot{X}} \mid u$, for any $s_{\dot{X}}$ and u , ($t_{\dot{X}} \equiv u \mid s_{\dot{X}}$ is symmetric) that is the most intricate one, for the other operators the proof is an easy verification of the properties. Note that the case $t_{\dot{X}} \equiv \langle Y \mid E \rangle$, for any Y and E , cannot be obtained because $t_{\dot{X}}$ is unfolded. Furthermore, note that in the case $t_{\dot{X}} \equiv s_{\dot{X}} \cdot u$, the statement about termination capability of $t_{\dot{X}}(v \cdot \underline{0})$ in the lemma is needed to derive that $s_{\dot{X}}(v \cdot \underline{0}) \cdot u \xrightarrow{\alpha} t'$ satisfies condition 3 in the case $s_{\dot{X}}(v \cdot \underline{0}) \downarrow$ and $u \xrightarrow{\alpha} t'$.

From $s_{\dot{X}}(v \cdot \underline{0}) \mid u \xrightarrow{\alpha} t'$ we have three possible cases corresponding to the operational rules for the \mid operator that yield an outgoing transition.

— $s_{\dot{X}}(v \cdot \underline{0}) \xrightarrow{\tau} s''$ and $s'' \mid u \xrightarrow{\alpha} t'$.

By induction we have that $s_{\dot{X}}(v \cdot \underline{0}) \xrightarrow{\tau} s''$ must satisfy one of the conditions of the lemma, thus we have three cases numbered according to the satisfied condition.

1. There are $s''_{\dot{X}}, v''$ such that $s'' \equiv s''_{\dot{X}}(v'' \cdot \underline{0})$ and $s''_{\dot{X}}$ is a τ -derivative of $s_{\dot{X}}$ for some w' labeling a path from v to v'' . Hence w' is such that $s_{\dot{X}}(w'.r.\underline{0}) \xrightarrow{\tau} s''_{\dot{X}}(r.\underline{0})$ and $s''_{\dot{X}}(r.\underline{0}) \xrightarrow{r}$.

By induction we have that $s''_{\dot{X}}(v'' \cdot \underline{0}) \mid u \xrightarrow{\alpha} t'$ must satisfy one of the conditions of the lemma, thus we have three subcases numbered according to the satisfied condition.

- sub 1. There are $t'_{\dot{X}}, v'$ such that $t' \equiv t'_{\dot{X}}(v' \cdot \underline{0})$ and $t'_{\dot{X}}$ is an α -derivative of $s''_{\dot{X}} \mid u$ for some w'' labeling a path from v'' to v' . Hence w'' is such that $s''_{\dot{X}}(w''.r.\underline{0}) \mid u \xrightarrow{\alpha} t'_{\dot{X}}(r.\underline{0})$ and $t'_{\dot{X}}(r.\underline{0}) \xrightarrow{r}$.
By observing that $s_{\dot{X}}(w'w''.r.\underline{0}) \xrightarrow{\tau} s''_{\dot{X}}(w''.r.\underline{0})$, we conclude that $s_{\dot{X}}(v \cdot \underline{0}) \mid u \xrightarrow{\alpha} t'$ satisfies condition 1 (with $w = w'w''$).

- sub 2. There is $t'_{\dot{X}}$ such that $t' \equiv t'_{\dot{X}}(v'' \cdot \underline{0})$ and, considered a fresh action r , $s''_{\dot{X}}(r.\underline{0}) \mid u \xrightarrow{\alpha} t'_{\dot{X}}(r.\underline{0})$.
By observing that $t'_{\dot{X}}$ is static because $s''_{\dot{X}}$ is static, we conclude that $s_{\dot{X}}(v \cdot \underline{0}) \mid u \xrightarrow{\alpha} t'$ satisfies condition 1 (with $w = w'$ and $v' \equiv v''$).

- sub 3. This case cannot be obtained because $s''_{\dot{X}}$ is static.

2. There is $s''_{\dot{X}}$ such that $s'' \equiv s''_{\dot{X}}(v \cdot \underline{0})$ and, considered a fresh action r , $s_{\dot{X}}(r.\underline{0}) \xrightarrow{\tau} s''_{\dot{X}}(r.\underline{0})$.

By induction we have that $s''_{\dot{X}}(v \cdot \underline{0}) \mid u \xrightarrow{\alpha} t'$ must satisfy one of the conditions of the lemma, thus we have three subcases numbered according to the satisfied condition.

- sub 1. There are $t'_{\dot{X}}, v'$ such that $t' \equiv t'_{\dot{X}}(v' \cdot \underline{0})$ and $t'_{\dot{X}}$ is an α -derivative of $s''_{\dot{X}} \mid u$ for some w labeling a path from v to v' . Hence w is such that $s''_{\dot{X}}(w.r.\underline{0}) \mid u \xrightarrow{\alpha} t'_{\dot{X}}(r.\underline{0})$ and $t'_{\dot{X}}(r.\underline{0}) \xrightarrow{r}$.
By observing that $s_{\dot{X}}(w.r.\underline{0}) \xrightarrow{\tau} s''_{\dot{X}}(w.r.\underline{0})$, we conclude that $s_{\dot{X}}(v \cdot \underline{0}) \mid u \xrightarrow{\alpha} t'$ satisfies condition 1.

- sub 2. There is $t'_{\dot{X}}$ such that $t' \equiv t'_{\dot{X}}(v \cdot \underline{0})$ and, considered a fresh action r , $s''_{\dot{X}}(r.\underline{0}) \mid u \xrightarrow{\alpha} t'_{\dot{X}}(r.\underline{0})$.

By observing that $s''_{\dot{X}}$ is static whenever $s_{\dot{X}}$ is static, we conclude that $s_{\dot{X}}(v \cdot \underline{0}) \mid u \xrightarrow{\alpha} t'$ satisfies condition 2.

sub 3. Considered a fresh action r , $s''_{\dot{X}}(r.\underline{0}) \mid u \xrightarrow{\alpha} t'$.

By observing that X occurs in $s_{\dot{X}}$ in the scope of a $+$ operator because X occurs in $s''_{\dot{X}}$ in the scope of a $+$ operator, we conclude that $s_{\dot{X}}(v \cdot \underline{0}) \mid u \xrightarrow{\alpha} t'$ satisfies condition 3.

3. Considered a fresh action r , $s_{\dot{X}}(r.\underline{0}) \xrightarrow{\tau} s''$.

We immediately conclude that $s_{\dot{X}}(v \cdot \underline{0}) \mid u \xrightarrow{\alpha} t'$ satisfies condition 3.

— $u \xrightarrow{\tau} u''$ and $s_{\dot{X}}(v \cdot \underline{0}) \mid u'' \xrightarrow{\alpha} t'$.

By an easy verification of the properties we conclude that $s_{\dot{X}}(v \cdot \underline{0}) \mid u \xrightarrow{\alpha} t'$ satisfies the same condition as that satisfied by $s_{\dot{X}}(v \cdot \underline{0}) \mid u'' \xrightarrow{\alpha} t'$.

— $s_{\dot{X}}(v \cdot \underline{0}) \xrightarrow{a} s'$, $u \xrightarrow{b} u'$, $\gamma(a,b) = \alpha$ and $t' \equiv s' \parallel u'$.

By an easy verification of the properties we conclude that $s_{\dot{X}}(v \cdot \underline{0}) \mid u \xrightarrow{\alpha} t'$ satisfies the same condition as that satisfied by $s_{\dot{X}}(v \cdot \underline{0}) \xrightarrow{a} s'$.

From $s_{\dot{X}}(v \cdot \underline{0}) \mid u \downarrow$ we have three possible cases, corresponding to the operational rules for the \mid operator that yield a terminating term, that are completely analogous to the three cases considered above for transitions: just termination \downarrow has to be considered instead of a,b or α outgoing transitions. In particular, for the first case considered above we have the same three cases corresponding to the condition that is satisfied: in the case of condition 1 and 2 we follow a similar argument as that in the subcase 3 (thus condition 1 cannot be obtained).

□

From the lemma above it immediately follows that, given an unfolded context $s_{\dot{X}}$, for any action $\alpha \in A \cup \{\tau\}$ and context $s'_{\dot{X}}$ such that $s'_{\dot{X}}$ is an α -derivative of $s_{\dot{X}}$ for some sequence of actions w , then a) and b) of the lemma, where we take $t_{\dot{X}} \equiv s_{\dot{X}}$ and $t'_{\dot{X}} \equiv s'_{\dot{X}}$, hold true. This is obtained from the lemma by considering a fresh action r' and by taking v to be $w.r'.\underline{0}$, $t_{\dot{X}}$ to be $s_{\dot{X}}$ and t' to be $s'_{\dot{X}}((r'.\underline{0}) \cdot \underline{0})$. Since the second and third conditions cannot hold in this case (the second one because $t' \xrightarrow{r'}$ and $t'_{\dot{X}}$ cannot syntactically include r' because $t_{\dot{X}}$ does not; the third one because t' syntactically includes r' while $t_{\dot{X}}$ does not), the first one must hold. Moreover, from $t'_{\dot{X}}(v' \cdot \underline{0}) \xrightarrow{r'}$ (because $t' \equiv s'_{\dot{X}}((r'.\underline{0}) \cdot \underline{0})$ and we take $t'_{\dot{X}}(v' \cdot \underline{0}) \equiv t'$), the fact that $t'_{\dot{X}}$ does not syntactically include r' (because $t_{\dot{X}}$ does not), and the fact that v' is reachable from $w.r'.\underline{0}$, we derive $v' \equiv r'.\underline{0}$, hence also $t'_{\dot{X}} \equiv s'_{\dot{X}}$.

Definition 5.8 Let t, t' be open terms. t' is a one-step unfolding of t if t has a subterm $\langle Y \mid E \rangle$, for some Y and E , and t' is obtained from t by replacing it with $\langle t_Y \mid E \rangle$. t' is a multi-step unfolding of t if: $t' \equiv t$ or t' is a one-step unfolding of t'' and t'' is a multi-step unfolding of t . Let $t_{\dot{X}}, t'_{\dot{X}}$ be closed contexts and t a closed term. We say that $t'_{\dot{X}}$ is *unfolding of $t_{\dot{X}}$ w.r.t. t* if there exists an unfolded context $t''_{\dot{X}}$ and a variable Y such that $t''_{\dot{X}}\{X/Y\}$ is a multi-step unfolding of $t_{\dot{X}}$ and $t'_{\dot{X}} = t''_{\dot{X}}\{t/Y\}$.

Definition 5.9 An action α (possibly τ) is not restricted by a context $t_{\dot{X}}$ if, for every

substitution θ of the non- X free variables in $t_{\dot{X}}$, we have that there exists α' (possibly τ) and closed context $t'_{\dot{X}}$ such that $t'_{\dot{X}}$ is an α' -derivative of $t_{\dot{X}}\theta$ for α . We say that a set of actions $S \subseteq A \cup \{\tau\}$ is not restricted by a context $t_{\dot{X}}$ if for any $\alpha \in S$ such a condition holds true.

Proposition 5.10 Let t be a closed term that includes a recursive specification E that has finitely many equations but is not essentially finite-state, i.e. some occurrence of some variable $Y \in V(E)$ violates the seriality condition. If it holds that:

1. There exists a (possibly zero-length) path from t to $t_{\dot{X}}(\langle Y|E' \rangle)$, for some E' obtained from E by substitution of its free variables (if present) and for some context $t_{\dot{X}}$ having X in the scope of one of the operators $\parallel, \llbracket, \mid, \partial_H, \tau_I, \rho_f$ or in left-hand side of the operator \cdot and $t_{\dot{X}}$ is such that there exists an α transition from $t_{\dot{X}}(\langle Y|E' \rangle)$ to $t'_{\dot{X}}(t')$, where $t'_{\dot{X}}$ is an α -derivative of an unfolding of $t_{\dot{X}}$ w.r.t. $\langle Y|E' \rangle$ for some action sequence labelling a path that goes from $\langle Y|E' \rangle$ to t' .
2. There exists a (possibly zero-length) path labelled over $S \subseteq A \cup \{\tau\}$ from t' to $t_{\dot{X}}(\langle Y|E' \rangle)$.
3. For any $n \geq 1$, $S \cup \{\alpha\}$ is not restricted by the static context $free(t_{\dot{X}}^n)$.

Then t has infinitely many states.

Proof. We show by induction on n that, for every $n \geq 1$, there is a path from t to a state $free(t_{\dot{X}}^n)(t')\theta$ for some substitution θ of its free variables. Note that, by Lemma 5.7, $t'_{\dot{X}}$ is static and includes at least one operator because it is an α -derivative of an unfolding of $t_{\dot{X}}$. Therefore, proving this yields the conclusion that t has infinitely many states.

If $n = 1$ we directly have that, from the assumption 1 above, there is a path from t to $t'_{\dot{X}}(t')$, where, by definition, $t'_{\dot{X}}$ is obtained from $free(t'_{\dot{X}})$ by substitution of its free variables.

In the case $n > 1$ we resort to the induction hypothesis, i.e. we assume that there is a path from t to a state $free(t_{\dot{X}}^{n-1})(t')\theta$ for some substitution θ of its free variables. Since, from the assumption 2 above, there is a path labelled over $S \subseteq A \cup \{\tau\}$ from t' to $t_{\dot{X}}(\langle Y|E' \rangle)$; from the assumption 1 above, there is a transition α from $t_{\dot{X}}(\langle Y|E' \rangle)$ to $t'_{\dot{X}}(t')$; and, from the assumption 3 above $S \cup \{\alpha\}$ is not restricted by the static context $free(t_{\dot{X}}^{n-1})$, we have that, by statements a) and b) of Lemma 5.7, there is a path from $free(t_{\dot{X}}^{n-1})(t')\theta$ to $free(t_{\dot{X}}^{n-1})(t'_{\dot{X}}(t'))\theta'$ for some substitution θ' . We therefore conclude that there exists a substitution θ'' such that there is a path from t to a state $free(t_{\dot{X}}^n)(t')\theta''$. \square

Of course, it may be the case that, for terms t considered in the proposition above, the produced transition system modulo bisimilarity has only finitely many states.

6. An Axiomatization Complete for Finite Behaviours

Now we will present a sound axiomatization which is ground-complete for the process algebra TCP+REC_f. The axioms in Fig. 2 together with the axioms in Fig. 4 form such

$\langle X E \dot{\cup} \{Y = t\} \rangle = \langle \langle X E \rangle Y = t \rangle$ if $X \neq Y$	Dec
$\langle X X = t \rangle = \langle t X = t \rangle$	Unf
$s = t\{s/X\} \Rightarrow s = \langle X X = t \rangle$ if $X = t$ guarded	Fold
$\langle X X = X + t \rangle = \langle X X = t \rangle$	Ung
$\langle X X = \tau.(X + t) + s \rangle = \langle X X = \tau.(t + s) \rangle$	WUng
$\tau_I(\langle X X = t \rangle) = \langle X X = \tau_I(t) \rangle$ if X is serial in t	Hid

Fig. 4. Axioms for recursion.

an axiomatization. In the axioms of Fig. 4 the symbol $\dot{\cup}$ stands for disjoint union. Note that the axioms in Fig. 4 are *axiom schemes*: we have these axioms for each possible term s, 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

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

Since, thanks to the decomposition axiom, we deal with recursive specifications that are in the form $\langle X \mid \{X = t\} \rangle$, we denote them just with $\langle X \mid X = t \rangle$.

The unfolding axiom (Unf) is Milner's standard one (corresponding to the *Recursive Definition Principle* in ACP): 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 (corresponding to the *Recursive Specification Principle* in ACP): it states that if y is a solution for X in E , and E is guarded, then $y = \langle X \mid E \rangle$.

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:

$$\begin{aligned} \langle X \mid X = \tau.X + t \rangle &= \langle X \mid X = \tau.t \rangle \\ \langle X \mid X = \tau.(X + t) + s \rangle &= \langle X \mid X = \tau.X + t + s \rangle \end{aligned}$$

The first one is obtained from (WUng) by just taking $t = \underline{0}$. The second one is obtained from (WUng) as follows:

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

by directly applying (WUng) and then

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

by applying (WUng) where we take $s = t + s$ and $t = \underline{0}$.

As explained in the introduction, the axiom (Hid) is used to get rid of weak unguardedness generated by the hiding operator. It allows one 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 6.5). Notice that the “ X serial in t ” condition in axiom (*Hid*) is needed for it to be sound. This is because if X occurs inside an operator like relabeling or parallel that can change the type of the actions in I that X executes (so that their type is no longer in I), then such actions are hidden by $\langle X|X = \tau_I(t) \rangle$ but not by $\tau_I(\langle X|X = t \rangle)$. For instance, if f is a relabeling function that turns a into b and $I = \{a\}$, $\tau_I(\langle X|X = a.\underline{1} + \rho_f(X) \rangle)$ is not equivalent to $\langle X|X = \tau_I(a.\underline{1} + \rho_f(X)) \rangle$, because the former can do a b transition, while the latter cannot do a b transition.

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 unguardedness removal effect that here we obtain by our axiom *Hid* (plus the *WUng* axiom), but in the different context of branching bisimilarity, we have to consider the much more complex set of conditional equations called CFAR (Cluster Fair Abstraction Rule) (Vaandrager 1986) we already mentioned in Sect. 2.

Proposition 6.1 The axiomatization formed by the axioms in Fig. 2 and 4 is sound for TCP+REC and the model of transition systems modulo observational congruence generated by the rules in Fig. 1 and 3.

Proof. Most of the axioms are standard. We provide a full proof for the new axiom (*Hid*) $\tau_I(\langle X|X = t \rangle) = \langle X|X = \tau_I(t) \rangle$ if X is serial in t .

We show that

$\beta = \{(\tau_I(\langle s|X = t \rangle), \tau_I(\langle s|X = \tau_I(t) \rangle)) \mid s \text{ contains at most } X \text{ free and } X \text{ is serial in } s\}$ satisfies the conditions:

- if $\tau_I(\langle s|X = t \rangle) \xrightarrow{\alpha} u$ then, for some u', u'' ,
 $\tau_I(\langle s|X = \tau_I(t) \rangle) \xrightarrow{\alpha} u'', u'' \Leftrightarrow_w u'$ and $(u, u') \in \beta$,
- if $\tau_I(\langle s|X = t \rangle) \downarrow$ then $\tau_I(\langle s|X = \tau_I(t) \rangle) \downarrow$;
- and symmetrically for a move and termination capabilities of $\tau_I(\langle s|X = \tau_I(t) \rangle)$.

This implies that β is a weak bisimulation up to \Leftrightarrow_w (see the revised version of (Milner 1989a) corrected according to (Sangiorgi and Milner 1992)), hence $\beta \subseteq \Leftrightarrow_w$. From this result it follows that $\tau_I(\langle X|X = t \rangle) \Leftrightarrow_{rw} \langle X|X = \tau_I(t) \rangle$ because $\tau_I(\langle X|X = t \rangle) \xrightarrow{\alpha} u$ if and only if $\tau_I(\langle t|X = t \rangle) \xrightarrow{\alpha} u$ and, similarly, $\langle X|X = \tau_I(t) \rangle \xrightarrow{\alpha} u$ if and only if $\langle \tau_I(t)|X = \tau_I(t) \rangle \equiv \tau_I(\langle t|X = \tau_I(t) \rangle) \xrightarrow{\alpha} u$.

In the following we prove that β satisfies the condition above by induction on the height of the inference tree by which α transitions of $\tau_I(\langle s|X = t \rangle)$ or its termination capability are inferred.

The base cases of the induction are the following ones (distinguished by the form of s):

- If $s \equiv \underline{0}$ or $s \equiv \underline{1}$ then the conditions above trivially hold.
- If $s \equiv \alpha.s'$ then $\tau_I(\langle s|X = t \rangle) \equiv \tau_I(\alpha.(\langle s'|X = t \rangle))$ and $\tau_I(\langle s|X = \tau_I(t) \rangle) \equiv$

$\tau_I(\alpha.(\langle s'|X = \tau_I(t) \rangle))$.

Concerning transitions α we have the following two cases.

- $\tau_I(\alpha.(\langle s'|X = t \rangle)) \xrightarrow{\tau} \tau_I(\langle s'|X = t \rangle)$ and $\alpha \in I \cup \{\tau\}$. We directly have $\tau_I(\alpha.(\langle s'|X = \tau_I(t) \rangle)) \xrightarrow{\tau} \tau_I(\langle s'|X = \tau_I(t) \rangle)$ and the targets are related by β .
- $\tau_I(\alpha.(\langle s'|X = t \rangle)) \xrightarrow{\alpha} \tau_I(\langle s'|X = t \rangle)$ and $\alpha \notin I \cup \{\tau\}$. We directly have $\tau_I(\alpha.(\langle s'|X = \tau_I(t) \rangle)) \xrightarrow{\alpha} \tau_I(\langle s'|X = \tau_I(t) \rangle)$ and the targets are related by β .

Concerning termination capability, we have that $\tau_I(\alpha.(\langle s'|X = t \rangle)) \downarrow$ obviously cannot hold because $\alpha.(\langle s'|X = t \rangle) \not\downarrow$.

We now consider the inductive step. We have the following cases based on the form of s :

- If $s \equiv X$ then $\tau_I(\langle s|X = t \rangle) \equiv \tau_I(\langle X|X = t \rangle)$ and $\tau_I(\langle s|X = \tau_I(t) \rangle) \equiv \tau_I(\langle X|X = \tau_I(t) \rangle)$.

Since $\tau_I(\langle X|X = t \rangle) \xrightarrow{\alpha} u$, it must be $\alpha \notin I$ and $\langle X|X = t \rangle \xrightarrow{\alpha'} v$ with $u \equiv \tau_I(v)$ where: $\alpha' = \alpha$ if $\alpha \neq \tau$, $\alpha' \in I$ otherwise. Furthermore, it must be that $\langle t|X = t \rangle \xrightarrow{\alpha'} v$ by a shorter inference. As a consequence we derive $\tau_I(\langle t|X = t \rangle) \xrightarrow{\alpha} u$. By induction we derive $\tau_I(\langle t|X = \tau_I(t) \rangle) \xrightarrow{\alpha} u''$ with $u'' \stackrel{w}{\Leftarrow} u'$ and $(u, u') \in \beta$. As a consequence $\langle X|X = \tau_I(t) \rangle \xrightarrow{\alpha} u''$ and, since $\alpha \notin I$, $\tau_I(\langle X|X = \tau_I(t) \rangle) \xrightarrow{\alpha} \tau_I(u'')$. Since u'' has hiding as the outermost operator (because is derived by a transition from a term that has hiding as the outermost operator), we also have that $\tau_I(u'')$ is isomorphic to u'' (hence they are weak equivalent).

Concerning termination capability, we have that, by performing the same steps as for the case of transitions, $\tau_I(\langle X|X = t \rangle) \downarrow$ requires $\langle t|X = t \rangle \downarrow$, hence $\tau_I(\langle t|X = t \rangle) \downarrow$ and, by induction, $\tau_I(\langle t|X = \tau_I(t) \rangle) \downarrow$.

- If $s \equiv s' + s''$ then $\tau_I(\langle s|X = t \rangle) \equiv \tau_I(\langle s'|X = t \rangle + \langle s''|X = t \rangle)$ and $\tau_I(\langle s|X = \tau_I(t) \rangle) \equiv \tau_I(\langle s'|X = \tau_I(t) \rangle + \langle s''|X = \tau_I(t) \rangle)$.

Since $\tau_I(\langle s'|X = t \rangle + \langle s''|X = t \rangle) \xrightarrow{\alpha} u$, it must be $\alpha \notin I$ and $\langle s'|X = t \rangle + \langle s''|X = t \rangle \xrightarrow{\alpha'} v$ with $u \equiv \tau_I(v)$ where: $\alpha' = \alpha$ if $\alpha \neq \tau$, $\alpha' \in I$ otherwise. Now we have two cases.

- If $\langle s'|X = t \rangle \xrightarrow{\alpha'} v$ then $\tau_I(\langle s'|X = t \rangle) \xrightarrow{\alpha} u$ and (by induction) $\tau_I(\langle s'|X = \tau_I(t) \rangle) \xrightarrow{\alpha} u''$ with $u'' \stackrel{w}{\Leftarrow} u'$ and $(u, u') \in \beta$. Therefore, it must be $\langle s'|X = \tau_I(t) \rangle \xrightarrow{\alpha'} v''$ with $u'' \equiv \tau_I(v'')$ and: $\alpha'' = \alpha$ if $\alpha \neq \tau$, $\alpha'' \in I$ otherwise. As a consequence, $\langle s'|X = \tau_I(t) \rangle + \langle s''|X = \tau_I(t) \rangle \xrightarrow{\alpha''} v''$ and, finally, $\tau_I(\langle s'|X = \tau_I(t) \rangle + \langle s''|X = \tau_I(t) \rangle) \xrightarrow{\alpha} u''$.
- If $\langle s''|X = t \rangle \xrightarrow{\alpha'} v$ then the result is derived in a similar way.

Concerning termination capability, the proof follows the same steps as for transitions.

- If $s \equiv \langle Y|Y = s' \rangle$, with $Y \neq X$, then $\tau_I(\langle s|X = t \rangle) \equiv \tau_I(\langle Y|Y = \langle s'|X = t \rangle \rangle)$ and $\tau_I(\langle s|X = \tau_I(t) \rangle) \equiv \tau_I(\langle Y|Y = \langle s'|X = \tau_I(t) \rangle \rangle)$.

Since $\tau_I(\langle Y|Y = \langle s'|X = t \rangle \rangle) \xrightarrow{\alpha} u$, it must be $\alpha \notin I$ and $\langle Y|Y = \langle s'|X = t \rangle \rangle \xrightarrow{\alpha'} v$ with $u \equiv \tau_I(v)$ where: $\alpha' = \alpha$ if $\alpha \neq \tau$, $\alpha' \in I$ otherwise. Hence, it must be $\langle \langle s'|X = t \rangle |Y = \langle s'|X = t \rangle \rangle \xrightarrow{\alpha'} v$. As a consequence $\tau_I(\langle \langle s'|X = t \rangle |Y = \langle s'|X = t \rangle \rangle) \equiv$

$\tau_I(\langle\langle s'|Y = s' \rangle|X = t \rangle) \xrightarrow{\alpha} u$. By induction we have $\tau_I(\langle\langle s'|Y = s' \rangle|X = \tau_I(t) \rangle) \xrightarrow{\alpha} u''$ with $u'' \xleftrightarrow{w} u'$ and $(u, u') \in \beta$. Therefore, it must be $\langle\langle s'|Y = s' \rangle|X = \tau_I(t) \rangle \equiv \langle\langle s'|X = \tau_I(t) \rangle|Y = \langle s'|X = \tau_I(t) \rangle \rangle \xrightarrow{\alpha''} v''$ with $u'' \equiv \tau_I(v'')$ and: $\alpha'' = \alpha$ if $\alpha \neq \tau$, $\alpha'' \in I$ otherwise. As a consequence, $\langle Y|Y = \langle s'|X = \tau_I(t) \rangle \rangle \xrightarrow{\alpha'} v''$ and, finally, $\tau_I(\langle Y|Y = \langle s'|X = \tau_I(t) \rangle \rangle) \xrightarrow{\alpha} u''$.

Concerning termination capability, the proof follows the same steps as for transitions.

— If $s \equiv s' \cdot s''$ then $\tau_I(\langle s|X = t \rangle) \equiv \tau_I(s' \cdot \langle s''|X = t \rangle)$ and $\tau_I(\langle s|X = \tau_I(t) \rangle) \equiv \tau_I(s' \cdot \langle s''|X = \tau_I(t) \rangle)$ because X cannot occur inside s' .

Since $\tau_I(s' \cdot \langle s''|X = t \rangle) \xrightarrow{\alpha} u$, it must be $\alpha \notin I$ and $s' \cdot \langle s''|X = t \rangle \xrightarrow{\alpha'} v$ with $u \equiv \tau_I(v)$ where: $\alpha' = \alpha$ if $\alpha \neq \tau$, $\alpha' \in I$ otherwise. Now we have two cases.

- If $s' \xrightarrow{\alpha'} z$ with $v = z \cdot \langle s''|X = t \rangle$ then we have directly that $s' \cdot \langle s''|X = \tau_I(t) \rangle \xrightarrow{\alpha'} z \cdot \langle s''|X = \tau_I(t) \rangle$, hence $\tau_I(s' \cdot \langle s''|X = \tau_I(t) \rangle) \xrightarrow{\alpha} \tau_I(z \cdot \langle s''|X = \tau_I(t) \rangle)$
- If $s' \downarrow$ and $\langle s''|X = t \rangle \xrightarrow{\alpha'} v$ then $\tau_I(\langle s''|X = t \rangle) \xrightarrow{\alpha} u$ and (by induction) $\tau_I(\langle s''|X = \tau_I(t) \rangle) \xrightarrow{\alpha} u''$ with $u'' \xleftrightarrow{w} u'$ and $(u, u') \in \beta$. Therefore, it must be $\langle s''|X = \tau_I(t) \rangle \xrightarrow{\alpha''} v''$ with $u'' \equiv \tau_I(v'')$ and: $\alpha'' = \alpha$ if $\alpha \neq \tau$, $\alpha'' \in I$ otherwise. As a consequence, $\langle s''|X = \tau_I(t) \rangle \xrightarrow{\alpha''} v''$ and, finally, $\tau_I(\langle s''|X = \tau_I(t) \rangle) \xrightarrow{\alpha} u''$.

Concerning termination capability, the proof follows the same steps as for transitions.

— If $s \equiv s' \parallel s''$ or $s \equiv s' \ll s''$ or $s \equiv s' | s''$ or $s \equiv \partial_H(s')$ or $s \equiv \tau_I(s')$ or $s \equiv \rho_f(s')$ then the condition trivially holds because X cannot occur inside s' or s'' .

A completely symmetric inductive proof is performed when we start from α transitions and termination capability of $\tau_I(\langle s|X = \tau_I(t) \rangle)$ in the conditions above. \square

Notice that the axioms are actually valid over TCP+REC, so also on terms that contain recursive specifications which are not essentially finite-state (the axiom *Hid* contains a recursive specification which is not essentially finite-state).

In the following we present the completeness result. First, we need to define normal forms and to present two technical lemmas.

Definition 6.2 *Normal forms* are terms made up of only $\underline{0}, \underline{1}, X, a.t', \tau.t', t'+t''$ and $\langle X|E \rangle$, where E is guarded and contains one equation only.

Lemma 6.3 Any closed normal form t can be turned by the axiomatization in Fig. 2 and 4 into the form

$$\sum_{1 \leq i \leq n} \alpha_i.t_i + \{\underline{1}\}$$

where $\xrightarrow{\alpha_i} t_i$, with $1 \leq i \leq n$, are the outgoing transitions of t (no outgoing transitions corresponds to the sum being $\underline{0}$) and $\underline{1}$ is present if and only if $t \downarrow$, according to the model of transition system defined in Fig. 1 and 3.

Proof. We show that any closed normal form t can be turned by the axiomatization into the form $\sum_{1 \leq i \leq n} \alpha_i.t_i + \{\underline{1}\}$ with the above properties by induction on the maximal

length of the inference trees by which α transitions of t or its termination capability are inferred. From this result we can conclude that the lemma holds for any closed normal form t because, since normal forms include only guarded recursion, for any t we have only a finite number of inference trees yielding outgoing transitions.

The base cases of the induction ($t \equiv \underline{0}$ or $t \equiv \underline{1}$ or $t \equiv \alpha.t'$) are trivial because they are in the desired form already.

The inductive cases are the following ones:

- if $t \equiv t' + t''$ then t can be turned in the desired form by just summing the terms obtained by applying the inductive argument to t' and t'' (t is equated by the axiomatization to such a term by substitutivity of subterms and the operational rules for “+” just gather up the outgoing transitions and the termination capability).
- if $t \equiv \langle X | \{X = t'\} \rangle$ then t can be turned in the desired form by directly considering the term obtained by applying the inductive argument to $\langle t' | \{X = t'\} \rangle$ (t is equated by the axiomatization to such a term by means of the unfolding axiom *Unf* and the operational rules for the recursion operator just leave the outgoing transitions and the termination capability unchanged).

□

Lemma 6.4 Let t', t'' be closed normal forms. $t \equiv t' \cdot t''$ or $t \equiv t' \parallel t''$ or $t \equiv t' \parallel\!\! \parallel t''$ or $t \equiv t' | t''$ or $t \equiv \partial_H(t')$ or $t \equiv \tau_I(t')$ or $t \equiv \rho_f(t')$ can be turned by the axiomatization in Fig. 2 and 4 into the form

$$\sum_{1 \leq i \leq n} \alpha_i.t_i + \{\underline{1}\}$$

where $\xrightarrow{\alpha_i} t_i$, with $1 \leq i \leq n$, are the outgoing transitions of t (no outgoing transitions corresponds to the sum being $\underline{0}$) and $\underline{1}$ is present if and only if $t \downarrow$, according to the model of transition system in Fig. 1 and 3.

Proof. Let t', t'' be closed normal forms and $t'_{next} \equiv \sum_{i \leq n} \alpha'_i.t'_i + \{\underline{1}\}$ and $t''_{next} \equiv \sum_{i \leq m} \alpha''_i.t''_i + \{\underline{1}\}$ be the terms obtained from t', t'' by applying Lemma 6.3.

Let us first consider the case of sequence, i.e. $t \equiv t' \cdot t''$. We initially have

$$t' \cdot t'' = t'_{next} \cdot t'' = \sum_{i \leq n} ((\alpha'_i.t'_i) \cdot t'') + \{\underline{1} \cdot t''\}$$

where the second summand is present if and only if $\underline{1}$ is present in t'_{next} . We therefore have

$$t'_{next} \cdot t'' = \sum_{i \leq n} \alpha'_i.(t'_i \cdot t'') + \{t''_{next}\}$$

It is immediate to observe that, being $\xrightarrow{\alpha'_i} t'_i$, with $i \leq n$, and $\xrightarrow{\alpha''_i} t''_i$, with $i \leq m$, the outgoing transitions of t' and t'' , respectively, and being $\underline{1}$ present in t'_{next} (t''_{next}) if and only if $t' \downarrow$ ($t'' \downarrow$), the arguments of the above sum correspond to the transitions/termination capability derived for t from the operational rules of sequence.

The cases of left merge, i.e. $t \equiv t' \parallel\!\! \parallel t''$, restriction, i.e. $t \equiv \partial_H(t')$, hiding, i.e. $t \equiv \tau_I(t')$,

and relabeling $t \equiv \rho_f(t')$ are similarly proved by performing the following transformations:

$$\begin{aligned}
t' \parallel t'' &= t'_{next} \parallel t'' = \sum_{i \leq n} (\alpha'_i.t'_i \parallel t'') = \sum_{i \leq n} \alpha'_i.(t'_i \parallel t'') \\
\partial_H(t') &= \partial_H(t'_{next}) = \sum_{i \leq n} \partial_H(\alpha'_i.t'_i) + \{\underline{1}\} = \sum_{i \leq n, \alpha'_i \notin H} \alpha'_i.\partial_H(t'_i) + \{\underline{1}\} \\
\tau_I(t') &= \tau_I(t'_{next}) = \sum_{i \leq n} \tau_I(\alpha'_i.t'_i) + \{\underline{1}\} = \sum_{i \leq n, \alpha'_i \in I} \tau.\tau_I(t'_i) + \sum_{i \leq n, \alpha'_i \notin I} \alpha'_i.\tau_I(t'_i) + \{\underline{1}\} \\
\rho_f(t') &= \rho_f(t'_{next}) = \sum_{i \leq n} \rho_f(\alpha'_i.t'_i) + \{\underline{1}\} = \sum_{i \leq n} \rho_f(\alpha'_i).\rho_f(t'_i) + \{\underline{1}\}
\end{aligned}$$

We now consider the case of synchronization merge, i.e. $t \equiv t' | t''$. We initially have:

$$t' | t'' = t'_{next} | t''_{next} = \sum_{i \leq n} \left(\sum_{j \leq m} (\alpha'_i.t'_i | \alpha''_j.t''_j) \right) + \{\underline{1}\}$$

where $\underline{1}$ is present if and only if it is present in both t'_{next} and t''_{next} , hence

$$\begin{aligned}
t'_{next} | t''_{next} &= \\
&\sum_{i \leq n, \alpha'_i = \tau} (\tau.t'_i | t'') + \sum_{j \leq m, \alpha''_j = \tau} (t' | \tau.t''_j) + \sum_{i \leq n, j \leq m, (\alpha'_i, \alpha''_j) \in \text{dom}(\gamma)} \gamma(\alpha'_i, \alpha''_j).(t'_i \parallel t''_j) + \{\underline{1}\}
\end{aligned}$$

We show that we can turn any $t \equiv t' | t''$ into $\sum_{1 \leq i \leq n} \alpha_i.t_i + \{\underline{1}\}$ such that the arguments of the sum correspond to the transitions/termination capability of t by inducing on the following measure: the maximal length of the sequences of τ transitions performable by t' plus the maximal length of the sequences of τ transitions performable by t'' . From this result we can conclude that any such t can be turned in the desired form because, since normal forms include only guarded recursion, t', t'' cannot include cycles of τ loops (and are finite-state), hence the sequences of τ transitions they can perform are bounded.

- The base case of the induction corresponds to such a measure being 0, i.e. both t' and t'' cannot perform τ transitions. This means that when transforming t in the sum-form above we have that the first two sums do not obtain, hence the assert obviously holds.
- The inductive case is performed by just observing that the summands $\tau.t'_i | t''$ and $t' | \tau.t''_j$ obtained by transforming t in the sum-form above can be rewritten into $t'_i | t''$ and $t' | t''_j$, respectively. For such terms we can apply the induction hypothesis and turn them into the form $\sum_{1 \leq i \leq n} \alpha''_i.t''_i + \{\underline{1}\}$ such that the arguments of the sum correspond to their transitions/termination capability. Therefore, since $\xrightarrow{\alpha'_i} t'_i$, with $i \leq n$, and $\xrightarrow{\alpha''_i} t''_i$, with $i \leq m$, are the outgoing transitions of t' and t'' , respectively, since $\underline{1}$ is present in t'_{next} (t''_{next}) if and only if $t' \downarrow$ ($t'' \downarrow$) and since, according to the operational rules for synchronization merge, $t' | t''$ is (additionally) endowed with the transitions/termination capability of $t'_i | t''$ ($t' | t''_j$) whenever $t' \xrightarrow{\tau} t'_i$ ($t'' \xrightarrow{\tau} t''_j$), the arguments of the sum obtained by turning such terms into $\sum_{1 \leq i \leq n} \alpha''_i.t''_i + \{\underline{1}\}$ inside the sum-form above correspond to the transitions/termination capability of $t' | t''$.

Let us then consider the case of parallel operator, i.e. $t \equiv t' \parallel t''$. We initially have

$$t' \parallel t'' = t'_{next} \parallel t'' + t''_{next} \parallel t' + t'_{next} | t''_{next}$$

We then apply the transformation for $t'_{next} \parallel t''$ considered in the proof for the case of left merge (and we also apply it to $t''_{next} \parallel t'$) and the transformation for $t'_{next} | t''_{next}$ considered in the proof for the case of synchronization merge. Here, however, instead of dealing with the first and second sums of the sum form obtained from $t'_{next} | t''_{next}$ by means of an inductive transformation, we just get rid of them as follows. Since, for any i and j , we have $\tau.t'_i | t'' = t'_i | t''$ and $t' | \tau.t''_j = t' | t''_j$ and such terms already occur in the transformation of $t'_{next} \parallel t''$ and $t''_{next} \parallel t'$ (by additionally applying axiom M to parallel) in the form $\tau.(t'_i | t'')$ and $\tau.(t' | t''_j)$, respectively, we derive

$$t' \parallel t'' = \sum_{i \leq n} \alpha'_i.(t'_i \parallel t'') + \sum_{i \leq m} \alpha''_i.(t' \parallel t''_i) + \sum_{i \leq n, j \leq m, (\alpha'_i, \alpha''_j) \in \text{dom}(\gamma)} \gamma(\alpha'_i, \alpha''_j).(t'_i \parallel t''_j) + \{\underline{1}\}$$

where, in the second sum, we also have exploited the commutativity of “ \parallel ”. It is immediate to observe that, being $\xrightarrow{\alpha'_i} t'_i$, with $i \leq n$, and $\xrightarrow{\alpha''_i} t''_i$, with $i \leq m$, the outgoing transitions of t' and t'' , respectively, and being $\underline{1}$ present in t'_{next} (t''_{next}) if and only if $t' \downarrow$ ($t'' \downarrow$), the arguments of the above sum correspond to the transitions/termination capability derived for t from the operational rules of parallel. \square

Proposition 6.5 The axiomatization formed by the axioms in Fig. 2 and by the axioms in Fig. 4 is ground-complete for $\text{TCP}+\text{REC}_f$ and the model of transition systems modulo observational congruence generated by the rules in Fig. 1 and 3.

Proof. We show, by structural induction over the syntax of (possibly open) terms t of $\text{TCP}+\text{REC}_f$ whose free variables do not occur in the scope of one of the operators $\parallel, \ll, |, \partial_H, \tau_I, \rho_f$ or on the left-hand side of the operator \cdot , that t can be turned into normal form (that is closed if t is closed). 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 $\underline{0}$ and $\underline{1}$, instead of just one) and completeness over such terms has been proved in (Milner 1989b). Since we do not have \cdot or \parallel operators in normal forms the presence of the two ways of termination does not change the proof: it is just sufficient to consider $\underline{1}$ as a distinguished prefix followed by $\underline{0}$.

The base cases of the induction ($t \equiv \underline{0}$ or $t \equiv \underline{1}$ 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 $t \equiv t' \ll t''$ 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 the closed term t''' obtained by replacing both t' and t'' inside t has a finite transition system. Let $t_1 \dots t_n$ be the states of the transition system of t''' , $t_n \equiv t'''$. It can be

easily seen that, due to Lemma 6.4, for each $i \in \{1 \dots n\}$, there exist m_i , $\{\alpha_j^i\}_{j \leq m_i}$ (denoting actions), $\{k_j^i\}_{j \leq m_i}$ (denoting natural numbers) such that we can derive $t_i = \sum_{j \leq m_i} \alpha_j^i \cdot t_{k_j^i} + \{\underline{1}\}$. Hence we can characterize the behavior of t''' by means of a set of equations ($t_1 \dots t_n$ are the solution of a regular recursive specification with n variables). We can, therefore, turn t''' in normal form similarly as done in (Milner 1989b) in the proof of the unique solution of guarded sets of equations theorem. In particular, we show that there is a term t'''' in normal form such that we can derive $t'''' = t_n \equiv t'''$ as follows. For each i , from 1 to n , we do the following. If i is such that $\exists j \leq m_i : k_j^i = i$ we have, by applying *Fold*, that $t_i = \langle X | X = \sum_{j \leq m_i : k_j^i \neq i} \alpha_j^i \cdot t_{k_j^i} + \sum_{j \leq m_i : k_j^i = i} \alpha_j^i \cdot X + \{\underline{1}\} \rangle$. Note that axiom *Fold* is applicable because, by exploiting the inductive argument, t' and t'' have been turned 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, i.e. according to the definition in (Milner 1989b) the equation set considered is guarded. Then we replace each subterm t_i occurring in the equations for $t_{i+1} \dots t_n$ 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 main difference is that t'' may include free variables. Let t''' be the term obtained from t by replacing t' and t'' with their normal form, obtained by exploiting the inductive argument. Let us denote with $c(t''')$ the closed term obtained from t''' by replacing each free occurrence of a variable X by $a_X \cdot \underline{0}$, where a_X is a fresh action. Note that in t''' free variables may occur inside the normal form of t'' only, i.e. in the subterm to the right of “.”. We know that $c(t''')$ has a finite transition system, hence $c(t''')$ can be turned into normal form with the procedure of the previous item: by considering its states, by transforming them in sum of prefixes leading to other states using Lemma 6.4 (note that in this case only states containing the “.” operator need to be transformed in this way) and by then deriving an equivalent term in normal form by applying the *Fold* axiom. The normal form t'''' for the open term t''' is obtained by following exactly the same derivation procedure (and therefore applying the same axioms) as described above for deriving a normal form from the corresponding term $c(t''')$ (that yields a corresponding normal form $c(t''''')$). In particular, a set of open terms $t_1 \dots t_n$ must be considered such that $c(t_1) \dots c(t_n)$ are the states of $c(t''')$ and for each of them a transformation into a sum of prefixes and open variables $t_i = \sum_{j \leq m_i} \alpha_j^i \cdot t_{k_j^i} + \sum_{j \leq m_i} X_j^i + \{\underline{1}\}$ is obtained by following exactly the same derivation procedure as described in Lemma 6.4 that allows us to correspondingly derive $c(t_i) = \sum_{j \leq m_i} \alpha_j^i \cdot c(t_{k_j^i}) + \sum_{j \leq m_i} a_{X_j^i} \cdot \underline{0} + \{\underline{1}\}$. It is possible to follow the same derivation procedure when X_j^i variables replace $a_{X_j^i} \cdot \underline{0}$ prefixes because variables X_j^i cannot occur inside t_i in the subterm to the left of “.” (which is a closed subterm), hence in the derivation of Lemma 6.4 the axiom *A10* (that is used to move prefixes from inside to outside of a “.” operator, representing their execution) is never applied to an $a_{X_j^i}$ prefix. It is immediate to see that all the other axioms used in the derivation are still applicable when X_j^i variables replace $a_{X_j^i} \cdot \underline{0}$ prefixes.

- 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 (because it is in normal form).

We first show, by structural induction, that for any (possibly open) normal form t'' , $\tau_I(t'')$ 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 ($t'' \equiv \underline{0}$ or $t'' \equiv \underline{1}$ 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*.

Notice that, due to the usage of axiom *Hid* in the last item, the equational transformation procedure from $\tau_I(t'')$ to $\tau_I(t''')$ arising from the above induction works on TCP+REC.

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 \parallel 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.

□

7. 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 rooted branching bisimilarity instead of considering observational congruence. In particular we claim that we can find a ground-complete axiomatization for essentially finite-state behaviours modulo branching bisimilarity by taking the axiomatization of (van Glabbeek 1993), extending the syntax as we have done, and adding as only extra axiom our axiom (*Hid*).

References

- J. C. M. Baeten and M. Bravetti. A ground-complete axiomatization of finite state processes in process algebra. In M. Abadi and L. de Alfaro, editors, *Proc. of the 16th International Conference on Concurrency Theory (CONCUR 2005) San Francisco, CA, USA*, volume 3653 of *LNCS*, pages 248–262. Springer, 2005.
- J. C. M. Baeten and M. Bravetti. A generic process algebra. In L. Aceto and A. D. Gordon, editors, *Proc. of the Workshop Essays on Algebraic Process Calculi (APC 25)*, volume 162 of *Electr. Notes Theor. Comput. Sci.*, pages 65–71. Elsevier, 2006.
- J. C. M. Baeten and R.J. van Glabbeek. Merge and termination in process algebra. In K.V. Nori, editor, *Proc. 7th Conf. on Foundations of Software Technology and Theoretical Computer Science, Pune, India*, volume 287 of *LNCS*, pages 153–172. Springer, 1987.
- 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.
- J.C.M. Baeten, T. Basten, and M.A. Reniers. *Process algebra (equational theories of communicating processes)*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2008.
- J.C.M. Baeten and J.A. Bergstra. Process algebra with propositional signals. *Theoretical Computer Science*, 177(2):381–406, 1997.
- 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.
- J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1/3):109–137, 1984.
- J.A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985.
- 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.

- 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.
- 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.
- S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984.
- 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.
- 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.
- 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>.
- R.J. van Glabbeek. Notes on the methodology of CCS and CSP. *Theoretical Computer Science*, 177(6):329–349, 1997.
- R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996.
- C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.
- R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
- R. Milner. A complete axiomatization for observational congruence of finite-state behaviours. *Information and Computation*, 81:227–247, 1989.
- D. Sangiorgi and R. Milner. The problem of “weak bisimulation up to”. In R. Cleaveland, editor, *Proceedings CONCUR'92*, number 630 in *LNCS*, pages 32–46. Springer Verlag, 1992.
- F.W. Vaandrager. Verification of two communication protocols by means of process algebra. Technical Report report CS-R8608, CWI Amsterdam, 1986.