A ground-complete axiomatization of finite state processes in process algebra

Citation for published version (APA):

Document status and date:
Published: 01/01/2005

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
A Ground-Complete Axiomatization of Finite State Processes in Process Algebra

J.C.M. Baeten
Divison of Computer Science, Technische Universiteit Eindhoven, jsb@win.tue.nl
and
M. Bravetti
Department of Computer Science, Università di Bologna, bravetti@cs.unibo.it

Abstract

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

1 Introduction

The problem of developing a sound and complete axiomatization for a weak form of bisimulation (abstracting from internal \(\tau\) activities) over a process algebra expressing finite-state processes with both guarded and (weakly and fully) unguarded recursion has been solved by Robin Milner [15]. His solution has been developed in the context of a basic process algebra (basic CCS) made up of visible prefix \(a.t\), silent prefix \(\tau.t\), summation \(t' + t''\) and recursion \(recX.t\) (based on least transition system solution), whose model is assumed to be finite-state transition systems modulo observational congruence (rooted weak bisimulation). Such a solution is crucially based on three axioms: one for fully unguarded recursion

\[(F\text{Ung}) \quad recX.(X + t) = recX.t\]

and two for weakly unguarded recursion

\[(W\text{Ung}1) \quad recX.((\tau.X + t) = recX.\tau.t)
\[(W\text{Ung}2) \quad recX.((\tau.(X + t) + s) = recX.\tau.(X + t + s)).\]

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

\[(\text{Unfold}) \quad \text{rec}X.t = t\{\text{rec}X.t/X\}\]
\[(\text{Fold}) \quad t' = t\{t'/X\} \Rightarrow t' = \text{rec}X.t \quad \text{if} \ X \ \text{is guarded in} \ t\]

However Milner’s result is crucially based on the fact that the signature of the process algebra under consideration is very simple. For example if we extend the signature to full CCS (by e.g. considering parallel composition and restriction), we have that the axioms above are no longer sufficient to get rid of unguarded recursion. In other words, even if two CCS terms are both finite-state it may be that they are not equated by an axiomatization including the standard CCS axioms (the axioms for CCS without the \text{rec}X.t recursion operator) plus the axioms for unguarded and guarded recursion above. An example is the following:

\[
( (\text{rec}X.a.X) \mid (\text{rec}X.a.X) ) \setminus a
\]

where “\mid” and “\setminus” denote CCS parallel composition and restriction, respectively. The model of such a term has just one state with a \(\tau\) self-loop, but cannot be equated by the axiomatization to the equivalent term recX.\(\tau\).X or to \(\tau.\). The problem is that, since the process above produces unguarded recursion (a loop with only \(\tau\) transitions in the transition system), we cannot apply the folding axiom (\text{Fold}). We should first remove unguarded recursion, but the three axioms (\text{FUNg}, (WUNg1), (WUNg2) only work with the restricted signature (which does not include the parallel and restriction operators).

In this paper we consider a generic process algebra of which the standard process algebra ACP, CCS and CSP are subalgebras of reduced expressions. More precisely such an algebra is an extension of the algebra TCP [1, 2] (which extends ACP by including successful termination \(\epsilon\) and prefixing à la CCS) with a recursion operator \(\langle X | E \rangle\) which computes minimal fixpoint solutions of systems of equations (denoted by \(E = \{X = t_X, Y = t_Y, \ldots\}\) over processes and consider an initial variable \(X\) among variables \(V\) defined by the system of equations \(E\). Such an operator (which extends the similar operator introduced in [7] with the possibility of nested recursion operators inside recursion operators) encompasses both the CCS \text{rec}X.t operator (which is obtained by taking \(E = \{X = t\}\)) and the standard way to express recursion in ACP (where usually only guarded recursion is considered via systems of equations \(E\)). As we will see, such an algebra, called TCP+REC, is endowed with sequencing “\(t' : t''\)”, hiding “\(\tau(t)\)”, restriction “\(\partial_I(t)\)”, relabeling “\(\rho_I(t)\)”, and parallel composition “\(t' \parallel t''\)” à la ACP (where a communication function \(\gamma\) is assumed to compute the type of communicating actions).

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

In order to guarantee that transition systems generated by the operational semantics are indeed finite-state, we provide a syntactical constraint for the systems of equations \(E = E(V)\) involved in recursion operators \(\langle X | E \rangle\). Such a constraint is similar to that considered in [8]: in essence we disallow variables in \(V\) occurring in the right-hand side of equations in \(E\) (that are bound by the \(\langle X | E \rangle\) operator) to be in the scope of static operators like hiding, restriction, relabeling and parallel composition or in the left-hand side of a sequencing operator. For example \(\langle X | \{X = \tau_I(a.X)\} \rangle\) for any hiding set \(I\), which produces an infinite-state transition system, is a term rejected by the constraint that we consider. Note however that recursion can be included in the scope of static operators (or in the left-hand side of sequencing) as in the case of the CCS term \(( (\text{rec}X.a.X) \mid (\text{rec}X.a.X) ) \setminus a\) 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 a static operator or in the lefthand-side of sequencing then it produces an infinite-state transition system. We call TCP+RECf 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+RECf.

The main result of the paper is the introduction of an axiomatization that is ground-complete over the signature of TCP+RECf: 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

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

which allows the hiding operator (the only static operator which may generate unguarded recursion) to be exchanged with the recursion operator. We will show that by using such a crucial axiom, that was previously considered also in [13] (where the author just showed it to be sound), it is possible to achieve completeness in the finite-state case when static operators are considered, thus extending Milner’s result. The main idea is that, by means of this axiom, we can first move the hiding operator inside recursion and more generally outside-in traversing the whole syntactic structure of the term considered (so to get the effect of hiding on the actions syntactically occurring in the term), and then (by applying it in the reversed way) inside-out again. Supposing that we are turning the term into normal form (essentially basic CCS where recursion is guarded) by means of syntactical induction, once we have done the procedure above we can apply Milner’s rule for unguarded recursion in the term inside the hiding operator, thus getting a term in normal form on which the hiding operator has no longer any effect. As a consequence we can get rid of it like we do with any other static operator by using the Fold axiom.

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

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

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

1.1 Acknowledgements

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

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

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

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

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

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

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

**Definition 2 (Weak Bisimilarity)** Define \( s \Rightarrow t \) if there is a sequence of 0 or more \( \tau \)-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 \( s, t, s', t' \in S \), 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 \( s \bowtie w t \), 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 \( s \bowtie RWt \), if and only there is a weak bisimulation relation in which the pair \((s, t)\) satisfies the root condition.

3 Process Algebra

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

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

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

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

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

This process algebra is generic, in the sense that most features of commonly used process algebras can be embedded in it. In the following, we made use of [12, 13] and [4].

We consider a subtheory corresponding to CCS, see [16]. This is done by omitting the signature elements $ε, \cdot, \|_L, \parallel$. Next, we specialize the parameter set $A$ by separating it into three parts: a set of names $A$, a set of co-names $\bar{A}$ and a set of communications $A^*$ such that for each $a \in A$ there is exactly one $\bar{a} \in \bar{A}$ and exactly one $a^* \in A^*$. The communication function $γ$ is specialized to having as the only defined communications $γ(a, \bar{a}) = γ(\bar{a}, a) = a^*$, and then the CCS parallel composition operator $\parallel_{CCS}$ can be defined by the formula

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

We consider a subtheory corresponding to ACP, see [6]. This is done by defining, for each $a \in A$, a new constant $a$ by $a = a.e$, and then omitting the signature elements $ε, δ, ρ_f$.

We consider a subtheory corresponding to CSP, see [14]. The non-deterministic choice operator $\sqcup$ can be defined by

$$x \sqcup y = τ.x + τ.y,$$

but the external choice operator $\Box$ cannot be defined directly, as possible non-determinism is removed at the start of the process. It can be axiomatized as shown by Brookes in [9]. The parameter set $A$ is specialized into two parts: a set of names $A$ and a set of communications $A^*$ such that for each $a \in A$ there is exactly one $a^* \in A^*$. The communication function $γ$ is specialized to having as the only defined communications $γ(a, a) = a^*$, and further, we use the renaming function $f$ that has $f(a^*) = a$. Then, the CSP parallel composition operator
\[
\begin{align*}
\epsilon & \downarrow \\
x \overset{\alpha}{\rightarrow} x' & \quad \alpha, x \overset{\alpha}{\rightarrow} x \\
x + y \overset{\alpha}{\rightarrow} x' & \quad x + y \overset{\alpha}{\rightarrow} y' \\
x \overset{\alpha}{\rightarrow} x' & \quad x \downarrow, y \overset{\alpha}{\rightarrow} y' \\
x \cdot y \overset{\alpha}{\rightarrow} x' \cdot y & \quad x \cdot y \overset{\alpha}{\rightarrow} y' \\
x \overset{\alpha}{\rightarrow} x', y \overset{b}{\rightarrow} y', \gamma(a, b) = c & \quad x \downarrow, y \overset{\alpha}{\rightarrow} y' \\
x \parallel y \overset{\tau}{\rightarrow} x' \parallel y' & \quad x \parallel y \overset{\alpha}{\rightarrow} x' \parallel y \\
x \overset{\alpha}{\rightarrow} x', y \overset{b}{\rightarrow} y', \gamma(a, b) = c & \quad x \parallel y \overset{\alpha}{\rightarrow} x' \parallel y \\
x \parallel y \overset{\tau}{\rightarrow} x' \parallel y' & \quad x \parallel y \overset{\alpha}{\rightarrow} x' \parallel y \\
x \overset{\alpha}{\rightarrow} x', x' \mid y \overset{\alpha}{\rightarrow} z & \quad y \overset{\tau}{\rightarrow} y', x \mid y \overset{\alpha}{\rightarrow} z \\
x \mid y \overset{\alpha}{\rightarrow} z & \quad y \overset{\tau}{\rightarrow} y', x \mid y \overset{\alpha}{\rightarrow} z \\
\partial_H(x) \overset{\alpha}{\rightarrow} \partial_H(x') & \quad x \downarrow \quad \partial_H(x) \downarrow \\
x \overset{\alpha}{\rightarrow} x', \alpha \not\in H & \quad \tau_I(x) \overset{\alpha}{\rightarrow} \tau_I(x') \\
x \downarrow & \quad x \overset{\tau}{\rightarrow} x' \\
\rho_f(x) \overset{[\alpha]}{\rightarrow} \rho_f(x') & \quad \rho_f(x) \overset{[\alpha]}{\rightarrow} \rho_f(x') \\
\tau_I(x) & \quad x \downarrow \\
\rho_f(x) & \quad \rho_f(x) \downarrow
\end{align*}
\]

Table 1: Deduction rules for TCP.

\|s, parametrized by a set of names \(S \subseteq \mathcal{A}\), can be defined by the formula
\[x \parallel y = \rho_f(\partial_S(x \parallel y)).\]

4 Recursion

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

Let \(V\) be a set of variables ranging over processes, ranged over by \(X, Y\). According to a terminology which is usual in the ACP setting, a recursive specification \(E = E(V)\) is a set of equations \(E = \{X = t_X \mid X \in V\}\) where each \(t_X\) is a term over the signature in question and variables from \(V\). A solution of a recursive specification \(E(V)\) is a set of transition systems \(\{y_X \mid X \in V\}\) such that the equations of \(E(V)\) correspond to equivalent transition systems, if for all \(X \in V\), \(y_X\) is substituted for \(X\). Mostly, we are interested in one particular variable \(X \in V\), called the initial variable.

Let \(t\) be a term containing a variable \(X\). We call an occurrence of \(X\) in \(t\) guarded if this occurrence of \(X\) is in the scope of an action prefix operator (not \(\tau\) 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.
\[
x + y = y + x \quad \text{A1} \quad x \parallel y = x \parallel y \parallel x + x \parallel y \parallel M
\]
\[
(x + y) + z = x + (y + z) \quad \text{A2}
\]
\[
x + x = x \quad \text{A3} \quad \delta \parallel x = \delta \quad \text{LM1}
\]
\[
(x + y) \cdot z = x \cdot z + y \cdot z \quad \text{A4} \quad \epsilon \parallel x = \delta \quad \text{LM2}
\]
\[
(x \cdot y) \cdot z = x \cdot (y \cdot z) \quad \text{A5} \quad \alpha \cdot x \parallel y = \alpha \cdot (x \parallel y) \quad \text{LM3}
\]
\[
x + \delta = x \quad \text{A6} \quad (x + y) \parallel z = x \parallel z + y \parallel z \quad \text{LM4}
\]
\[
\delta \cdot x = \delta \quad \text{A7}
\]
\[
\epsilon \cdot x = x \quad \text{A8} \quad x \parallel y = y \parallel x \quad \text{SM1}
\]
\[
x \cdot \epsilon = x \quad \text{A9} \quad \delta \parallel x = \delta \quad \text{SM2}
\]
\[
(\alpha x) \cdot y = \alpha \cdot (x \cdot y) \quad \text{A10} \quad \epsilon \parallel \epsilon = \epsilon \quad \text{SM3}
\]
\[
\partial H(\delta) = \delta \quad \text{D1} \quad \alpha x \parallel b y = c \cdot (x \parallel y) \quad \text{if } \gamma(a, b) = c \quad \text{SM4}
\]
\[
\partial H(\epsilon) = \epsilon \quad \text{D2} \quad \alpha x \parallel \epsilon = \delta \quad \text{SM5}
\]
\[
\partial H(\alpha x) = \delta \quad \text{if } a \in H \quad \text{D3} \quad (x + y) \parallel z = x \parallel z + y \parallel z \quad \text{SM7}
\]
\[
\partial H(\alpha x) = \alpha \cdot \partial H(x) \quad \text{otherwise} \quad \text{D4}
\]
\[
\partial H(x + y) = \partial H(x) + \partial H(y) \quad \text{D5} \quad \rho f(\delta) = \delta \quad \text{RN1}
\]
\[
\rho f(\epsilon) = \epsilon \quad \text{RN2}
\]
\[
\tau I(\delta) = \delta \quad \text{TI1} \quad \rho f(\alpha x) = f(a) \cdot f(x) \quad \text{RN3}
\]
\[
\tau I(\epsilon) = \epsilon \quad \text{TI2} \quad \rho f(\tau x) = \tau \cdot f(x) \quad \text{RN4}
\]
\[
\tau I(\alpha x) = \tau \cdot \tau_I(x) \quad \text{if } a \in I \quad \text{TI3} \quad \rho f(x + y) = \rho f(x) + \rho f(y) \quad \text{RN5}
\]
\[
\tau I(\alpha x) = \alpha \cdot \tau_I(x) \quad \text{otherwise} \quad \text{TI4}
\]
\[
\tau I(x + y) = \tau_I(x) + \tau_I(y) \quad \text{TI5}
\]
\[
\alpha \cdot \tau x = \alpha x \quad \text{T1} \quad \tau x + x = \tau x \quad \text{T2}
\]
\[
\alpha(\tau x + y) = \alpha(\tau x + y) + \alpha x \quad \text{T3} \quad \tau x \parallel y = x \parallel y \quad \text{T4}
\]

Table 2: Axioms of TCP.

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

The process algebras ACP, CCS and CSP handle this situation in different ways. In ACP, variables occurring in unguarded recursive specifications are treated as (constrained) variables, and not as processes. In CCS, where recursive specifications are made via so-called “constants”, ranged over by \( A, B, \ldots \), or equivalently by the \( \text{rec}X.t \) operator, where \( t \) is a term containing variable \( X \), from the set of solutions the solution will be chosen that has the least transitions in the generated transition system. Thus, the solution chosen for the equation \( \{ X = X \} \) has no transitions, is the process \( \delta \), and the solution chosen for \( \{ X = \tau X \} \) has only a \( \tau \)-transition to itself, a process that is bisimilar to \( \tau \cdot \delta \) in observational congruence.
Finally, also in CSP a solution will be chosen, but a different one, the least deterministic one. Thus, both CCS and CSP use a least fixed point construction, but with respect to a different ordering relation. In CSP, the solution chosen for the equation \( \{X = X\} \) is the \emph{chaos} process \( \bot \), a process that satisfies \( x + \bot = \bot \) for all processes \( x \) (for an extension of TCP with such a process, see [2], based on [3]).

Here we will introduce in TCP the possibility of performing (not guarded) recursive specifications by means of an operator \( \langle X | E \rangle \) (where \( E = E(V) \) is a recursive specification and \( X \) a variable in \( V \) which acts as the initial variable) which, similarly as in CCS, yields the least transitions in the generated transition system. Note that our approach also encompasses recursive specifications in ACP which are usually assumed to be guarded. The extended signature gives rise to a process algebra that we call TCP+REC.

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

\[
t ::= \delta \mid \epsilon \mid a.t \mid \tau.t \mid t + t \mid t \cdot t \mid t \parallel t \mid t \downarrow t \mid t \mid \partial_H(t) \mid \gamma_I(t) \mid \rho_f(t) \mid X \mid \langle X | E \rangle
\]

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

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

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

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

Table 3 provides deduction rules for recursive specifications. Such rules are similar to those in [10], but we have the additional possibility of nesting recursion operators inside recursion operators. They come down to looking upon \( \langle X | E \rangle \) as the process \( \langle t_X | E \rangle \), which is defined as follows.

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

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

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

variable \( X \) of \( a.Y[\{Y = X + Y\}] \) is replaced by \( \langle X \mid \{X = a.Y[\{Y = X + Y\}]\} \rangle \) yielding:

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

\[
\langle t_X | E \rangle \xrightarrow{a \cdot y} \langle X | E \rangle \langle t_X | E \rangle \downarrow \langle X | E \rangle \downarrow
\]

Table 3: Deduction rules for recursion.

Together Table 1 and Table 3 provide a transition system space over the signature of TCP+REC.
In order to remain in the setting of processes with a finite-state model we now consider a
restricted syntax for constants \( \langle X|E \rangle \) which guarantees that transition systems generated by
the operational rules are indeed finite-state.

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

\[
X = \sum_{1 \leq i \leq n} \alpha_i X_i + \{ \epsilon \},
\]

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

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

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

**Proof** 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 an occurrence of a fresh action
\( a_X \).

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 \( t \equiv \delta \), then \( c(t) = \delta \) is obviously finite-state.
- if \( t \equiv \epsilon \), then \( c(t) = \epsilon \) is obviously finite-state.
- if \( t \equiv X \), then \( c(t) = a_X \) is obviously finite-state.

The inductive cases of the induction are the following ones:

- if \( t \equiv a \cdot t' \) or \( t \equiv \tau \cdot t' \) or \( t \equiv t' + t'' \) or \( t \equiv t' \cdot t'' \) or \( t \equiv t' \| t'' \) 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:

\[
S(\langle X|E \rangle) \subseteq \{ \langle X|E \rangle \} \cup \text{ren}( \bigcup_{Y \in V} S(c(t_Y)) )
\]
where \( \text{ren}(S) \) is a renaming function for terms which, for any \( Y \in V \), replaces every occurrence of \( a_Y \) with \( \langle Y | E \rangle \). Once proven 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 equation above indeed holds. First of all we assume that in \( \langle X | E \rangle \) bound variables inside \( E \) are \( \alpha \)-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 structural induction on \( t \), that for every transition \( t \rightarrow t' \) derived with the operational semantics it holds that:

- if there is no \( Y \in V \) such that \( \langle Y | E \rangle \) is included in \( t \) then there is no \( Y \in V \) such that \( \langle Y | E \rangle \) is included in \( t' \).
- if there is no \( Y \in V \) such that \( \langle Y | E \rangle \) is included in \( t \) inside the scope of one of the operators \( \parallel, \perp, |, \partial_H, \tau_I, \rho_f \) 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 \( t' \) inside the scope of one of the operators \( \parallel, \perp, |, \partial_H, \tau_I, \rho_f \) 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 \( t \) implies \( E'' = E \), then for every \( Y \in V \) it holds that the occurrence of \( \langle Y | E'' \rangle \) in \( t' \) 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 for \( t \), we have that the following holds:

\( t \in S(\langle X | E \rangle) \) implies:

- there is no \( Y \in V \) such that \( \langle Y | E \rangle \) is included in \( t \) inside the scope of one of the operators \( \parallel, \perp, |, \partial_H, \tau_I, \rho_f \) 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 \( t \) implies \( E'' = E \).

Next, given any transition \( t' \rightarrow t'' \), we define the auxiliary function \( \text{var}(t' \rightarrow t'') \) as follows:

- \( \text{var}(t' \rightarrow t'') = \perp \) if \( t' \rightarrow t'' \) is derived by using no operational rule of any \( \langle Y | E \rangle \), with \( Y \in V \).
- \( \text{var}(t' \rightarrow t'') = Y \) if \( t' \rightarrow t'' \) is derived by using the operational rule of \( \langle Y | E \rangle \) as the innermost operational rule applied for a recursion operator.

Note that, for the properties above of any \( t \in S(\langle X | E \rangle) \) it is not possible to derive \( t' \rightarrow t'' \) by means of more than one operational rule of any \( \langle Y | E \rangle \), with \( Y \in V \) that belong to different derivation branches (i.e. we are always in one of the two situations above).

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

Supposed that we are not in the first case, then given the derivation for \( t \), we consider the last transition \( t'' \rightarrow t''' \) such that \( \text{var}(t' \rightarrow t'') \neq \perp \) (we are sure that such a transition exists because the first transition in the derivation is of this kind). So let us consider the variable \( Y = \text{var}(t' \rightarrow t'' \rangle \) for such a transition. It is easy to see that there exists \( t''' \) such that \( c(t''' \rangle \) is derivable from \( c(t_Y) \) and \( t = \text{ren}(c(t''' \rangle) \). This is because:
- $t''$ is such that $\text{ren}(c(t')) \rightarrow t''$ (from the fact that $Y = \text{var}(t' \rightarrow t'')$ and for the properties above of any $t \in S(\langle X|E \rangle)$).

- given any $t_1, t_2$ such that $\text{var}(\text{ren}(c(t_1)) \rightarrow t_2) = \bot$, there exists $t_3$ such that $t_2 = \text{ren}(c(t_3))$ and $c(t_1) \rightarrow c(t_3)$ (by induction on the structure of $t_1$ inspecting each operational rule).

\[ \square \]

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 bisimulation. 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 $X = (a.e) + (b.X \cdot X)$, we do obtain infinitely many terms that are not bisimilar. However, note that the axioms that we will present in the following Sect. 5 remain valid, also if recursive specifications that are not essentially finite state are considered (obtaining a model of possibly infinite transition systems modulo bisimulation).

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

**Proposition 6** Let $t$ be a (closed) term that includes a recursion $\langle X|E \rangle$ such that the recursive specification $E$ has finitely many equations but is not essentially finite state. If one of the occurrences of variables in $V$ which violate the condition, i.e. which are in the scope of one of the operators $\parallel, \top, \bot, \partial, \tau, \rho$ or on the left-hand side of the operator $\cdot$, is reachable from $\langle X|E \rangle$, then $t$ has infinitely many states.

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

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

In the rest of this paper, we will consider system specifications made by using the process algebra TCP+$\text{REC}_f$ obtained by extending the signature of TCP with essentially finite state recursive specifications, i.e. we consider closed terms in the syntax above, where we additionally require that every recursive specification included is essentially finite-state.

5 Axiomatization

Now we will present a sound axiomatization which is ground-complete for the process algebra TCP+$\text{REC}_f$. The axioms in Table 2 together with the axioms in Table 4 form such an
\[ \langle X | E \cup \{ Y = t \} \rangle = \langle X | E \{ Y | Y = t \} / Y \rangle \] if \( X \neq Y \) \hspace{1cm} \text{Dec} \\
\[ \langle X | X = t \rangle = t \{ \langle X | X = t \} / X \} \] \hspace{1cm} \text{Unf} \\
y = t \{ y / X \} \Rightarrow y = \langle X | X = t \rangle \] if \( X = t \) guarded \hspace{1cm} \text{Fold} \\
\[ \langle X | X = X + t \rangle = \langle X | X = t \rangle \] \hspace{1cm} \text{Ung} \\
\[ \langle X | X = \tau.(X + t + s) \rangle = \langle X | X = \tau.(t + s) \rangle \] \hspace{1cm} \text{WUng} \\
\[ \tau_I(\langle X | X = t \rangle) = \langle X | X = \tau_I(t) \rangle \] \hspace{1cm} \text{Hid} 

Table 4: Axioms for recursion.

Axiomatization. In the axioms of Table 4 we use the usual operation \( \{ t / X \} \) for expressing syntactical replacement of a closed term \( t \) for every free occurrence of variable \( X \). Such an operation can be applied to a term \( t' \) or to the righthand-side of all equations in a recursive specification \( E = E(V) \) such that \( X \not\in V \) by writing \( t' \{ t / X \} \) and \( E \{ t / X \} \), respectively. Moreover the symbol \( \cup \) stands for disjoint union. Note that the axioms in Table 4 are axiom schemes: we have these axioms for each possible term \( t \).

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

\[
\langle X | \{ X = a.X + b.Y, Y = c.X + d.Y \} \rangle
\]

is turned into

\[
\langle X | \{ X = a.X + b.Y \{ Y = c.X + d.Y \} \} \rangle
\]

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

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

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

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

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

by directly applying (WUng) and then

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

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

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

Note that if we want to derive a ground-complete axiomatization in a setting where no construct is added for recursion, as usually done in the context of the ACP process algebra (so we just have closed terms over the syntax of TCP and we just consider sets of recursion equations over this syntax), then in order to achieve the effect of our axiom $H_{id}$ we have to add a much more complex set of conditional equations called CFAR (Cluster Fair Abstraction Rule) introduced in [17]. CFAR is a generalisation of the KFAR (Kozen’s Fair Abstraction Rule) introduced in [5].

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

**Proof** Most of the axioms are standard. See [13] for the axiom $(H_{id})$. \[\square\]

Notice that the axioms are actually valid on TCP+REC (the axiom $H_{id}$ contains a recursive specification which is not essentially finite state), so also on terms that contain recursive specifications which are not essentially finite state.

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

**Proof** We show, by structural induction over the syntax of (possibly open) terms $t$ of TCP+REC, such that free variables do not occur in the scope of one of the operators $\parallel, \perp, \delta, \tau, \rho$ or on the left-hand side of the operator $\cdot$, that $t$ can be turned into normal form, where normal forms are defined as follows. A term is normal form if it is made up of only $\delta, \epsilon, X, a, a', \tau, t', t + t''$ and $(X | E)$, where $E$ is guarded and contains one equation only. Proving this yields ground-completeness; this because normal forms are like terms of basic CCS (with the only difference that we have two non equivalent kinds of terminating processes $\delta$ and $\epsilon$, instead of just one) and completeness over such terms has been proved by Milner (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).

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

The inductive cases of the induction are the following ones:

- if $t \equiv a.a'$ 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 $t \equiv t' \perp t''$ or $t \equiv t' | t''$ or $t \equiv (X | E)$, then we can turn $t$ into normal form as follows. By exploiting the inductive argument over $t'$ and $t''$, and by observing that $t$ cannot include free variables, we know that $t$ has a finite transition system. Let $t_1, \ldots, t_n$ be the states of the transition system of $t$, $t_n \equiv t$. It can be easily seen that, for each $i \in \{1 \ldots n\}$, there exist $m_i, \{a^i_j\}_{j \leq m_i}$ (denoting actions), $\{k^i_j\}_{j \leq m_i}$ (denoting natural numbers) s.t. we can derive $t_i = \sum_{j \leq m_i} a^i_j t_{k^i_j} + \{\epsilon\}$. Hence we can characterize the behavior of $t$ by means of a set of equations similarly as in [15]. Moreover, similarly as for the unique solution of equations theorem of [15], we have that there is a term $t''$ in normal form such that we can derive $t'' = t_n \equiv t$. This can
be shown as follows. For each $i$, from 1 to $n$, we do the following. If $i$ is such that $\exists j \leq m_i : k_j^i = i$ we have, by applying $Fold$, that $t_i = \langle Y \mid X = \sum_{j \leq m_i} a_{k_j^i} t_{k_j^i} + \sum_{j \leq m_i} a_{k_j^i} X + \{e\} \rangle$. Note that axiom $Fold$ is applicable because, by exploiting the inductive argument, $t'$ and $t''$ are in normal form and contain guarded recursion only, hence (since the operators considered cannot turn visible actions into $\tau$ ones) every cycle in the derived transition system contains at least a visible action. Then we replace each subterm $t_i$ occurring in the equations for $t_{i+1} \ldots t_n$ with its equivalent term. When, in the equation for $t_n \equiv t$, we have replaced $t_{n-1}$, we are done.

- if $t \equiv t' \cdot t''$ then $t$ is turned into normal form similarly as in the previous item. The only difference is that $t''$ may include free variables. Supposing that $c(t'')$ denotes the closed term obtained from $t''$ by replacing each free occurrence of a variable $X$ by $a_X, \delta$, the procedure for obtaining the normal form from $t \equiv t' \cdot t''$ is the same followed for $t' \cdot c(t'')$ with the procedure of the previous item (note that when a state is reached such that any actions $a_X$ corresponding to free variables $X$ in $t''$ are immediately executable, the $\cdot$ operator has disappeared already).

- if $t \equiv \langle X \mid E \rangle$ then $t$ is turned into normal form by first exploiting the inductive argument over terms $\tau_Y$ where $Y \in V$, assuming $E = E(V)$, and then by applying axioms $Un$ and $WUn$ to get rid of generated unguarded recursion as in the standard approach of Milner (after decomposing multi-variable recursion with axiom $Dec$).

- if $t \equiv \tau_I(t')$ then $t$ is turned into normal form as follows. By exploiting the inductive argument over $t'$, we consider term $t''$ which is obtained by turning $t'$ into normal form. Observe that $t'$ (hence $t''$) cannot include free variables and that it has a finite transition system.

We first show, by structural induction on term $t''$, that $\tau_I(t'')$ can be turned into $\tau_I(t''')$, where $t'''$ is obtained from $t''$ by syntactically replacing each occurrence of an action in $I$ with $\tau$. To be precise, the induction works on (possibly open) terms $t'''$ of TCP+REC.

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

The inductive cases of the induction are the following ones:

- if $t'' \equiv a \cdot t''_1$ then we have the following two cases:
  * if $a \in I$ then, since $\tau_I(a \cdot t''_1)$ can be turned into $\tau \cdot \tau_I(t''_1)$, which by induction hypothesis can be turned into $\tau \cdot \tau_I(t'''_1)$, with $t'''_1$ such that each occurrence of an action in $I$ is replaced with $\tau$, we obtain term $t'''$ by the final transformation into $\tau_I(\tau \cdot t'''_1)$.
  * if $a \not\in I$ then it is a repetition of the previous case where $a$ is not turned into $\tau$.

- if $t'' \equiv \tau \cdot t''_1$ it is a repetition of the previous item where $\tau$ 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 $\tau$, we obtain term $t'''$ by the final transformation into $\tau_I(t'''_1 + t'''_2)$. 

14
- if \( t'' \equiv \langle X \mid \{ X = t'' \} \rangle \) then, since \( \tau_I(\langle X \mid \{ X = t'' \} \rangle) \) can be turned into \( \langle X \mid \{ X = \tau_I(t''') \} \rangle \) by means of axiom \textit{Hid}, which by induction hypothesis can be turned into \( \langle X \mid \{ X = \tau_I(t''') \} \rangle \), with \( t''' \) such that each occurrence of an action in \( I \) is replaced with \( \tau \), we obtain term \( t''' \) by the final transformation into \( \tau_I(\langle X \mid \{ X = t''' \} \rangle) \) by means again of axiom \textit{Hid}.

Then we use \textit{Ung} and \textit{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 \( \tau \) actions when the semantics is considered.

\[ \square \]

6 Conclusion

We just make some commentary about future work. First of all, we claim that the axiomatization that we presented is complete over all terms in the signature of TCP plus the recursion operator \( \langle X \mid E \rangle \) (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 left-hand-side of a sequence) provided that they are not reachable. Moreover, we plan to rebuild the whole machinery we showed here in the case of branching bisimulation instead of considering observational congruence. In particular we claim that we can find a ground-complete axiomatization for essentially finite state behaviours modulo branching bisimulation by taking the axiomatization of \cite{broadcast}, extending the syntax as we have done, and adding as only extra axiom our axiom (\textit{Hid}).

References


