Math. Struct. in Comp. Science (2008), *vol.* 18, *pp.* 1057–1089. © 2008 Cambridge University Press
doi:10.1017/S0960129508007111 First published online 18 September 2008 Printed in the United Kingdom

A ground-complete axiomatisation of finite-state processes in a generic process algebra

JOS C. M. BAETEN[†] and MARIO BRAVETTI[‡]

[†]*Division of Computer Science, Technische Universiteit Eindhoven, The Netherlands*
Email: josb@win.tue.nl
[‡]*Department of Computer Science, Università di Bologna, Italy*
Email: 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 that has been (and still is) working with them. In this paper we will first discuss these 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 of the basic mechanisms 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 specialised algebras: producing a complete axiomatisation for Milner's observational congruence in the presence of (unguarded) recursion and static operators. More precisely, we provide a syntactical characterisation (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. Conversely, we show that every process admits a specification in terms of such a restricted form of recursion. We then present an axiomatisation that 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 using a single axiom only.

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). Despite the conceptual similarities, these process algebras were developed from quite different starting viewpoints and gave rise to different approaches: CCS is heavily based on an observational bisimulation-based theory for communication over processes starting from an operational viewpoint; CSP was born as a theoretical version of a practical language for concurrency 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 originated from a

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

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 'ways of thinking' of the different communities that started working (and in many cases have continued working) with them.

In this paper we initially aim to clarify the differences between these approaches, which are often reflected in the usage of different terminology within the different communities, and to create the means for developing a unified view of process algebras. At first glance, it is easy to underestimate the impact of such differences. 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 becomes clear. As a first contribution of this paper, our study gives concrete form to 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, this algebra extends the algebra TCP (Baeten 2003; Baeten *et al.* 2008) (which extends ACP by including successful termination $\underline{1}$ and prefixing *à la* CCS) by the inclusion of a recursion operator $\langle X|E\rangle$ that computes the least transition relation satisfying a system of recursive equations (denoted $E = \{X = t_X, Y = t_Y, \ldots\}$) over processes, and considers an initial variable X among variables V defined by the system of equations E. This 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 of expressing recursion in ACP (where usually only guarded recursion is considered using systems of equations E). Note that, as 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 where 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 '$\partial_H(t)$', relabelling '$\rho_f(t)$' 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 that 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 (iii), we show how, by using TCP+REC, we can solve the problem of producing an axiomatisation that is complete over finite-state behaviours in the presence of unguarded recursion and static operators like parallel, hiding and restriction. Such an

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

axiomatisation and the related theorems are the second, and main, contribution of the paper.

The problem of developing a sound and complete axiomatisation 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 was 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 the 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

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

and two for weakly unguarded recursion

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

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

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, that is, it is shown that if two guarded terms are equivalent, they can be equated by the axiomatisation. This is done by exploiting the two axioms

$$recX.t = t\{recX.t/X\} \tag{Unfold}$$

$$t' = t\{t'/X\} \;\Rightarrow\; t' = recX.t \quad \text{if } X \text{ is guarded in } t \tag{Fold}$$

that express the existence and uniqueness of solutions in guarded recursion specifications. To be more precise, the axiomatisation obtained is shown to be complete for open terms, that is, 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, for example, considering parallel composition and restriction), the above axioms are no longer sufficient to get rid of unguarded recursion. In other words, even if two CCS terms are both finite state, it may be that they are not equated by an axiomatisation including the standard CCS axioms (the axioms for CCS without the $recX.t$ recursion operator) plus the above axioms for unguarded and guarded recursion. An example is

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

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 it cannot be equated by the axiomatisation to the equivalent term $recX.\tau.X$ or to $\tau.\underline{0}$. The problem is that this process produces unguarded recursion (a loop with only τ transitions in the transition system), so we cannot apply the folding axiom Fold. We should first remove the unguarded recursion, but the three axioms FUng, WUng1 and WUng2 only work with the restricted signature

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

(which does not include the parallel and restriction operators). As the main contribution of this paper, we show that by using TCP+REC and 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 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 the one 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*', that is, not in the scope of static operators like hiding, restriction, relabelling 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 our constraint (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 $((recX.a.X) \mid (recX.\bar{a}.X)) \setminus \{a\}$ considered earlier (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 proposed syntactical constraint is in some sense the weakest: if a (reachable) variable that is bound by an outer recursion operator occurs in the scope of static operators or in the left-hand side of a sequencing operator (and reachability is preserved by the static operators), then it produces an infinite-state transition system. We use $\mathrm{TCP{+}REC}_f$ to denote the process algebra that extends TCP with the recursion operator $\langle X|E\rangle$, where E satisfies the above constraint. Conversely, we show that in the context of the finite-state models under consideration, every process admits a specification in terms of $\mathrm{TCP{+}REC}_f$.

We, then produce, as a main result of the paper, an axiomatisation for TCP+REC that is *ground complete* over the signature of $\mathrm{TCP{+}REC}_f$: an equation can be derived from the axioms between closed terms exactly when the corresponding finite-state transition systems are observationally congruent. This axiomatisation is based on the introduction of the new axiom

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

which allows us to exchange the hiding operator (the only static operator that may generate unguarded recursion) with the recursion operator. This axiom is also considered in van Glabbeek (1997) without the seriality condition, which, however, is necessary to make it sound. We will show that with the inclusion of this 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, from outside to inside by 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 reverse direction) from inside to outside again. Assuming we are turning the term into

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

normal form (essentially basic CCS where recursion is guarded) by means of syntactical induction, once we have carried out the above procedure, 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 no longer has any effect. As a consequence, we can get rid of it using the Fold axiom in the same way as we do with any other static operator.

It is notable that in the axiomatisation we present we also make use of the following result, which we introduce here. Milner's two axioms for getting rid of weakly unguarded recursion (WUng1 and WUng2) can be expressed equivalently 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 note explicitly 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 (for example, for TCP) as an alternative to other techniques (such as the one in Bergstra and Klop (1985)).

The paper is structured as follows. In Section 2 we describe the different treatments of recursion and process variables in CCS, CSP and ACP. In Section 3 we present the model of processes that we consider: transition systems modulo observational congruence. In Section 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 Section 5 we present the proposed 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, conversely, every finite-state transition system can be expressed in terms of a TCP+REC$_f$ term) and give a formal argument supporting the claim that this syntactical constraint is the weakest that guarantees finite stateness. In Section 6 we present the axiomatisation and show that it is sound and ground complete for observational congruence over the TCP+REC$_f$ signature. Section 7 gives conclusions.

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

2. Process variables and recursion

The different viewpoint assumed in the ACP process algebra compared with, for example, the CCS process algebra gives rise to a different technical treatment of process variables in axiomatisations.

In CCS, axioms are considered as equations between terms, which can be expressed by using meta-variables P (as in, for example, $P + P = P$) standing for any term. The meaning is that the model generated by the term to the left of '=' is equivalent to the term to the right of '=' according to the notion of equivalence under consideration (for example, observational congruence for CCS). Terms to the left and 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, that is, terms where free variables X do not occur (or if they do occur, they are bound by, for example, a

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

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 with '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 in, for example, $x + x = x$). Note that here, unlike the case for CCS, the word process is used to denote any element in the model under consideration (for example, transition systems modulo branching bisimilarity). Such process variables act like meta-variables P in 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 process algebra considered. Equivalence over elements of the term model can then be assumed, for example, to be based on observational congruence like in CCS. In ACP, syntactical free variables like 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 (such as '$recX.P$' 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, that is, identities between terms that use process variables as in such axioms. This capability of deriving (open) equations from (open) equations is obtained by exploiting the axiom system derivation rules that allow us, for example, 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 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 in ACP of deriving an open equation corresponds to the capability in CCS of deriving an equation between two open terms, where syntactical free variables X are used instead of process variables. In the example above, the ability to derive $x + x + \underline{0} = x$ in ACP corresponds to the ability 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 use of the word 'calculus' to denote a process algebra. Unlike the case for CCS, in the ACP context the word calculus is only used if binding operators are introduced, and this is to emphasise the fact that in the presence of such operators we leave the purely algebraic domain. Finally, note that if in ACP we consider the model of labelled transition systems (or, optionally, the term model) modulo observational congruence, the notion of 'axiomatisation complete over closed terms' in the context of CCS corresponds to what in ACP is said to be *'ground complete'*: the axiomatisation is complete with respect to closed equations, that is, 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 'axiomatisation complete over open terms' in the context of CCS corresponds to what in ACP is said to be *'ground complete'*: the axiomatisation is complete with respect to (possibly) open equations, that is, identities between terms that (possibly) include process

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

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.

Now that we have explained these basic differences, 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, which are ranged over by X, Y. According to a terminology that is usual in the ACP setting (and which we also used 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, for example, 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

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

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

which actually make it possible to derive the solution, in ACP this property is expressed by using so-called 'principles'. The *Recursive Definition Principle*, which corresponds to the Unfold axiom, states that each recursive specification has a solution (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 using so-called '*constants*', which are ranged over by $A, B, \ldots$, or equivalently by the $recX.t$ operator, where t is a term containing variable X, from the set of solutions the solution will be that has the fewest transitions in the generated transition system chosen. 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

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

CCS such behaviour is expressed by the three axioms for unguarded recursion

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

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

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

which 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 Section 6). It is worth noting that if unguardedness is just caused 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: for example, 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 in ACP also, but in an indirect way, using 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), which was 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, unlike 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 $\bot$, a process that satisfies $x + \bot = \bot$ for all processes x (for an extension of ACP with such a process see Baeten and Bergstra (1997)).

3. Behaviours modulo observational congruence

Here 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 that 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 $\tau \notin A$.

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

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

The definition of weak bisimulation equivalence that we consider in the following is the usual extension of the standard one that is adopted when successful termination is distinguished from unsuccessful termination. Such a distinction is technically needed to have 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 labelled 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 $\alpha \in L$, we have either $\alpha = \tau$ and $(s',t) \in R$ or there are states t^*, t'', t' such that $t \Rightarrow t^* \xrightarrow{a} t'' \Rightarrow t'$ and $(s',t') \in R$.
2. Whenever $(s,t) \in R$ and $s \downarrow$, 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 $s \leftrightarrows_w t$, 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 $s \leftrightarrows_{rw} t$, if and only 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's 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), which was 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* $\gamma : A \times A \to A$. The function γ is partial, commutative and associative. The signature elements are as follows:

— 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.

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

— 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 distinguish between successful termination ($\underline{1}$) and unsuccessful termination ($\underline{0}$). Sequential composition is more general than action prefixing.
— Binary operator $\|$ denotes *parallel composition.*
In order to give a finite axiomatisation of parallel composition, there are two variations on this operator: the auxiliary operators $\lfloor\!\lfloor$ (left-merge) and $|$ (synchronisation merge). In the parallel composition $x \| y$, the separate components x and y may execute a step independently (denoted $x \lfloor\!\lfloor y$ and $y \lfloor\!\lfloor x$, respectively), or they may synchronise 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 *relabelling* for each $f : A \to A$.

In the following we will use:

— meta-variables x, y to range over processes of our process algebra, that is, transition-systems possibly denoted using 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 $+$, 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 $I = \varnothing$) to stand for $\underline{0}$.

We turn the set of closed terms (that is, terms containing no variables) over the signature of the algebra into a transition-system space by providing so-called operational rules – see Figure 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 axiomatisation can be provided that is *ground complete*, that is, 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 Figure 2.

4.2. *Theory of Communicating Processes with recursive specifications*

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

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

$$\underline{1}\downarrow \qquad \alpha.x \xrightarrow{\alpha} x$$

$$\frac{x \xrightarrow{\alpha} x'}{x+y \xrightarrow{\alpha} x'} \qquad \frac{y \xrightarrow{\alpha} y'}{x+y \xrightarrow{\alpha} y'} \qquad \frac{x\downarrow}{x+y\downarrow} \qquad \frac{y\downarrow}{x+y\downarrow}$$

$$\frac{x \xrightarrow{\alpha} x'}{x\cdot y \xrightarrow{\alpha} x'\cdot y} \qquad \frac{x\downarrow, y \xrightarrow{\alpha} y'}{x\cdot y \xrightarrow{\alpha} y'} \qquad \frac{x\downarrow, y\downarrow}{x\cdot y\downarrow}$$

$$\frac{x \xrightarrow{a} x', y \xrightarrow{b} y', \gamma(a,b)=c}{x \parallel y \xrightarrow{c} x' \parallel y'} \qquad \frac{x\downarrow, y\downarrow}{x\parallel y\downarrow} \qquad \frac{x \xrightarrow{\alpha} x'}{x\parallel y \xrightarrow{\alpha} x'\parallel y} \qquad \frac{y \xrightarrow{\alpha} y'}{x\parallel y \xrightarrow{\alpha} x\parallel y'}$$

$$\frac{x \xrightarrow{a} x', y \xrightarrow{b} y', \gamma(a,b)=c}{x \mid y \xrightarrow{c} x' \parallel y'} \qquad \frac{x\downarrow, y\downarrow}{x\mid y\downarrow} \qquad \frac{x \xrightarrow{\alpha} x'}{x\between y \xrightarrow{\alpha} x'\parallel y}$$

$$\frac{x \xrightarrow{\tau} x', x' \mid y \xrightarrow{\alpha} z}{x\mid y \xrightarrow{\alpha} z} \qquad \frac{y \xrightarrow{\tau} y', x \mid y' \xrightarrow{\alpha} z}{x\mid y \xrightarrow{\alpha} z} \qquad \frac{x \xrightarrow{\tau} x', x' \mid y\downarrow}{x\mid y\downarrow} \qquad \frac{y \xrightarrow{\tau} y', x \mid y'\downarrow}{x\mid y\downarrow}$$

$$\frac{x \xrightarrow{\alpha} x', \alpha \notin H}{\partial_H(x) \xrightarrow{\alpha} \partial_H(x')} \qquad \frac{x\downarrow}{\partial_H(x)\downarrow} \qquad \frac{x \xrightarrow{\alpha} x', \alpha \notin I}{\tau_I(x) \xrightarrow{\alpha} \tau_I(x')} \qquad \frac{x \xrightarrow{a} x', a \in I}{\tau_I(x) \xrightarrow{\tau} \tau_I(x')}$$

$$\frac{x\downarrow}{\tau_I(x)\downarrow} \qquad \frac{x \xrightarrow{a} x'}{\rho_f(x) \xrightarrow{f(a)} \rho_f(x')} \qquad \frac{x \xrightarrow{\tau} x'}{\rho_f(x) \xrightarrow{\tau} \rho_f(x')} \qquad \frac{x\downarrow}{\rho_f(x)\downarrow}$$

Fig. 1. Deduction rules for TCP.

Since as the model for our theory we are considering transition systems modulo observational congruence, the guardedness criterion for recursive specifications (which we discussed in Section 2) is as follows. Let t be a term containing a variable X. We say an occurrence of X in t is *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 say a recursive specification is *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 then dividing out one of the congruence relations of strong bisimilarity or observational congruence,

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

$x + y = y + x$	A1	$x \parallel y = x \between y + y \between x + x \mid y$	M
$(x + y) + z = x + (y + z)$	A2		
$x + x = x$	A3	$\underline{0} \between x = \underline{0}$	LM1
$(x + y) \cdot z = x \cdot z + y \cdot z$	A4	$\underline{1} \between x = \underline{0}$	LM2
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	A5	$\alpha.x \between y = \alpha.(x \parallel y)$	LM3
$x + \underline{0} = x$	A6	$(x + y) \between z = x \between z + y \between z$	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.

we have that 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 also add the possibility of including 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 is a variable in V that acts as the initial variable) that, similarly as in CCS, yields the least transition relation satisfying the recursive specification. 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, which we call TCP+REC.

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

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

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.

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

$$\frac{\langle t_X|E\rangle \xrightarrow{a} y}{\langle X|E\rangle \xrightarrow{a} y} \qquad \frac{\langle t_X|E\rangle \downarrow}{\langle X|E\rangle \downarrow}$$

Fig. 3. Deduction rules for recursion.

Note that terms t included in recursive specifications are again part of the same syntax, that is, they may again include recursive specifications. In the following we will use t_X to denote the term defining variable X (that is, $X = t_X$) 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 above setting, 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 that contains the occurrence of X in the right-hand term, but X may be bound by an outer recursive specification, as in, for example:

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

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

Figure 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 $\langle X|E\rangle$ as the process $\langle t_X|E\rangle$ given by the following definition.

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 $t\{\langle X|E\rangle/X \mid X \in V\}$, that is, 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, for example, in

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

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

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

yielding

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

Taken together, Figures 1 and 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.

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

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; 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 (Milner 1989a). This is done by omitting the signature elements $\underline{1}, \cdot, \between, |$. Next, we specialise the parameter set A by separating it into three parts: a set of names $\mathscr{A}$, a set of co-names $\bar{\mathscr{A}}$ and a set of communications $\mathscr{A}_c$ such that for each $a \in \mathscr{A}$ there is exactly one $\bar{a} \in \bar{\mathscr{A}}$ and exactly one $a_c \in \mathscr{A}_c$. The communication function γ is specialised so that the only defined communications are $\gamma(a, \bar{a}) = \gamma(\bar{a}, a) = a_c$, and then the CCS parallel composition operator $|_{CCS}$ can be defined by the formula

$$x |_{CCS} y \stackrel{def}{=} \tau_{\mathscr{A}_c}(x \between y).$$

We consider a subtheory corresponding to ACP_τ (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 $\underline{1}, ., \rho_f$.

We consider a subtheory corresponding to CSP (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 $\between_S$ is concerned, we specialise the parameter set A into two parts: a set of names $\mathscr{A}$ and a set of communications $\mathscr{A}_c$ such that for each $a \in \mathscr{A}$ there is exactly one $a_c \in \mathscr{A}_c$. The communication function γ is specialised so that the only defined communications are $\gamma(a, a) = a_c$, and, furthermore, we use the renaming function f that has $f(a_c) = a$. Then $x \between_S y$, where x and y are processes using names over $\mathscr{A}$ only and $S \subseteq \mathscr{A}$, can be defined by the formula

$$x \between_S y \stackrel{def}{=} \rho_f(\partial_{S \cup (\mathscr{A}_c - S_c)}(x \between y))$$

where we use 'S_c' to denote the set of names $\{a_c \mid a \in S\}$ and '$-$' to express set difference. Notice that just adopting the naive communication function $\gamma(a, a) = a$ would not work because, for example, if we try to translate $a.x \between_{\{a\}} a.y$ into $a.x \between a.y$, we can erroneously do independent a moves; if, instead, we consider $\partial_{\{a\}}(a.x \between a.y)$, the synchronisation on a is erroneously blocked. As far as the CSP *external choice* operator $\Box$ is concerned, we further specialise the set of names $\mathscr{A}$ into three parts: a set of names $\mathscr{B}$, and two sets of names $\mathscr{B}_1$ and $\mathscr{B}_2$ such that for each $a \in \mathscr{B}$ there is exactly one name $a_1 \in \mathscr{B}_1$ and one name $a_2 \in \mathscr{B}_2$. The communication function γ is not changed (no further communication is added). Finally, we use the renaming functions f' and f'' that have $f'(a_1) = a$ and $f''(a_2) = a$. Then $x \Box y$, where x and y are processes using names over $\mathscr{B}$ only, can be

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

defined by the formula

$$x \,\Box\, y \stackrel{def}{=} \rho_{f' \cup f''}((\rho_{f'^{-1}}(x) \parallel \rho_{f''^{-1}}(y)) \;\; \lfloor\!\lfloor_{\mathscr{B}_1 \cup \mathscr{B}_2} \;\; (\mathscr{B}_1{}^* \underline{1} + \mathscr{B}_2{}^* \underline{1}))$$

where, given a set of names B and a process x, 'B^*x' stands for

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

Note that the definition above does not work for versions of $\Box$ (like, for example, the one given 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 $\langle X|E\rangle$ that 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$_f$ obtained by extending the signature of TCP with essentially finite-state recursive specifications; that is, 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 $\parallel, \lfloor\!\lfloor, \mid, \partial_H, \tau_I, \rho_f$, 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 say E is *essentially finite state* if E has only finitely many equations and all variables are serial in the right-hand sides of all equations of E. We say E is *regular* if E has only finitely many equations and each equation is of the form

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

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), we can always introduce a fresh action r by extending with r the parameters A and γ of the theory under consideration (γ can,

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

possibly, be extended to include communication with r): for terms over the signature with the previous parameters A and γ (that is, where the new action r does not appear) we have unchanged transition systems/bisimilarities/equalities. In this paper we assume that γ remains 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. We begin by defining $c(t)$ 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:

— 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 are:

— 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 $t \equiv t' \between t''$ or $t \equiv t' \mid t''$ 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

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

where $ren(t')$ 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 a set of terms where such a renaming is applied to every term in the set). Once we have proved that the above statement holds, $c(t)$ is obviously finite state by an inductive argument over terms t_Y, for every $Y \in V$.

In the following we prove that the above inclusion does indeed hold. First 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 that 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, 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 $\parallel, \between, \mid, \partial_H, \tau_I, \rho_f$ or on the left-hand side of the operator $\cdot$, then there is

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

no $Y \in V$ such that $\langle Y|E\rangle$ is included in s' inside the scope of one of the operators $\|, \between, |, \partial_H, \tau_I, \rho_f$ or on the left-hand side of the operator $\cdot$.

— If for every $Y \in V$ the occurrence of $\langle Y|E''\rangle$ in s implies $E'' = E$, then for every $Y \in V$ the occurrence of $\langle Y|E''\rangle$ in s' implies $E'' = E$.

This can be proved easily by analysis of each operational rule by 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 true. $u \in S(\langle X|E\rangle)$ 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 $\|, \between, |, \partial_H, \tau_I, \rho_f$ or on the left-hand side of the operator $\cdot$.

— For every $Y \in V$, 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 if and only if $s \xrightarrow{\alpha} s'$ 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* $\langle Y|E\rangle$ if and only if $s \xrightarrow{\alpha} s'$ 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 characterise 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 $u \in S(\langle X|E\rangle)$, 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 $u = ren(c(t'))$.

Assuming 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 $u = ren(c(t'))$. This is because:

— s' is such that $ren(c(t_Y)) \xrightarrow{\alpha} s'$ and $ren(c(t_Y)) \xrightarrow{\alpha} s'$ can be inferred without E. This follows from the fact that $s \xrightarrow{\alpha} s'$ can be inferred by using $\langle Y|E\rangle$ and for the properties above that characterise states in $S(\langle X|E\rangle)$: that is, $\langle Y|E\rangle$ 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 $var(ren(c(t_1)) \xrightarrow{\alpha'} v')$ can be inferred without E, there exists t_1' such that $v' = ren(c(t_1'))$ and $c(t_1) \xrightarrow{\alpha'} c(t_1')$. This follows by induction on the height of the inference tree of transitions $var(ren(c(t_1)) \xrightarrow{\alpha'} v')$ 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 generated infinitely many states, it can still be the case that there are only finitely many states modulo bisimilarity. For instance, for the recursive equation $X = \tau_{\{a\}}(a.X)$, a new abstraction operator is generated at each iteration, but all the generated terms are

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

bisimilar since the abstraction operator is idempotent. Of course, in other cases, such as for $X = (a.\underline{1}) + (b.X \cdot X)$, we do obtain infinitely many terms that are not bisimilar. However, it should be noted that the axioms we will present in Section 6 remain valid even if recursive specifications that are not essentially finite state are considered (thereby 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 unnecessarily disregard terms that generate finite-state transition systems, but first we need to introduce some machinery related to the representation of contexts, together with 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 $t^n_{\dot{X}}$, with $n \geqslant 0$ to stand for the term inductively defined as follows:

$$t^0_{\dot{X}} \equiv X$$
$$t^n_{\dot{X}} \equiv t_{\dot{X}}\left(t^{n-1}_{\dot{X}}\right) \text{ for } n > 0.$$

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, that is, 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, that is, $w.t \equiv \alpha.t$ if $w = \alpha$, and $w.t \equiv \alpha.(w'.t)$ if $w = \alpha w'$. Finally, we implicitly assume that when we introduce some fresh action r, the renaming functions f occurring inside terms under consideration leave r unchanged, that is, 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, if and only if, considering 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 occurence 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 $t_{\dot{X}}(v \cdot \underline{0}) \xrightarrow{\alpha} t'$ for some $\alpha \in A \cup \{\tau\}$ and closed term t', then at least one of the following conditions is true:

1. There exist a closed context $t'_{\dot{X}}$ and a closed term v' that is reachable from v *via* a path labelled by some sequence of actions w, with $t' \equiv t'_{\dot{X}}(v' \cdot \underline{0})$ and $t'_{\dot{X}}$ is an α-derivative of $t_{\dot{X}}$ for w, such that:

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

(a) $t'_{\dot{X}}$ is static and:
   — if $t_{\dot{X}}$ has X in the scope of one of the operators $\|, \lfloor\!\lfloor, |, \partial_H, \tau_I, \rho_f$ 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' labelled 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 $t' \equiv t'_{\dot{X}}(v \cdot \underline{0})$ and, considering a fresh action r, we have $t_{\dot{X}}(r.\underline{0}) \xrightarrow{\alpha} t'_{\dot{X}}(r.\underline{0})$. 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, considering a fresh action r, we have $t_{\dot{X}}(r.\underline{0}) \xrightarrow{\alpha} t'$.

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

Proof. The proof is by induction on the height of the inference tree by which transitions $t_{\dot{X}}(v \cdot \underline{0}) \xrightarrow{\alpha} t'$ of $t_{\dot{X}}(v \cdot \underline{0})$ 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 $s_{\dot{X}}$ – the lemma is obvious in these cases (Conditions 1 and 2, respectively, obviously hold).

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

We will 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) as this is the most intricate one – the proof for the other operators 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, in the case $t_{\dot{X}} \equiv s_{\dot{X}} \cdot u$, the statement about the termination capability of $t_{\dot{X}}(v \cdot \underline{0})$ in the lemma is needed to derive the fact 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, $s_{\dot{X}}(v \cdot \underline{0}) \xrightarrow{\tau} s''$ must satisfy one of the conditions of the lemma, thus we have three cases, which are numbered according to the condition satisfied.

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

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

1.1. There are $t'_{\dot{X}}$ and 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'' labelling 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 $w = w'w''$).

1.2. 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 $v' \equiv v''$).

1.3. 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, considering a fresh action r, we have $s_{\dot{X}}(r.\underline{0}) \xrightarrow{\tau} s''_{\dot{X}}(r.\underline{0})$.
By induction, $s''_{\dot{X}}(v \cdot \underline{0}) \mid u \xrightarrow{\alpha} t'$ must satisfy one of the conditions of the lemma, thus we have three subcases, which are numbered according to the condition satisfied.

2.1. There are $t'_{\dot{X}}$ and 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 labelling 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.

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

2.3. Considering a fresh action r, we have $s''_{\dot{X}}(r.\underline{0}) \mid u \xrightarrow{\alpha} t'$.
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. Considering a fresh action r, we have $s_{\dot{X}}(r.\underline{0}) \xrightarrow{\tau} s''$.
We can 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, which correspond to the operational rules for the $\mid$ operator that yield a terminating term, and are completely analogous to the three cases considered above for transitions: we only need to consider termination $\downarrow$ rather than a,b or α outgoing transitions. In particular, for the first case considered

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

above, we have the same three cases corresponding to the condition that is satisfied: in the case of Conditions 1 and 2, we follow a similar argument to that in subcase 3 (thus Condition 1 cannot be obtained). □

From the above lemma we have the following direct consequence. 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, we have that (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 $w.r'.\underline{0}$ and $t_{\dot{X}}$ 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 $\langle Y|E\rangle$, for some Y and E, and t' is obtained from t by replacing it with $\langle t_Y|E\rangle$. 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 be a closed term. We say that $t'_{\dot{X}}$ *is an unfolding of* $t_{\dot{X}}$ *with respect to* 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 $t_{\dot{X}}$, there exists α' (possibly τ) and a 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, that is, some occurrence of some variable $Y \in V(E)$ violates the seriality condition. Then t has infinitely many states if:

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 $\|, \between, |, \partial_H, \tau_I, \rho_f$ 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 $t_{\dot{X}}$ with respect to $\langle Y|E'\rangle$ 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 $n \geqslant 1$, $S \cup \{\alpha\}$ is not restricted by the static context $free(t'^n_{\dot{X}})$.

Proof. We show by induction on n that for every $n \geqslant 1$ there is a path from t to a state $free(t'^n_{\dot{X}})(t')\theta$ for some substitution θ of its free variables. Note that by Lemma 5.7, $t'_{\dot{X}}$ is

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

$\langle X\|E \,\tilde{\cup}\, \{Y = t\}\rangle = \langle\langle X\|E\rangle\|Y = t\rangle$ if $X \neq Y$	Dec
$\langle X\|X = t\rangle = \langle t\|X = t\rangle$	Unf
$s = t\{s/X\} \quad \Rightarrow \quad s = \langle X\|X = t\rangle$ 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.

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 have directly from Assumption 1 above that 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, that is, we assume that there is a path from t to a state $free(t'^{n-1}_{\dot{X}})(t')\theta$ for some substitution θ of its free variables. We have:

— from 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 Assumption 1 above, there is a transition α from $t_{\dot{X}}(\langle Y\,|E'\rangle)$ to $t'_{\dot{X}}(t')$, and

— from Assumption 3 above, $S \cup \{\alpha\}$ is not restricted by the static context $free(t'^{n-1}_{\dot{X}})$.

So, by Lemma 5.7 (a) and (b), there is a path from $free(t'^{n-1}_{\dot{X}})(t')\theta$ to $free(t'^{n-1}_{\dot{X}})(t'_{\dot{X}}(t'))\theta'$ for some substitution θ'. We therefore conclude that there exists a substitution θ'' such that there is a path from t to a state $free(t'^{n}_{\dot{X}})(t')\theta''$. □

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

6. An axiomatisation that is complete for finite behaviours

We will now present a sound axiomatisation that is ground complete for the process algebra TCP+REC$_f$. The axioms in Figure 2 together with the axioms in Figure 4 form such an axiomatisation. In the axioms of Figure 4, the symbol $\tilde{\cup}$ stands for disjoint union. Note that the axioms in Figure 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

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

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

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

The unfolding axiom Unf is Milner's standard one (corresponding to the *Recursive Definition Principle* in ACP): it states that the constant $\langle X|E\rangle$ is a solution of the recursive specification E. Thus, each recursive specification has a solution. The folding axiom (Fold) is also 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 $y = \langle X|E\rangle$.

Axioms Ung, WUng and Hid are used to deal with unguarded specifications. Ung, which is the same as in Milner's axiomatisation, is the axiom that deals with variables not in the scope of any prefix operator (fully unguarded recursion). On the other hand, WUng and Hid are needed to get rid of weakly unguarded recursion.

WUng gets rid of weakly unguarded recursion arising purely from prefixing and summation. It is easy to see that it replaces the two axioms of Milner:

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

The first is obtained from WUng by taking $t = \underline{0}$. The second 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 $t = \underline{0}$.

As explained in the introduction, the axiom Hid is used to get rid of weak unguardedness generated by the hiding operator. It allows us to turn a term into a form for which the standard axioms for weak unguardedness can be used (see the proof of Proposition 6.5 below). 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 relabelling 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 relabelling 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.

Note that if we want to derive a ground-complete axiomatisation in a setting where no construct is added for recursion, as is usually done in the context of the ACP process algebra (so we just have closed terms over the syntax of TCP and just consider sets of recursion equations over this syntax), then in order to achieve the unguardedness removal effect that we obtain here 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), which we have already mentioned in Section 2.

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

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

Proof. Most of the axioms are standard. We provide a full proof for the new axiom

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

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''

$$\tau_I(\langle s|X = \tau_I(t)\rangle) \xrightarrow{\alpha} u'', \quad u'' \leftrightarrows_w u' \quad \text{and} \quad (u, u') \in \beta.$$

— 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 the termination capabilities of $\tau_I(\langle s|X = \tau_I(t)\rangle)$.

This implies that β is a weak bisimulation up to $\leftrightarrows_w$ (see the revised version of Milner (1989a) as corrected by Sangiorgi and Milner (1992)), hence $\beta \subseteq \leftrightarrows_w$. From this result it follows that $\tau_I(\langle X|X = t\rangle) \leftrightarrows_{rw} \langle X|X = \tau_I(t)\rangle$ 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 (distinguished by the form of s) are:

— If $s \equiv \underline{0}$ or $s \equiv \underline{1}$, the above conditions hold trivially.

— If $s \equiv \alpha.s'$, then $\tau_I(\langle s|X = t\rangle) \equiv \tau_I(\alpha.(\langle s'|X = t\rangle))$ and $\tau_I(\langle s|X = \tau_I(t)\rangle) \equiv \tau_I(\alpha.(\langle s'|X = \tau_I(t)\rangle))$.
We have the following two cases for transitions α:

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

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

For the induction step we have the following cases based on the form of s:

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

— 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 $\tau_I(\langle X|X = t\rangle) \xrightarrow{\alpha} u$, we must have $\alpha \notin I$ 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, we must also have $\langle t|X = t\rangle \xrightarrow{\alpha'} v$ 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 $u'' \leftrightarrow_w u'$ and $(u, u') \in \beta$. As a consequence, $\langle X|X = \tau_I(t)\rangle \xrightarrow{\alpha} u''$ and, since $\alpha \notin I$, we have $\tau_I(\langle X|X = \tau_I(t)\rangle) \xrightarrow{\alpha} \tau_I(u'')$. Since u'' has hiding as the outermost operator (because it 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).
For the termination capability, we have, by performing the same steps as for the case of transitions, that $\tau_I(\langle X|X = t\rangle)\downarrow$ 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 $\tau_I(\langle s'|X = t\rangle + \langle s''|X = t\rangle) \xrightarrow{\alpha} u$, we must have $\alpha \notin I$ 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 $u'' \leftrightarrow_w u'$ and $(u, u') \in \beta$. Therefore, we must have $\langle s'|X = \tau_I(t)\rangle \xrightarrow{\alpha''} v''$ with $u'' \equiv \tau_I(v'')$, and $\alpha'' = \alpha$ if $\alpha \neq \tau$ and $\alpha'' \in I$ 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 $\langle s''|X = t\rangle \xrightarrow{\alpha'} v$, the result is derived in a similar way.

For the 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 $\tau_I(\langle Y|Y = \langle s'|X = t\rangle\rangle) \xrightarrow{\alpha} u$, we must have $\alpha \notin I$ 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, we must have $\langle\langle s'|X = t\rangle|Y = \langle s'|X = t\rangle\rangle \xrightarrow{\alpha'} v$. 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 $u'' \leftrightarrow_w u'$ and $(u, u') \in \beta$. Therefore, we must have $\langle\langle s'|Y = s'\rangle|X = \tau_I(t)\rangle \equiv \langle\langle s'|X = \tau_I(t)\rangle|Y = \langle s'|X = \tau_I(t)\rangle\rangle \xrightarrow{\alpha''} v''$ 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, $\tau_I(\langle Y|Y = \langle s'|X = \tau_I(t)\rangle\rangle) \xrightarrow{\alpha} u''$.
For the 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 $\tau_I(s' \cdot \langle s''|X = t\rangle) \xrightarrow{\alpha} u$, we must have $\alpha \notin I$ 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 $v = z \cdot \langle s''|X = t\rangle$, 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 $\tau_I(s' \cdot \langle s''|X = \tau_I(t)\rangle) \xrightarrow{\alpha} \tau_I(z \cdot \langle s''|X = \tau_I(t)\rangle)$

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

– 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 $u'' \leftrightarrows_w u'$ and $(u, u') \in \beta$. Therefore, we must have $\langle s''|X = \tau_I(t)\rangle \xrightarrow{\alpha''} v''$ 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, $\tau_I(\langle s''|X = \tau_I(t)\rangle) \xrightarrow{\alpha} u''$.

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

— If $s \equiv s' \parallel s''$ or $s \equiv s' \lfloor\!\lfloor s''$ or $s \equiv s' \mid s''$ or $s \equiv \partial_H(s')$ or $s \equiv \tau_I(s')$ or $s \equiv \rho_f(s')$, 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 the termination capability of $\tau_I(\langle s|X = \tau_I(t)\rangle)$ in the conditions above. □

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

Before we finally present the completeness result, we first 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 axiomatisation in Figures 2 and 4 into the form

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

where $\xrightarrow{\alpha_i} t_i$, with $1 \leqslant i \leqslant n$, 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 Figures 1 and 3.

Proof. We show that any closed normal form t can be turned by the axiomatisation into the form $\sum_{1 \leqslant i \leqslant n} \alpha_i.t_i + \{\underline{1}\}$ with the above properties using 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 already in the desired form.

The inductive cases are:

— If $t \equiv t' + t''$, then t can be turned into the desired form by just summing the terms obtained by applying the inductive argument to t' and t'' (t is equated by the axiomatisation 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 into the desired form by direct consideration of the term obtained by applying the inductive argument to $\langle t'|\{X = t'\}\rangle$ (t is equated by the axiomatisation to such a term by means of the unfolding axiom Unf, and the

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

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 $t \equiv t' \between t''$ or $t \equiv t' \mid t''$ or $t \equiv \partial_H(t')$ or $t \equiv \tau_I(t')$ or $t \equiv \rho_f(t')$ can be turned by the axiomatisation in Figures 2 and 4 into the form

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

where $\stackrel{\alpha_i}{\rightarrow} t_i$, with $1 \leqslant i \leqslant n$, 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 Figures 1 and 3.

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

We first consider the case of sequence, that is, $t \equiv t' \cdot t''$. We initially have

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

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

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

One can see immediately that, since $\stackrel{\alpha'_i}{\rightarrow} t'_i$, with $i \leqslant n$, and $\stackrel{\alpha''_i}{\rightarrow} t''_i$, with $i \leqslant m$, are the outgoing transitions of t' and t'', respectively, and as $\underline{1}$ is 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, that is, $t \equiv t' \between t''$, restriction, that is, $t \equiv \partial_H(t')$, hiding, that is, $t \equiv \tau_I(t')$, and relabelling, that is, $t \equiv \rho_f(t')$, are proved similarly by performing the following transformations:

$$t' \between t'' = t'_{next} \between t'' = \sum_{i \leqslant n} (\alpha'_i.t'_i \between t'') = \sum_{i \leqslant n} \alpha'_i.(t'_i \parallel t'')$$

$$\partial_H(t') = \partial_H(t'_{next}) = \sum_{i \leqslant n} \partial_H(\alpha'_i.t'_i) + \{\underline{1}\} = \sum_{i \leqslant n, \alpha'_i \notin H} \alpha'_i.\partial_H(t'_i) + \{\underline{1}\}$$

$$\tau_I(t') = \tau_I(t'_{next}) = \sum_{i \leqslant n} \tau_I(\alpha'_i.t'_i) + \{\underline{1}\} = \sum_{i \leqslant n, \alpha'_i \in I} \tau.\tau_I(t'_i) + \sum_{i \leqslant n, \alpha'_i \notin I} \alpha'_i.\tau_I(t'_i) + \{\underline{1}\}$$

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

We now consider the case of synchronisation merge, that is, $t \equiv t' \mid t''$. We initially have

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

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

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

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

We show that we can turn any $t \equiv t' \mid t''$ into $\sum_{1 \leqslant i \leqslant n} \alpha_i.t_i + \{\underline{1}\}$ 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 into 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, that is, neither t' nor t'' can perform τ transitions. This means that when transforming t in the sum form above, the first two sums do not occur, and thus the assertion obviously holds.

— The inductive case is performed by just observing that the summands $\tau.t'_i \mid t''$ and $t' \mid \tau.t''_j$ obtained by transforming t into the sum form above can be rewritten into $t'_i \mid t''$ and $t' \mid t''_j$, respectively. For such terms we can apply the induction hypothesis and turn them into the form $\sum_{1 \leqslant i \leqslant n} \alpha'''_i.t'''_i + \{\underline{1}\}$ such that the arguments of the sum correspond to their transitions/termination capability. Therefore, since $\stackrel{\alpha'_i}{\rightarrow} t'_i$, with $i \leqslant n$, and $\stackrel{\alpha''_i}{\rightarrow} t''_i$, with $i \leqslant m$, are the outgoing transitions of t' and t'', respectively, and 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 synchronisation merge, $t' \mid t''$ is (additionally) endowed with the transitions/termination capability of $t'_i \mid t''$ ($t' \mid t''_j$) whenever $t' \stackrel{\tau}{\rightarrow} t'_i$ ($t'' \stackrel{\tau}{\rightarrow} t''_i$), the arguments of the sum obtained by turning such terms into $\sum_{1 \leqslant i \leqslant n} \alpha'''_i.t'''_i + \{\underline{1}\}$ inside the sum form above correspond to the transitions/termination capability of $t' \mid t''$.

We now consider the case of the parallel operator, that is, $t \equiv t' \parallel t''$. We initially have

$$t' \parallel t'' = t'_{next} \mathbin{\lfloor\!\lfloor} t'' + t''_{next} \mathbin{\lfloor\!\lfloor} t' + t'_{next} \mid t''_{next}.$$

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

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

the form $\tau.((t'_i \mid t'') + t''')$ and $\tau.((t' \mid t''_j) + t''')$, respectively, we derive

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

where in the second sum we have also exploited the commutativity of '$\parallel$'. One can then see immediately that since $\stackrel{\alpha'_i}{\rightarrow} t'_i$, with $i \leqslant n$, and $\stackrel{\alpha''_i}{\rightarrow} t''_i$, with $i \leqslant m$, are the outgoing transitions of t' and t'', respectively, and as $\underline{1}$ is 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. □

Proposition 6.5. The axiomatisation formed by the axioms in Figure 2 and by the axioms in Figure 4 is ground complete for TCP+REC$_f$ and the model of transition systems modulo observational congruence generated by the rules in Figures 1 and 3.

Proof. We show by structural induction over the syntax of (possibly open) terms t of TCP+REC$_f$ whose free variables do not occur in the scope of one of the operators $\parallel, \lfloor\!\lfloor, \mid, \partial_H, \tau_I, \rho_f$ 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 is because normal forms are like terms of basic CCS (the only difference being that we have two non-equivalent kinds of terminating processes $\underline{0}$ and $\underline{1}$ instead of just one) and completeness over such terms was 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:

— If $t \equiv a.t'$ or $t \equiv \tau.t'$ or $t \equiv t' + t''$, then t can be turned into normal form by directly exploiting the inductive argument over t' and t''.

— If $t \equiv t' \parallel t''$ or $t \equiv t' \lfloor\!\lfloor t''$ or $t \equiv t' \mid t''$ or $t \equiv \partial_H(t')$ or $t \equiv \rho_f(t')$, 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 $t_1 \ldots t_n$ be the states of the transition system of t''', $t_n \equiv t'''$. It is easy to see that, due to Lemma 6.4, for each $i \in \{1 \ldots n\}$, there exist m_i, $\{\alpha^i_j\}_{j \leqslant m_i}$ (denoting actions) and $\{k^i_j\}_{j \leqslant m_i}$ (denoting natural numbers) such that we can derive $t_i = \sum_{j \leqslant m_i} \alpha^i_j.t_{k^i_j} + \{\underline{1}\}$. Hence we can characterise the behaviour of t''' by means of a set of equations ($t_1 \ldots t_n$ are the solution of a regular recursive specification with n variables). We can, therefore, turn t''' into normal form in a similar way to what is 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 $\exists j \leqslant m_i : k^i_j = i$, applying Fold gives us $t_i = \langle X | X = \sum_{j \leqslant m_i : k^i_j \neq i} \alpha^i_j.t_{k^i_j} + \sum_{j \leqslant m_i : k^i_j = i} \alpha^i_j.X + \{\underline{1}\}\rangle$. Note that axiom Fold is applicable because, by exploiting the inductive argument, t' and t'' have been turned into normal form and contain guarded recursion only, so (since

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

the operators considered cannot turn visible actions into τ ones) every cycle in the derived transition system contains at least one visible action, that is, according to the definition in Milner (1989b), the equation set under consideration 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 in a similar way to 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 forms, which are obtained by exploiting the inductive argument. We will use $c(t''')$ to denote the closed term obtained from t''' by replacing each free occurrence of a variable X by $a_X.\underline{0}$, where a_X is a fresh action. Note that free variables may only occur in t''' inside the normal form of t'', that is, in the subterm to the right of '$\cdot$'. We know that $c(t''')$ has a finite transition system, hence $c(t''')$ can be turned into normal form using the procedure of the previous item: by considering its states, by transforming them into a sum of prefixes leading to other states using Lemma 6.4 (note that in this case only states containing the '$\cdot$' 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 $c(t''')$ (which 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 $t_i = \sum_{j \leqslant m_i} \alpha_j^i.t_{k_j^i} + \sum_{j \leqslant m_i} X_j^i + \{\underline{1}\}$ is obtained by following exactly the same derivation procedure as described in Lemma 6.4, which allows us to derive $c(t_i) = \sum_{j \leqslant m_i} \alpha_j^i.c(t_{k_j^i}) + \sum_{j \leqslant m_i} a_{X_j^i}.\underline{0} + \{\underline{1}\}$ correspondingly. It is possible to follow the same derivation procedure when X_j^i variables replace $a_{X_j^i}.\underline{0}$ prefixes because variables X_j^i cannot occur inside t_i in the subterm to the left of '$\cdot$' (which is a closed subterm), hence in the derivation of Lemma 6.4, the axiom $A10$ (which is used to move prefixes from inside to outside of a '$\cdot$' operator, representing their execution) is never applied to an $a_{X_j^i}$ prefix. One can see immediately that all the other axioms used in the derivation are still applicable when X_j^i variables replace $a_{X_j^i}.\underline{0}$ 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 any 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'', we can turn $\tau_I(t'')$ into $\tau_I(t''')$, where t''' is obtained from t'' by syntactically replacing each occurrence of an action in I with τ.

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

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 are:

- If $t'' \equiv a.t_1''$, 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 the 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 $a \notin I$, we have a repeat of the previous case where a is not turned into τ.
- If $t'' \equiv \tau.t_1''$, we have a repeat 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 the 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 the 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 any generated unguarded recursion into t''' as in Milner's standard approach, thus getting a guarded t''''.

Finally, we consider $\tau_I(t'''')$ and apply the same technique as for, for example, the $\|$ operator to turn it into normal form (exploiting the fact that t'''' is guarded, finite state and does not include free variables). In particular, we can now do this because the application of the hiding operator has no effect on the labels of transitions, and hence it cannot generate cycles made up purely of τ actions when the semantics is considered. □

7. Conclusion

In conclusion, we will just make some comments about future work. First, we claim that the axiomatisation we have presented is complete over all terms in the signature of TCP plus the recursion operator $\langle X|E\rangle$ (without syntactical restriction) that are finite state, that is, we can also include terms with variables bound by an outer recursion operator that are in the scope of static operators (or on the left-hand side of a sequence) provided they are not reachable. Moreover, we plan to rebuild the whole machinery developed here for the case of rooted branching bisimilarity instead of observational congruence. In particular, we claim that we can find a ground-complete axiomatisation for essentially finite-state behaviours modulo branching bisimilarity by taking the axiomatisation of van

https://doi.org/10.1017/S0960129508007111 Published online by Cambridge University Press

Glabbeek (1993) extending the syntax as we have done, and adding our axiom Hid as the only extra axiom.

8. Acknowledgements

We would like to thank Rob van Glabbeek and the anonymous reviewers for their useful remarks and suggestions. The replacement of Milner's two axioms for weakly unguarded recursion by a single axiom was also found independently by Rob van Glabbeek, but never published.

References

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

Baeten, J. C. M., Basten, T. and Reniers, M. A. (2008) *Process algebra (equational theories of communicating processes)*, Cambridge Tracts in Theoretical Computer Science, Cambridge University Press.

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

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

Baeten, J. C. M. and Bravetti, M. (2005) A ground-complete axiomatisation of finite state processes in process algebra. In: Abadi, M. and de Alfaro, L. (eds.) Proc. of the 16th International Conference on Concurrency Theory (CONCUR 2005). *Springer-Verlag Lecture Notes in Computer Science* **3653** 248–262.

Baeten, J. C. M. and Bravetti, M. (2006) A generic process algebra. In: Aceto, L. and Gordon, A. D. (eds.) Proc. of the Workshop Essays on Algebraic Process Calculi (APC 25). *Electr. Notes Theor. Comput. Sci.* **162** 65–71.

Baeten, J. C. M. and van Glabbeek, R. J. (1987) Merge and termination in process algebra. In: Nori, K. V. (ed.) Proc. 7th Conf. on Foundations of Software Technology and Theoretical Computer Science, Pune, India. *Springer-Verlag Lecture Notes in Computer Science* **287** 153–172.

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

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

Bergstra, J. A. and Klop, J. W. (1986) Verification of an alternating bit protocol by means of process algebra. In: Bibel, W. and Jantke, K. P. (eds.) Proc. Mathematical Methods of Specification and Synthesis of Software Systems. *Springer-Verlag Lecture Notes in Computer Science* **215** 9–23.

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

Bravetti, M. and Gorrieri, R. (2002) Deciding and axiomatizing weak ST bisimulation for a process algebra with recursion and action refinement. *ACM Transactions on Computational Logic* **3** (4) 465–520.

Brookes, S. D., Hoare, C. A. R. and Roscoe, A. W. (1984) A theory of communicating sequential processes. *Journal of the ACM* **31** (3) 560–599.

van Glabbeek, R.J. (1987) Bounded nondeterminism and the approximation induction principle in process algebra. In: Brandenburg, F.J., Vidal-Naquet, G. and Wirsing, M. (eds.) Proceedings STACS'87. *Springer-Verlag Lecture Notes in Computer Science* **247** 336–347.

van Glabbeek, R.J. (1993) A complete axiomatization for branching bisimulation congruence of finite-state behaviours. In: Borzyszkowski, A.M. and Sokolowski, S. (eds.) Proc. MFCS'93. *Springer-Verlag Lecture Notes in Computer Science* **711** 473–484.

van Glabbeek, R.J. (1994) On the expressiveness of ACP (extended abstract). In: Ponse, A., Verhoef, C. and van Vlijmen, S.F.M. (eds.) *Proceedings First Workshop on the Algebra of Communicating Processes, ACP94*, Workshops in Computing, Springer-Verlag 188–217. (Available at `http://boole.stanford.edu/pub/acp.ps.gz`.)

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

van Glabbeek, R.J. and Weijland, W.P. (1996) Branching time and abstraction in bisimulation semantics. *Journal of the ACM* **43** (3) 555–600.

Hoare, C.A.R. (1985) *Communicating Sequential Processes*, Prentice Hall.

Milner, R. (1989a) *Communication and Concurrency*, Prentice Hall.

Milner, R. (1989b) A complete axiomatization for observational congruence of finite-state behaviours. *Information and Computation* **81** 227–247.

Sangiorgi, D. and Milner, R. (1992) The problem of 'weak bisimulation up to'. In: Cleaveland, R. (ed.) Proceedings CONCUR'92. *Springer-Verlag Lecture Notes in Computer Science* **630** 32–46.

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