Axiomatizing GSOS with Predicates[*]

Luca Aceto Georgiana Caltais Eugen-Ioan Goriac Anna Ingolfsdottir
[luca,gcaltais10,egoriac10,annai]@ru.is
ICE-TCS, School of Computer Science, Reykjavik University, Iceland

In this paper, we introduce an extension of the GSOS rule format with predicates such as termination, convergence and divergence. For this format we generalize the technique proposed by Aceto, Bloom and Vaandrager for the automatic generation of ground-complete axiomatizations of bisimilarity over GSOS systems. Our procedure is implemented in a tool that receives SOS specifications as input and derives the corresponding axiomatizations automatically. This paves the way to checking strong bisimilarity over process terms by means of theorem-proving techniques.

1 Introduction

One of the greatest challenges in computer science is the development of rigorous methods for the specification and verification of reactive systems, *i.e.*, systems that compute by interacting with their environment. Typical examples include embedded systems, control programs and distributed communication protocols. Over the last three decades, process algebras, such as ACP [4], CCS [16] and CSP [14], have been successfully used as common languages for the description of both actual systems and their specifications. In this context, verifying whether the implementation of a reactive system complies to its specification reduces to proving that the corresponding process terms are related by some notion of behavioural equivalence or preorder [13].

One approach to proving equivalence between two terms is to exploit the equational style of reasoning supported by process algebras. In this approach, one obtains a (ground-)complete axiomatization of the behavioural relation of interest and uses it to prove the equivalence between the terms describing the specification and the implementation by means of equational reasoning, possibly in conjunction with proof rules to handle recursively-defined process specifications.

Finding a "finitely specified", (ground-)complete axiomatization of a behavioural equivalence over a process algebra is often a highly non-trivial task. However, as shown in [2] in the setting of bisimilarity [16, 17], this process can be automated for process languages with an operational semantics given in terms of rules in the GSOS format of Bloom, Istrail and Meyer [8]. In that reference, Aceto, Bloom and Vaandrager provided an algorithm that, given a GSOS language as input, produces as output a "conservative extension" of the original language with auxiliary operators together with a finite axiom system that is sound and ground-complete with respect to bisimilarity (see, *e.g.*, [1, 12, 15, 18] for further results in this line of research). As the operational specification of several operators often requires a clear distinction between successful termination and deadlock, an extension of the above-mentioned approach to the setting of GSOS with a predicate for termination was proposed in [6].

[*]The authors have been been partially supported by the projects "New Developments in Operational Semantics" (nr. 080039021), "Meta-theory of Algebraic Process Theories" (nr. 100014021), and "Extending and Axiomatizing Structural Operational Semantics: Theory and Tools" (nr. 110294-0061) of the Icelandic Research Fund. The work on the paper was partly carried out while Luca Aceto and Anna Ingolfsdottir held an Abel Extraordinary Chair at Universidad Complutense de Madrid, Spain, supported by the NILS Mobility Project.

In this paper we contribute to the line of the work in [2] and [6]. Inspired by [6], we introduce the *preg* rule format, a natural extension of the GSOS format with an arbitrary collection of predicates such as termination, convergence and divergence. We further adapt the theory in [2] to this setting and give a procedure for obtaining ground-complete axiomatizations for bisimilarity over *preg* systems. More specifically, we develop a general procedure that, given a *preg* language as input, automatically synthesizes a conservative extension of that language and a finite axiom system that, in conjunction with an infinitary proof rule, yields a sound and ground-complete axiomatization of bisimilarity over the extended language. The work we present in this paper is based on the one reported in [2, 6]. However, handling more general predicates than immediate termination requires the introduction of some novel technical ideas. In particular, the problem of axiomatizing bisimilarity over a *preg* language is reduced to that of axiomatizing that relation over finite trees whose nodes may be labelled with predicates. In order to do so, one needs to take special care in axiomatizing negative premises in rules that may have positive and negative premises involving predicates and transitions.

The results of the current paper have been used for the implementation of a Maude [10] tool [3] that enables the user to specify *preg* systems in a uniform fashion, and that automatically derives the associated axiomatizations. The tool is available at `http://goriac.info/tools/preg-axiomatizer/`. This paves the way to checking bisimilarity over process terms by means of theorem-proving techniques for a large class of systems that can be expressed using *preg* language specifications.

Paper structure. In Section 2 we introduce the *preg* rule format. In Section 3 we introduce an appropriate "core" language for expressing finite trees with predicates. We also provide a ground-complete axiomatization for bisimilarity over this type of trees, as our aim is to prove the completeness of our final axiomatization by head normalizing general *preg* terms, and therefore by reducing the completeness problem for arbitrary languages to that for trees.

Head normalizing general *preg* terms is not a straightforward process. Therefore, following [2], in Section 4 we introduce the notion of smooth and distinctive operation, adapted to the current setting. These operations are designed to "capture the behaviour of general *preg* operations", and are defined by rules satisfying a series of syntactic constraints with the purpose of enabling the construction of head normalizing axiomatizations. Such axiomatizations are based on a collection of equations that describe the interplay between smooth and distinctive operations, and the operations in the signature for finite trees. The existence of a sound and ground-complete axiomatization characterizing the bisimilarity of *preg* processes is finally proven in Section 5. A technical discussion on why it is important to handle predicates as first class notions, instead of encoding them by means of transition relations, is presented in Section 6. In Section 7 we draw some conclusions and provide pointers to future work.

2 GSOS with predicates

In this section we present the *preg* systems which are a generalization of GSOS [8] systems.

Consider a countably infinite set V of *process variables* (usually denoted by x, y, z) and a signature Σ consisting of a set of *operations* (denoted by f, g). The set of *process terms* $\mathbb{T}(\Sigma)$ is inductively defined as follows: each variable $x \in V$ is a term; if $f \in \Sigma$ is an operation of arity l, and if $S_1, \ldots, S_l$ are terms, then $f(S_1, \ldots, S_l)$ is a term. We write $T(\Sigma)$ in order to represent the set of *closed process terms* (*i.e.*, terms that do not contain variables), ranged over by t, s. A *substitution* σ is a function of type $V \to \mathbb{T}(\Sigma)$. If the range of a substitution is included in $T(\Sigma)$, we say that it is a *closed substitution*. Moreover, we write $[x \mapsto t]$ to represent a substitution that maps the variable x to the term t. Let $\vec{x} = x_1, \ldots, x_n$ be a sequence of pairwise distinct variables. A Σ-*context* $C[\vec{x}]$ is a term in which at most the variables $\vec{x}$ appear. For instance, $f(x, f(x,c))$ is a Σ-context, if the binary operation f and the constant c are in Σ.

Let $\mathcal{A}$ be a finite, nonempty set of *actions* (denoted by a, b, c). A *positive transition formula* is a triple (S,a,S') written $S \xrightarrow{a} S'$, with the intended meaning: process S performs action a and becomes process S'. A *negative transition formula* (S,a) written $S \overset{a}{\nrightarrow}$, states that process S cannot perform action a. Note that S, S' may contain variables. The "intended meaning" applies to closed process terms.

We now define *preg* – *pr*edicate *e*xtension of the *G*SOS rule format. Let $\mathcal{P}$ be a finite set of *predicates* (denoted by P,Q). A *positive predicate formula* is a pair (P,S), written PS, saying that process S satisfies predicate P. Dually, a *negative predicate formula* $\neg P\,S$ states that process S does not satisfy predicate P.

Definition 1 (*preg* rule format)**.** *Consider $\mathcal{A}$, a set of actions, and $\mathcal{P}$, a set of predicates.*

1. *A preg* transition rule *for an l-ary operation f is a deduction rule of the form:*

$$\frac{\begin{array}{cc}\{x_i \xrightarrow{a_{ij}} y_{ij} \mid i \in I^+, j \in I_i^+\} & \{P_{ij}x_i \mid i \in J^+, j \in J_i^+\} \\ \{x_i \overset{b}{\nrightarrow} \mid i \in I^-, b \in \mathcal{B}_i\} & \{\neg Q x_i \mid i \in J^-, Q \in \mathcal{Q}_i\}\end{array}}{f(x_1,\ldots,x_l) \xrightarrow{c} C[\vec{x},\vec{y}]}$$

   where

   (a) $x_1,\ldots,x_l$ *and* y_{ij} $(i \in I^+, j \in J^+)$ *are pairwise distinct variables;*

   (b) $I^+, J^+, I^-, J^- \subseteq L = \{1,\ldots,l\}$ *and each* I_i^+ *and* J_i^+ *is finite;*

   (c) a_{ij}, b *and* c *are actions in* $\mathcal{A}$ *(*$\mathcal{B}_i \subseteq \mathcal{A}$*); and*

   (d) P_{ij} *and* Q *are predicates in* $\mathcal{P}$ *(*$\mathcal{Q}_i \subseteq \mathcal{P}$*).*

2. *A preg* predicate rule *for an l-ary operation f is a deduction rule similar to the one above, with the only difference that its conclusion has the form $P(f(x_1,\ldots,x_l))$ for some $P \in \mathcal{P}$.*

Let ρ be a *preg* (transition or predicate) rule for f. The symbol f is the *principal operation* of ρ. All the formulas above the line are *antecedents* and the formula below is the *consequent*. We say that a position i for ρ is *tested positively* if $i \in I^+ \cup J^+$ and $I_i^+ \cup J_i^+ \neq \emptyset$. Similarly, i is *tested negatively* if $i \in I^- \cup J^-$ and $\mathcal{B}_i \cup \mathcal{Q}_i \neq \emptyset$. Whenever ρ is a transition rule for f, we say that $f(\vec{x})$ is the *source*, $C[\vec{x},\vec{y}]$ is the *target*, and c is the *action* of ρ. Whenever ρ is a predicate rule for f, we call $f(\vec{x})$ the *test* of ρ.

In order to avoid confusion, if in a certain context we use more than one rule, e.g. ρ, ρ', we parameterize the corresponding sets of indices with the name of the rule, *e.g.*, I_ρ^+, $J_{\rho'}^-$.

Definition 2 (*preg* system)**.** *A* preg *system is a pair $G = (\Sigma_G, \mathcal{R}_G)$, where Σ_G is a finite signature and $\mathcal{R}_G = \mathcal{R}_G^{\mathcal{A}} \cup \mathcal{R}_G^{\mathcal{P}}$ is a finite set of* preg *rules over Σ_G ($\mathcal{R}_G^{\mathcal{A}}$ and $\mathcal{R}_G^{\mathcal{P}}$ represent the transition and, respectively, the predicate rules of G).*

Consider a *preg* system G. Formally, the operational semantics of the closed process terms in G is fully characterized by the relations $\rightarrow_G \subseteq T(\Sigma_G) \times \mathcal{A} \times T(\Sigma_G)$ and $\ltimes_G \subseteq \mathcal{P} \times T(\Sigma_G)$, called the (unique) *sound and supported* transition and, respectively, predicate relations. Intuitively, soundness guarantees that $\rightarrow_G$ and $\ltimes_G$ are closed with respect to the application of the rules in $\mathcal{R}_G$ on $T(\Sigma_G)$, *i.e.*, $\rightarrow_G$ (resp. $\ltimes_G$) contains the set of all possible transitions (resp. predicates) process terms in $T(\Sigma_G)$ can perform (resp. satisfy) according to $\mathcal{R}_G$. The requirement that $\rightarrow_G$ and $\ltimes_G$ be supported means that all the transitions performed (resp. all the predicates satisfied) by a certain process term can be "derived" from the deductive system described by $\mathcal{R}_G$. As a notational convention, we write $S \xrightarrow{a}_G S'$ and $P_G S$ whenever $(S,a,S') \in \rightarrow_G$ and $(P,S) \in \ltimes_G$. We omit the subscript G when it is clear from the context.

Lemma 1. *Let G be a* preg *system. Then, for each $t \in T(\Sigma_G)$ the set $\{(a,t') \mid t \xrightarrow{a} t', a \in \mathcal{A}\}$ is finite.*

Next we introduce the notion of *bisimilarity* – the equivalence over processes we consider in this paper.

Definition 3 (Bisimulation)**.** *Consider a* preg *system $G = (\Sigma_G, \mathcal{R}_G)$. A symmetric relation $R \subseteq T(\Sigma_G) \times T(\Sigma_G)$ is a* bisimulation *iff:*

1. *for all $s,t,s' \in T(\Sigma_G)$, whenever $(s,t) \in R$ and $s \xrightarrow{a} s'$ for some $a \in \mathcal{A}$, then there is some $t' \in T(\Sigma_G)$ such that $t \xrightarrow{a} t'$ and $(s',t') \in R$;*
2. *whenever $(s,t) \in R$ and Ps ($P \in \mathcal{P}$) then Pt.*

Two closed terms s and t are bisimilar *(written $s \sim t$) iff there is a bisimulation relation R such that $(s,t) \in R$.*

Proposition 1. *Let G be a* preg *system. Then $\sim$ is an equivalence relation and a congruence for all operations f of G.*

Definition 4 (Disjoint extension)**.** *A* preg *system G' is a disjoint extension of a* preg *system G, written $G \sqsubseteq G'$, if the signature and the rules of G' include those of G, and G' does not introduce new rules for operations in G.*

It is well known that if $G \sqsubseteq G'$ then two terms in $T(\Sigma_G)$ are bisimilar in G if and only if they are bisimilar in G'.

From this point onward, our focus is to find a *sound and ground-complete axiomatization of bisimilarity on closed terms* for an arbitrary *preg* system G, *i.e.*, to identify a (finite) axiom system E_G so that $E_G \vdash s = t$ *iff* $s \sim t$ for all $s,t \in T(\Sigma_G)$. The method we apply is an adaptation of the technique in [2] to the *preg* setting. The strategy is to incrementally build a finite, head-normalizing axiomatization for general *preg* terms, *i.e.*, an axiomatization that, when applied recursively, reduces the completeness problem for arbitrary terms to that for synchronization trees. This way, the proof of ground-completeness for G reduces to showing the equality of closed tree terms.

3 Preliminary steps towards the axiomatization

In this section we start by identifying an appropriate language for expressing finite trees with predicates. We continue in the style of [2], by extending the language with a kind of restriction operator used for expressing the inability of a process to perform a certain action or to satisfy a given predicate. (This operator is used in the axiomatization of negative premises.) We provide the structural operational semantics of the resulting language, together with a sound and ground-complete axiomatization of bisimilarity on finite trees with predicates.

3.1 Finite trees with predicates

The language for trees we use in this paper is an extension with predicates of the language BCCSP [13]. The syntax of BCCSP consists of closed terms built from a constant δ (*deadlock*), the binary operator $\_+\_$ (*nondeterministic choice*), and the unary operators $a.\_$ (*action prefix*), where a ranges over the actions in a set $\mathcal{A}$. Let $\mathcal{P}$ be a set of predicates. For each $P \in \mathcal{P}$ we consider a process constant κ_P, which "witnesses" the associated predicate in the definition of a process. Intuitively, κ_P stands for a process that only satisfies predicate P and has no transition.

A finite tree term t is built according to the following grammar:

$$t ::= \delta \mid \kappa_P \ (\forall P \in \mathcal{P}) \mid a.t \ (\forall a \in \mathcal{A}) \mid t + t. \tag{1}$$

Intuitively, δ represents a process that does not exhibit any behaviour, $s+t$ is the nondeterministic choice between the behaviours of s and t, while $a.t$ is a process that first performs action a and behaves like t afterwards. The operational semantics that captures this intuition is given by the rules of BCCSP:

$$\frac{}{a.x \xrightarrow{a} x}\,(rl_1) \qquad \frac{x \xrightarrow{a} x'}{x+y \xrightarrow{a} x'}\,(rl_2) \qquad \frac{y \xrightarrow{a} y'}{x+y \xrightarrow{a} y'}\,(rl_3)$$

Figure 1: The semantics of BCCSP

As our goal is to extend BCCSP, the next step is to find an appropriate semantics for predicates. As can be seen in Fig. 1, action performance is determined by the shape of the terms. Consequently, we choose to define predicates in a similar fashion.

Consider a predicate P and the term $t = \kappa_P$. As previously mentioned, the purpose of κ_P is to witness the satisfiability of P. Therefore, it is natural to consider that κ_P satisfies P.

Take for example the *immediate termination* predicate $\downarrow$. As a term $s+s'$ exhibits the behaviour of both s and s', it is reasonable to state that $(s+s')\downarrow$ if $s\downarrow$ or $s'\downarrow$. Note that for a term $t = a.t'$ the statement $t\downarrow$ is in contradiction with the meaning of immediate termination, since t can initially only execute action a. Predicates of this kind are called *explicit predicates* in what follows.

Consider now the *eventual termination* predicate $\lightning$. In this situation, it is proper to consider that $(s+t)\lightning$ if $s\lightning$ or $t\lightning$ and, moreover, that $a.s\lightning$ if $s\lightning$. We refer to predicates such as $\lightning$ as *implicit predicates* (that range over a set $\mathcal{P}^{\mathcal{I}}$ included in $\mathcal{P}$), since their satisfiability propagates through the structure of tree terms in an implicit fashion. We denote by $\mathcal{A}_P$ (included in $\mathcal{A}$) the set consisting of the actions a for which this behaviour is permitted when reasoning on the satisfiability of predicate P.

The rules expressing the semantics of predicates are:

$$\frac{}{P\kappa_P}\,(rl_4) \quad \frac{Px}{P(x+y)}\,(rl_5) \quad \frac{Py}{P(x+y)}\,(rl_6) \quad \frac{Px}{P(a.x)}, \forall P \in \mathcal{P}^{\mathcal{I}}\ \forall a \in \mathcal{A}_P\ (rl_7)$$

Figure 2: The semantics of predicates

The operational semantics of trees with predicates is given by the set of rules (rl_1)–(rl_7) illustrated in Fig. 1 and Fig. 2. For notational consistency, we make the following conventions. Let $\mathcal{A}$ be an action set and $\mathcal{P}$ a set of predicates. Σ_{FTP} represents the signature of finite trees with predicates. $T(\Sigma_{\mathit{FTP}})$ is the set of (closed) tree terms built over Σ_{FTP}, and $\mathcal{R}_{\mathit{FTP}}$ is the set of rules (rl_1)–(rl_7). Moreover, by *FTP* we denote the system $(\Sigma_{\mathit{FTP}}, \mathcal{R}_{\mathit{FTP}})$.

Discussion on the design decisions. At first sight, it seems reasonable for our framework to allow for language specifications containing rules of the shape $\frac{}{P(x+y)}$, or just one of (rl_5) and (rl_6). We decided, however, to disallow them, as their presence would invalidate standard algebraic properties such as the idempotence and the commutativity of $\_+\_$.

Without loss of generality we avoid rules of the form $\frac{}{P(a.x)}$. As far as the user is concerned, in order to express that $a.x$ satisfies a predicate P, one can always add the witness κ_P as a summand: $a.x+\kappa_P$. This decision helped us avoid some technical problems for the soundness and completeness proofs for the case of the restriction operator $\partial_{\mathcal{B},\mathcal{Q}}$, which is presented in Section 3.3.

Due to the aforementioned restriction, we also had to leave out universal predicates with rules of the form $\frac{Px\ Py}{P(x+y)}$. However, the elimination of universal predicates is not a theoretical limitation to what one can express, since a universal predicate can always be defined as the negation of an existential one.

As a last approach, we thought of allowing the user to specify existential predicates using rules of the form $\frac{P_1x...P_nx}{P(x+y)}(*)$ and $\frac{P_1y...P_ny}{P(x+y)}(**)$ (instead of (rl_5) and (rl_6)). However, in order to maintain the validity of the axiom $x+x=x$ in the presence of rules of these forms, it would have to be the case that one of the predicates P_i in the premises is P itself. (If that were not the case, then let t be the sum of the constants witnessing the P_i's for a rule of the form $(*)$ above with a minimal set of set premises. We have that $t+t$ satisfies P by rule $(*)$. On the other hand, Pt does not hold since none of the P_i is equal to P and no rule for P with a smaller set of premises exists.) Now, if a rule of the form $(*)$ has a premise of the form Px, then it is subsumed by (rl_5) which we must have to ensure the validity of laws such as $\kappa_P = \kappa_P + \kappa_P$.

3.2 Axiomatizing finite trees

In what follows we provide a finite sound and ground-complete axiomatization (E_{FTP}) for bisimilarity over finite trees with predicates.

The axiom system E_{FTP} consists of the following axioms:

$$\begin{array}{llll} x+y = y+x & (A_1) & x+x = x & (A_3) \\ (x+y)+z = x+(y+z) & (A_2) & x+\delta = x & (A_4) \\ \multicolumn{3}{l}{a.(x+\kappa_P) = a.(x+\kappa_P)+\kappa_P, \forall P \in \mathcal{P}^{\mathcal{I}}\ \forall a \in \mathcal{A}_P} & (A_5) \end{array}$$

Figure 3: The axiom system E_{FTP}

Axioms (A_1)–(A_4) are well-known [16]. Axiom (A_5) describes the propagation of witness constants for the case of implicit predicates.

We now introduce the notion of terms in *head normal form*. This concept plays a key role in the proofs of completeness for the axiom systems generated by our framework.

Definition 5 (Head Normal Form). *Let Σ be a signature such that $\Sigma_{FTP} \subseteq \Sigma$. A term t in $T(\Sigma)$ is in* head normal form *(for short, h.n.f.) if*

$$t = \sum_{i\in I} a_i.t_i + \sum_{j\in J} \kappa_{P_j}, \text{and the } P_j \text{ are all the predicates satisfied by } t.$$

The empty sum $(I=\emptyset, J=\emptyset)$ is denoted by the deadlock constant δ.

Lemma 2. *E_{FTP} is head normalizing for terms in $T(\Sigma_{FTP})$. That is, for all t in $T(\Sigma_{FTP})$, there exists t' in $T(\Sigma_{FTP})$ in h.n.f. such that $E_{FTP} \vdash t = t'$ holds.*

Proof. The reasoning is by induction on the structure of t. ☐

Theorem 1. *E_{FTP} is sound and ground-complete for bisimilarity on $T(\Sigma_{FTP})$. That is, it holds that $(\forall t,t' \in T(\Sigma_{FTP}))\,.E_{FTP} \vdash t = t'$ iff $t \sim t$.*

3.3 Axiomatizing negative premises

A crucial step in finding a complete axiomatization for *preg* systems is the “axiomatization” of negative premises (of the shape $x \overset{a}{\nrightarrow}, \neg Px$). In the style of [2], we introduce the restriction operator $\partial_{\mathcal{B},\mathcal{Q}}$, where $\mathcal{B} \subseteq \mathcal{A}$ and $\mathcal{Q} \subseteq \mathcal{P}$ are the sets of initially forbidden actions and predicates, respectively. The semantics of $\partial_{\mathcal{B},\mathcal{Q}}$ is given by the two types of transition rules in Fig. 4.

$$\frac{x \xrightarrow{a} x'}{\partial_{\mathcal{B},\mathcal{Q}}(x) \xrightarrow{a} \partial_{\emptyset,\mathcal{Q}\cap\mathcal{P}^{\mathcal{I}}}(x')} \text{ if } a \notin \mathcal{B}\ (rl_8) \quad \frac{Px}{P(\partial_{\mathcal{B},\mathcal{Q}}(x))} \text{ if } P \notin \mathcal{Q}\ (rl_9)$$

Figure 4: The semantics of $\partial_{\mathcal{B},\mathcal{Q}}$

Note that $\partial_{\mathcal{B},\mathcal{Q}}$ behaves like the one step restriction operator in [2] for the actions in $\mathcal{B}$, as the restriction on the action set disappears after one transition. On the other hand, for the case of predicates in $\mathcal{Q}$, the operator $\partial_{\mathcal{B},\mathcal{Q}}$ resembles the CCS restriction operator [16] since, due to the presence of implicit predicates, not all the restrictions related to predicate satisfaction necessarily disappear after one step, as will become clear in what follows.

We write E^{∂}_{FTP} for the extension of E_{FTP} with the axioms involving $\partial_{\mathcal{B},\mathcal{Q}}$ presented in Fig. 5. $\mathcal{R}^{\partial}_{FTP}$ stands for the set of rules (rl_1)–(rl_9), while FTP^{∂} represents the system $(\Sigma^{\partial}_{FTP}, \mathcal{R}^{\partial}_{FTP})$.

$$\begin{array}{llllll}
\partial_{\mathcal{B},\mathcal{Q}}(\delta) = \delta & & (A_6) & \partial_{\mathcal{B},\mathcal{Q}}(a.x) = \sum_{P\notin\mathcal{Q},P(a.x)} \kappa_P & \text{if } a \in \mathcal{B} & (A_9)\\
\partial_{\mathcal{B},\mathcal{Q}}(\kappa_P) = \delta & \text{if } P \in \mathcal{Q} & (A_7) & \partial_{\mathcal{B},\mathcal{Q}}(a.x) = \partial_{\emptyset,\mathcal{Q}}(a.x) & \text{if } a \notin \mathcal{B} & (A_{10})\\
\partial_{\mathcal{B},\mathcal{Q}}(\kappa_P) = \kappa_P & \text{if } P \notin \mathcal{Q} & (A_8) & \partial_{\emptyset,\mathcal{Q}}(a.x) = a.\partial_{\emptyset,\mathcal{Q}\cap\mathcal{P}^{\mathcal{I}}}(x) & & (A_{11})
\end{array}$$
$$\partial_{\mathcal{B},\mathcal{Q}}(x+y) = \partial_{\mathcal{B},\mathcal{Q}}(x) + \partial_{\mathcal{B},\mathcal{Q}}(y)\ (A_{12})$$

Figure 5: The axiom system $E^{\partial}_{FTP} \setminus E_{FTP}$

Axiom (A_6) states that it is useless to impose restrictions on δ, as δ does not exhibit any behaviour. The intuition behind (A_7) is that since a predicate witness κ_P does not perform any action, inhibiting the satisfiability of P leads to a process with no behaviour, namely δ. Consequently, if the restricted predicates do not include P, the resulting process is κ_P itself (see (A_8)). Inhibiting the only action a process $a.t$ can perform leads to a new process that, in the best case, satisfies some of the predicates in $\mathcal{P}^{\mathcal{I}}$ satisfied by t (by (rl_7)) if $\mathcal{Q} \neq \mathcal{P}^{\mathcal{I}}$ (see (A_9)). Whenever the restricted action set $\mathcal{B}$ does not contain the only action a process $a.t$ can perform, then it is safe to give up $\mathcal{B}$ (see (A_{10})). As a process $a.t$ only satisfies the predicates also satisfied by t, it is straightforward to see that $\partial_{\emptyset,\mathcal{Q}}(a.t)$ is equivalent to the process obtained by propagating the restrictions on implicit predicates deeper into the behaviour of t (see (A_{11})). Axiom (A_{12}) is given in conformity with the semantics of $\_+\_$ ($s+t$ encapsulates both the behaviours of s and t).

Remark 1. *For the sake of brevity and readability, in Fig. 5 we presented (A_9), which is a schema with infinitely many instances. However, it can be replaced by a finite family of axioms. See Appendix D in the full version of the paper available at* `http://www.ru.is/faculty/luca/PAPERS/axgsos.pdf` *for details.*

Theorem 2. *The following statements hold for E^{∂}_{FTP}:*

1. *E^{∂}_{FTP} is sound for bisimilarity on $T(\Sigma^{\partial}_{FTP})$.*

2. $\forall t \in T(\Sigma^{\partial}_{\mathit{FTP}}), \exists t' \in T(\Sigma_{\mathit{FTP}})\ s.t.\ E^{\partial}_{\mathit{FTP}} \vdash t = t'$.

As proving completeness for *FTP*$^{\partial}$ can be reduced to showing completeness for *FTP* (already proved in Theorem 1), the following result is an immediate consequence of Theorem 2:

Corollary 1. *$E^{\partial}_{\mathit{FTP}}$ is sound and complete for bisimilarity on $T(\Sigma^{\partial}_{\mathit{FTP}})$.*

4 Smooth and distinctive operations

Recall that our goal is to provide a sound and ground-complete axiomatization for bisimilarity on systems specified in the *preg* format. As the *preg* format is too permissive for achieving this result directly, our next task is to find a class of operations for which we can build such an axiomatization by "easily" reducing it to the completeness result for *FTP*, presented in Theorem 1. In the literature, these operations are known as *smooth and distinctive* [2]. As we will see, these operations are incrementally identified by imposing suitable restrictions on *preg* rules. The standard procedure is to first find the *smooth* operations, based on which one determines the *distinctive* ones.

Definition 6 (Smooth operation)**.**

1. *A* preg *transition rule is* smooth *if it is of the following format:*

$$\frac{\{x_i \xrightarrow{a_i} y_i \mid i \in I^+\} \qquad \{P_i x_i \mid i \in J^+\} \qquad \{x_i \overset{b}{\not\rightarrow} \mid i \in I^-, b \in \mathcal{B}_i\} \quad \{\neg Q x_i \mid i \in J^-, Q \in \mathcal{Q}_i\}}{f(x_1,\ldots,x_l) \xrightarrow{c} C[\vec{x},\vec{y}]}$$

where

(a) I^+, J^+, I^-, J^- *disjointly cover the set* $L = \{1,\ldots,l\}$,

(b) *in the target* $C[\vec{x},\vec{y}]$ *we allow only:* y_i $(i \in I^+)$, x_i $(i \in I^- \cup J^-)$.

2. *A* preg *predicate rule is* smooth *if it has the form above, its premises satisfy condition (1a) and its conclusion is* $P(f(x_1,\ldots,x_l))$ *for some* $P \in \mathcal{P}$.

3. *An operation f of a* preg *system is* smooth *if all its (transition and predicate) rules are smooth.*

By Definition 6, a rule ρ is smooth if it satisfies the following properties:

- a position i cannot be tested both positively and negatively at the same time,
- positions tested positively are either from I^+ or J^+ and they are not tested for the performance of multiple transitions (respectively, for the satisfiability of multiple predicates) within the same rule, and
- if ρ is a transition rule, then the occurrence of variables at positions $i \in I^+ \cup J^+$ is not allowed in the target of the consequent of ρ.

Remark 2. *Note that we can always consider a position i that does not occur as a premise in a rule for f as being negative, with the empty set of constraints (i.e. either $i \in I^-$ and $\mathcal{B}_i = \emptyset$, or $i \in J^-$ and $\mathcal{Q}_i = \emptyset$).*

Definition 7 (Distinctive operation)**.** *An operation f of a* preg *system is* distinctive *if it is smooth and:*

- *for each argument i, either all rules for f test i positively, or none of them does, and*
- *for any two distinct rules for f there exists a position i tested positively, such that one of the following holds:*

- *both rules have actions that are different in the premise at position i,*
- *both rules have predicates that are different in the premise at position i,*
- *one rule has an action premise at position i, and the other rule has a predicate test at the same position i.*

According to the first requirement in Definition 7, we state that for a smooth and distinctive operation f, a position i is *positive* (respectively, *negative*) for f if there is a rule for f such that i is tested positively (respectively, negatively) for that rule.

The existence of a family of smooth and distinctive operations "describing the behaviour" of a general *preg* operation is formalized by the following lemma:

Lemma 3. *Consider a* preg *system G. Then there exist a* preg *system G', which is a disjoint extension of G and FTP, and a finite axiom system E such that*

1. *E is sound for bisimilarity over any disjoint extension G'' of G', and*
2. *for each term t in $T(\Sigma_G)$ there is some term t' in $T(\Sigma_{G'})$ such that t' is built solely using smooth and distinctive operations and E proves $t = t'$.*

4.1 Axiomatizing smooth and distinctive *preg* operations

To start with, consider, for the good flow of the presentation, that we only handle explicit predicates (*i.e.*, we take $\mathcal{P}^{\mathcal{I}} = \emptyset$). Towards the end of the section we discuss how to extend the presented theory to implicit predicates. We proceed in a similar fashion to [2] by defining a set of laws used in the construction of a complete axiomatization for bisimilarity on terms built over smooth and distinctive operations. The strength of these laws lies in their capability of reducing terms to their head normal form, thus reducing completeness for general *preg* systems to completeness of E_{FTP} (which has already been proved in Section 3.2).

Definition 8. *Let f be a smooth and distinctive l-ary operation of a* preg *system G, such that $FTP^{\partial} \sqsubseteq G$.*

1. *For a positive position $i \in L = \{1,\ldots,l\}$, the* distributivity law *for i w.r.t. f is given as follows:*

$$f(X_1,\ldots,X_i'+X_i'',\ldots,X_l) = f(X_1,\ldots,X_i',\ldots,X_l) + f(X_1,\ldots,X_i'',\ldots,X_l).$$

2. *For a rule $\rho \in \mathcal{R}$ for f the* trigger law *is, depending on whether ρ is a transition or a predicate rule:*

$$f(\vec{X}) = \begin{cases} c.C[\vec{X},\vec{y}] & , \rho \in \mathcal{R}^{\mathcal{A}} \quad \text{(action law)} \\ \kappa_P & , \rho \in \mathcal{R}^{\mathcal{P}} \quad \text{(predicate law)} \end{cases}$$

   where

$$X_i \equiv \begin{cases} a_i.y_i & , i \in I^+ \\ \kappa_{P_i} & , i \in J^+ \\ \partial_{\mathcal{B}_i,\mathcal{Q}_i}(x_i) & , i \in I^- \cup J^- \end{cases} .$$

3. *Suppose that for $i \in L$, term X_i is in one of the forms $\delta, z_i, \kappa_{P_i}, a.z_i, a.z_i + z_i'$ or $\kappa_{P_i} + z_i$. Suppose further that for each rule for f there exists $X_j \in \vec{X}$ ($j \in \{1,\ldots,l\}$) s.t. one of the following holds:*
   - *$j \in I^+$ and ($X_j \equiv \delta$ or $X_j \equiv b.z_j$ ($b \neq a_j$) or $X_j \equiv \kappa_Q$, for some Q),*

- *$j \in J^+$ and ($X_j \equiv \delta$ or $X_j \equiv \kappa_Q$ ($Q \neq P_j$) or $X_j \equiv b.z_j$, for some b),*
- *$j \in I^-$ and $X_j \equiv b.z_j + z'_j$, where $b \in \mathcal{B}_j$,*
- *$j \in J^-$ and $X_j \equiv \kappa_Q + z_j$, where $Q \in \mathcal{Q}_j$.*

Then the deadlock law *is as follows:*

$$f(\vec{X}) = \delta.$$

Example 1. *Consider the* right-biased sequential composition *operation $\_ ;^r \_$, whose semantics is given by the rules $\frac{x\downarrow\ y \xrightarrow{a} y'}{x ;^r y \xrightarrow{a} y'}$, $\frac{x\downarrow\ y\downarrow}{(x ;^r y)\downarrow}$, and $\frac{x\downarrow\ y\uparrow}{(x ;^r y)\uparrow}$, where $\downarrow$ and $\uparrow$ are, respectively, the* immediate termination *and* immediate divergence *predicates. $\_ ;^r \_$ is one of the auxiliary operations generated by the algorithm for deriving smooth and distinctive operations when axiomatizing the* sequential composition *in the presence of the two mentioned predicates.*

The laws derived according to Definition 8 for this system are:

$$\begin{array}{rcl@{\qquad}rcl}
(x+y) ;^r z & = & x ;^r z \ + \ y ;^r z & \delta ;^r y & = & \delta \\
x ;^r (y+z) & = & x ;^r y \ + \ z ;^r z & k_\uparrow ;^r y & = & \delta \\
k_\downarrow ;^r a.y & = & a.y & a.x ;^r y & = & \delta \\
k_\downarrow ;^r k_\downarrow & = & k_\downarrow & x ;^r \delta & = & \delta \\
k_\downarrow ;^r k_\uparrow & = & k_\uparrow & & \ldots &
\end{array}$$

Theorem 3. *Consider G a* preg *system such that FTP$^\partial \sqsubseteq G$. Let $\Sigma \subseteq \Sigma_G \setminus \Sigma^\partial_{FTP}$ be a collection of smooth and distinctive operations of G. Let E_G be the finite axiom system that extends E^∂_{FTP} with the following axioms for each $f \in \Sigma$:*

- *for each positive argument i of f, a distributivity law (Definition 8.1),*
- *for each transition rule for f, an action law (Definition 8.2),*
- *for each predicate rule for f, a predicate law (Definition 8.2), and*
- *all deadlock laws for f (Definition 8.3).*

The following statements hold for E_G, for any G' such that $G \sqsubseteq G'$:

1. *E_G is sound for bisimilarity on $T(\Sigma_{G'})$.*
2. *E_G is head normalizing for $T(\Sigma \cup \Sigma^\partial_{FTP})$.*

Obtaining the soundness of the action law (Definition 8.2) requires some care when allowing for specifications with implicit predicates ($\mathcal{P}^\mathcal{I} \neq \emptyset$). Consider a scenario in which a transition rule for a smooth and distinctive operation f is of the form $\frac{H}{f(\vec{X}) \xrightarrow{c} C[\vec{X},\vec{y}]}$. Assume the closed instantiation $\vec{X} = \vec{s}$, $\vec{y} = \vec{t}$ and assume that $P(c.C[\vec{s},\vec{t}])$ holds for some predicate P in $\mathcal{P}^\mathcal{I}$. This means that $P(C[\vec{s},\vec{t}])$ holds. In order to preserve the soundness of the action law, $P(f(\vec{s}))$ should also hold, but this is impossible since f is distinctive. One possible way of ensuring the soundness of the action law in the presence of implicit predicates is to stipulate some syntactic consistency requirements on the language specification. One sufficient requirement would be that if predicate rule $\frac{H'}{P(C[\vec{z},\vec{y}])}$ is derivable, then the system should contain a predicate rule $\frac{H''}{P(f[\vec{z}])}$ with $H'' \subseteq H'$. This is enough to guarantee that if the right-hand side of the action law satisfies P then so does the left-hand side.

5 Soundness and completeness

Let us summarize our results so far. By Theorem 3, it follows that, for any *preg* system $G \sqsupseteq \mathit{FTP}^{\partial}$, there is an axiomatization that is head normalizing for $T(\Sigma \cup \Sigma^{\partial}_{\mathit{FTP}})$, where $\Sigma \subseteq \Sigma_G \setminus \Sigma^{\partial}_{\mathit{FTP}}$ is a collection of smooth and distinctive operations of G. Also, as hinted in Section 4 (Lemma 3), there exists a sound algorithm for transforming general *preg* operations to smooth and distinctive ones.

So, for any *preg* system G, we can build a *preg* system $G' \sqsupseteq G$ and an axiomatization $E_{G'}$ that is head normalizing for $T(\Sigma_{G'})$. This statement is formalized as follows:

Theorem 4. *Let G be a* preg *system. Then there exist $G' \sqsupseteq G$ and a finite axiom system $E_{G'}$ such that*

1. *$E_{G'}$ is sound for bisimilarity on $T(\Sigma_{G'})$,*
2. *$E_{G'}$ is head normalizing for $T(\Sigma_{G'})$,*

and moreover, G' and $E_{G'}$ can be effectively constructed from G.

Proof. The result follows immediately by Theorem 3 and by the existence of an algorithm used for transforming general *preg* to smooth and distinctive operations. ☐

Remark 3. *Theorem 4 guarantees ground-completeness of the generated axiomatization for well-founded* preg *specifications, that is,* preg *specifications in which each process can only exhibit finite behaviour.*

Let us further recall an example given in [2]. Consider the constant ω, specified by the rule $\omega \xrightarrow{a} \omega$. Obviously, the corresponding action law $\omega = a.\omega$ will apply for an infinite number of times in the normalization process. So the last step in obtaining a complete axiomatization is to handle infinite behaviour.

Let t and t' be two processes with infinite behaviour (remark that the infinite behaviour is a consequence of performing actions for an infinite number of times, so the extension to predicates is not a cause for this issue). Since we are dealing with finitely branching processes, it is well known that if two process terms are bisimilar at each finite depth, then they are bisimilar. One way of formalizing this requirement is to use the well-known *Approximation Induction Principle* (AIP) [5, 7].

Let us first consider the operations $\pi_n(\cdot)$, $n \in \mathbb{N}$, known as *projection operations*. The purpose of these operations is to stop the evolution of processes after a certain number of steps. The AIP is given by the following conditional equation:

$$x = y \text{ if } \pi_n(x) = \pi_n(y) \ (\forall n \in \mathbb{N}).$$

We further adapt the idea in [2] to our context, and model the infinite family of projection operations $\pi_n(\cdot)$, $n \in \mathbb{N}$, by a binary operation $\cdot/\cdot$ defined as follows:

$$\frac{x \xrightarrow{a} x' \quad h \xrightarrow{c} h'}{x/h \xrightarrow{a} x'/h'}\ (rl_{10}) \qquad \frac{Px}{P(x/h)}\ (rl_{11})$$

where c is an arbitrary action. Note that $\cdot/\cdot$ is a smooth and distinctive operation.

The role of variable h is to "control" the evolution of a process, *i.e.*, to stop the process in performing actions, after a given number of steps. Variable h (the "hourglass" in [2]) will always be instantiated with terms of the shape c^n, inductively defined as: $c^0 = \delta$, $c^{n+1} = c.c^n$.

Let $G = (\Sigma_G, \mathcal{R}_G)$ be a *preg* system. We use the notation $G_/$ to refer to the *preg* system $(\Sigma_G \cup \{\cdot/\cdot\}, \mathcal{R}_G \cup \{(rl_{10}), (rl_{11})\})$ – the extension of G with $\cdot/\cdot$. Moreover, we use the notation E_{AIP} to refer to the axioms for the smooth and distinctive operation $\cdot/\cdot$, derived as in Section 4.1 – Definition 8.

We reformulate AIP according to the new operation $\cdot/\cdot$:

$$x = y \text{ if } x/c^n = y/c^n \ (\forall n \in \mathbb{N})$$

Lemma 4. *AIP is sound for bisimilarity on $T(\Sigma_{\mathit{FTP}_/})$.*

In what follows we provide the final ingredients for proving the existence of a ground-complete axiomatization for bisimilarity on *preg* systems. As previously stated, this is achieved by reducing completeness to proving equality in *FTP*. So, based on AIP, it would suffice to show that for any closed process term t and natural number n, there exists an *FTP* term equivalent to t at moment n in time:

Lemma 5. *Consider G a* preg *system. Then there exist $G' \sqsupseteq G_/$ and $E_{G'}$ with the property: $\forall t \in T(\Sigma_{G'}), \forall n \in \mathbb{N}, \exists t' \in T(\Sigma_{\mathit{FTP}})$ s.t. $E_{G'} \vdash t/c^n = t'$.*

At this point we can prove the existence of a sound and ground-complete axiomatization for bisimilarity on general *preg* systems:

Theorem 5 (Soundness and Completeness)**.** *Consider G a* preg *system. Then there exist $G' \sqsupseteq G_/$ and $E_{G'}$ a finite axiom system, such that $E_{G'} \cup E_{\mathit{AIP}}$ is sound and complete for bisimilarity on $T(\Sigma_{G'})$.*

6 Motivation for handling predicates as first-class notions

In the literature on the theory of rule formats for Structural Operational Semantics (especially, the work devoted to congruence formats for various notions of bisimilarity), predicates are often neglected at first and are only added to the considerations at a later stage. The reason is that one can encode predicates quite easily by means of transition relations. One can find a number of such encodings in the literature—see, for instance, [11, 19]. In each of these encodings, a predicate P is represented as a transition relation $\xrightarrow{P}$ (assuming that P is a fresh action label) with a fixed constant symbol as target. Using this translation, one can axiomatize bisimilarity over *preg* language specifications by first converting them into "equivalent" standard GSOS systems, and then applying the algorithm from [2] to obtain a finite axiomatization of bisimilarity over the resulting GSOS system.

In light of this approach, it is natural to wonder whether it is worthwhile to develop an algorithm to axiomatize *preg* language specifications directly. One possible answer, which has been presented several times in the literature [19], is that often one does not want to encode a language specification with predicates using one with transitions only. Sometimes, specifications using predicates are the most natural ones to write, and one should not force a language designer to code predicates using transitions. (However, one can write a tool to perform the translation of predicates into transitions, which can therefore be carried out transparently to the user/language designer.) Also, developing an algorithm to axiomatize GSOS language specifications with predicates directly yields insight into the difficulties that result from the first-class use of, and the interplay among, various types of predicates, as far as axiomatizability problems are concerned. These issues would be hidden by encoding predicates as transitions. Moreover, the algorithm resulting from the encoding would generate axioms involving predicate-prefixing operators, which are somewhat unintuitive.

Naturalness is, however, often in the eye of the beholder. Therefore, we now provide a more technical reason why it may be worthwhile to develop techniques that apply to GSOS language specifications with predicates as first-class notions, such as the *preg* ones. Indeed, we now show how, using predicates, one can convert any standard GSOS language specification G into an equivalent *positive* one with predicates G^+.

Given a GSOS language G, the system G^+ will have the same signature and the same set of actions as G, but uses predicates $\text{cannot}(a)$ for each action a. The idea is simply that "$x\,\text{cannot}(a)$" is the predicate formula that expresses that "x does not afford an a-labelled transition". The translation works as follows.

1. Each rule in G is also a rule in G^+, but one replaces each negative premise in each rule with its corresponding positive predicate premise. This means that $x \overset{a}{\nrightarrow}$ becomes $x\,\text{cannot}(a)$.
2. One adds to G^+ rules defining the predicates $\text{cannot}(a)$, for each action a. This is done in such a way that $p\,\text{cannot}(a)$ holds in G^+ exactly when $p \overset{a}{\nrightarrow}$ in G, for each closed term p and action a. More precisely, we proceed as follows.
   (a) For each constant symbol f and action a, add the rule

   $$\frac{}{f\,\text{cannot}(a)}$$

   whenever there is no transition rule in G with f as principal operation and with an a-labelled transition as its consequent.

   (b) For each operation f with arity at least one and action a, let $R(f,a)$ be the set of rules in G that have f as principal operation and an a-labelled transition as consequent. We want to add rules for the predicate $\text{cannot}(a)$ to G^+ that allow us to prove the predicate formula $f(p_1,\ldots,p_l)\,\text{cannot}(a)$ exactly when $f(p_1,\ldots,p_l)$ does not afford an a-labelled transition in G. This occurs if, for each rule in $R(f,a)$, there is some premise that is not satisfied when the arguments of f are $p_1,\ldots,p_l$. To formalize this idea, let $H(R(f,a))$ be the collection of premises of rules in $R(f,a)$. We say that a choice function is a function $\phi : R(f,a) \to H(R(f,a))$ that maps each rule in $R(f,a)$ to one of its premises. Let

   $$\begin{aligned} \text{neg}(x \xrightarrow{a} x') &= x\,\text{cannot}(a) \quad \text{and} \\ \text{neg}(x \overset{a}{\nrightarrow}) &= x \xrightarrow{a} x', \quad \text{for some } x'. \end{aligned}$$

   Then, for each choice function ϕ, we add to G^+ a predicate rule of the form

   $$\frac{\{\text{neg}(\phi(\xi)) \mid \xi \in R(f,a)\}}{f(x_1,\ldots,x_l)\,\text{cannot}(a)},$$

   where the targets of the positive transition formulae in the premises are chosen to be all different.

The above construction ensures the validity of the following lemma.

Lemma 6. *For each closed term p and action a,*

1. *$p \xrightarrow{a} p'$ in G if, and only if, $p \xrightarrow{a} p'$ in G^+;*
2. *$p\,\text{cannot}(a)$ in G^+ if, and only if, $p \overset{a}{\nrightarrow}$ in G^+ (and therefore in G).*

This means that two closed terms are bisimilar in G if, and only if, they are bisimilar in G^+. Moreover, two closed terms are bisimilar in G^+ iff they are bisimilar when we only consider the transitions (and not the predicates $\text{cannot}(a)$).

The language G^+ modulo bisimilarity can be axiomatized using our algorithm without the need for the exponentially many restriction operators. The conversion to positive GSOS with predicates discussed above does incur in an exponential blow-up in the number of rules, but it gives an alternative way of generating ground-complete axiomatizations for standard GSOS languages to the one proposed in [2]. In general, it is useful to have several approaches in one's toolbox, since one may choose the one that is "less expensive" for the specific task at hand. Moreover, using positive GSOS operations, one can also try to extend the methods from the full version of the paper [1] (see Section 7.1 in the technical report available at `http://www.ru.is/~luca/PAPERS/cs011994.ps`) to optimize these axiomatizations. We are currently working on applying such methods to positive *preg* systems with universal as well as existential predicates, and on extending our tool [3] accordingly.

It is worth noting that the predicates cannot(a) are not implicit, therefore the restrictions presented at the end of Section 4.1 need not to be imposed.

7 Conclusions and future work

In this paper we have introduced the *preg* rule format, a natural extension of GSOS with arbitrary predicates. Moreover, we have provided a procedure (similar to the one in [2]) for deriving sound and ground-complete axiomatizations for bisimilarity of systems that match this format. In the current approach, explicit predicates are handled by considering constants witnessing their satisfiability as summands in tree expressions. Consequently, there is no explicit predicate P satisfied by a term of shape $\Sigma_{i\in I} a_i.t_i$.

The procedure introduced in this paper has also enabled the implementation of a tool [3] that can be used to automatically reason on bisimilarity of systems specified as terms built over operations defined by *preg* rules.

Several possible extensions are left as future work. It would be worth investigating the properties of positive *preg* languages. By allowing only positive premises we eliminate the need of the restriction operators ($\partial_{\mathcal{B},\mathcal{Q}}$) during the axiomatization process. This would enable us to deal with more general predicates over trees, such as those that may be satisfied by terms of the form $a.t$ where a ranges over some subset of the collection of actions.

Another direction for future research is that of understanding the presented work from a coalgebraic perspective. The extensions from [2] to the present paper, might be thought as an extension from coalgebras for a functor $\mathscr{P}(\mathcal{A} \times Id)$ to a functor $\mathscr{P}(\mathcal{P}) \times \mathscr{P}(\mathcal{A} \times Id)$ where $\mathscr{P}$ is the powerset functor, $\mathcal{A}$ is the set of actions and $\mathcal{P}$ is the set of predicates. Also the language *FTP* coincides, apart from the recursion operator, with the one that would be obtained for the functor $\mathscr{P}(\mathcal{P}) \times \mathscr{P}(\mathcal{A} \times Id)$ in the context of Kripke polynomial coalgebras [9].

Finally, we plan to extend our axiomatization theory in order to reason on the bisimilarity of guarded recursively defined terms, following the line presented in [1].

Acknowledgments. The authors are grateful for the useful comments and suggestions from Alexandra Silva and three anonymous reviewers.

References

[1] Luca Aceto (1994): *Deriving Complete Inference Systems for a Class of GSOS Languages Generation Regular Behaviours*. In Jonsson & Parrow [15], pp. 449–464, doi:`10.1007/BFb0015025`.

[2] Luca Aceto, Bard Bloom & Frits Vaandrager (1994): *Turning SOS rules into equations*. *Inf. Comput.* 111, pp. 1–52, doi:`10.1006/inco.1994.1040`. Available at `http://portal.acm.org/citation.cfm?id=184662.184663`.

[3] Luca Aceto, Georgiana Caltais, Eugen-Ioan Goriac & Anna Ingolfsdottir (2011): *PREG Axiomatizer : A ground bisimilarity checker for GSOS with predicates*. In: *CALCO 2011*, LNCS, Springer. Available at `http://goriac.info/tools/preg-axiomatizer/`. To appear.

[4] J. C. M. Baeten, T. Basten & M. A. Reniers (2010): *Process Algebra: Equational Theories of Communicating Processes*. *Cambridge Tracts in Theoretical Computer Science* 50, Cambridge University Press, Cambridge.

[5] J. C. M. Baeten & W. P. Weijland (1990): *Process Algebra*. Cambridge University Press, New York, NY, USA.

[6] Jos C. M. Baeten & Erik P. de Vink (2004): *Axiomatizing GSOS with termination*. *J. Log. Algebr. Program.* 60-61, pp. 323–351, doi:`10.1016/j.jlap.2004.03.001`.

[7] J A Bergstra & J W Klop (1986): *Verification of an alternating bit protocol by means of process algebra*. In: *Proceedings of the International Spring School on Mathematical method of specification and synthesis of software systems '85*, Springer-Verlag New York, Inc., New York, NY, USA, pp. 9–23. Available at `http://portal.acm.org/citation.cfm?id=16663.16664`.

[8] Bard Bloom, Sorin Istrail & Albert R. Meyer (1995): *Bisimulation can't be traced*. *J. ACM* 42, pp. 232–268, doi:`10.1145/200836.200876`.

[9] Marcello M. Bonsangue, Jan J. M. M. Rutten & Alexandra Silva (2009): *An Algebra for Kripke Polynomial Coalgebras*. In: *LICS*, IEEE Computer Society, pp. 49–58, doi:`10.1109/LICS.2009.18`.

[10] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer & Carolyn L. Talcott, editors (2007): *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*. *Lecture Notes in Computer Science* 4350, Springer.

[11] Sjoerd Cranen, MohammadReza Mousavi & Michel A. Reniers (2008): *A Rule Format for Associativity*. In Franck van Breugel & Marsha Chechik, editors: *Proceedings of the 19th International Conference on Concurrency Theory (CONCUR'08)*, *Lecture Notes in Computer Science* 5201, Springer-Verlag, Berlin, Germany, Toronto,Canada, pp. 447–461, doi:`10.1007/978-3-540-85361-9_35`.

[12] M. Gazda & W.J. Fokkink (2010): *Turning GSOS into equations for linear time-branching time semantics*. *2nd Young Researchers Workshop on Concurrency Theory - YR-CONCUR'10, Paris* Available at `http://www.cs.vu.nl/~wanf/pubs/gsos.pdf`.

[13] R.J. van Glabbeek (2001): *The Linear Time - Branching Time Spectrum I. The Semantics of Concrete, Sequential Processes*. In A. Ponse S.A. Smolka J.A. Bergstra, editor: *Handbook of Process Algebra*, Elsevier, pp. 3–99.

[14] C.A.R. Hoare (1985): *Communicating Sequential Processes*. Prentice-Hall International, Englewood Cliffs.

[15] Bengt Jonsson & Joachim Parrow, editors (1994): *CONCUR '94, Concurrency Theory, 5th International Conference, Uppsala, Sweden, August 22-25, 1994, Proceedings*. *Lecture Notes in Computer Science* 836, Springer.

[16] R. Milner (1989): *Communication and Concurrency*. Prentice-Hall International, Englewood Cliffs.

[17] D. Park (1981): *Concurrency and automata on infinite sequences*. In P. Deussen, editor: *5th GI Conference,* Karlsruhe, Germany, *Lecture Notes in Computer Science* 104, Springer-Verlag, pp. 167–183, doi:`10.1007/BFb0017309`.

[18] Irek Ulidowski (2000): *Finite axiom systems for testing preorder and De Simone process languages*. *Theor. Comput. Sci.* 239(1), pp. 97–139, doi:`10.1016/S0304-3975(99)00214-5`.

[19] Chris Verhoef (1995): *A Congruence Theorem for Structured Operational Semantics with Predicates and Negative Premises*. *Nordic Journal on Computing* 2(2), pp. 274–302.