Saturated Semantics for Reactive Systems *

Filippo Bonchi
University of Pisa

Barbara König
University of Duisburg-Essen

Ugo Montanari
University of Pisa

Abstract

The semantics of process calculi has traditionally been specified by labelled transition systems (LTS), but with the development of name calculi it turned out that reaction rules (i.e., unlabelled transition rules) are often more natural. This leads to the question of how behavioural equivalences (bisimilarity, trace equivalence, etc.) defined for LTS can be transferred to unlabelled transition systems. Recently, in order to answer this question, several proposals have been made with the aim of automatically deriving an LTS from reaction rules in such a way that the resulting equivalences are congruences. Furthermore these equivalences should agree with the standard semantics, whenever one exists.

In this paper we propose saturated semantics, based on a weaker notion of observation and orthogonal to all the previous proposals, and we demonstrate the appropriateness of our semantics by means of two examples: logic programming and a subset of the open π-calculus. Indeed, we prove that our equivalences are congruences and that they coincide with logical equivalence and open bisimilarity respectively, while equivalences studied in previous works are strictly finer.

1 Introduction

The operational semantics of process calculi is usually given in terms of transition systems labelled with actions, which, when visible, represent both observations and interactions with the external world. The abstract semantics is given in terms of behavioural equivalences, which depend on the action labels and on the amount of branching structure considered. Behavioural equivalences are often congruences with respect to the operations of the language, and this property expresses the compositionality of the abstract semantics.

A simpler approach, inspired by classical formalisms like λ-calculus, Petri nets, term and graph rewriting, and pioneered by the Chemical Abstract Machine [3], defines operational semantics by means of *structural axioms* and *reaction rules*. Process calculi representing complex systems, in particular those able to generate and communicate names, are often defined in this way, since structural axioms give a clear idea of the intended structure of the states while reaction rules, which are often non-conditional, give a direct account of the possible steps. Transitions caused by reaction rules, however, are not labelled, since they represent evolutions of the system without interactions with the external world. Thus reduction semantics in itself is neither abstract nor compositional. To enhance the expressiveness of reduction semantics, Leifer and Milner proposed in [12] a systematic method for deriving bisimulation congruences from reduction rules. The main idea is the following: a process p can do a move with label $C[-]$ and become p' iff $C[p] \rightsquigarrow p'$. This definition was inspired by the work of Sewell [20]. Also, the approach of observing contexts imposed on agents at each step was introduced in [16], yielding the notion of *dynamic bisimilarity*.

Leifer and Milner introduced also the categorical notions of relative pushout (RPO) and idem relative pushout (IPO) in order to specify a/the minimal context that allows the state to react with a given rule. This construction leads to labelled transition systems (LTS) that use only contexts generated by IPOs, and not all contexts, as labels, and thus are smaller than in the latter case. Bisimilarity on this LTS is a congruence under rather restrictive conditions. A generalisation to reactive systems over G-categories has been proposed by Sassone and Sobociński [19, 18]. Recently other extensions to open systems and to weak semantics were introduced in [10] and in [4] respectively. The approach has been applied to bigraphs [14] and DPO graph rewriting [6].

The above constructions start from actionless reduction rules and have fundamental motivations in terms of minimality of basic definitions. However in most interactive systems some notion of observation is built in, and it is difficult to derive the corresponding semantics purely by using contexts, as testified by the lack of results where the ordinary semantics of a process description language is derived from reduction rules. For instance, Milner and Sangiorgi in [15] introduced the notion of *barbed bisimulation*, where only reactions are considered, but where states

*Research partially supported by the DFG project SANDS, the IST 2004-16004 SENSORIA, and the MIUR PRIN 2005015824 ART.

are labelled by barbs (potential interactions with the environment). Even considering only labelled transitions, the RPO/IPO paradigm can be used to add relevant experiments to a transition system for which bisimilarity is not a congruence. In this line, Ferrari, Montanari and Tuosto in [8] considered a fragment of the π-calculus where name fusions are contexts and where IPO constructions add the transitions with the minimal fusions needed by the symbolic transition system [17] of the open π-calculus. But the resulting abstract semantics is strictly finer than open bisimilarity.

Another interesting interpretation of the RPO/IPO construction is in terms of models of computation tailored to the needs of the general server-to-client bindings required by the new web service applications. When a new service is discovered, not only the service must adapt to the client, e.g. accepting a list of parameters, but also the client must sometimes adapt to the server, in order to establish the connection. Moreover, the minimal possible adaptation should be sought, in order to minimise the possible degradation. Suitable modelling of the details of the negotiation may lead to formalisations able to take advantage of the semantic properties guaranteed by the RPO/IPO constructions. The above symmetrical server-client adaptation reminds us of the unification step of logic programming, where a goal and a clause adapt reciprocally in the most general way. Quite interestingly, in the observational view of logic programming [5] the label of a goal reduction is exactly the instantiation of the goal imposed by the unification step, as required by the RPO/IPO construction.

In this paper our aim is, as in the ordinary case, to derive a bisimilarity congruence from given reduction rules. However we introduce in the transition system *all* context-labelled transitions which make a state and a rule match. We call the resulting equivalences *saturated*. Saturated equivalences are coarser than ordinary ones and have nice properties, e.g., they are trivially congruences, but the LTS is infinite-branching in more cases. Here we develop a *semi-saturated* technique that allows to compute saturated equivalences without considering all matching contexts. In fact, if we call Alice the player choosing the move and Bob the player choosing a matching reply, we prove that if Alice chooses an IPO move and Bob replies with any matching move, the resulting equivalence is again the saturated one, even if the moves to be considered are usually much less. In order to apply this technique we require less restrictive conditions than for the ordinary equivalences: instead of requiring the existence of all redex RPOs we need only redex IPOs, i.e., we allow several local minima.

Indeed we show that in some relevant cases saturated equivalences are exactly what we want, while ordinary equivalences are too fine. In the paper we discuss two important cases: logic programming and π-calculus.

We model logic programming in a way similar to [5]. It turns out that saturated trace congruence coincides with the ordinary logic semantics of logic programming, while the ordinary trace congruence yields a finer semantics, know in the logic programming community as S-semantics [7]. Interestingly enough, a goal (i.e. a conjunction of atomic goals) and the head of a clause must adapt in two different ways: both must be instantiated, but in addition the head must be ($\wedge$-)composed with other formulas which stay idle in the reduction. We are able to obtain both adaptations at the same time within our approach, without resorting to an infinite number of rules, as it is usually the case for the ordinary construction, since agents are normally forced to be closed. In fact in our encoding we will have only one rule for each Horn clause. Several authors (see for instance [10]) consider the restriction to closed agents a big limitation of the label derivation approaches.

For the π-calculus we refer to the above mentioned paper [8], where the RPO/IPO approach yields the symbolic transition system of a fragment of the calculus. Again, while ordinary bisimilarity congruence yields a finer semantics, the saturated bisimilarity congruence yields the ordinary semantics of open π-calculus.

The main contribution of the paper is the appreciation of saturated equivalences (bisimilarity and trace). Saturated bisimilarity (in the sense of all contexts) was already known in the literature [12], but it was dismissed as not promising. In this paper we show an alternative definition which considers fewer contexts and we exhibit two important examples where saturated equivalences yield the most natural notions. Our alternative definition works under weaker conditions than those required in [12]. The construction proposed for logic programming is original and, in our opinion, particularly interesting because, at our knowledge, it is the only example in the literature of reactive systems, where the rules are both instantiated and contextualised.

Structure of the paper. In Section 2, we first review Leifer and Milner's theory of reactive systems, and then we recall some basic concepts of logic. In Section 3, we introduce the main theoretical contributions of the paper, and in Sections 4 and 5 we apply our results to logic programming and to open π-calculus. Throughout the paper, we will use as running example the reactive semantics of CCS [13] with the reaction rule $a.P \mid \overline{a}.Q \rightsquigarrow P \mid Q$.

2 Background

Reactive Systems. Here we summarise the theory of reactive systems proposed in [12] to derive labelled transition systems and bisimulation congruences from a given reaction semantics. The theory is centred on the concepts of *term*, *context* and *reaction rules*: contexts are arrows of a category, terms are arrows having as domain 0 (a special

object that denotes no holes), and reaction rules are pairs of terms.

Definition 1 (Reactive System)**.** *A* reactive system $\mathbb{C}$ *consists of:*

1. *a category* $\mathbf{C}$
2. *a distinguished object* $0 \in |\mathbf{C}|$
3. *a composition-reflecting subcategory* $\mathbf{D}$ *of* reactive contexts
4. *a set of pairs* $\mathcal{R} \subseteq \bigcup_{I \in |\mathbf{C}|} \mathbf{C}(0, I) \times \mathbf{C}(0, I)$ *of* reaction rules.

The reactive contexts are those in which a reaction can occur. By composition-reflecting we mean that $d; d' \in \mathbf{D}$ implies $d, d' \in \mathbf{D}$.

Note that the rules have to be ground, i.e., left-hand and right-hand sides have to be terms without holes and, moreover, with the same codomain. Having ground rules is a simplification often made, but there is some work which tries to overcome this constraint [10].

From reaction rules one generates the reaction relation by closing them under all reactive contexts. Formally the *reaction relation* is defined by taking $p \rightsquigarrow q$ if there is $\langle l, r \rangle \in \mathcal{R}$ and $d \in \mathbf{D}$ such that $p = l; d$ and $q = r; d$.

Thus the behaviour of a reactive system is expressed as an unlabelled transition system. On the other hand many useful behavioural equivalences are only defined for LTSs. In order to obtain an LTS, we can plug a term p into some context $C[-]$ and observe if a reaction occurs. In this case we have that $p \xrightarrow{C[-]}$. Categorically speaking this means that $p; C[-]$ matches $l; d$ for some rule $\langle l, r \rangle \in \mathcal{R}$ and some reactive context d. This situation is formally depicted by diagram (i) in Figure 1: a commuting diagram like this is called a *redex square*.

Definition 2 (context transition system)**.** *The* context transition system *(CTS for short) is defined as follows:*

- *states: arrows* $p : 0 \to I$ *in* $\mathbf{C}$*, for arbitrary* I*;*
- *transitions:* $p \xrightarrow{C[-]}_C q$ *iff* $C[p] \rightsquigarrow q$*.*

Note that this labelled transition system is often infinite-branching since all contexts that allow reactions may occur as labels. Another problem of *CTS* is that it has redundant transitions. For example, consider the term $a.0$ of CCS. The observer can put this term into the context $\overline{a}.0 \mid -$ and observe a reaction. This correspond to the transition $a.0 \xrightarrow{\overline{a}.0|-}_C 0|0$. However we also have $a.0 \xrightarrow{p|\overline{a}.0|-}_C p \mid 0 \mid 0$ as a transition, yet p does not contribute to the reaction. Hence we need a notion of "minimal context that allows a reaction". Leifer and Milner define idem pushouts (IPOs) to capture this notion.

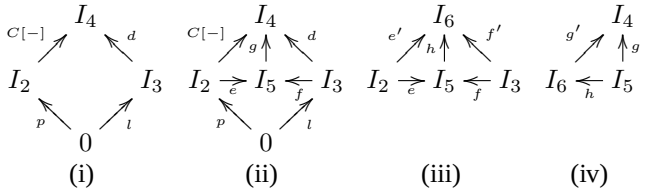


Figure 1. Redex Square and RPO

Definition 3 (RPO)**.** *Let the diagrams in Figure 1 be in some category* $\mathbf{C}$*. Let (i) be a commuting diagram. Any tuple* $\langle I_5, e, f, g \rangle$ *which makes (ii) commute is called a* candidate *for (i). A* relative pushout (RPO) *is the smallest such candidate. More formally, it satisfies the universal property that given any other candidate* $\langle I_6, e', f', g' \rangle$*, there exists a unique mediating morphism* $h : I_5 \to I_6$ *such that (iii) and (iv) commute.*

Definition 4 (IPO)**.** *A commuting square such as diagram (i) of Figure 1 is called* idem pushout (IPO) *if* $\langle I_4, c, d, id_{I_4} \rangle$ *is its RPO.*

Definition 5 (redex RPOs)**.** *A reactive system* has redex RPOs *if every redex square has an RPO.*

Definition 6 (IPO-Labelled Transition System)**.** *The* IPO-labelled transition system *(ILTS for short) is defined as follows:*

- *states:* $p : 0 \to I$ *in* $\mathbf{C}$*, for arbitrary* I*;*
- *transitions:* $p \xrightarrow{C[-]}_I r; d$ *iff* $d \in \mathbf{D}$*,* $\langle l, r \rangle \in \mathcal{R}$ *and the diagram (i) in Figure 1 is an IPO.*

In other words, if inserting p into the context $C[-]$ matches $l; d$, and $C[-]$ is the "smallest" such context (according to the IPO condition), then p transforms to $r; d$ with label $C[-]$, where r is the reduct of l.

Bisimilarity on *ILTS* is referred to as *standard bisimilarity* (denoted by $\sim_{IPO}$), and Leifer and Milner have shown that if the reactive system has redex RPOs, then it is a congruence (i.e., it is preserved under all contexts).

It can be easily shown that bisimilarity over *CTS* is a congruence as well. In this paper we will focus on this bisimilarity, which will be called *saturated bisimilarity* (denoted by $\sim_{SAT}$). In [12], it is referred to as $\sim_4$, and the authors show that $\sim_{IPO} \subseteq \sim_{SAT}$.

Logic Programming. As an application domain for saturated semantics we will now introduce logic programming and semantic equivalences of logic formulas.

A *logic signature* Γ is a pair (Σ, Π), where Σ is a set of *function symbols* and Π is a set of *predicate symbols* with an associated arity. As usual, given a set X of variables,

we denote by $T_\Sigma(X)$ the free Σ-algebra over X. A *term* over X is an element of $T_\Sigma(X)$. Given a term t, $Var(t)$ is the smallest set of variables X such that $t \in T_\Sigma(X)$. An *atomic formula* over X has the form $P(t_1, \ldots, t_n)$ where P is a predicate with arity n, and $t_1, \ldots, t_n$ are terms over X. A *formula* is a finite conjunction of atomic formulas: $a_1 \wedge \cdots \wedge a_n$ where $\wedge$ is associative and it has the *empty formula* $\Box$ as unit. Note that in the standard definition $\wedge$ is also commutative, but to simplify our construction, as it is the case in Prolog, we do not consider it to be commutative (however the resulting behaviour is the same).

If X and Y are sets of variables, a *substitution* from X to Y is a function $\sigma : X \to T_\Sigma(Y)$. If t is a term over X and σ a substitution from X to Y, then the term over Y, obtained by simultaneously substituting in t all the occurrences of the variables in X with their image under σ, is called the application of σ to t and written $t;\sigma$ (or $\sigma(t)$). If σ is a substitution from X to Y, and σ' from Y to Z, then $\sigma;\sigma'$ from X to Z is defined by applying σ' to each image of the variables in X under σ. Given $\sigma : X \to T_\Sigma(Y)$ and $X' \subseteq X$ the *restriction* of σ to X', written $\sigma \restriction X'$, is the substitution $\sigma' : X' \to T_\Sigma(Y)$ acting as σ on X'.

A substitution σ is *more general* than σ' if there exists a substitution θ such that $\sigma' = \sigma;\theta$. Two substitutions ψ and ϕ *unify* if there exists a substitution σ such that $\psi;\sigma = \phi;\sigma$, in this case σ is a *unifier* of ψ and ϕ. It is well-known that if ψ and ϕ unify, then there exists a unifier that is more general than all the others, called the *most general unifier* (*mgu* for short). It is also well-known that an *mgu* is the coequalizer in the category of substitutions [9], and in [5] it is shown that the *mgu* of substitutions with disjoint sets of variables corresponds to a pushout (this will be detailed later).

A *logic program* is a finite collection of *Horn clauses*, i.e., expressions of the form $h :\!- b$ where h is an atomic formula called the *head* of a clause, and b is a formula called the *body*. Rules in Table 1 define the operational semantics of logic programming. A goal $g = a_1 \wedge \cdots \wedge a_n$ reacts with a clause $c = h :\!- b$ if a_i, an atomic formula of the goal g, unifies with $\rho(h)$ (where ρ substitutes the variables of h with fresh variables not appearing in g). Let σ be the *mgu* of a_i and $\rho(h)$, then g reacts and becomes $g' = \sigma(a_1) \wedge \cdots \wedge \sigma(a_{i-1}) \wedge \sigma(b) \wedge \sigma(a_{i+1}) \wedge \cdots \wedge \sigma(a_n)$. A *refutation* of g is a derivation $g \Rightarrow_{\sigma_1} g_2 \Rightarrow_{\sigma_2} \cdots \Rightarrow_{\sigma_n} g_n$ ending with the empty formula (i.e. $g_n = \Box$). In this case $\sigma = \sigma_1; \ldots; \sigma_n \restriction Var(g)$ is a *computed answer substitution* of g.

Now, given a logic program, when are two goals equivalent? First note that we already have an LTS, but bisimulation is quite uninteresting in this case because we would like to consider as equivalent two goals with different branching behaviour. Here the interesting point is if, and when, two goals can be refuted. The first naive equivalence that comes to mind is: g_1 can be refuted iff g_2 can be refuted. This equivalence is however not a congruence.

$$\frac{h :\!- b \in P \;\; \sigma = \mathit{mgu}(a, \rho(h))}{P \Vdash a \Rightarrow_\sigma \sigma(\rho(b))}$$

where ρ renames to globally fresh names

$$\frac{P \Vdash g \Rightarrow_\sigma f}{P \Vdash g_1 \wedge g \wedge g_2 \Rightarrow_\sigma \sigma(g_1) \wedge f \wedge \sigma(g_2)}$$

Table 1. Operational rules for SLD-resolution

Logic equivalence (denoted by $\simeq_L$) equates g_1 and g_2 if and only if, for any ground substitution σ, $\sigma(g_1)$ is refuted iff $\sigma(g_2)$ is refuted. In [7], *S-equivalence* (denoted by $\simeq_S$) is proposed: g_1 and g_2 have the same set of computed answer substitutions. Another interesting equivalence is *correct answer equivalence* (denoted by $\simeq_C$) that equates two goals iff they have the same set of correct answer substitutions (defined as follows). Let $\xrightarrow{\sigma}$ be the transition system defined by changing the premise of the first rule of Table 1: we do not require anymore that σ is the mgu, but only that it unifies a and $\rho(h)$ i.e, $\sigma(a) = \sigma(\rho(h))$. If $g \xrightarrow{\sigma_1} g_2 \xrightarrow{\sigma_2} \ldots \xrightarrow{\sigma_n} \Box$ we say that $\sigma = \sigma_1; \ldots; \sigma_n \restriction Var(g)$ is a *correct answer substitution* of g. In other words σ is a correct answer substitution of g iff $\sigma(g)$ is a logical consequence of the program.

In [5], it is shown that, if we work with an infinite set of function symbols, $g_1 \simeq_L g_2$ iff $g_1 \simeq_C g_2$.

The following example shows that S-equivalence is somehow too detailed and that logic equivalence is more abstract.

Example 1. *Consider the following program, where y is a variable and a is a constant:*

$$P(y) :\!- \Box \qquad P(a) :\!- \Box \qquad Q(y) :\!- \Box$$

Now consider the goals $P(x)$ and $Q(x)$. They are refuted by any ground substitution, which means that they are logic equivalent (and also correct answer equivalent). However, they are not S-equivalent: in fact the set of computed answer substitutions for $P(x)$ is $\{\epsilon, [a/x]\}$, while the computed answer substitutions for $Q(x)$ are $\{\epsilon\}$.

3 Saturated Semantics

In Section 2 we have shown that given a reactive system one can define two LTSs: the *CTS*, where the labels are all contexts allowing a reaction, and the *ILTS*, where labels are the minimal contexts allowing a reaction. On those LTSs we can define various kinds of equivalences, such as bisimilarity, trace and failure equivalence. The term *saturated semantics* stands for equivalences defined on the *CTS*, while *standard semantics* stands for equivalences defined on the *ILTS*.

Theorem 1. *Saturated bisimilarity is the coarsest bisimulation on $\rightsquigarrow$ that is also a congruence.*

In our opinion, the standard semantics (using IPOs as labels) is not really observational since the observer has to know exactly the right amount of information that the process needs to react, while saturated semantics are truly observational: the observer plugs the process into some context and observes if a reaction occurs. However, with the current definition it is hard to show that two systems are saturated bisimilar, since *CTS* is often infinite-branching and bisimilarity must consider all possible moves.

3.1 Semi-Saturated Bisimulation

Here we propose an alternative and, (in some cases) finitary characterisation of saturated bisimilarity: in the bisimulation game, one player proposes an IPO transition and the other answers with a contextual transition.

Definition 7 (semi-saturated bisimulation)**.** *A symmetric relation R is a* semi-saturated bisimulation *if whenever $p\,R\,q$, then $p \xrightarrow{C[-]}_I p'$ implies $q \xrightarrow{C[-]}_C q'$ and $p'\,R\,q'$.*

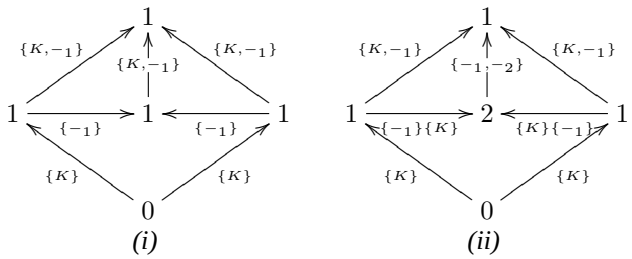
We call the union of all semi-saturated bisimulations semi-saturated bisimilarity *(denoted by $\sim_{SS}$).*

Theorem 2 states that under very weak conditions this kind of bisimilarity coincides with saturated bisimilarity (and thus it is a congruence). In this way we can prove that two processes are saturated bisimilar just starting with IPO moves that are sometimes (see, e.g., Corollary 1) finite in number. Once an IPO move is chosen, the context $C[-]$ is fixed, and thus only the $\rightsquigarrow$ moves from $C[q]$ must be considered. Milner and Leifer have shown that $\sim_{IPO}$ is a congruence if the reactive system has redex RPOs, i.e., if for each redex there exist an RPO. For $\sim_{SS}$ it is sufficient to require that the reactive system has redex IPOs.

Definition 8 (redex IPOs)**.** *A reactive system* has redex IPOs, *if every redex square has at least one IPO as candidate.*

Clearly this constraint is weaker than having redex RPOs, and hence our results can be applied to a larger number of reactive systems. Having RPOs means to have a minimum candidate (i.e., a candidate smaller than all the others), while having IPOs allows to have several minimal candidates (also not comparable among them). The following example (introduced in [21] and inspired by [12]) exemplifies the difference between redex IPOs and redex RPOs.

Example 2 (Abstract Bunch Contexts)**.** *An abstract bunch context is a string of multisets containing elements from some alphabet $\mathcal{K}$ and places (i.e., holes). Abstract bunch contexts form a category having natural numbers as objects and abstract bunch contexts of length n having m holes as arrows $m \to n$. Composition of $a : m \to n$ and $b : n \to o$ is defined by plugging the n multiset of a into the n holes of b. Finally, the identity id_n is $\{-_1\}\{-_2\}\ldots\{-_n\}$.*
This category does not have RPOs: consider the exterior squares in diagrams (i) and (ii) below (note that they are equal). This square has no RPOs since it has as candidates the arrows inside which are not comparable (in the sense that neither is smaller than the other). But note that both are IPOs, since they have as candidates only isomorphic diagrams.



Theorem 2. *In a reactive system having redex-IPOs, semi-saturated bisimilarity coincides with saturated bisimilarity (i.e., $p \sim_{SS} q \iff p \sim_{SAT} q$).*

Theorem 3. *In a reactive system having redex-IPOs, a symmetric relation R is a semi-saturated bisimulation iff whenever $p\,R\,q$, then $p \xrightarrow{c}_I p'$ implies the existence of d, e such that $d; e = c$, $q \xrightarrow{d}_I q'$ and $p'\,R\,q'; e$.*

Theorem 3 offers another characterisation of semi-saturated bisimilarity (and thus of saturated bisimilarity) that resembles open [17], asynchronous [1] and large [2] bisimilarity.

3.2 Saturated Trace Equivalences

Besides bisimulation, many other equivalences have been defined on LTSs. Here we introduce ϕ-trace equivalence, a quite general equivalence, parametric with respect to a property ϕ, that generalises trace and S-equivalence of logic programming. This equivalence can be instantiated both on the IPO and on the contextual LTS and, as we did for bisimulation, we define a semi-saturated version of it and we show that it corresponds to saturated equivalence.

Definition 9 (ϕ-trace equivalence)**.** *Let X be a set of states, L a set of labels and $\rightarrow \subseteq X \times L \times X$ a transition relation. Let $-;- : L \times L \to L$ be an associative operator on labels and let ϕ be a property on X. We say that $p, q \in X$ are ϕ-trace equivalent ($p \simeq^{\phi} q$) if the following conditions hold:*

- *$\phi(p)$ if and only if $\phi(q)$,*
- *if $p \overset{l}{\twoheadrightarrow} p' \wedge \phi(p')$ then $q \overset{l}{\twoheadrightarrow} q' \wedge \phi(q')$,*
- *if $q \overset{l}{\twoheadrightarrow} q' \wedge \phi(q')$ then $p \overset{l}{\twoheadrightarrow} p' \wedge \phi(p')$,*

where $p \overset{l}{\twoheadrightarrow} p'$ *iff* $p \xrightarrow{l_1} p_2 \ldots p_n \xrightarrow{l_n} p'$ *and* $l = l_1; l_2; \ldots; l_n$ *with* $n \geq 1$.

Note that if ϕ holds in every state of X and ; is string concatenation, then we obtain the classical trace semantics for $\rightarrow$, while if ϕ holds just for the empty goal $\Box$, $\rightarrow$ is the SLD transition relation and if ; is composition of substitutions, then we obtain S-equivalence of logic programming.

In the rest of this section we will study this equivalence in the setting of reactive systems, and we will fix the ; operator to be context composition. As we did for bisimilarity, we can define this equivalence on the *ILTS* (*standard ϕ-trace equivalence* denoted by $\simeq^{\phi}_{I}$) or on the *CTS* (*saturated ϕ-trace equivalence* denoted by $\simeq^{\phi}_{SAT}$).

In order to obtain a congruence we have to require the following conditions:

1. ϕ is defined on all arrows, and the arrows satisfying ϕ form a composition-reflecting subcategory;
2. all contexts are reactive.

The first requirement is not very strong, and we will show that in our encoding of logic programming, setting $\phi(a) \Leftrightarrow a = \Box$ defines a composition-reflecting sub-category. The second constraint is rather restrictive, but there are many formalisms for which it holds, as for example DPO graph rewriting or logic programming.

Theorem 4. *In a reactive system where all contexts are reactive $\simeq^{\phi}_{SAT}$ is a congruence.*

Standard bisimilarity is a congruence under the constraint of having all redex RPOs, while here standard ϕ-trace equivalence is a congruence under the assumption that RPOs exist not only for redex squares but also for squares where the four arrows are contexts (in reactive systems RPOs are only required for squares where one of the lower arrows is a redex). We say that a reactive system has *redex and context RPOs* if it satisfies this constraint. We have to require this condition since we are working with the transitive closure of $\rightarrow_I$. A similar condition is needed in [4] where the authors require to have all RPOs, in order to show that weak bisimulation is a congruence.

Theorem 5. *In a reactive system with redex and context RPOs, where all contexts are reactive and ϕ defines a composition-reflecting subcategory, $\simeq^{\phi}_{I}$ is a congruence.*

As for bisimulation we can define a semi-saturated version of ϕ-trace equivalence.

Definition 10. *Let $\mathbb{C}$ be a reactive system, and ϕ a property on the arrows of* $\mathbf{C}$*. We say that p and q are* semi-saturated ϕ-trace *equivalent ($p \simeq^{\phi}_{SS} q$) if the following holds:*

- *if* $\phi(p)$ *then* $q \overset{l}{\twoheadrightarrow}_I q' \wedge \phi(q')$ *and* $\phi(q')$,
- *if* $p \overset{l}{\twoheadrightarrow}_I p' \wedge \phi(p')$ *then* $q \overset{l}{\twoheadrightarrow}_C q'$ *and* $\phi(q')$,
- *if* $q \overset{l}{\twoheadrightarrow}_I q' \wedge \phi(q')$ *then* $p \overset{l}{\twoheadrightarrow}_C p'$ *and* $\phi(p')$,

where $\twoheadrightarrow_I$ and $\twoheadrightarrow_C$ are the transitive closures of $\longrightarrow_I$ and $\longrightarrow_C$.

As semi-saturated bisimilarity corresponds to saturated bisimilarity, semi-saturated ϕ-trace equivalence is saturated ϕ-trace equivalence, under the weak constraint of the existence of redex IPOs.

Theorem 6. *In a reactive system with redex IPOs, where all contexts are reactive, and such that ϕ defines a composition-reflecting subcategory, then $\simeq^{\phi}_{SS} = \simeq^{\phi}_{SAT}$.*

4 Logic Programs as Reactive Systems

In this section we will show how logic programs can be seen as reactive systems and how the theory developed above can be applied in this framework. Consider two basic sorts $\mathtt{t}$ for terms and $\mathtt{p}$ for formulas (predicates are atomic formulas). We use ϵ to denote the empty string and $\mathtt{t}^n$ to denote the string composed of n occurrences of $\mathtt{t}$. Given a logic signature $\Gamma = (\Sigma, \Pi)$, we define Γ' as the signature Γ enriched with the symbols $\wedge$ that takes two formulas and returns one formula and $\Box$ a constant formula. Let E be the set of axioms describing that $\wedge$ is associative (not commutative) and has identity $\Box$. Let X_p and X_t be sets of predicate and term variables. We use $T_{\Gamma'/E}(X_p, X_t)$ to denote the Γ'-algebra freely generated by (X_p, X_t) quotiented by E. A term of this algebra in sort $\mathtt{p}$ is a logic formula having term and predicate variables from X_t and X_p.

Definition 11. *The category* $\mathrm{Th}[\Gamma'/E]$ *is the free* algebraic theory *[11] associated to the specification* Γ', E.

This category has been used in [5] as base category for a tile system for logic programming. Usually *algebraic theories* are applied to a one sorted signature and the resulting category has natural numbers as objects, while here it is applied to a two sorted signature and it has strings of sorts (i.e., elements of $\{\mathtt{t}, \mathtt{p}\}^*$) as objects. For example, an object $\mathtt{p}^n\mathtt{t}^m$ can be thought of as representing n ordered *canonical predicate variables* (i.e., variables indexed from 1 to n) $p_1, \ldots, p_n$ and m ordered *canonical term variables* $x_1, \ldots, x_m$. To avoid confusion, it must be clear that the canonical variables are just placeholders, i.e., their scope is only local. The arrows from s_1 to s_2 are s_1-tuples of elements of $T_{\Gamma'/E}$ with s_2 canonical variables and the composition of arrows is term substitution.

The subcategory of the arrows of the form $\mathtt{t}^n \rightarrow \mathtt{t}^m$ is isomorphic to the category of finite substitutions on Σ (with canonical sets of variables) and the arrows $\mathtt{t} \rightarrow \epsilon$ are closed term over Σ, while arrows $\mathtt{p} \rightarrow \epsilon$ are closed

formulas over Γ'. Arrows $\mathsf{p} \to \mathsf{t}^n$ are formulas over n canonical term variables, while arrows $\mathsf{p} \to \mathsf{pt}^n\mathsf{p}$ are formulas over n canonical term variables and two canonical predicate variables. Consider for example $\langle P(x_1, x_2) \wedge p_1, f(x_1), Q(f(x_2)), p_5\rangle$ where x_1, x_2 are terms variables and p_1, p_5 are predicate variables. This tuple corresponds to an arrow from ptp^2 to $\mathsf{t}^2\mathsf{p}^5$. Note also that the above tuple can represent also an arrow from ptp^2 to tptp^4.

Furthermore the above tuple can be seen as an arrow having as codomain objects $\mathsf{t}^n\mathsf{p}^m$ for $n \geq 2$ and $m \geq 5$, i.e. the codomain does not define the exact index of (term or predicate) variables, but the maximum index that the variables can have. In the following for a goal g and a natural number n larger than the maximal index of variables appearing in g, we will write g^n to denote the arrow $\mathsf{p} \to \mathsf{t}^n$.

In the classical interpretation by Leifer and Milner, the arrows having domain objects different from 0 (the distinguished object) are seen as contexts which can be precomposed with terms. In our reactive system these arrows are substitutions which instantiate the variables of formulas. Horn clauses, not only must be instantiated by substitutions, but they must be also contextualised with the $\wedge$ operator.

In the rest of this section we will use the formula $f_1 = P(s(x_1), x_2) \wedge P(x_1, t(x_3))$ and the clause $c_1 = P(y_1, t(y_2)) \ :- Q(y_1)$ as running example. The head of the c_1 must be instantiated (e.g., substituting y_1 with x_1 and y_2 with x_3) and contextualised (plugging it into $P(s(x_1), x_2) \wedge [-]$) in order to match f_1.

Similar problems arise with process calculi where the rules usually are not ground, and have to be instantiated and contextualised. For example, the redex of the CCS rule $a.P \mid \overline{a}.Q \rightsquigarrow P \mid Q$ matches $\nu a.(a.0 \mid \overline{a}.0)$ instantiating P, Q to 0 and plugging the left-hand side into the context $\nu a.[-]$. Usually this problem is avoided by creating infinitely many rules corresponding to all possible instantiations of the rule, and then considering only contextualisation, as it is done for bigraphs [14]. This approach causes the problem of having infinitely many rules and consequently infinitely many transitions. In [10] the notion of open reactive systems is developed in order to overcome this problem, but the resulting theory is quite restrictive. Here we propose a different approach: we simulate contextualisation by substitutions by supplying appropriate variables in the rules. The redex of a rule is not simply an arrow of the form $h : \mathsf{p} \to \mathsf{t}^n$ that can only be instantiated, but it is an arrow $p_1 \wedge h \wedge p_2 : \mathsf{p} \to \mathsf{pt}^n\mathsf{p}$ that can be instantiated and contextualised (by instantiating the variables p_1 and p_2). In this way, we also get a finite branching *ILTS*.

Thus, in our reactive system, the head of the clause c_1 above becomes $p_1 \wedge P(y_1, t(y_2)) \wedge p_2$ and, in this way, the head can match the goal instantiating p_1 to $P(s(x_1), x_2)$, p_2 to $\Box$ and y_1 to x_1 and y_2 to x_3.

Summarizing, we can say that we allow only substitutions and simulate contextualisations by substitutions by supplying appropriate variables in the rules (see below). In order to integrate this idea with the theory of reactive systems we have "reversed" the arrows, i.e., a formula over n term variables becomes $\mathsf{p} \to \mathsf{t}^n$ (instead of the maybe more intuitive $\mathsf{t}^n \to \mathsf{p}$).

Definition 12. *Given a logic program P on a signature Γ, we define a reactive system $R(P)$ as follows:*

- $\mathrm{Th}[\Gamma'/E]$ *is the underlying category*
- p *is the distinguished object*
- *all contexts are reactive*
- *for each clause $h \ :- b$, let n be the largest index of variables contained in h and b; then we add the rule*

$$(p_1 \wedge h \wedge p_2 \ , \ p_1 \wedge b \wedge p_2)$$

where left and right-hand sides are arrows $\mathsf{p} \to \mathsf{pt}^n\mathsf{p}$ and p_1, p_2 are predicate variables.

Note that h and b do not necessarily have the same number of variables, while our theory requires that left-hand and right-hand side of a rule have the same interface (i.e., they must be arrows with the same target). In this case we extend the smaller interface.

A generic redex square of this reactive system is depicted in diagram (i) of Figure 2. Arrow c is a substitution that instantiates the variables of g, while arrow d instantiates the variables of h and contextualises h, instantiating the predicate variables p_1 and p_2. Thus for any reaction step an atom of the goal is unified with the head of a clause and p_1 is instantiated with the formula on the left of the chosen atom, and p_2 is instantiated with the formula on the right.

Lemma 1. *The exterior square of diagram (i) in Figure 2 commutes if and only if there exist formulas g_1, g_2 and an atomic formula a such that $g = g_1 \wedge a \wedge g_2$, $p_1; d = g_1; c$, $p_2; d = g_2; c$ and $h; d = a; c$.*

In general, in $R(P)$, given a rule and a goal, there exist several ways of unifying them: one for each atom of the goal that can match the head h. Consider for example the redex of c_1 and the goal f_1. The head of c_1 unifies both with the left predicate of f_1 and with the right one. This means that, given a redex and a goal—seen as arrows—there usually exists no a minimal way of matching them (i.e., no pushout exists). The following lemma assures that each commuting square fixes a "way" of matching, i.e., chooses the atom of the goal that unifies h.

Lemma 2. *Let the exterior square in diagram (i) of Figure 2 be commuting. Let g_1, a, g_2 be formulas as described in Lemma 1. Then for each candidate $\langle e, f, i\rangle$, the following hold: $p_1; f = g_1; e$, $p_2; f = g_2; e$ and $h; f = a; e$.*

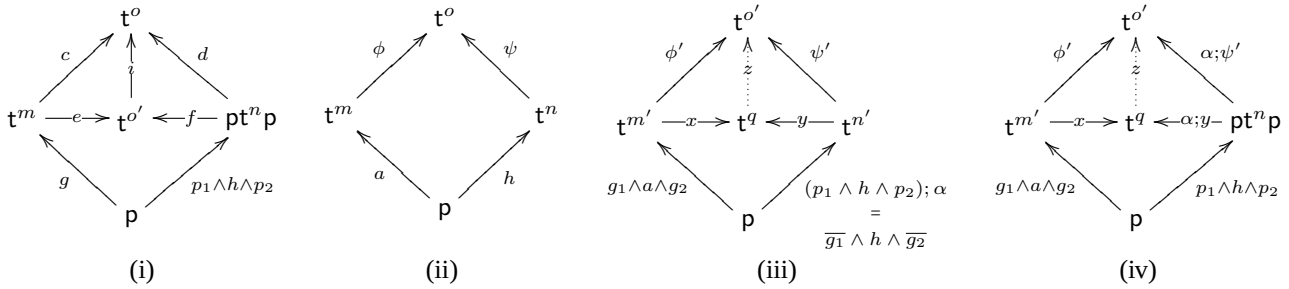


Figure 2. Redex squares, pushouts and RPOs in a reactive system $R(P)$

As a next step we are going to show that in our reactive system a redex RPO is the *mgu* of a and h, together with the instantiation of p_1 and p_2 to appropriate formulas. We start by recalling a theorem from [5].

Theorem 7. *Given two substitutions of terms a and b with disjoint sets of variables, their mgu is the pushout of the arrows a^m and b^n, for m, n larger than the maximal index of variables of a and b.*

Remember that if two substitutions can unify, then there exists an *mgu*. This, together with Theorem 7, assures that for each commuting square of substitutions there exists a pushout. Moreover this result holds not only for substitutions but also for atomic goals since two atomic goals unify iff they consist of the same predicate and the terms within the predicate unify. In the rest of the paper we use $\overline{g}$ to denote a formula having the same predicate symbols as g, but without function symbols and where all variables are different. For example $\overline{f_1} = P(u_1, u_2) \wedge P(u_3, u_4)$. Note that the arrow d of a generic redex square (see Figure 2(i)) can always be decomposed into $\alpha; \psi'$ where α instantiates p_1 and p_2 to $\overline{g_1}$ and $\overline{g_2}$ and ψ' is a substitution. It is exactly this arrow α that chooses which atom of the goal matches h.

The following lemma generalises the theorem above to non-atomic formulas of the form $g_1 \wedge a \wedge g_2$ and $\overline{g_1} \wedge b \wedge \overline{g_2}$.

Lemma 3. *Let a and h be atomic formulas. In Figure 2 $\langle \phi, \psi \rangle$ is the pushout of a and h (depicted in diagram (ii)) if and only if $\langle \phi', \psi' \rangle$ is the pushout of $g_1 \wedge a \wedge g_2$ and $\overline{g_1} \wedge h \wedge \overline{g_2}$ (see diagram (iii)), where ϕ' is equal to ϕ on $Var(a)$ and the identity on the others variables, and ψ' is equal to ψ on $Var(h)$ and such that $g_1; \phi = \overline{g_1}; \psi$ and $g_2; \phi = \overline{g_2}; \psi$.*

The meaning of this lemma is more intuitive if one considers formulas. Suppose that a and h unify, and let $\langle \phi, \psi \rangle$ be their *mgu*. Then also $g_1 \wedge a \wedge g_2$ and $\overline{g_1} \wedge h \wedge \overline{g_2}$ unify and the *mgu* is the *mgu* of a and h (since all the variables of $\overline{g_1}$ and $\overline{g_2}$ are different and can be instantiated to $g_1; \phi$ and $g_2; \psi$).

The following lemma is central since it shows the relationship between RPOs and pushouts: if we fix a way of matching (the arrow α), then we have only one minimal unifier (i.e, pushout) while if we do not fix it, we have several minimal unifiers (i.e., RPOs) one for each way of matching (i.e., for each α).

Lemma 4. *Let a and h be atomic formulas, and α as described above i.e., such that $(p_1 \wedge h \wedge p_2); \alpha = \overline{g_1} \wedge h \wedge \overline{g_2}$. In Figure 2 $\langle x, y \rangle$ is the pushout of $g_1 \wedge a \wedge g_2$ and $\overline{g_1} \wedge h \wedge \overline{g_2}$, and z the mediating morphism (as depicted in diagram (iii)) iff $\langle x, \alpha; y, z \rangle$ is the RPO of the diagram (iv).*

Then, given a commuting square, this fixes a way of matching (i.e., one α) and so there exists a minimal unifier, that is the *mgu* between the head of a clause h and chosen atom a of the formula g.

Theorem 8. *$R(P)$ has redex and context RPOs.*

In the rest of this section we will show that S-equivalence corresponds to standard ϕ-trace equivalence, while correct answer equivalence corresponds to saturated ϕ-trace equivalence. We start by showing that $\rightarrow_C$ corresponds to $\rightarrow$ (as defined in Section 2) while $\rightarrow_I$ corresponds to $\Rightarrow$ (i.e., SLD transitions).

Theorem 9. *Let P be a logic program. Let f, g be two formulas and m, n larger than the maximal index of variables appearing in f and g. Furthermore let σ be a substitution, and let $\theta : \mathsf{t}^m \rightarrow \mathsf{t}^n$ be equal to σ on $Var(f)$ and id otherwise. Then:*

- *$P \Vdash f \xrightarrow{\sigma} g$ iff in $R(P)$ it holds that $f^m \xrightarrow{\theta}_C g^n$,*
- *$P \Vdash f \Rightarrow_\sigma g$ iff in $R(P)$ it holds that $f^m \xrightarrow{\theta}_I g^n$.*

Corollary 1. *In $R(P)$, the ILTS is finite-branching.*

Note that S-equivalence and correct answer equivalence are ϕ-trace equivalence where the predicate ϕ holds only for the empty goal. Formally we define the predicate $\Box()$ over all the arrows of the category $\mathrm{Th}[\Gamma'/E]$: $\Box(a)$ holds iff a is an arrow obtained by decomposing $\Box^n : \mathsf{p} \rightarrow \mathsf{t}^n$, where $\Box^n$ is $\Box : \mathsf{p} \rightarrow \epsilon$ with the interface extended with n extra term variables. Essentially $\Box()$ holds for all term substitutions and for empty formulas. The predicate $\Box()$ defines a composition reflecting subcategory and, since all contexts are reactive, we can apply our theoretical results to $\simeq_I^\Box$, $\simeq_{SAT}^\Box$ and $\simeq_{SS}^\Box$: these three equivalences are congruences (w.r.t. substitutions) and $\simeq_{SAT}^\Box = \simeq_{SS}^\Box$.

Now we show that the first corresponds to $\simeq_S$, while the second (and then also the third) correspond to $\simeq_C$ (that, in the case of infinitely many function symbols, is $\simeq_L$).

Theorem 10. $\simeq_S = \simeq_I^\Box$, $\quad \simeq_C = \simeq_{SAT}^\Box$.

5 Saturated Bisimilarity is Open Bisimilarity

In [8], a reactive system for a subset of the π-calculus is defined in order to study how to model symbolic semantics via reactive systems. The reactive system constructed there is rather complicated, and for this reason we do not fully report it here. Instead we focus on those aspects that relate saturated bisimilarity to *open bisimilarity*. The subset of the π-calculus considered there is the standard π-calculus without matching, τ-prefixes and restriction. The operational semantic is the symbolic LTS whose labels are either actions or fusions. An output $\bar{a}x$, and an input $b(y)$ can synchronise leading to a transition $\xrightarrow{a=b}$. If a and b are equal $a = b$ is the identity fusion denoted by ϵ. Note that also in the original paper introducing open bisimilarity [17], the theory is first developed for the calculus without restriction and distinctions to simplify the presentation. A totally ordered set of names $\{a_1, a_2, \dots\}$ is assumed. Briefly, the underlying category of the defined reactive system has the natural numbers (representing the free names of a process) plus $\star$ as objects. A π-process p is represented as an arrow $p_m : \star \to m$ where $m \geq max\{k \mid a_k \in fn(p)\}$. The contexts in the category represent silent actions ($\tau^m : m \to m$), output actions ($\overline{a_i}^m a_j : m \to m$) and input actions ($a_i{}^m : m \to m+1$) and reaction rules are essentially transitions of the ordinary open π-calculus. When a rule is applied to a process, the IPO construction recreates a transition labeled exactly by the corresponding action, thus essentially embedding the LTS of the ordinary open π-calculus in the *ILTS*. However also fusions ($[a_i = a_j]_m : m \to m-1$) are possible contexts, and when a synchronization rule is selected for a process which has the input and the output actions on different channels, the IPO construction generates a fusion for them. As a consequence, the resulting *ILTS* is essentially the symbolic LTS of the open π calculus.

Lemma 5 (from [8])**.** *Let p be a process of our subset of π and $m \geq max\{k \mid a_k \in fn(p)\}$. Furthermore let $\to$ and $\to_I$ be the symbolic and the IPO transition relations. Then*

$$p \xrightarrow{\overline{a_h}a_k} p' \Leftrightarrow p_m \xrightarrow{\overline{a_h}^m a_k}_I p'_m,$$

$$p \xrightarrow{a_i(a_j)} p' \Leftrightarrow p_m \xrightarrow{a_i{}^m}_I p'_{m+1}, \quad p \xrightarrow{\epsilon} p' \Leftrightarrow p_m \xrightarrow{\tau^m}_I p'_m,$$

$$p \xrightarrow{a_i=a_j} p' \Leftrightarrow p_m \xrightarrow{\tau^m;[a_i=a_j]_m}_I p'_m; [a_i = a_j]_m.$$

In [8] it is shown that the reactive system has redex RPOs and hence the resulting equivalence $\sim_{IPO}$ is a congruence. However, this does not coincide with open bisimilarity but with *syntactical bisimilarity*, formally defined below.

Definition 13 (Open/Syntactical Bisimilarity)**.** *A symmetric relation R is an* open bisimulation *if whenever $p\,R\,q$ it holds:*

- *if $p \xrightarrow{\alpha} p'$ then $q \xrightarrow{\alpha} q'$ and $p'\,R\,q'$,*
- *if $p \xrightarrow{a=b} p'$ then ($q \xrightarrow{a=b} q' \vee q \xrightarrow{\epsilon} q'$) and $\sigma(p')\,R\,\sigma(q')$.*

where α is an input, an output or ϵ and σ is a fusion that fuses a to b. The union of all open bisimulation is open bisimilarity *(denoted by $\sim_O$).*

Syntactical bisimilarity *(denoted by $\sim_{SYN}$) is obtained by replacing the last condition of open bisimulation with:*

- *if $p \xrightarrow{a=b} p'$ then $q \xrightarrow{a=b} q'$ and $\sigma(p')\,R\,\sigma(q')$.*

It is immediate to see that $\sim_{SYN} \subseteq \sim_O$ since the conditions for matching transitions for $\sim_O$ are weaker than that the ones for $\sim_{SYN}$. The following example shows that $\sim_{SYN}$ is strictly finer.

Example 3. *Consider the following processes:*

- $p = (\bar{a}b \mid a'(c)) + (\bar{d}e \mid d(f))$
- $q = \bar{a}b.a'(c) + a'(c).\bar{a}b + (\bar{d}e \mid d(f))$

It holds that $p \sim_O q$ since the move $p \xrightarrow{a=a'}$ is matched by the (unique) synchronisation of q. However, $p \not\sim_{SYN} q$ since the transition $p \xrightarrow{a=a'}$ cannot be matched by q.

With Lemma 5 one can show that $\sim_{IPO}$ coincides with $\sim_{SYN}$ (see [8]), in fact in $\sim_{IPO}$ if Alice proposes a fusion moves, then Bob must answer with the same fusion, while in open bisimilarity Bob can answer with a less restrictive fusion. But this is exactly what happens with saturated bisimilarity. In fact look at the characterisation of semi-saturated bisimulation given by Theorem 3. If $p_m \xrightarrow{\tau^m;[a_i=a_j]_m}_I p'_m; [a_i = a_j]_m$, then q_m can answer with $q_m \xrightarrow{\tau^m;[a_i=a_j]_m}_I q'_m; [a_i = a_j]_m$ where $p'_m; [a_i = a_j]_m\,R\,q'_m; [a_i = a_j]_m$ (in this case arrow d of Theorem 3 is $\tau^m; [a_i = a_j]_m$ and $e = id$), or $q_m \xrightarrow{\tau^m}_I q'_m$ where $p'_m; [a_i = a_j]_m\,R\,q'_m; [a_i = a_j]_m$ ($d = \tau^m$, $e = [a_i = a_j]_m$).

Theorem 11. $\sim_O = \sim_{SAT}$.

6 Conclusions and Future Work

In this paper we have proposed a semi-saturated technique for efficiently characterising certain congruences that are usually coarser than those presented by Leifer and Milner in [12]. Our approach applies to different kinds of semantics (here we have handled bisimilarity and trace semantics, but we are confident that it applies to others). In this paper we have integrated semi-saturation within the IPO framework, but it could be applied also to G-reactive systems [18] and open reactive systems [10] where, in our opinion, it might help to relax the constraints of the theory. Another advantage of semi-saturation is that it can be

applied to a larger class of reactive systems, because we require only the existence of redex IPOs and not necessarily of redex RPOs.

Besides our examples, there are other cases where saturated bisimilarity seems to be appropriate. In $\sim_{IPO}$, if Alice proposes a fusion move, then Bob must answer with the same syntactic fusion, while in open bisimilarity Bob can answer with a "smaller label" (as it happens in saturated bisimilarity). We conjecture that the same can be said for *asynchronous bisimilarity* [1], since similarly to open bisimilarity, an input move of Alice can be matched with a τ move of Bob. Here we want to emphasize that the "shape" of asynchronous and open bisimulations is really similar to that of semi-saturated bisimulation as expressed by Theorem 3.

The question is still open of where saturated equivalences are appropriate. We have shown that for logic programming and symbolic open π-calculus they capture exactly the right congruences. However, when trying to derive a reasonable LTS semantics from a reduction semantics of process calculi, saturated bisimilarity seems to be too coarse. In fact let us consider two processes that always diverge i.e., such that for every reactive context into which they can be put, they can always react: they will always be saturated bisimilar, since they react in the same contexts. Consider for example the following CCS processes: $P = \tau.P$ and $Q = \tau.Q + a.P$. Putting them into any possible context, we will always get two processes that always diverge. In the standard CCS semantics these processes are definitely considered different. We are confident that a mixed approach, where some labeled transitions are present also in the initial reduction system, might be successful also for contextualizing process calculi. In fact this was already the case for dynamic bisimilarity [16] and for our symbolic π-calculus example. More interesting results could probably be obtained by minimizing the transition labels in the initial system, or by observing actions also in the states as for barbed bisimulation.

Another original contribution of this paper is the encoding of logic programs as reactive systems, where the IPO semantics correspond to S-equivalence while the saturated semantics corresponds to logical equivalence. The encoding of logic programs proposes a new way of handling non-ground rules in reactive systems: even within the theoretical framework proposed by Leifer and Milner we can use arrows that can both instantiate and contextualise the rules. In this way we can work with a finite number of rules and not with infinitely many as it happens, for example, with bigraphs. We conjecture that this approach can be extended to all contexts of the form $[-] \mid p$.

Acknowledgements We would like to thank Roberto Bruni, Andrea Corradini, Fabio Gadducci, Emilio Tuosto and the anonymous referees for their helpful comments.

References

[1] R. M. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous pi-calculus. In *Proc. of CONCUR '96*, volume 1119 of *LNCS*, 147–162. Springer, 1996.

[2] P. Baldan, A. Bracciali, and R. Bruni. Bisimulation by unification. In *Proc. of AMAST '02*, volume 2422 of *LNCS*, 254–270. Springer, 2002.

[3] G. Berry and G. Boudol. The chemical abstract machine. *Theor. Comput. Sci.*, 96:217–248, 1992.

[4] R. Bruni, F. Gadducci, U. Montanari, and P. Sobocinski. Deriving weak bisimulation congruences from reduction systems. In *Proc. of CONCUR '05*, volume 3653 of *LNCS*, 293–307. Springer, 2005.

[5] R. Bruni, U. Montanari, and F. Rossi. An interactive semantics of logic programming. *TPLP*, 1(6):647–690, 2001.

[6] H. Ehrig and B. König. Deriving bisimulation congruences in the DPO approach to graph rewriting. In *Proc. of FoSSaCS '05*, volume 2987 of *LNCS*, 151–166. Springer, 2004.

[7] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. Declarative modeling of the operational behavior of logic languages. 69(3):289–318, 1989.

[8] G. Ferrari, U. Montanari, and E. Tuosto. Model checking for nominal calculi. In *Proc. of FoSSaCS '05*, volume 3441 of *LNCS*, 1–24. Springer, 2005.

[9] J. Goguen. What is unification? A categorical view of substitution, equation and solution. In M. Nivat and H. Aït-Kaci, editors, *Resolution of Equations in Algebraic Structures*, 217–261. 1989.

[10] B. Klin, V. Sassone, and P. Sobocinski. Labels from reductions: Towards a general theory. In *Proc. of CALCO '05*, volume 3629 of *LNCS*, 30–50. Springer, 2005.

[11] F. Lawvere. Some algebraic problems in the context of functorial semantics of algebraic theories. In *Proc. of the Midwest Category Seminar II*, volume 61, 41–61, 1968.

[12] J. J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. In *Proc. of CONCUR '00*, volume 1877 of *LNCS*, 243–258. Springer, 2000.

[13] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

[14] R. Milner. Bigraphical reactive systems. In *Proc. of CONCUR '01*, volume 2154 of *LNCS*, 16–35. Springer, 2001.

[15] R. Milner and D. Sangiorgi. Barbed bisimulation. In *Proc. of ICALP '92*, volume 623 of *LNCS*, 685–695. Springer, 1992.

[16] U. Montanari and V. Sassone. Dynamic congruence vs. progressing bisimulation for ccs. *Fundam. Inform.*, 16(1):171–199, 1992.

[17] D. Sangiorgi. A theory of bisimulation for the pi-calculus. *Acta Inf.*, 33(1):69–97, 1996.

[18] V. Sassone and P. Sobocinski. Locating reaction with 2-categories. *Theor. Comput. Sci.*, 333(1-2):297–327, 2005.

[19] V. Sassone and P. Sobociński. Reactive systems over cospans. In *Proc. of LICS '05*, 311–320. IEEE, 2005.

[20] P. Sewell. From rewrite to bisimulation congruences. In *Proc. of CONCUR '98*, volume 1466 of *LNCS*, 269–284. Springer, 1998.

[21] P. Sobociński. *Deriving process congruences from reaction rules.* PhD thesis, 2004.