On the Complexity of Bisimilarity for Value-Passing Processes (Extended Abstract)

Michele Boreale[1] and Luca Trevisan[2]

[1] Istituto per l'Elaborazione dell'Informazione, Consiglio Nazionale delle Ricerche, Via S. Maria 46, 56126 Pisa, Email: `michele@dsi.uniroma1.it`
[2] Dipartimento di Scienze dell'Informazione, Università di Roma "La Sapienza", Via Salaria 113, 00198 Roma, Email: `trevisan@dsi.uniroma1.it`

Abstract. We study the complexity of deciding bisimilarity between non-deterministic processes with explicit primitives for manipulating data values. In particular, we consider a language with value-passing (input/output of data) and parametric definitions of processes. We distinguish the case in which data cannot be tested and the case in which a simple equality test over data is permitted.
In the first case, our main result shows that the problem is PSPACE-hard for the full calculus. In the second case, we first show that the problem is *co*NP-complete in the fragment with value-passing and no parametric definitions. We then define a compositional polynomial-time translation of the full calculus to the fragment with parametric definitions but no value-passing. The translation preserves bisimilarity. This fact establishes the decidability of the full calculus and the PSPACE-hardness of the fragment without value-passing. In other words, once parametric definitions and equality test are allowed, the adding of value-passing does not increase neither the expressive nor the computational power.

1 Introduction

Recently, there has been a renewed interest in process calculi with explicit primitives to manipulate data values. In particular, several enriched versions of Milner's CCS [Mil80, Mil89, JP93, HL95, HL93] have been studied. In *pure*, i.e. data-less, process calculi such as CCS, beside standard operators for describing behaviours of processes (such as non-determinism $+$ and parallel composition $|$) only pure-synchronization actions (also called "pure" actions) are provided. By contrast, process calculi with explicit treatment of data contain primitives for expressing transmission and receipt of values at communication ports: this feature is known as *value-passing*. Using the notation of [Mil80], output of v at port a is written $\overline{a}v.$, while input at a is written $a(x).$; here the variable x acts as a formal parameter. Besides being exchanged, usually data values can be used as *parameters* in recursively defined processes and tested by means of predicates to control the execution flow. Languages with explicit manipulation of values permit a natural description of realistic systems. As an example, the recursively defined process $C(x)$:

$$C(x) \quad \Leftarrow \quad [x < o](a(y).C(y) + \overline{b}x.C(x)) \; + \; [x \geq o]Error(x)$$

specifies a memory cell whose initial content is a number x; as long as this content is less than an overflow value o, the cell can either receive a new value at a, or transmit its content at b; as soon as the value x equals or exceeds o, a recovery process $Error$ is called.

A very peculiar kind of value-passing language is Milner, Parrow and Walker's π-*calculus* [MPW92], where the values being exchanged among processes are communication ports themselves (*name-passing*). This permits the description of systems with dynamical communication linkage.

When analysing concurrent systems, a central problem is to be able to decide whether two given descriptions (usually regarded as a specification and as an implementation) are equivalent or not, according to a chosen notion of equivalence (*verification*). The algebraic aspects of this problem are becoming now well-understood, also for value-passing processes [HL93, PS93, BD94]. On the contrary, a lot of questions concerning the decidability and the computational complexity of verification remain unanswered. A basic problem is to determine meaningful fragments of the calculi with values over which the verification problem is decidable. Then, a fundamental issue is to determine the abstract computational complexity of each of these fragments w.r.t. verification. Answering such questions would improve our understanding of the mathematical nature of processes. In practical cases, it could provide us with useful information to locate sources of inefficiency. In the present work, we will try to address some of these issues. We will restrict our attention to one of the most widely studied equivalences, Milner's bisimulation equivalence (or simply "bisimilarity"), written $\sim$ and described e.g. in [Mil89].

For processes manipulating values, a non-trivial aspect of the problem is that they have usually an operational description in terms of an infinite state-transition graph (they are *infinite state*), at least if the domain of data values is infinite. This is due to the fact that each input action $a(x)$. gives rise to infinitely many actual transitions, one for each different value. In [JP93], Jonsson and Parrow concentrate on a particular class of processes with values, the *data independent* ones, which cannot test data nor perform any kind of operation over them. They prove that the bisimilarity problem for such processes can be transformed into a bisimilarity problem for finite-state processes. For the latter, decision algorithms exist [PT87, KS90], which are polynomial in the sizes of the involved graphs (that can be however much larger than the syntactical size of the processes). A detailed comparison of our work with [JP93] is contained in Section 6.

In the present paper, we consider a calculus for describing non-deterministic processes. It should be naturally embedded in every "reasonable" language with explicit data manipulation. More precisely, besides permitting the execution of pure actions, we allow data values to be exchanged, used as parameters in recursive definitions and tested for equality. The latter is done via the *matching* predicate $[a = b]$, also considered in the π-calculus [MPW92]. This is perhaps the most elementary form of test one would admit on data. Not even negative tests, to decide inequality of data, are permitted.

Our goal is to classify and separate the computational complexity of the two basic operations for manipulating data, value-passing and parametric recursive definitions. This will be done both for the data-independent case (where matching is excluded) and for the data-dependent one (where matching is included).

In each of the two cases, we consider separately three (sub-)languages, obtained from the calculus with pure actions by adding either or both of value-passing and recursive definitions. Then we asses the decidability and the difference in complexity of these languages. In this analysis, we refer to the complexity classes P,NP, *co*NP and PSPACE. Recall that the latter contains both NP and *co*NP and that it is believed that this containment is strict (see e.g. [BC93]).

In the data-independent regime, we first note that the bisimilarity problem is solvable in polynomial time for the calculi allowing either, but not both, of recursive definitions or value-passing. Then we prove that the problem is PSPACE-hard for the full language, i.e.: every problem in the class PSPACE is not more difficult than the bisimilarity problem over the matching-free calculus. This improves on a NP-hardness result due to Jonsson and Parrow and is, to the best of authors' knowledge, the highest known lower-bound to the complexity of a decidable bisimilarity over a meaningful language.

In the data-dependent regime, we first show that, in the sublanguage with value-passing but no recursive definitions, the bisimilarity problem is decidable and is as difficult as the most difficult problems in *co*NP, i.e. it is *co*NP-complete. Then we analyze the complexity of the full language, with both value-passing and recursive definitions. We show that the language can be compositionally translated down to the fragment without value-passing, in a way that preserves bisimulation equivalence. The translation can be carried out in a time polynomial in the sizes of the processes. The result is interesting for two reasons. First, it gives us a procedure for deciding the bisimilarity problem in the full language, since the problem is easily seen to be decidable in the fragment without value-passing. Second, it ensures that the problem for the fragment without value-passing is just as complex as for the full language (which is of course PSPACE-hard). It is important to point out that the matching predicate plays a crucial role in the definition of the translation.

To sum up, in the absence of matching, value-passing and recursive definitions are separately tractable, but if we join them together the bisimilarity problem becomes very complex (PSPACE-hard). If matching is allowed, the presence of value-passing itself makes the problem *co*NP-complete. By contrast, the presence of recursive definitions themselves makes the problem PSPACE-hard; then, the adding of value-passing does not increase neither the expressive nor the computational power. These results are also summarized in Table 1.

The rest of the paper is organized as follows. In Section 2, syntax and semantics of the considered language are presented, and a few notions from complexity theory are recalled. Section 3 deals with the complexity of data-independent pro-

Language	Complexity	Reduces to ...
VP ($\mathcal{IL}_v$)	P	all
RD ($\mathcal{IL}_r$)	P	all
VP, RD ($\mathcal{IL}_{v,r}$)	PSPACE-hard	$\mathcal{L}_r$, $\mathcal{L}_{v,r}$
M, VP ($\mathcal{L}_v$)	*co*NP-complete	$\mathcal{IL}_{v,r}$, $\mathcal{L}_r$, $\mathcal{L}_{v,r}$
M, RD ($\mathcal{L}_r$)	PSPACE-hard	$\mathcal{L}_{v,r}$
M, VP, RD ($\mathcal{L}_{v,r}$)	PSPACE-hard	$\mathcal{L}_r$

VP = value-passing, RD = recursive definitions, M = matching.

Table 1. The complexity results of the paper.

cesses. As to data-dependent processes, the treatment of value-passing is contained in Section 4, while the relationship between the full language and the fragment without value-passing is investigated in Section 5. Comparison with related work and conclusive remarks are contained in Section 6.

2 Preliminaries

2.1 The Language

Below, we present first the syntax and then operational and bisimulation semantics of the language. The notation we use is that of value-passing CCS [Mil80, Mil89] and of π-calculus [MPW92]. We assume the following sets:

- a countable set *Act* of *pure actions* or *communications ports*, ranged over by $a, a', \ldots$;
- a countable set Var of *variables*, ranged over by $x, y, \ldots$;
- a set Val of *values*, ranged over by $v, v', \ldots$, containing at least two distinct elements;
- a countable set *Ide* of *identifiers* each having a non-negative *arity*. *Ide* is ranged over by *Id* and capital letters and is disjoint from the previous sets.

A *value expression* is either a variable or a value. Value expressions are ranged over by $e, e', \ldots$. We also consider the set $\overline{Act} = \{\overline{a} \mid a \in Act\}$ of *co-actions*, which represent output synchronizations. The set $Act \cup \overline{Act}$ will be ranged over by c.

The set of *terms* of our language, ranged over by $P, Q, \ldots$, is given by the operators of *pure synchronization prefix, input prefix, output prefix, non-determinism, matching* and *identifier*, according to the following grammar:

$$P ::= c.P \;\mid\; a(x).P \;\mid\; \overline{a}e.P \;\mid\; \sum_{i \in I} P_i \;\mid\; [e_1 = e_2]P \;\mid\; Id(e_1, \ldots, e_k)$$

where k is the arity of Id. We always assume that the index set I in $\sum_{i \in I} P_i$ is finite and sometimes write $P_1 + \cdots + P_n$ for $\sum_{i \in \{1,\ldots,n\}} P_i$. When I is empty, we use the symbol $\mathbf{0}$: $\mathbf{0} \overset{def}{=} \sum_{i \in \emptyset} P_i$. When no confusion may arise, we write c for $c.\mathbf{0}$.

An occurrence of a variable x in a term P is said to be *bound* if it is within the scope of an input prefix $a(x)$; otherwise it is said a *free* occurrence. The set of variables which have a bound occurrence in P is denoted by $bvar(P)$, while the set of variables which have a free occurrence in P is denoted by $fvar(P)$; $var(P)$ is $bvar(P) \cup fvar(P)$. We define $val(P)$ as the set of values occurring in P. The *size* of a term P, indicated by $|P|$, is the number of symbols appearing in it; e.g., if $P = a(x).\overline{a}x.a'.\mathbf{0} + Id(x)$ then $|P| = 9$.

Substitution of the distinct variables $x_1, \ldots, x_n$ with the values $v_1, \ldots, v_n$, indicated by $\{v_1/x_1, \ldots, v_n/x_n\} = \{\tilde{v}/\tilde{x}\}$ and *composition* of two substitutions σ and σ', denoted by $\sigma\sigma'$, are defined as expected. We let $\sigma, \ldots$ range over substitutions. The function *val* is extended to substitutions in the obvious way and such notations as $val(P, Q, \sigma)$ will mean $val(P) \cup val(Q) \cup val(\sigma)$.

We presuppose an arbitrarily fixed *finite* set Eq of *identifiers definitions*, each of the form

$$Id(x_1, \ldots, x_k) \Leftarrow P$$

where $k \geq 0$ is the arity of Id. We require that the x_i are pairwise distinct and that $fvar(P) \subseteq \{x_1, \ldots, x_k\}$. In Eq, each identifier has a single definition. The requirement for the set Eq to be finite is motivated by the fact that we are only interested in syntactically finite processes.

Note that we have not made any assumption on whether the sets Var, Val and Act are pairwise disjoint or not. There are two particularly interesting cases: (i) if Act, Var and Val are pairwise disjoint we get a proper sublanguage of value-passing CCS [Mil80, Mil89]; this case will be referred to as the *simple value-passing* case; (ii) if $Act = Var = Val$, we get a proper sublanguage of the π-calculus [MPW92]; this case will be referred to as the *name-passing* case.

Most of our results will not depend on a particular such assumption. Also, they will not depend on whether Val is finite or infinite (though, of course, if the name-passing assumption is made, Val must be infinite, since Act is).

A process term P is said to be *closed* if $fvar(P) - Val = \emptyset$; in this case, P is said to be a *process*. According to this definition, all terms are processes in a name-passing setting. Processes are the terms we are most interested in. As we shall see, bisimulation semantics will be defined only over the set of processes.

Since we are interested in determining the contributions of different operators to the complexity of deciding bisimilarity, it is convenient to single different (sub)languages out of the syntax defined above. The *data-independent* languages $\mathcal{IL}_v$, $\mathcal{IL}_r$ and $\mathcal{IL}_{v,r}$ are defined as follows:

- $\mathcal{IL}_v$ contains all operators, but identifiers and matching;
- $\mathcal{IL}_r$ contains all operators, but input and output prefixes and matching;
- $\mathcal{IL}_{v,r}$ contains all operators, but matching.

The *data-dependent* languages $\mathcal{L}_v$, $\mathcal{L}_r$ and $\mathcal{L}_{v,r}$, are defined similarly, but with matching in addition. In particular, $\mathcal{L}_{v,r}$ is the full language.

The operational behaviour of our processes is defined by means of a transition relation. Its elements are triples (P, μ, P') written as $P \xrightarrow{\mu} P'$. Here, μ can be of three different forms: c, $\overline{a}v$ or $a(v)$. A *pure action* c represents a synchronization through the port c, without passing of data involved. An *output action* $\overline{a}v$ means transmission of the datum v through the port a. An *input action* $a(v)$ represents receipt of the datum v through the port a. We let μ range over actions. The transition relation is defined by the inference rules in Table 2.1. Note that $\xrightarrow{\mu}$ leads processes into processes.

$$(Sync)\ c.P \xrightarrow{c} P$$

$$(Inp)\ a(x).P \xrightarrow{a(v)} P\{v/x\},\ v \in Val \qquad (Out)\ \overline{a}v.P \xrightarrow{\overline{a}v} P$$

$$(Match)\frac{P \xrightarrow{\mu} P'}{[v = v]P \xrightarrow{\mu} P'} \qquad (Sum)\frac{P_j \xrightarrow{\mu} P'}{\sum_{i \in I} P_i \xrightarrow{\mu} P'}\ j \in I$$

$$(Ide)\frac{P\{\tilde{v}/\tilde{x}\} \xrightarrow{\mu} P'}{Id(\tilde{v}) \xrightarrow{\mu} P'} \qquad \text{if } Id(\tilde{x}) \Leftarrow P \text{ is in } Eq$$

Table 2. Inference rules for the transition relation $\xrightarrow{\mu}$.

On the top of the transition relation $\xrightarrow{\mu}$, we define *strong bisimulation equivalence* $\sim$, [Mil89, MPW92, PS93] as usual:

Definition 1 (Strong bisimulation equivalence). A binary symmetric relation $\mathcal{R}$ over processes in $\mathcal{L}_{v,r}$ is a *bisimulation* if, whenever $P\,\mathcal{R}\,Q$ and $P \xrightarrow{\mu} P'$, there exists Q' s.t. $Q \xrightarrow{\mu} Q'$ and $P'\,\mathcal{R}\,Q'$. We let $P \sim Q$, and say that P *is bisimilar to* Q, if and only if $P\,\mathcal{R}\,Q$, for some bisimulation $\mathcal{R}$.

From now on, we will omit the adjective "strong". A drawback of the above definition is that it requires considering the whole (possibly infinite) set of transitions of the two processes being compared. We will rely on an alternative "finitary" definition of bisimulation. It differs from the standard one in that, on the input action clause, case-analysis on just a *finite* set of values is required. In the sequel, we say that a value v is *fresh* if v does not occur in any previously mentioned process, nor in the set Eq.

Definition 2 (F-bisimulation). Let $\mathcal{R}$ be a symmetric relation over processes. We say that $\mathcal{R}$ is a *F-bisimulation* if, whenever $P\ \mathcal{R}\ Q$, then:

– $P \xrightarrow{\mu} P'$, with μ not an input action, implies $Q \xrightarrow{\mu} Q'$ for some Q' s.t. $P' \mathcal{R}\, Q'$, and
– for some fresh v_0, $P \xrightarrow{a(v)} P'$, with $v \in val(P, Q, Eq) \cup \{v_0\}$, implies $Q \xrightarrow{a(v)} Q'$ for some Q' s.t. $P' \mathcal{R}\, Q'$.

Define $P \sim_F Q$ if and only if $P \mathcal{R} Q$ for some F-bisimulation $\mathcal{R}$.

Intuitively, doing case-analysis on input actions by considering just one fresh value suffices, because, under certain conditions, bisimulation is preserved by replacements of values with fresh values. Indeed, we have:

Theorem 3. *$P \sim Q$ if and only if $P \sim_F Q$.*

2.2 Complexity Classes and Hard Problems

In the paper, we will measure the complexity of deciding bisimilarity for P and Q with a set of identifier definitions Eq, in function of the sum of the syntactical sizes of P, Q and of the processes occurring in Eq.

We will deal with the complexity classes P, NP, *co*NP and PSPACE, and with the notions of *polynomial reducibility*, *hardness* and *completeness*. It is known that P $\subseteq$ NP, *co*NP $\subseteq$ PSPACE, and it is strongly conjectured that all these classes are distinct. A problem is hard for a class $\mathcal{C}$ if every problem in $\mathcal{C}$ is polynomial-time reducible to it; a $\mathcal{C}$-hard problem is said to be $\mathcal{C}$-complete if it belongs to $\mathcal{C}$. Formal definitions can be found in any textbook of computational complexity theory, such as [BC93, Pap94].

3 Data-Independent Calculi

In this section we will deal with the complexity of the bisimulation problem in the three data-independent calculi. We will first restrict ourselves to the simple value-passing case (i.e. we assume that Var, Val and Act are pairwise disjoint) and we will argue how the achieved results apply to the name-passing case.

Theorem 4. *The bisimilarity problem for $\mathcal{IL}_v$ and $\mathcal{IL}_r$ is in* P.

Let us now consider the complexity of the bisimilarity problem in $\mathcal{IL}_{v,r}$. Jonsson and Parrow proved that such a problem is decidable and NP-hard [JP93]. We will improve on this result and we will show that the problem is indeed PSPACE-hard, by reducing to it a well-known PSPACE-complete problem. We first need some preliminary definitions in order to introduce *quantified boolean formulas.*

Let $U = \{x_1, \ldots, x_n\}$ be a set of boolean variables. If x is a variable in U, then x and $\neg x$ are said to be *literals.* A *(conjunctive) 3-clause* over U is the conjunction of three literals. A *formula in 3DNF* (disjunctive normal form) is the disjunction of a set of 3-clauses, e.g. $\phi \overset{def}{=} (x_1 \wedge \neg x_3 \wedge \neg x_4) \vee (x_2 \wedge \neg x_3 \wedge x_4)$. A *truth assignment* for U is a function $t : U \to \{\texttt{true}, \texttt{false}\}$. Given an assignment t, we associate in the usual way a truth value to literals, clauses and formulas. With a slight abuse of notation, we will admit that a literal may also be a member of the set $\{\texttt{true}, \texttt{false}, \neg\texttt{true}, \neg\texttt{false}\}$. Each assignment t will map these special literals to the expected truth values. Moreover, let $\phi\{b_1/x_1, \ldots, b_k/x_k\}$, where $b_i \in \{\texttt{true}, \texttt{false}\}$, be the formula obtained from ϕ by substituting b_i to x_i for $i = 1, \ldots, k$.

Definition 5. A *quantified boolean formula* (in short, QBF) is a formula $\phi = Q_1 x_1.Q_2 x_2 \ldots Q_n x_n.\phi'$, where ϕ' is a formula in 3DNF, $\{x_1, \ldots, x_n\}$ is the set of variables occurring in ϕ' and, for any $i = 1, \ldots, n$, $Q_i \in \{\exists, \forall\}$ is a *quantifier.*

Definition 6. A *quantified boolean formula* $\phi = Q_1 x_1.Q_2 x_2 \ldots .Q_n x_n.\phi'$ is *valid* if one of the following conditions holds:

1. $n = 0$ and in ϕ' there is a true clause c;
2. $Q_1 = \exists$, and $Q_2 x_2 \ldots Q_n x_n.\phi'\{\texttt{true}/x_1\}$ or $Q_2 x_2 \ldots Q_n x_n.\phi'\{\texttt{false}/x_1\}$ is valid;
3. $Q_1 = \forall$, and $Q_2 x_2 \ldots Q_n x_n.\phi'\{\texttt{true}/x_1\}$ and $Q_2 x_2 \ldots Q_n x_n.\phi'\{\texttt{false}/x_1\}$ are both valid.

Given a QBF ϕ, the QBF problem consists of deciding whether ϕ is valid: this is a PSPACE-complete problem [SM73], and it is easy to see that it remains PSPACE-complete even when restricted to formulas $\phi = Q_1 x_1 Q_2 x_2 \ldots, Q_n x_n.\phi'$ such that n is even and $Q_i = \exists$ if and only if i is odd. Let us call RQBF this restricted problem. We now come to describing the actual reduction.

Let $\phi = \exists x_1.\forall x_2.\ldots\forall x_n.\phi'$ be an instance of RQBF, where $\phi' = c_1 \vee \ldots \vee c_m$, and $c_i = l_i^1 \wedge l_i^2 \wedge l_i^3$ for $i = 1, \ldots, m$. Let us define the processes $B_0, \ldots, B_n$, $T_0, \ldots, T_n$, $E_0, \ldots, E_n$ as shown in Table 3. There, in the clause for B_n, i_j is the index of the variable x_{i_j} occurring in literal l_i^j, and $w_{i_j} = y_{i_j}$ if $l_i^j = x_{i_j}$, while $w_{i_j} = z_{i_j}$ if $l_i^j = \neg x_{i_j}$.

We will prove that $B_0 \sim T_0$ if and only if ϕ is valid. The proof is split into three technical lemmas.

Lemma 7. *For any i, $0 \leq i \leq n$ and for any $(v_1, \ldots, v_i) \in \{\texttt{true}, \texttt{false}\}^i$, $B_i(v_1, \ldots, v_i, \neg v_1, \ldots, \neg v_i)$ is either bisimilar to E_i or bisimilar to T_i.*

Lemma 8. *For any i, $0 \leq i \leq n$, T_i is not bisimilar to E_i.*

Lemma 9. *Consider any i, $0 \leq i \leq n$ and any $(v_1, \ldots, v_i) \in \{\texttt{true}, \texttt{false}\}^i$. Let $\psi' \overset{def}{=} \phi'\{v_1/x_1, \ldots, v_i/x_i\}$, and define $\psi \overset{def}{=} Q_{i+1} x_{i+1} \ldots \forall x_n.\psi'$, where Q_{i+1} is the $(i+1)$-th quantifier in ϕ. Then, ψ is valid if and only if $B_i(v_1, \ldots, v_i, \neg v_1, \ldots, \neg v_i) \sim T_i$.*

Proof. (Sketch) We proceed by induction on $n - i$. If $i = n$, then the proof is trivial. Now, fix any $(v_1, \ldots, v_i) \in \{\texttt{true}, \texttt{false}\}^i$ and let ψ' and ψ be as described in the hypothesis. We can assume by inductive hypothesis that for any $v_{i+1} \in \{\texttt{true}, \texttt{false}\}$, $B_{i+1}(v_1, \ldots, v_{i+1}, \neg v_1, \ldots, \neg v_{i+1}) \sim T_{i+1}$ if

$$E_n \Leftarrow \sum_{\substack{(v_1,v_2,v_3)\in\{\mathtt{true},\mathtt{false}\}^3 \\ (v_1,v_2,v_3)\neq(\mathtt{true},\mathtt{true},\mathtt{true})}} \overline{a}v_1.\overline{a}v_2.\overline{a}v_3$$

$$T_n \Leftarrow \sum_{(v_1,v_2,v_3)\in\{\mathtt{true},\mathtt{false}\}^3} \overline{a}v_1.\overline{a}v_2.\overline{a}v_3$$

$$B_n(y_1,\dots,y_n,z_1,\dots,z_n) \Leftarrow \sum_{i=1}^{m} \overline{a}w_{i_1}.\overline{a}w_{i_2}.\overline{a}w_{i_3} + \sum_{\substack{(v_1,v_2,v_3)\in\{\mathtt{true},\mathtt{false}\}^3 \\ (v_1,v_2,v_3)\neq(\mathtt{true},\mathtt{true},\mathtt{true})}} \overline{a}v_1.\overline{a}v_2.\overline{a}v_3$$

For any even i, $0 \leq i \leq n-2$:

$$\begin{aligned} B_i(y_1,\dots,y_i,z_1,\dots,z_i) &\Leftarrow a.B_{i+1}(y_1,\dots,y_i,\mathtt{true},z_1,\dots,z_i,\mathtt{false}) + \\ &\quad a.B_{i+1}(y_1,\dots,y_i,\mathtt{false},z_1,\dots,z_i,\mathtt{true}) + a.E_{i+1} \\ T_i &\Leftarrow a.T_{i+1} + a.E_{i+1} \\ E_i &\Leftarrow a.E_{i+1} \end{aligned}$$

For any odd i, $1 \leq i \leq n-1$:

$$\begin{aligned} B_i(y_1,\dots,y_i,z_1,\dots,z_i) &\Leftarrow a.B_{i+1}(y_1,\dots,y_i,\mathtt{true},z_1,\dots,z_i,\mathtt{false}) + \\ &\quad a.B_{i+1}(y_1,\dots,y_i,\mathtt{false},z_1,\dots,z_i,\mathtt{true}) + a.T_{1+1} \\ T_i &\Leftarrow a.T_{i+1} \\ E_i &\Leftarrow a.E_{i+1} + a.T_{i+1} \end{aligned}$$

Table 3. The reduction from QBF to bisimilarity in $\mathcal{IL}_{v,r}$.

and only if $Q_{i+2}x_{i+2},\dots,\forall x_n.\psi'\{v_{i+1}/x_{i+1}\}$ is valid. We have to distinguish two cases, depending on whether i is odd or even. Let i be even (the other case is similar), then $Q_{i+1} = \exists$. Due to lemmas 7 and 8, we have that $B_i(v_1,\dots,v_i,\neg v_1,\dots,\neg v_i) \sim T_i$ if and only if a value $v_{i+1} \in \{\mathtt{true},\mathtt{false}\}$ exists such that $B_{i+1}(v_1,\dots,v_{i+1},\neg v_1,\dots,\neg v_{i+1}) \sim T_{i+1}$. By inductive hypothesis, the latter holds if and only if either $\forall x_{i+2}\dots\forall x_n.\psi'\{\mathtt{false}/x_{i+1}\}$ is valid or $\forall x_{i+2}\dots\forall x_n.\psi'\{\mathtt{false}/x_{i+1}\}$ is valid, that is, if and only if $\psi = \exists x_{i+1}\dots\forall x_n.\psi'$ is valid. □

The following corollary is just a special case ($i = 0$) of the previous lemma.

Corollary 10. *$B_0 \sim T_0$ if and only if ϕ is valid.*

The definition of the identifiers can be easily constructed in polynomial time, thus it immediately follows the main result of this section.

Theorem 11. *The bisimilarity problem in $\mathcal{IL}_{v,r}$ is* PSPACE-*hard*

Let us now consider the name-passing case arising when $Act = Val = Var$. The results regarding $\mathcal{IL}_v$ and $\mathcal{IL}_{v,r}$ still apply, while bisimilarity in $\mathcal{IL}_r$ can be shown to be PSPACE-hard by using the reduction of Table 3, provided that we replace the output action $\overline{a}v$ with the simple action v.

4 Data-Dependent Value-Passing

In this section we will show that the bisimilarity problem for the calculus $\mathcal{L}_v$ is *co*NP-complete. We will first present a reduction from the *co*NP-complete problem 3-TAUTOLOGY, thus establishing the *co*NP-hardness of the bisimilarity problem. Then we will show that it belongs to the class *co*NP.

The 3-TAUTOLOGY problem consists in testing whether a given formula ϕ in 3DNF (see the preceding section) is a tautology or not. From the results of [Coo71] it follows that any problem in *co*NP is polynomial-time reducible to 3-TAUTOLOGY, that is, the 3-TAUTOLOGY problem is *co*NP-hard.

Theorem 12. *The bisimilarity problem in $\mathcal{L}_v$ is* coNP*-hard.*

Proof. (Sketch) It is sufficient to prove that the 3-TAUTOLOGY problem is polynomial-time reducible to the bisimilarity problem in $\mathcal{L}_v$. Let $\phi = c_1 \vee \ldots \vee c_m$ be an instance of 3-TAUTOLOGY over the set of variables $\{x_1, \ldots, x_n\}$, let $c_i = l_i^1 \wedge l_i^2 \wedge l_i^3$ for $i = 1, \ldots, m$, and let x_{i_j} be the variable occurring in literal l_j^i. Let also b_{i_j} stand for `true` if $l_i^j = x_{i_j}$, and for `false` otherwise. Consider the processes $P(\phi)$, Q, P', Q' as defined in Table 4.

$$\begin{aligned} P(\phi) &\overset{def}{=} a(y_1)\ldots a(y_n).P' \\ Q &\overset{def}{=} a(y_1)\ldots a(y_n).Q' \\ P' &\overset{def}{=} a.a + \textstyle\sum_{i=1}^{n} a.([y_i = \texttt{true}]a + [y_i = \texttt{false}]a) \\ &\quad + \textstyle\sum_{i=1}^{m} [y_{i_1} = b_{i_1}][y_{i_2} = b_{i_2}][y_{i_3} = b_{i_3}]a \\ Q' &\overset{def}{=} a + a.a \end{aligned}$$

Table 4. The reduction from 3-TAUTOLOGY to bisimilarity in $\mathcal{L}_v$.

It is possible to prove that ϕ is a tautology if and only if, for any $(v_1, \ldots, v_n) \in Val^n$, $P'\{v_1/y_1, \ldots, v_n/y_n\} \sim Q'$. This implies that $Q \sim P(\phi)$ if and only if ϕ is a tautology. By observing that Q and $P(\phi)$ are computable in polynomial time in the size of ϕ, the theorem follows. □

Theorem 13. *The bisimilarity problem in $\mathcal{L}_v$ is in* coNP*.*

Proof. (Sketch) We prove that the *inequivalence* problem (given P, Q in $\mathcal{L}_v$, decide whether $P \not\sim Q$) is in NP. To this aim, it is sufficient to consider the nondeterministic algorithm in Figure 1 and to show that the following properties hold:

1. the algorithm runs in polynomial time (in the sizes of the terms);
2. if $P \sim Q$ all computations of the algorithm lead to rejection;

3. if $P \not\sim Q$ there exists a computation of the algorithm leading to acceptance.

Due to lack of space, details are omitted. The only subtle point is the third one, where also Theorem 3 is exploited. □

```
Algorithm Non-equiv
Input: $P, Q$
begin
  if not $B(P,Q)$ then accept
  else reject
end

$B(P,Q)$
begin
  Fix $v_0$ fresh; guess $v \in val(P,Q) \cup \{v_0\}$;
  $I := \{(\mu, P') \mid P \xrightarrow{\mu} P'$ and if $\mu$ is an input action then $\mu = a(v)$, for some $a$ $\}$;
  $J := \{(\mu, Q') \mid Q \xrightarrow{\mu} Q'$ and if $\mu$ is an input action then $\mu = a(v)$, for some $a$ $\}$;
  for each $\big((\mu, P'), (\mu, Q')\big) \in I \times J$ do $b(\mu, P', Q') := B(P', Q')$;
  return $\big(\ \forall (\mu, P') \in I . \exists (\mu, Q') \in J : b(\mu, P', Q')\ \wedge$
            $\forall (\mu, Q') \in J . \exists (\mu, P') \in I : b(\mu, P', Q')\ \big)$
  end
```

Fig. 1. A nondeterministic algorithm for detecting inequivalence of processes in $\mathcal{L}_v$.

Corollary 14. *The bisimilarity problem in* $\mathcal{L}_v$ *is* coNP*-complete.*

5 Reducing Value-passing to Identifiers and Matching

We will exhibit a polynomial-time reduction of $\mathcal{L}_{v,r}$ to $\mathcal{L}_r$. It is convenient here to separate the case of simple value-passing (Val, Var and Act disjoint) and the case of name-passing ($Var = Val = Act$). We first deal with simple value-passing, and then indicate the necessary modifications to accommodate name-passing.

We will first give an informal account of the translation. As a first approximation, the idea is to express each input process $a(x).P$ as a nondeterministic sum $\sum_{v \in V} av.P\{v/x\}$. Here, each av is a pure action uniquely associated with the channel a and the value v; V is a set of values, which is finite, but large enough to represent all "relevant" input actual parameters. The idea stems from Definition 2 and from Milner's translation of CCS with values into pure CCS (with infinite summation [Mil89]). However, in the presence of nested input actions, this solution would give rise to an exponential explosion of the size of translated term. To overcome this drawback, we exploit the ability of identifiers of handling parameters. The idea is to translate $a(x).P$ as $\sum_{v\in V} av.A(v)$, where A is an auxiliary identifier defined by $A(x) \Leftarrow T$ and T is the translation of the subterm P.

We assume an arbitrarily large supply of auxiliary identifiers of arity j, A_1, A_2, $A_3, \ldots$, for any $j \geq 0$, each distinct from the identifiers defined in Eq. These auxiliary identifiers will be ranged over by the letter A.

The actual translation consists of two parts: for each term P in $\mathcal{L}_{v,r}$, we have to specify, in $\mathcal{L}_r$, a term $[\![P]\!]$, and a set of identifiers definitions, $\mathcal{D}(P)$, which defines the auxiliary identifiers occurring in $[\![P]\!]$. The definitions of $[\![P]\!]$ and $\mathcal{D}(P)$ are reported in Table 5. The definition of $[\![P]\!]$ is parametric with a chosen non-empty set $V_0 \subseteq_{fin} Val$ of values, appearing in the clauses for input and output prefixes. Thus we should have written $[\![P]\!]_{V_0}$ in place of $[\![P]\!]$; we have omitted the subscript $_{V_0}$ as no confusion can arise. Note that the definition of $[\![P]\!]$ does not depend on that of $\mathcal{D}(P)$, while the latter does depend on the former.

$[\![P]\!]$ is defined as:	$\mathcal{D}(P)$ is defined as:
$[\![c.P]\!] = c.[\![P]\!]$	$\mathcal{D}(c.P) = \mathcal{D}(P)$
$[\![\overline{a}v.P]\!] = \overline{av}.[\![P]\!]$	$\mathcal{D}(\overline{a}v.P) = \mathcal{D}(P)$
$[\![\overline{a}x.P]\!] = \sum_{v\in V_0}[x = v]\overline{av}.A(\tilde{y})$	$\mathcal{D}(\overline{a}x.P) = \{A(\tilde{y}) \Leftarrow [\![P]\!]\} \cup \mathcal{D}(P)$
where $\tilde{y} = fvar([\![P]\!])$	
$[\![a(x).P]\!] = \sum_{v\in V_0} av.A(\tilde{y}, v)$	$\mathcal{D}(a(x).P) = \{A(\tilde{y}, x) \Leftarrow [\![P]\!]\} \cup \mathcal{D}(P)$
where $\tilde{y} = fvar([\![P]\!]) - \{x\}$	
$[\![[e_1 = e_2]P]\!] = [e_1 = e_2][\![P]\!]$	$\mathcal{D}([e_1 = e_2]P) = \mathcal{D}(P)$
$[\![\sum_{i\in I} P_i]\!] = \sum_{i\in I}[\![P_i]\!]$	$\mathcal{D}(\sum_{i\in I} P_i) = \bigcup_{i\in I} \mathcal{D}(P_i)$
$[\![Id(\tilde{e})]\!] = Id(\tilde{e})$	$\mathcal{D}(Id(\tilde{e})) = \emptyset$

Table 5. The reduction of $\mathcal{L}_{v,r}$ to $\mathcal{L}_r$.

The translation has to be applied to the set of identifiers definitions, Eq, as follows:

Definition 15. Let us define $\mathcal{D}(Eq)$ as

$$\bigcup_{Id(\tilde{y}) \Leftarrow P \in Eq} \left(\{Id(\tilde{y}) \Leftarrow [\![P]\!]\} \cup \mathcal{D}(P)\right).$$

The reduction proof can be split in two parts: *completeness* (if $P \sim Q$ in $\mathcal{L}_{v,r}$, then their translations are bisimilar in $\mathcal{L}_r$) and *correcteness* (if the translations of P and Q are bisimilar in $\mathcal{L}_r$, then P and Q are bisimilar in $\mathcal{L}_{v,r}$).

Theorem 16 (Completeness). *For any two processes P_0 and Q_0 in $\mathcal{L}_{v,r}$, if $P_0 \sim Q_0$ then $[\![P_0]\!] \sim [\![Q_0]\!]$ in $\mathcal{L}_r$ equipped with the set of identifier definitions $\mathcal{D}(P_0) \cup \mathcal{D}(Q_0) \cup \mathcal{D}(Eq)$.*

Proof. (Sketch) For any two terms P and Q, define $P \prec Q$ if P is a *subterm* of Q, where the standard definition of subterm is extended by the axiom: $R\{\tilde{v}/\tilde{x}\} \prec Id(\tilde{v})$ if $Id(\tilde{x}) \Leftarrow R$ is in Eq.

Let $\mathcal{L}_r$ be equipped with the identifiers definitions $\mathcal{D}(P_0) \cup \mathcal{D}(Q_0) \cup \mathcal{D}(Eq)$. Over $\mathcal{L}_r$, consider the relation $\mathcal{R}$ defined thus:

$$\{([\![P]\!]\sigma_1, [\![Q]\!]\sigma_2) \mid [\![P]\!]\sigma_1 \text{ and } [\![Q]\!]\sigma_2 \text{ are closed, } P \prec P_0, \tag{1}$$

$$Q \prec Q_0 \text{ and } P\sigma_1 \sim Q\sigma_2 \text{ in } \mathcal{L}_{v,r} \}. \tag{2}$$

Then it is not difficult to prove that $\mathcal{R}$ is a bisimulation *up to* $\sim$ [Mil89] and, hence, $\mathcal{R} \subseteq \sim$. $\square$

We now come to the correctness part. This is slightly more difficult, because we have to choose appropriately the parameter parameter V_0 of the translation. The choice depends also on whether or not Val is infinite. In the next theorem, we assume that Val is infinite; the case when Val is finite will be easily accommodated afterward. Intuitively, V_0 must contain all "relevant" values, i.e. all values appearing in the two processes being compared and in their subterms, *plus* a reserve of fresh values.

Theorem 17 (Correctness). *For any two processes P_0 and Q_0 in $\mathcal{L}_{v,r}$, if $[\![P_0]\!] \sim [\![Q_0]\!]$ in $\mathcal{L}_r$ equipped with $\mathcal{D}(P_0) \cup \mathcal{D}(Q_0) \cup \mathcal{D}(Eq)$, then $P_0 \sim Q_0$.*

Proof. (Sketch) Let the parameter V_0 of the translation be set as $V_0 = val(P_0, Q_0, Eq) \cup V$, for some $V \subseteq_{fin} Val$ s.t. $V \cap val(P_0, Q_0, Eq) = \emptyset$ and $|V| = |P_0| + |Q_0| + \sum_{Id(\tilde{x}) \Leftarrow R \in Eq} |R|$. Over $\mathcal{L}_{v,r}$, define the relation $\mathcal{R}$ as follows:

$$\{(P\sigma_1, Q\sigma_2) \mid P\sigma_1 \text{ and } Q\sigma_2 \text{ are closed, } P \prec P_0,\, Q \prec Q_0,$$
$$val(\sigma_1, \sigma_2) \subseteq V_0 \text{ and } [\![P]\!]\sigma_1 \sim [\![Q]\!]\sigma_2 \text{ in } \mathcal{L}_r\}.$$

It is not hard to show that $\mathcal{R}$ is an F-bisimulation, and hence (by Theorem 3) $P_0 \sim Q_0$. $\square$

If Val is finite, then the above theorem can be proven by just letting $V_0 = Val$.

It is easily seen that the translation can be carried out in polynomial-time with the size of the problem. Thus, putting together Theorems 16 and 17, we get:

Theorem 18. *The equivalence problem in $\mathcal{L}_{v,r}$ is polynomial-time reducible to the equivalence problem in $\mathcal{L}_r$. Consequently, the equivalence problem in $\mathcal{L}_r$ is* PSPACE*-hard.*

We indicate now the necessary changes to accommodate the name-passing case ($Act = Var = Val$). In a name-passing input action $a(x).$, not only the formal parameter x, but also the channel a is subject to be possibly instantiated. It then suffices to replace the output (both $\overline{a}v.$ and $\overline{a}x.$) clauses and the input clause of the definition of $[\![\,.\,]\!]$ in Table 5 with the following two:

$$[\![a(x).P]\!] = \textstyle\sum_{v \in V_0}[a = v] \sum_{w \in V_0} vw.A(\tilde{y}, w) \qquad \text{where } \tilde{y} = fvar([\![P]\!]) - \{x\}$$

$$[\![\overline{a}y.P]\!] = \textstyle\sum_{v \in V_0}[a = v] \sum_{w \in V_0}[y = w]\overline{vw}.A(\tilde{y}) \quad \text{where } \tilde{y} = fvar([\![P]\!])\,.$$

The remaining clauses and the definition of $\mathcal{D}$ are left unchanged. It is easy to see that the translation is still polynomial and that the reduction proofs carry over essentially without modifications.

6 Conclusions

In this paper we have studied the decidability and the complexity of bisimilarity in fragments of CCS with values and of the π-calculus. We considered both a data-independent setting, in which processes are allowed to send and receive data, but cannot do any test on them, and a simple data-dependent one, in which processes can only perform equality tests. In the literature, some variant form of bisimulation have been proposed, such as *late* bisimilarity [MPW92, PS93] and *open* bisimilarity [San93]. Most of the results presented in previous sections extend to these equivalences. In particular, both late and open bisimilarity are PSPACE-hard over the data-independent processes, because the three equivalences coincide in this case (see e.g. [PS93]).

Our paper is mainly related to [JP93]. There, Jonsson and Parrow prove that bisimilarity is decidable in the data-independent language $\mathcal{IL}_{v,r}$, by showing that the infinitely many transitions due to an input action can be reduced to a single, suitably chosen, *schematic* action [JP93]. The latter is characterized as the receipt of the least value (w.r.t. to a fixed ordering of values) not "used" in the considered process. This approach yields the polynomial-time tractability of some restricted cases. On the other hand, the technique cannot be used in a data-dependent setting, mainly because in the presence of the equality test, determining the set of "used" values of a process becomes very complex (perhaps undecidable). In this paper, we have taken a less radical approach to deal with the infinite-state problem: instead of substituting infinitely many actions with a single one, we replace them with a "moderate" number of actions (the ones corresponding to the set V_0). Jonsson and Parrow also show that $\mathcal{IL}_{v,r}$ is NP-hard, by means of a quite involved reduction from the CLIQUE problem. Here, we have for the same language a stronger result with an easier technique.

In [HL95, HL93, San93, BD94, EL95], notions of *symbolic* bisimulation are investigated for both CCS with value-passing and π-calculus, aiming at a more efficient representation of bisimilarity. Our results show that, even for very simple fragments, it is very unlikely that efficient algorithms exist. It remains to be seen whether symbolic techniques give some benefits on the average.

A question that is left open by the present work is the exact complexity of bisimilarity in $\mathcal{IL}_{v,r}$ and $\mathcal{L}_{v,r}$. Moreover, other interesting fragments of CCS with values should be considered from a complexity point of view. For example, the parallel composition operator $|$ has been considered in the case of *trace equivalence* [MS94], but nothing is known regarding bisimilarity.

References

[BC93] D.P. Bovet and P. Crescenzi. *Introduction to the Theory of Complexity.* Prentice Hall, 1993.

[BD94] M. Boreale and R. De Nicola. A symbolic semantics for the π-calculus. In *Proc. of CONCUR '94, LNCS 836*, pages 299–314. Springer-Verlag, 1994.

[Coo71] S.A. Cook. The complexity of theorem proving procedures. In *Proc. of STOC '71*, pages 151–158, 1971.

[EL95] U.H. Engberg and K.S. Larsen. Efficient simplification of bisimulation formulas. In *Proc. of TACAS '95*, pages 89–103, 1995.

[HL93] M. Hennessy and H. Lin. Proof systems for message-passing process algebras. In *Proc. of CONCUR '93, LNCS 715.* Springer-Verlag, 1993.

[HL95] M. Hennessy and H. Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138:353–389, 1995.

[JP93] B. Jonsson and J. Parrow. Deciding bisimulation equivalences for a class of non-finite state programs. *Information and Computation*, 107:272–302, 1993.

[KS90] P.C. Kannellakis and S.A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86:43–68, 1990.

[Mil80] R. Milner. *A Calculus of Communicating Systems.* LNCS, 92. Springer-Verlag, 1980.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part I and II. *Information and Computation*, 100:1 –41 and 42–78, 1992.

[MS94] A.J. Mayer and L.J. Stockmeyer. The complexity of word problems – this time with interleaving. *Information and Computation*, 115:293–311, 1994.

[Pap94] C.H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[PS93] J. Parrow and D. Sangiorgi. Algebraic theories for name-passing calculi. Technical Report ECS–LFCS–93–262, University of Edinburgh, Department of Computer Science, 1993. To appear in *Information and Computation.*

[PT87] R. Paige and R.E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[San93] D. Sangiorgi. A theory of bisimulation for the π-calculus. In *Proc. of CONCUR '93, LNCS 715.* Springer-Verlag, 1993. To appear in *Acta Informatica.*

[SM73] L. Stockmeyer and A. Meyer. Word problems requiring exponential time. In *Proc. of STOC '73*, pages 1–9, 1973.

This article was processed using the LaTeX macro package with LLNCS style