A Complexity Analysis of Bisimilarity for Value-Passing Processes*

MICHELE BOREALE AND LUCA TREVISAN
Dipartimento di Scienze dell'Informazione
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma
Email: {michele,trevisan}@dsi.uniroma1.it

February 8, 1996

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 shows that the fragment without value-passing is computationally equivalent to the full calculus. For the latter, bisimilarity is then proved to be EXP-complete.

Finally, we add to our language a parallel composition operator and show that, for a certain restricted syntactic format, the bisimilarity problem is still decidable and EXP-complete.

Corresponding author:
Michele Boreale
Dipartimento di Scienze dell'Informazione
Università degli Studi di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
Email: michele@dsi.uniroma1.it

*An extended abstract of part of the material contained in this paper appears in the *Proceedings of 15th Conference on Software Technology and Theoretical Computer Science (FSTTCS95)*, published by Springer Verlag.

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 $C(x)$:

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

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, PS95, 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 (also called "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 9.

In the present paper, we consider a calculus for describing non-deterministic processes that 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, w.r.t. the syntactical size of processes, 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). More precisely, in each of the two cases, we consider separately three (sub-)languages, obtained from the calculus with pure actions and non-deterministic choice 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 NP, $co\mathsf{NP}$, PSPACE and EXP (the latter contains the first three and PSPACE contains the first two, 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. For the calculus allowing both these primitives, we then prove that the problem is PSPACE-hard. This improves on a NP-hardness result due to Jonsson and Parrow.

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 $co\mathsf{NP}$-complete. Then we analyze the complexity of the full language, with both value-passing and recursive definitions. We show that this 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. It is important to point out that the matching predicate plays a crucial role in the definition of the translation. We then prove that bisimilarity for the full language is EXP-hard. To the best of authors' knowledge, the latter represents the highest complexity lower-bound ever determined for a decidable bisimilarity over a meaningful language.

Finally, we consider adding to the language a binary *parallel composition* operator $(P \mid Q)$. We show that, for a restricted format of the resulting calculus, where parallel composition does not appear inside recursive definitions (*finite-control* processes), the bisimilarity problem is still decidable and EXP-complete (this implies that the fragments without parallel compositions are all in EXP).

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 $co\mathsf{NP}$-complete. By contrast, the presence of recursive definitions themselves makes the problem EXP-complete; then, the adding of value-passing and of (a limited form of) parallel composition does not increase neither the expressive nor the computational power. These results are also summarized in Table 1.

The most important conclusion we can draw out of this analysis is that, in the presence of values, most of the complexity is not due to value-passing, nor to parallel composition, but to parametric recursive definitions.

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 establishes some basic properties of operational semantics and bisimulation that will be used throughout the

Language	Complexity
$\mathcal{L}_v$	P
$\mathcal{L}_r$	P
$\mathcal{L}_{v,r}$	PSPACE-hard
$\mathcal{L}_{m,v}$	*co*NP-complete
$\mathcal{L}_{m,r}$	EXP-complete
$\mathcal{L}_{m,v,r}$	EXP-complete
$\mathcal{L}_{m,v,r,p}$	EXP-complete

v = value-passing, r = recursive definitions,
m = matching, p = finite-control parallel composition.

Table 1: The complexity results of the paper.

rest of the paper. Section 4 deals with the complexity of data-independent processes. As to data-dependent processes, value-passing is dealt with in Section 5, while the relationship between the language with all the primitives for handling values and the fragment without value-passing is investigated in Section 6. In Section 7, we establish that these two calculi are EXP-hard. Section 8 deals with a language with parallel composition in addition. Comparison with related work and conclusive remarks are contained in Section 9.

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:

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

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 $\mathbf{0}$: $\mathbf{0} \stackrel{\text{def}}{=} \sum_{i \in \emptyset} P_i$. We will sometimes abbreviate $\alpha.\mathbf{0}$ simply as α, for any action prefix α.

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 $|P| = 9$.

We indicate by $\{v_1/x_1, \ldots, v_n/x_n\}$, $n \geq 0$, the simultaneous substitution of the distinct variables $x_1, \ldots, x_n$ with the values $v_1, \ldots, v_n$. This may involve renaming of bound names with fresh names, as usual, to prevent captures of free names (see [MPW92]). We we let also $\tilde{x}$ (resp. $\tilde{v}$) range over tuples of variables $(x_1, \ldots, x_n)$ (resp. of values $(v_1, \ldots, v_n)$), and abbreviate often $\{v_1/x_1, \ldots, v_n/x_n\}$ as $\{\tilde{v}/\tilde{x}\}$. We let ρ, $\sigma, \sigma', \ldots$ range over substitutions; composition of two substitutions σ and σ' is defined as expected and indicated by $\sigma\sigma'$. We also extend val and var over substitutions in the obvious way, by letting $val(\{\tilde{v}/\tilde{x}\}) = \tilde{v}$ and $var(\{\tilde{v}/\tilde{x}\}) = \tilde{x}$. By a slight abuse of notation, expressions such as $val(P, Q, \sigma)$ will be used to indicate $val(P) \cup val(Q) \cup val(\sigma)$; furthermore, $\tilde{x}$ (resp. $\tilde{v}$) will be used sometimes also to indicate a set of variables (resp. values), rather than a tuple.

We presuppose an arbitrarely 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. We will consider two particularly interesting cases:

- Act, Var and Val are pairwise disjoint. This gives rise to a sublanguage of value-passing CCS [Mil80, Mil89] and will be referred to as the *simple value-passing* case.

- $Act = Var = Val$. This gives rise to a sublanguage of the π-calculus [MPW92] and will be referred to as the *name-passing* case.

Most of our results will not depend on a particular such assumption. All results, but one in Section 7, do 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 $\mathcal{L}_v$, $\mathcal{L}_r$ and $\mathcal{L}_{v,r}$ all contain pure actions prefixes and summation and furthermore:

- $\mathcal{L}_v$ contains input/output prefixes;

- $\mathcal{L}_r$ contains parametric recursive definitions;

- $\mathcal{L}_{v,r}$ contains both input/output prefixes and parametric recursive definitions.

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

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

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

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

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

The *data-dependent* languages $\mathcal{L}_{m,v}$, $\mathcal{L}_{m,r}$ and $\mathcal{L}_{m,v,r}$, are defined similarly, but with matching in addition. In particular, $\mathcal{L}_{m,v,r}$ 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. On the top of the transition relation $\xrightarrow{\mu}$, we define *strong bisimulation equivalence* $\sim$, [Mil89, MPW92, PS95] as usual:

Definition 2.1 (Strong bisimulation equivalence) *A binary symmetric relation $\mathcal{R}$ over processes in $\mathcal{L}_{m,v,r}$ 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".

2.2 Complexity classes, hard problems, and alternating Turing machines

In the paper, we will measure the complexity of deciding bisimilarity between 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 terms occurring in Eq.

We will deal with the complexity classes P, NP, $co\mathsf{NP}$, PSPACE, $\mathsf{LIN\text{-}EXP}$ and EXP and with the notions of *polynomial reducibility*, *hardness* and *completeness*. Let us denote by DTIME($f(n)$) (respectively, SPACE($f(n)$)) the class of languages decidable by deterministic Turing machines that, for any input x of size n, halt within $f(n)$ steps (respectively, use at most $f(n)$ cells of the tape). Let also NTIME($f(n)$) be the class of languages decidable by non-deterministic Turing machines that, for any input x of size n, halt within $f(n)$ steps. Then

$$\mathsf{P} \stackrel{\text{def}}{=} \bigcup_{k>1} \text{DTIME}(n^k), \quad \mathsf{NP} \stackrel{\text{def}}{=} \bigcup_{k>1} \text{NTIME}(n^k)\ ,$$

$co\mathsf{NP}$ is the set of languages whose complement is in NP,

$$\mathsf{PSPACE} \stackrel{\text{def}}{=} \bigcup_{k>1} \mathrm{SPACE}(n^k), \ \mathsf{LIN\text{-}EXP} \stackrel{\text{def}}{=} \bigcup_{k>1} \mathrm{DTIME}(2^{kn}), \ \text{ and } \mathsf{EXP} \stackrel{\text{def}}{=} \bigcup_{k>1} \mathrm{DTIME}(2^{n^k}) \ .$$

It is known that $\mathsf{P} \subseteq \mathsf{NP}$, $co\mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP}$, and it is strongly conjectured that all these classes are distinct. Furthermore, $\mathsf{P} \subset \mathsf{LIN\text{-}EXP} \subset \mathsf{EXP}$ and the these three classes are provably 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 $\mathcal{C}$. It is easy to show that a problem is $\mathsf{LIN\text{-}EXP}$-hard if and only if it is EXP-hard. See e.g. [BC93, Pap94] for a more complete introduction to complexity classes. Here we recall the following result due to Hartmanis and Stearns that states the provable intractability of $\mathsf{LIN\text{-}EXP}$-hard problem.

Theorem 2.2 ([HS65]) *For any* $\mathsf{LIN\text{-}EXP}$*-hard problem A, a constant c_A exists such that no algorithm can solve A with a worst case running time smaller than $2^{n^{c_A}}$.*

In the following we shall outline the characterization of $\mathsf{LIN\text{-}EXP}$ as the class of languages decided by *alternating Turing machines* (in short, ATM) working with linear space. This characterization will be exploited in Section 7 in order to prove the EXP-hardness of bisimilarity in $\mathcal{L}_{m,r}$.

Definition 2.3 *An alternating Turing machine AT is a five-tuple $AT = (Q, q_0, g, \Sigma, \delta)$ where*

- *Q is the set of* states*;*
- *q_0 is the* initial state*;*
- *$g : Q \to \{\wedge, \vee, \mathbf{accept}, \mathbf{reject}\}$;*
- *Σ is the* tape alphabet*;*
- *$\delta \subseteq Q \times (\Sigma \cup \{\Box\}) \times Q \times \Sigma \times \{L, R\}$ is the* next move relation*.*

Where $\Box$ is a distinguished *blank* symbol, not belonging to Σ, that represents unused parts of the tape. The function g partitions Q into four sets: the set $Q_U = \{q \in Q | g(q) = \wedge\}$ of *universal states*, the set $Q_E = \{q \in Q | g(q) = \vee\}$ of *existential states*, the set $Q_A = \{q \in Q | g(q) = \mathbf{accept}\}$ of *accepting states*, and the set $Q_R = \{q \in Q | g(q) = \mathbf{reject}\}$ of *rejecting states*. Accepting and rejecting states are also called *halting* states.

Definition 2.4 *A* configuration *of an ATM AT is a string*

$$\bar{c} = (q_1, s_1, \ldots, q_n, s_n) \in ((Q \cup \{\perp\}) \cdot \Sigma)^*$$

such that exactly one index $j \in \{1, \ldots, n\}$ exists such that $q_j \neq \perp$.

Intuitively, $\bar{c} = (\perp, s_1, \ldots, \perp, s_{j-1}, q, s_j, \perp, s_{j+1}, \ldots, \perp, s_n)$ represents the global state of machine AT when n cells of the tape have been used, the head is on the j-th cell, the content of the tape is $s_1, \ldots, s_n$, and the finite control is in state q. We will denote by $\mathcal{GC}_{AT}$ the set of configurations of machine AT. A configuration is said to be *halting* (respectively, *existential*, *universal*) if it contains a halting (respectively, existential, universal) state. The *initial* configuration of AT with input $x = (x_1, \ldots, x_k)$ is $c_0(x) \stackrel{\text{def}}{=} (q_0, x_1, \perp, x_2, \ldots, \perp, x_k)$.

With a slight abuse of notation, we will denote by $\delta(\bar{c})$ the set of configurations $\bar{c}$' such that $\bar{c}$ can evolve in one step into $\bar{c}$' according to the relation δ. Whenever $\bar{c}' \in \delta(\bar{c})$ we will write

$\bar{c} \vdash \bar{c}'$. Let $\vdash^*$ be the transitive and reflexive closure of $\vdash$: we will denote by $\mathcal{GC}_{AT(x)}$ the set $\{\bar{c} \in \mathcal{GC}_{AT} |\ \bar{c}_0(x) \vdash^* \bar{c}\}$ and call it the *computation tree* of AT with input x. In this paper we shall only consider *time-bounded* ATM's, that is, machines having a finite computation tree for any input.

Acceptance is defined in a quite involved way for general ATM's (see [CKS81]). In the case of timed bounded ATM's, however, a much simpler inductive definition can be given.

Definition 2.5 (Acceptance) *Let AT be a time-bounded ATM, x be a string, $\bar{c} \in \mathcal{GC}_{AT(x)}$ be a configuration.*

1. *If $\bar{c}$ is a halting configuration, then we say that $\bar{c}$ is an* accepting *configuration if it contains an accepting state, otherwise we say that it is a* rejecting *configuration.*
2. *If $\bar{c}$ is an universal configuration, then we say that it is accepting if all the configurations in $\delta(\bar{c})$ are accepting, otherwise we say that it is rejecting.*
3. *If $\bar{c}$ is an existential configuration, then we say that it is accepting if at least one configuration in $\delta(\bar{c})$ is accepting, otherwise we say that it is rejecting.*

We say that AT *accepts* input x if the initial configuration of AT with input x is accepting. A language L is *decided* by an alternating Turing machine AT if AT accepts x if and only if $x \in L$. The following theorem has been proved by Chandra, Kozen, and Stockmeyer.

Theorem 2.6 ([CKS81]) *Every language $L \in$ LIN-EXP is decidable by an alternating Turing machine AT_L working with linear space and exponential time.*

Using standard techniques from the theory of Turing machines, we may assume without loss of generality that the AT_L of the above theorem's statement is such that, for any input x of size n, only the cells of the tape containing x are accessed, all the computation paths of $AT(x)$ have the same length and if $\bar{c} \in \mathcal{GC}_{AT(x)}$ is a universal (respectively, existential) configuration, then all the configurations in $\delta(\bar{c})$ are existential (respectively, universal). In the following, we shall call such a machine a *canonical linear-space alternating Turing machine.*

3 Basic Properties of the Language

In this section, we will define some concepts and fix some properties of the language which will be used throughout the rest of the paper. We will first define precisely subterms and then the *bisimulation up-to* proof technique; finally an alternative characterization of $\sim$ will be proven. The latter is the most relevant result of the section. Throughout the section, terms are assumed to be taken from the full language $\mathcal{L}_{m,v,r}$.

The standard definition of subterm has to be slightly extended to cope with identifiers.

Definition 3.1 (Subterms) *We say that P is a subterm of Q if $P \prec Q$, where $\prec$ is the smallest reflexive and transitive binary relation over $\mathcal{L}_{m,v,r}$ generated by the following axioms:*

P	$\prec$	$\alpha.P$	*for any action prefix α*
P	$\prec$	$[e_1 = e_2]P$	
P_i	$\prec$	$\sum_{i\in I} P_i$	*for each $i \in I$*
$P\sigma$	$\prec$	$Id(\tilde{y})\sigma$	*for each definition $Id(\tilde{y}) \Leftarrow P$ in Eq and for each substitution σ.*

Subterms enjoy the following properties, whose easy proofs are omitted. Here and in the sequel, $val(Eq)$ is $\{v : v$ appears in $Eq\}$.

Lemma 3.2 *Suppose that $P \prec Q$. Then*

1. $|P| \leq max\, \{|Q|\} \cup \{|R| \,:\, R$ *appears in* $Eq\}$;
2. $val(P) \subseteq val(Q) \cup val(Eq)$;
3. $P\sigma \prec Q\sigma$, *for any substitution* σ.

We now come to bisimulation. A first useful fact about it derives immediately from the rules of operational semantics.

Lemma 3.3 *For any identifier definition $Id(\tilde{y}) \Leftarrow P$ in Eq, $Id(\tilde{v}) \sim P\{\tilde{v}/\tilde{y}\}$.*

In the proof of reduction from $\mathcal{L}_{m,v,r}$ to $\mathcal{L}_{m,r}$, we will exploit a proof technique due to Milner. Intuitively, it allows us to match process derivatives "up to" bisimilarity. We indicate composition of relation by juxtaposition. Thus $P' \sim \mathcal{R} \sim Q'$ below means that there exist P'' and Q'' s.t. $P' \sim P''$ and $P''\mathcal{R}Q''$ and $Q'' \sim Q'$.

Definition 3.4 (Bisimulation up to $\sim$) *Let $\mathcal{R}$ be a symmetric binary relation over processes. $\mathcal{R}$ is a* bisimulation up to $\sim$ *if, whenever $P\,\mathcal{R}\,Q$ and $P \xrightarrow{\mu} P'$, there exists Q' such that $Q \xrightarrow{\mu} Q'$ and $Q' \sim \mathcal{R} \sim P'$.*

The proof of the following theorem, which states the soundness of the technique, is reported in [Mil89].

Theorem 3.5 *Let $\mathcal{R}$ be a bisimulation up to $\sim$ and let P, Q be two processes. Then $P\,\mathcal{R}\,Q$ implies $P \sim Q$.*

We will rely on a "finitary" characterization 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* for an agent term P if v does not occur in the P, nor in the set Eq.

Definition 3.6 (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 there exists a v_0 fresh for P and Q s.t.:*

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

We let $P \sim_F Q$ 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. It is worth to notice that the latter fact is not true in general. As an example, $B(0) \sim_F a$, where $B(x) \Leftarrow [x = 0]a$; but replacing 0 with 1 in $B(0)$, we obtain $B(1) \not\sim_F a$, because $B(1) \not\xrightarrow{a}$. Therefore, a certain care when dealing with such replacements is needed. Before proving the alternative characterization, we need a few properties of the transition system. In the following lemma and in the next theorem, we will suppose for simplicity that Var, Val and Act are disjoint (the name-passing case requires only notational changes, which are covered, e.g., in [MPW92]). Here and in the sequel, a tuple of values $\tilde{v}$ is fresh for an agent term P if each component of $\tilde{v}$ is fresh for P and all components are distinct.

Lemma 3.7 *Let P be an agent term with $fvar(P) \subseteq \tilde{x}$, and let $\tilde{v}$ and $\tilde{w}$ be two tuples of names fresh for P. Suppose that $P\{\tilde{v}/\tilde{x}\} \xrightarrow{\mu} P'$. Then:*

1. *if $\mu \in Act$ or $\mu = \overline{a}v_0$ or $\mu = a(v_0)$, with $v_0 \notin \tilde{v}$, then $P' = P_1\{\tilde{v}/\tilde{x}\}$ and $P\{\tilde{w}/\tilde{x}\} \xrightarrow{\mu} P_1\{\tilde{w}/\tilde{x}\}$, for some P_1;*
2. *if $\mu = \overline{a}v_i$ (resp. $\mu = a(v_i)$), with $v_i \in \tilde{v}$, then $P' = P_1\{\tilde{v}/\tilde{x}\}$ and $P\{\tilde{w}/\tilde{x}\} \xrightarrow{\overline{a}w_i} P_1\{\tilde{w}/\tilde{x}\}$ (resp. $P\{\tilde{w}/\tilde{x}\} \xrightarrow{a(w_i)} P_1\{\tilde{w}/\tilde{x}\}$), for some P_1;*
3. *if $\mu = a(v_0)$ and $v_0 \notin \tilde{v} \cup val(P, Eq)$ then $P' = P_1\{\tilde{v}v_0/\tilde{x}y\}$ for some $y \notin \tilde{x}$ and $P\{\tilde{w}/\tilde{x}\} \xrightarrow{a(w_0)} P_1\{\tilde{w}w_0/\tilde{x}y\}$, for any $w_0 \notin \tilde{w} \cup val(P, Eq)$, for some P_1.*

In all cases, $\tilde{v}$ and $\tilde{w}$ are fresh for P_1.

PROOF: By induction on the derivation of the transition $P\{\tilde{v}/\tilde{x}\} \xrightarrow{\mu} P'$. □

Theorem 3.8 (Alternative characterization of bisimulation) *$P \sim Q$ if and only if $P \sim_F Q$.*

PROOF: Clearly $P \sim Q$ implies $P \sim_F Q$, since the defining clauses for $\sim$ are stronger than those for $\sim_F$. We show now the converse. More precisely, we show that the relation

$$\mathcal{R} = \{(P\{\tilde{w}/\tilde{x}\}, Q\{\tilde{w}/\tilde{x}\}) \,:\, fvar(P,Q) \subseteq \tilde{x} \text{ and } P\{\tilde{v}/\tilde{x}\} \sim_F Q\{\tilde{v}/\tilde{x}\} \text{ for } \tilde{v},\, \tilde{w} \text{ fresh for } P \text{ and } Q\,\}$$

is a $\sim$-bisimulation. This fact implies the thesis for $\tilde{w} = \tilde{x} = \emptyset$. Suppose that $P\{\tilde{w}/\tilde{x}\}\,\mathcal{R}\,Q\{\tilde{w}/\tilde{x}\}$ and that $P\{\tilde{w}/\tilde{x}\} \xrightarrow{\mu} P'$, for any μ. We have to find a suitable "matching" transition from $Q\{\tilde{w}/\tilde{x}\}$. We can distinguish three cases:

1. $\mu \in Act$ or $\mu = \overline{a}w_0$ or $\mu = a(w_0)$, with $w_0 \in val(P, Q, Eq)$;
2. $\mu = \overline{a}w_i$ or $\mu = a(w_i)$, with $w_i \in \tilde{w}$;
3. $\mu = a(w_0)$ and $w_0 \notin \tilde{w} \cup val(P, Q, Eq)$.

Each of the cases 1-3 can be dealt with by relying on the corresponding part of Lemma 3.7. We concentrate here on case 3, as the other two are more easily dealt with. By definition of $\mathcal{R}$, it is $P\{\tilde{v}/\tilde{x}\} \sim_F Q\{\tilde{v}/\tilde{x}\}$, for some fresh $\tilde{v}$. According to the definition of $\sim_F$, there exists a v_0 fresh for $P\{\tilde{v}/\tilde{x}\}$ and $Q\{\tilde{v}/\tilde{x}\}$ s.t. every $a(v_0)$-move from $P\{\tilde{v}/\tilde{x}\}$ can be matched by $Q\{\tilde{v}/\tilde{x}\}$. We can suppose w.l.o.g. that $v_0 \notin \tilde{v}$ (the case $v_0 = v_i \in \tilde{v}$, which implies $x_i \notin fvar(P, Q)$, easily reduces to the case $v_0 \notin \tilde{v}$).

Now, applying Lemma 3.7.3 to the transition $P\{\tilde{w}/\tilde{x}\} \xrightarrow{a(w_0)} P'$, we have that $P' = P_1\{\tilde{w}w_0/\tilde{x}y\}$, for suitable P_1 and $y \notin \tilde{x}$ and furthermore that $P\{\tilde{v}/\tilde{x}\} \xrightarrow{a(v_0)} P_1\{\tilde{v}v_0/\tilde{x}y\}$. Since $P\{\tilde{v}/\tilde{x}\} \sim_F Q\{\tilde{v}/\tilde{x}\}$, we deduce that

$$Q\{\tilde{v}/\tilde{x}\} \xrightarrow{a(v_0)} Q_2 \sim_F P_1\{\tilde{v}v_0/\tilde{x}y\}.$$

Applying Lemma 3.7.3 to the above transition, we get that $Q_2 = Q_1\{\tilde{v}v_0/\tilde{x}y\}$, for some Q_1 s.t. $\tilde{v}$ is fresh for Q_1, and

$$Q\{\tilde{w}/\tilde{x}\} \xrightarrow{a(w_0)} Q_1\{\tilde{w}w_0/\tilde{x}y\} \stackrel{\text{def}}{=} Q'.$$

Now, it is easy to see that $P'\,\mathcal{R}\,Q'$: letting $\tilde{v}' \stackrel{\text{def}}{=} \tilde{v}v_0$, $\tilde{w}' \stackrel{\text{def}}{=} \tilde{w}w_0$ and $\tilde{x}' \stackrel{\text{def}}{=} \tilde{x}y$, we have that $P_1\{\tilde{v}'/\tilde{x}'\} \sim_F Q_1\{\tilde{v}'/\tilde{x}'\}$ and that $\tilde{v}'$ and $\tilde{w}'$ are fresh for P_1 and Q_1. These facts imply the wanted $P' = P_1\{\tilde{w}'/\tilde{x}'\}\,\mathcal{R}\,Q_1\{\tilde{w}'/\tilde{x}'\} = Q'$. □

4 Data-Independent Calculi

In this section we will deal with the complexity of the bisimilarity 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 then we will argue how the achieved results extend to the name-passing case.

Recall that in [KS90, PT87] it has been shown that the bisimilarity problem for finite labeled transition systems can be solved in time polynomial in the sizes of the systems. Therefore, in order to establish decidability of bisimilarity in a given language, it suffices to show how to reduce the problem to a bisimilarity problem over finite labeled transition systems. This reduction is shown to be possible for data-independent languages in [JP93].

Theorem 4.1 *The bisimilarity problems for $\mathcal{L}_v$ and $\mathcal{L}_r$ are in* P.

PROOF: From the results of [JP93] it follows that, given two data-independent processes P and Q we can construct two finite labeled transition systems G_P and G_Q such that $P \sim Q$ if and only if G_P is bisimilar to G_Q. Furthermore, for processes in $\mathcal{L}_v$, it is easy to see that the construction of Parrow and Walker can be carried out in polynomial-time in the syntactic sizes of P and Q.

Let us now consider $\mathcal{L}_r$. It is easy to see that if A is an identifier of arity n and $(u_1, \ldots, u_n)$, $(v_1, \ldots, v_n)$ are two n-tuples of values, then $A(u_1, \ldots, u_n) \sim A(v_1, \ldots, v_n)$. We can thus assume without loss of generality that every identifier in $\mathcal{L}_r$ has arity zero. Under this assumption, it easily follows that any term P has an associated labeled transition system whose size is polynomially bounded in the size of P and Eq.

The theorem follows from the above considerations and the results of [KS90, PT87]. $\square$

Let us now consider the complexity of the bisimilarity problem in $\mathcal{L}_{v,r}$. 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. 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*. A *truth assignment* for U is a function $t : U \to \{\mathtt{true}, \mathtt{false}\}$. If x is a variable in U, then x and $\neg x$ are said to be *literals* over U. The literal x is true under t if $t(x) = \mathtt{true}$ and is false otherwise, while the literal $\neg x$ is true under t if $t(x) = \mathtt{false}$ and is false otherwise.

A *(conjunctive) 3-clause* over U is the conjunction of three literals, e.g. $c = x_1 \wedge \neg x_3 \wedge \neg x_4$. A clause is true under a truth assignment t if all its literals are true under t. A *boolean formula in 3-disjunctive normal form* (in short, a *formula in 3DNF*) is the disjunction of a set of 3-clauses, e.g. $\phi = (x_1 \wedge \neg x_3 \wedge \neg x_4) \vee (x_2 \wedge \neg x_3 \wedge x_4)$. A formula is true under a truth assignment t if at least one of its clauses is true under t. A formula is a *tautology* if it is true under *any* truth assignment. With a slight abuse of notation, we will admit that a literal may also be a member of the set $\{\mathtt{true}, \mathtt{false}, \neg\mathtt{true}, \neg\mathtt{false}\}$. 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 $b_i \in \{\mathtt{true}, \mathtt{false}\}$, be the formula obtained from ϕ by substituting b_i to x_i for $i = 1, \ldots, k$.

Definition 4.2 (Quantified boolean formula) *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 4.3 (Validity) *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:*

$$
\begin{array}{rcl}
E_n & \Leftarrow & \sum_{\substack{(v_1,v_2,v_3)\in\{\mathtt{true},\mathtt{false}\}^3 \\ (v_1,v_2,v_3)\neq(\mathtt{true},\mathtt{true},\mathtt{true})}} \overline{a}v_1.\overline{a}v_2.\overline{a}v_3 \\
T_n & \Leftarrow & \sum_{(v_1,v_2,v_3)\in\{\mathtt{true},\mathtt{false}\}^3} \overline{a}v_1.\overline{a}v_2.\overline{a}v_3 \\
B_n(y_1,\ldots,y_n,z_1,\ldots,z_n) & \Leftarrow & \sum_{i=1}^m \overline{a}w_{i_1}.\overline{a}w_{i_2}.\overline{a}w_{i_3}+ \\
 & & \sum_{\substack{(v_1,v_2,v_3)\in\{\mathtt{true},\mathtt{false}\}^3 \\ (v_1,v_2,v_3)\neq(\mathtt{true},\mathtt{true},\mathtt{true})}} \overline{a}v_1.\overline{a}v_2.\overline{a}v_3 \\
\multicolumn{3}{l}{\text{For any even } i,\ 0\le i\le n-2:} \\
B_i(y_1,\ldots,y_i,z_1,\ldots,z_i) & \Leftarrow & a.B_{i+1}(y_1,\ldots,y_i,\mathtt{true},z_1,\ldots,z_i,\mathtt{false})+ \\
 & & a.B_{i+1}(y_1,\ldots,y_i,\mathtt{false},z_1,\ldots,z_i,\mathtt{true})+a.E_{i+1} \\
T_i & \Leftarrow & a.T_{i+1}+a.E_{i+1} \\
E_i & \Leftarrow & a.E_{i+1} \\
\multicolumn{3}{l}{\text{For any odd } i,\ 1\le i\le n-1:} \\
B_i(y_1,\ldots,y_i,z_1,\ldots,z_i) & \Leftarrow & a.B_{i+1}(y_1,\ldots,y_i,\mathtt{true},z_1,\ldots,z_i,\mathtt{false})+ \\
 & & a.B_{i+1}(y_1,\ldots,y_i,\mathtt{false},z_1,\ldots,z_i,\mathtt{true})+a.T_{1+1} \\
T_i & \Leftarrow & a.T_{i+1} \\
E_i & \Leftarrow & a.E_{i+1}+a.T_{i+1}
\end{array}
$$

Table 3: The reduction from RQBF to bisimilarity in $\mathcal{L}_{v,r}$.

1. *$n = 0$ and in ϕ' there is a true clause c;*
2. *$Q_1 = \exists$ and either $Q_2x_2\ldots,Q_nx_n.\phi'\{\mathtt{true}/x_1\}$ or $Q_2x_2\ldots,Q_nx_n.\phi'\{\mathtt{false}/x_1\}$ is valid;*
3. *$Q_1 = \forall$ and both $Q_2x_2\ldots,Q_nx_n.\phi'\{\mathtt{true}/x_1\}$ and $Q_2x_2\ldots,Q_nx_n.\phi'\{\mathtt{false}/x_1\}$ are 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 $\phi = Q_1x_1Q_2x_2\ldots,Q_nx_n.\phi'$ such that n is even and $Q_i = \exists$ if and only if Q_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 definition of 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 4.4 *For any i, $0 \le i \le n$ and for any $(v_1,\ldots,v_i) \in \{\mathtt{true},\mathtt{false}\}^i$, $B_i(v_1,\ldots,v_i,\neg v_1,\ldots,\neg v_i)$ is either bisimilar to E_i or bisimilar to T_i.*

Proof: We will prove the statement by induction on $k = n - i$. For $i = n$, let $(v_1,\ldots,v_n) \in \{\mathtt{true},\mathtt{false}\}^n$; if $B_n(v_1,\ldots,v_n,\neg v_1,\ldots,\neg v_n)$ can perform the sequence of actions $\overline{a}\mathtt{true}.\overline{a}\mathtt{true}.\overline{a}\mathtt{true}$, then it is bisimilar to T_n, otherwise it is clearly bisimilar to E_n.

Suppose now that for any $(v_1,\ldots,v_{i+1}) \in \{\mathtt{true},\mathtt{false}\}^{i+1}$ we have that $B_{i+1}(v_1,\ldots,v_{i+1},\neg v_1,\ldots,\neg v_{i+1})$ is either bisimilar to T_{i+1} or to E_{i+1}, we will prove the statement for i. If we consider any $(v_1,\ldots,v_i) \in \{\mathtt{true},\mathtt{false}\}^i$ and we look at the possible transitions for the process $B \stackrel{\mathrm{def}}{=} B_i(v_1,\ldots,v_i,\neg v_1,\ldots,\neg v_i)$, then there are two possibilities.

1. If i is odd, B can evolve into $B' \stackrel{\text{def}}{=} B_{i+1}(v_1, \ldots, v_i, \texttt{true}, \neg v_1, \ldots, \neg v_i, \texttt{false})$ or into $B'' \stackrel{\text{def}}{=} B_{i+1}(v_1, \ldots, v_i, \texttt{false}, \neg v_1, \ldots, \neg v_i, \texttt{true})$ or into T_{i+1} doing action a. If B' and B'' are both bisimilar to T_{i+1}, then $B \sim T_i$, otherwise B' or B'' is bisimilar to E_{i+1} and thus $B \sim E_i$.

2. If i is even, then B can evolve into $B' \stackrel{\text{def}}{=} B_{i+1}(v_1, \ldots, v_i, \texttt{true}, \neg v_1, \ldots, \neg v_i, \texttt{false})$ or into $B'' \stackrel{\text{def}}{=} B_{i+1}(v_1, \ldots, v_i, \texttt{false}, \neg v_1, \ldots, \neg v_i, \texttt{true})$ or into E_{i+1} doing action a. If B' and B'' are both bisimilar to E_{i+1}, then $B \sim E_i$, otherwise B' or B'' is bisimilar to T_{i+1} and thus $B \sim T_i$.

$\square$

Lemma 4.5 *For any i, $0 \leq i \leq n$, $T_i \not\sim E_i$.*

Proof: We proceed again by induction on $n - i$. $E_n \not\sim T_n$ because T_n can execute the sequence of actions $\overline{a}\texttt{true}.\overline{a}\texttt{true}.\overline{a}\texttt{true}$, while E_n cannot. Assuming $E_{i+1} \not\sim T_{i+1}$, then

1. if i is odd, then $E_i \stackrel{a}{\longrightarrow} E_{i+1}$, and the only possible transition for T_i is $T_i \stackrel{a}{\longrightarrow} T_{i+1}$, thus $E_i \not\sim T_i$;

2. if i is even, then $T_i \stackrel{a}{\longrightarrow} T_{i+1}$, and the only possible transition for E_i is $E_i \stackrel{a}{\longrightarrow} E_{i+1}$, thus $E_i \not\sim T_i$.

$\square$

Lemma 4.6 *Consider any i, $0 \leq i \leq n$ and any $(v_1, \ldots, v_i) \in \{\texttt{true}, \texttt{false}\}^i$. Let*

- $\psi' \stackrel{\text{def}}{=} \phi'\{v_1/x_1, \ldots, v_i/x_i\}$;
- $\psi \stackrel{\text{def}}{=} Q_{i+1}x_{i+1} \ldots \forall x_n.\psi'$, *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: As in the previous lemmas, 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 and only if $Q_{i+2}x_{i+2}, \ldots, \forall x_n.\psi'\{v_{i+1}/x_{i+1}\}$ is valid. Again, we have to distinguish two cases.

1. If i is even, then $Q_{i+1} = \exists$. Due to Lemmas 4.4 and 4.5, we have that $B_i(v_1, \ldots, v_i, \neg v_1, \ldots, \neg v_i) \sim T_i$ if and only if a value $v_{i+1} \in \{\texttt{true}, \texttt{false}\}$ exists such that $B_{i+1}(v_1, \ldots, v_{i+1}, \neg v_1, \ldots, \neg v_{i+1}) \sim T_{i+1}$. By inductive hypothesis, the latter holds if and only if either $\forall x_{i+2} \ldots \forall x_n.\psi'\{x_{i+1}/\texttt{true}\}$ is valid or $\forall x_{i+2} \ldots \forall x_n.\psi'\{x_{i+1}/\texttt{false}\}$ is valid, that is, if and only if $\psi = \exists x_{i+1} \ldots \forall x_n.\psi'$ is valid.

2. If i is odd, then $Q_{i+1} = \forall$. Due to Lemmas 4.4 and 4.5, we have that $B_i(v_1, \ldots, v_i, \neg v_1, \ldots, \neg v_i) \sim T_i$ if and only if both $B_{i+1}(v_1, \ldots, v_i, \texttt{true}, \neg v_1, \ldots, \neg v_i, \texttt{false})$ and $B_{i+1}(v_1, \ldots, v_i, \texttt{false}, \neg v_1, \ldots, \neg v_i, \texttt{true})$ are bisimilar to T_i. By inductive hypothesis, the latter holds if and only if both $\exists x_{i+2} \ldots \forall x_n.\psi'\{x_{i+1}/\texttt{true}\}$ and $\exists x_{i+2} \ldots \forall x_n.\psi'\{x_{i+1}/\texttt{false}\}$ are valid, that is, if and only if $\psi = \forall x_{i+1} \ldots \forall x_n.\psi'$ is valid.

$\square$

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

$$
\begin{aligned}
P(\phi) &\stackrel{\text{def}}{=} a(y_1)\dots a(y_n).P' \\
Q &\stackrel{\text{def}}{=} a(y_1)\dots a(y_n).Q' \\
Q' &\stackrel{\text{def}}{=} a + a.a \\
P' &\stackrel{\text{def}}{=} a.a + R + \textstyle\sum_{i=1}^{m} P_i, \text{ where:} \\
R &\stackrel{\text{def}}{=} \textstyle\sum_{i=1}^{n} a.([y_i = \texttt{true}]a + [y_i = \texttt{false}]a) \\
P_i &\stackrel{\text{def}}{=} [y_{i_1} = b_{i_1}][y_{i_2} = b_{i_2}][y_{i_3} = b_{i_3}]a \text{ for } i \in \{1, \dots, m\}
\end{aligned}
$$

Table 4: The reduction from 3-Tautology to bisimilarity in $\mathcal{L}_{m,v}$.

Corollary 4.7 *$B_0 \sim T_0$ 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 4.8 *The bisimilarity problem in $\mathcal{L}_{v,r}$ is* PSPACE*-hard*

Let us now briefly consider the name-passing case arising when $Act = Val = Var$. The bisimilarity problem in $\mathcal{L}_v$ can be proved to be in P using the same argument of the proof of Theorem 4.1. The results given by Jonsson and Parrow are stated for the simple value passing case, however it is not hard to check that the proofs carry over in the name passing case. A more interesting case arises with $\mathcal{L}_r$. Indeed we can assume that $\{\texttt{true}, \texttt{false}\} \subset Var = Val = Act$ and then repeat the PSPACE-hardness proof for $\mathcal{L}_r$ by replacing the output actions $\overline{a}v$ with the simple action v. The PSPACE-hardness result clearly extends to $\mathcal{L}_{v,r}$. Decidability follows by an easy extension of Jonsson's and Parrow's results.

5 Data-Dependent Value-Passing

In this section we will show that the bisimilarity problem for the calculus $\mathcal{L}_{m,v}$ 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 5.1 *The bisimilarity problem in $\mathcal{L}_{m,v}$ is* coNP*-hard.*

Proof: It is sufficient to prove that the 3-Tautology problem is polynomial-time reducible to the bisimilarity problem in $\mathcal{L}_{m,v}$. Let $\phi = c_1 \vee \dots \vee c_m$ be an instance of 3-Tautology over the set of variables $\{x_1, \dots, x_n\}$, let $c_i = l_i^1 \wedge l_i^2 \wedge l_i^3$ for $i = 1, \dots, m$, 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. We will first prove that ϕ is a tautology if and only if, for any $(v_1, \dots, v_n) \in Val^n$, $P'\{v_1/y_1, \dots, v_n/y_n\} \sim Q'$. From this fact, it easily follows that $Q \sim P(\phi)$ if and only if ϕ is a tautology. Note that $P'\{v_1/y_1, \dots, v_n/y_n\} \sim Q'$ if and only if one of its summands is equivalent to a.

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 μ is an input action then $\mu = a(v)$, for some a $\}$;
  $J := \{(\mu, Q') \mid Q \xrightarrow{\mu} Q'$ and if μ 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

Figure 1: A nondeterministic algorithm for detecting inequivalence of processes in $\mathcal{L}_{m,v}$.

Let ϕ be a tautology, let $(v_1, \ldots, v_n) \in Val^n$. If $(v_1, \ldots, v_n) \notin \{\mathtt{true}, \mathtt{false}\}^n$, then one of the summands of the term R is equivalent to a, and thus $P'\{v_1/y_1, \ldots, v_n/y_n\} \sim Q'$. If, instead, $(v_1, \ldots, v_n) \in \{\mathtt{true}, \mathtt{false}\}^n$, then consider the truth assignment t such that $t(x_i) = v_i$ for $i = 1, \ldots, n$. Since ϕ is a tautology, then ϕ is true under t, that is, a clause c_j exists such that c_j is true under t. Thus, it is easy to see that $P_j\{v_1/y_1, \ldots, v_n/y_n\}$ is equivalent to a, and hence $P'\{v_1/y_1, \ldots, v_n/y_n\}$ is equivalent to Q'.

Assume now that, for any $(v_1, \ldots, v_n) \in Val^n$, $P'\{v_1/y_1, \ldots, v_n/y_n\} \sim Q'$. Consider any truth assignment t for $\{x_1, \ldots, x_n\}$, and let $(v_1, \ldots, v_n) \in Val^n$ be defined such that $v_i \stackrel{\mathrm{def}}{=} t(x_i)$. One of the summands of $P'\{v_1/y_1, \ldots, v_n/y_n\}$ is equivalent to a, and it cannot be any of the summands in R (they are all equivalent to $a.a$). Thus, a P_j exists such that $P_j\{v_1/y_1, \ldots, v_n/y_n\}$ is equivalent to a: it follows that the clause c_j is true under t. We have thus shown that for any truth assignment t, at least one of the clauses of ϕ is true under t, and this implies that ϕ is a tautology. $\square$

Theorem 5.2 *The bisimilarity problem in* $\mathcal{L}_{m,v}$ *is in* coNP.

PROOF: We will prove that the *inequivalence* problem (given P, Q in $\mathcal{L}_{m,v}$, decide whether $P \not\sim Q$) is in NP. Let us consider the nondeterministic algorithm in Figure 1. We will prove that:

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.

Runs in polynomial time. Consider the procedure $B(P,Q)$. We prove by induction on $|P| + |Q|$ that $B(P,Q)$ runs in a time $\mathcal{O}(|P| * |Q|)$. Note that the sum of the sizes of all terms present in the set I is less or equal than $|P|$; indeed each action prefix (including the input ones) enabled in P gives rise to exactly one pair in I. Similarly for J and Q. Now $B(P,Q)$ runs in a time

$$\mathcal{O}\Big(\sum_{(\,(\mu,P'),(\mu,Q')\,) \in I \times J} T(P', Q') \Big)$$

where $T(P', Q')$ is the running time of $B(P', Q')$. By inductive hypothesis, the latter is $\mathcal{O}(|P'|*|Q'|)$. Thus, the running time of $B(P,Q)$ is $\mathcal{O}(\sum_{((\mu,P'),(\mu,Q'))\in I\times J} |P'| * |Q'|)$. Rearranging summands, the latter is re-written as $\mathcal{O}(\sum_{(\mu,P')\in I} |P'| * (\sum_{(\mu,Q')\in J} |Q'|))$; due to the above stated facts on I and J, this expression is less or equal than $\sum_{(\mu,P')\in I} |P'| * |Q|$, which is in turn less or equal than $|P| * |Q|$.

Always rejects bisimilar processes. Suppose $P \sim Q$. We will prove that `Non-equiv` always rejects with input (P, Q), i.e. that all computations of $B(P,Q)$ yield `true`, by induction on $|P|+|Q|$. Since $P \sim Q$, it follows by definition that for any $(\mu, P') \in I$, there is $(\mu, Q') \in J$ such that $P' \sim Q'$; by inductive hypothesis, the latter implies that $B(P', Q') =$ `true`. Similarly, for any $(\mu, Q') \in J$, there is $(\mu, P') \in I$ such that $B(P', Q') =$ `true`. It thus follows that $B(P,Q) =$ `true` (note that a formula universally quantified over an empty set is always true).

Non-deterministically accepts non-bisimilar processes. Let $P \not\sim Q$. We will prove that there is a computation of $B(P,Q)$ that yields `false`, by induction on $|P| + |Q|$. We rely on the finitary characterization of $\sim$, Definition 3.6. It is easy to see that, when determining whether $P \not\sim_F Q$, the fresh value v_0 to check in the input action clause can be chosen arbitrarily. More precisely, by exploiting Lemma 3.7, one can prove the following. Suppose that $P \not\sim_F Q$ and fix *any* fresh v_0. Then one of the following two cases arises:

1. an action μ, with $val(\mu) \subseteq val(P,Q) \cup \{v_0\}$, and a process P' exist such that $P \xrightarrow{\mu} P'$ and for any process Q' such that $Q \xrightarrow{\mu} Q'$, $P' \not\sim Q'$, or

2. an action μ, with $val(\mu) \subseteq val(P,Q) \cup \{v_0\}$, and a process Q' exist such that $Q \xrightarrow{\mu} Q'$ and for any process P' such that $P \xrightarrow{\mu} P'$, $P' \not\sim Q'$.

Let us assume that first case holds (the second one is perfectly symmetrical). There are two sub-cases, either μ is not an input action or $\mu = a(v)$, for some $v \in val(P,Q) \cup \{v_0\}$. We deal only with the latter, since the former is easier. Consider now the computation of $B(P,Q)$ where v is guessed. From 1. above, we have that for each $(\mu, Q') \in J$, $P' \not\sim Q'$. Thus, by inductive hypothesis, for any such (μ, Q') there is a computation of $B(P', Q')$ s.t. $b(\mu, P', Q') =$ `false`. This implies that there is a computation of $B(P,Q)$ which returns `false`.

□

Corollary 5.3 *The bisimilarity problem in $\mathcal{L}_{m,v}$ is* coNP*-complete.*

6 Reducing Value-passing to Identifiers and Matching

We will exhibit a polynomial-time reduction of bisimilarity in $\mathcal{L}_{m,v,r}$ to bisimilarity in $\mathcal{L}_{m,r}$. 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 will 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. The basic idea stems from Definition 3.6 and from Milner's translation of CCS with values into pure CCS with infinite summation [Mil89]. As a first approximation, we 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. 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. Thus, we 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.

$[\![P]\!]$ is defined as:

$$\begin{array}{rcl}
[\![c.P]\!] & = & c.[\![P]\!] \\[1ex]
[\![\overline{a}v.P]\!] & = & \overline{av}.[\![P]\!] \\[1ex]
[\![\overline{a}x.P]\!] & = & \sum_{v\in V_0}[x=v]\overline{av}.A(\tilde{y}) \\
& & \text{where } \tilde{y} = fvar([\![P]\!]) \\[1ex]
[\![a(x).P]\!] & = & \sum_{v\in V_0} av.A(\tilde{y},v) \\
& & \text{where } \tilde{y} = fvar([\![P]\!]) - \{x\} \\[1ex]
[\![[e_1=e_2]P]\!] & = & [e_1=e_2][\![P]\!] \\[1ex]
[\![\sum_{i\in I} P_i]\!] & = & \sum_{i\in I}[\![P_i]\!] \\[1ex]
[\![Id(\tilde{e})]\!] & = & Id(\tilde{e})
\end{array}$$

$\mathcal{D}(P)$ is defined as:

$$\begin{array}{rcl}
\mathcal{D}(c.P) & = & \mathcal{D}(P) \\[1ex]
\mathcal{D}(\overline{a}v.P) & = & \mathcal{D}(P) \\[1ex]
\mathcal{D}(\overline{a}x.P) & = & \{A(\tilde{y}) \Leftarrow [\![P]\!]\} \cup \mathcal{D}(P) \\[1ex]
\mathcal{D}(a(x).P) & = & \{A(\tilde{y},x) \Leftarrow [\![P]\!]\} \cup \mathcal{D}(P) \\[1ex]
\mathcal{D}([e_1=e_2]P) & = & \mathcal{D}(P) \\[1ex]
\mathcal{D}(\sum_{i\in I} P_i) & = & \bigcup_{i\in I}\mathcal{D}(P_i) \\[1ex]
\mathcal{D}(Id(\tilde{e})) & = & \emptyset
\end{array}$$

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

We come now to describing the actual translation. 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 letters $A, A', \ldots$. We assume that each input action $a(v)$ (resp. output action $\overline{a}v$) is injectively associated a pure action av, (resp. $\overline{av}$). We can suppose that, given any finite set of processes in $\mathcal{L}_{m,r}$ and any action $a(v)$ or $\overline{a}v$, pure actions av and $\overline{av}$ are distinct from any pure action occurring in the given set of processes. The translation consists of two parts: for each term P in $\mathcal{L}_{m,v,r}$, we have to specify, in $\mathcal{L}_{m,r}$, 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 6. The definition 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. In the sequel, $\mathcal{D}(P) \cup \mathcal{D}(Q)$ is abbreviated as $\mathcal{D}(P, Q)$.

Remark 6.1 In the clauses of input prefix and in the clauses of output prefix with variable, the auxiliary identifier A appearing in the definition of $[\![P]\!]$ is assumed to be the same as the one appearing in the definition of $\mathcal{D}(P)$. Furthermore, it is assumed that each such identifier A appears in $[\![P]\!]$ and is defined in $\mathcal{D}(P)$ at most once. Similarly, we assume that, given any two agent terms, P and Q, their encodings $[\![P]\!]$ and $[\![Q]\!]$ do not share any auxiliary identifier.

The above requirements can always be fulfilled by suitable renamings of auxiliary identifiers. Formally, they could have been incorporated in the definitions by fixing some total ordering of auxiliary identifiers; then, whenever a new auxiliary identifier would be needed, the least not yet used identifier would be picked up.

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

Definition 6.2 *Let us define* $[\![Eq]\!]$ *as* $\bigcup_{Id(\tilde{y})\Leftarrow P\, \in Eq}(\{Id(\tilde{y}) \Leftarrow [\![P]\!]\} \cup \mathcal{D}(P))$.

The reduction proof is split in two parts: *completeness* (if $P \sim Q$ in $\mathcal{L}_{m,v,r}$, then their translations are bisimilar in $\mathcal{L}_{m,r}$) and *correctness* (if the translations of P and Q are bisimilar in $\mathcal{L}_{m,r}$, then P and Q are bisimilar in $\mathcal{L}_{m,v,r}$). Before proving these results, we state some facts which will be useful in the sequel.

The following lemma, whose easy proof is omitted, sums up some elementary properties of the translation:

Lemma 6.3

1. *For any term P, $fvar(P) = fvar(\llbracket P \rrbracket)$.*
2. *Let σ be a substitution. Then $\llbracket P \rrbracket\sigma = \llbracket P\sigma \rrbracket$;*
3. *If $P' \prec P$ then $\llbracket P' \rrbracket \prec \llbracket P \rrbracket$ and $\mathcal{D}(P') \subseteq \mathcal{D}(P)$.*

The key for proving the correctness and completeness of our translation is given by the next proposition. It relates the transitions of P to those of $\llbracket P \rrbracket$. Recall that, in its statement, V_0 is the set of values w.r.t. which our encoding is parameterized.

Proposition 6.4 (Operational correspondence) *Let P be a term and σ be a substitution whose variables are $fvar(P)$, so that $P\sigma$ is closed. Let $\mathcal{L}_{m,r}$ be equipped with a set of identifiers definitions containing $\mathcal{D}(P) \cup \llbracket Eq \rrbracket$.*

1. *$P\sigma \xrightarrow{a} P'$ implies $\llbracket P \rrbracket\sigma \xrightarrow{a} \llbracket P' \rrbracket$. Vice-versa, if $\llbracket P \rrbracket\sigma \xrightarrow{a} P_1$ then $P_1 = \llbracket P' \rrbracket$ and $P\sigma \xrightarrow{a} P'$, for some P'.*
2. *$P\sigma \xrightarrow{\overline{a}v} P'$ implies $\llbracket P \rrbracket\sigma \xrightarrow{\overline{a}v} \sim \llbracket P' \rrbracket\sigma$. Vice-versa, if $\llbracket P \rrbracket\sigma \xrightarrow{\overline{a}v} P_1$ then $P_1 \sim \llbracket P' \rrbracket\sigma$ and $P\sigma \xrightarrow{\overline{a}v} P'$, for some P'.*
3. *$\llbracket P \rrbracket\sigma \xrightarrow{av} P_1$ implies $P_1 \sim \llbracket P' \rrbracket\sigma\{v/x\}$, for some $P' \prec P$ and x s.t. $P\sigma \xrightarrow{a(v)} P'\sigma\{v/x\}$.*
4. *$P\sigma \xrightarrow{a(v)} P_1$, with $v \in V_0$, implies $P_1 = P'\sigma\{v/x\}$, for some $P' \prec P$ and x s.t. $\llbracket P \rrbracket\sigma \xrightarrow{av} \sim \llbracket P' \rrbracket\sigma\{v/x\}$.*

PROOF: By induction on the derivation of the transitions. Parts 1 and 2 are trivial, while parts 3 and 4 are similar to each other. As an example, we check item 3.

The only non-trivial cases are those in which the last rule applied is Inp or Ide. Below, we examine these two cases in detail.

Case $P = a(x).P'$. By definition of translation, we have:

$$\llbracket P \rrbracket\sigma = (\sum_{v \in V_0} av.A(\tilde{x}, v))\sigma = \sum_{v \in V_0} av.A(\tilde{x}\sigma, v) \xrightarrow{av} A(\tilde{x}\sigma, v) \stackrel{\text{def}}{=} P_1$$

with $\tilde{x} = fvar(\llbracket P' \rrbracket) - \{x\}$ and $A(\tilde{x}, x) \Leftarrow \llbracket P' \rrbracket$ in $\mathcal{D}(P)$. By Lemma 3.3, in $\mathcal{L}_{m,r}$ we have

$$P_1 = A(\tilde{x}\sigma, v) = A(\tilde{x}, x)\sigma\{v/x\} \sim \llbracket P' \rrbracket\sigma\{v/x\}.$$

Furthermore it is $P' \prec P$. Finally, applying the Inp rule, we have:

$$P\sigma = a(x).P'\sigma \xrightarrow{a(v)} P'\sigma\{v/x\}$$

which concludes the case.

Case $P = Id(\tilde{e})$, with $Id(\tilde{y}) \Leftarrow P'$ in Eq. For a suitable substitution σ', with variables in $\tilde{y}$, we can write $Id(\tilde{e}) = Id(\tilde{y})\sigma'$. Recall now that $[\![Id(\tilde{e})]\!] = Id(\tilde{e}) = Id(\tilde{y})\sigma'$ and that $Id(\tilde{y}) \Leftarrow [\![P']\!]$ is in $\mathcal{D}(D)$. By hypothesis the last step in the derivation of the transition is:

$$Ide\frac{[\![P']\!]\sigma'\sigma \xrightarrow{av} P_1}{Id(\tilde{y})\sigma'\sigma = [\![Id(\tilde{e})]\!]\sigma \xrightarrow{av} P_1}.$$

Since $\mathcal{D}(P') \subseteq \mathcal{D}(Eq)$, by inductive hypothesis, we have in $\mathcal{L}_{m,r}$:

$$P_1 \sim [\![P'']\!]\sigma'\sigma\{v/x\} \tag{1}$$

for some $P'' \prec P'$ and x, and furthermore $P'\sigma'\sigma \xrightarrow{a(v)} P''\sigma'\sigma\{v/x\}$ in $\mathcal{L}_{m,v,r}$. From the latter, applying Ide, we get in $\mathcal{L}_{m,v,r}$:

$$Id(\tilde{e})\sigma \xrightarrow{a(v)} P''\sigma'\sigma\{v/x\}.$$

Now, $P'' \prec P'$ implies $P'' \prec Id(\tilde{y})$ and therefore (Lemma 3.2.2) $P''\sigma' \prec Id(\tilde{e})$. Furthermore, (Lemma 6.3.2) $[\![P'']\!]\sigma' = [\![P''\sigma']\!]$, thus from (1) we get $P_1 \sim [\![P''\sigma']\!]\sigma\{v/x\}$. To sum up, we have obtained: $Id(\tilde{e})\sigma \xrightarrow{a(v)} (P''\sigma')\sigma\{v/x\}$, with $P''\sigma' \prec Id(\tilde{e})$ and $P_1 \sim [\![P''\sigma']\!]\sigma\{v/x\}$, that is the wanted statement. □

Theorem 6.5 (Completeness of the translation) *For any two processes P_0 and Q_0 in $\mathcal{L}_{m,v,r}$, $P_0 \sim Q_0$ implies $[\![P_0]\!] \sim [\![Q_0]\!]$ in $\mathcal{L}_{m,r}$ equipped with $\mathcal{D}(P_0, Q_0) \cup [\![Eq]\!]$.*

Proof: Over $\mathcal{L}_{m,r}$, consider the relation $\mathcal{R}$ so defined:

$$\{([\![P]\!]\sigma_1, [\![Q]\!]\sigma_2) \,:\, [\![P]\!]\sigma_1 \text{ and } [\![Q]\!]\sigma_2 \text{ are closed, } P \prec P_0,\, Q \prec Q_0 \text{ and } P\sigma_1 \sim Q\sigma_2 \text{ in } \mathcal{L}_{m,v,r}\}.$$

We will show that $\mathcal{R}$ is a bisimulation up to $\sim$, hence $\mathcal{R} \subseteq \sim$: this fact implies the thesis.

Let $[\![P]\!]\sigma_1 \,\mathcal{R}\, [\![Q]\!]\sigma_2$. We can suppose w.l.o.g. that the variables of σ_1 are $fvar(P)$ and that the variables of σ_2 are $fvar(Q)$. Note that, in virtue of Lemma 6.3.3, $\mathcal{D}(P_0, Q_0)$ contains $\mathcal{D}(P, Q)$. Suppose that $[\![P]\!]\sigma_1 \xrightarrow{\mu} P_1$. We have to show that there exist Q_1, P', Q', σ_1' and σ_2' s.t. $[\![Q]\!]\sigma_2 \xrightarrow{\mu} Q_1$ and $P_1 \sim [\![P']\!]\sigma_1' \,\mathcal{R}\, [\![Q']\!]\sigma_2' \sim Q_1$. By definition, $[\![P']\!]\sigma_1' \,\mathcal{R}\, [\![Q']\!]\sigma_2'$ means:

$$P'\sigma_1' \text{ and } P'\sigma_1' \text{ are closed, } P' \prec P_0,\, Q' \prec Q_0 \text{ and } P'\sigma_1' \sim Q'\sigma_2'.$$

We treat only the case when μ is of the form av, since the other cases are much easier. Now, $[\![P]\!]\sigma_1 \xrightarrow{av} P_1$ implies that $v \in V_0$ and (Prop. 6.4.3)

$$P_1 \sim [\![P']\!]\sigma_1\{v/x\} \text{ with } P' \prec P \tag{2}$$

and, furthermore:

$$P\sigma_1 \xrightarrow{a(v)} P'\sigma_1\{v/x\}\,.$$

Note that $P'\sigma_1\{v/x\}$ must be closed. Since $P\sigma_1 \sim Q\sigma_2$, from the above transition it follows:

$$Q\sigma_2 \xrightarrow{a(v)} Q_2 \text{ with } P'\sigma_1\{v/x\} \sim Q_2. \tag{3}$$

Applying Proposition 6.4.4 to the above transition, we get:

$$Q_2 = Q'\sigma_2\{v/y\} \text{ with } Q' \prec Q \tag{4}$$

where $Q'\sigma_2\{v/y\}$ must be closed, and furthermore:

$$[\![Q]\!]\sigma_2 \xrightarrow{av} Q_1 \text{ with } Q_1 \sim [\![Q']\!]\sigma_2\{v/y\}\,. \tag{5}$$

We prove now that the latter is the wanted matching transition. Define $\sigma_1' = \sigma_1\{v/x\}$ and $\sigma_2' = \sigma_2\{v/y\}$. Note that $P' \prec P \prec P_0$ implies $P' \prec P_0$, and similarly $Q' \prec Q_0$; moreover, from (3) and (4), we have $P'\sigma_1' \sim Q_2 = Q'\sigma_2'$, thus by definition $[\![P']\!]\sigma_1' \,\mathcal{R}\, [\![Q']\!]\sigma_2'$. Finally, from (2) and (5), $P_1 \sim [\![P']\!]\sigma_1'$ and $Q_1 \sim [\![Q']\!]\sigma_2'$, which closes up the bisimulation. □

We now come to the correctness part. This is slightly more difficult, because we have to choose appropriately the 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. The latter must be polynomial in the size of the problem but large enough to provide fresh values for input transitions at each stage of the processes' evolution.

Theorem 6.6 (Correctness of the translation) *Let P_0 and Q_0 be processes in $\mathcal{L}_{m,v,r}$. For a polynomial-size choice of the parameter V_0, if $[\![P_0]\!] \sim [\![Q_0]\!]$ in $\mathcal{L}_{m,r}$ equipped with $\mathcal{D}(P_0,Q_0) \cup [\![Eq]\!]$ then $P_0 \sim Q_0$.*

Proof: 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 $|V| = 2*k+1$, where $k = max(\{|P_0|, |Q_0|\} \cup \{|P| :$ P appears in $Eq\})$.

Over $\mathcal{L}_{m,v,r}$, define the relation $\mathcal{R}$ as follows:

$$\begin{array}{ll}\{(P\sigma_1, Q\sigma_2): & P\sigma_1 \text{ and } Q\sigma_2 \text{ are closed}, P \prec P_0, Q \prec Q_0,\ val(\sigma_1,\sigma_2) \subseteq V_0 \\ & \text{and } [\![P]\!]\sigma_1 \sim [\![Q]\!]\sigma_2 \text{ in } \mathcal{L}_{m,r}\}.\end{array}$$

We will show that $\mathcal{R}$ is an F-bisimulation: this fact and Theorem 3.8 imply the thesis. Let $P\sigma_1 \,\mathcal{R}\, Q\sigma_2$. We can suppose w.l.o.g. that the variables of σ_1 are $fvar(P)$ and that the variables of σ_2 are $fvar(Q)$. We only check the input clause (the second one) of Definition 3.6, since the other one is easier.

Note that $V_0 \supseteq val(P\sigma_1, Q\sigma_2, Eq)$ (Lemma 3.2.2). Furthermore V_0 contains a value v_0 s.t. $v_0 \notin val(P\sigma_1, Q\sigma_2, Eq)$: in fact, v_0 since $P \prec P_0$, $Q \prec Q_O$, in virtue of Lemma 3.2.1, $P\sigma_1$, $Q\sigma_2$ and Eq together contain no more than $2*k$ distinct values. Therefore, according to Definition 3.6, clause two, it will suffice to show that for any transition $P\sigma_1 \xrightarrow{a(v)} P_1$, with $v \in V_0$, there is a transition $Q\sigma_2 \xrightarrow{a(v)} Q_1$ s.t. $P_1 \,\mathcal{R}\, Q_1$. The latter means that there exist P', Q', σ_1' and σ_2' s.t.

$$val(\sigma_1',\sigma_2') \subseteq V_0, \text{ and } P_1 = P'\sigma_1',\ Q_1 = Q'\sigma_1', \text{ and } P' \prec P_0,\ Q' \prec Q_0 \text{ and } [\![P']\!]\sigma_1' \sim [\![Q']\!]\sigma_2'\,.$$

Thus, consider a transition $P\sigma_1 \xrightarrow{a(v)} P_1$, $v \in V_0$. Applying Proposition 6.4.4, we get

$$[\![P]\!]\sigma_1 \xrightarrow{av} \sim [\![P']\!]\sigma_1\{v/x\} \text{ with } P_1 = P'\sigma_1\{v/x\} \text{ and } P' \prec P. \tag{6}$$

Since $[\![P]\!]\sigma_1 \sim [\![Q]\!]\sigma_2$, it follows:

$$[\![Q]\!]\sigma_2 \xrightarrow{av} Q_2 \text{ with } [\![P']\!]\sigma_1\{v/x\} \sim Q_2. \tag{7}$$

Applying Proposition 6.4.3 to the above transition, we get:

$$Q_2 \sim [\![Q']\!]\sigma_2\{v/y\} \text{ for some } Q' \prec Q \text{ and } Q'\sigma_2\{v/y\} \text{ closed} \tag{8}$$

and furthermore:

$$Q\sigma_2 \xrightarrow{a(v)} Q'\sigma_2\{v/y\} \stackrel{\text{def}}{=} Q_1. \tag{9}$$

We prove now that the latter is the wanted matching transition. Define $\sigma_1' = \sigma_1\{v/x\}$ and $\sigma_2' = \sigma_2\{v/y\}$. Note that $P' \prec P \prec P_0$ implies $P' \prec P_0$, and similarly $Q' \prec Q_0$. Of course, $val(\sigma_1', \sigma_2') \subseteq V_0$ (as $v \in V_0$). From (6) and (9), $P_1 = P'\sigma_1'$ and $Q_1 = Q'\sigma_2'$, and both P_1 and Q_1 are closed. Now, from (7) and (8), it holds that $[\![P']\!]\sigma_1' \sim Q_2 \sim [\![Q']\!]\sigma_2'$, thus by definition $P_1 \mathcal{R} Q_1$. $\square$

The above theorem has been proven under the hypothesis that Val is infinite. Indeed, the case when Val is finite can be much more easily accommodated by letting $V_0 = Val$. We omit the details.

By inspection, it is easily seen that the translation can be carried out in polynomial-time with the size of the problem. Thus, by Theorems 6.5 and 6.6, we get that bisimilarity in $\mathcal{L}_{m,v,r}$ is polynomial-time reducible to bisimilarity in $\mathcal{L}_{m,r}$. Since the latter problem is decidable we have:

Theorem 6.7 *Bisimilarity in $\mathcal{L}_{m,v,r}$ is decidable and equivalent to bisimilarity in $\mathcal{L}_{m,r}$, up to polynomial-time reduction.*

6.1 Name-passing

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. A similar comment also holds for the channel name-passing output. We can accomplish a proper treatment of input and output actions by an appropriate use of matching. More precisely, it suffices to replace the output (both $\overline{a}v.$ and $\overline{a}x.$) clauses and the input clause of the definition of $[\![\,.\,]\!]$ in Table 6 with the following two:

$$\begin{array}{rcll} [\![a(x).P]\!] & = & \sum_{v\in V_0}[a=v]\sum_{w\in V_0} vw.A(\tilde{y},w) & \quad\text{where } \tilde{y} = fvar([\![P]\!]) - \{x\} \\[2ex] [\![\overline{a}y.P]\!] & = & \sum_{v\in V_0}[a=v]\sum_{w\in V_0}[y=w]\overline{vw}.A(\tilde{y}) & \quad\text{where } \tilde{y} = fvar([\![P]\!])\,. \end{array}$$

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 with few notational changes. We omit the details.

7 Identifiers and Matching Require Exponential Time

In this section, we will suppose that the set of values Val is infinite. Under this hypothesis, we shall prove the following result.

Lemma 7.1 *Let AT be a canonical linear space ATM. Then, for any string x of length n, we can compute, in time polynomial in n, two processes P and Q of $\mathcal{L}_{m,r}$ such that $P \sim Q$ if and only if x is accepted by AT.*

The above lemma and Theorem 2.6 will immediately imply the EXP-hardness of $\mathcal{L}_{m,r}$.

Our construction will be somehow similar to that of Section 4. We shall define three identifiers A, S, F[1]. As a first approximation, if $\bar{c}$ is a configuration of AT, then $A(\bar{c})$ is a process that *simulates* the computation of AT starting from configuration $\bar{c}$. In particular, if $\bar{c}_0$ is the starting configuration of AT with input x, then the labeled transition system of $A(\bar{c}_0)$ is " isomorphic"

[1] A stands for ATM, S for *success* and F for *failure*.

to $\mathcal{GC}_{AT(x)}$: there is a correspondence between configurations $\bar{c} \in \mathcal{GC}_{AT(x)}$ and processes $A(\bar{c})$, and between nondeterministic branching of the ATM and nondeterministic choice in the process. Furthermore, the processes corresponding to halting configurations $\bar{c} \in \mathcal{GC}_{AT(x)}$ can do a single action: a if $\bar{c}$ is accepting and b if $\bar{c}$ is rejecting. S (respectively, F) is defined to be identical to A, except that states corresponding to halting configurations always do a (respectively b). Thus, intuitively, A would be bisimilar to S in case AT accepts, and bisimilar to F otherwise.

Indeed, the above straightforward construction fails to express alternation of quantifiers in terms of bisimulation, and has to be slightly modified. For example, assume that an existential configuration $\bar{c}$ of $AT(x)$ branches into two configurations, one accepting and one rejecting. Then $\bar{c}$ is accepting, and we would like the corresponding process $A(\bar{c})$ to be bisimilar to $S(\bar{c})$. But $A(\bar{c})$ branches into both a rejecting and an accepting state, while $S(\bar{c})$ branches into two accepting states: thus $A(\bar{c})$ and $S(\bar{c})$ could not be bisimilar. This inconvenience can be overcome if we assume that each state corresponding to an existential configuration always branches into at least one state corresponding to a rejecting configuration, and that each state corresponding to a universal configuration always branches into at least one state corresponding to an accepting configuration.

The actual definition of identifiers A, S, and F is given in Tables 6–8. For simplicity, the definitions are given in the case of simple value-passing: the name-passing case requires few notational changes (adding the names of the channels and of the constants to the list of parameters in identifiers). As in Section 4, we shall split into three technical lemmas the proof that those identifiers indeed exhibit the desired behaviour.

Since no confusion can arise, in the following we will use $\mathcal{GC}$ as a shorthand for $\mathcal{GC}_{AT(x)}$.

Lemma 7.2 *Let $\bar{c}_1, \bar{c}_2 \in \mathcal{GC}$ be any two (not necessarily distinct) configurations that halt within the same number of steps, then the following holds.*

1. $S(\bar{c}_1) \not\sim F(\bar{c}_2)$.
2. $S(\bar{c}_1) \sim S(\bar{c}_2)$ *and* $F(\bar{c}_1) \sim F(\bar{c}_2)$.

Proof: We will prove that the lemma is true by induction on the number of steps required to move from $\bar{c}_1$ to a halting configuration. If $\bar{c}_1$ and $\bar{c}_2$ are halting configurations, then $S(\bar{c}_1)$ and $S(\bar{c}_2)$ can only perform action a, while $F(\bar{c}_1)$ and $F(\bar{c}_2)$ can only perform action b, thus $S(\bar{c}_1) \not\sim F(\bar{c}_2)$, $S(\bar{c}_1) \sim S(\bar{c}_2)$, and $F(\bar{c}_1) \sim F(\bar{c}_2)$.

Otherwise, assume that for any $\bar{c}'_1 \in \delta(\bar{c}_1)$ and for any $\bar{c}'_2 \in \delta(\bar{c}_2)$, the lemma is true for $\bar{c}'_1$ and $\bar{c}'_2$. We claim that $\bar{c}_1$ and $\bar{c}_2$ are either both existential or both universal. Recall that AT is canonical, and thus all computation paths have the same length. Since $\bar{c}_1$ and $\bar{c}_2$ halt within the same number of steps, it follows that they are reached from the initial configuration after the same number of steps. Furthermore, existential and universal configurations alternates at any step in AT (again because of canonicity), thus $\bar{c}_1$ and $\bar{c}_2$ are either both universal or both existential. We separately consider the two cases.

1. If $\bar{c}_1$ and $\bar{c}_2$ are both universal configurations, then the only actions done by $S(\bar{c}_1)$ are

   $$S(\bar{c}_1) \xrightarrow{a} S(\bar{c}'_1) \text{ for any } \bar{c}'_1 \in \delta(\bar{c}_1) \ ,$$

   while, considering any $c'_2 \in \delta(\bar{c}_2)$, we have that

   $$F(\bar{c}_2) \xrightarrow{a} F(\bar{c}'_2) \not\sim S(\bar{c}'_1) \text{ for any } \bar{c}'_1 \in \delta(\bar{c}_1)$$

   and thus $S(\bar{c}_1) \not\sim F(\bar{c}_2)$. Moreover, the only actions doable by $S(\bar{c}_2)$ are

$$
\begin{array}{lll}
\multicolumn{3}{l}{A(x_1, y_1, \ldots, x_n, y_n) \Leftarrow} \\
& \displaystyle\sum_{\substack{i\in\{1,\ldots,n\},\ q\in Q_E \\ \langle q,s,q',s',L\rangle\in\delta}} & [x_i = s][y_i = q]a.A(x_1, y_1, \ldots, x_{i-1}, q', s', \bot, x_{i+1}, y_{i+1}, \ldots, x_n, y_n) \\
& & + \ [x_i = s][y_i = q]a.F(x_1, y_1, \ldots, x_{i-1}, q', s', \bot, x_{i+1}, y_{i+1}, \ldots, x_n, y_n) \\
+ & \displaystyle\sum_{\substack{i\in\{1,\ldots,n\},\ q\in Q_E \\ \langle q,s,q',s',R\rangle\in\delta}} & [x_i = s][y_i = q]a.A(x_1, y_1, \ldots, x_{i-1}, y_{i-1}, s', \bot, x_{i+1}, q', \ldots, x_n, y_n) \\
& & + \ [x_i = s][y_i = q]a.F(x_1, y_1, \ldots, x_{i-1}, y_{i-1}, s', \bot, x_{i+1}, q', \ldots, x_n, y_n) \\
+ & \displaystyle\sum_{\substack{i\in\{1,\ldots,n\},\ q\in Q_U \\ \langle q,s,q',s',L\rangle\in\delta}} & [x_i = s][y_i = q]a.A(x_1, y_1, \ldots, x_{i-1}, q', s', \bot, x_{i+1}, y_{i+1}, \ldots, x_n, y_n) \\
& & + \ [x_i = s][y_i = q]a.S(x_1, y_1, \ldots, x_{i-1}, q', s', \bot, x_{i+1}, y_{i+1}, \ldots, x_n, y_n) \\
+ & \displaystyle\sum_{\substack{i\in\{1,\ldots,n\},\ q\in Q_U \\ \langle q,s,q',s',R\rangle\in\delta}} & [x_i = s][y_i = q]a.A(x_1, y_1, \ldots, x_{i-1}, y_{i-1}, s', \bot, x_{i+1}, q', \ldots, x_n, y_n) \\
& & + \ [x_i = s][y_i = q]a.S(x_1, y_1, \ldots, x_{i-1}, y_{i-1}, s', \bot, x_{i+1}, q', \ldots, x_n, y_n) \\
+ & \displaystyle\sum_{q\in Q_A} & [y_i = q]a \\
+ & \displaystyle\sum_{q\in Q_R} & [y_i = q]b
\end{array}
$$

Table 6: Definition of A.

$$
\begin{aligned}
S(x_1, y_1, \ldots, x_n, y_n) \Leftarrow & \\
& \sum_{\substack{i\in\{1,\ldots,n\},\ q\in Q_E \\ \langle q,s,q',s',L\rangle\in\delta}} [x_i = s][y_i = q]a.S(x_1, y_1, \ldots, x_{i-1}, q', s', \bot, x_{i+1}, y_{i+1}, \ldots, x_n, y_n) \\
& \qquad + [x_i = s][y_i = q]a.F(x_1, y_1, \ldots, x_{i-1}, q', s', \bot, x_{i+1}, y_{i+1}, \ldots, x_n, y_n) \\
+ & \sum_{\substack{i\in\{1,\ldots,n\},\ q\in Q_E \\ \langle q,s,q',s',R\rangle\in\delta}} [x_i = s][y_i = q]a.S(x_1, y_1, \ldots, x_{i-1}, y_{i-1}, s', \bot, x_{i+1}, q', \ldots, x_n, y_n) \\
& \qquad + [x_i = s][y_i = q]a.F(x_1, y_1, \ldots, x_{i-1}, y_{i-1}, s', \bot, x_{i+1}, q', \ldots, x_n, y_n) \\
+ & \sum_{\substack{i\in\{1,\ldots,n\},\ q\in Q_U \\ \langle q,s,q',s',L\rangle\in\delta}} [x_i = s][y_i = q]a.S(x_1, y_1, \ldots, x_{i-1}, q', s', \bot, x_{i+1}, y_{i+1}, \ldots, x_n, y_n) \\
& \qquad + [x_i = s][y_i = q]a.S(x_1, y_1, \ldots, x_{i-1}, y_{i-1}, s', \bot, x_{i+1}, q', \ldots, x_n, y_n) \\
+ & \sum_{q\in Q_A\cup Q_R} [y_i = q]a
\end{aligned}
$$

Table 7: Definition of S.

$$S(\bar{c}_2) \xrightarrow{a} S(\bar{c}'_2) \text{ for any } \bar{c}'_2 \in \delta(\bar{c}_2) \ ,$$

and this clearly implies that $S(\bar{c}_1) \sim S(\bar{c}_2)$. Similarly, we can show that $F(\bar{c}_1) \sim F(\bar{c}_2)$.

2. If $\bar{c}_1$ and $\bar{c}_2$ are existential configuration, then the only actions doable by $F(\bar{c}_1)$ are

$$F(\bar{c}_2) \xrightarrow{a} F(\bar{c}'_2) \text{ for any } \bar{c}'_2 \in \delta(\bar{c}_2) \ ,$$

while, considering any $c'_1 \in \delta(\bar{c}_1)$, we have that

$$S(\bar{c}_1) \xrightarrow{a} S(\bar{c}'_1) \not\sim F(\bar{c}'_2) \text{ for any } \bar{c}'_2 \in \delta(\bar{c}_2)$$

and thus $F(\bar{c}_2) \not\sim S(\bar{c}_1)$. As in the previous case, it is easy to see that $S(\bar{c}_1) \sim S(\bar{c}_2)$ and $F(\bar{c}_1) \sim F(\bar{c}_2)$.

□

Lemma 7.3 *For any $\bar{c} \in \mathcal{GC}$, either $A(\bar{c}) \sim S(\bar{c})$ or $A(\bar{c}) \sim F(\bar{c})$.*

Proof: We will proceed by induction on the number of residual steps. Let us suppose that $\bar{c}$ is a halting configuration, then either $\bar{c}$ is rejecting, and in this case $A(\bar{c})$ can only do action b, and therefore is bisimilar to $F(\bar{c})$, or $\bar{c}$ is accepting, and $A(\bar{c})$ can only do action a, and thus is bisimilar to $S(\bar{c})$.

Assume now, by inductive hypothesis, that for any configuration $\bar{c}' \in \delta(\bar{c})$ we have that $A(\bar{c}')$ is either bisimilar to $S(\bar{c}')$ or bisimilar to $F(\bar{c}')$, we have to distinguish two cases.

$$
\begin{array}{lll}
F(x_1, y_1, \ldots, x_n, y_n) \Leftarrow & & \\
& \displaystyle\sum_{\substack{i\in\{1,\ldots,n\},\ q\in Q_E \\ \langle q,s,q',s',L\rangle\in\delta}} & [x_i = s][y_i = q]a.F(x_1, y_1, \ldots, x_{i-1}, q', s', \perp, x_{i+1}, y_{i+1}, \ldots, x_n, y_n) \\
+ & \displaystyle\sum_{\substack{i\in\{1,\ldots,n\},\ q\in Q_E \\ \langle q,s,q',s',R\rangle\in\delta}} & [x_i = s][y_i = q]a.F(x_1, y_1, \ldots, x_{i-1}, y_{i-1}, s', \perp, x_{i+1}, q', \ldots, x_n, y_n) \\
+ & \displaystyle\sum_{\substack{i\in\{1,\ldots,n\},\ q\in Q_U \\ \langle q,s,q',s',L\rangle\in\delta}} & [x_i = s][y_i = q]a.S(x_1, y_1, \ldots, x_{i-1}, q', s', \perp, x_{i+1}, y_{i+1}, \ldots, x_n, y_n) \\
& + & [x_i = s][y_i = q]a.F(x_1, y_1, \ldots, x_{i-1}, q', s', \perp, x_{i+1}, y_{i+1}, \ldots, x_n, y_n) \\
+ & \displaystyle\sum_{\substack{i\in\{1,\ldots,n\},\ q\in Q_U \\ \langle q,s,q',s',R\rangle\in\delta}} & [x_i = s][y_i = q]a.S(x_1, y_1, \ldots, x_{i-1}, y_{i-1}, s', \perp, x_{i+1}, q', \ldots, x_n, y_n) \\
& + & [x_i = s][y_i = q]a.F(x_1, y_1, \ldots, x_{i-1}, y_{i-1}, s', \perp, x_{i+1}, q', \ldots, x_n, y_n) \\
+ & \displaystyle\sum_{q\in Q_A \cup Q_R} & [y_i = q]b
\end{array}
$$

Table 8: Definition of F.

1. If $\bar{c}$ is an existential configuration, then, for any $\bar{c}' \in \delta(\bar{c})$, $A(\bar{c})$ can evolve into $A(\bar{c}')$ or into $F(\bar{c}')$. If, for any $\bar{c}' \in \delta(\bar{c})$, we have that $A(\bar{c}') \sim F(\bar{c}')$, then $A(\bar{c})$ is bisimilar to $F(\bar{c})$; otherwise it is bisimilar to $S(\bar{c})$.

2. If $\bar{c}$ is an universal configuration, then, for any $\bar{c}' \in \delta(\bar{c})$, $A(\bar{c})$ can evolve into $A(\bar{c}')$ or into $S(\bar{c}')$. If, for any $\bar{c}' \in \delta(\bar{c})$ we have that $A(\bar{c}') \sim S(\bar{c}')$, then $A(\bar{c})$ is bisimilar to $S(\bar{c})$; otherwise it is bisimilar to $F(\bar{c})$.

□

Lemma 7.4 *For any $\bar{c} \in \mathcal{GC}$, $A(\bar{c}) \sim S(\bar{c})$ if and only if $\bar{c}$ is an accepting configuration.*

Proof: Again, we proceed by induction on the number of residual steps. If $\bar{c}$ is a halting configuration, then the proof is trivial.

Otherwise, let us assume that, for any $\bar{c}' \in \delta(\bar{c})$, $\bar{c}'$ is an accepting configuration if and only if $A(\bar{c}') \sim S(\bar{c}')$. We have to distinguish two cases: either $\bar{c}$ is existential or it is universal. We show only the firts case, as the second one is very similar.

Since $\bar{c}$ is existential, then it is accepting if and only if $\delta(\bar{c})$ contains at least one accepting configuration. By induction hypothesis, the latter statement holds if and only if:

$$\text{there is } \bar{c}' \in \delta(\bar{c}) \text{ s.t. } A(\bar{c}') \sim S(\bar{c}'). \tag{10}$$

We now prove that (10) holds if and only if $A(\bar{c}) \sim S(\bar{c})$. Let us assume that (10) holds. There are two non-trivial cases. First, consider the case when $A(\bar{c}) \xrightarrow{a} A(\bar{c}_1)$, for any $\bar{c}_1 \in \delta(\bar{c})$. From canonicity and Lemma 7.3, it is $A(\bar{c}_1) \sim S(\bar{c}_1)$ or $A(\bar{c}_1) \sim F(\bar{c}_1)$: in the first case the matching move for $S(\bar{c})$ is $A(\bar{c}) \xrightarrow{a} S(\bar{c}_1)$, in the second case it is $S(\bar{c}) \xrightarrow{a} F(\bar{c}_1)$. Consider the case when $A(\bar{c})$ has to match a transition $S(\bar{c}) \xrightarrow{a} S(\bar{c}_1)$ from $A(\bar{c})$. Then we have:

$$A(\bar{c}) \xrightarrow{a} A(\bar{c}') \sim S(\bar{c}') \sim S(\bar{c}_1)$$

where the first $\sim$ follows from (10) and the second one from canonicity and Lemma 7.2.

Conversely, assume that $A(\bar{c}) \sim S(\bar{c})$. Take any $c_1 \in \delta(\bar{c})$ and consider the transition $S(\bar{c}) \xrightarrow{a} S(\bar{c}_1)$. Then we must have for some $\bar{c}'$ $A(\bar{c}) \xrightarrow{a} K(\bar{c}') \sim S(\bar{c}_1)$, where $K = F$ or $K = A$. The case $K = F$ cannot arise, due to canonicity and Lemma 7.2. Thus it must be $K = A$. From canonicity and Lemma 7.2 it follows $A(\bar{c}') \sim S(\bar{c}_1) \sim S(\bar{c}')$, which validates (10).

□

Proof: (of Lemma 7.1) Let $\bar{c}_0$ be the initial configuration of AT with input x. Since the definition of the identifiers can be clearly constructed in time polynomial in n, the lemma follows from Lemma 7.4 by setting $P \stackrel{\text{def}}{=} A(\bar{c}_0)$ and $S \stackrel{\text{def}}{=} S(\bar{c}_0)$. □

Theorem 7.5 *The bisimilarity problem in the $\mathcal{L}_{m,r}$ calculus is* LIN-EXP*-hard and thus* EXP*-hard.*

We shall indeed see in the next section that the problem is in EXP. From the above theorem and from Theorem 2.2 the intractability of bisimilarity in $\mathcal{L}_{m,r}$ follows.

Corollary 7.6 *A constant $c > 0$ exists such that any algorithm that decides the bisimilarity problem in $\mathcal{L}_{m,r}$ has a worst-case running time no better than 2^{n^c}, where n is the size of the input.*

8 The Parallel Composition Operator

In this section we consider adding the *parallel composition* operator $|$ (see e.g. [Mil89]) to the language described in Section 2. We will show that, for a certain restricted syntactic format, that of *finite control* processes, the bisimilarity problem with parallel composition is decidable and EXP-complete. As a consequence, the bisimilarity problem is in EXP for all the fragments we have considered in the paper.

The syntax of the language $\mathcal{L}_{m,v,r}$ is extended with the clause

$$P ::= P \mid P \,.$$

All definitions and notions given for $\mathcal{L}_{m,v,r}$ (such as free variables, subterms etc.) are extended to the new language in the expected way. Following [Mil89], the operational semantics of the new operator is given by the rules:

$$(Par)\frac{P_1 \xrightarrow{\mu} P_1'}{P_1 \mid P_2 \xrightarrow{\mu} P_1' \mid P_2} \qquad (Com)\frac{P_1 \xrightarrow{\mu} P_1',\, P_2 \xrightarrow{\mu'} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2'}$$

with ($\mu = a$ and $\mu' = \overline{a}$) or ($\mu = a(v)$ and $\mu' = \overline{a}v$), *plus* the symmetric versions of the above rules, where the roles of P_1 and P_2 are exchanged. Here $\tau \notin Act$ is a new kind of action, called the *silent* action.

We cannot hope to extend in a simple way the reduction of Section 6, to get a translation from the new language to its fragment without value-passing primitives. The reason is that, contrary to what happens with $\mathcal{L}_{m,v,r}$, there is in general no finite upper-bound to the size reachable by a process with parallel composition during its execution. An example of this is shown in the process:

$$A(x) \Leftarrow a(x).(A(x) \mid A(x))\,.$$

As a consequence, we cannot determine a finite set of names V_0 containing enough fresh names for the translation of $[\![\,.\,]\!]_{V_0}$ to work. This phenomenon is only due to the presence of parallel composition inside recursive definitions. However, many processes commonly found in practice have a "static" structure, where parallel composition never occurs inside recursive definitions. It is therefore meaningful to confine ourselves to this class of parallel processes, which are called "finite control". The corresponding sub-language will be indicated with $\mathcal{L}_{m,v,r,p}$.

Let us call $\mathcal{L}_{m,r,p}$ the sublanguage of $\mathcal{L}_{m,v,r,p}$ without input/output primitives. We can extend the translation $[\![\,.\,]\!]_{V_0}$ to a translation from $\mathcal{L}_{m,v,r,p}$ to $\mathcal{L}_{m,r,p}$, by just adding the clauses:

$$[\![P_1 \mid P_2]\!] = [\![P_1]\!] \mid [\![P_2]\!] \quad \text{and} \quad \mathcal{D}(P_1 \mid P_2) = \mathcal{D}(P_1) \cup \mathcal{D}(P_2)\,.$$

The proofs of operational correspondences (Proposition 6.4) for the new encoding are easily extended, while the proof of Completeness (Theorem 6.5) carries over formally unchanged. We list now the few modifications necessary in the proof of Correctness (Theorem 6.6).

It is easy to see that if $P \prec P_0$ then the size of P cannot exceed

$$k \stackrel{\text{def}}{=} |P_0| * max\{|R| \,:\, R \text{ appears in } Eq\}.$$

In a similar way, we can determine an upper bound h to the size of every subterm of Q_0. Thus, we are sure that we can always find the fresh value v_0 needed in the proof by just taking, in the definition of V_0, a set $V \subseteq_{fin} Val$ s.t. $V \cap val(P_0, Q_0, Eq) = \emptyset$ and $|V| = max\{h, k\} + 1$. Note in particular that the size of V_0 is still polynomial w.r.t. the sizes of P_0, Q_0 and Eq. Modulo this modification, the proof carries over unchanged. A consequence of these considerations is the following:

Proposition 8.1 *The bisimilarity problem in $\mathcal{L}_{m,v,r,p}$ is equivalent to the bisimilarity problem in $\mathcal{L}_{m,r,p}$, up to polynomial-time reduction.*

Note that every process $P \in \mathcal{L}_{m,r,p}$ has a finite transition system; more precisely, since the size of every term reachable from P cannot exceed k, for the quantity k defined above, there are at most n^k different states in the transition system, where n is the number of different values present in P and in Eq; this cannot exceed the size of the bisimilarity problem. Note that k too is a polynomial function of the size of the problem. It follows the bisimilarity problem in $\mathcal{L}_{m,r,p}$ can be solved in exponential time using, for example, the algorithm by Page and Tarjan [PT87]. Putting together the latter fact, Theorem 6.7 and Theorem 7.5, the above Proposition 8.1, we have the following result of equivalence between languages:

Theorem 8.2 *The bisimilarity problems for the languages $\mathcal{L}_{m,r}$, $\mathcal{L}_{m,v,r}$, $\mathcal{L}_{m,r,p}$ and $\mathcal{L}_{m,v,r,p}$ are all* EXP*-complete.*

It is worth to notice that, even in the absence of values, the presence of parallel composition implies an exponential blow-up of the number of states. This is implicitly present, for example, in the so-called "expansion law" [Mil89]: $a \mid b \sim a.b + b.a$. In general, it is easy to see that the process $a_1 \mid \cdots \mid a_n$ (for distinct a_i's) has 2^n states. However, the above theorem tells us that the computational complexity due to parallel composition itself is not greater than that caused by the handling of data-values.

9 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 variants of bisimulation have been proposed, such as *late* bisimilarity [MPW92, PS95] and *open* bisimilarity [San93]. Many of the results presented in the paper 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. [PS95]).

Our paper is mainly related to [JP93]. There, Jonsson and Parrow prove that bisimilarity is decidable in the data-independent language $\mathcal{L}_{v,r}$, 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 $\mathcal{L}_{v,r}$ 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.

A question that is left open by the present work is the exact complexity of bisimilarity in $\mathcal{L}_{v,r}$. This language looks quite simpler than $\mathcal{L}_{m,v,r}$. Indeed, it is possible to show that a process of $\mathcal{L}_{m,v,r}$ exists that is not bisimilar to any process of $\mathcal{L}_{v,r}$, that is, $\mathcal{L}_{m,v,r}$ has more expressive power than

$\mathcal{L}_{v,r}$. Even if such observation has no immediate complexity-theoretic implication, we suspect that a polynomial-space algorithm exists for the bisimilarity problem in $\mathcal{L}_{v,r}$.

In the data-dependent setting, we showed the EXP-completeness of the bisimilarity problem. This is the stronger intractability result proved to date for a decidable bisimilarity problem. The reduction establishing such result does not make use of the value-passing operators: this is hardly surprising in view of our proof that the full language can be compositionally translated in polynomial time into the fragment $\mathcal{L}_{m,r}$ without value-passing.

The parallel composition operator is known to create an exponential blow-up of the size of the labeled transition systems. This motivates the common belief that adding such operator to a language generally increases the complexity of the bisimilarity problem. Indeed, we have showed that if the parallel composition operator is never used inside recursive definition, then adding it to $\mathcal{L}_{m,v,r}$ does not increase the complexity of bisimilarity, that remains EXP-complete (Theorem 8.2). The latter result implies that, given two processes P and Q in $\mathcal{L}_{m,v,r,p}$, we can compute in polynomial time two processes P' and Q' in $\mathcal{L}_{m,r}$ such that $P \sim Q$ if and only if $P' \sim Q'$. However, the result does not imply any apparent relationship between the syntactic structure of P and Q and that of P' and Q'. It would be interesting to find, in the spirit of the translation from $\mathcal{L}_{m,v,r,p}$ to $\mathcal{L}_{m,r,p}$, a compositional reduction from $\mathcal{L}_{m,r,p}$ to $\mathcal{L}_{m,r}$ that would show how to "express" the parallel composition operator using the other operators.

In [HL95, HL93, San93, BD94], 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.

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- Extended abstract. In B. Jonsson and J. Parrow, editors, *Proceedings of Concur '94, LNCS 836*, pages 299–314. Springer-Verlag, Berlin, 1994. Full version to appear in *Information and Computation.*

[CKS81] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[Coo71] S.A. Cook. The complexity of theorem proving procedures. In *Proceedings of the 3rd ACM Symposium on Theory of Computing*, pages 151–158, 1971.

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

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

[HS65] J. Hartmanis and R.E. Stearns. On the computational complexity of algorithms. *Transactions of the AMS*, 117:285–306, 1965.

[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. Kanellakis 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, Berlin, 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.

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

[PS95] J. Parrow and D. Sangiorgi. Algebraic theories for name-passing calculi. *Information and Computation*, 120(2):174–197, 1995.

[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 E. Best, editor, *Proceedings of CONCUR '93, LNCS 715*. Springer-Verlag, Berlin, 1993. To appear in *Acta Informatica*.

[SM73] L. Stockmeyer and A. Meyer. Word problems requiring exponential time. In *Proceedings of the 5th ACM Symposium on Theory of Computing*, pages 1–9, 1973.