Efficient Minimization up to Location Equivalence

Ugo Montanari[1]*, Marco Pistore[1]* and Daniel Yankelevich[2]**

[1] Dipartimento di Informatica, Università di Pisa
[2] Departamento de Informática, Universidad de Buenos Aires

Abstract. *Location equivalence* is a bisimulation based equivalence for process calculi which is able to take into account the distributed nature of processes; the underlying idea is that each action occurs at a particular location.
The definition of bisimulation for location equivalence is not the standard one, since it must deal with the creation of new locations, and this leads to the necessity of using specific algorithms. In particular these algorithms work only on pairs of agents and do not allow to find the minimal representative for a class of equivalent agents.
In this paper we associate to every agent a labeled transition system (in which the informations on the locations appear in the labels) so that location-equivalent agents are mapped into transition systems which are bisimilar according to the ordinary definition of bisimulation. The main consequence of this result is that the standard algorithms for ordinary bisimulation can be re-used, and in particular the *partitioning algorithm* which allows to obtain the minimal realization of a single agent.

1 Introduction

Communication protocols and distributed systems tend to be difficult to understand and they usually present complex behaviors. For this reason, there has been a considerable interest in finding automatic methods to validate and verify this kind of systems, both in academy and industry. Even entire conferences are dedicated to this problem [15, 16].

The effort invested in this research gave rise to many different tools and methods to verify distributed systems. A number of such tools and methods are based on the idea of comparing the actual behavior of a protocol or distributed system with its expected behavior, described by a specification [7]. Hence, the languages used are equipped with an equivalence relation between programs (and specifications). In general, one is not forced to use the same specification and programming language, it suffices that both languages can be *compiled* to a

* Research supported in part by Esprit Basic Research project CONFER and by Progetto Coordinato CNR "*Strumenti per la Verifica di Proprietà Critiche di Sistemi Concorrenti e Distribuiti*".
** Research supported in part by Universidad de Buenos Aires, under UBACyT project EX186, and by Università di Pisa.

common model. For reactive, distributed systems, operational models are very adequate. Among these models, the most widely acknowledged as useful models of concurrent systems are the transition systems. Hence, the equivalence between specifications and programs is actually defined as an equivalence between transition systems.

In most cases, a variant of the so-called *bisimulation equivalence* is used [9]. This equivalence has an objective advantage from the point of view of verification, and it is that a well known algorithm exists to verify it, namely the partition refinement algorithm [13]. Besides checking for bisimilarity, this algorithm finds the transition system that is minimal in the class of equivalent (bisimilar) transition systems. This is particularly interesting since this minimal realization can replace the original agent for all successive checks of properties. Moreover, minimization is also important if one is validating a large system, consisting of many programs composed in parallel, like in $p_1 \mid \ldots \mid p_n$. It can be of substantial importance in this case to get the small representations of p_i, namely p_i', and then construct the transition system for $p_1' \mid \ldots \mid p_n'$. Since the size of the transition system can be as big as the product of the sizes of the parallel components, the reduction in size in each p_i may have a great impact on the overall construction.

Clearly, the equivalence relation is a semantic relation, in the sense that it takes into account semantic information. Hence, when choosing a particular equivalence one is fixing the meaning of programs and specifications. The operational model describes the behavior of a system, but does not fix the semantics of the language. The equivalence, that abstracts away details of the operational model that are not relevant for the semantics, is needed to define the meaning of programs and specifications.

The equivalence described so far, used in most tools and methodologies, is based on the so called *interleaving semantics*. Hence, when using any of these tools, the meaning of the protocols and distributed systems is forced to be the one given in the interleaving semantics.

One drawback of this model is that parallelism is not considered a primitive concept, and it can be reduced to nondeterminism. This means that, for example, the programs $a \mid b$ and $a.b + b.a$ are identified, where $\mid$ is the parallel composition, $.$ the sequential composition, $+$ the nondeterministic choice, and a, b any actions. A consequence of this fact is that a system distributed in more sites is equivalent to a system executing the same actions in only one site. This is clearly not the intuition of protocol and distributed systems programmers and designers.

Moreover, some properties of interest cannot be expressed using this semantics. For instance, the notion of *local deadlock*, i.e., a deadlock in one site that is not a global deadlock, cannot be expressed. The reason is very simple: suppose two processes provide the same service, concurrently, in two different sites. If one of them stops because a deadlock occurs, the global behavior is not affected: the system will still be able to provide the same service.

In interleaving semantics, notions such as degree of parallelism, causality, local clocks, etc. are not taken into account. Many proposals have been done to give semantics to concurrent systems considering these aspects. Among these al-

ternative semantic models, one approach gives particular interest to the *location* where an event takes place [3, 4]. In this model, called *locality semantics*, each action occurs in a particular place. Hence, the programs $a \mid b$ and $a.b + b.a$ are distinguished: the first one may perform an a and a b in different places, while the second executes the actions in only one place. Intuitively, the first program is distributed in two sites, while the second runs on a single processor. Moreover, as it is possible to detect where an action occurs, it is possible to express conditions such as local deadlocks, as shown in [3].

As it was first proposed, this semantics gives rise to infinite transition systems, even for very simple programs. From a technical point of view, each action that a program executes creates a new location name associated with that event. Hence, the transition system describing the behavior of a program does not contain cycles: each cycle is *unfolded* and the compact representation of behaviors given by transition systems is lost.

Different techniques were proposed to deal with this problem, and alternative characterizations of the same equivalence, that were not infinite in the sense described above, were developed [8, 14, 6, 11, 12].

From the point of view of verification, this is not the only problem that this new semantics poses. Even with non infinite transition systems, each program may choose different names for locations. The equivalence must be checked up to bijections of names. For instance, the programs $a \mid b$ and $b \mid a$ must be identified. This means that the correspondence between the left side of one program and the right side of the other must be established. This correspondence cannot be established statically once and for all, since locations may be created (by a fork action) or destroyed (by a join). Hence, one has to dynamically construct a mapping between location names.

A consequence of this fact is that partition refinement techniques cannot be used in order to check location bisimulation equivalence. Only the so-called *on the fly* methods can be used. These methods actually construct the equivalence dynamically, on the fly, and in consequence may construct the bijection as they go.

While on the fly techniques have been shown to be very useful in some situations, partition algorithms are better in other contexts. Having *both* techniques available adds flexibility that may help in the automatization of the verification process. Moreover, on the fly techniques cannot be used to construct a minimal transition system for a given program.

In this work, we show how partition refinement techniques can be used to check location equivalence. Hence, our algorithm can be used, in particular, to get a minimal representative with respect to location equivalence.

The main idea underlying this technique can be described as follows. First, we define transition systems with states labeled by sets of location names, showing which locations are in use in each state. Second, in order to make identical the names corresponding in the bijection, we choose the new names following a standard order. A similar idea was used in [14] in the so-called numbered transition system: locations are chosen following a strict ordering. However, this

is not enough to guarantee identity of names, if the new location is chosen as the first locations *not presently used*, since some locations may appear as presently used while in fact being not relevant. Actually, what is needed is a notion of relevance in the future computations of the program. We introduce a semantic concept of *active location*, and automata using only active locations are called *irredundant*. Finally, we show that ordinary bisimulation of irredundant automata coincides with location equivalence. From this result on semantics, it follows that the usual partition algorithm can be used.

The paper is organized as follows. In Section 2 some background is presented, mainly about the example language used (CCS) and about location equivalence. Section 3 introduces structural axioms. These are simplification axioms that are used to enlarge the class of programs to which the algorithm may be applied. Many of these axioms have been used as rules of thumb in the implementation of some systems. For instance, they express the notion that useless $\cdot \mid \mathrm{nil}$ constructs can be eliminated.

In Section 4 we introduce location automata, that are simple transition systems whose states are enriched with location names; and we define the notion of bisimulation of location automata. Section 5 introduces the notion of active location and of irredundant automata and presents the main result of the paper, namely the theorem that shows that our algorithm is sound.

In Section 6, we analyze the complexity of our algorithm. Even with this finer semantics, the worst-case complexity of bisimulation checking does not change. Section 7 is devoted to concluding remarks.

2 Background

In this section we briefly recall the approach to locality semantics introduced in [3, 4, 8]. Differently from the *static* approach of [1] — where the distributed nature of agents is made explicit by assigning different locations to their parallel components, like in $a.(l :: p \mid m :: q)$ — in [3, 4, 8] a more observational point of view is preferred. Location names are assigned *dynamically*, during the process of observation: the meaning of transition $l :: a.p \xrightarrow[lm]{a} l :: m :: p$ is that the observer sees an action a emanating from a particular sublocation of l and associates name m to this sublocation[3].

Let Λ be a set of *atomic actions* (ranged over by $\alpha, \beta, \ldots$) and $\bar{\Lambda} = \{\bar{\alpha} \mid \alpha \in \Lambda\}$ a set of *action complements* disjoint from Λ. $Act = \Lambda \cup \bar{\Lambda}$ (ranged over by $a, b, \ldots$) is the set of *visible actions* (the operator $\bar{\cdot}$ is extended to Act in such a way that $\bar{\bar{\alpha}} = \alpha$), $\tau \notin Act$ is the *invisible action* and $Act_\tau = Act \cup \{\tau\}$ (ranged over by μ).

Let Var be a set of *process variables* (ranged over by $x, y, \ldots$) and Loc a totally ordered denumerable set of *locations* (ranged over by $l, m, \ldots$; $u, v, \ldots$ denote sequences of locations).

[3] We refer to [5] for further comparisons of static and dynamic approach.

CCS location terms are defined by the following abstract syntax (the order of the operators gives their precedence):

$$p ::= \text{nil} \mid \mu.p \mid l::p \mid p \setminus \alpha \mid p \mid p \mid p + p \mid x \mid \text{rec}\, x.p$$

CCS location agents (ranged over by $p, q, \ldots$) are guarded (in a rec subterm the process variable appears only within prefix contexts), closed (without free process variables) location terms. We call $\mathbf{P}_{Loc}$ the set of all location agents. The set of location names that occur in p is denoted by $\text{loc}(p)$; an agent p is *pure* if $\text{loc}(p) = \emptyset$.

Following definitions are derived from those of [4].

Definition 1 (standard transitions). The *standard transitions* are defined by the following axioms and inference rules:

$$\mu.p \xrightarrow{\mu} p$$

$$p \xrightarrow{\mu} p' \quad \text{implies} \quad l::p \xrightarrow{\mu} l::p'$$

$$p \xrightarrow{\mu} p' \quad \text{implies} \quad p \setminus \alpha \xrightarrow{\mu} p' \setminus \alpha \quad \text{if} \quad \mu \notin \{\alpha, \bar{\alpha}\}$$

$$p \xrightarrow{\mu} p' \quad \text{implies} \quad p \mid q \xrightarrow{\mu} p' \mid q \quad \text{and} \quad q \mid p \xrightarrow{\mu} q \mid p'$$

$$p \xrightarrow{a} p' \quad \text{and} \quad q \xrightarrow{\bar{a}} q' \quad \text{implies} \quad p \mid q \xrightarrow{\tau} p' \mid q'$$

$$p \xrightarrow{\mu} p' \quad \text{implies} \quad p + q \xrightarrow{\mu} p' \quad \text{and} \quad q + p \xrightarrow{\mu} p'$$

$$p[\text{rec}\, x.p/x] \xrightarrow{\mu} p' \quad \text{implies} \quad \text{rec}\, x.p \xrightarrow{\mu} p'$$

Definition 2 (location transitions). The *location transitions* are defined by the following axioms and inference rules:

$$a.p \xrightarrow[l]{a} l::p \quad \text{for all} \quad l \in Loc$$

$$p \xrightarrow[u]{a} p' \quad \text{implies} \quad l::p \xrightarrow[lu]{a} l::p'$$

$$p \xrightarrow[u]{a} p' \quad \text{implies} \quad p \mid q \xrightarrow[u]{a} p' \mid q \quad \text{and} \quad q \mid p \xrightarrow[u]{a} q \mid p'$$

The rules for $+$, $\setminus$ and rec are analogous to the corresponding rules of Definition 1.

Notice that there is no synchronization rule for the location transitions: since the invisible transitions do not occur in a particular location, the rules of Definition 1 are used for them.

We will use the following notation for weak transitions: $\stackrel{\epsilon}{\Rightarrow} = (\xrightarrow{\tau})^*$ and $\stackrel{a}{\underset{u}{\Rightarrow}} = \stackrel{\epsilon}{\Rightarrow} \xrightarrow[u]{a} \stackrel{\epsilon}{\Rightarrow}$.

Definition 3 (location equivalence). A relation $\mathcal{R} \subseteq \mathbf{P}_{Loc} \times \mathbf{P}_{Loc}$ is a *location simulation* if $p \mathcal{R} q$ implies:

- for each $p \stackrel{a}{\underset{ul}{\Rightarrow}} p'$, with $l \notin \text{loc}(p, q)$, there exists some $q \stackrel{a}{\underset{ul}{\Rightarrow}} q'$ with $p' \mathcal{R} q'$;
- for each $p \stackrel{\epsilon}{\Rightarrow} p'$ there exists some $q \stackrel{\epsilon}{\Rightarrow} q'$ with $p' \mathcal{R} q'$.

A relation $\mathcal{R}$ is a *location bisimulation* if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are location simulations. Two processes p and q are *location equivalent* (written $p \approx_l q$) if $p \mathcal{R} q$ for some location bisimulation $\mathcal{R}$.

Condition $l \notin \mathrm{loc}(p,q)$ does not appear in [4]. In [8], however, it has been pointed out that no discriminating power is added if we are allowed to choose a location twice in a computation and that our definition is equivalent to the one in [4].

3 Incremental Location Equivalence

To check the equivalence of two CCS agents we have first to build finite transition systems corresponding to them. The aim of this section is to introduce the ideas that allow to accomplish this finite construction for a wide class of agents.

Each CCS agent can be seen as a system in which a set of *sequential processes* act in parallel, sharing a set of channels, some of which are global (unrestricted) whereas some other are local (restricted). Each sequential process is represented by a term of the form

$$s ::= \mu.p \mid p+p \mid \mathrm{rec}\, x.p$$

that can be considered as a "program" describing the possible behaviors of the sequential process.

These sequential processes are then connected by means of the operators of parallel composition, restriction and location prefixing, that allow to describe the structure of the system in which the processes act.

From this point of view the two parallel composition bars in

$$p = \alpha.p_1 \,|(\beta.(p_2 \,|\, p_3) + \gamma.p_4)$$

have different meanings, since the outermost indicates two processes that can act in parallel, whereas the innermost represents a possible *fork*, a future activation of two processes. Another consequence of this point of view is that the agents

$$(s_1 \,|\, s_2)\backslash\alpha\backslash\beta \qquad \text{and} \qquad (s_2 \,|\, s_1)\backslash\beta\backslash\alpha$$

should not be distinguished, since they represent the same processes acting with the same interconnection structure.

We thus introduce a set of *structural axioms*, in the style of the Chemical Abstract Machine [2] and of the π-calculus [10], which identify all such agents[4].

Par $p \,|\, \mathrm{nil} \equiv p \qquad p \,|\, q \equiv q \,|\, p \qquad p \,|(q \,|\, r) \equiv (p \,|\, q) \,|\, r$

Res $p\backslash\alpha \,|\, q \equiv (p \,|\, q)\backslash\alpha$ if α does not appear free in q

$\mathrm{nil}\backslash\alpha \equiv \mathrm{nil} \qquad p\backslash\alpha\backslash\beta \equiv p\backslash\beta\backslash\alpha$

$p\backslash\alpha \equiv p[\beta/\alpha]\backslash\beta$ if β does not appear in p

Loc $l::\mathrm{nil} \equiv \mathrm{nil} \qquad l::(p \,|\, q) \equiv (l::p) \,|(l::q) \qquad l::(p\backslash\alpha) \equiv (l::p)\backslash\alpha$

Consider for instance:

$$p = \mathrm{rec}\, x.(\alpha.\gamma.\mathrm{nil} \,|\, \bar{\gamma}.\beta.x)\backslash\gamma$$

$$p \xrightarrow{\alpha} \xrightarrow{\tau} \xrightarrow{\beta} (\mathrm{nil} \,|\, p)\backslash\gamma = p'$$

[4] We do not care about applying the structural axioms inside a sequential process.

The transition system corresponding to agent p in ordinary CCS is infinite (since $p \neq p'$), but using the structural axioms a finite one is generated (since $p \equiv p'$). The axioms we have introduced are sufficient to associate a finite standard transition system (i.e., whose transitions are generated using only the rules of Definition 1) to each *finitary* agent[5]. An agent is finitary if the degree of parallelism that it has and that it can reach in its future evolutions is (finitely) bounded.

Definition 4 (finitary agents). The *degree of parallelism* $\mathrm{par}(p)$ of an agent p is defined as

$$\begin{array}{rlrl} \mathrm{par}(\mathrm{nil}) &= 0 & \mathrm{par}(\mu.p) &= 1 \\ \mathrm{par}(l::p) &= \mathrm{par}(p) & \mathrm{par}(p \setminus \alpha) &= \mathrm{par}(p) \\ \mathrm{par}(p \mid q) &= \mathrm{par}(p) + \mathrm{par}(q) & \mathrm{par}(p+q) &= 1 \\ \mathrm{par}(\mathrm{rec}\, x.p) &= 1 & & \end{array}$$

A CCS agent p is *finitary* if $\max\{\mathrm{par}(p') \mid p \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_1} p'\} < \infty$.

A syntactical condition which implies that an agent is finitary is the absence of parallel compositions in the bodies of recursive definitions. However, there are many interesting finitary agents, like $\mathrm{rec}\, x.a.(b.\delta.d.x \mid c.\bar{\delta}.\mathrm{nil}) \setminus \delta$, which do not satisfy this condition.

The axioms **Par**, **Res** and **Loc** are not sufficient in the context of location equivalence: consider the agent $p = \mathrm{rec}\, x.a.x$ and its computation

$$p \xrightarrow[l]{a} l::p \xrightarrow[lm]{a} l::m::p \xrightarrow[lmn]{a} l::m::n::p \longrightarrow \cdots$$

The location prefixes continue to grow during the computation, and this must be avoided to obtain finite transition systems. Using the axiom

Del $\quad l::m::p \equiv m::p$

the previous computation of p could be transformed into:

$$p \xrightarrow[l]{a} l::p \xrightarrow[lm]{a} l::m::p \equiv m::p \xrightarrow[ml]{a} m::l::p \equiv l::p \xrightarrow[lm]{a} l::m::p \equiv m::p \longrightarrow \cdots$$

which is cyclic. However, this axiom is not correct for the location equivalence of Definition 3, since

$$l::m::a.p \xrightarrow[lmn]{a} l::m::n::p$$

whereas

$$m::a.p \xrightarrow[mn]{a} m::n::p$$

and the two labels do not correspond; this happens because the whole sequence of locations is observed in the label of a transition.

Now we give a slightly different definition of location equivalence in which only the newly created location and its direct parent are observed. It can be shown that this new location equivalence coincides with the classical one for the class of pure CCS agents.

[5] Also a smaller set of axioms is sufficient to this purpose. Our set of axioms, however, is very natural, and allows the identification of more agents.

Proposition 5 (incremental location equivalence). *Let p_0 and q_0 be two pure CCS agents. Then $p_0 \approx_l q_0$ iff $m :: p_0 \approx_{\Delta l} m :: q_0$ for some location m, where the* incremental location equivalence $\approx_{\Delta l}$ *is the maximal symmetric relation such that $p \approx_{\Delta l} q$ implies:*

- *for each $p \overset{a}{\underset{umn}{\Longrightarrow}} p'$ there exists some $q \overset{a}{\underset{vmn}{\Longrightarrow}} q'$ with $p' \approx_{\Delta l} q'$;*
- *for each $p \overset{\epsilon}{\Rightarrow} p'$ there exists some $q \overset{\epsilon}{\Rightarrow} q'$ with $p' \approx_{\Delta l} q'$.*

Del is a correct axiom for this alternative characterization and it allows, combined with the other axioms, to associate to each agent a flat structure of locations. Conceptually, these axioms show that agents can be seen in location semantics as collections (multisets) of sequential sub-agents acting in different locations. This intuitive fact, used in [6] to represent location agents, gets, in this way, a formal foundation using simple structural axioms.

From now on $p \equiv q$ means that p and q are the same agent up to axioms **Par**, **Res**, **Loc** and **Del**.

Proposition 6. *Using the structural axioms* **Par**, **Res**, **Loc** *and* **Del**, *every location agent p can be written in the following form:*

$$p \equiv (p_0 \mid l_1 :: p_1 \mid \cdots \mid l_n :: p_n) \setminus \alpha_1 \cdots \setminus \alpha_m$$

$$p_i = s_{i1} \mid \cdots \mid s_{in_i}$$

where locations l_i are all distinct and s_{ij} are sequential processes.

Notice that even the introduction of axiom **Del** is not sufficient to associate a finite location transition system (i.e., whose transitions are generated using the rules of Definition 2 for visible actions) to each finitary agent. In fact, even to very simple agents like $l :: a.b.\mathrm{nil}$ correspond infinitely many derivatives, since the transition

$$l :: a.b.\mathrm{nil} \overset{a}{\underset{lm}{\longrightarrow}} l :: m :: b.\mathrm{nil} \equiv m :: b.\mathrm{nil}$$

can occur for every $m \in Loc$. As we formally show in the following section, however, in checking location equivalence it is not necessary to consider all these different transitions, since they lead to states which differ only for the particular choice of location names.

4 Location Automata

Definition 7 (location automaton). A *location automaton* is a tuple $A = \langle Q, w, \mapsto, q_0 \rangle$ where:

- Q is a set of *states*;
- $w : Q \to 2^{Loc}_f$ associates to each state a finite set of locations;
- $\mapsto$ is a set of *transitions*; each transition has the form $q \overset{a}{\underset{l}{\mapsto_\sigma}} q'$ (*visible transition*) or the form $q \overset{\tau}{\mapsto_\sigma} q'$ (*invisible transition*), where:

- $q, q' \in Q$ are the *source* and *target* states;
- $l \in w(q)$ is the location of the transition;
- $\sigma : w(q') \hookrightarrow w(q) \cup \{\star\}$ ($\sigma : w(q') \hookrightarrow w(q)$ for an invisible transition) is the injective (inverse) *renaming* corresponding to the transition; the newly created location is denoted with the special mark $\star \notin Loc$;

– $q_0 \in Q$ is the *initial state*; we require that $w(q_0) = \{l\}$ for some $l \in Loc$.

A location automaton is an automaton particularly suited for dealing with locations. Each state p is labeled by the set $w(p)$ of locations used in that state. These locations have a meaning that is local, private to the state. Hence, the particular choice of location names cannot by itself make a distinction between two states of the location automaton.

Each visible transition $\overset{a}{\underset{l}{\longmapsto}}_\sigma$ represents an action a occurring in a location l of the source state. Due to the local meaning of locations, each transition must also specify the correspondence between the locations of the source and those of the target. This correspondence is obtained via the renaming σ, which permits also to deduce which locations of the source are forgotten in the target and which (if any) location of the target is the newly created location.

Following the approach of the previous sections, invisible transitions do not occur in a particular location and cannot create a new location.

The weak transitions of a location automaton can be defined as follows:

– $p \overset{\epsilon}{\Longmapsto}_\sigma p'$ if, for some $n \geq 0$, $p \overset{\tau}{\longmapsto}_{\sigma_1} \overset{\tau}{\longmapsto}_{\sigma_2} \cdots \overset{\tau}{\longmapsto}_{\sigma_n} p'$ and $\sigma = \sigma_1 \circ \sigma_2 \circ \cdots \circ \sigma_n$;

– $p \overset{a}{\underset{l}{\Longmapsto}}_\sigma p'$ if $p \overset{\epsilon}{\Longmapsto}_{\sigma_1} \overset{a}{\underset{m}{\longmapsto}}_{\sigma_2} \overset{\epsilon}{\Longmapsto}_{\sigma_3} p'$, $l = \sigma_1(m)$ and $\sigma = \sigma_1 \circ \sigma_2 \circ \sigma_3$.

The composition $\circ$ is defined so that $\sigma_1 \circ \sigma_2(n) = \star$ if $\sigma_2(n) = \star$ and $\sigma_1 \circ \sigma_2(n) = \sigma_1(\sigma_2(n))$ otherwise.

On a location automaton a bisimulation is not simply a relation on states: also a partial correspondence between the locations of the states has to be specified and the same states can be in relation via more than one correspondence. The requirement that just one location is used in the initial state allows to fix the initial correspondence (i.e., the correspondence between the locations of the initial states) when two location automata are checked for equivalence.

Definition 8 (la-bisimulation). Two location automata A and B are *location-automaton bisimilar*, written $A \approx_{la} B$, if there is some set $\mathcal{R}$ of triples, called *la-bisimulation*, such that:

– if $\langle p, \delta, q \rangle \in \mathcal{R}$ then $p \in Q_A$, $q \in Q_B$ and $\delta : w_A(p) \multimap w_B(q)$ is a partial bijection;

– $\langle q_{0A}, \delta_0, q_{0B} \rangle \in \mathcal{R}$, where δ_0 maps the location associated to q_{0A} to the location associated to q_{0B};

– for each $p \overset{a}{\underset{l}{\Longmapsto}}_\sigma p'$ in A (resp. $q \overset{a}{\underset{l}{\Longmapsto}}_\rho q'$ in B) there exist some δ' and some $q \overset{a}{\underset{\delta(l)}{\Longmapsto}}_\rho q'$ in B (resp. $p \overset{a}{\underset{\delta^{-1}(l)}{\Longmapsto}}_\sigma p'$ in A) such that $\langle p', \delta', q' \rangle \in \mathcal{R}$ and $\delta'(m) = n$ implies $\sigma(m) = \star = \rho(n)$ or $\delta(\sigma(m)) = \rho(n)$;

- for each $p \stackrel{\epsilon}{\mapsto}_\sigma p'$ in A (resp. $q \stackrel{\epsilon}{\mapsto}_\rho q'$ in B) there exist some δ' and some $q \stackrel{\epsilon}{\mapsto}_\rho q'$ in B (resp. $p \stackrel{\epsilon}{\mapsto}_\sigma p'$ in A) such that $\langle p', \delta', q' \rangle \in \mathcal{R}$ and $\delta'(m) = n$ implies $\delta(\sigma(m)) = \rho(n)$.

Notice that if p and q correspond via δ in some bisimulation $\mathcal{R}$, then to each visible transition of p a visible transition of q must correspond, such that *i*) the two transitions perform the same action, *ii*) they occur in corresponding locations (via δ), and *iii*) the reached states are related in $\mathcal{R}$ by some δ' which relates two locations of the target states only if they both are the newly created locations or if their corresponding locations in the source states are related by δ (two locations of the target states can be not related also if the corresponding locations are related in the source states).

Now we show how to associate location automata to pure CCS agents so that location equivalent agents are mapped into la-equivalent automata. In the construction it is useful to transform each reached state as described in Proposition 6, to keep the number of generated states small. To this purpose, it is also important to identify those states which differ only for a injective renaming of the locations: as previously noted, such states are not distinguishable in the context of location automata.

So we can define a function **norm** which, given an agent p, returns a pair $\langle p', \sigma \rangle$, where p' is obtained transforming p in the form described in Proposition 6 and then by normalizing also the location names $l_1, \ldots, l_n$ (for instance by replacing them with the first n locations of *Loc*), whereas $\sigma : \mathrm{loc}(p') \hookrightarrow \mathrm{loc}(p)$ describes which location of p corresponds to a location of p'.

Definition 9 (from agents to location automata). Let p_0 be a pure CCS agent and l_0 be the minimal[6] location of *Loc*. The location automaton $\mathbf{aut}(p_0) = \langle Q, \mathrm{loc}, \longmapsto, p_0 \rangle$ is so defined: $l_0 :: p_0 \in Q$ and whenever $p \in Q$ then:

- if $p \xrightarrow[ulm]{a} p'$, with $m \notin \mathrm{loc}(p)$, and $\langle p'', \sigma \rangle = \mathbf{norm}(p')$ then $p'' \in Q$ and $p \stackrel{a}{\underset{l}{\longmapsto}}_{[*/m] \circ \sigma} p''$;
- if $p \xrightarrow{\tau} p'$ and $\langle p'', \sigma \rangle = \mathbf{norm}(p')$ then $p'' \in Q$ and $p \stackrel{\tau}{\longmapsto}_\sigma p''$.

Notice that the locations associated to a state are exactly the locations that appear syntactically in the state.

In the previous definition, when we deal with visible transitions we require that $m \notin \mathrm{loc}(p)$; as stated in the remarks after Definition 3, this does not reduce the discriminating power. Moreover, since we start from $l_0 :: p_0$ and normalize the reached states as described in Proposition 6, it is easy to show that all the visible transitions considered in the construction have the form $p \xrightarrow[lm]{a} p'$ (i.e., $u = \epsilon$). Finally, the particular location m which is chosen as the new location does not play any role in the construction of the location automaton, due to the

[6] Remember that set *Loc* is totally ordered and denumerable.

use of $\star$ to denote the created location and to the normalization of the target agent p'.

Theorem 10. *Given two pure agents p and q, $p \approx_l q$ iff* $\mathtt{aut}(p) \approx_{la} \mathtt{aut}(q)$.

The proof of this theorem is based on the alternative characterization $\approx_{\Delta l}$ of the location equivalence given in Proposition 5.

The previous theorem also holds for other definitions of the function **norm**: for instance **norm** can be defined simply as the identity or may only perform a renaming of the locations. In these cases an infinite automaton would correspond to finite state agents such as $\mathrm{rec}\, x.a.x$. Actually, function **norm** can be used to implement different tricks to reduce the computation time in an heuristic way. The normalization function we have chosen can be computed very efficiently and allows to build finite location automata for the class of finitary CCS agents.

Proposition 11. *A pure CCS agent is finitary iff the corresponding location automaton is finite.*

5 Irredundant Automaton and Unfolding

In the location automaton, not all the locations associated to a state are involved in the computations that can be performed starting from the state. The locations that are never used in these computations can be safely deleted, obtaining a more compact structure.

Definition 12 (active locations). Given a location automaton A, the sets of *active locations* corresponding to the states of A, denoted by $\mathrm{al}(p)$ with $p \in Q_A$, are the smallest sets such that:

- if $p \overset{a}{\underset{l}{\longmapsto}}_\sigma p'$ then $l \in \mathrm{al}(p)$;
- if $p \overset{a}{\underset{l}{\longmapsto}}_\sigma p'$, $m \in \mathrm{al}(p')$ and $\sigma(m) \neq \star$ then $\sigma(m) \in \mathrm{al}(p)$;
- if $p \overset{\tau}{\longmapsto}_\sigma p'$ and $m \in \mathrm{al}(p')$ then $\sigma(m) \in \mathrm{al}(p)$.

Definition 13 (irredundant reduction). Let $A = \langle Q, w, \mapsto, q_0 \rangle$ be a location automaton. Its *irredundant reduction* is the location automaton $\Downarrow A = \langle Q, \mathrm{al}, \mapsto', q_0 \rangle$ where $\mapsto'$ is obtained from $\mapsto$ by restricting the transition renamings concerning a state p to the active locations $\mathrm{al}(p)$.

We say that an automaton A is *irredundant* if $\Downarrow A = A$.

Proposition 14. *Let A be a location automaton. Then $\Downarrow A \approx_{la} A$.*

A location automaton A can be visited beginning from the initial state. In this visit, the global meaning of the private locations of the reached states should be

remembered[7]. If the global meaning corresponding to the locations of a reached state p is given by $\sigma : \mathrm{loc}(p) \hookrightarrow Loc$ and transition $p \overset{a}{\underset{l}{\mapsto}}_\rho q$ is followed, the global meaning for q is given essentially by $\sigma \circ \rho$. However, a global meaning has to be associated also to the location created in the transition (the location of the target state mapped in $\star$ by the transition renaming). To this purpose we use a function **new**, which gets a transition $p \overset{a}{\underset{l}{\mapsto}}_\rho p'$ and a global meaning σ for the locations of p and returns a new location name. A possible definition of **new** is as follows:

$$\mathbf{new}(p \overset{a}{\underset{l}{\mapsto}}_\rho p', \sigma) = \begin{cases} \sigma(l) & \text{if } l \notin \rho(w(p')) \\ \min\{Loc \setminus \sigma(\rho(w(p')))\} & \text{otherwise} \end{cases}$$

This function **new** reuses the same location in which the action occurs if this location is not used in the target state anymore; if it is still used, it chooses the first unused location of the target state. Following this definition, if a sequential process of the form $a.p$ is acting in location l, after the execution of a the process p is still located in l (no generation of new locations is needed in this case).

To formalize the idea of visiting a location automaton A, we associate to A a standard labeled transition system (called the *unfolding* of A); each state of the unfolding is a pair ⟨state of the location automaton, global meaning of its location⟩ and each visible transition has the form

$$\langle p, \sigma \rangle \xrightarrow[l[m]]{a} \langle p', \sigma' \rangle$$

where a is an action, l is the location in which the action occurs and m is the newly created location.

Definition 15 (unfolding). The *unfolding* corresponding to a location automaton $A = \langle Q, w, \mapsto, q_0 \rangle$ is the labeled transition system $\mathbf{unf}(A) = \langle Q_u, \rightarrow, q_{0u} \rangle$ defined as follows:

- the initial state is $q_{0u} = \langle q_0, \sigma_0 \rangle \in Q_u$, where σ_0 maps the location corresponding to q_0 into the minimal location l_0;
- if $\langle p, \sigma \rangle \in Q_u$ and $p \overset{a}{\underset{l}{\mapsto}}_\rho p'$ then $\langle p', \sigma' \rangle \in Q_u$ and $\langle p, \sigma \rangle \xrightarrow[\sigma(l)[m]]{a} \langle p', \sigma' \rangle$, where $\sigma' = (\sigma \cup (\star \mapsto m)) \circ \rho$ and $m = \mathbf{new}(p \overset{a}{\underset{l}{\mapsto}}_\rho p', \sigma)$;
- if $\langle p, \sigma \rangle \in Q_u$ and $p \overset{\tau}{\underset{}{\mapsto}}_\rho p'$ then $\langle p', \sigma' \rangle \in Q_u$ and $\langle p, \sigma \rangle \xrightarrow{\tau} \langle p', \sigma' \rangle$, where $\sigma' = \sigma \circ \rho$.

It is easy to show that there are la-equivalent automata with non-equivalent unfoldings. This happens because two corresponding states of the location automata can have a different number of locations, and this can lead to different choices in the unfoldings when a new location has to been chosen.

[7] A state can be visited more than once, with different meanings for the private locations.

In fact axiom **Del** is not sufficient to erase all the inactive locations. For instance consider

$$l :: \alpha.p \,|\, m :: (\beta.q \setminus \beta)$$

In this case location m guards a deadlocked process, so it in not active in the agent. In the agent

$$(l :: a.\gamma.b.p \,|\, m :: \bar{\gamma}.\mathrm{nil}) \setminus \gamma$$

location m is non active since the process guarded by m can only act as a partner of an invisible transition.

The following theorem express the main result of this paper: given two irredundant location automata, then they are la-equivalent if and only if the corresponding unfoldings are equivalent. This allows to apply a standard partitioning algorithm for checking the equivalence of two automata and to obtain minimal (standard) automata corresponding to them.

Theorem 16. *If A and B are irredundant location automata then $A \approx_{la} B$ iff* $\mathbf{unf}(A) \approx \mathbf{unf}(B)$.

Corollary 17. *Given two pure agents p and q, $p \approx_l q$ iff* $\mathbf{unf}(\Downarrow\mathbf{aut}(p)) \approx \mathbf{unf}(\Downarrow\mathbf{aut}(q))$.

6 Partitioning Algorithm and Complexity

Corollary 17 suggests an algorithm for checking location equivalence of two CCS agents p and q:

1. construct (separately) the location automata corresponding to p and q;
2. discover (separately) the active locations of the two automata and get the irredundant reductions: start marking the locations that are active due the first condition of Definition 12 and continue marking all the locations reachable following the dependencies in the other conditions of Definition 12; at the end discard the unmarked locations;
3. unwind (separately) the obtained irredundant automata;
4. use a standard algorithm for checking the weak equivalence of the obtained transition systems (for instance, partition refinement [13]).

This algorithm works for all finitary agents, since for these we are sure that finite location automata (and hence finite unwindings) can be built.

The following proposition gives a bound to the time complexity of checking location equivalence for finitary agents in terms of the syntactical length and of the maximal reachable degree of parallelism.

Proposition 18. *Let p and q be pure CCS agents. If h is their syntactical length and*

$$\max\{\mathrm{par}(r) \mid p \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_i} r \text{ or } q \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_i} r\} = k,$$

the location equivalence of p and q can be checked in $2^{\mathcal{O}(k \cdot (\log h + \log k))}$.

For the standard CCS equivalence (still using the structural axioms), the corresponding bound is $2^{\mathcal{O}(k \cdot \log h)}$; so the two bounds coincide for the class of agents for which the maximal reachable degree of parallelism k is polynomially bounded in the syntactical size[8]. In particular this is true for the class of agents without parallel composition within a recursive definition.

Notice that, though the upper bounds are very similar, there exist particular agents for which the complexity of checking the ordinary (interleaving) equivalence is substantially smaller than the upper bound, whereas the complexity of checking the location equivalence is close to the bound. This happens in all the cases in which the agents consist of many sequential processes in parallel but only a small number of global states can be reached from them.

An extreme example is given by the agent

$$p = p_1 \,|\, p_2 \,|\, \cdots \,|\, p_n \quad \text{with} \quad p_i = \operatorname{rec} x.\alpha_i.x$$

The only standard transitions the agent can perform are $p \xrightarrow{\alpha_i} p$ so just one state is required in this case. If we consider location transitions, instead, there are lots of reachable states, since initially all the processes p_i share the same location and new locations are created every time a process p_i acts for the first time, leading to exponentially many configurations.

7 Concluding Remarks

In this paper we associate to each CCS agent a labeled transition system in which locality informations appear in the labels, so that location equivalent agents are mapped into transition system which are bisimilar according to the ordinary notion of bisimulation. As a consequence, standard algorithms can be used on these transition systems. The worst-case complexity is similar to that of pure CCS.

The use of locations in the transition systems obtained after unfolding is similar to the one proposed in [6]; however, since no notion of active locations was present in [6], Theorem 16 and Corollary 17 do not hold in that context; there, the construction of the bijection between the locations of two agents that are checked for equivalence can be avoided — and hence ordinary algorithms can be used — only by avoiding location re-use, which leads to infinite transition systems for finitary agents like $\operatorname{rec} x.a.(b.\delta.d.x \,|\, c.\bar{\delta}.\mathrm{nil}) \setminus \delta$.

The theory has been presented on CCS, since it is simple, it has been used to describe both specifications and programs and because it is the language used in the original presentation of location semantics. It is important to point out that our result is not related to CCS: our technique can be used for any language, provided it can be equipped with an operational semantics with locations.

Location automata in particular seem to us a quite general operational model for dealing with location semantics. Our hope is that also different approaches to

[8] It is interesting to notice that a computable function does not exists that bounds k in function of h for all the finitary agents.

locality semantics can be mapped into this model, so that the results of Section 5 can be re-applied.

As stated in the Introduction, the reduction of location equivalence to ordinary bisimulation equivalence is important to obtain minimal realizations. These are interesting both from a theoretical point of view — equivalent agents give rise to the same (up to isomorphism) minimal realization — and a practical point of view — smaller state spaces can be obtained. It is important to stress out, however, that the realizations are *minimal* for the particular choice of function **new**. Different definitions are possible, which, for some particular agents, can give rise to dramatically smaller "minimal" realizations. Matter of further research is the possibility of defining better or optimal **new** functions, which still allow to map equivalent irredundant automata to equivalent transition systems.

References

1. L. Aceto. A static view of localities. INRIA Report 1483, 1991. To appear in *Formal Aspects of Computing.*
2. G. Berry and G. Boudol. The chemical abstract machine. In *Proc. POPL.* ACM, 1990.
3. G. Boudol, I. Castellani, M. Hennessy and A. Kiehn. Observing localities. *Theoretical Computer Science*, 114: 31–61, 1993.
4. G. Boudol, I. Castellani, M. Hennessy and A. Kiehn. A theory of processes with localities. INRIA Report 1632, 1991. Extended abstract in *Proc. CONCUR'92*, LNCS 630, 1992.
5. I. Castellani. Observing distribution in processes: static and dynamic localities. INRIA Report 2276, 1994.
6. F. Corradini and R. De Nicola. Distribution and locality of concurrent systems. In *Proc. ICALP'94*, LNCS 920, pages 154–165. Springer Verlag, 1994.
7. P. Inverardi and C. Priami. Evaluation of tools for the analysis of communicating systems. In *Bullettin of EATCS*, 45, 1991.
8. A. Kiehn. Local and global causes. Tech. Rep. 42/23/91, Institut für Informatik, TU München, 1991.
9. R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.
10. R. Milner. The polyadic π-calculus: a tutorial. In *Logic and Algebra of Specification*, NATO ASI Series F, Vol. 94. Springer Verlag, 1993.
11. U. Montanari and D. Yankelevich. A parametric approach to localities. In *Proc. ICALP'92*, LNCS 623. Springer Verlag, 1992.
12. U. Montanari and D. Yankelevich. Location Equivalence in a Parametric Setting. *Theoretical Computer Science*, 149: 299–332, 1995.
13. R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.
14. D. Yankelevich. *Parametric Views of Process Description Languages.* PhD Thesis. Dipartimento di Informatica, Università di Pisa, 1993. Available as report TD-23/93.
15. *Proceedings of the Conference on Computer-Aided Verification – CAV'95*, LNCS 939. P. Wolper Ed., Springer Verlag, 1995.
16. *Proceedings of the International Symposium on Protocol Specification, Testing and Verification – PSTV'95.* IFIP WG 6.1, 1995.