Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 137 (2005) 151–174
www.elsevier.com/locate/entcs

Tagged Systems: A Framework for the Specification of History-dependent Properties*

Fernando Rosa-Velardo Clara Segura David de Frutos-Escrig

Departamento de Sistemas Informáticos y Programación
Universidad Complutense de Madrid, Spain
e-mail: {fernandorosa,csegura,defrutos}@sip.ucm.es

Abstract

We propose a general method for the treatment of history-dependent runtime errors. When one has to control this kind of errors, a tagged version of the language is usually defined, in which tags capture only the necessary information of the history of processes. We will characterize such tagged languages as being quotients of the reachability tree defined by the computations of the original language. From this fact we can conclude that the property characterized by each tagged language is indeed a property of the original one. In this way, we can work in a common framework, instead of defining an ad hoc semantics for each property. In particular, we could still use the analysis machinery existing in the calculus in order to prove that or other related properties. We have applied this methodology to the study of resource access control in a distributed π-calculus, called $D\pi$. In particular, we have proved that the tagged version of $D\pi$ is indeed a *tagging* according to our definition.

Keywords: Security properties, enhanced semantics, static analyses.

1 Introduction

Several frameworks for the definition of security properties can be found in the literature [7,8,3,4,1]. In order to specify a property sometimes it is enough to detect some erroneous states of the analyzed systems. Some of those erroneous states can be statically identified, regardless the way in which the

* Work partially supported by the Spanish project TIC 2003-01000.

doi:10.1016/j.entcs.2005.01.043

error-prone state has been reached. An example of such a security property is *confidentiality*, in which the erroneous states are those where a secret piece of data is being exhibited. [1]

On the other hand, some security properties cannot be proved to hold by mere inspection of the current state. This is the case of history-dependent properties. An example of scenario in which this happens is when in order to access a resource, a process needs to have been given prior permission to use it [7].

In the former case, all that is needed to ensure that no error can occur is to identify those erroneous states, for example by means of a suitable type system, in such a way that they will not be typeable. Then, if the original system is typeable and we have subject reduction for the typed semantics, we can guarantee that no error can ever occur.

However, in the latter case, this is not possible, as errors are not a property of a single state, but of the whole computation. The most natural way of performing the necessary trace analysis is by somehow remembering along the computation the information that is needed for the characterization of the considered kind of error, tagging the processes with this information. In the previous papers on the subject, such as [7] or [10], the corresponding enriched semantics is defined in an ad hoc way, not relating it in an explicit way with the original semantics, thus making necessary specific non-reusable proofs for the corresponding relation between the original and the enriched semantics.

We propose in this paper a systematic methodology for the treatment of these errors, that begins by defining a tagged language and a unary relation on tagged processes. We will characterize such tagged languages as being quotients of the reachability tree defined by the computations of the original language. From this fact we can conclude that the property characterized by each tagged language is indeed a property of the original one. In this way, we can work in a common framework, instead of defining an ad hoc semantics for each property. In particular, we could again use a type-based analysis to guarantee that no erroneous tagged state is ever reached along the computation.

Additionally, we prove that the set of quotient systems is a complete lattice and profit from its algebraic structure by defining the merging of different taggings of the same system in order to capture the conjunction of two properties.

The remainder of the paper is organized as follows: in Section 2 we define tagged transition systems; in Section 3 we define quotient transition systems

[1] Clearly by means of such a simple property we could not cover implicit flows of information.

and prove that they are essentially tagged transition systems. Section 4 shows simple ways of deriving tagged languages; in Section 5 we present an example of the use of this methodology: we show that tagged $D\pi$ [7,6] is a tagging according to our definition and, therefore, a quotient. Section 6 exploits the algebraic structure of quotient systems lattice by defining the combination of different taggings of the same system. Finally, Section 7 presents our conclusions and proposes some future work.

2 Tagged Transition Systems

In this section we will formally define what a tagged language will be for us. We start by defining the tagged syntax generated by a given many-sorted signature [2,5] with respect to a symbol of the signature and the considered set of labels.

Definition 2.1 Given a many-sorted signature $(\mathcal{S}, \Sigma)$, let E be the $\mathcal{S}$-sorted term algebra of Σ, f a symbol of Σ and $\mathcal{L}$ a set of labels ($\Gamma, \Delta, \ldots \in \mathcal{L}$). We will call $(f, \mathcal{L})$-tagged syntax of E, and we will denote it by $\bar{E}_f^{\mathcal{L}}$ (or sometimes just by $\bar{E}$), to the $\mathcal{S}$-sorted term algebra of $\Sigma_f^{\mathcal{L}} = (\Sigma - \{f\}) \cup \{f_\Gamma | \Gamma \in \mathcal{L}\}$, where $arity(f_\Gamma) = arity(f)$ for every $\Gamma \in \mathcal{L}$.

In terms of abstract syntax trees, tagged terms are usual terms where some nodes, either leaves in case the distinguished symbol f is a constant operator, or internal nodes otherwise, are annotated with labels in $\mathcal{L}$. A more general version of tagged syntax, $\bar{E}_{f_1,f_2,\ldots}^{\mathcal{L}_1,\mathcal{L}_2,\ldots}$, would consider a collection of symbols $f_1, f_2, \ldots$ in Σ, and a (possibly) different set of labels for each one of them.

Example 2.2 Let us consider the signature $(\mathcal{S}, \Sigma)$, where $\mathcal{S} = \{Nat\}$ and $\Sigma = \{Zero, Succ, +, \times\}$, with $arity(Zero) = Nat$, $arity(Succ) = Nat \to Nat$ and $arity(+) = arity(\times) = Nat \to Nat \to Nat$. Let us also consider the set of labels $\mathcal{L} = \{\square, \bigcirc\}$. The $\mathcal{S}$-sorted term algebra of Σ defines a set of arithmetic expressions. One possible tagged syntax is that generated by $\Sigma_{+,\times}^{\mathcal{L},\mathcal{L}} = \{Zero, Succ, \oplus, \otimes, \boxplus, \boxtimes\}$, which defines ordinary expressions in which the operation symbols are substituted by "encircled" or "framed" symbols. Figure 1 shows the abstract syntax trees of expressions $1+(0\times1)$ and $1\oplus(0\boxtimes1)$.

We will use ordinary labelled transition systems.

Definition 2.3 A *labelled transition system* S is a tuple $S = (E, V, \to, e_0)$, where E is the set of states, V is the set of transition labels, $e_0 \in E$ is the initial state and $\to \subset E \times V \times E$ is the transition relation. If $(e_1, a, e_2) \in \to$ we will write $e_1 \xrightarrow{a} e_2$.

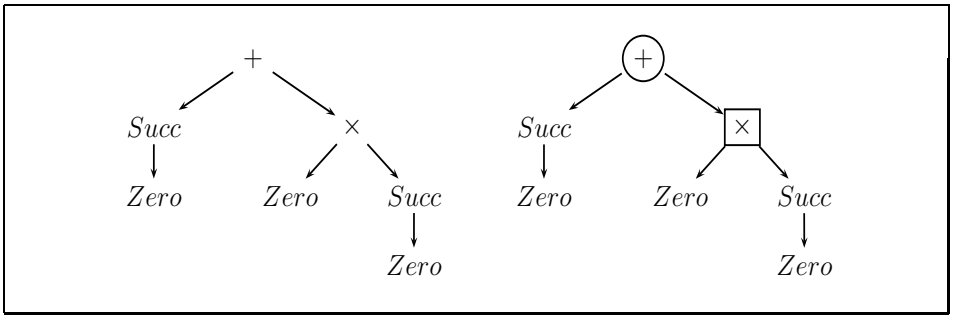


Figure 1. Abstract syntax trees for $1 + (0 \times 1)$ and $1 \oplus (0 \boxtimes 1)$

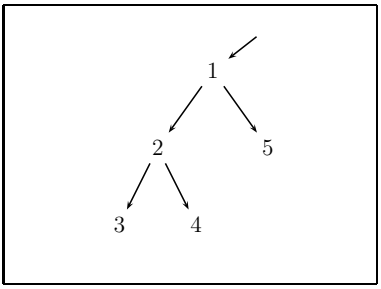


Figure 2. Transition System of Example 2.6

We will write $e \Rightarrow e'$ if there exist $a_1, \ldots, a_n \in V$ and $e_1, \ldots, e_n \in E$ such that $e \xrightarrow{a_1} e_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} e_n = e'$. We will call the latter a computation sequence, $\mathcal{C}$ the set of computation sequences, and $Reach(S) = \{e \in E \mid e_0 \Rightarrow e\}$ the set of reachable states.

Definition 2.4 Let $S = (E, V, \rightarrow, e_0)$ and $T = (F, V, \mapsto, f_0)$ be two labelled transition systems. We will say that S and T are isomorphic, and we will write $S \approx T$, if there exists a mapping $h : Reach(S) \rightarrow Reach(T)$ such that:

(i) h is a bijection between $Reach(S)$ and $Reach(T)$.

(ii) $h(e_0) = f_0$

(iii) $e \xrightarrow{a} e' \Leftrightarrow h(e) \overset{a}{\mapsto} h(e')$ for all $e \in Reach(S)$

We will say that such h is an isomorphism of labelled transition systems.

Next we will define when a transition system over the tagged syntax is a *tagging* of the original transition system. We will denote by $O(e)$ the term obtained from a tagged term e once we have removed all the tags in it, defined by structural induction as:

$$O(h(\overline{e_1}, \ldots, \overline{e_n}) = h(O(\overline{e_1}), \ldots, O(\overline{e_n})) \text{ if } f \neq h$$
$$O(f_\Gamma(\overline{e_1}, \ldots, \overline{e_n}) = f(O(\overline{e_1}), \ldots, O(\overline{e_n}))$$

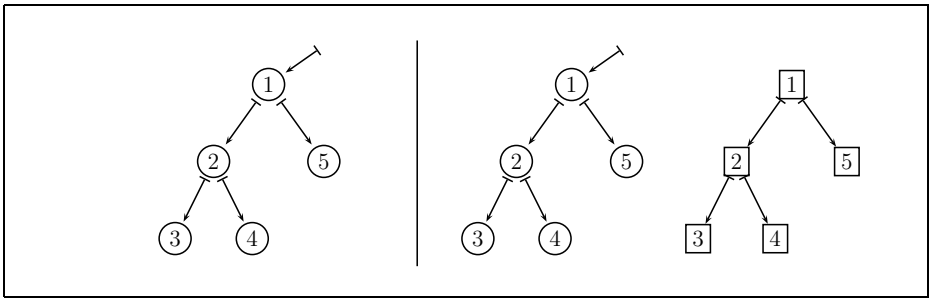


Figure 3. Tagged Systems of Example 2.6

Definition 2.5 Let E be the $\mathcal{S}$-sorted term algebra of Σ and $\bar{E}$ a $(f, \mathcal{L})$-tagged syntax of E. Also let $S = (E, V, \rightarrow, e_0)$ and $T = (\bar{E}, V, \mapsto, \bar{e}_0)$ be two transition systems with $O(\bar{e}_0) = e_0$. We will say that T is a tagging of S if the following conditions hold:

(i) If $\bar{e} \overset{a}{\mapsto} \bar{e}'$ then $O(\bar{e}) \overset{a}{\rightarrow} O(\bar{e}')$.

(ii) If $e \overset{a}{\rightarrow} e'$ then for all $\bar{e}$ reachable from $\bar{e}_0$ such that $O(\bar{e}) = e$ there exists a unique $\bar{e}'$ with $O(\bar{e}') = e'$ such that $\bar{e} \overset{a}{\mapsto} \bar{e}'$.

The first condition is a correctness property for tagged systems: they do not introduce new transitions, that is, all their transitions come from untagged ones. The second is a completeness condition: we have a unique tagged version for each original transition. Besides, uniqueness states that we cannot separately remember different things about the same past.

Example 2.6 Let us consider the sorted signature $(\mathcal{S}, \Sigma)$ where $\mathcal{S} = \{Nat\}$, $\Sigma = \{1, 2, 3, 4, 5\}$ with $arity(i) = Nat$ and the set of tags $\mathcal{L} = \{\circ, \square\}$. The terms obtained when we tag every symbol with tags in $\mathcal{L}$, are just the elements in $\{1, \ldots, 5\}$ enclosed by $\circ$ or $\square$. Now let us consider the transition system defined by the graph in Figure 2. In this simple example, in which the graph defining the transition system is in fact a tree, a tagged system will consist of one or several copies of the original system, with at most one for each different way of annotating the initial symbol, like in the examples shown in Figure 3. However, these copies could partially overlap for non-reachable terms, getting for instance the examples shown in Figure 4.

Notice that in the definition of tagged systems we allow the presence of some *garbage nodes* (under certain conditions), that will not be reachable from the initial state. We do this in order to get a more flexible definition of tagged systems, so that we do not need an exact knowledge of the set of reachable terms to define a tagged system. Of course, in practice, we will try to define them with the least possible amount of garbage, in order to have systems as

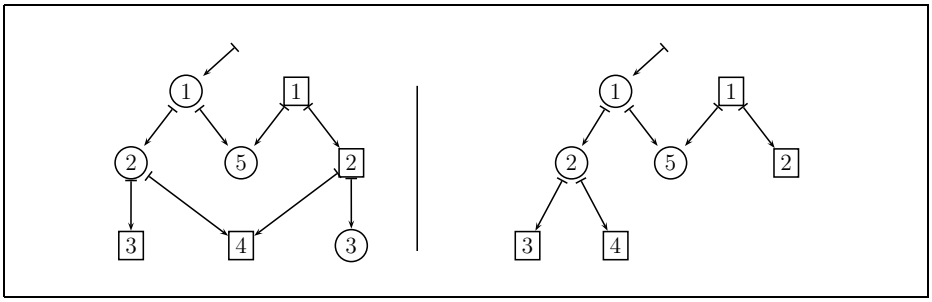


Figure 4. More Tagged Systems of Example 2.6

simple as possible. [2]

Note that in this case all the tagged systems are indeed isomorphic: if we remove every non reachable term they become equal, up to the names of states. This is so because in this simple example the transition graph is a tree and, therefore, each node determines a unique path from the initial node (the transition graph and the reachability tree are isomorphic).

Example 2.7 As a new example, illustrating non isomorphic systems, let us consider the transition system defined by the graph on the left of Figure 5, that is not a tree. In particular, we can interpret it as a system in which an agent can be at three different servers and move from one to other by following the arrows. The corresponding tagged systems on the right can now discriminate between different ways of reaching state 3. If we consider that server 2 is malicious and we do not want our agents to move to 2 then we are interested in the property captured by the tagged system in the middle. There, tag ○ identifies computations in which server 2 has been visited, as opposed to tag □. If we were only interested in which server has been first visited then we would use the transition system on the right.

These simple examples show that it is the existence of several reachable nodes with different labels what makes labels useful: just by checking the label in the reached state we have a (partial) knowledge of the way we reached that node.

3 Alternative characterization of tagged systems

In this section we will characterize the tagged systems defined in the previous section as quotients of their reachability tree. To be more precise, tagged

[2] Garbage nodes are useless, but at the same time they are absolutely innocuous, out of the increasing of the size of the system.

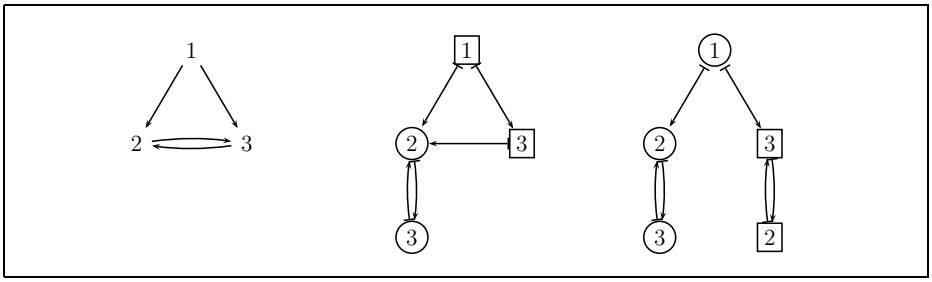


Figure 5. Transition System (left) and Tagged Systems (right) of Example 2.7

systems are isomorphic (in the sense we have defined) to the computation tree, once we have identified some of its nodes. We start by defining the computation trees.

Definition 3.1 Given $S = (E, V, \rightarrow, e_0)$ a labelled transition system and $\mathcal{C}$ the set of computation sequences of S, we define the corresponding computation tree $S^t = (\mathcal{C}, V, \mapsto, e_0)$, as the transition system obtained by taking $\mapsto \subset \mathcal{C} \times V \times \mathcal{C}$, defined by the rule

$$\frac{e \xrightarrow{a} e' \quad last(L) = e}{L \overset{a}{\mapsto} (L \xrightarrow{a} e')}$$

where "*last*" is the function that gives us the last state of each computation sequence.

The computation tree is simply the transition system defined by the reachability tree of the original system. It merely accumulates in its states the sequence of states that have been visited along the computation and the transitions between them. Thus, it contains all the information about the history of the process. We will sometimes call it the *totally tagged system*.

However, in general we will only be interested in a particular aspect of this information. We will use equivalence relations among computations to identify the required information.

Definition 3.2 We will say that an equivalence relation $\sim \subset \mathcal{C} \times \mathcal{C}$ is *suitable* if it satisfies the following conditions:

(i) $L_1 \sim L_2 \Rightarrow last(L_1) = last(L_2)$

(ii) $L_1, L_2 \in \mathcal{C}$ *and* $L_1 \sim L_2 \Rightarrow \forall a \in V$ and $e \in E$ such that $last(L_1) \xrightarrow{a} e$ (and therefore $last(L_2) \xrightarrow{a} e$) it holds $(L_1 \xrightarrow{a} e) \sim (L_2 \xrightarrow{a} e)$.

For example, the relations $\sim_\perp$ such that $L_1 \sim_\perp L_2 \Leftrightarrow last(L_1) = last(L_2)$ and $\sim_\top$ such that $L_1 \sim_\top L_2 \Leftrightarrow L_1 = L_2$ are trivial examples of suitable equivalences.

By means of suitable equivalences we will characterize the information about the past we are interested in. For instance, $\sim_\perp$ is the relation that forgets everything about the past, while $\sim_\top$ is that remembering absolutely everything. Now we define the quotient transition system generated by such an equivalence relation.

Definition 3.3 Let $S = (E, V, \rightarrow, e_0)$ be a labelled transition system and $\sim \subset \mathcal{C} \times \mathcal{C}$ a suitable equivalence relation. We define $S^\sim = (\mathcal{C}/_\sim, V, \mapsto_\sim, [e_0])$, and call it a quotient transition system of S with respect to $\sim$, as the transition system defined by the rule

$$\frac{L \overset{a}{\mapsto} L'}{[L] \overset{a}{\mapsto}_\sim [L']}$$

where $L, L' \in \mathcal{C}$.

The first condition in Definition 3.2 is introduced so that the quotient transition system is uniformly defined, in the sense that its transitions do not depend on the choice of the representatives of the equivalence classes. The second condition states that the relation remembers at least the final state reached by the computation.

For the equivalences $\sim_\perp$ and $\sim_\top$ defined above, it turns out that $S^{\sim_\perp}$ and $S^{\sim_\top}$ are isomorphic to S and S^t, respectively.

Quotient transition systems are a conservative extension of ordinary transition systems in the following sense.

Proposition 3.4 *If* $[L_1], [L_2] \in \mathcal{C}/_\sim$ *are two equivalence classes such that* $[L_1] \overset{a}{\mapsto}_\sim [L_2]$ *then* $last(L_1) \overset{a}{\rightarrow} last(L_2)$.

Proof Let us suppose that $[L_1] \overset{a}{\mapsto}_\sim [L_2]$. By definition of $\mapsto_\sim$ there exist L_1' and L_2' such that $L_1 \sim L_1'$, $L_2 \sim L_2'$ and $L_1' \overset{a}{\mapsto} L_2'$. As well, by definition of $\mapsto$, it holds that $last(L_1') \overset{a}{\mapsto} last(L_2')$ and, since $\sim$ was suitable and $L_i \sim L_i'$, it follows that $last(L_i) = last(L_i')$ and, therefore, we are done. □

Now let us take a closer view at the structure of the set of quotient transition systems.

Definition 3.5 We denote by $\mathbf{S}$ the set of quotient transition systems, that is, $\mathbf{S} = \{S^\sim \mid \sim \text{ suitable}\}$, and define the operators $\sqcup$ and $\sqcap$ by $S^{\sim_1} \sqcup S^{\sim_2} \overset{\triangle}{=} S^{\sim_1 \cap \sim_2}$ and $S^{\sim_1} \sqcap S^{\sim_2} \overset{\triangle}{=} S^{\sim_1 \cup^* \sim_2}$, where $R_1 \cup^* R_2$ is the transitive closure of $R_1 \cup R_2$.

The order induced by those operators is $S^{\sim_1} \preceq S^{\sim_2} \Leftrightarrow \sim_1 \supseteq \sim_2$, that is, the more the information captured by the relation, the upper the corresponding quotient system is in the lattice. Then we have the following

Proposition 3.6 $(\boldsymbol{S}, \sqcap, \sqcup)$ *is a complete lattice that has* S *and* S^t *as bottom and top elements, respectively.*

Proof First, $(\mathbf{S}, \sqcap, \sqcup)$ is a lattice, since both $\sqcap$ and $\sqcup$ are idempotent, commutative, associative and satisfy the absorption law. To prove that it is complete we have to see that every subset of $\mathbf{S}$ has a supremum and an infimum. Let $\{S^{\sim_i} \mid i \in I\}$ be a subset of $\mathbf{S}$ and take $\sim_{inf} = \bigcup^*_{i\in I} \sim_i$ and $\sim_{sup} = \bigcap_{i\in I} \sim_i$. Both $\sim_{inf}$ and $\sim_{sup}$ are suitable equivalence relations and $S^{\sim_{inf}}$ and $S^{\sim_{sup}}$ are the infimum and the supremum of $\{S^{\sim_i} \mid i \in I\}$, respectively. Also, if $\sim$ is a suitable equivalence relation then $\sim_\top \subseteq \sim \subseteq \sim_\bot$ and, therefore $S^{\sim_\bot} \preceq S^{\sim} \preceq S^{\sim_\top}$. To conclude, it remains to see that S and S^t are isomorphic to $S^{\sim_\bot}$ and $S^{\sim_\top}$, respectively. The latter is trivial. Let us see the former. Let us consider the function h given by $h(e) = [e]_{\sim_\bot}$ for $e \in E$, that is, the quotient mapping restricted to empty computations sequences. Let us see that h is a bijection satisfying the required conditions. First, h is bijective since it has $\overline{last}([L]_{\sim_\bot}) = last(L)$ as inverse. Note that it is well defined since $\sim_\bot$ is suitable. Moreover, since $[e \xrightarrow{a} e']_{\sim_\bot} = [e']_{\sim_\bot}$, it follows that $e \xrightarrow{a} e' \Leftrightarrow h(e) \overset{a}{\mapsto} h(e')$, for all $e \in Reach(S)$, as required by Definition 2.4. □

Now let us state the main theorem of this section.

Proposition 3.7 *Let* $S = (E, V, \rightarrow, e_0)$ *be a labelled transition system and* $T = (\bar{E}, V, \mapsto, \bar{e}_0)$ *a tagging of* S*. Then* T *is isomorphic to a quotient transition system of* S*.*

Proof Let us suppose that T is a tagging of S. We must find a suitable relation $\sim$, such that T will be isomorphic to $S^{\sim}$. We will get it by defining a function π over the computation sequences of S such that $O(\pi(L)) = last(L)$, and taking $\sim$ as the kernel of π, that is, $L_1 \sim L_2 \overset{\triangle}{\Leftrightarrow} \pi(L_1) = \pi(L_2)$. For each computation L, we define $\pi(L)$ by induction on its length. The base case is straightforward, $\pi(e_0) = \bar{e}_0$, that satisfies $O(\pi(e_0)) = e_0 = last(e_0)$. Let us suppose that we have just defined $\pi(L) = \bar{e}$ and we want to define $\pi(L \xrightarrow{a} e')$. If $e = last(L)$, since $e \xrightarrow{a} e'$ and we are assuming that $O(\bar{e}) = e$, there exists a single $\bar{e}'$ satisfying condition 2 of the definition of tagged systems. Then we take $\pi(L \xrightarrow{a} e') = \bar{e}'$, that also satisfies $O(\bar{e}') = last(L \xrightarrow{a} e')$.

It is easy to see that the so defined relation is suitable and that T is isomorphic to $S^{\sim}$, just taking $h([L]) = \pi(L)$, that is well defined by definition of $\sim$, is a bijection between reachable terms and satisfies the conditions to be an isomorphism between transition systems. □

The definition of tagged system induces a commutative diagram for every computation sequence, as depicted in Figure 6. In this diagram function π is

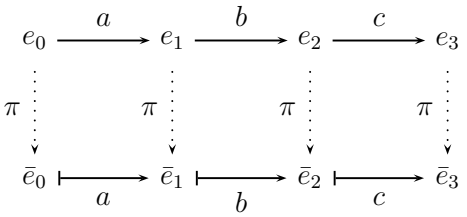


Figure 6. Commutative diagram induced by a tagged system

defined over the computation sequence reaching each e_i. The converse is also true.

Proposition 3.8 *Let $S = (E, V, \rightarrow, e_0)$ be a labelled transition system and $\sim$ a suitable equivalence relation. Then $S^{\sim}$ is isomorphic to a tagging of S.*

Proof Let $S = (E, V, \rightarrow, e_0)$ be a labelled transition system and $\sim$ a suitable relation over its computation sequences. We must define a tagging of S, $\bar{S}$, isomorphic to $S^{\sim}$. First, we will tag every term of the syntax, so that tagged terms can be seen as pairs (e, Γ) with $e \in E$. Now, we take as set of labels the set $\mathcal{C}/_{\sim}$ of equivalence classes defined by the suitable relation $\sim$. Then we define $\mapsto$ by the rule

$$\frac{e \xrightarrow{a} e' \quad last(L) = e}{(e, [L]) \overset{a}{\mapsto} (e', [L \xrightarrow{a} e'])}$$

The transition relation $\mapsto$ is well defined since $\sim$ is suitable. Then, it holds that $\bar{S} = (E \times \mathcal{C}/_{\sim}, V, \mapsto, (e_0, [e_0]))$ is a tagging of S. To conclude the proof, we just have to check that the function $h([L]) = (last(L), [L])$ defines an isomorphism between $\bar{S}$ and $S^{\sim}$, what is immediate. □

In our Example 2.6 the lattice generated by the transition system is degenerate, since the bottom the top elements are isomorphic. Therefore, all the tagged systems are also isomorphic. Let us now study a more complex example.

Example 3.9 Let us now consider the language $D\pi$ [3] [7] and the process $\ell[\![*(go\ \kappa.go\ \ell)]\!]$ which spawns trivial agents that go to location κ and then back to ℓ, where they die. We will use the set of labels $\mathcal{L} = \{\square, \bigcirc\}$ to label the transition system that the term generates. For each natural number n let us denote also by n the term $\kappa[\![go\ \ell]\!] \mid \overset{n}{\ldots} \mid \kappa[\![go\ \ell]\!] \mid \ell[\![*(go\ \kappa.go\ \ell)]\!]$, that is, the term representing the state in which exactly n agents are at κ. In Figure 7

[3] That language will be formally presented in Section 5, although we consider that this particular example can be understood here without going into those formal details.

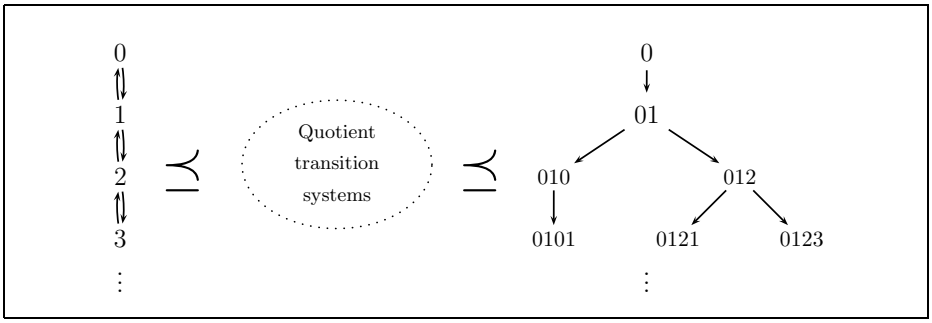


Figure 7. Transition graph (left) and reachability tree (right) of Example 3.9

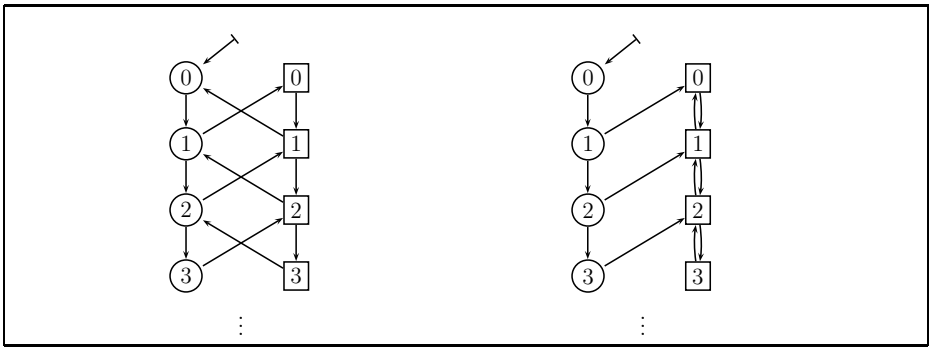


Figure 8. Two quotient transition systems for Example 3.9

we present the transition graph (left) and the corresponding reachability tree (right).

Then the two transition graphs shown in Figure 8 define a couple of quotient transition systems. The one on the left determines the parity of the number of agents that have died, while the one on the right only determines whether any agent has ever died. Therefore, in this second case state (n) corresponds to computation $01 \ldots n$, while state $\boxed{n}$ identifies every computation sequence ending in state n, but $01 \ldots n$. In terms of the used suitable relation, $\sim$ identifies every computation of the form $01 \ldots n$ (the right-branch of the reachability tree) only with itself, while for the rest of the nodes, it relates every computation ending in the same state.

Proposition 3.7 states the existence of a quotient system isomorphic to a tagging of a transition system. However, such quotient system is not necessarily unique, because our notion of isomorphism totally disregards the information of states.

Example 3.10 In Figure 9, the graph on the left defines a simple transition

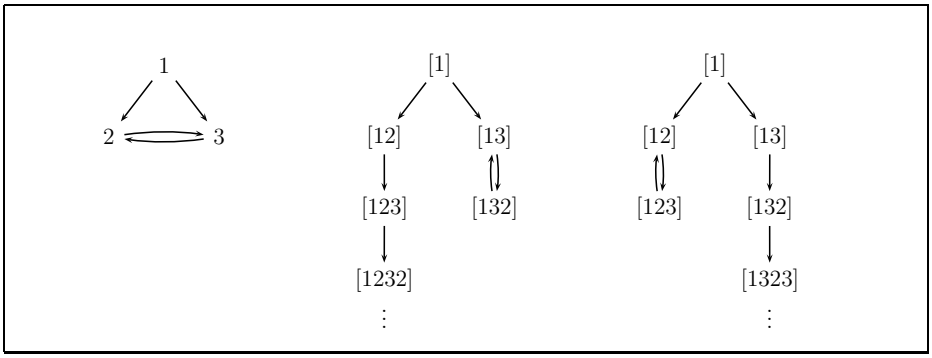


Figure 9. Isomorphic quotient systems of Example 3.10

system and the others are different quotient systems of the latter. The quotient systems are indeed isomorphic but intuitively they capture different properties of the original system: the one on the left identifies those computations starting by 3, while the one on the right identifies the computations starting by 2.

To get uniqueness (see Theorem 3.15) we will need a refined version of isomorphism that also preserves some properties of states. We will formally define these properties by means of functions that map states to a common domain.

Definition 3.11 Given $S = (E, V, \rightarrow, e_0)$ and $T = (F, V, \mapsto, f_0)$ and a function $f : E \cup F \rightarrow States$ mapping states in E and F to a set $States$, we say that a mapping h is a f-isomorphism if:

(i) h is an isomorphism of transition systems.

(ii) $f(h(e)) = f(e)$ for every $e \in Reach(S)$.

We will write $S \approx_f T$ if there exists a f-isomorphism between S and T.

We want our isomorphism to respect the reached state, both in tagged and quotient systems. Therefore we will identify these transition systems up to st-isomorphisms, being $st(\overline{e}) = O(e)$ and $st([L]_\sim) = last(L)$. To be more precise, and abusing notation, when comparing quotient systems we will use $st : \mathcal{C}/_{\sim_1} \cup \mathcal{C}/_{\sim_2} \rightarrow E$, and when comparing a tagged system and a quotient system we will use $st : \bar{E}^{\mathcal{L}}_f \cup \mathcal{C}/_\sim \rightarrow E$. We will need the following two lemmas.

Lemma 3.12 *If* $[L_0]_\sim \overset{a}{\mapsto}_\sim [L_1]_\sim$, $[L_0]_\sim \overset{a}{\mapsto}_\sim [L_2]_\sim$ *and* $last(L_1) = last(L_2)$ *then* $[L_1]_\sim = [L_2]_\sim$.

Proof If $[L_0]_\sim \overset{a}{\mapsto}_\sim [L_1]_\sim$ and $[L_0]_\sim \overset{a}{\mapsto}_\sim [L_2]_\sim$ then there exist L_0', L_0'', L_1' and L_2' such that $L_0' \sim L_0 \sim L_0''$, $L_i \sim L_i'$ for $i = 1, 2$, $L_0' \overset{a}{\mapsto} L_1'$ and $L_0'' \overset{a}{\mapsto} L_2'$.

Since $last(L_1) = last(L_2)$ and $\sim$ is suitable, it follows that there exists e such that $L_1' = L_0' \stackrel{a}{\mapsto} e$ and $L_2' = L_0'' \stackrel{a}{\mapsto} e$. Again, since $L_0' \sim L_0''$ and $\sim$ is suitable it holds $L_1' \sim L_2'$. By transitivity of $\sim$ we get $L_1 \sim L_2$ and therefore $[L_1]_\sim = [L_2]_\sim$. □

Lemma 3.13 *If the quotient systems $S^{\sim_1}$ and $S^{\sim_2}$ are st-isomorphic then there exists one and only one st-isomorphism between $S^{\sim_1}$ and $S^{\sim_2}$, namely the function h defined by $h([L]_{\sim_1}) = [L]_{\sim_2}$.*

Proof Let h be a *st*-isomorphism between $S^{\sim_1}$ and $S^{\sim_2}$. Let us see that $h([L]_{\sim_1}) = [L]_{\sim_2}$ by induction on the length of L. If $L = e_0$ then $h([e_0]_{\sim_1}) = [e_0]_{\sim_2}$ because h is an isomorphism and $[e_0]_{\sim_1}$ and $[e_0]_{\sim_2}$ are the initial states of $S^{\sim_1}$ and $S^{\sim_2}$, respectively. Now suppose $L = L' \stackrel{a}{\to} e$. The induction hypothesis tells us that $h([L']_{\sim_1}) = [L']_{\sim_2}$. Since $L' \stackrel{a}{\mapsto} L$ then $[L']_{\sim_i} \stackrel{a}{\mapsto}_{\sim_i} [L]_{\sim_i}$. Since h is an isomorphism, for $i = 1$ we get $h([L']_{\sim_1}) = [L']_{\sim_2} \stackrel{a}{\mapsto}_{\sim_2} h([L]_{\sim_1})$. Let $L'' \in \mathcal{C}$ such that $[L'']_{\sim_2} = h([L]_{\sim_1})$. Since h preserves states, $last(L'') = st(h([L]_{\sim_1})) = st([L]_{\sim_1}) = last(L)$. Then we can apply the previous lemma to obtain $h([L]_{\sim_1}) = [L'']_{\sim_2} = [L]_{\sim_2}$. □

Lemma 3.14 *If $S^{\sim_1} \approx_{st} S^{\sim_2}$ then $\sim_1 = \sim_2$.*

Proof Since $S^{\sim_1} \approx_{st} S^{\sim_2}$ there exists a *st*-isomorphism h as in the previous lemma. Now, $L \sim_1 L' \Leftrightarrow [L]_{\sim_1} = [L']_{\sim_1} \Leftrightarrow h([L]_{\sim_1}) = h([L']_{\sim_1}) \Leftrightarrow [L]_{\sim_2} = [L']_{\sim_2} \Leftrightarrow L \sim_2 L'$, where the second equivalence holds because h is bijective. □

Notice that Example 3.10 was not a counterexample for the previous lemma since $S^{\sim_1}$ and $S^{\sim_2}$ were isomorphic but, in fact, they were not *st*-isomorphic. Now we can prove our theorem.

Theorem 3.15 *If T is a tagging of S then there exists a sigle quotient $S^\sim$ such that $T \approx_{st} S^\sim$.*

Proof

- Existence: the equivalence $\sim$ and the mapping h, as defined in the proof of Proposition 3.7, that is $h([L]_\sim) = \pi(L)$, are such that h defines an isomorphism between T and $S^\sim$. Moreover, h is a *st*-isomorphism since $st(h([L]_\sim)) = st(\pi(L)) = last(L) = st([L]_\sim)$.
- Uniqueness: Suppose that there exist $\sim_1$ and $\sim_2$ such that $S^{\sim_1} \approx_{st} T$ and $T \approx_{st} S^{\sim_2}$. By transitivity of $\approx_{st}$ it holds that $S^{\sim_1} \approx_{st} S^{\sim_2}$ and therefore, applying the previous lemma, $S^{\sim_1} = S^{\sim_2}$.

□

4 Deriving Tagged Languages

Now we address the problem of deciding whether an operational semantics, defined by a reduction relation over a tagged syntax, is in fact a tagging (and therefore a quotient) of the original reduction semantics, which was our initial goal.

Tagged semantics will be parameterized over the set of possible initial tagging functions $tag : E \to \bar{E}$, that will depend on the specific application we are interested in. In particular, tagging functions will satisfy $O \circ tag = Id_E$, but not the other way around. Typically, this function will capture on tags the static information needed to formalize the property in which we are interested.

In general, a semantics over the tagged syntax is a tagging of another operational semantics with respect to an initial tagging function tag if for every P the transition system that generates $tag(P)$ is a tagging of the transition system generated by P.

Definition 4.1 We will say that a reduction relation $\bar{\mathcal{R}} = (\bar{E}, V, \mapsto)$ is a tagging of $\mathcal{R} = (E, V, \to)$ with respect to the initial tagging function tag if for every $e \in E$ the labelled transition system $(\bar{E}, V, \mapsto, tag(e))$ is a tagging of $(E, V, \to, e)$.

We will just consider a particular case, that will be enough for our current purposes: that in which the reduction relation is structurally defined. Let us suppose that $\to$ is defined by means of a set of axioms $A_i : P_i \stackrel{a_i}{\to} Q_i$ for $i = 1, 2, \ldots$ and a set of structural rules R_i of the form

$$\frac{P \stackrel{a}{\to} Q}{C_i(P) \stackrel{a}{\to} C_i(Q)}$$

with contexts $C_i = f^i(t_1, .., \Box, .., t_{n_i})$, $i = 1, 2, \ldots$, where the f^i's used in the contexts are not distinguished symbols. Then, these structural rules can be directly considered as rules for the tagged syntax. In this setting, if we replace every axiom A_i by the set of corresponding tagged versions $\overline{A}_i$ for every way of tagging the P_i's, we obtain a tagging of $\mathcal{R}$.

Proposition 4.2 *Let us consider $\mathcal{R}$ defined by means of the axioms A_i as above and a set of axioms $\bar{A}_i : \bar{P}_i \stackrel{a_i}{\mapsto} \bar{Q}_i$, with $i = 1, 2, \ldots$, such that $O(\bar{P}_i) = P_i$ and $O(\bar{Q}_i) = Q_i$. Assume that each $\bar{A}_i$ is defined for every possible tagging*[4]

[4] To be exact, we will have an axiom $\bar{A}_{i,\bar{P}_i} : \bar{P}_i \stackrel{a_i}{\to} \bar{Q}_i$ for each $\bar{P}_i$ such that $O(\bar{P}_i) = P_i$, but usually all of them will have a homogenous structure. This is why we proposed above that more concise notation.

of P_i and let $\bar{\mathcal{R}}$ be the reduction relation defined by the set of axioms $\bar{A}_1, \bar{A}_2, \ldots$ and the set of compositional rules $R_1, R_2, \ldots$ Then $\bar{\mathcal{R}}$ is a tagging of $\mathcal{R}$.

Proof Given P_0, we prove condition 1 by rule induction on the definition of $\mapsto$ and condition 2 by rule induction on the definition of $\rightarrow$:

(i) If $\bar{P} \overset{a}{\mapsto} \bar{P}'$ then $O(\bar{P}) \overset{a}{\rightarrow} O(\bar{P}')$
  (a) $\bar{A}_i : \bar{P}_i \overset{a_i}{\mapsto} \bar{Q}_i$. By hypothesis $A_i : O(\bar{P}_i) \overset{a_i}{\mapsto} O(\bar{Q}_i)$.
  (b) For the sake of readability, let us suppose that f^i is a unary constructor, so that we have $R_i : \bar{P} = f^i(\bar{Q}) \overset{a}{\mapsto} f^i(\bar{Q}') = \bar{P}'$ with $\bar{Q} \overset{a}{\mapsto} \bar{Q}'$. By the induction hypothesis we have $O(\bar{Q}) \overset{a}{\rightarrow} O(\bar{Q}')$. Since the symbol f^i is not distinguished, $O(\bar{P}) = O(f^i(\bar{Q})) = f^i(O(\bar{Q}))$, and we can apply rule R_i to obtain $O(\bar{P}) = f^i(O(\bar{Q})) \overset{a}{\rightarrow} f^i(O(\bar{Q}')) = O(f^i(\bar{Q}')) = O(\bar{P}')$.

(ii) If $P \overset{a}{\rightarrow} P'$ and $\bar{P} \in \mathit{Reach}(\mathit{tag}(P_0))$ such that $O(\bar{P}) = P$ then there exists a unique $\bar{P}'$ with $O(\bar{P}') = P'$ and $\bar{P} \overset{a}{\mapsto} \bar{P}'$:
  (a) $A_i : P_i \overset{a_i}{\rightarrow} Q_i$. By hypothesis, for every tagging $\bar{P}_i$ of P_i we have a unique tagged version of the axiom, $\bar{A}_i$.
  (b) Again, let us suppose that $P = f^i(Q) \overset{a}{\rightarrow} P' = f^i(Q')$ with $Q \overset{a}{\rightarrow} Q'$. Given $\bar{P}$ as above, since f^i is not distinguished, it must be the case that $\bar{P} = f^i(\bar{Q})$ with $O(\bar{Q}) = Q$. Now we can apply the induction hypothesis: there exists a single $\bar{Q}'$ such that $O(\bar{Q}') = Q'$ and $\bar{Q} \overset{a}{\mapsto} \bar{Q}'$. Now we can take $\bar{P}' = f^i(\bar{Q}')$, that is the only term such that $O(\bar{P}') = P'$ and $\bar{P} \overset{a}{\mapsto} \bar{P}'$.

□

This is the simplest way of deriving a tagged semantics from a given semantics. The same can be done if the latter is defined by means of a structural congruence, when the distinguished symbols do not appear either in its axioms or in the rules. In this case these axioms and rules can also be applied to the tagged relation, thus defining a structural equivalence between tagged terms. Then, if we add to the set of given previous reduction rules the corresponding congruence rule we also obtain a tagging of the original semantics. This is the case we will study in the following section.

5 Tagged $D\pi$

In this section we will apply the proposed methodology to the language $D\pi$ [7], which is a distributed version of the π-calculus. In fact, for simplicity, we will consider the version presented in [6]. There, a type-based analysis is defined for detecting those systems where there exists a violation of the permissions for using resources (more specifically, channels). In order to prove the correctness

Names		**Values**	
$e ::= k$	locality	$u, v ::=$ bv	base values
$\mid a$	channel	$\mid e$	name
		$\mid x$	variable
Patterns		$\mid u@v$	localized name
$X, Y ::= x$	variable	$\mid \tilde{u}$	tuple
$\mid X@z$	localized pattern		
$\mid \tilde{X}$	tuple		
Processes		**Systems**	
$P, Q ::= \ldots\ldots$		$M, N ::= 0$	empty
$\mid go\ u.P$	movement	$\mid M \mid N$	composition
$\mid (\nu e : E)P$	restriction	$\mid (\nu_k e : E)N$	restriction
		$\mid k[\![P]\!]$	agent

Figure 10. Syntax of $D\pi$

of the analysis, an enhanced semantics is defined over a tagged syntax, where agents are decorated by permissions that are accumulated by them along the execution of the system. Then a subject reduction theorem and a safety theorem are proved, so that well-typed agents do not violate permissions in the enhanced semantics. However there is no formalized connection between the enhanced semantics and the original one and consequently, the property captured by the type system is not clearly related with the original semantics. We think that a way to express the property in the original calculus is needed. In this section we provide such way, as we prove that tagged $D\pi$ is a tagging of $D\pi$ according to our definition and therefore, a quotient. This leads to the conclusion that the type system captures a property of the original system. Additionally, the generality of our approximation would allow us to do the same for any property we would like to capture with a static analysis.

The syntax of $D\pi$ is defined in Figure 10. The dots stand for the usual primitives of the π-calculus [9]: termination, composition, replication, input, output and conditional. A new construct $k[\![P]\!]$, meaning that P is located at k, and a primitive go for movement are introduced. Created names can be channels, but also localities. Such names are local, so localized names such as $u@v$ are needed when using global references.

In Figure 11 you can find the definition of the operational semantics of the language. The structural congruence $\equiv$ (that we do not show here) and most of the rules are similar to those of π-calculus. Only rules (R-GO) and (R-COMM) are specific of $D\pi$. The former moves one process from one location to another, while the latter restricts communication to be local, that is, between processes within the same location.

$$M \mid (\nu_k e : E)N \equiv (\nu_k e : E)(M \mid N) \text{ if } e \notin fn(M)$$

(R-GO)	$\ell[\![go\ k.P]\!] \rightarrow k[\![P]\!]$	
(R-COMM)	$k[\![a!\langle v\rangle P]\!] \mid k[\![a?(X:T).Q]\!] \rightarrow k[\![P]\!] \mid k[\![Q\{X := v\}]\!]$	
(R-EQ$_1$)	$k[\![if\ u = u\ then\ P\ else\ Q]\!] \rightarrow k[\![P]\!]$	
(R-EQ$_2$)	$k[\![if\ u = v\ then\ P\ else\ Q]\!] \rightarrow k[\![Q]\!]$	if $u \neq v$
(S-COPY)	$k[\![*P]\!] \rightarrow k[\![P]\!] \mid k[\![*P]\!]$	
(S-SPLIT)	$k[\![P \mid Q]\!] \rightarrow k[\![P]\!] \mid k[\![Q]\!]$	
(S-NEW)	$k[\![(\nu e : E)P]\!] \rightarrow (\nu_k e : E)k[\![P]\!]$	if $e \neq k$

$$\text{(R-STR)}\quad \frac{N \rightarrow N'}{(\nu e)N \rightarrow (\nu e)N'} \qquad \frac{N \rightarrow N'}{M \mid N \rightarrow M \mid N'} \qquad \frac{N \equiv M \quad M \rightarrow M' \quad M' \equiv N'}{N \rightarrow N'}$$

Figure 11. Semantics of $D\pi$

$$M \mid (\nu_k e : E)N \equiv (\nu_k e : E)(M \mid N) \text{ if } e \notin fn(M)$$

(R-GO)	$\ell[\![go\ k.P]\!]_\Gamma \rightarrow k[\![P]\!]_\Gamma$	
(R-COMM)	$k[\![a!\langle v\rangle P]\!]_\Gamma \mid k[\![a?(X:T).Q]\!]_\Delta \rightarrow k[\![P]\!]_\Gamma \mid k[\![Q\{X := v\}]\!]_{\Delta \sqcap \{_k v:T\}}$	
(R-EQ$_1$)	$k[\![if\ u = u\ then\ P\ else\ Q]\!]_\Gamma \rightarrow k[\![P]\!]_\Gamma$	
(R-EQ$_2$)	$k[\![if\ u = v\ then\ P\ else\ Q]\!]_\Gamma \rightarrow k[\![Q]\!]_\Gamma$	if $u \neq v$
(S-COPY)	$k[\![*P]\!]_\Gamma \rightarrow k[\![P]\!]_\Gamma \mid k[\![*P]\!]_\Gamma$	
(S-SPLIT)	$k[\![P \mid Q]\!]_\Gamma \rightarrow k[\![P]\!]_\Gamma \mid k[\![Q]\!]_\Gamma$	
(S-NEW)	$k[\![(\nu e : E)P]\!]_\Gamma \rightarrow (\nu_k e : E)k[\![P]\!]_{\Gamma,\{_k e:E\}}$	if $e \neq k$

$$\text{(R-STR)}\quad \frac{N \rightarrow N'}{(\nu e)N \rightarrow (\nu e)N'} \qquad \frac{N \rightarrow N'}{M \mid N \rightarrow M \mid N'} \qquad \frac{N \equiv M \quad M \rightarrow M' \quad M' \equiv N'}{N \rightarrow N'}$$

Figure 12. Tagged axioms of $D\pi$

Tagged $D\pi$ is a version of $D\pi$ in which the construct $[\![\ ]\!]$ is substituted by $[\![\ ]\!]_\Gamma$, where Γ ranges over typing environments, that is, partial functions from locality identifiers to $\mathcal{K}$, the set of *locality types*, whose formal definition is here irrelevant [7]. Intuitively, they assign permissions (to use channels) to agents at each locality. For instance, if an agent is tagged with Γ and $\Gamma(k) = K$ then it has permissions at k specified by K.

The semantics of Tagged $D\pi$ is defined by the axioms and rules in Figure 12. There, $\Gamma, \{_k e : E\}$ denotes the extension of Γ at k with the new name e having type E. The most important rule is (R-COMM) which allows the acquisition of new permissions through communication.

Then the set of tags $\mathfrak{L}$ is the set of typing environments and the distinguished symbol is $[\![\ ]\!]$. By mere inspection of the rules defining the tagged

language it can be seen that they satisfy the conditions imposed in the previous section. Indeed, we have a tagged version of each axiom, for every way of tagging the term on its left hand-side. The rest of the rules of the original language can also be considered as rules of the tagged one, since the symbol $[\![\ ]\!]$ does not appear in their definition. Then we have the following

Theorem 5.1 *Tagged $D\pi$ is a* tagging *of $D\pi$.*

According to this result and our characterization of tagged systems it follows that the transition system generated by each term of *Tagged $D\pi$* is isomorphic to a quotient transition system. Therefore, every property defined over *Tagged $D\pi$* is also a property of $D\pi$. In particular, every subset of tagged terms (e.g., that of erroneous terms) defines a subset of computations (that of erroneous computations).

The result is true whatever the initial tagging function is. However, the choice is crucial to adequately formalize the runtime errors we have in mind: in this case the resource access errors. In [7] the chosen function *tag* is only defined for typed processes of the original calculus, so that if M is not typeable then $tag(M) = \emptyset$. As a consequence, it is not possible to describe the resource access errors (violation of permissions) produced by non-typeable systems. In our opinion the definition of resource access error should be independent of typability. Then we should prove that the type system rules out erroneous systems.

Following this approach we tag every original process, simply by adding to the tags the created names, which are exactly those the agents have initially permission to use. We will do so using the mappings $tag_\Gamma(M)$, where Γ is the initial knowledge of M. Let us consider the system defined by $(\nu a : A)(k[\![b!\langle a\rangle]\!] \mid k[\![b?(x : A).x!\langle c\rangle]\!])$. Though the channel a is created with the two agents under its scope, only the agent on the left knows of its existence. In fact, the scope of the names can always be extended via extrusion. In particular, that system and $(\nu a : A)k[\![b!\langle a\rangle]\!] \mid k[\![b?(x : A).x!\langle c\rangle]\!]$ are equivalent. In order to deal with this fact, we will restrict ambient domains in $tag_\Gamma(M)$ to the set of free names in M, that is, we will have the invariant $dom(\Gamma) = fn(M)$, and we will denote by Γ_M the restriction $\Gamma\,|_{fn(M)}$. Now we can define the initial tagging function.

Definition 5.2 Let Φ be the empty (partial) function in $\mathcal{L}$ (that with empty domain). Then we define the function $tag : \mathcal{M} \to \bar{\mathcal{M}}$ by:

- $tag(M) = tag_\Phi(M)$ if M is closed
- $tag_\Gamma(0) = 0$
- $tag_\Gamma(M \mid N) = tag_{\Gamma_M}(M) \mid tag_{\Gamma_N}(N)$

- $tag_\Gamma((\nu_\ell e : E)M) = (\nu_\ell e : E)tag_{\Gamma,\{_\ell e:E\}}(M)$ if $e \in fn(M)$
- $tag_\Gamma((\nu_\ell e : E)M) = (\nu_\ell e : E)tag_\Gamma(M)$ if $e \notin fn(M)$
- $tag_\Gamma(\ell[\![P]\!]) = \ell[\![P]\!]_\Gamma$

In [7] an error relation on the corresponding tagged systems $M \to err$ is defined, meaning that a process can violate its permissions. To remove these errors, a type system $\vdash$ for the original language and another one $\Vdash$ for the tagged systems were defined. Now we need the following lemma.

Lemma 5.3 *If* $\Gamma \vdash M$ *then* $\Gamma \Vdash tag_\Gamma(M)$.

Proof The proof is straightforward by induction on the rules used to derive $\Gamma \vdash M$ (see [7]):

(i) $\Gamma \vdash 0$. Since $tag_\Gamma(0) = 0$ and $\Gamma \Vdash 0$ the thesis follows.

(ii) $\Gamma \vdash k[\![P]\!]$. Then it is the case that $\Gamma \vdash_k P$. Since $\Gamma <: \Gamma$ it follows that $\Gamma \Vdash k[\![P]\!]$.

(iii) $\Gamma \vdash M_1 \mid M_2$ with $\Gamma \vdash M_i$. By strengthening it follows that $\Gamma_{M_i} \vdash M_i$ and by the induction hypothesis, $\Gamma_{M_i} \Vdash tag_{\Gamma_{M_i}}(M_i)$ and, by weakening, $\Gamma \Vdash tag_{\Gamma_{M_i}}(M_i)$. Since $tag_\Gamma(M_1 \mid M_2) = tag_{\Gamma_{M_1}}(M_1) \mid tag_{\Gamma_{M_2}}(M_2)$ it follows that $\Gamma \Vdash tag_\Gamma(M_1 \mid M_2)$.

(iv) $\Gamma \vdash (\nu_k e : E)M$ with $\Gamma, \{_k e : E\} \vdash M$. By induction hypothesis it follows that $\Gamma, \{_k e : E\} \Vdash tag_{\Gamma,\{_k e:E\}}(M)$
   (a) If $e \in fn(M)$ then $tag((\nu_k e : E)M) = (\nu_k e : E)tag_{\Gamma,\{_k e:E\}}(M)$ and we can conclude $\Gamma \Vdash tag_\Gamma((\nu_k e : E)M)$
   (b) If $e \notin fn(M)$, by strengthening it also holds that $\Gamma \Vdash tag_{\Gamma,\{_k e:E\}}(M)$ and since then $tag((\nu_k e : E)M) = (\nu_k e : E)tag_{\Gamma,\{_k e:E\}}(M)$, the thesis follows.

□

A subject reduction theorem and a safety theorem for the tagged system were proved in [7]. Then we have our safety theorem.

Theorem 5.4 *If* $\Gamma \vdash M$ *and* $tag_\Gamma(M) \mapsto^* M'$ *then* $M' \not\to err$.

And the following

Corollary 5.5 *If* $\Phi \vdash M$ *and* $tag(M) \mapsto^* M'$ *then* $M' \not\to err$.

This result also appears in [7], but no interpretation of it is given in terms of the original semantics of the language. Instead, we can now interpret the fact that $M \not\to err$ as the impossibility to use a resource when we have not received the permission to do it along the current computation, since the tags remember exactly the set of accumulated permissions.

Example 5.6 As a very simple example, let us consider

$$M = (\nu\ell : loc\{a : res\langle T\rangle\})(\nu_k b : res\langle T\rangle)(\nu_\ell c : T)\ell[\![b!\langle c\rangle]\!]$$

We can tag M, getting

$$tag(M) = (\nu\ell : loc\{a : res\langle T\rangle\})(\nu_k b : res\langle T\rangle)(\nu_\ell c : T)\ell[\![b!\langle c\rangle]\!]_\Gamma$$

with

$$\Gamma = \{\ell : loc\{a : res\langle T\rangle, c : T\}, k : loc\{b : res\langle T\rangle\}\}$$

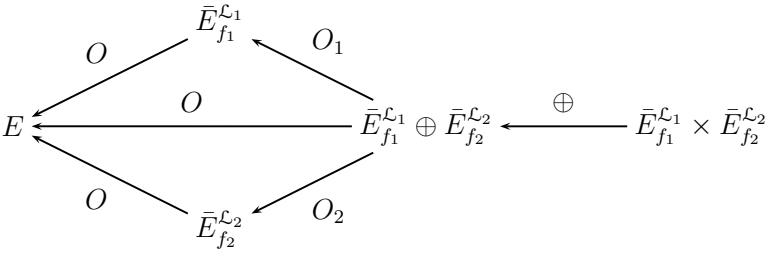
However, M is not typeable and, in fact, $tag(M) \to err$, since it attempts to use channel b at locality l, which is not allowed by Γ. Of course, since types cannot be complete, it could be the case that M is not typeable but $tag(M)$ will not produce any error.

6 Applications of the algebraic structure of quotient systems lattice

In this section we are giving a brief overview of how we can profit the algebraic structure of the quotient systems lattice, in particular for the combination of different taggings of the same system. We start by defining the merging of two different taggings of a given syntax.

Definition 6.1 Given a many-sorted signature $(\mathcal{S}, \Sigma)$, let E be the $\mathcal{S}$-sorted term algebra of Σ, f_1 and f_2 two symbols of Σ and $\mathcal{L}_1$ and $\mathcal{L}_2$ two (non-empty) sets of labels. We define the merging of $\bar{E}_{f_1}^{\mathcal{L}_1}$ and $\bar{E}_{f_2}^{\mathcal{L}_2}$,and we denote it $\bar{E}_{f_1}^{\mathcal{L}_1} \oplus \bar{E}_{f_2}^{\mathcal{L}_2}$, as $\bar{E}_{f_1,f_2}^{\mathcal{L}_1,\mathcal{L}_2}$ if $f_1 \neq f_2$ or as $\bar{E}_{f_1}^{\mathcal{L}_1\times\mathcal{L}_2}$ when $f_1 = f_2$. This definition can be easily generalized to consider different distinguished symbols and sets of tags.

If we take O_1 and O_2 as the functions deleting labels from $\mathcal{L}_2$ and $\mathcal{L}_1$, respectively, given $\overline{e}^1 \in \bar{E}_{f_1}^{\mathcal{L}_1}$ and $\overline{e}^2 \in \bar{E}_{f_2}^{\mathcal{L}_2}$ such that $O(\overline{e}^1) = O(\overline{e}^2)$ there exists only one $\overline{e} \in \bar{E}_{f_1}^{\mathcal{L}_1} \oplus \bar{E}_{f_2}^{\mathcal{L}_2}$, that we denote $\overline{e}^1 \oplus \overline{e}^2$,such that $O_1(\overline{e}) = \overline{e}^1$ and $O_2(\overline{e}) = \overline{e}^2$. Then we can define a partial operator $\oplus$ from the set $\bar{E}_{f_1}^{\mathcal{L}_1} \times \bar{E}_{f_2}^{\mathcal{L}_2}$ to $\bar{E}_{f_1}^{\mathcal{L}_1} \oplus \bar{E}_{f_2}^{\mathcal{L}_2}$, as depicted below. It simply takes a term tagged in two ways and merges those tags.



Now let us make use of these notations to combine different taggings.

Definition 6.2 Let $S = (E, V, \rightarrow, e_0)$ be a labelled transition system of E and $T_1 = (\bar{E}_{f_1}^{\mathcal{L}_1}, V, \mapsto_1, \overline{e_0}^1)$ and $T_2 = (\bar{E}_{f_2}^{\mathcal{L}_2}, V, \mapsto_2, \overline{e_0}^2)$ two different taggings of S. We define the merging of T_1 and T_2 as $T_1 \oplus T_2 = (\bar{E}_{f_1}^{\mathcal{L}_1} \oplus \bar{E}_{f_2}^{\mathcal{L}_2}, V, \mapsto, \overline{e_0}^1 \oplus \overline{e_0}^2)$, where $\mapsto$ is defined as follows:

$$\frac{O_i(\overline{e_1}) \overset{a}{\mapsto}_i O_i(\overline{e_2}) \quad i = 1, 2}{\overline{e_1} \overset{a}{\mapsto} \overline{e_2}}$$

Proposition 6.3 *If T_1 and T_2 are taggings of the transition system S then $T_1 \oplus T_2$ is a tagging of S.*

Proof We must check that the two conditions for a transition system over a tagged syntax to be a tagging:

- If $\overline{e_1} \overset{a}{\mapsto} \overline{e_2}$ then $O(\overline{e_1}) \overset{a}{\rightarrow} O(\overline{e_2})$: by definition of $\mapsto$, $O_1(\overline{e_1}) \overset{a}{\mapsto}_1 O_1(\overline{e_2})$. T_1 is a tagging of S and, therefore, $O(O_1(\overline{e_1})) \overset{a}{\rightarrow} O(O_1(\overline{e_2}))$, that is (see the diagram above), $O(\overline{e_1}) \overset{a}{\rightarrow} O(\overline{e_2})$.
- Let $e_1 \overset{a}{\rightarrow} e_2$ and $\overline{e_1} \in Reach(T_1 \oplus T_2)$ such that $O(\overline{e_1}) = e_1$. Then $\overline{e_1} = \overline{e_1}^1 \oplus \overline{e_1}^2$ with $\overline{e_1}^i \in Reach(T_i)$ (i=1,2). Since T_1 and T_2 are taggings of S and $O(\overline{e_1}^i) = e_1$ there exist a single $\overline{e_2}^1$ and a single $\overline{e_2}^2$ such that $\overline{e_1}^i \overset{a}{\mapsto}_i \overline{e_2}^i$. Then we can take $\overline{e_2} = \overline{e_2}^1 \oplus \overline{e_2}^2$, which is the only term in $\bar{E}_{f_1}^{\mathcal{L}_1} \oplus \bar{E}_{f_2}^{\mathcal{L}_2}$ such that $O(\overline{e_2}) = e_2$ and $\overline{e_1} \overset{a}{\mapsto} \overline{e_2}$.

□

Moreover, $T_1 \oplus T_2$ captures exactly those properties captured by T_1 and T_2, as formalized by the following theorem.

Theorem 6.4 *Let $S = (E, V, \rightarrow, e_0)$ be a labelled transition system of E and $T_1 = (\bar{E}_{f_1}^{\mathcal{L}_1}, V, \mapsto_1, \overline{e_0}^1)$ and $T_2 = (\bar{E}_{f_2}^{\mathcal{L}_2}, V, \mapsto_2, \overline{e_0}^2)$ two different taggings of S. Suppose that $S^{\sim_1}$ and $S^{\sim_2}$ are the quotient systems st-isomorphic to T_1 and T_2, respectively. Then $T_1 \oplus T_2$ is st-isomorphic to $S^{\sim_1} \sqcup S^{\sim_2}$.*

Proof Let us suppose that h_i are *st*-isomorphisms between $S^{\sim_i}$ and T_i, for $i = 1, 2$, and take $h : \mathcal{C}/_{\sim} \rightarrow \bar{E}_{f_1}^{\mathcal{L}_1} \oplus \bar{E}_{f_2}^{\mathcal{L}_2}$, defined as $h([L]_{\sim}) = h_1([L]_{\sim_1}) \oplus h_2([L]_{\sim_2})$, where $\sim = \sim_1 \cap \sim_2$.

- h is well defined: if $L \sim L'$ then $L \sim_i L'$ for $i = 1, 2$ by definition of $\sim$. Since h_i is well defined, $h_i([L]_{\sim_i}) = h_i([L']_{\sim_i})$ for any $i = 1, 2$ and therefore, $h([L]_{\sim}) = h([L']_{\sim})$. Moreover, $\oplus$ is defined for $h_1([L]_{\sim_1})$ and $h_2([L]_{\sim_2})$ because $O(h_i([L]_{\sim_i})) = st(h_i([L]_{\sim_i})) = st([L]_{\sim_i}) = last(L)$ for $i = 1, 2$ and therefore, $O(h_1([L]_{\sim_1})) = O(h_2([L]_{\sim_2}))$
- h is injective: $h([L]_{\sim}) = h([L']_{\sim}) \Rightarrow h_1([L]_{\sim_1}) \oplus h_2([L]_{\sim_2}) = h_1([L']_{\sim_1}) \oplus h_2([L']_{\sim_2})$. This implies (by definition of $\oplus$) that $h_i([L]_{\sim_i}) = h_i([L']_{\sim_i})$ for

$i = 1, 2$. Since h_1 and h_2 are injective, $[L]_{\sim_i} = [L']_{\sim_i}$ for $i = 1, 2$ and therefore, $[L]_\sim = [L']_\sim$.

- h is surjective: let $\overline{e} \in Reach(T_1 \oplus T_2)$ and take $O_i(\overline{e}) = \overline{e}^i \in Reach(T_i)$. Since h_i is surjective there exist $[L_i]_{\sim_i}$ such that $h_i([L_i]_{\sim_i}) = \overline{e}^i$. Again, since $\overline{e} \in Reach(T_1 \oplus T_2)$ there exists $L \in \mathcal{C}$ such that $L_1 \sim_1 L \sim_2 L_2$. Then, $h([L]_\sim) = h_1([L]_{\sim_1}) \oplus h_2([L]_{\sim_2}) = h_1([L_1]_{\sim_1}) \oplus h_2([L_2]_{\sim_2}) = \overline{e}^1 \oplus \overline{e}^2 = \overline{e}$.
- h is an isomorphism of transition systems:

(i) $h([e_0]_\sim) = h_1([e_0]_{\sim_1}) \oplus h_2([e_0]_{\sim_2}) = \overline{e_0}^1 \oplus \overline{e_0}^2$, since h_1 and h_2 are isomorphisms.

(ii) $[L]_\sim \overset{a}{\mapsto}_\sim [L']_\sim \Leftrightarrow [L]_{\sim_i} \overset{a}{\mapsto}_{\sim_i} [L']_{\sim_i} \Leftrightarrow h_i([L]_{\sim_i}) \overset{a}{\mapsto}_i h_i([L']_{\sim_i}) \Leftrightarrow h([L]_\sim) \overset{a}{\mapsto} h([L']_\sim)$

- h preserves states: $st(h([L]_\sim)) = st(h_1([L]_{\sim_1}) \oplus h_2([L]_{\sim_2})) = O(h_1([L]_{\sim_1})) = st(h_1([L]_{\sim_1})) = st([L]_{\sim_1}) = last(L) = st([L]_\sim)$.

□

Example 6.5 Let us recall our Example 3.9 in Section 3. Since every symbol in the syntax is tagged, every tag in the merged system is a pair (Γ, Δ) with $\Gamma, \Delta \in \{\square, \circ\}$. To simplify the notation we will take[5] $\circ = (\circ, \circ)$, $\square = (\square, \square)$ and $\diamond = (\circ, \square)$ (we will not need the tag $(\square, \circ)$ since terms tagged with it will not be reachable). The merging of the two properties, shown in Figure 13, tell us whether any agent has died, and in such a case, whether this happened an even number of times. Theorem 6.4 tells us that this new transition system captures the conjunction of both properties, that is, terms tagged with $\circ$ identify those computations in which no agent has died, $\square$ identifies those computations in which an odd number of agents have died and $\diamond$ identifies those computations in which an even, but not null, number of agents have died. However, when we consider the meet of both transition systems (the disjunction of properties) we go all the way down to the bottom of the lattice.

7 Conclusion and Future Work

We think that the present work is a rather simple, but conceptually important justification of the use of tagged languages for the analysis of systems. The fact that these systems are quotients of the corresponding reachability trees tells us that the properties defined over them are, indeed, also properties of the original semantics, thus keeping us within the same original framework for every possible property of our interest. Besides, the work has been developed

[5] Obviously, we are here overloading the notation in order to preserve the symbols used in the figures.

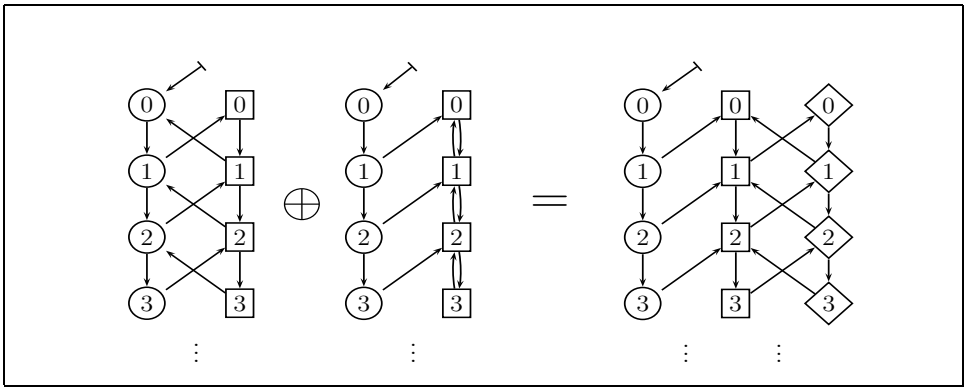


Figure 13. Merging of tagged systems

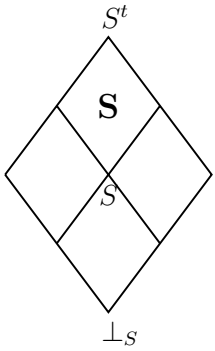


Figure 14. Extended quotient systems lattice

in a very general setting that can be systematically applied to any particular language and property of interest.

Here we have just applied the algebraic structure to a very simple situation, that where we want to study several properties at the same time. It may be worth a further study of the algebraic structure of the set of tagged languages, trying to exploit it to study the combination of several properties in a more general setting that includes, for example, the disjunction of properties, that seems to be trickier.

So far, we have considered equivalences remembering at least the reached state, thus getting always some tagged systems. However, this constraint could be relaxed to obtain a superset of our class of quotient systems (see Figure 14). Then the bottom would represent the tagged system forgetting absolutely everything, even the reached state. Additionally, we would have transition systems lying in the diamond of the bottom of Figure 14, remembering only certain properties about the reached state, and transition systems combining both approaches, the ones in the diamonds at both sides of the figure, that

remember some things about the past of the process but still only part of the information about the reached state. This could be of interest for instance when the syntax is defined as the term algebra of a given signature, modulo an equational theory. In that case we do not only want to identify computations reaching identical terms, but also those reaching terms that are equivalent in that theory. Of course, extra soundness conditions between that theory and the labelled transition system should hold.

We plan to study all these equivalences in a systematic way, thus trying to get their properties in a uniform way, to avoid the development of ad hoc proofs for each particular case in which we could be interested.

References

[1] M. Boreale and M. Buscemi. A framework for the analysis of security protocols. In *Concurrency Theory, 13th International Conference, CONCUR'02*, volume 2421 of *LNCS*, pages 483–498. Springer, 2002.

[2] Hartmut Ehrig and B. Mahr. *Fundamentals of Algebraic Specification I.* Springer-Verlag New York, Inc., 1985.

[3] R. Foccardi and R. Gorrieri. Classification of security properties (Part I: Information flow). In *Foundations of Security Analysis and Design, FOSAD'00*, volume 2171 of *LNCS*, pages 331–396. Springer, 2001.

[4] R. Foccardi, R. Gorrieri, and F. Martinelli. Classification of security properties (Part II: Network security). In *Foundations of Security Analysis and Design II, FOSAD'02*, volume 2946 of *LNCS*, pages 139–185. Springer, 2004.

[5] M. Hennessy. *The Semantics of Programming Languages: An Elementary Introduction Using Structural Operational Semantics.* John Wiley and Sons, 1990.

[6] M. Hennessy, M. Merro, and J. Rathke. Towards a behavioural theory of access and mobility control in distributed systems. In *Foundations of Software Science and Computational Structures, 6th International Conference, FOSSACS'03*, volume 2620 of *LNCS*, pages 282–298. Springer, 2003.

[7] M. Hennessy and J. Riely. Resource access control in systems of mobile agents. In *High-Level Concurrent Languages, HLCL'98*, volume 16 of *ENTCS*, pages 3–17. Elsevier, 1998.

[8] M. Hennessy and J. Riely. Information flow vs. resource access in the asynchronous pi-calculus. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 24(5):566–591, 2002.

[9] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992.

[10] B. Pierce and D. Sangiorgi. Typing and Subtyping for Mobile Processes. In *Proceedings 8th IEEE Logics in Computer Science, LICS'93*, pages 376–385. IEEE Computer Society Press, 1993.