IMPLEMENTING TILE SYSTEMS: SOME EXAMPLES FROM PROCESS CALCULI

R. BRUNI
Dipartimento di Informatica, Università di Pisa, Corso Italia, 40
56125 PISA, ITALIA
E-mail: bruni@di.unipi.it

J. MESEGUER
Computer Science Laboratory, SRI International, 333 Ravenswood Av.
Menlo Park, CA 94025-3493, U.S.A.
E-mail: meseguer@csl.sri.com

U. MONTANARI
Dipartimento di Informatica, Università di Pisa, Corso Italia, 40
56125 PISA, ITALIA
E-mail: ugo@di.unipi.it

This work reports on some useful applications of the tile model to the specification and execution of CCS-like *process calculi*. This activity is part of our ongoing research on the relation between *tile logic* and *rewriting logic*.

1 Overview

Tile Logic[1,2] is a framework for modular descriptions of the dynamic evolution of concurrent systems, extending *rewriting logic*[3,4] (in the non-conditional case), by side-effects and rewriting synchronization. These aspects are very important when modelling *process algebras*, because they allow us to represent the dynamic interaction between processes and "the rest of the world". Since rewriting logic is the semantic basis of several language implementation efforts, it is interesting to map tile logic back into rewriting logic in a conservative way, to obtain executable specifications of tile systems. However, as explained in[5], a correct rewriting implementation of tile logic must deal with two issues, namely the possible identification of different tile proofs, and the development of a meta-layer to control the rewritings. By exploiting the *reflective* capabilities[6,7] of the language *Maude*[8], the meta-layer can be specified as a kernel of *internal strategies*[9], which may help the user to direct the computation and to collect the possible results (Maude is a language that has been recently developed at SRI International, and it is based on rewriting logic).

In tile logic, a set of rules combines certain basic modules, which may interact through their interfaces. Roughly speaking, modules are just *open* (e.g., partially specified) configurations of the system. The behaviour of a system as a whole consists of a coordinated evolution of its sub-modules. The name "tile" is due to the graphic representation of such rules:

$$\begin{tikzcd} \circ \arrow[r, "s"] \arrow[d, "a"'] & \circ \arrow[d, "b"] \\ \circ \arrow[r, "s'"'] & \circ \end{tikzcd}$$

also written $s \xrightarrow[b]{a} s'$, states that the *initial configuration* s of the system evolves to the *final configuration* s' producing an *effect* b, which can be observed by the rest of the system. However, such a step is allowed only if the subcomponents of s (which is in general an *open* configuration) evolve to the subcomponents of s', producing the *trigger* a. The vertices $\circ$ of the tile are called *interfaces*. Tiles can be composed horizontally (through side effects), vertically (computational evolutions of a certain component), and in parallel (concurrent steps) to generate larger steps. By analogy with rewriting logic, tiles can be considered as special sequents subject to certain inference rules.

In the translation from tile logic to rewriting logic, the basic idea is to "stretch" tiles into ordinary rewriting cells as pictured below, maintaining the capability to distinguish between configurations and effects.

$$\begin{tikzcd} \circ \arrow[r, "s"] \arrow[d, "a"'] & \circ \arrow[d, "b"] \\ \circ \arrow[r, "s'"'] & \circ \end{tikzcd} \quad \mapsto \quad \begin{tikzcd} & \circ \arrow[dr, bend left, "b"] & \\ \circ \arrow[ur, bend left, "s"] \arrow[dr, bend right, "a"'] & \Downarrow & \circ \\ & \circ \arrow[ur, bend right, "s'"'] & \end{tikzcd}$$

It may happen that both configurations and effects rely on a common *auxiliary* structure, e.g. for tupling, projecting, and permuting interfaces. In this case the auxiliary structure becomes "shared" in the stretched translation. In [5] two such situations are studied in detail, namely *process tile logic* and *term tile logic*, where both configurations and effects are respectively net-process-like and term-like structures. As a main result, given a tile system $\mathcal{R}$, a sequent $s \xrightarrow[b]{a} s'$ is entailed by $\mathcal{R}$ in tile logic if and only if a sequent $s; b \Rightarrow a; s'$ is entailed by the stretched version of $\mathcal{R}$ in rewriting logic and its proof satisfies some additional constraints (see [10,5]). Moreover, for a class of tile systems (called *uniform*) the additional constraints are automatically satisfied by the fact that the initial state of the sequent $s; b \Rightarrow a; s'$ is the sequential composition of a configuration followed by an effect, and that the final state is the sequential composition of an effect followed by a configuration. If the tile system is not uniform, then also the actual proof term decorating the derivation has to be taken into account. However, since at present we do not have any meaningful example of non uniform systems we are not really interested in having such an implementation. In any case we need to define some rewriting strategies, exploring the tree of nondeterministic rewritings, until a *successful* state is reached. For instance, a general notion of *success* for uniform tile systems consists of `VH` states, i.e., states that are the sequential composition of an effect followed by a configuration.

A key point is that the internal strategies defined for simulating tile systems, can also be thought of as general meta-strategies for nondeterministic rewriting systems equipped with a predicate of *success*. We refer the interested reader to [5] for a detailed description of some internal strategies. In this paper we show some example of their application to implement concurrent process calculi. In particular, in Section 2 we define executable implementations of CCS-like languages, preserving their original operational semantics. The two case studies considered here are the tile specification of *finite CCS* given in [2] (here discussed in Section 2.1) and and the tile specification of *CCS with locations* [11] given in [12] (here discussed in Section 2.2), exemplifying respectively term tile logic and process tile logic.

2 Rewriting Processes

Milner's *Calculus for Communicating Systems* (CCS) [13] is among the best well-known and studied concurrency models. We present here executable implementations of CCS-like languages defined through the translation in Maude of suitable tile systems.

The advantage of modelling process algebras with tile logic should be evident just considering the usual *action prefix* operation, denoted by $\mu.\_$. When applied to a certain process P it returns a process $\mu.P$, which can perform an action μ and then behaves like P. The corresponding tile is represented below and can be composed horizontally with the identity cell of any process P to model the computation step associated to the action prefix.

$$
\begin{tikzcd}
\circ \arrow[r, "\mu.\_"] \arrow[d, "id"'] & \circ \arrow[d, "\mu"] \\
\circ \arrow[r, "id"'] & \circ
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
\circ \arrow[r, "P"] \arrow[d, "id"'] & \circ \arrow[r, "\mu.\_"] \arrow[d, "id"'] & \circ \arrow[d, "\mu"] \\
\circ \arrow[r, "P"'] & \circ \arrow[r, "id"'] & \circ
\end{tikzcd}
$$

Let nil be the inactive process, and consider the process $p = \mu_1.\mu_2.nil$. If process p tries to execute action μ_2 before executing μ_1 it gets stuck because there is no tile having μ_2 as trigger and $\mu_1.\_$ as initial configuration. In a non-conditional rewriting system, this is not necessarily true, because rewriting steps can be freely contextualized (and instantiated). This problem is well-known in rewriting logic, and some ad-hoc solutions have been already proposed in the literature [14,15]. Our methodology could offer a unifying view for a wide class of related problems.

It follows that a typical query in a tile system could be: "derive all (some of) the tiles with a given horizontal source s and vertical target b". A surprising thing, in the translation of a tile system, is that queries start with a vertical target rather than a source. The obvious explanation consists in the use of terms to define the vertical arrows. In this case this is the only correct procedure. However, for CCS-like process calculi [2], we realized that the vertical and horizontal dimensions could be swapped in such a way that the intuitive queries are of the kind "derive all one-step transitions for a given agent P". This is possible because the vertical signature consists of *unary* actions. So we can (1) reverse the vertical arrows in the tile system and then (2) rotate clockwise of 90 degrees the tiles when implementing the system, as illustrated below for the action prefix tile:

$$\begin{tikzcd} 1 \arrow[r, "\mu.\_"] \arrow[d, "id"'] & 1 \arrow[d, "\mu"] \\ 1 \arrow[r, "id"'] & 1 \end{tikzcd} \Longrightarrow^{(1)} \begin{tikzcd} 1 \arrow[r, "\mu.\_"] & 1 \\ 1 \arrow[r, "id"'] \arrow[u, "id"] & 1 \arrow[u, "\mu"'] \end{tikzcd} \Longrightarrow^{(2)} \begin{tikzcd} 1 \arrow[r, "id"] \arrow[d, "id"'] & 1 \arrow[d, "\mu.\_"] \\ 1 \arrow[r, "\mu"'] & 1 \end{tikzcd}$$

$$(a) \Downarrow \qquad\qquad\qquad\qquad \Downarrow(b)$$

$$\begin{tikzcd} 1 \arrow[r, bend left, "\mu(\mu.P)"] \arrow[r, bend right, "P"'] \arrow[r, phantom, "\Downarrow"] & 1 \end{tikzcd} \qquad\qquad \begin{tikzcd} 1 \arrow[r, bend left, "\mu.P"] \arrow[r, bend right, "\mu(P)"'] \arrow[r, phantom, "\Downarrow"] & 1 \end{tikzcd}$$

Examinining the 2-cell translations the motivation for this way of swapping the arrows results very clear. The cell on the left (a) states that if we try to force the process $\mu.P$ to perform a μ action, it succeeds. The other cell (b) states that process $\mu.P$ may perform the action μ. Consequently, an implementation using the first kind of rules can only be used to *test* CCS processes, whereas an implementation based on the second kind of rules generates all and only the possible evolutions of the system. In any case we need to define some rewriting strategies, exploring the tree of nondeterministic rewritings, until a successful configuration is reached. For instance, a general notion of success for uniform tile systems consists in reaching a state which is the sequential composition of an effect follwed by a configuration.

2.1 Finite CCS

Let Δ (ranged over by α) be the set of *basic actions*, and let $\bar{\Delta}$ be the set of *complementary actions* (where $\bar{\_}$) is an involutive function such that $\Delta = \bar{\bar{\Delta}}$

and $\Delta \cap \bar{\Delta} = \emptyset$). We denote by Λ (ranged over by λ) the set $\Delta \cup \bar{\Delta}$. Let $\tau \notin \Lambda$ be a *distinguished* action, and let $Act = \Lambda \cup \{\tau\}$ (ranged over by μ) be the set of CCS actions. Then, a *finite CCS process* is a term generated by the following grammar:

$$P ::= nil \mid \mu.P \mid P\backslash\alpha \mid P[\alpha/\beta] \mid P+P \mid P|P.$$

We let P, Q, R, ... range over the set $Proc$ of CCS processes. Assuming the reader familiar with the notation, we give just an informal description of CCS algebra operators: the constant nil yields the *inactive* process; $\mu.P$ is a process behaving like P but only after the execution of action μ; $P\backslash\alpha$ is the process P with actions α and $\bar{\alpha}$ blocked by *restriction* $\_\backslash\alpha$; $P[\alpha/\beta]$ behaves like P with actions α and $\bar{\alpha}$ relabelled with β and $\bar{\beta}$; $P+Q$ is the *nondeterministic (guarded) sum* of processes P and Q; finally $P|Q$ is the parallel composition of processes P and Q. The dynamic behaviour of CCS processes is usually described by a transition system, presented in the SOS style.

Definition 1 (Operational Semantics of CCS) *The* CCS transition system *is given by the relation $T \subseteq Proc \times Act \times Proc$ inductively generated from the following set of axioms and inference rules*

$$\frac{}{\mu.P \xrightarrow{\mu} P} \qquad \frac{P \xrightarrow{\mu} Q}{P\backslash\alpha \xrightarrow{\mu} Q\backslash\alpha}\ \mu \notin \{\alpha, \bar{\alpha}\} \qquad \frac{P \xrightarrow{\mu} Q}{P+R \xrightarrow{\mu} Q} \qquad \frac{P \xrightarrow{\mu} Q}{R+P \xrightarrow{\mu} Q}$$

$$\frac{P \xrightarrow{\mu} Q}{P[\alpha/\beta] \xrightarrow{\mu[\alpha/\beta]} Q[\alpha/\beta]} \qquad \frac{P \xrightarrow{\mu} Q}{P|R \xrightarrow{\mu} Q|R} \qquad \frac{P \xrightarrow{\alpha} Q,\ P' \xrightarrow{\bar{\alpha}} Q'}{P|P' \xrightarrow{\tau} Q|Q'} \qquad \frac{P \xrightarrow{\mu} Q}{R|P \xrightarrow{\mu} R|Q}$$

where $P \xrightarrow{\mu} Q$ means that $(P, \mu, Q) \in T$, and the action $\mu[\alpha/\beta]$ is defined as β if $\mu = \alpha$, as $\bar{\beta}$ if $\mu = \bar{\alpha}$, and as μ otherwise.

The operational meaning is that a process P may perform an action μ becoming Q iff it is possible to inductively construct a sequence of rule applications to conclude that $P \xrightarrow{\mu} Q$. More generally, a process P_0 may evolve to process P_n iff there exists a *computation* $P_0 \xrightarrow{\mu_1} P_1 \ldots P_{n-1} \xrightarrow{\mu_n} P_n$. In [2] it is shown how to associate a tile system to CCS. We adapt their definition to define the term tile logic for finite CCS.

Remark 1 *If both configurations and effects are terms over two signatures Σ_H and Σ_V, then we can assume a standard representation of basic tiles having the form*

$$\begin{tikzcd} \underline{n} \arrow[r, "\vec{h}"] \arrow[d, "\vec{v}"'] & \underline{m} \arrow[d, "u"] \\ \underline{k} \arrow[r, "g"'] & \underline{1} \end{tikzcd}$$

(with $\vec{h} \in T_{\Sigma_H}(X_n)^m$, $g \in T_{\Sigma_H}(X_k)$, $\vec{v} \in T_{\Sigma_V}(X_n)^k$, and $u \in T_{\Sigma_V}(X_m)$, where $X_i = \{x_1, ..., x_i\}$ is a chosen set of variables, totally ordered by $x_{j_1} < x_{j_2}$ iff $j_1 < j_2$) as a sequent $n \triangleleft \langle \vec{h} \rangle \xrightarrow[\langle u \rangle]{\langle \vec{v} \rangle} \langle g \rangle$ where the number of variables in the "upper-left" corner of the tile is made explicit (the values m and k can be easily retrieved from the lengths of the term vectors decorating the tile).

Definition 2 (Tiles for finite CCS) *The term tile system $\mathcal{R}_{CCS}$ has the signature $\Sigma_A = \{\mu : 1 \longrightarrow 1 \mid \mu \in Act\}$ as horizontal signature, the signature Σ_P of CCS processes as vertical signature, and the following basic tiles:*

$$act_\mu : 1 \triangleleft \langle x_1 \rangle \xrightarrow[\langle x_1 \rangle]{\langle \mu . x_1 \rangle} \langle \mu(x_1) \rangle \qquad res_{\mu,\alpha} : 1 \triangleleft \langle \mu(x_1) \rangle \xrightarrow[\langle x_1 \backslash \alpha \rangle]{\langle x_1 \backslash \alpha \rangle} \langle \mu(x_1) \rangle \quad (\mu \notin \{\alpha, \bar{\alpha}\})$$

$$rel_{\mu,\alpha,\beta} : 1 \triangleleft \langle \mu(x_1) \rangle \xrightarrow[\langle x_1[\alpha/\beta] \rangle]{\langle x_1[\alpha/\beta] \rangle} \langle t \rangle \quad with\ t = \begin{cases} \beta(x_1) \ if\ \mu = \alpha \\ \bar{\beta}(x_1) \ if\ \mu = \bar{\alpha} \\ \mu(x_1) \ otherwise \end{cases}$$

$$\langle +_\mu : 2 \triangleleft \langle \mu(x_1), x_2 \rangle \xrightarrow[\langle x_1 + x_2 \rangle]{\langle x_1 \rangle} \langle \mu(x_1) \rangle \qquad +\rangle_\mu : 2 \triangleleft \langle x_1, \mu(x_2) \rangle \xrightarrow[\langle x_1 + x_2 \rangle]{\langle x_2 \rangle} \langle \mu(x_1) \rangle$$

$$\rfloor_\mu : 2 \triangleleft \langle \mu(x_1), x_2 \rangle \xrightarrow[\langle x_1 | x_2 \rangle]{\langle x_1 | x_2 \rangle} \langle \mu(x_1) \rangle \qquad \lfloor_\mu : 2 \triangleleft \langle x_1, \mu(x_2) \rangle \xrightarrow[\langle x_1 | x_2 \rangle]{\langle x_1 | x_2 \rangle} \langle \mu(x_1) \rangle$$

$$\|_\lambda : 2 \triangleleft \langle \lambda(x_1), \bar{\lambda}(x_2) \rangle \xrightarrow[\langle x_1 | x_2 \rangle]{\langle x_1 | x_2 \rangle} \langle \tau(x_1) \rangle$$

As explained at the beginning of Section 2, here the vertical dimension is associated to process descriptions, whereas the horizontal dimension represents the *opposite* of the dynamic evolution of the system (we say *opposite*, because the arrows representing the actions performed by the system are reversed from their computational-driven intuitive direction).

The following result establishes the correspondence between the set-theoretic view of the operational semantics T, and the sequents entailed by $\mathcal{R}_{CCS}$.

Proposition 1 *The tile system $\mathcal{R}_{CCS}$ is uniform, and for any CCS agents P and Q, and action μ:*

$$P \xrightarrow{\mu} Q \in T \iff \mathcal{R}_{CCS} \vdash 0 \triangleleft \langle \rangle \xrightarrow[\langle Q \rangle]{\langle P \rangle} \langle \mu(x_1) \rangle .$$

In Maude it is possible to use *membership assertions* (e.g., assertions such as t:S meaning that the term t has sort S) and subsort specifications to obtain a very general notion of success predicate for uniform term tile systems, such as $\mathcal{R}_{CCS}$. Due to lack of space we refer the interested reader to [5] for its detailed description. The idea is that given a uniform term tile system $\mathcal{R}$ with horizontal signature Σ_H and vertical signature Σ_V, we define a rewriting logic with the following sorts: the sort W containing the variables of the system as constants; the sort H containing the terms over the signature Σ_H and variables in W (similarly for the sort V); the sort HV containing the terms over the signature $\Sigma_{H \cup V}$ and variables in W such that they are decomposable as terms over Σ_V applied to terms over Σ_H (similarly for VH); sorts QH, QV, QHV, and QVH are *quoted versions* of the sorts described before (we will denote the quoted version of a signature Σ_S by $\Sigma_{S'}$, adopting the convention that all the operators of the latter are *syntactically* quoted versions of the operators in Σ_S); and the sort U containing terms over the signature $\Sigma_{H \cup V \cup H' \cup V'}$ and variables in W.

Then, we add two operations quote and unquote that allow the translation of a term into its quoted version and vice versa. Finally, the rewrite rules are the stretched and quoted versions of the tiles in $\mathcal{R}$, and the unary operator top : U -> U is introduced together with two rules start and end to mark the term to be rewritten, and to begin and terminate the rewriting computation:

```
crl [start] : top(x1) => top(quote(x1)) if x1 : HV .
crl [end] : top(x1) => top(unquote(x1)) if x1 : QVH .
```

An important theorem states that a sequent $\vec{h} \xrightarrow[\vec{v}]{u} g$ is entailed by $\mathcal{R}$ if and only if the sequent $\mathtt{top}(\mathtt{u}(\vec{\mathtt{h}}(\vec{x}))) \Rightarrow \mathtt{top}(\mathtt{g}(\vec{\mathtt{v}}(\vec{x})))$ is entailed in $\hat{\mathcal{R}}$. It follows that a very simple success predicate ok can be defined as follows:

```
ceq ok(top(t)) = true iff t : VH .
```

2.2 CCS with Locations

In the philosophy of *true concurrent semantics*, the notion of *concurrency* cannot be reduced to nondeterminism via interleaving as it happens for the implementation that we have considered in the last section. To overcome this problem, one possibility is to define a relation representing those pairs of events that can occur in any order, i.e., the *commuting diamonds* of the transition system [16,17]. A uniform treatment for both the operational and the abstract concurrent semantics of a CCS-like process calculus with locations [11] is provided in [12], using (an enriched) process tile logic. In this section we propose a graph-like presentation of those systems, which can be easily implemented in Maude. Locations are introduced to allow the external observer to see an action together with the location where it takes place. As an example, this approach distinguishes process $\alpha.\beta.nil + \beta.\alpha.nil$ from $\alpha.nil|\beta.nil$ because the second process can separately perform α and β in different places, while the first process cannot.

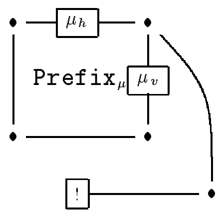
Let *Loc* be a (totally ordered) denumerable set of *locations*, ranged over by l. A *located process* P is a term generated by the following grammar:

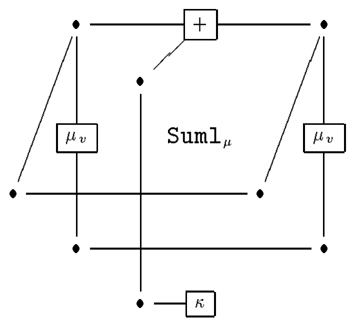
$$G ::= nil \mid \mu.G \mid G+G \mid G|G \qquad P ::= G \mid l :: P \mid P|P$$

where, for the sake of simplicity, we distinguish the *ground processes* (i.e., processes without locations), ranged over by G, G', etc.

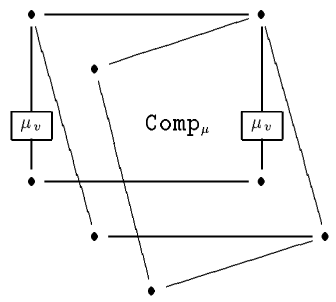
The operational semantics is defined by a transition system whose labels consist of actions together with *strings of locations*, denoted by u. In a synchronization, the strings associated to the synchronizing actions are paired. To define the concurrent operational semantics, a *concurrency relation*, usually denoted by (_ then _ χ _ then _), can be defined on the algebra of transitions and computations to identify the commuting diamonds of the system, following the approach proposed in [17]: some axioms identify the basic diamonds, consisting of two transitions performed by two processes composed in parallel, then some inductive rules propagate the diamonds in all the possible contexts. Ferrari and Montanari propose [12] a tile rewrite system such that a translation $\{\!|\_|\!\}$ from transitions to tiles can be inductively defined with the property that each diamond (t_1 then t_2 χ t_3 then t_4) implies that $\{\!|t_1|\!\} \cdot \{\!|t_2|\!\} = \{\!|t_3|\!\} \cdot \{\!|t_4|\!\}$. Due to space limitation, here we just give a brief survey of their tile system, using *hypergraphs* to model configurations and effects. Each *hyperarc* is represented with a labelled box connected to its source and target nodes. Unlabelled arcs represent *sharing*. The labels of horizontal hyperarc are taken over the signature $\Sigma_S = \{+ : 2 \longrightarrow 1, ! : 0 \longrightarrow 1, \kappa : 1 \longrightarrow 0\} \cup \bigcup_{\mu \in Act}\{\mu_h : 1 \longrightarrow 1\}$, and vertical hyperarcs are labelled over the signature $\Sigma_D = \{T : 2 \longrightarrow 2\} \cup \bigcup_{\mu \in Act}\{\mu_v : 1 \longrightarrow 1\}$. Notice that each hyperarc is labelled with an operator whose arity and multiplicity exactly match the number of source and target nodes. Each node intuitively represents a place where actions may occur, i.e., a location.

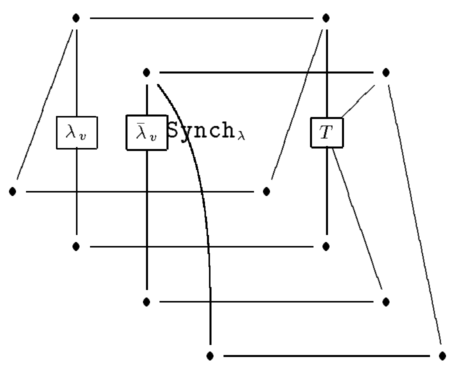
We briefly comment the tiles defined in [12], also showing the graphical representation of most of them. Tile $\mathtt{prefix}_\mu$ states that a "prefix" hyperarc can execute the corresponding action evolving to a new location. The link to the old location is maintained, since other agents could be attached there. Two tiles ($\mathtt{Suml}_\mu$ and $\mathtt{Sumr}_\mu$) model the nondeterministic sum. Whenever the "left" ("right") process makes a move, then the other process is eliminated via a nil binding κ. The locations promoted by the evolving process are propagated forward.





Rule $\texttt{Comp}_\mu$ is the most important one. It states that if two processes share the same location and one of them is making a move, then its subprocess will be allocated to the new location, whereas the parallel process will remain linked to the old location. We do not need to distinguish between a $\texttt{Compl}_\mu$ (left) and a $\texttt{Compr}_\mu$ (right), because they can be obtained one from the other using auxiliary permutation tiles. Rule $\texttt{Synch}_\lambda$ allows the synchronization of two concurrent complementary moves to take place. The resulting action T states that the two "new" locations will be both correlated to the two "old" locations, where the complementary actions took place. Two last rules `Twin` and `TwinComp` show how the T events can be propagated through sharing.





As for finite CCS, we "rotate" the rules before translating them in Maude. Multiple pointers to the same node name model sharing of locations, and explicit name management embeds interface permutations. Let us take a standard denumerable set of nodes $\{\texttt{n}(i) \mid i \in \mathbb{N}\}$ (sort `Node`). The operator `ls` constructs lists of nodes (sort `NodeList`), where the empty list is denoted by `nil`. Similarly, the operator `set` constructs set of nodes (sort `NodeSet`), the constant `empty` representing the empty set of nodes. A hyperarc (sort `Edge`) is then a triple consisting of a list of source nodes, a label (of sort `Label`) and a list of target nodes. The operator `edge` allows the construction of generic hyperarcs. A hypergraph (sort `EdgeMSet`) is just a (multi)set collection of hyperarcs (constructor `ms` and neutral element `zero`). We introduce an operator `top` to mark the whole actual configuration of the system (sort `State`). We define also some useful operators to extract information of various kind: `proj(`NL`)` returns the set of nodes contained in the node list NL, `sources(`MS`)` returns the set of source nodes contained in the hypergraph MS, `targets(`MS`)` returns the set of target nodes contained in the hypergraph MS, `vertices(`MS`)` returns the set of nodes contained in the hypergraph MS, `isIn(`E`,`ES`) = true` iff the node E is in the set of nodes ES, `children(`E`,`MS`)` returns the set of children of node E in the hypergraph MS, and `desc(`E`,`MS`)` returns the set of descendants of node E in the hypergraph MS. The label of a hyperarc can be either of sort `HSign` (associated to Σ_S), or of sort `VSign` (associated to Σ_D). We fix a denumerable set of basic actions $\{\mathsf{a}(i) \mid i \in \mathbb{N}\}$ and their complementary actions $\{\mathtt{bar}(\mathsf{a}(i)) \mid i \in \mathbb{N}\}$ (sort `Channel`), together with a special action `tau`. Given an action μ we denote the associated prefix operators μ_h in Σ_S and μ_v in Σ_D respectively by $\mathtt{h}(\mu)$ and $\mathtt{v}(\mu)$. Regarding the other operators, $+$ is denoted by `plus`, ! is denoted by `dis`, κ is denoted by `codis` and T is denoted by `t`. A special horizontal operator `alias` is introduced to propagate possible renamings caused by rule $\mathtt{Suml}_\mu$ and $\mathtt{Sumr}_\mu$. We are now ready to translate tiles into rewrite rules.

```
vars BA BA' : Channel . vars E E' Ea Eb : Node .
var Mu : Act . var MS : EdgeMSet . vars ES ES' : NodeSet .
rl [prefix] : top(ms(edge(E',h(Mu),E),MS)) =>
              top(ms(edge(E',v(Mu),E),edge(nil,dis,E),MS)) .
rl [suml] : top(ms(edge(E',v(Mu),Ea),
                   edge(ls(Ea,Eb),plus,E),MS)) =>
            top(ms(edge(Eb,codis,nil),edge(E',v(Mu),E),
                   edge(Ea,alias,E),MS)) .
*** sumr is analogous
crl [synch] : top(ms(edge(E,v(BA),Ea),
                     edge(E',v(BA'),Eb),MS)) =>
              top(ms(edge(ls(E,E'),t,ls(Ea,Eb)),MS))
    if bar(BA) == BA' .
```

The other tiles are either trivial or special cases of the previous ones, and thus omitted. The problem is that not all the rewritings are correct, hence we have to filter computations. This can be done at the meta-level using the strategies to collect rewritings. We need only to define a notion of *suc-*

cess. In particular, we have just to check if the actual state is acyclic and decomposable in a hypergraph with labels in `HSign` followed by an hypergraph with labels in `VSign`. A predicate `ok` accomplishing this task can be defined as follows, using some auxiliary operators: `okHV(`MS`) = true` iff MS is horizontal-vertical decomposable; `acyclic(`MS`) = true` iff MS is acyclic; `disjoint(`ES`,`ES'`) = true` iff $ES \cap ES' = \emptyset$; and `horiz(`MS`) = true` iff all the hyperarcs of MS have label in `HSign`.

It can be proved that successful configurations (according to the predicate `ok`) reachable in `LOCCCS` starting from the representation of a process P are all and only the behaviours of P in the tile system of Ferrari and Montanari.

3 Concluding Remarks

We have implemented and experimented in Maude the translation of two relevant tile systems for CCS-like calculi. Then, the computations of simple processes in the meta-strategies have been tested using the meta-strategies of [5]. Our experiments are encouraging, since Maude seems to offer a good trade-off between rewriting kernel efficiency and layer-swapping management (from terms to their meta-representations and vice versa).

Acknowledgments

Research supported by Office of Naval Research Contracts N00014-95-C-0225 and N00014-96-C-0114, by National Science Foundation Grant CCR-9633363, and by the Information Technology Promotion Agency, Japan, as part of the Industrial Science and Technology Frontier Program "New Models for Software Architechture" sponsored by NEDO (New Energy and Industrial Technology Development Organization). Also research supported in part by U.S. Army contract DABT63-96-C-0096 (DARPA); CNR Integrated Project *Metodi e Strumenti per la Progettazione e la Verifica di Sistemi Eterogenei Connessi mediante Reti di Comunicazione*; and Esprit Working Groups *CONFER2* and *COORDINA*. Research carried out in part while the first and the third authors were visiting at Computer Science Laboratory, SRI International, and the third author was a visiting scholar at Stanford University.

References

1. F. Gadducci. On the Algebraic Approach to Concurrent Term Rewriting. PhD Thesis TD-96-02. Department of Computer Science, University of Pisa (1996).

2. F. Gadducci and U. Montanari. The Tile Model. In: G. Plotkin, C. Stirling, M. Tofte, Eds., *Proof, Language and Interaction: Essays in Honour of Robin Milner*, MIT Press, to appear. Also Technical Report TR-96-27, Department of Computer Science, University of Pisa (1996).
3. J. Meseguer. Rewriting as a Unified Model of Concurrency. SRI Technical Report, CSL-90-02R, February 1990. Revised June 1990.
4. J. Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency. *Theoretical Computer Science* **96** (1992). pp. 73–155.
5. R. Bruni, J. Meseguer, and U. Montanari. Process and Term Tile Logic. Technical Report, SRI International, to appear.
6. M. Clavel and J. Meseguer. Reflection and Strategies in Rewriting Logic. In: J. Meseguer, Ed., *Proc. First International Workshop on Rewriting Logic and its Applications. ENTCS* **4** (1996).
7. M. Clavel and J. Meseguer. Axiomatizing Reflective Logics and Languages. In: G. Kiczales, Ed., *Proc. Reflection'96.* San Francisco (1996).
8. M.G. Clavel, S. Eker, P. Lincoln, and J. Meseguer. Principles of Maude. In: J. Meseguer, Ed., *Proc. First International Workshop on Rewriting Logic and its Applications. ENTCS* **4** (1996).
9. M. Clavel and J. Meseguer. Internal Strategies in a Reflective Logic. In: B. Gramlich, and H. Kirchner, Eds., *Proc. of the CADE-14 Workshop on Strategies in Automated Deduction.* Townsville, Australia, (1997).
10. J. Meseguer and U. Montanari. Mapping Tile Logic into Rewriting Logic. In: F. Parisi-Presicce, Ed., *Proc. 12th WADT Workshop on Algebraic Development Techniques*, Springer *LNCS* **1376** (1998).
11. G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn. Observing Localities. *Theoretical Computer Science* **114** (1993), pp. 31–61.
12. G.L. Ferrari and U. Montanari. Tiles for Concurrent and Located Calculi. In: C. Palamidessi, J. Parrow, Eds., *Proceedings of EXPRESS'97. ENTCS* **7** (1997).
13. R. Milner. *Communication and Concurrency.* Prentice-Hall (1989).
14. N. Martí-Oliet and J. Meseguer. *Rewriting Logic as a Logical and Semantic Framework.* SRI T.R., CSL-93-05 (1993). To appear in D. Gabbay, Ed., *Handbook of Philosophical Logic*, Kluwer Academic Publishers.
15. P. Viry. Rewriting Modulo a Rewrite System Technical Report TR-95-20, Department of Computer Science, University of Pisa (1995).
16. G. Boudol and I. Castellani. Flow Models of Distributed Computations: Three Equivalent Semantics for CCS. *Information and Computation* **114***(2)* (1993), pp. 247–314.
17. G.L. Ferrari and U. Montanari. Towards the Unification of Models for Concurrency. In *Proceeding CAAP'90.* Springer *LNCS* **431** (1990).