Mapping Tile Logic into Rewriting Logic*

José Meseguer[1] and Ugo Montanari[2]

[1] Computer Science Laboratory, SRI International, Menlo Park,
meseguer@csl.sri.com
[2] Dipartimento di Informatica, Università di Pisa, ugo@di.unipi.it

Abstract. *Rewriting logic* extends to concurrent systems with state changes the body of theory developed within the algebraic semantics approach. It is both a foundational tool and the kernel language of several implementation efforts (Cafe, ELAN, Maude). *Tile logic* extends (unconditional) rewriting logic since it takes into account state changes with side effects and synchronization. It is especially useful for defining compositional models of computation of reactive systems, coordination languages, mobile calculi, and causal and located concurrent systems. In this paper, the two logics are defined and compared using a recently developed algebraic specification methodology, *membership equational logic*. Given a theory T, the rewriting logic of T is the free monoidal *2*-category, and the tile logic of T is the free monoidal *double* category, both generated by T. An extended version of monoidal 2-categories, called *2VH-categories*, is also defined, able to include in an appropriate sense the structure of monoidal double categories. We show that 2VH-categories correspond to an extended version of rewriting logic, which is able to embed tile logic, and which can be implemented in the basic version of rewriting logic using suitable *internal strategies*. These strategies can be significantly simpler when the theory is *uniform*. A uniform theory is provided in the paper for CCS, and it is conjectured that uniform theories exist for most process algebras.

1 Introduction

Rewriting logic [27, 28, 31] extends to concurrent systems with state changes the body of theory developed within the algebraic semantics approach. It can also be

* 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 while the second author was on leave at Computer Science Laboratory, SRI International, and visiting scholar at Stanford University.

considered as a semantic framework for concurrency. Its aim as a foundational tool is to faithfully express a wide range of models of computation. As shown in [27, 31], a rewriting theory $\mathcal{R}$ yields a cartesian 2-category [22] $\mathcal{L}(\mathcal{R})$ which does for $\mathcal{R}$ what a Lawvere theory $\mathcal{L}(T)$ does for an equational theory T. Moreover, the models of a rewrite theory $\mathcal{R}$ can be considered as 2-product-preserving 2-functors $M : \mathcal{L}_{\mathcal{R}} \to \mathbf{Cat}$, that is, as structures based on the 2-category **Cat** rather than on the category **Set**.

Rewriting logic is not only interesting as a theoretical framework. Several language implementation efforts (Cafe [18], ELAN [3], Maude [29, 10]) in various countries have adopted rewriting logic as their semantic basis, and support either executable specifications or parallel programming in rewriting logic. In particular the object-oriented language Maude developed at SRI International is based on rewriting logic and is efficiently implemented. Maude is also equipped with important extra features, like internal execution strategies and reflective logic, which allow it to easily embed a variety of other logics. A workshop has been recently dedicated to all the aspects of rewriting logic [32].

The tile model [19, 20, 21] relies on certain rewrite rules with side effects, called *tiles*, reminiscent of SOS rules [38] and SOS *contexts* [24]. A related model is also *structured transition systems* [13]. The tile model has been conceived with similar aims and similar algebraic structure as rewriting logic, and it extends rewriting logic (in the unconditional case), since it takes into account state changes with side effects and syncronization.

Tile systems can be seen as monoidal double categories [14] and tiles themselves as double cells. However, the tile model (as rewriting logic) can be presented in a purely logical form [21], tiles being just special sequents subject to certain inference rules and to certain proof normalization axioms. The fact that tile logic extends rewriting logic is reflected at the categorical level by the observation that 2-categories are a special case of double categories.

We now briefly introduce the tile model. A tile has the form:

$$s \xrightarrow[b]{a} s'$$

and states that the *initial configuration* s of the system evolves to the *final configuration* s' producing an *effect* b. However s is in general *open* (not closed) and the rewrite step producing the effect b is actually possible only if the subcomponents of s also evolve producing the *trigger* a. Both trigger a and effect b are called *observations*, and model the interaction, during a computation, of the system being described with its environment. More precisely, both system configurations are equipped with an *input* and an *output interface*, and the trigger just describes the evolution of the input interface from its initial to its final configuration. Similarly for the effect. It is convenient to visualize a tile as a two-dimensional structure (see Fig. 1), where the horizontal dimension corresponds to the extension of the system, while the vertical dimension corresponds to the extension of the computation. Actually, we should also imagine a third dimension (the thickness of the tile), which models parallelism: configurations, observations, interfaces and tiles themselves are all supposed to consist of several components in parallel.

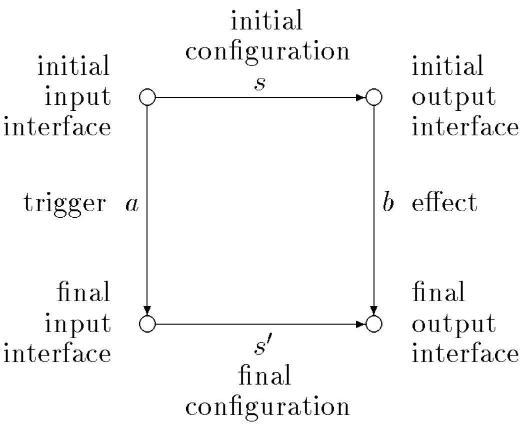

Fig. 1. A tile.

Both configurations and observations are assumed to be equipped with operations of parallel and sequential composition. Similarly, tiles themselves possess three operations of composition: parallel ($\_\otimes\_$), horizontal ($\_*\_$), and vertical ($\_\cdot\_$) composition.

The operation of parallel composition is self explanatory. Vertical composition models sequential composition of transitions and computations. Horizontal composition corresponds to synchronization: the effect of the first tile acts as trigger of the second tile, and the resulting tile expresses the synchronized behavior of both.

Computing in *tile logic* consists of starting from a set of basic tiles called *rewrite rules* and of applying the composition rules in all possible ways. In general the structure of a tile obtained in this way is specified by a *proof*, built up from the basic tiles used in the derivation and from the composition operations performed on them, up to certain normalizing axioms (which are those of monoidal double categories).

A tile reduces to a rewriting logic rule when trigger and effect are identities, i.e., when input and output interfaces stay idle during the step. In rewriting logic, the rewriting process is fully asynchronous, in the sense that matching rules can be independently applied, except when their redexes overlap. A tile with nontrivial trigger or effect, instead, can be applied only when it matches the whole state, or when it can be composed (usually horizontally or in parallel) with other tiles to build such a global tile.

Abstract semantics can be defined on tiles in the obvious way, i.e., by considering triggers and effects as external observations. Equivalences and congruences of configurations can then be defined in terms of traces, or via bisimilarity, or in other ways known for process algebras[3]. Logics of properties, and verification methods can also be defined in the usual way. We do not consider abstract semantics in this paper. Definitions and some important semantic properties (e.g., conditions for ensuring that bisimilarity is a congruence) are presented in [21].

Tiles have been used with success for modeling in detail several classes of applications. The most obvious is *coordination languages* [9]. In this research area, a common distinction is drawn between coordinators and basic (software) agents. Basic agents are considered as black boxes (to ensure interoperability) which interact with the coordinators via suitable protocols. Coordinators can be hierarchically nested and can compete and/or cooperate to achieve certain global goals, which can be fine grain, as transactions, or coarse grain, as work plans in computer-supported collaborative work (CSCW). Tiles are suitable for modeling coordination languages and systems, since triggers and effects naturally represent coordination protocols, and tiles themselves define the behavior of coordinators. In [36], a simple coordination model based on graph rewriting and synchronization is presented, and its behavior is directly described by a particular but important class of tiles (algebraic tiles). The hard computational problem of tile synchronization (which in general is NP-complete) is reduced to a distributed version of constraint solving, for which effective approaches exist. In [7, 8] the simplest possible interpretation is taken for system configurations, triggers and effects: markings of P/T Petri nets. However, horizontal composition of tiles yields *transition synchronization*, an important feature missing in ordinary nets (where only *token* synchronization is provided), and usually achieved only through complex constructions.

A main advantage of tiles is to offer an SOS-like compositional framework where data structures are not restricted to be syntactic terms. A small variation over ordinary terms which has proved very expressive is term graphs [1, 12]. They are analogous to terms, but the sharing of subterms can be specified also for closed[4] (i.e., without variables) term graphs. A first-class notion of sharing is essential for several important applications. One of them is *mobile processes*. Here the shared entities are names, which can be free or bound (i.e., global or local). Names usually are links to communication channels, or to objects, or to remote resources in general. Names have no meaning in themselves, except for specifying sharing, and in fact can be α-converted freely. Models of computation based on free and bound names are widespread (e.g., logic programming, λ-calculus, or process algebras with restriction or hiding), but the most general case is represented by models able to communicate names, and thus such that, when sending a private name to the external world, must make global a previously local name (extrusion). The most studied model of this kind is the π-calculus [35]. Name extrusion and handling makes π-calculus infinite branching,

[3] In tile rewrite systems modeling process algebras, the tiles with a closed agent as initial configuration have empty trigger, i.e., they can be considered as transitions of a labelled transition system, the label being the effect.

[4] Terms can share variables, but shared subterms of closed terms can be freely copied, always yielding an equivalent term.

and requires special notions of bisimulation. Tiles equipped with term graphs handle names as wires, i.e., every sharing connection must be explicitly defined. As a consequence, extrusion does not present any problem and can be specified using finite branching only. A presentation based on tiles of the ordinary abstract semantics of asynchronous π-calculus can be found in [15]. It employs the ordinary notion of tile bisimilarity.

Models for *concurrent and distributed systems* are the last application area we are describing. When concurrency is a primitive notion, models of process calculi usually include commuting transition diamonds[5] and observations of abstract locations or of causality links between events. Term graphs are again the most convenient structure. Sharing is used within configurations for modeling the operator $\_ \mid \_$ of process algebras, which in this context means sharing *the same location*. Within observations, sharing is used to express the fact that two events share *the same cause*, or, equivalently, that the same location has two different sublocations. In [16] it is shown that both operational and abstract semantics of CCS [34] with locations [4] can be conveniently rephrased within the tile model. Again, infinite branching is avoided and the ordinary notion of bisimilarity is employed.

The aim of this paper is to build a bridge from tile logic to rewriting logic. More generally, we think that it is conceptually useful to embed both models into a common semantic framework which makes clear not only their connections, but also their general role. For this purpose, a recent specification methodology, *membership equational logic* [30, 5, 33] and more specifically *partial membership equational logic* [33] (*PMEqtl*), is employed. *PMEqtl* theories consist of a partial ordering of sorts, a signature of operations, and Horn clauses whose atomic formulas are either equations $t = t'$, or assertions $t : s$ stating the membership of a term in a certain sort. *PMEqtl* extends order-sorted equational logic and supports partiality, subsorts, operator overloading and error specification. The key aspect of *PMEqtl* relevant to this paper is the ability to specify partial algebras with overloading, subsorts, and operations with equationally specified domains of definition. In fact, categories, and *a fortiori* 2-categories and double categories, are partial algebras of this kind, where the operation of arrow and cell composition is defined only if certain conditions are satisfied. A self-contained description of a simplified version of *PMEqtl*, *one-kinded partial membership equational logic*, is included. In particular, for this version of *PMEqtl* the notion of *tensor product* of theories is developed.

Tensor product (see for instance [25, 37]) is a well-known construction for ordinary algebraic (Lawvere) theories. Its importance can be understood by observing that the algebraic structures of a theory T can be defined not only on sets, the standard case $\mathbf{Alg}_T(\mathbf{Set})$, but also on any category $\mathbf{C}$ with suitable products or limits, to yield a category $\mathbf{Alg}_T(\mathbf{C})$. In particular, given two theories T and T', we can consider T'-algebras on the category of T-algebras, that

[5] Commuting transition diamonds define *concurrent* pairs of events, i.e., pairs of events which can occur in any order.

is the category $\mathbf{Alg}_{T'}(\mathbf{Alg}_T(\mathbf{C}))$, or instead T-algebras on the category of T'-algebras ($\mathbf{Alg}_T(\mathbf{Alg}_{T'}(\mathbf{C}))$). Regardless of the order, we obtain the same result up to isomorphism, namely the category of algebras for the *tensor product* $T \otimes T'$ of both theories. That is, we have isomorphisms:

$$\mathbf{Alg}_{T'}(\mathbf{Alg}_T(\mathbf{Set})) \simeq \mathbf{Alg}_T(\mathbf{Alg}_{T'}(\mathbf{Set})) \simeq \mathbf{Alg}_{T\otimes T'}(\mathbf{Set}).$$

The tensor product of theories is a very robust notion, rooted in the foundations of algebraic semantics, and very useful too, since the tensor theory $T \otimes T'$ can be effectively constructed. The construction for *PMEqtl* presented in this paper is somewhat more complex than, but analogous to, the construction for algebraic theories.

The relevance of the tensor product construction for tiles is that the theory of monoidal double categories can be obtained as the tensor product of the theory of categories (twice) with the theory of monoids. Thus, one can argue that if the desired model of computation must have operations of parallel and horizontal composition (to build more parallel and larger systems) and of vertical composition (to build longer computations), then monoidal double categories are the most natural answer. Of course it is always possible to add further structure restricting the models of interest to suitable subcategories.

Besides its use in specifying different models of computation by the tensor pruduct of theories, *PMEqtl* is also used in the paper for defining theory morphisms, which yield (as for Lawvere theories) corresponding adjunctions between categories of models. Free constructions of this kind are employed to define the algebraic semantics of tile rewrite systems.

The actual connection between rewriting logic and tile logic is obtained by defining in *PMEqtl* an extended version of 2-categories, called 2VH-categories, able to include in an appropriate sense the structure of double categories. In 2VH-categories, the operations of horizontal and vertical composition between tiles are derived operations based on those between 2-cells. It is possible to see that 2VH-categories correspond to an *extended* rewriting logic (*ERWL*), which is able to embed tile logic, and which can be implemented in the basic version of rewriting logic using suitable *internal strategies*. These strategies can be significantly simpler when the theory is *uniform*. In this case the formal translation presented in the paper yields an efficient implementation procedure. Interestingly enough, this is the case for CCS [34] and, we conjecture, for most process algebras. In the paper the case study of (finite) CCS is carried out in detail, and the corresponding rewriting theory is extracted and applied to a simple example.

The paper is organized as follows. Section 2 contains the background on partial membership equational logic, with double categories as an example of application of the tensor product construction. Section 3 introduces 2-categories as a special case of double categories, 2VH-categories, and computads[6] as models of rewrite systems. Section 4 defines tile rewrite systems, tile logic and extended

[6] Computads in this paper are essentially impoverished versions of monoidal double categories, being equipped with sets of double cells where no operation of composition is defined.

rewriting logic, and proves the adequacy of the latter. Also a tile rewrite system for CCS is introduced, which is uniform. An example of a nonuniform rewrite system is also provided.

2 Partial Membership Equational Logic

This section defines the basic notions of *partial membership equational logic* [33] (*PMEqtl*). This is a logic of partial algebras with subsorts and subsort polymorphism whose sentences are Horn clauses on equations $t = t'$ and membership assertions $t : s$. We treat here the *one kinded* case, in which the poset of sorts has a single connected component. A more detailed exposition for the many-kinded case can be found in [33].

2.1 Partial Algebras and Membership Equational Theories

Definition 1. *(signature)* A *signature* is a triple $\Omega = (S, \leq, \Sigma)$, with $(S, \leq)$ a poset with a top element $\top$, and $\Sigma = \{\Sigma_k\}_{k \in N}$ a family of sets indexed by natural numbers. Poset $(S, \leq)$ is called the poset of *sorts* of Ω. □

Definition 2. *(partial Ω-algebra)* Given a signature $\Omega = (S, \leq, \Sigma)$, a *partial Ω-algebra* A assigns:

(i) to each $s \in S$ a set A_s, in such a way that whenever $s \leq s'$, we have $A_s \subseteq A_{s'}$;
(ii) to each $f \in \Sigma_k$, $k \geq 0$, a partial function $A_f : A_\top^k \rightharpoonup A_\top$.

Given two partial Ω-algebras A and B, an *Ω-homomorphism* is a function $h : A_\top \to B_\top$ such that:

(i) for each $s \in S$, $h(A_s) \subseteq B_s$;
(ii) for each $f \in \Sigma_k$, $k \geq 0$, and $\mathbf{a} \in A_\top^k$, if $A_f(\mathbf{a})$ is defined, then $B_f(h^k(\mathbf{a}))$ is also defined and equal to $h(A_f(\mathbf{a}))$.

Notice that, because of condition **(i)**, for each $s \in S$ the function h restricts to a function $h|_s : A_s \to B_s$.

This determines a category $\mathbf{PAlg}_\Omega$.

□

Definition 3. *(declaration, formula and sentence)*

Let $\Omega = (S, \leq, \Sigma)$ be a signature. Given a set of variables $X = \{x_1, \ldots, x_m\}$, a *variable declaration* $\tilde{X}$ is a sequence

$$x_1 : \overline{s_1}, \ldots, x_m : \overline{s_m}$$

where for each $i = 1, \ldots, m$, $\overline{s_i}$ is a set of sorts $\{s_{i1}, \ldots, s_{il_i}\}$.

Atomic Ω-formulas are either equations

$$t = t'$$

where $t, t' \in T_\Sigma(X)$ (with $T_\Sigma(X)$ the usual free Σ-algebra on variables X) or membership assertions of the form

$$t : s$$

where $t \in T_\Sigma(X)$.

General Ω-sentences are then Horn clauses of the form

$$\forall \tilde{X}\ t = t' \Leftarrow u_1 = v_1\ \wedge\ \ldots\ \wedge\ u_n = v_n\ \wedge\ w_1 : s_1\ \wedge\ \ldots\ \wedge\ w_m : s_m$$

or of the form

$$\forall \tilde{X}\ t : s \Leftarrow u_1 = v_1\ \wedge\ \ldots\ \wedge\ u_n = v_n\ \wedge\ w_1 : s_1\ \wedge\ \ldots\ \wedge\ w_m : s_m$$

where the t, t', u_i, v_i and w_j are all terms in $T_\Sigma(X)$.

Definition 4. *(theory, model)*

Given a partial Ω-algebra A and a variable declaration $\tilde{X}$, we can define assignments $a : \tilde{X} \to A$ in the obvious way (if $x : \overline{s}$, and $s \in \overline{s}$, then we must have $a(x) \in A_s$) and then we can define a partial function $\overline{a} : T_\Sigma(X) \rightharpoonup A_\top$, extending a in the obvious way. For atomic sentences we then define satisfaction by

$$A, a \models\ t = t'$$

meaning that $\overline{a}(t)$ and $\overline{a}(t')$ are both defined and $\overline{a}(t) = \overline{a}(t')$ (that is, we take an *existence equation* interpretation) and by

$$A, a \models\ t : s$$

meaning that $\overline{a}(t)$ is defined and $\overline{a}(t) \in A_s$.

Satisfaction of Horn clauses is then defined in the obvious way. Given a set Γ of Ω-sentences, we then define $\mathbf{PAlg}_{\Omega,\Gamma}$ as the full subcategory of $\mathbf{PAlg}_\Omega$ determined by those partial Ω-algebras that satisfy all the sentences in Γ. In other words, the pair $T = (\Omega, \Gamma)$ is a *theory*, and $\mathbf{PAlg}_T = \mathbf{PAlg}_{\Omega,\Gamma}$ is the category of its models.

Example 1. (category)

The theory of categories T_{Cat} is a theory in partial membership equational logic. Its poset of sorts has sorts *Object* and *Arrow* with *Object* $\leq$ *Arrow*. There are two unary *domain* and *codomain* operations d and c, and a binary composition operation _;_. The theory is presented by the following axioms, for which we use a self-explaining Maude-like notation [10]. In the following, to avoid extra levels of indexing, we will denote theories by their Maude name, e.g., CAT or CAT instead of T_{Cat}.

```
fth CAT is
  sorts Object Arrow .
  subsorts Object < Arrow .
  ops d(_) c(_) _;_ .
  vars f g h : Arrow .
```

```
  var a : Object .
  mbs d(f) , c(f) : Object .
  eq d(a) = a .
  eq c(a) = a .
  cmb f;g : Arrow iff c(f) = d(g) .
  ceq d(f;g) = d(f) if c(f)=d(g) .
  ceq c(f;g) = c(g) if c(f)=d(g) .
  ceq a;f = f if d(f) = a .
  ceq f;a = f if c(f) = a .
  ceq (f;g);h = f;(g;h) if c(f) = d(g) and c(g) = d(h) .
endfth
```

It is easy to check that a model of CAT is exactly a category (in which objects coincide with identity arrows), and that a CAT-homomorphism is exactly a functor. □

Definition 5. *(signature and theory morphism)*
Given two signatures $\Omega = (S, \leq, \Sigma)$ and $\Omega' = (S', \leq', \Sigma')$, a *signature morphism* $H : \Omega \to \Omega'$ is given by:

1. a monotonic function $H : (S, \leq) \to (S', \leq')$, and
2. an N-indexed family of functions $\{H_k : \Sigma_k \to \Sigma'_k\}_{k \in N}$.

Such a signature morphism induces a forgetful functor $U_H : \mathbf{PAlg}_{\Omega'} \to \mathbf{PAlg}_{\Omega}$, where for each $A' \in \mathbf{PAlg}_{\Omega'}$ we have:

1. for each $s \in S$, $U_H(A')_s = A'_{H(s)}$;
2. for each[7] $f \in \Sigma_k$, $U_H(A')_f = A'_{H(f)} \cap (\underbrace{A'_{H(\top)} \times \ldots \times A'_{H(\top)}}_{k} \times A'_{H(\top)})$;
3. for each Ω'-homomorphism $h' : A' \to B'$, $U_H(h') = h'|_{H(\top)} : A'_{H(\top)} \to B'_{H(\top)}$, which is well-defined as a restriction of h' because h' is sort-preserving.

Given theories (Ω, Γ) and (Ω', Γ'), a *theory morphism* $H : (\Omega, \Gamma) \to (\Omega', \Gamma')$ is a signature morphism $H : \Omega \to \Omega'$ such that $U_H(\mathbf{PAlg}_{\Omega',\Gamma'}) \subseteq \mathbf{PAlg}_{\Omega,\Gamma}$, so that U_H restricts to a forgetful functor $U_H : \mathbf{PAlg}_{\Omega',\Gamma'} \to \mathbf{PAlg}_{\Omega,\Gamma}$. □

The reader is referred to [33] for proof-theoretical conditions on Γ and Γ' to ensure that a signature morphism $H : \Omega \to \Omega'$ is a theory morphism $H : (\Omega, \Gamma) \to (\Omega', \Gamma')$.

Proposition 6. (free construction associated to a theory morphism [33])
Given a theory morphism $H : (\Omega, \Gamma) \to (\Omega', \Gamma')$, its associated forgetful functor $U_H : \mathbf{PAlg}_{\Omega',\Gamma'} \to \mathbf{PAlg}_{\Omega,\Gamma}$ has a left adjoint $F_H : \mathbf{PAlg}_{\Omega,\Gamma} \to \mathbf{PAlg}_{\Omega',\Gamma'}$. □

[7] Notice that $U_H(A')_f = A'_{H(f)}$ would not be correct in general, since $U_H(A')_\top = A'_{H(\top)}$, where $H(\top)$ is not necessarily $\top$.

Definition 7. *(conservative, complete and persistent morphism)*
A theory morphism $H : (\Omega, \Gamma) \to (\Omega', \Gamma')$ is *conservative* (resp. *complete*, *persistent*) *w.r.t. sort* s if, for each algebra $A \in \mathbf{PAlg}_{\Omega,\Gamma}$, the component $(\eta_A)_s : A_s \to (U_H(F_H(A)))_s$ corresponding to s of the unit of the adjunction associated to H is injective (resp. surjective, bijective). Morphism H is *conservative* (resp. *complete*, *persistent*) if it is conservative (resp. complete, persistent) w.r.t. all sorts $s \in S$. □

Definition 8. *(subalgebra)*
Given a signature $\Omega = (S, \leq, \Sigma)$ and a partial Ω-algebra A, an *Ω-subalgebra* B of A is an S-sorted family of subsets $\{B_s \subseteq A_s\}_{s \in S}$ such that:

(i) it is closed under the operations of Σ, that is, for each $f \in \Sigma_k$, and for each tuple $(b_1, \ldots, b_k) \in B_\top^k$, if $A_f(b_1, \ldots, b_k)$ is defined, then $A_f(b_1, \ldots, b_k) \in B_\top$;
(ii) it is closed under subsorts, in the sense that for each sort $s \in S$ we have $B_s = A_s \cap B_\top$.

It is clear that B with such operations and sorts is itself an Ω-algebra, and that the inclusion function $B \subseteq A$ is an Ω-homomorphism. □

It is also easy to prove the following.

Lemma 9. *For any set Γ of Horn sentences in partial membership equational logic, the category $\mathbf{PAlg}_{\Omega,\Gamma}$ is closed under Ω-subalgebras, i.e., if $A \in \mathbf{PAlg}_{\Omega,\Gamma}$ and B is an Ω-subalgebra of A, then $B \in \mathbf{PAlg}_{\Omega,\Gamma}$.* □

Example 2. (subcategory)
For the theory $(\Omega, \Gamma) = CAT$ of Example 1, the subalgebras of a category $\mathbf{C}$ are exactly its subcategories. □

Note that the notion of Ω-subalgebra just defined is strictly stronger than that of Ω-monomorphism. It is easy to check that, in the category $\mathbf{PAlg}_\Omega$, $m : C \to A$ is a monomorphism iff the associated function $m : C_\top \to A_\top$ is injective. Of course, by taking the smallest image of m, any such monomorphism always factors through an isomorphism and an inclusion $C \xrightarrow{\sim} B \hookrightarrow A$, where B is a *weak* subalgebra, as defined below.

Definition 10. *(weak subalgebra)*
Given a signature $\Omega = (S, \leq, \Sigma)$ and a partial Ω-algebra A, a *weak Ω-subalgebra* of A is a partial Ω-algebra B such that $B_\top \subseteq A_\top$ and such that the inclusion map $B \hookrightarrow A$ is an Ω-homomorphism. □

In general, given a set Γ of Horn sentences in partial membership equational logic, and a partial algebra $A \in \mathbf{PAlg}_{\Omega,\Gamma}$, a weak subalgebra B of A need not satisfy the sentences Γ. For example, given a nonempty category $\mathbf{C}$, the weak subalgebra with same arrows and objects as $\mathbf{C}$, but with all operations everywhere undefined is not a category. However, for $(\Omega, \Gamma) = CAT$, the following relationship happens to hold between subalgebras and weak subalgebras.

Example 3. Given a category $\mathbf{C}$, if $\mathbf{D} \subseteq \mathbf{C}$ is a weak subalgebra and $\mathbf{D}$ itself is a category, then $\mathbf{D} \subseteq \mathbf{C}$ is a subalgebra, that is, a subcategory.

2.2 The Tensor Product Construction

Following lines similar to those in [39], and using also the categorical axiomatization of canonical inclusions in a category $\mathbf{C}$ as a poset category of special monos $\mathbf{I} \subseteq \mathbf{C}$ satisfying suitable axioms suggested in [26], one can, given a signature $\Omega = (S, \leq, \Sigma)$, define partial algebras in a category $\mathbf{C}$ with finite limits and a suitable poset of canonical inclusions $\mathbf{I}$. Each sort s has an associated object A_s, and if $s \leq s'$ there is a canonical inclusion $A_s \hookrightarrow A'_s$ in $\mathbf{I}$. Given $f \in \Sigma_k$ we associate to it an arrow $Dom(A_f) \to A_\top$ in $\mathbf{C}$, where $Dom(A_f)$ is a subobject with a canonical inclusion $Dom(A_f) \hookrightarrow A_\top^k$. In this way we can define categories $\mathbf{PAlg}_\Omega(\mathbf{C})$ and $\mathbf{PAlg}_{\Omega,\Gamma}(\mathbf{C})$ so that our categories $\mathbf{PAlg}_\Omega$ and $\mathbf{PAlg}_{\Omega,\Gamma}$ are the special case $\mathbf{PAlg}_\Omega(\mathbf{Set})$ and $\mathbf{PAlg}_{\Omega,\Gamma}(\mathbf{Set})$. It is not hard to check that $\mathbf{PAlg}_\Omega$ and $\mathbf{PAlg}_{\Omega,\Gamma}$ are categories with limits, and that Ω-subalgebra inclusions $A \subseteq B$ constitute a poset category of canonical inclusions. Therefore, given two theories $T = (\Omega, \Gamma)$ and $T' = (\Omega', \Gamma')$, we can consider the category $\mathbf{PAlg}_T(\mathbf{PAlg}_{T'})$. For example, for T the theory of monoids and T' the theory of categories, $\mathbf{PAlg}_T(\mathbf{PAlg}_{T'})$ is the category of strict monoidal categories (see Section 3.2).

Since the poset $\mathbf{I} \subseteq \mathbf{PAlg}_{T'}$ of canonical inclusions that we have chosen is that of subalgebra inclusions, given a theory $T = (\Omega, \Gamma)$, each subsort relation $s \leq s'$ in Ω will be interpreted as a T'-subalgebra inclusion $A_s \hookrightarrow A'_s$, and each partial k-ary operation f in Ω will be interpreted as a T'-subalgebra inclusion $Dom(A_f) \hookrightarrow A_\top^k$ together with a T'-homomorphism $Dom(A_f) \to A_\top$. Furthermore, in order for these interpretations to yield a T-algebra structure in $\mathbf{PAlg}_{T'}$, the axioms Γ must be satisfied.

In a way analogous to algebraic theories [25, 17], to lim theories [40] and to sketches [23], there is then a theory $T \otimes T'$ in partial membership equational logic such that

$$\mathbf{PAlg}_{T \otimes T'} \simeq \mathbf{PAlg}_T(\mathbf{PAlg}_{T'}) \simeq \mathbf{PAlg}_{T'}(\mathbf{PAlg}_T).$$

Notice that we could have chosen a bigger poset $\mathbf{I}' \subseteq \mathbf{PAlg}_{T'}$ of subalgebra inclusions, yielding a looser definition of $\mathbf{PAlg}_T(\mathbf{PAlg}_{T'})$. A natural choice for $\mathbf{I}'$ would have been the set of weak subalgebra inclusions. This would yield a notion of tensor product of theories equivalent to the tensor product of their corresponding sketches. However, as we have already pointed out, the notion of weak subalgebra is too loose, giving rise in general to somewhat unintuitive models; for this reason we favor instead the notion of tensor product associated to subalgebras. Nevertheless, in the special case $T' = CAT$, because of the property mentioned in Example 3, the definition of $\mathbf{PAlg}_T(\mathbf{PAlg}_{CAT})$ is the same whether we choose subalgebras or instead weak subalgebras as canonical inclusions in $\mathbf{PAlg}_{CAT}$.

The explicit definition of $T \otimes T'$ is as follows.

Definition 11. *(tensor product)*
Let $T = (\Omega, \Gamma)$ and $T' = (\Omega', \Gamma')$ be theories in partial membership equational logic, with $\Omega = (S, \leq, \Sigma)$ and $\Omega' = (S', \leq', \Sigma')$. Then their *tensor product* $T \otimes T'$ is the theory with signature $\Omega \otimes \Omega'$ having:

(i) poset of sorts $(S, \leq) \times (S', \leq')$;
(ii) signature $\Sigma \otimes \Sigma'$, with an operator $f^l \in (\Sigma \otimes \Sigma')_n$ for each $f \in \Sigma_n$, and with an operator $g^r \in (\Sigma \otimes \Sigma')_m$ for each $g \in \Sigma'_m$. In particular, for f a constant in Σ_0 we get a constant f^l in $(\Sigma \otimes \Sigma')_0$.

The axioms of $T \otimes T'$ are the following:

A. Inherited Axioms.
For each axiom in Γ

$$\alpha \;=\; \forall(x_1 : \overline{s_1}, \ldots, x_m : \overline{s_m})\ \varphi(\mathbf{x}) \Leftarrow c(\mathbf{x})$$

with $\overline{s_i} = \{s_{i1}, \ldots, s_{il_i}\}$, $1 \leq i \leq m$, we introduce an axiom

$$\alpha^l \;=\; \forall(x_1 : \overline{s_1^l}, \ldots, x_m : \overline{s_m^l})\ \varphi^l(\mathbf{x}) \Leftarrow c^l(\mathbf{x})$$

with $\overline{s_i^l} = \{(s_{i1}, \top'), \ldots, (s_{il_i}, \top')\}$, $1 \leq i \leq m$, and with φ^l, c^l the obvious translations of φ, c obtained by replacing each $f \in \Sigma$ by its corresponding f^l.
Similarly, we define for each axiom $\beta \in \Gamma'$ the axiom β^r and impose all these axioms.

B. Subalgebra Axioms.
(i) For each $f \in \Sigma_n$ and each $s' \in S'$, $s' \neq \top'$, we introduce the axiom:

$$\begin{aligned}&\forall(x_1 : (\top, s'), \ldots, x_n : (\top, s'))\\ &\qquad f^l(x_1, \ldots, x_n) : (\top, s') \Leftarrow f^l(x_1, \ldots, x_n) : (\top, \top').\end{aligned}$$

(ii) For each $g_m \in \Sigma'_m$ and each $s \in S$, $s \neq \top$, we introduce the axiom:

$$\begin{aligned}&\forall(x_1 : (s, \top'), \ldots, x_m : (s, \top'))\\ &\qquad g^r(x_1, \ldots, x_m) : (s, \top') \Leftarrow g^r(x_1, \ldots, x_m) : (\top, \top').\end{aligned}$$

(iii) For each $(s, s') \in S \times S'$ with $s \neq \top$ and $s' \neq \top'$, we have the axiom:

$$\forall x : (\top, \top')\ \ x : (s, s') \Leftarrow x : (\top, s')\ \wedge\ x : (s, \top').$$

C. Homomorphism Axioms.

For each $f \in \Sigma_n, g \in \Sigma'_m, n + m \geq 0$, we introduce the axiom:

$$\begin{aligned}\forall \mathbf{x}\ \ & f^l(g^r(\mathbf{x_{1\cdot}}), \ldots, g^r(\mathbf{x_{n\cdot}})) = g^r(f^l(\mathbf{x_{\cdot 1}}), \ldots, f^l(\mathbf{x_{\cdot m}})) \Leftarrow\\ &\Leftarrow \textstyle\bigwedge_{1 \leq i \leq n} g^r(\mathbf{x_{i\cdot}}) : (\top, \top')\ \ \wedge\ \ \bigwedge_{1 \leq j \leq m} f^l(\mathbf{x_{\cdot j}}) : (\top, \top').\end{aligned}$$

where

$$
\begin{aligned}
\mathbf{x} \;\; &= \{x_{ij} : (\top, \top')\}_{1\leq j\leq m}^{1\leq i\leq n}, \\
\mathbf{x_{i\cdot}} &= \{x_{ij} : (\top, \top')\}_{1\leq j\leq m}, \quad 1 \leq i \leq n \\
\mathbf{x_{\cdot j}} &= \{x_{ij} : (\top, \top')\}_{1\leq i\leq n}, \quad 1 \leq j \leq m.
\end{aligned}
$$

□

The essential property of $T \otimes T'$ is expressed in the following theorem, whose proof will be given elsewhere.

Theorem 12. (models of the tensor product)
Let T, T' be theories in partial membership equational logic. Then we have the following isomorphisms of categories:

$$\mathbf{PAlg}_{T\otimes T'} \simeq \mathbf{PAlg}_T(\mathbf{PAlg}_{T'}) \simeq \mathbf{PAlg}_{T'}(\mathbf{PAlg}_T).$$

□

A useful property of the tensor product of theories that we will make use of without proof is its *functoriality* in the category of theories. Therefore, if $H : T_1 \to T_2$ and $G : T_1' \to T_2'$ are theory morphisms, we have an associated theory morphism:

$$H \otimes G : T_1 \otimes T_1' \to T_2 \otimes T_2'.$$

It can be shown that the tensor product of theories is associative and commutative up to isomorphism, that is, that we have natural isomorphisms of theories $T \otimes T' \simeq T' \otimes T$ and $T \otimes (T' \otimes T'') \simeq (T \otimes T') \otimes T''$ giving a symmetric monoidal category structure to the category of theories.

Example 4. (double category)
A *double category* has been defined [14] as a category structure on **Cat**, the category of categories, that is, an object of $\mathbf{PAlg}_{CAT}(\mathbf{PAlg}_{CAT}) = \mathbf{DCat}$. The theory $CAT \otimes CAT$ then axiomatizes double categories in partial membership equational logic.
Spelling out the specification of $T \otimes T'$ for the case of $T = T' = CAT$ we get the following poset of sorts, where $Square$ is the top:

$$
\begin{array}{ll}
(Object, Object) = Object, & (Arrow, Arrow) = Square, \\
(Arrow, Object) = Harrow, & (Object, Arrow) = Varrow, \\
Object \leq Harrow \leq Square, & Object \leq Varrow \leq Square.
\end{array}
$$

For the operations in $\Omega \otimes \Omega'$ we adopt the following N-E-W-S notation:

$$d^l = w,\ c^l = e,\ d^r = n,\ c^r = s,\ (\_;\_)^l = \_ * \_,\ (\_;\_)^r = \_ \cdot \_.$$

The presentation of double categories in Maude-like notation is thus as follows.

```
fth DCAT is
  sorts Object Harrow Varrow Square .
  subsorts Object < Harrow Varrow < Square .
  ops n(_) e(_) w(_) s(_) _*_ _·_ .
  vars f h : Harrow .
  vars u v : Varrow .
  vars A B C D : Square .
  *** Inherited Axioms: Horizontal
  mbs w(A) , e(A) : Varrow .
  eq w(v) = v .
  eq e(v) = v .
  cmb A*B : Square iff e(A) = w(B) .
  ceq w(A*B) = w(A) if e(A) = w(B) .
  ceq e(A*B) = e(B) if e(A) = w(B) .
  ceq v*A = A if w(A) = v .
  ceq A*v = v if e(A) = v .
  ceq (A*B)*C = A*(B*C) if e(A) = w(B) and e(B) = w(C) .
  *** Inherited Axioms: Vertical
  mbs n(A) , s(A) : Harrow .
  eq n(h) = h .
  eq s(h) = h .
  cmb A·B : Square iff s(A) = n(B) .
  ceq n(A·B) = n(A) if s(A) = n(B) .
  ceq s(A·B) = s(B) if s(A) = n(B) .
  ceq h·A = A if n(A) = h .
  ceq A·h = h if s(A) = h .
  ceq (A·B)·C = A·(B·C) if s(A) = n(B) and s(B) = n(C) .
  *** Subalgebra Axioms
  cmb A : Object if A : Harrow and A : Varrow .
  mbs w(h) , e(h) , n(v) , s(v) : Object .
  mb f*h : Harrow .
  mb u·v : Varrow .
  *** Homomorphism Axioms
  eq n(w(A)) = w(n(A)) .
  eq n(e(A)) = e(n(A)) .
  eq s(w(A)) = w(s(A)) .
  eq s(e(A)) = e(s(A)) .
  ceq w(A·B) = w(A)·w(B) if s(A) = n(B) .
  ceq e(A·B) = e(A)·e(B) if s(A) = n(B) .
  ceq n(A*B) = n(A)*n(B) if e(A) = w(B) .
  ceq s(A*B) = s(A)*s(B) if e(A) = w(B) .
  ceq (A*B)·(C*D) = (A·C)*(B·D) if e(A) = w(B)
      and e(C) = w(D) and s(A) = n(C) and s(B) = n(D) .
endfth
```

Notice that in the above axiomatization we do not present the literal instances of the axioms, but equivalent forms. For example, we get `w(h) : Object` from $w(h) : Varrow$ (by inherited axioms), plus $w(h) : Harrow$ (by the subalgebra axiom properly speaking), plus the subalgebra axiom forcing $Harrow \cap Varrow = Object$. $\square$

In the following, we enrich our Maude-like notation with the tensor product construction. The presentation of double categories thus becomes much simpler:

```
fth DCAT is CAT ⊗ CAT renamed by (
  sorts (Object,Object) to Object . (Arrow,Arrow) to Square .
  sorts (Arrow,Object) to Harrow . (Object,Arrow) to Varrow .
  ops d left to w . c left to e . d right to n . c right to s .
  ops _;_ left to _*_ . _;_ right to _·_ )
endfth
```

3 Relating 2-Categories and Double Categories

We define 2-categories and 2VH-categories as models of suitable *PMEqtl* theories. We then enrich such structures to their monoidal version by tensoring with the theory of monoids, define computads and study the relatively free constructions associated to these categories of models.

3.1 2-Categories and 2VH-Categories

We now consider 2-categories [22]. They are probably the best known kind of enriched category. In particular, they yield models of rewriting logic in a very natural way [27]. Here, however, they can be considered as special cases of double categories, where vertical arrows are identified with objects. In 2-categories, squares are called cells, and horizontal arrows are called arrows. Moreover, north and south source and target are called d and c, while west and east source and target are l and r. Also, horizontal composition is denoted $\_;\_$ and vertical composition is denoted $\_ \circ \_$. We can therefore specify 2-categories as follows:

```
fth 2CAT is including DCAT renamed by (
  sort Square to Cell  Harrow to Arrow  Varrow to Object .
  ops w to l . e to r . n to d . s to c .
  ops _*_ to _;_ . _·_ to _∘_ ) .
endfth
```

We now introduce an extended version of 2-categories, called 2VH-categories, able to include in an appropriate sense the structure of double categories. In the following Maude-like presentation, the theory 2CAT is imported as such, without any renaming. In addition, new sorts *Harrow*, *Varrow* and *Square* are introduced, which correspond to the homonymous sorts of double categories. The poset of sorts is shown in Figure 2.

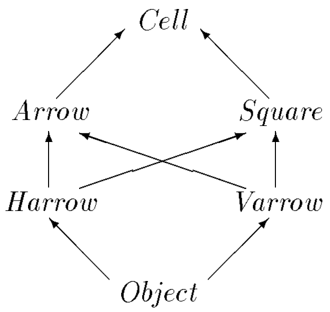

Fig. 2. The sort poset for 2VH-categories.

The basic intuition is that, if we are given a 2-category with subcategories *Harrow* and *Varrow* of *Arrow* such that they are disjoint except for objects, and such that the horizontal and vertical components can be recovered from their composition, then we can form a double category by considering squares with horizontal and vertical sides, and we can define their horizontal and vertical composition by using the already existing cell composition of the 2-category.

```
fth 2VHCAT is including 2CAT
  sorts Harrow Varrow Square .
  subsorts Object < Harrow Varrow < Arrow Square < Cell .
  ops n(_) e(_) w(_) s(_) _*_ _·_ .
  vars f h : Harrow .
  vars u v : Varrow .
  var t : Arrow .
  vars p q : Square .
  vars A B : Cell .
  cmb t : Object  if  t : Harrow and t : Varrow .
  cmb n(A) : Cell iff A : Square and n(A) : Harrow .
  cmb e(A) : Cell iff A : Square and e(A) : Varrow .
  cmb w(A) : Cell iff A : Square and w(A) : Varrow .
  cmb s(A) : Cell iff A : Square and s(A) : Harrow .
  ceq n(q) = h if d(q) = h;v .
  ceq e(q) = v if d(q) = h;v .
  ceq w(q) = v if c(q) = v;h .
  ceq s(q) = h if c(q) = v;h .
  eq d(q) = n(q);e(q) .
  eq c(q) = w(q);s(q) .
  cmb f;h : Harrow  iff r(f) = l(h) .
  cmb u;v : Varrow  iff r(u) = l(v) .
  cmb p*q : Square iff e(p) = w(q) .
  cmb A*B : Cell iff A , B : Square and e(A) = w(B) .
  cmb p·q : Square iff s(p) = n(q) .
  cmb A·B : Cell iff A , B : Square and s(A) = n(B) .
```

```
  ceq p*q = (n(p);q)∘(p;s(q))  if  e(p) = w(q) .
  ceq p·q = (p;e(q))∘(w(p);q)  if  s(p) = n(q) .
endfth
```

We comment on the above presentation. The basic idea is to represent a square q as a special cell where the domain arrow is the composition of the north horizontal arrow and the east vertical arrow. Similarly, the codomain arrow of q is the composition of the west vertical arrow and the south horizontal arrow. This is the meaning of equations $n(q) = h \Leftarrow d(q) = h; v$ and $e(q) = v \Leftarrow d(q) = h; v$, and of the equation $d(q) = n(q); e(q)$. There are also three analogous equations for $c(q)$.

To make this idea precise it is necessary to make both horizontal arrows and vertical arrows into subcategories of the category of arrows of the 2-category. This is expressed by the subsort ordering $Harrow, Varrow < Arrow$ and by clauses $f; g : Harrow \Leftrightarrow r(f) = l(g)$ and $u; v : Varrow \Leftrightarrow r(u) = l(v)$. Furthermore, the clause $t : Object \Leftarrow t : Harrow \ \wedge \ t : Varrow$ states that horizontal arrows and vertical arrows are disjoint except for objects.

Notice that the operations $\_ * \_$ and $\_ \cdot \_$ of horizontal and vertical square composition are defined only on adjacent squares (and not on other cells) according to clauses $A * B : Cell \Leftrightarrow A, B : Square \ \wedge \ e(A) = w(B) \ \wedge \ A * B : Square$. Similarly, N-E-W-S operations are defined only on squares according to the clause $n(A) : Cell \Leftrightarrow A : Square \ \wedge \ n(A) = Harrow$, and the other three analogous clauses.

Finally, the operation of horizontal composition of squares is defined in terms of horizontal and vertical composition of cells according to equation $p * q = (n(p); q) \circ (p; s(q)) \Leftarrow e(p) = w(q)$. Similarly for vertical composition of squares.

We then have a natural theory morphism from DCAT to 2VHCAT.

Proposition 13. *Let V be the signature morphism from DCAT to 2VHCAT relating homonymous sorts and operators. Then V is a theory morphism.* □

The proof is not difficult, and corresponds to showing that all the axioms of double categories can be proved from the axioms of 2VH-categories. The theory morphism above can be trivially specified in Maude-like notation.

```
view V from DCAT to 2VHCAT is
endview
```

3.2 Monoidal Models and Computads

In this section we add a monoidal structure to the models previously presented. This is quite important for making our approach expressive, since the monoidal operation represents parallel composition. The extension is almost effortless thanks to the tensor product of theories introduced in Section 2.2. In fact it is enough to introduce a theory MON of monoids and then to apply the tensor construction. We directly show all our extensions in Maude-like notation. We also introduce computads, which are an impoverished version of monoidal double categories, able to model tile rewrite systems.

The theory of monoids is as follows, where the monoidal operation is $\_ \otimes \_$.

```
fth MON is
  sort Monoid .
  ops 1 _⊗_ .
  vars a b c : Monoid .
  mb a⊗b : Monoid .
  eq a⊗1 = a .
  eq 1⊗a = a .
  eq a⊗(b⊗c) = (a⊗b)⊗c .
endfth
```

The theory of (strict) monoidal categories can be specified as follows:

```
fth MONCAT is MON ⊗ CAT renamed by (
  sorts (Monoid,Object) to Object . (Monoid,Arrow) to Arrow .
  ops 1 left to 1 . ⊗ left to ⊗ .
  ops d right to d . c right to c . ; right to ; )
endfth
```

Notice that we overload the symbol $\otimes$ using it for both the tensor construction on theories and for the monoidal operation within a theory.

In the above construction we have two subalgebra axioms:

$$\mathbf{1} : Object, \quad a \otimes b : Object$$

while homomorphism axioms are the well-known axioms of monoidal product[8]:

$$d(f \otimes g) = d(f) \otimes d(g),\ c(f \otimes g) = c(f) \otimes c(g),$$
$$f_1 \otimes g_1 \ ;\ f_2 \otimes g_2 = (f_1; f_2) \otimes (g_1; g_2).$$

We can now derive the theory of monoidal double categories.

```
fth MONDCAT is MON ⊗ DCAT renamed by (
  sorts (Monoid,Object) to Object . (Monoid,Square) to Square .
  sorts (Monoid,Harrow) to Harrow . (Monoid,Varrow) to Varrow .
  ops 1 left to 1 . ⊗ left to ⊗ .
  ops n right to n . e right to e . s right to s .
  ops w right to w . * right to *  . · right to · )
endfth
```

[8] Homomorphism axioms between the $\mathbf{1}$ operation of the monoid and the operations of the category:

$$d(\mathbf{1}) = \mathbf{1}, \quad c(\mathbf{1}) = \mathbf{1}, \quad \mathbf{1}\,;\mathbf{1} = \mathbf{1}$$

are already subsumed by other existing axioms.

Subalgebra axioms and homomorphism axioms are as follows:

$$\begin{gathered}\mathbf{1}\colon Object,\ a\otimes b\colon Object,\ f\otimes g\colon Harrow,\ u\otimes v\colon Varrow,\\ n(A\otimes B)=n(A)\otimes n(B),\ e(A\otimes B)=e(A)\otimes e(B),\\ w(A\otimes B)=w(A)\otimes w(B),\ s(A\otimes B)=s(A)\otimes s(B),\\ A_1\otimes B_1\ ;\ A_2\otimes B_2=(A_1;A_2)\otimes(B_1;B_2),\\ A_1\otimes B_1\ *\ A_2\otimes B_2=(A_1*A_2)\otimes(B_1*B_2).\end{gathered}$$

Notice that we could have obtained the same theory MONDCAT by applying the tensor product construction to MONCAT and CAT, or to CAT and MONCAT (but not to MONCAT and MONCAT!). We call $\mathbf{MonDCat}=\mathbf{PAlg}_{MONDCAT}$ the category of monoidal double categories.

We now introduce the monoidal version of 2VH-categories and of the theory morphism V.

```
fth MON2VHCAT is MON ⊗ 2VHCAT renamed by (
  sorts (Monoid,Object) to Object . (Monoid,Arrow) to Arrow .
  sorts (Monoid,Cell) to Cell . (Monoid,Harrow) to Harrow .
  sorts (Monoid,Varrow) to Varrow . (Monoid,Square) to Square .
  ops 1 left to 1 . ⊗ left to ⊗ .
  ops d right to d . c right to c .
  ops l right to l . r right to r . ; right to ; .
  ops ∘ right to ∘ . n right to n . e right to e .
  ops w right to w . s right to s .
  ops * right to * . · right to · )
endfth
```

We call $\mathbf{Mon2VHCat}=\mathbf{PAlg}_{MON2VHCAT}$ the category of monoidal 2VH-categories.

```
view W from MONDCAT to MON2VHCAT is id ⊗ V .
endview
```

We now present the main result of the paper.

Theorem 14. *The theory morphism W from MONDCAT to MON2VHCAT is:*

(i) *complete;*
(ii) *persistent w.r.t. sorts Object, Harrow and Varrow.*

□

We will be interested in the following property:

Definition 15. *(uniform monoidal double category)*
A monoidal double category $\mathbf{D}$ is *uniform* if the monoidal 2VH-category $F_W(\mathbf{D})$ satisfies the following additional conditional membership axiom:

$$\forall(A\colon Cell,\ \ f,h\colon Harrow,\ \ u,v\colon Varrow)\quad A\colon Square\ \Leftarrow\ d(A)=f;v\ \wedge\ c(A)=u;h.$$

□

Whenever this axiom is satisfied, it is easy to check whether a cell is a square, because it is enough to check whether the condition in the right member of the clause is satisfied.

We now give, as usual in Maude-like form, the presentation of *computads*, which (when provided with suitable generators) correspond to tile rewrite systems. Computads are reminiscent of monoidal double categories in that they also have a monoidal category of horizontal arrows and a monoidal category of vertical arrows sharing the objects. However, instead of squares they have *rules*, for which no operation is defined, unless they happen to be horizontal or vertical arrows. For horizontal composition this is expressed by the rule:

$$A * B : \mathit{Rule} \Leftrightarrow A, B : \mathit{Harrow} \ \wedge \ e(A) = w(B) \ \wedge \ A * B : \mathit{Harrow}$$

which states that horizontal composition $A*B$ of two elements (of the top sort) is defined iff they are adjacent horizontal arrows, and in this case it is an horizontal arrow.

```
fth CTD is
  including MONCAT renamed by (
  sorts Object to Object . Arrow to Harrow .
  ops 1 to 1 . ⊗ to ⊗ .
  ops d to w . c to e . ; to * ) .
  including MONCAT renamed by (
  sorts Object to Object . Arrow to Varrow .
  ops 1 to 1 . ⊗ to ⊕ .
  ops d to n . c to s . ; to ∘ ) .
  sort Rule .
  subsorts Object < Harrow Varrow < Rule .
  vars a b : Object .
  vars f g : Harrow .
  vars u v : Varrow .
  vars A B : Rule .
  mb f⊗g: Harrow .
  mb u⊕v: Varrow .
  eq a⊗b = a⊕b .
  cmb A : Harrow if A⊗B : Rule .
  cmb B : Harrow if A⊗B : Rule .
  cmb A : Varrow if A⊕B : Rule .
  cmb B : Varrow if A⊕B : Rule .
  cmb f*g : Harrow iff e(f) = w(g) .
  cmb A*B : Rule iff A , B : Harrow and e(A) = w(B) .
  cmb u·v : Varrow iff s(u) = n(v) .
  cmb A·B : Rule iff A , B : Varrow and s(A) = n(B) .
  cmb A : Object if A : Harrow and A : Varrow .
  mbs w(h) , e(h) , n(v) , s(v) : Object .
```

```
  mbs w(A) , e(A) : Varrow .
  mbs n(A) , s(A) : Harrow .
  eq n(w(A)) = w(n(A)) .
  eq n(e(A)) = e(n(A)) .
  eq s(w(A)) = w(s(A)) .
  eq s(e(A)) = e(s(A)) .
endfth
```

Notice that the structure of monoidal categories is imported for both horizontal and vertical arrows. *A priori* the two monoidal operators $\otimes$ and $\oplus$ on vertical and horizontal arrows are different (but they coincide on objects). However, they will be both mapped to the same operator when defining the theory morphism from computads to monoidal double categories.

We call $\mathbf{Ctd} = \mathbf{PAlg}_{CTD}$ the category of computads. Computads are essentially an impoverished version of monoidal double categories and can be used to present a monoidal double category as the free extension of a given computad. Therefore one can easily show the existence of a theory morphism from CTD to MONDCAT.

Proposition 16. (morphism from CTD to MONDCAT)

Let S be the signature morphism from CTD to MONDCAT mapping sort Rule to sort Square, operator $\oplus$ to operator $\otimes$ and for the rest relating homonymous sorts and operators. Then S is a theory morphism. □

The above property can be easily proved, since the axioms which must be satisfied by A are the same which must be satisfied by $U_S(A)$. The only exception is when a square $\alpha \in A_{Square}$ is not a horizontal arrow. Then, as an element of $U_S(A)_{Rule}$, horizontal composition is not defined on α, and thus no related axioms must be satisfied. Similarly, if $\alpha \in A_{Square}$ is not a vertical arrow, in $U_S(A)_{Rule}$, vertical composition is not defined on it. Finally, if $\alpha \in A_{Square}$ is neither a horizontal nor a vertical arrow, in $U_S(A)_{Rule}$, no monoidal operation is defined on it.

The signature morphism above can be specified in Maude-like notation.

```
view S from CTD to MONDCAT is
  sort Rule to Square .
  op ⊕ to ⊗ .
endview
```

We can now compose W and S to obtain a morphism Z from CTD to MON2VHCAT and apply the result of Proposition 6 in a couple of interesting cases.

```
view Z from CTD to MON2VHCAT is S;W .
endview
```

Proposition 17. (adjunctions from **Ctd** to **MonDCat** and to **Mon2VHCat**)
The forgetful functor U_S : **MonDCat** $\to$ **Ctd** *has a left adjoint* F_S : **Ctd** $\to$ **MonDCat**. *Similarly, the forgetful functor* U_Z : **Mon2VHCat** $\to$ **Ctd** *has a left adjoint* F_Z : **Ctd** $\to$ **Mon2VHCat**. *Furthermore,* F_Z *can be obtained by composing* F_S *with the left adjoint* F_W *to the forgetful functor* U_W : **Mon2VHCat** $\to$ **MonDCat**. □

4 Tile Logic and Enriched Rewriting Logic

We now formally introduce (the monoidal version of) tile rewrite systems and tile logic.

Definition 18. *(many-sorted hyper-signature)*
Given a set S of sorts, a *(many-sorted, hyper) signature* is an $S^* \times S^*$-indexed family of sets $\Sigma = \{\Sigma_{v,w}\}_{(v,w)\in S^*\times S^*}$, where S^* denotes the free monoid on set S. Each $f \in \Sigma_{v,w}$ is denoted $f : v \to w$. □

Definition 19. *(strict monoidal category freely generated by a signature)*
Given a signature Σ, $\mathbf{M}(\Sigma)$ is the strict monoidal category freely generated by Σ. □

Notice that the above well known free construction can be easily seen to be the left adjoint functor construction associated to a theory morphism. It is enough to define an (ordinary, total, algebraic) theory MONGRPH of *monoidal graphs* consisting of nodes (equipped with a monoidal operation) and of hyper-arcs (equipped with source and target operations), and to consider the theory morphism G: MONGRPH $\to$ MONCAT mapping nodes into objects and hyper-arcs into arrows; then $\mathbf{M} = F_G$. Since Σ is a model of MONGRPH, $\mathbf{M}(\Sigma)$ is then a model of MONCAT, i.e., a strict monoidal category.

Definition 20. *(tile rewrite system, tile logic)*
Let Σ_h and Σ_v be two disjoint signatures, called the *horizontal* and the *vertical* signature respectively. They must share the same set of sorts S.

A *tile rewrite system* $\mathcal{R} = C(S, \Sigma_h, \Sigma_v, R)$ is the computad where the set of objects is the free monoid on S, the horizontal (resp. vertical) arrows are the arrows of $\mathbf{M}(\Sigma_h)$ (resp. $\mathbf{M}(\Sigma_v)$), and the set of rules is R.

The *tile logic* of $\mathcal{R}$ is the monoidal double category $F_S(\mathcal{R})$ freely generated from $\mathcal{R}$ by the left adjoint functor F_S described by Proposition 17.

In the following, for α either a rule in $\mathcal{R}$ or a square $\alpha \in F_S(\mathcal{R})_{Square}$ we write $\alpha : f \xrightarrow[v]{u} g$ if $n(\alpha) = f$, $e(\alpha) = v$, $w(\alpha) = u$ and $s(\alpha) = g$, and we also write $\mathcal{R} \vdash_{tile} f \xrightarrow[v]{u} g$. □

As shown in [27], the Lawvere 2-theory $\mathcal{L}_{\mathcal{R}}$ of an unconditional rewrite theory $\mathcal{R}$ is the free 2-category with finite 2-products generated by $\mathcal{R}$ when viewed as an appropriate computad. Therefore, in 2-categorical terms rewriting logic is the logic associated to the models of the theory $P2CAT$ of 2-categories with finite 2-products. Regarding the cartesian 2-product $\_\times\_$ as a monoidal operation $\_\otimes\_$ exactly corresponds to defining a theory morphism $Y : MON2CAT \to P2CAT$, where $MON2CAT$ is the tensor product $MON \otimes 2CAT$. The forgetful functor $U_Y : \mathbf{P2Cat} \to \mathbf{Mon2Cat}$ has then a left adjoint F_Y, which allows us to consider the notion of a *monoidal rewrite theory* as an appropriate computad $\mathcal{R}$ that has first a free extension to a monoidal Lawvere theory $\mathcal{L}_{\mathcal{R}}^{\otimes} \in \mathbf{Mon2Cat}$, and whose standard 2-Lawvere theory is $\mathcal{L}_{\mathcal{R}} = F_Y(\mathcal{L}_{\mathcal{R}}^{\otimes})$.

The desired link between tile logic and rewriting logic can be obtained by considering monoidal rewrite theories that generate monoidal Lawvere theories not just in **Mon2Cat**, but in the richer category **Mon2VHCat**. We shall call such rewrite theories *enriched rewrite theories*. As computads they coincide with tile rewrite systems $\mathcal{R}$, that is, they can be regarded as computads $\mathcal{R} \in \mathbf{Ctd}$. Their logic is then captured by their monoidal 2VH Lawvere theory, namely $F_Z(\mathcal{R}) \in \mathbf{Mon2VHCat}$.

Definition 21. *(enriched rewriting logic)*
Given a tile rewrite system $\mathcal{R} = C(S, \Sigma_h, \Sigma_v, R)$, the *enriched rewriting logic* (*ERWL*) of $\mathcal{R}$ is the monoidal 2VH-category $F_Z(\mathcal{R})$ freely generated from $\mathcal{R}$ by the left adjoint functor F_Z described by Proposition 17.
We write $\mathcal{R} \vdash_{ERWL} f \xrightarrow[v]{u} g$ iff there is a square $\alpha : f \xrightarrow[v]{u} g \;\; \in F_Z(\mathcal{R})_{Square}$.
□

Tile logic and enriched rewriting logic are then systematically related by the left adjoint F_W to the forgetful functor $U_W : \mathbf{Mon2VHCat} \to \mathbf{MonDCat}$, that for any tile rewrite system $\mathcal{R}$ gives us a unit map

$$\eta : \; F_S(\mathcal{R}) \to U_W(F_Z(\mathcal{R}))$$

mapping tile logic into enriched rewriting logic, since (by Proposition 17) we have $F_Z(\mathcal{R}) = F_W(F_S(\mathcal{R}))$. Since by Theorem 14 we know that η is surjective on squares and bijective on arrows, we then obtain the following adequacy property for this mapping.

Corollary 22. (adequacy of enriched rewriting logic)
Given a tile rewrite system $\mathcal{R} = C(S, \Sigma_h, \Sigma_v, R)$ *we have*

$$\mathcal{R} \vdash_{tile} f \xrightarrow[v]{u} g \iff \mathcal{R} \vdash_{\mathrm{ERWL}} f \xrightarrow[v]{u} g$$

□

The practical importance of this result is that we can use an implementation of rewriting logic such as those currently available (Cafe [18], ELAN [3], Maude [10]) to perform deductions in tile logic. Notice that the above equivalence depends crucially on the rewrite proof α being a *square* $\alpha : f; v \to u; h$, since in $F_Z(\mathcal{R})$ we can have many rewrite proofs α that are not squares and therefore are *not* valid tile proofs. The constraint that the rewrite proof α must be a square can be imposed and implemented by means of the notion of an *internal strategy* [11] that restricts the rewrite proofs by requiring the overall proof to be an appropriate composition of squares yielding a square. Of course, if the tile rewrite system is uniform in the sense of the following definition, then the strategy imposing the constraint that proofs are squares can be defined much more simply, since one only needs to check the sides of the overall proof.

Definition 23. *(uniform tile rewrite system)*
A tile rewrite system $\mathcal{R}$ is *uniform* if its associated monoidal double category $F_S(\mathcal{R})$ is uniform (see Definition 15). □

As an interesting example of a uniform system, we now present CCS, Milner's Calculus for Communicating Systems [34]. For the tile presentation, we follow [21].

Example 5. (tile and rewriting logic for CCS)

Syntax of CCS. Let Δ be the alphabet for basic actions (which is ranged over by α) and $\overline{\Delta}$ the alphabet of complementary actions ($\Delta = \overline{\overline{\Delta}}$ and $\Delta \cap \overline{\Delta} = \emptyset$); the set $\Lambda = \Delta \cup \overline{\Delta}$ will be ranged over by λ. Let $\tau \notin \Lambda$ be a distinguished action, and let $\Lambda \cup \{\tau\}$ (ranged over by μ) be the set of CCS actions.

The syntax of finite CCS agents is defined by the following grammar:

$$P ::= \quad nil \quad \Big| \quad \mu.P \quad \Big| \quad P\backslash\alpha \quad \Big| \quad P + P \quad \Big| \quad P \mid P.$$

Operational Semantics of CCS. Given a process P, its dynamic behaviour can be described by a suitable transition system, along the lines of the SOS approach, where the transition relation is freely generated from the following set of inference rules.

$$\frac{}{\mu.P \xrightarrow{\mu} P} \qquad \frac{P \xrightarrow{\mu} Q \;\; \mu \not\in \{\alpha, \overline{\alpha}\}}{P\backslash\alpha \xrightarrow{\mu} Q\backslash\alpha}$$
$$\frac{P \xrightarrow{\mu} Q}{P+R \xrightarrow{\mu} Q} \qquad \frac{P \xrightarrow{\mu} Q}{R+P \xrightarrow{\mu} Q}$$
$$\frac{P \xrightarrow{\mu} Q}{P|R \xrightarrow{\mu} Q|R} \qquad \frac{P \xrightarrow{\alpha} Q,\; P' \xrightarrow{\overline{\alpha}} Q'}{P|P' \xrightarrow{\tau} Q|Q'} \qquad \frac{P \xrightarrow{\mu} Q}{R|P \xrightarrow{\mu} R|Q}.$$

Given the transition

$$((a.nil + b.nil) \mid \overline{a}.nil)\backslash a \xrightarrow{\tau} (nil \mid nil)\backslash a,$$

its proof is as follows:

$$\frac{\dfrac{\dfrac{\overline{a.nil \xrightarrow{a} nil}}{a.nil + b.nil \xrightarrow{a} nil} \quad \dfrac{}{\overline{a}.nil \xrightarrow{\overline{a}} nil}}{(a.nil + b.nil) \mid \overline{a}nil \xrightarrow{\tau} nil \mid nil}}{((a.nil + b.nil) \mid \overline{a}nil)\backslash a \xrightarrow{\tau} (nil \mid nil)\backslash a}$$

We now present the tile rewrite system for CCS.

Signatures of the CCS Rewrite System. There is only one sort $\underline{1}$. The free monoid generated by it is represented by underlined natural numbers with $\underline{n} \otimes \underline{m} = \underline{m+n}$. For the horizontal signature the operators are: $nil \in (\Sigma_h)_{\underline{0},\underline{1}}$; $\mu.\_$ and $\_\backslash\alpha \in (\Sigma_h)_{\underline{1},\underline{1}}$; $\_+\_$ and $\_\mid\_ \in (\Sigma_h)_{\underline{2},\underline{1}}$; and $!(\_) \in (\Sigma_h)_{\underline{1},\underline{0}}$. The latter constructor, called *eraser*, is needed to discard the rejected alternative after a choice step. For the vertical signature the operators are $\underline{\mu}.\_ \in (\Sigma_v)_{\underline{1},\underline{1}}$.

Rules of the CCS Rewrite System.

$$\begin{array}{lll}
\texttt{Pref}_\mu : \ \mu \xrightarrow[\underline{\mu}]{\underline{1}} \underline{1} & \texttt{Res}_\mu : \ \backslash\alpha \xrightarrow[\underline{\mu}]{\underline{\mu}} \backslash\alpha & \text{for} \quad \mu \notin \{\alpha, \overline{\alpha}\} \\
\texttt{Suml}_\mu : \ + \xrightarrow[\underline{\mu}]{\underline{\mu}\otimes\underline{1}} \underline{1}\otimes ! & \texttt{Sumr}_\mu : \ + \xrightarrow[\underline{\mu}]{\underline{1}\otimes\underline{\mu}} !\otimes\underline{1} & \\
\texttt{Compl}_\mu : \mid \xrightarrow[\underline{\mu}]{\underline{\mu}\otimes\underline{1}} \mid & \texttt{Compr}_\mu : \mid \xrightarrow[\underline{\mu}]{\underline{1}\otimes\underline{\mu}} \mid & \texttt{Synch}_\lambda : \mid \xrightarrow[\underline{\tau}]{\underline{\lambda}\otimes\underline{\overline{\lambda}}} \mid .
\end{array}$$

We associate a name to every rule of the tile rewrite system in order to refer to them later. The rules closely correspond to the ordinary SOS rules. For instance rule $\texttt{Pref}_\mu$ states, as its SOS counterpart, that constructor μ can be deleted, i.e., it can be replaced by the identity $\underline{1}$. Furthermore, the trigger is also the identity, and thus the corresponding SOS rule is an axiom. Finally, the effect is $\underline{\mu}$ and this corresponds to the label of the transition in the SOS case. As another example, rule $\texttt{Suml}_\mu$ defines left choice. The initial configuration is the constructor $+ : \underline{2} \to \underline{1}$, while the final configuration is $\underline{1}\otimes! : \underline{2} \to \underline{1}$, which states that the first component is preserved and the second component is discarded. The trigger states that in the first component we must have an action $\underline{\mu}$ while no action (i.e., identity action) is required on the second component. Action $\underline{\mu}$ is then transferred to the effect. We call **CCS** the computad defined in this way.

The tile corresponding to the previous example is obtained as follows:

$$\alpha = (((nil * \texttt{Prefix}_{\mathtt{a}}) \otimes (nil * b) * \texttt{Suml}_{\mathtt{a}}) \otimes (nil * \texttt{Prefix}_{\overline{\mathtt{a}}})) * \texttt{Synch}_{\mathtt{a}} * \texttt{Res}_\tau .$$

CCS Rules and Tiles as Cells. According to Definition 21, we can now employ the functor F_Z to derive the enriched rewriting logic semantics of our CCS rewrite system. For this purpose, we first specify the cell version of our rewrite rules and then apply *ERWL* rewriting.

$$\begin{array}{ll}
\texttt{Pref}_\mu : \ \underline{\mu}[\mu(p)] \to p & \texttt{Res}_\mu : \ \underline{\mu}[p\backslash\alpha] \to \underline{\mu}[p]\backslash\alpha \quad \text{for } \mu \notin \{\alpha, \overline{\alpha}\} \\
\texttt{Suml}_\mu : \ \underline{\mu}[p+q] \to \underline{\mu}[p]\otimes!(q) & \texttt{Sumr}_\mu : \ \underline{\mu}[p+q] \to \underline{\mu}[q]\otimes!(p) \\
\texttt{Compl}_\mu : \ \underline{\mu}[p \mid q] \to \underline{\mu}[p] \mid q & \texttt{Compr}_\mu : \ \underline{\mu}[p \mid q] \to p \mid \underline{\mu}[q] \\
\texttt{Synch}_\lambda : \ \underline{\tau}[p \mid q] \to \underline{\lambda}[p] \mid \underline{\overline{\lambda}}[q]. &
\end{array}$$

We use standard term notation for left and right hand sides of rewriting rules, i.e., for the domain and codomain arrows of cells. This is possible since almost all our horizontal and vertical constructors are term constructors, i.e., return one value. The only exception is discharger, which returns no value. To accommodate for the discharger, we extend the ordinary term notation introducing the monoidal operation. Notice that the parallel composition $t\otimes!(t')$ still returns one value, and thus can be safely used in a subterm position. It is easy to see that, thanks to the homomorphism axioms of monoidal categories, the following properties hold, where t and t' are terms without variables:

$$t \otimes !(t') = !(t') \otimes t \qquad !(t) \otimes !(t') = !(t') \otimes !(t) \qquad t''(t \otimes !(t')) = t''(t) \otimes !(t').$$

To distinguish between horizontal and vertical arrows, when composed to form an arrow, we use square brackets. For instance in the left hand side $\underline{\tau}[p \mid q]$ of $\mathtt{Synch}_\lambda$, the horizontal part is inside the brackets and the vertical part is outside, while in the right hand side $\underline{\lambda}[p] \mid \underline{\overline{\lambda}}[q]$ the vertical part is inside the brackets and the horizontal part is outside.

We can now show the derivation in *ERWL* style for our running example.

$$\begin{array}{lll}
\underline{\tau}[((a.nil + b.nil) \mid \overline{a}.nil)\backslash a] & \rightarrow [\underline{\tau}[(a.nil + b.nil) \mid \overline{a}.nil]]\backslash a & \rightarrow \\
([\underline{a}[a.nil + b.nil]] \mid [\underline{\overline{a}}[\overline{a}.nil]])\backslash a & \rightarrow ([\underline{a}[a.nil]] \otimes !(b.nil) \mid [\underline{\overline{a}}[\overline{a}.nil]])\backslash a & \rightarrow \\
([\underline{a}[a.nil]] \otimes !(b.nil) \mid nil)\backslash a & \rightarrow (nil \otimes !(b.nil) \mid nil)\backslash a & = \\
(nil \mid nil)\backslash a \otimes !(b.nil). & &
\end{array}$$

□

It is natural to proceed top down in the proof, as in a goal-oriented evaluation of SOS inference rules. We thus start with the initial configuration and the expected action. Notice that the presence of more than one level of square brackets in all the intermediate terms of our *ERWL* derivation makes clear that all its sub-derivations are cells but not squares. Only the entire derivation is a square, and actually it is easy to see that it can be made equal, employing the axioms of MON2VHCAT, to the square α defined above.

In general it is not enough to check that the domain (resp. codomain) arrow of a cell is the composition of a horizontal and a vertical arrow (resp. a vertical and a horizontal arrow) to make sure that the cell is a square. The following well-known "four bricks" computad provides a conterexample.

Example 6. (four bricks counterexample)
Let S consist of the underlined natural numbers, and let all constructors, both horizontal and vertical, have $\underline{1}$ as source and target. The set of rules is as follows:

$$\alpha = f_1 \xrightarrow[v_1']{v_1} g_4' * h_1 \qquad \beta = g_2 \xrightarrow[u_2]{v_1' \cdot w_2} g_2' \qquad \epsilon = h_1 \xrightarrow[w_2]{w_4} h_3$$
$$\gamma = h_3 * g_2' \xrightarrow[v_3]{v_3'} f_3 \qquad \delta = g_4' \xrightarrow[w_4 \cdot v_3']{u_4} g_4.$$

It is easy to see that the cell

$$f_1; \beta; v_3 \quad \circ \quad \alpha; w_2; g_2', v_3 \quad \circ \quad v_1; g_4'; \epsilon; g_2'; v_3 \quad \circ \quad v_1; g_4'; w_4; \gamma \quad \circ \quad v_1; \delta; f_3$$

is not a square. In fact, even if domain and codomain arrows are of the right form, this cell cannot be obtained from the rules just employing horizontal and vertical compositions $*$ and $\cdot$.

□

However, counterexamples as shown above cannot be found for the CCS rewrite system because of the following property:

Proposition 24. *The tile rewrite system* **CCS** *of Example 5 is uniform.*

The above result guarantees that a very simple implementation exists of the tile logic for CCS into rewriting logic. In fact there is no need to keep the proof terms while rewriting, since to show that an *ERWL* derivation corresponds to a tile derivation it is enough to check the domain and codomain arrows of the resulting cell.

5 Conclusion

In this paper we have employed partial membership equational logic to define (monoidal) double categories, 2-categories and their generalization, 2VH-categories. The aim was to build a tight connection between two very general models of computation, tile logic and rewriting logic. Tile logic is especially useful for defining compositional models of computation of reactive systems, coordination languages, mobile calculi, and causal and located concurrent systems. Rewriting logic extends the algebraic semantics approach to concurrent systems with state changes and is the basis of several existing languages. In particular, the language Maude developed at SRI is efficiently implemented and is equipped with important extra features, like execution strategies and reflective logic, which allow it to easily embed a variety of other logics. We have shown that rewriting logic derivation (of squares) within 2VH-categories coincides with tile logic derivation within double categories, when starting from the same computad. Thus if an implementation of 2VH-categories within Maude is provided using internal strategies, the models of computation specified via tile rewrite sistems can be actually run in Maude. Also, the notion of uniform tile rewrite system was introduced, for which the internal strategy is particularly simple, and a tile rewrite system for CCS was presented which is actually uniform.

Several lines of future research are possible and promising at this point. One consists of actually building in Maude the internal strategies above and of carrying out experiments about the flexibility and efficiency of execution in Maude of tile specifications. In particular, it will be very important to define specific formats of tile rewrite systems which guarantee uniformity. We are confident that most process algebras will fall in these classes. Results in this direction would make it easy to experiment with Maude about new calculi and about verification techniques for reactive systems. Finally, the definition of more refined theories can be tackled for particular versions of the tile model. For example, most of the applications described in the introduction require a version of tiles (and thus of double categories) which is *symmetric* monoidal rather than simply monoidal, and some of them require *cartesian* double categories. Definitions in the general case (where the structure should appear on both dimensions) are far from trivial [2], since natural transformations on double categories are defined in terms of four functors from a double to a four-fold category. Upcoming work by Roberto Bruni and the authors [6] proposes suitable notions of symmetric monoidal and cartesian double categories, and shows that the relation between monoidal double categories and monoidal 2VH-categories exploited in this paper can be smoothly extended to the symmetric monoidal and cartesian case. Also, preliminary results show for instance the possibility of programming and executing in Maude the tile systems recently proposed in the literature [16] for located and mobile calculi.

6 Acknowledgments

We would like to thank Narciso Martí-Oliet and Roberto Bruni for their comments.

References

1. H.P. Barendregt, M.C.J.D. van Eekelrn, J.R.W. Glauert, J.R. Kennaway, M.J. Plasmeijer, M.R. Sleep, *Term Graph Reduction*, Proc. PARLE, Springer LNCS 259, 141–158, 1987.
2. A. Bastiani, C. Ehresmann *Multiple Functors I: Limits Relative to Double Categories*, Cahiers de Topologie ed Géométrie Différentielle **15** (3), 1974, pp. 545-621.
3. P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, and M. Vittek. ELAN: A logical framework based on computational systems. In *Proc.* 1^{st} *Intl. Workshop on Rewriting Logic and its Applications*, ENTCS, North Holland, 1996.
4. G. Boudol, I. Castellani, M. Hennessy, A. Kiehn, *Observing Localities*, Theoretical Computer Science, 114: 31–61, 1993.
5. Adel Bouhoula, Jean-Pierre Jouannaud, and José Meseguer. Specification and proof in membership equational logic. In M. Bidoit and M. Dauchet, editors, *Proceedings of TAPSOFT'97*. Springer LNCS 1214, 1997.
6. R. Bruni, J. Meseguer, and U. Montanari. *Process and Term Tile Logic.* Technical Report, SRI International, in preparation.
7. R. Bruni, U. Montanari, *Zero-Safe Nets, or Transition Synchronization Made Simple*. In: Catuscia Palamidessi, Joachim Parrow, Eds, EXPRESS'97, ENTCS, Vol. 7, 1997.
8. R. Bruni, U. Montanari, *Zero-Safe Nets: The Individual Token Approach.* In: Francesco Parisi-Presicce, Ed., Proc. 12th WADT Workshop on Algebraic Development Techniques, Spinger LNCS, 1998, this volume.
9. P. Ciancarini, C. Hankin, Eds., *Coordination Languages and Models*, LNCS 1061, 1996.
10. Manuel G. Clavel, Steven Eker, Patrick Lincoln, and José Meseguer. Principles of Maude. In: J. Meseguer, Guest Ed., First International Workshop on Rewriting Logic and its Applications, ENTCS 4 (1996).
11. M. Clavel, J. Meseguer, *Internal Strategies in a Reflective Logic.* In: J. Meseguer, Guest Ed., First International Workshop on Rewriting Logic and its Applications, ENTCS 4 (1996).
12. A. Corradini, F. Gadducci, *A 2-Categorical Presentation of Term Graph Rewriting.* In: Eugenio Moggi, Giuseppe Rosolini, Eds., Category Theory and Computer Science 1997, Springer LNCS 1290, 1997, pp.87-105.

13. A. Corradini, U. Montanari, *An Algebraic Semantics for Structured Transition Systems and its Application to Logic Programs*, Theoretical Computer Science **103**, 1992, pp.51-106.
14. C. Ehresmann, *Catégories Structurées*: I and II, Ann. Éc. Norm. Sup. 80, Paris (1963), 349-426; III, Topo. et Géo. diff. V, Paris (1963).
15. G. Ferrari, U. Montanari, *A Tile-Based Coordination View of the Asynchronous π-calculus.* In: Igor Prvara, Peter Ruzicka, Eds., Mathematical Foundations of Computer Science 1997, Springer LNCS 1295, 1997, pp. 52-70.
16. G. Ferrari, U. Montanari, *Tiles for Concurrent and Located Calculi.* In: Catuscia Palamidessi, Joachim Parrow, Eds, EXPRESS'97, ENTCS, Vol. 7.
17. P. Freyd. Algebra valued functors in general and tensor products in particular. *Coll. Math.*, 14:89–106, 1966.
18. K. Futatsugi and T. Sawada. Cafe as an extensible specification environment. In *Proc. of the Kunming International CASE Symposium, Kunming, China, November*, 1994.
19. F. Gadducci, *On the Algebraic Approach to Concurrent Term Rewriting*, PhD Thesis, Università di Pisa, Pisa. Technical Report TD-96-02, Department of Computer Science, University of Pisa, 1996.
20. F. Gadducci, U. Montanari, *The Tile Model.* In: Gordon Plotkin, Colin Stirling, and Mads 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
21. F. Gadducci, U. Montanari, *Tiles, Rewriting Rules and CCS.* In: J. Meseguer, Guest Ed., 1st Int. Workshop on Rewriting Logic and its Applications, ENTCS 4 (1996).
22. G.M. Kelly, R.H. Street, *Review of the Elements of 2-categories*, Lecture Notes in Mathematics 420, 1974, pp. 75-103.
23. C. Lair, Etude Générale de la Catégorie des esquisses, Esquisses Math. 24 (1974).
24. K.G. Larsen, L. Xinxin, *Compositionality Through an Operational Semantics of Contexts*, in Proc. ICALP'90, LNCS 443, 1990, pp. 526-539.
25. F.W. Lawvere. Some algebraic problems in the context of functorial semantics of algebraic theories. In *Proc. Midwest Category Seminar II*, pages 41–61. Springer Lecture Notes in Mathematics No. 61, 1968.
26. N. Martí-Oliet, J. Meseguer, Inclusion and Subtypes I: First-order Case, *J. Logic Computat.*, Vol.6 No.3, pp.409-438, 1996.
27. J. Meseguer, *Rewriting as a Unified Model of Concurrency*, SRI Technical Report, CSL-93-02R, 1990. See the appendix on *Functorial Semantics of Rewrite Systems.*
28. J. Meseguer, *Conditional Rewriting Logic as a Unified Model of Concurrency*, Theoretical Computer Science **96**, 1992, pp. 73-155.
29. J. Meseguer. A logical theory of concurrent objects and its realization in the Maude language. In Gul Agha, Peter Wegner, and Akinori Yonezawa, Eds., *Research Directions in Concurrent Object-Oriented Programming*, pp. 314–390. MIT Press, 1993.
30. J. Meseguer. Membership algebra. Lecture and abstract at the Dagstuhl Seminar on "Specification and Semantics," July 9, 1996.
31. J. Meseguer, *Rewriting Logic as a Semantic Framework for Concurrency: A Progress Report*, in: U. Montanari and V. Sassone, Eds., *CONCUR'96: Concurrency Theory*, Springer LNCS 1119, 1996, 331-372.
32. J. Meseguer, Ed., *Procs. Rewriting Logic and Applications*, First International Workshop, ENTCS 4 (1996).

33. J. Meseguer, *Membership Equational Logic as a Logical Framework for Equational Specification.* In: Francesco Parisi-Presicce, Ed., Proc. 12th WADT Workshop on Algebraic Development Techniques, Spinger LNCS, 1998, this volume.
34. R. Milner, *Communication and Concurrency*, Prentice-Hall, 1989.
35. R. Milner, J. Parrow, D. Walker, *A Calculus of Mobile Processes* (parts I and II), Information and Computation, 100:1–77, 1992.
36. U. Montanari, F. Rossi, *Graph Rewriting and Constraint Solving for Modelling Distributed Systems with Synchronization*, in: Paolo Ciancarini and Chris Hankin, Eds., Coordination Languages and Models, LNCS 1061, 1996, pp. 12-27. Full paper submitted for publication.
37. B. Pareigis, *Categories and Functors*, Academic Press, 1970.
38. G. Plotkin, *A Structural Approach to Operational Semantics*, Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.
39. A. Poigné. Algebra categoricaly. In D. Pitt et al., editor, *Category Theory and Computer Programming*, volume 240 of *LNCS*, pages 76–102. Springer-Verlag, 1985.
40. P. Gabriel and F. Ulmer. *Lokal präsentierbare Kategorien.* Springer Lecture Notes in Mathematics No. 221, 1971.