UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

TECHNICAL REPORT: TR-02-09

# Functorial models for contextual pre-nets

Roberto Bruni[1] José Meseguer[2] Ugo Montanari[3]
Vladimiro Sassone[4]

June 25, 2002

ADDRESS: Corso Italia 40, 56125 Pisa, Italy. TEL: +39 050 2212700 FAX: +39 050 2212726

# Functorial models for contextual pre-nets ⋆

Roberto Bruni[1], José Meseguer[2], Ugo Montanari[3], and Vladimiro Sassone[4]

[1] Dipartimento di Informatica, Università di Pisa, Italia,
(also visiting scholar at the University of Illinois).
[2] University of Illinois at Urbana-Champaign, IL, USA.
[3] Dipartimento di Informatica, Università di Pisa, Italia.
[4] COGS, University of Sussex, Brighton, UK.
bruni@di.unipi.it, meseguer@cs.uiuc.edu, ugo@di.unipi.it, vs@susx.ac.uk

**Abstract.** The algebraic models of computation for contextual nets that have been proposed in the literature either rely on non-free monoid of objects, or introduce too many behaviors that must be somewhat filtered out. In this report, we exploit partial membership equational logic to define a suitable theory of models, where the meaningful concurrent computations can be selected by means of membership predicates.

## 1 Introduction

Thanks to their friendly formulation as multiset rewrite systems and to their graphical presentation, Petri nets [23,24] are an appealing formalism for the specification and study of concurrent and distributed systems: states consist of *token* distributions over the set of *places* and *transitions* can atomically fetch the tokens in their presets and generate new tokens according to their postsets. In particular, several transitions can execute concurrently when they work on mutually disjoint sets of tokens.

Nets with read arcs [29] (also introduced separately with different names, such as contextual nets [22], nets with test arcs [7], and nets with activator arcs [14]) encompass a non-destructive reading operation not present in the basic Petri net model. In fact, read arcs allow multiple concurrent readings of the same resource, an operation whose need arises naturally in many distributed systems, while the naïve encoding of read arcs as self-loops in ordinary Petri nets serializes all the accesses to read tokens with a dramatic loss of concurrency. Nets with read arcs have been used to model a variety of applications, such as transaction serializability in databases [25], concurrent constraint programming [21], asynchronous systems [28], and analysis of cryptographic protocols [9].

As a drawback, the presence of read arcs introduces some complication in the mathematical characterization of computations, leading to the study of suitable extensions of well-studied domains and models for Petri nets. Extensions of this kind include: the asymmetric event structures of [1], the match-share categories of [11], and the monoids of places proposed in [15] and fully developed in [6] and [20].

⋆ Research supported by the IST Project AGILE and by the Italian MIUR Project COMETA. The first author is also supported by an Italian CNR fellowship on Information Sciences and Technologies, and by the Computer Science Dept. of the Univ. of Illinois at Urbana-Champaign.

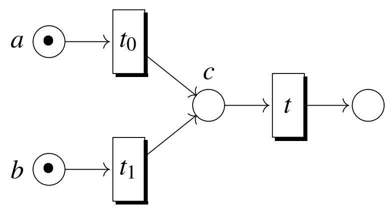
In this paper we extend the so-called "Petri nets are monoids" approach initiated in [17] to find a functorial characterization of the monoidal category of concurrent computations. In particular, we improve upon [11], where such computations were shown to be faithfully embedded in a large, freely generated category. Our approach is to define a typing discipline – expressed by membership predicates in partial membership equational logic [16] – that characterizes in that category the valid computations, distinguishing them from garbage expressions.

**Structure of the paper**

In Section 2 we summarize the techniques used for defining functorial models for Petri nets. Section 3 describes the technical problems arising when extending the approach to nets with read arcs, and Section 4 presents our solution. Section 5 gives our conclusions.

## 2 On the algebraic semantics of Petri nets

*Petri nets* are one of the most studied models for concurrency, thanks to their natural representation of concurrent and distributed systems based on multiset rewriting. Their flexibility has encouraged many different semantical interpretations. In particular, an overall distinction can be drawn between *collective* and *individual token philosophies* (see, e.g., [12]). According to the collective token philosophy (*CTph*), net semantics should not distinguish between different tokens in the same place, because any such token is *operationally equivalent* to all the others. The individual token philosophy (*ITph*) relies instead on the observation that the different origins and histories that each token has must be accounted for, because choosing different tokens can make an event causally dependent on different past events, and causal dependencies may influence the degree of concurrency in the computations. In the classical example below, for instance, after $t_0$ and $t_1$ have fired, a firing of $t$ will appear as caused by one of them, according to which of two tokens in $c$ is consumed. Also, two instances of $t$ may fire concurrently that only differ in their causal histories.




The "Petri nets are monoids" approach [17] is an algebraic approach to the analysis of concurrent net computations based on the observation that the monoidal structure of markings can be lifted to transitions (and then to computations), in such a way that the suitably axiomatized terms of the new algebra yield an initial model for the concurrent semantics of the nets according to the *CTph*. This construction respects the intuitive *simulation morphisms* between nets, when these are seen as graphs with structured nodes, as it can be expressed as a functor $\mathcal{T}$ from the category **Petri** of place/transitions (PT for short) nets (as objects) and simulation morphisms (as arrows) to the category

**CMonCat** of strictly symmetric strict monoidal categories (as objects) and symmetric monoidal functors (as arrows). Moreover, the functor $\mathcal{T}$ is the left adjoint to an obvious forgetful functor from the full subcategory of **CMonCat** consisting of categories whose set of objects is a free monoid.

Note that the *functorial* character of the construction is important for at least two reasons: (1) working within categories, we make explicit the associated *morphisms*, which correspond to appropriate notions of "simulation" or "refinement" between nets; (2) functors act on objects and behave consistently on their simulation maps, preserving them. Furthermore, when functors are *adjoints* they preserve limits or colimits, yielding good compositionality properties, since complex models can ofter be expressed as (co)limits of their simpler constituents [30].

Since the publication of [17], several studies have extended the functorial construction from the *CTph* towards the *ITph* [10,19,27]. Building on the notion of *process* presented in [13], the idea has been to take semantic models in the category of *symmetric monoidal categories*. But all the constructions proposed lacked functoriality. The difficulty in dealing with the *ITph* is that net morphisms in **Petri** allow replacing two different tokens $a$ and $b$ in the source net by, say, the same token $c$ in the target net. In this way, an ambiguity about the origin of $c$ is introduced that confuses causal histories in the target net and makes a functorial treatment impossible. A first solution was proposed in [27] based on pseudo functors (see also [19]).

In [4], we introduced *pre-nets*, which are more suitable than PT nets to be given a functorial semantics according to the *ITph*. A pre-net is essentially an implementation of a PT net, where the abstract data structure of multisets is refined into a more concrete *string* structure, and where each transition $t\colon u \to v$ is simulated by *one*, arbitrarily fixed, linear implementation $t_{\bar{u},\bar{v}}\colon \bar{u} \to \bar{v}$ for some linearizations $\bar{u}$ and $\bar{v}$ of $u$ and $v$.[1] Although resorting to pre-nets (instead of PT nets) might at first appear unnatural to net enthusiasts, our formal approach to the *ITph* benefits from several good properties:

- All the pre-net implementations of the same net share the same semantic model, i.e. the semantics is independent of the choice of linearizations.
- Algebraic models of pre-nets are freely generated and, as part of adjunctions, preserve colimit constructions, allowing a form of compositional reasoning.

In [4] it is shown that the construction can be conveniently expressed at the level of algebraic theories of the form $(\Sigma, E)$, rather than at the level of their categories of models, i.e. of $(\Sigma, E)$-algebras. Essentially, if `PETRI` is the theory of PT nets and `CMONCAT` is the theory of strictly symmetric monoidal categories, then there is a theory morphism form `PETRI` to `CMONCAT` that induces a forgetful functor between the category of `CMONCAT`-algebras (i.e., strictly symmetric monoidal categories) and the category of `PETRI`-algebras (i.e., PT nets). The left-adjoint to this forgetful functor is the free construction that associates to each PT net the strictly symmetric monoidal category of its concurrent computations. In such category, objects are the markings of the net, arrows

[1] We observe, lest confusion arises, that pre-nets differ sharply from phrase-structure grammars, because pre-nets do not distinguish between terminal and non-terminal symbols, and strings can be permuted before performing any step. Grammars only generate monoidal categories, with no symmetries.

are computations, (arrow) composition models progression in time of a computation, while tensor product accounts for concurrent activities. For instance, in the example above, $t_0;t$ represents the sequential execution of $t_0$ and $t$, while $t_0 \otimes t_1$ stands for the concurrent firing of $t_0$ and $t_1$. In the individual token philosophy, the strict symmetry – characteristic of the collective token interpretation – must be given up to model the causal flows of tokens in computations. The order of transitions in a parallel composition, say $t_0 \otimes t_1$, determines the order of tokens "in the output" and, consequently, the causal connections to the activities that may follow. For instance, $(t_0 \otimes t_1)\,;(t \otimes id_c)$ represents the computation where $t$ depends causally on $t_0$ (that is, it consumes the instance of $c$ generated by that transitions). We are allowed to exchange $t_0$ and $t_1$ in the tensor product only if we keep track of this and maintain the correct order of output tokens, as e.g. in $(t_1 \otimes t_0)\,;\gamma;\,(t \otimes id_c)$, for $\gamma$ the swap symmetry on $c \otimes c$. (A thorough discussion and the details are given, e.g., in [26], but see also [10,19].) As explained above, we can relate the theory `PRENETS` of *pre-nets* (where pre- and post-sets of transitions are taken in the free monoid of places instead than in the free commutative monoid) to the theory `SMONCAT` of symmetric monoidal categories (details in [4]).

The above-mentioned theories can be conveniently expressed in *partial membership equational logic* (**PMEqtl**, see [16,18] for self-contained presentations), taking advantage of membership predicates and subsorting to model objects as a special kind of arrows (the identities), and of partiality to model sequential composition, defined only if the codomain of the first arrow coincides with the domain of the second arrow. Moreover, the notion of tensor product of theories allows a more modular presentation of concepts; for example, we can define the theory of monoidal categories as the tensor product of the theory of monoids and that of categories.

## 3 Atoms, electrons and match-share categories

The extension of the approach to nets with read arcs has been considered in [5] (full version in [6]), by relying on non-free monoids of objects, and in [11], exploiting match-share categories in place of symmetric monoidal categories.

Regarding [5], the idea is to model each token $a$ as an *atom* that can emit "negative" particles $\bar{a}$ (*electrons*) while keeping track of their number, i.e., as suggested in [15], we have that for all $k \in \mathbb{N}$, $a = a^k \otimes \bigotimes_{i=1}^{k} \bar{a}$, where $a^k$ represents an atom that has released exactly $k$ particles to the environment. Then, by replacing context arcs on $a$ with self-loop arcs on $\bar{a}$, we obtain an axiomatic construction of the monoidal category of concurrent net computations. The approach of [5] deals satisfactorily with both the collective and the individual token philosophy; possibly, a remaining concern is that non-free monoids of objects sit uneasily with the traditional intuition of tokens as atomic pieces of data that one should not be able to decompose. The problem with the construction in [11] is instead that the freely generated model of computations has too many arrows, representing computations that contextual nets will never be able to perform.

In this paper we improve upon [11] by selecting suitable theories in partial membership equational logic in order to distinguish 'good' arrows – corresponding to computations – from meaningless ones.

```
ops d(_) c(_): Arrow -> Object.      *** domain and codomain
op  _⊗_: Arrow -> Object.            *** monoidal product
op  e : Object.                      *** unit of _⊗_
op  _;_ .                            *** Arrow composition (partial op.)
op  γ(_,_): Object Object -> Arrow.  *** symmetric natural transformation
```

**Fig. 1.** Operators in SMONCAT.

We refer the reader to the appendix of [4] for the essentials of partial membership equational logic. Instead, for the reader's convenience, we summarize in Appendix A the description of the theories of monoids, categories, monoidal categories and symmetric monoidal categories. Here we just remark that SMONCAT includes two sorts called Object and Arrow (with Object a subsort of Arrow, written Object < Arrow), and six operators (see Figure 1) satisfying the axioms of symmetric monoidal categories.

The idea presented in [11] is to model multiple concurrent readings by introducing in the class of net computations suitable transformations that take care of creating as many copies as needed (*sharing* phase) and then reassembling all copies after the reading (*matching* phase). These two transformations are called duplicators and co-duplicators and are denoted by $\nabla$ and $\Delta$ respectively. It is worth observing that they are "non-natural", in the technical sense that the naturality axioms $f;\nabla = \nabla; f\otimes f$ and $\Delta; f = f\otimes f;\Delta$ do not hold.

The theory of match-share categories is summarized in Figure 2. The right-hand side of the figure gives a pictorial representation of the main axioms of the left-hand side. The first group of axioms expresses the coherence of $\nabla$, and the second that of $\Delta$. The third group of axioms states how the two transformations interact together. If we look at $\nabla(\mathtt{a})$ as a wiring establishing two connections between the object $a$ in the domain and the occurrences of $a$ in the codomain, and dually for $\Delta(\mathtt{a})$, the last two axioms say that the multiplicity of connections is not important, and that the connections work in both directions, i.e. it is not important how objects are connected but just the fact that they are connected by an undirected path of wiring.

The theory of match-share categories is a conservative extension of the theory of symmetric monoidal categories and therefore the construction between (pre-)nets and symmetric monoidal categories can be straightforwardly extended to match-share categories. For modeling read arcs, the idea is to first view read arcs as self-loops (i.e. pairs of inbound and outbound arcs), so that a transition $t\colon u \xrightarrow{w} v$ from $u$ to $v$ in context $w$ is regarded as an ordinary pre-net transition $[t]\colon u\otimes w \longrightarrow v\otimes w$, and then apply the free construction to the resulting pre-net, building a match-share category of computations. The special role of $w$ – a "context" marking represented as an ordinary one – is dealt with by copying $\nabla$ and matching $\Delta$. This however generates arrows that do not represent admissible computations of the net. The construction is not resource-conscious, and the distinction between read arcs and pre/post-sets is lost, since each token can be matched and shared in all possible ways.

On the other hand, once we replace read arcs with self-loops, we can form the free symmetric monoidal category of computations of the pre-net. Such category distin-

```
fth MSCAT is
 including SMONCAT.
 ops ∇(_) Δ(_) : Object -> Arrow.

 vars a b : Object.

 eq  d(∇(a)) = a.
 eq  c(∇(a)) = a⊗a.
 eq  ∇(e) = e.
 eq  ∇(a⊗b) = (∇(a)⊗∇(b));(a⊗γ(a,b)⊗b).
 eq  ∇(a);(∇(a)⊗a) = ∇(a);(a⊗∇(a)).
 eq  ∇(a);γ(a,a) = ∇(a).

 eq  d(Δ(a)) = a⊗a.
 eq  c(Δ(a)) = a.
 eq  Δ(e) = e.
 eq  Δ(a⊗b) = (a⊗γ(b,a)⊗b);(Δ(a)⊗Δ(b)).
 eq  (Δ(a)⊗a);Δ(a) = (a⊗Δ(a));Δ(a).
 eq  γ(a,a);Δ(a) = Δ(a).

 eq  ∇(a);Δ(a) = a.
 eq  Δ(a);∇(a) = (a⊗∇(a));(Δ(a)⊗a).
endfth
```

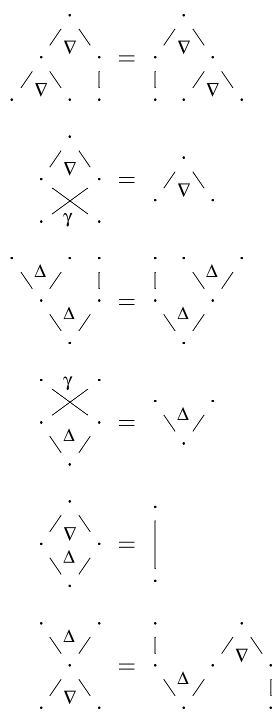


**Fig. 2.** Theory of match-share categories.

guishes arrows that represent the same concurrent computation, in that the construction enforces sequentialization of all multiple readings of the same resource. For instance, if $t\colon a \xrightarrow{c} b$, the fact that $t$ can fire two concurrent instances from $a \otimes a \otimes c$ will not be reflected. However, the monoidal and the match-share category can be combined via a mapping from the former to the latter that: (1) identifies all computations that are distinguished because of the order in which multiple readings are performed; and (2) selects only the admissible computations of the net with read arcs.

**Definition 1.** *Let $R$ be a pre-net with read arcs and let $[R]$ denote the pre-net with the same places as $R$ and transitions $\{[t]\colon u \otimes w \longrightarrow v \otimes w \mid t\colon u \xrightarrow{w} v \in R\}$. Let $\mathbf{S}([R])$ be the free symmetric monoidal category generated by $[R]$ and let $\mathbf{MS}([R])$ be the free match-share category generated by $[R]$. The symmetric monoidal functor $\mathcal{E}\colon \mathbf{S}([R]) \to \mathbf{MS}([R])$ is defined on generators by:*

$$\mathcal{E}(a) \stackrel{\text{def}}{=} a$$
$$\mathcal{E}([t]) \stackrel{\text{def}}{=} (u \otimes \nabla_w);([t] \otimes w);(v \otimes \Delta_w)$$

*for any place $a$ and transition $t\colon u \xrightarrow{w} v \in R$.*

```
fth RAUT is including MON.
 sort Rtrans.
 subsort Monoid < Rtrans.
 ops pre(_) post(_) ctx(_) : Rtrans -> Monoid.
 var u : Monoid.
 eq  pre(u) = e.
 eq  post(u) = e.
 eq  ctx(u) = u.
endfth.
```

**Fig. 3.** Theory of read-automata.

**Proposition 1 (cfr. [11]).** *The image $\mathcal{E}(\mathbf{S}([R])) \subseteq \mathbf{MS}([R])$ is isomorphic (via a symmetric monoidal functor) to the category of concatenable contextual processes of $R$.*

The question that then arises is how to tell whether an arrow of $\mathbf{MS}([R])$ belongs to $\mathcal{E}(\mathbf{S}([R]))$. We answer this by reformulating the construction at the level of theories in partial membership equational logic, thus expressing a typing discipline for discarding all meaningless arrows from $\mathbf{MS}([R])$ and keeping all the good ones.

## 4 Functorial models for pre-nets with read arcs

The first step is to define the theory of "programs," that is our base category of nets. It is technically convenient to consider a larger class of nets, whose states are element of a generic, non-free monoid, as expressed in Figure 3. The class of pre-nets with read arcs is then embedded as the full subclass whose states are free monoids (generated from the set of places), and the results can be extended via the obvious embedding.

The theory `RAUT` has three operations, `pre(_)`, `post(_)`, and `ctx(_)`, that define respectively source, target and (read) context of each read-transition in `Rtrans`. Idle transitions are included by the subsorting relation `Monoid` $<$ `Rtrans`. Sort `Monoid` comes from the theory `MON` of monoids, consisting of a total operation $\otimes$ which is associative and has the constant `e` as unit (see Figure 10 in Appendix A).

The second step is to refine the theory `MSCAT` by adding sorts and operators that are needed to characterize the class of meaningful arrows. Thus, we add two sorts `Rtrans` and `Rarrow`, with `Object` $<$ `Rtrans` $<$ `Rarrow` $<$ `Arrow`: the sort `Rtrans` is for embedding basic transitions, and the sort `Rarrow` is for collecting all correct computations. Among the operators, we add those of `RAUT` for source, target and context of basic transitions. Note that these operators, unlike those for domain and codomain, are not defined for all arrows, but only for the elements of `Rtrans`. Note also that they are related to the domain and codomain of transitions by the first two equations of the theory. The main ingredient is the operator `mk(_)`, which models the embedding $\mathcal{E}$ described above, namely $\mathtt{mk}(t) = [t]$, for any transition $t$.

The third step consists in expressing the adjunction between the class of programs and that of models. This task is accomplished by the trivial embedding morphism `RV` expressed in Figure 5.

```
fth RCOMP is including MSCAT.
 sorts Rtrans Rarrow.
 subsorts Object < Rtrans < Rarrow < Arrow.
 ops pre(_) post(_) ctx(_) : Rtrans -> Object.
 op  mk(_) : Rtrans -> Arrow.
 vars h k : Rarrow.
 var    t : Rtrans.
 var    u : Object.
 mb  h⊗k : Rarrow.
 mb  γ(u,v) : Rarrow.
 cmb h;k : Rarrow if c(h) = d(k).
 eq  pre(t)⊗ctx(t) = d(t).
 eq  post(t)⊗ctx(t) = c(t).
 eq  pre(u) = e.
 eq  post(u) = e.
 eq  ctx(u) = u.
 eq  d(mk(t)) = d(t).
 eq  c(mk(t)) = c(t).
 eq  mk(u) = u.
 eq  (pre(t)⊗∇(ctx(t)));(mk(t)⊗ctx(t));(post(t)⊗Δ(ctx(t))) = t.
endfth.
```

**Fig. 4.** Theory of read-computations.

```
view RV from RAUT to RCOMP is
  sort Monoid to Object.
endview.
```

**Fig. 5.** The view RV.

**Proposition 2.** *The view* RV *is a theory morphism.*

*Proof.* It is trivial to verify that all axioms in RAUT are respected by RV. For example, the sort Monoid is mapped to the sort Object, which has the same monoidality axioms (because the theory RCOMP includes the theory of match-share categories and hence also the axioms of monoidal categories). The remaining axioms fixing the behaviour of the three operations, pre(_), post(_), and ctx(_) on the elements of sort Monoid are present also in RCOMP for the sort Object, where Monoid is mapped. □

By Proposition 2 and because of the properties of theory morphisms [16], we know that there is a right-adjoint forgetful functor $\mathcal{U}_{\mathtt{RV}}$ from the category of RCOMP-algebras to the category of RAUT-algebras, which includes all pre-nets with read arcs. We denote by $\mathcal{F}_{\mathtt{RV}}$ the left-adjoint going in the opposite direction.

**Lemma 1.** *Given a pre-net with read arcs $R$, its initial* RCOMP*-algebra $\mathcal{F}_{\mathtt{RV}}(R)$ is a match-share category.*

*Proof.* The free functor $\mathcal{F}_{\mathtt{RV}}$ guarantees that the elements of sort `Arrow` of $\mathcal{F}_{\mathtt{RV}}(R)$ are built by composing objects, transitions $t \in R$, symmetries, duplicators and co-duplicators, together with the additional elements $\mathtt{mk}(t)$ for any $t \in R$. The axioms of match-share categories are enforced on all the elements of `Arrow` by inclusion of the theory `MSCAT` into `RCOMP`. ☐

The fourth and final step is to show that the sort `Rarrow` can be used to characterize all meaningful computations of $R$.

We recall that a lluf subcategory **A** of a category **C** is just a subcategory having all the objects of **C**.

**Definition 2.** *Given a pre-net with read arcs R, we let* $\mathtt{Rarrow}(R)$ *denote the lluf subcategory of the match-share category* $\mathcal{F}_{\mathtt{RV}}(R)$ *whose arrows have sort* `Rarrow`.

**Lemma 2.** *For any pre-net with read arcs R, an element t has sort* `Rtrans` *in* $\mathcal{F}_{\mathtt{RV}}(R)$ *if and only if t is a transition of R or t is a string of places.*

*Proof.* The only if part follows trivially from the definition of the theory morphism `RV`. The if part can be proved by noting that $\mathcal{F}_{\mathtt{RV}}(R)$ is the initial model, and hence the only membership predicates that can enforce $t : \mathtt{Rtrans}$ are those derived from subsorting ($\mathtt{Object} < \mathtt{Rtrans}$). ☐

**Lemma 3.** *The category* $\mathtt{Rarrow}(R)$ *is symmetric monoidal.*

*Proof.* The fact that $\mathtt{Rarrow}(R)$ is a symmetric monoidal category trivially follows from the definition of the theory `RCOMP` in Figure 4: The subsorting guarantees that all objects are also arrows; the membership predicates state that the sort `Rarrow` is closed under sequential and parallel composition, and that all the symmetries have sort `Rarrow`; the axioms for symmetric monoidal categories are inherited from the theory `MSCAT`, where they hold for all elements of `Arrow`, which `Rarrow` is a subsort of. ☐

**Theorem 1.** *The category* $\mathbf{MS}([R])$ *is isomorphic (via a match-share functor* $\mathcal{S}$*) to* $\mathcal{F}_{\mathtt{RV}}(R)$.

*Proof.* As observed in the proof of Lemma 1, the match-share category $\mathcal{F}_{\mathtt{RV}}(R)$ is generated by composing $t$ and $\mathtt{mk}(t)$ (for any transition $t$) with identities, symmetries and (co-)duplicators in all possible ways. On the other hand, any expression of sort `Arrow` can be equivalently expressed as the parallel and sequential composition of just the $\mathtt{mk}(t)$'s with identities, symmetries and (co-)duplicators, because of the equation

```
eq  (pre(t)⊗∇(ctx(t)));(mk(t)⊗ctx(t));(post(t)⊗Δ(ctx(t))) = t.
```

that allows replacing all occurrences of $t$. Note that if $t = u$ for some object $u$, then $\mathtt{mk}(u) = u$. Hence the constructor $\mathtt{mk}(\_)$ cannot be applied to identities for generating new arrows. Moreover, no other axioms involving $t : \mathtt{Rtrans}$ are present that could further quotient out the elements of sort `Arrow`.

Let us consider the match-share functor $\mathcal{S} : \mathbf{MS}([R]) \to \mathcal{F}_{\mathtt{RV}}(R)$ sending $[t]$ to $\mathtt{mk}(t)$ (and being the identity otherwise) which is well-defined by initiality of $\mathbf{MS}([R])$. It can be easily checked that the functor $\mathcal{S}$ is full and faithful, it preserves symmetries and (co-)duplicators, and it defines an isomorphism on objects (and thus on arrows). ☐

**Theorem 2.** *The category $\mathcal{E}(\mathbf{S}([R]))$ is isomorphic (via a symmetric monoidal functor $\mathcal{R}$) to* $\mathtt{Rarrow}(R)$.

*Proof.* The functor $\mathcal{R}$ is $\mathcal{S}$ restricted to $\mathcal{E}(\mathbf{S}([R]))$. In fact, suppose that $\alpha \in \mathcal{E}(\mathbf{S}([R]))$, then an arrow $\beta \in \mathbf{S}([R])$ must exist such that $\mathcal{E}(\beta) = \alpha$. Let $\mathcal{Q}\colon \mathbf{S}([R]) \to \mathtt{Rarrow}(R)$ be the symmetric monoidal functor sending $[t]$ to $t$ and preserving identities, symmetries, sequential composition and monoidal composition. Then it is straightforward that $\mathcal{S}(\alpha) = \mathcal{Q}(\beta)$ and hence $\mathcal{S}(\alpha)$ has sort $\mathtt{Rarrow}$. The functor $\mathcal{R}$ is an isomorphism because it is injective on the generators (the transitions of the net) and preserves the operations of symmetric monoidal categories strictly. $\square$

Note that the categories $\mathcal{E}(\mathbf{S}([R]))$ and $\mathtt{Rarrow}(R)$ are not match-share categories, and hence the functor $\mathcal{R}$ is not a match-share functor.

Theorem 2 defines a typing discipline for selecting the admissible computations from the larger class $\mathbf{MS}([R])$. Since, under appropriate assumptions [2], membership predicates allow automated verification in languages like Maude [8], the construction RV answers to the ambiguity of $\mathcal{E}$.

Note that for the arrows in $\mathtt{Rarrow}(R)$ only the operations of domain and codomain are defined, not those involved with contexts. However, the properties of the initial model can be exploited to factor out the domain and codomain of arrows in $\mathtt{Rarrow}(R)$ into their consumed, read and produced parts. We show this below.

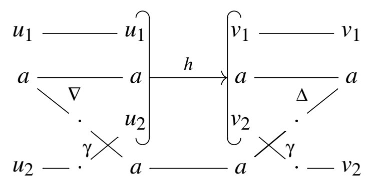



**Fig. 6.** A read object $a$ for the arrow $h$.

**Definition 3.** *Let $h \in \mathtt{Rarrow}(R)$ and let $a$ be an object with $\mathtt{d}(h) = u_1 \otimes a \otimes u_2$ and $\mathtt{c}(h) = v_1 \otimes a \otimes v_2$ for suitable objects $u_1$, $u_2$, $v_1$, $v_2$. The object $a$ is said* to be read in $h$ *if $h$ can be written as (cf. Figure 6)*

$$\Big(u_1 \otimes \nabla(a) \otimes u_2\Big); \Big(u_1 \otimes a \otimes \gamma(a,u_2)\Big); \Big(h \otimes a\Big); \Big(v_1 \otimes a \otimes \gamma(v_2,a)\Big); \Big(v_1 \otimes \Delta(a) \otimes v_2\Big).$$

**Lemma 4.** *Let $t : \mathtt{Rtrans}$. Then, $\mathtt{ctx}(t)$ is read in $h$.*

*Proof.* Since $t : \mathtt{Rtrans}$, then

$$t = \Big(\mathtt{pre}(t) \otimes \nabla(\mathtt{ctx}(t))\Big); \Big(\mathtt{mk}(t) \otimes \mathtt{ctx}(t)\Big); \Big(\mathtt{post}(t) \otimes \Delta(\mathtt{ctx}(t))\Big)$$

and therefore, by substituting $t$, the expression

$$\Big(\mathtt{pre}(t) \otimes \nabla(\mathtt{ctx}(t))\Big); \Big(t \otimes \mathtt{ctx}(t)\Big); \Big(\mathtt{post}(t) \otimes \Delta(\mathtt{ctx}(t))\Big)$$

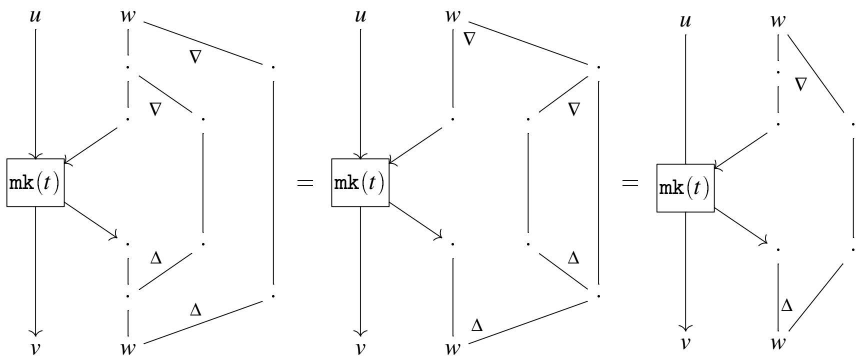



**Fig. 7.** The proof of Lemma 4, graphically.

becomes

$$\Big(\mathtt{pre}(t)\otimes\nabla(\mathtt{ctx}(t))\Big);$$
$$\Big(\Big(\Big(\mathtt{pre}(t)\otimes\nabla(\mathtt{ctx}(t))\Big);\Big(\mathtt{mk}(t)\otimes\mathtt{ctx}(t)\Big);\Big(\mathtt{post}(t)\otimes\Delta(\mathtt{ctx}(t))\Big)\Big)\otimes\mathtt{ctx}(t)\Big);$$
$$\Big(\mathtt{post}(t)\otimes\Delta(\mathtt{ctx}(t))\Big).$$

By the functoriality of $\otimes$, we can rearrange the above expression as

$$\Big(\mathtt{pre}(t)\otimes\Big(\nabla(\mathtt{ctx}(t));\Big(\nabla(\mathtt{ctx}(t))\otimes\mathtt{ctx}(t)\Big)\Big)\Big);$$
$$\Big(\mathtt{mk}(t)\otimes\mathtt{ctx}(t)\otimes\mathtt{ctx}(t)\Big);$$
$$\Big(\mathtt{post}(t)\otimes\Big(\Big(\Delta(\mathtt{ctx}(t))\otimes\mathtt{ctx}(t)\Big);\Delta(\mathtt{ctx}(t))\Big)\Big).$$

Then, by coherence of $\nabla$ and $\Delta$ we have:

$$\Big(\mathtt{pre}(t)\otimes\Big(\nabla(\mathtt{ctx}(t));\Big(\mathtt{ctx}(t)\otimes\nabla(\mathtt{ctx}(t))\Big)\Big)\Big);$$
$$\Big(\mathtt{mk}(t)\otimes\mathtt{ctx}(t)\otimes\mathtt{ctx}(t)\Big);$$
$$\Big(\mathtt{post}(t)\otimes\Big(\Big(\mathtt{ctx}(t)\otimes\Delta(\mathtt{ctx}(t))\Big);\Delta(\mathtt{ctx}(t))\Big)\Big).$$

Again, by functoriality of $\otimes$, we get:

$$\Big(\mathtt{pre}(t)\otimes\nabla(\mathtt{ctx}(t))\Big);$$
$$\Big(\mathtt{mk}(t)\otimes\Big(\nabla(\mathtt{ctx}(t));\Delta(\mathtt{ctx}(t))\Big)\Big);$$
$$\Big(\mathtt{post}(t)\otimes\Delta(\mathtt{ctx}(t))\Big).$$

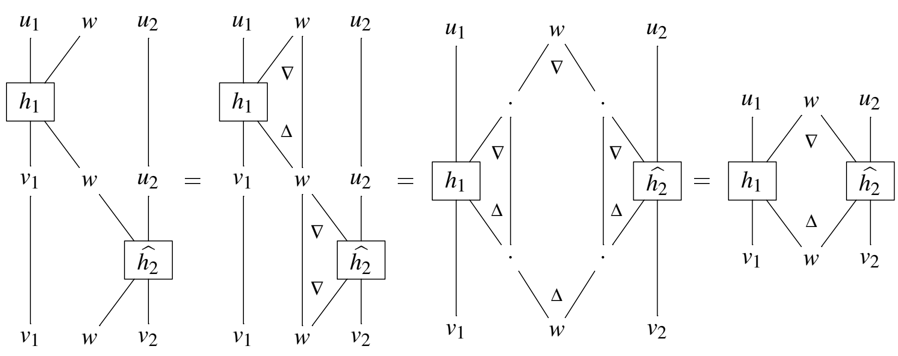



**Fig. 8.** The proof of Proposition 3, graphically.

Finally, by the interaction of $\nabla$ and $\Delta$ in match-share categories, we have:

$$\Big(\mathtt{pre}(t)\otimes\nabla(\mathtt{ctx}(t))\Big);\Big(\mathtt{mk}(t)\otimes\mathtt{ctx}(t)\Big);\Big(\mathtt{post}(t)\otimes\Delta(\mathtt{ctx}(t))\Big)$$

which is of course equal to $t$. The whole proof is graphically illustrated in Figure 7, where for simplicity we let $u = \mathtt{pre}(t)$, $v = \mathtt{post}(t)$ and $w = \mathtt{ctx}(t)$. □

The marking read – and not consumed – by $h$ is the maximum marking read by $h$, and it can be characterized as follows.

**Definition 4.** *Let $h \in \mathtt{Rarrow}(R)$. The arrow $h$ is* pure *if $\mathtt{d}(h) = u\otimes w$ and $\mathtt{c}(h) = v\otimes w$, with $(u\otimes\nabla(w));(h\otimes w);(v\otimes\Delta(w)) = h$ and no other object in $u$ and $v$ is read. The object $w$ is called the* context *of $h$ and denoted by $\mathtt{ctx}(h)$, while $u$ and $v$ are denoted respectively by $\mathtt{pre}(h)$ and $\mathtt{post}(h)$. We denote by $\widehat{h}$ the* twisted *version of $h$ obtained by exchanging the position of the context with that of the pre- and post-set (respectively, in the domain and codomain of $h$), i.e. $\widehat{h} = \gamma(w,u);h;\gamma(v,w)$.*

The following result follows easily from Lemma 4.

**Corollary 1.** *Any arrow $h \in \mathtt{Rtrans}(R)$ is pure.*

**Lemma 5.** *Let $h \in \mathtt{Rarrow}(R)$ be pure. Then $\widehat{h} \in \mathtt{Rarrow}(R)$.*

*Proof.* The assertion follows by $\mathtt{Rarrow}(R)$ being closed w.r.t. composition with symmetries.

**Lemma 6.** *Let $h \in \mathtt{Rarrow}(R)$ be pure, with $\mathtt{pre}(h) = u$, $\mathtt{post}(h) = v$ and $\mathtt{ctx}(h) = w$. Then $\widehat{h} = (\nabla(w)\otimes u);(w\otimes\widehat{h});(\Delta(w)\otimes v)$.*

The following result shows that computations which are serialized on contexts are equivalent to the concurrent executions with multiple readings of the context.

**Proposition 3.** *Let* $h_1, h_2 \in \mathtt{Rarrow}(R)$ *be pure arrows, with* $\mathtt{pre}(h_i) = u_i$, $\mathtt{post}(h_i) = v_i$ *and* $\mathtt{ctx}(h_i) = w$ *for* $i = 1,2$*. Then:*

$$\begin{aligned}(h_1 \otimes u_2);(v_1 \otimes \widehat{h_2}) &= (u_1 \otimes \nabla(w) \otimes u_2);(h_1 \otimes \widehat{h_2});(v_1 \otimes \nabla(w) \otimes v_2)\\ &= (u_1 \otimes \widehat{h_2});(h_1 \otimes v_2)\end{aligned}$$

*Proof.* The proof exploits Lemmas 4 and 6 and is (partially) illustrated in Figure 8:

– we first make explicit that the arrows $h_1$ and $\widehat{h_2}$ read the context $w$ by applying the laws:

$$\begin{aligned}h_1 &= (u_1 \otimes \nabla(w));(h_1 \otimes w);(v_1 \otimes \Delta(w))\\ \widehat{h_2} &= (\nabla(w) \otimes u_1);(w \otimes \widehat{h_2});(\Delta(w) \otimes v_1)\end{aligned}$$

that are valid for pure arrows;
– then, we apply the axioms of match-share categories to rearrange the matching and sharing of $w$ to have enough concurrent copies of it available at the same time and use functoriality of the tensor to shift $h_1$ and $\widehat{h_2}$ in parallel;
– finally, we get rid of additional copies by applying back the laws of pure arrows.

The equality with the computation where $\widehat{h_2}$ precedes $h_1$ can be proven in a completely analogous way and is thus omitted. □

## 5 Conclusion

Previous approaches to extending the "Petri nets are monoids" to nets with read arcs have either relied on structured tokens or have defined a too rich category of computations, where it was difficult to filter out meaningless arrows.

We have employed theories in partial membership equational logic to solve the latter problem. Specifically, we have introduced a suitable theory RCOMP that provides us with a typing discipline to select all and only the correct concurrent computations.

We conclude by mentioning that a non-initial match-share category of abstract models for nets with read arcs has been used in [3], based on categories of (co)spans in **Set**. However, the models in [3] do not retain all the information about the concurrent computations of the net: they just keep track of which (initial) resources have been read throughout the computation and thus can be concurrently accessed from the environment.

## References


1. P. Baldan, A. Corradini, and U. Montanari. Contextual Petri nets, asymmetric event structures, and processes. *Inform. and Comput.*, 171(1):1–49, 2001.
2. A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theoret. Comput. Sci.*, 236:35–132, 2000.

3. R. Bruni and F. Gadducci. Some algebraic laws for spans (and their connections with multirelations). In W. Kahl, D.L. Parnas, and G. Schmidt, editors, *Proceedings of RelMiS 2001, Relational Methods in Software*, volume 44.3 of *Elect. Notes in Th. Comput. Sci.* Elsevier Science, 2001. Also published as Technical Report, Bericht Nr. 2001-02, Fakultat fur Informatik, Universitat der Bundeswehr Munchen.
4. R. Bruni, J. Meseguer, U. Montanari, and V. Sassone. Functorial models for petri nets. *Inform. and Comput.*, 170(2):207–236, 2001.
5. R. Bruni and V. Sassone. Algebraic models for contextual nets. In U. Montanari, J.D.P. Rolim, and E. Welzl, editors, *Proceedings of ICALP 2000, 27th Int. Coll. on Automata, Languages and Programming*, volume 1853 of *Lect. Notes in Comput. Sci.*, pages 175–186. Springer Verlag, 2000.
6. R. Bruni and V. Sassone. Two algebraic process semantics for contextual nets. In H. Ehrig, G. Juhás, J. Padberg, and G. Rozenberg, editors, *Advances in Petri Nets: Unifying Petri Nets*, volume 2128 of *Lect. Notes in Comput. Sci.*, pages 427–456. Springer Verlag, 2001.
7. S. Christensen and N.D. Hansen. Coloured petri nets extended with place capacities, test arcs and inhibitor arcs. In M.A. Marsan, editor, *Proceedings of ICATPN'93, 14th Int. Conf. on Application and Theory of Petri Nets*, volume 691 of *Lect. Notes in Comput. Sci.*, pages 186–205. Springer Verlag, 1993.
8. M. Clavel, F. Durán, S. Eker, N. Martí-Oliet, J. Meseguer, and J. Quesada. *Maude: Specification and programming in rewriting logic*. SRI International, 1999. Available at `http://maude.csl.sri.com/manual`.
9. F. Crazzolara and G. Winskel. Events in security protocols. In *8th ACM conference on Computer and Communications Security, CCS '01 Proceedings*, pages 96–105, 2001.
10. P. Degano, J. Meseguer, and U. Montanari. Axiomatizing the algebra of net computations and processes. *Acta Inform.*, 33(7):641–667, 1996.
11. F. Gadducci and U. Montanari. Axioms for contextual net processes. In *Proceedings of ICALP'98, 25th International Colloquium on Automata, Languages and Programming*, volume 1443 of *Lect. Notes in Comput. Sci.*, pages 296–308. Springer Verlag, 1996.
12. R.J. van Glabbeek and G.D. Plotkin. Configuration structures. In D. Kozen, editor, *Proceedings of LICS'95, 10th Annual IEEE Symposium on Logics in Computer Science*, pages 199–209. IEEE Computer Society Press, 1995.
13. U. Goltz and W. Reisig. The non-sequential behaviour of Petri nets. *Inform. and Comput.*, 57:125–147, 1983.
14. R. Janicki and M. Koutny. Semantics of inhibitor nets. *Inform. and Comput.*, 123(1):1–16, 1995.
15. J. Meseguer. Rewriting logic as a semantic framework for concurrency: A progress report. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR'96, 7th International Conference on Concurrency Theory*, volume 1119 of *Lect. Notes in Comput. Sci.*, pages 331–372. Springer Verlag, 1996.
16. J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *Proceedings of WADT'97, 12th Workshop on Recent Trends in Algebraic Development Techniques*, volume 1376 of *Lect. Notes in Comput. Sci.*, pages 18–61. Springer Verlag, 1998.
17. J. Meseguer and U. Montanari. Petri nets are monoids. *Inform. and Comput.*, 88(2):105–155, 1990.
18. J. Meseguer and U. Montanari. Mapping tile logic into rewriting logic. In F. Parisi-Presicce, editor, *Proceedings of WADT'97, 12th Workshop on Recent Trends in Algebraic Development Techniques*, volume 1376 of *Lect. Notes in Comput. Sci.*, pages 62–91. Springer Verlag, 1998.
19. J. Meseguer, U. Montanari, and V. Sassone. Representation theorems for Petri nets. In Ch. Freksa, M. Jantzen, and R. Valk, editors, *Foundations of Computer Science: Potential -*

*Theory - Cognition, to Wilfried Brauer on the occasion of his sixtieth birthday*, volume 1337 of *Lect. Notes in Comput. Sci.*, pages 239–249. Springer Verlag, 1997.
20. J. Meseguer, P.C. Ölveczky, and M.-O. Stehr. Rewriting logic as a unifying framework for Petri nets. In H. Ehrig, G. Juhás, J. Padberg, and G. Rozenberg, editors, *Advances in Petri Nets: Unifying Petri Nets*, volume 2128 of *Lect. Notes in Comput. Sci.*, pages 250–303. Springer Verlag, 2001.
21. U. Montanari and F. Rossi. Contextual occurrence nets and concurrent constraint programming. In *Graph Transformations in Computer Science*, volume 776 of *Lect. Notes in Comput. Sci.*, pages 280–295. Springer Verlag, 1994.
22. U. Montanari and F. Rossi. Contextual nets. *Acta Inform.*, 32:545–596, 1995.
23. C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für Instrumentelle Mathematik, Bonn, 1962.
24. W. Reisig. *Petri Nets: An Introduction*. EATCS Monographs on Theoretical Computer Science. Springer Verlag, 1985.
25. G. Ristori. *Modelling Systems with Shared Resources via Petri Nets*. PhD thesis, Computer Science Department, University of Pisa, 1994.
26. V. Sassone. An axiomatization of the algebra of Petri net concatenable processes. *Theoret. Comput. Sci.*, 170(1-2):277–296, 1996.
27. V. Sassone. An axiomatization of the category of Petri net computations. *Math. Struct. in Comput. Sci.*, 8(2):117–151, 1998.
28. W. Vogler. Efficiency of asynchronous systems and read arcs in Petri nets. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Proceeding of ICALP'97, 24th International Colloquium on Automata, Languages and Programming*, volume 1256 of *Lect. Notes in Comput. Sci.*, pages 538–548. Springer Verlag, 1997.
29. W. Vogler. Partial order semantics and read arcs. In I. Prívara and P. Ruzicka, editors, *Proceeding of MFCS'97, 22nd International Symposium on Mathematical Foundations of Computer Science*, volume 1295 of *Lect. Notes in Comput. Sci.*, pages 508–517. Springer Verlag, 1997.
30. G. Winskel and M. Nielsen. Models for concurrency. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*. Oxford University Press, 1995.

## A Theories in partial membership equational logic

The theory of categories $\mathtt{CAT}$ is defined as follows. Its poset of sorts has sorts $\mathtt{Object}$ and $\mathtt{Arrow}$ with $\mathtt{Object} < \mathtt{Arrow}$. There are two unary total operations $\mathtt{d}(\_)$ and $\mathtt{c}(\_)$, for *domain* and *codomain*, and a binary composition operation $\_;\_$ defined if and only if the codomain of the first argument is equal to the domain of the second argument. The complete definition of the theory is given in Figure 9. By convention, functions with given domain and codomain are total on that domain and codomain. It is easy to check that a model (cf. [18] for the details) of $\mathtt{CAT}$ is exactly a category (in which objects coincide with identity arrows), and that $\mathtt{CAT}$-homomorphisms are just functors.

The theory $\mathtt{MON}$ of monoids is even simpler. It has a unique sort $\mathtt{Monoid}$ and two total operators: the associative tensor $\_ \otimes \_$ and the unit element $\mathtt{e}$, which behaves as the identity for $\_ \otimes \_$. The theory $\mathtt{MON}$ is defined in Figure 10.

Then, by exploiting the tensor product of theories $\_ \otimes \_$ defined in [18], the theory of monoidal categories can be obtained by combining the theories $\mathtt{MON}$ and $\mathtt{CAT}$ as illustrated in Figure 11.

```
fth CAT is
 sorts Object Arrow.
 subsort Object < Arrow.
 ops d(_) c(_) : Arrow -> Object.
 op  _;_.
 var      a : Object.
 vars f g h : Arrow.
 eq  d(a) = a.
 eq  c(a) = a.
 ceq a;f = f   if d(f) == a.
 ceq f;a = f   if c(f) == a.
 cmb f;g : Arrow   if c(f) == d(g).
 ceq c(f) = d(g)   if f;g : Arrow.
 ceq d(f;g) = d(f)   if c(f) == d(g).
 ceq c(f;g) = c(g)   if c(f) == d(g).
 ceq (f;g);h = f;(g;h)   if c(f) == d(g) and c(g) == d(h).
endfth
```

**Fig. 9.** The theory `CAT`.

```
fth MON is
 sort Monoid.
 op e : -> Monoid.
 op _⊗_ : Monoid Monoid -> Monoid [assoc id: e].
endfth
```

**Fig. 10.** The theory `MON`.

```
fth MONCAT is MON ⊗ CAT renamed by (
    sort (Monoid,Object) to Object.
    sort (Monoid,Arrow) to Arrow.
    op  e left to e.
    op _⊗_ left to _⊗_.
    op _;_ right to _;_.
    op d(_) right to d(_).
    op c(_) right to c(_). ).
endfth
```

**Fig. 11.** The theory `MONCAT`.

```
fth MON ⊗ CAT is
 sorts (Monoid,Object) (Monoid,Arrow).
 subsort (Monoid,Object) < (Monoid,Arrow).
 op e left : -> (Monoid,Arrow). *** (B) subalgebra axioms are implicit
 op _⊗_ left : (Monoid,Arrow) (Monoid,Arrow) -> (Monoid,Arrow).
 ops d(_) right c(_) right : (Monoid,Arrow) -> (Monoid,Object).
 op  _;_ right.
 var        a : (Monoid,Object).
 vars f g h k : (Monoid,Arrow).
 eq  (f⊗g left)⊗h left = f⊗(g⊗h left) left. *** (A) inherited axioms
 eq  f⊗(e left) left = f.
 eq  (e left)⊗f left = f.
 eq  d(a) right = a.
 eq  c(a) right = a.
 ceq a;f right = f if d(f) right == a.
 ceq f;a right = f if c(f) right == a.
 cmb f;g right : (Monoid,Arrow) if c(f) right == d(g) right.
 ceq c(f) right = d(g) right if f;g right : (Monoid,Arrow).
 ceq d(f;g right) right = d(f) right if c(f) right == d(g) right.
 ceq c(f;g right) right = c(g) right if c(f) right == d(g) right.
 ceq (f;g right);h right = f;(g;h right) right
     if c(f) right == d(g) right and c(g) right == d(h) right.
 eq  d(e left) right = e left. *** (C) homomorphism axioms
 eq  c(e left) right = e left.
 eq  (e left);(e left) right = e left.
 eq  d(f⊗g left) right = (d(f) right)⊗(d(g) right) left.
 eq  c(f⊗g left) right = (c(f) right)⊗(c(g) right) left.
 ceq (f;g right)⊗(h;k right) left = (f⊗h left);(g⊗k left) right
     if f;g right : (Monoid,Arrow) and h;k right : (Monoid,Arrow).
endfth
```

**Fig. 12.** Explicit definition of the theory MON ⊗ CAT.

```
fth SMONCAT is including MONCAT.
 op γ(_,_) : Object Object -> Arrow.
 vars a a' b b' c : Object.
 vars        f f' : Arrow.
 eq d(γ(a,b)) = a⊗b.
 eq c(γ(a,b)) = b⊗a.
 eq γ(a,e) = a.
 eq γ(e,a) = a.
 eq γ(a⊗b,c) = (a⊗γ(b,c));(γ(a,c)⊗b).
 eq γ(a,b);γ(b,a) = a⊗b.
 ceq (f⊗f');γ(b,b') = γ(a,a');(f'⊗f)
    if d(f) == a and d(f') == a' and c(f) == b and c(f') == b'.
endfth
```

**Fig. 13.** The theory SMONCAT.

Note that the tensor product construction $\mathtt{MON} \otimes \mathtt{CAT}$ has the sort poset originated from the product of the two sort posets in `MON` and `CAT` and operators "*opM* `left`" and "*opC* `right`" for each operator *opM* in `MON` and *opC* in `CAT`. The axioms of $\mathtt{MON} \otimes \mathtt{CAT}$ are generated by combining the axioms of `MON` and `CAT` and can be partitioned in three classes: (A) inherited axioms; (B) subalgebra axioms; (C) homomorphism axioms. The complete definition of the theory $\mathtt{MON} \otimes \mathtt{CAT}$ is in Figure 12. We omit discussing the details here. The theory `MONCAT` just renames sorts and operators by a more friendly notation.

Finally, the theory of symmetric monoidal categories `SMONCAT` is defined in Figure 13, by adding the symmetric natural transformation $\gamma(\_,\_)$.