Algebraic Theories for Contextual Pre-nets[⋆]

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

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

Abstract. The algebraic models of computation for contextual nets that have been proposed in the literature either rely on a non-free monoid of objects, or introduce too many fictitious behaviors that must be somewhat filtered out. In this paper, 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 [25, 26] 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.

Contextual nets [24] (also introduced separately with different names, such as nets with read arcs [30], nets with test arcs [8], and nets with activator arcs [16]) 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 and phenomena, such as transaction serializability in databases [11], concurrent constraint programming [23], asynchronous systems [29], and analysis of cryptographic protocols [10].

As a drawback, the presence of read arcs introduces some complication in the mathematical characterization of computations, leading to the development of suitable extensions of well-studied domains and models for Petri nets. Extensions of this kind include: the asymmetric event structures of [2], the match-share categories of [13], and the monoids of places proposed in [17] and fully developed in [7] and in [22].

[⋆] Research supported by the FET-GC Project IST-2001-32747 AGILE, by the MIUR Project COFIN 2001013518 CoMeta. The first author is also supported by an Italian CNR fellowship, and by the CS Department of the University of Illinois at Urbana-Champaign.

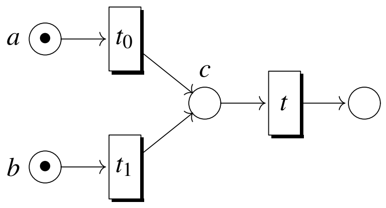
In this paper we extend the so-called "Petri nets are monoids" approach initiated in [19] to find a neat algebraic characterization of the monoidal category of concurrent computations in the presence of read arcs. In particular, we improve upon [13], where such computations were shown to be faithfully embedded in a too large, freely generated category. Our approach is to define a typing discipline – expressed by membership predicates *term* : `Sort` in partial membership equational logic [18] – that characterizes in that category the valid computations, distinguishing them from "garbage" expressions. Moreover, by considering pre-nets as "implementations" of ordinary Petri nets (in the sense explained in [5] and recalled in Section 2), we are able to give a *functorial* construction, respecting the simulation morphisms between nets, a result not achieved in all previous proposals in the literature [17, 13, 7].

Synopsis. 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. Proofs omitted for space limitation can be found in the technical report [6].

We assume the reader has some familiarity with some basic concepts from category theory as, e.g., the notion of natural transformation, adjunction and monoidal category.

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., [14]). 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*) says that the different origins and histories of tokens 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 look as caused by one of them and concurrent to the other, depending on 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 [19] is an algebraic approach to the analysis of concurrent semantics based on the observation that the monoidal structure of markings can be lifted to computations, in such a way that the suitably axiomatized terms of the new algebra exactly correspond to the concurrent computations of place/transitions

Petri nets (PT nets), according to the *CTph*. This construction respects the intuitive *simulation morphisms* between nets, when these are seen as graphs with structured nodes. This is expressed as a functor $\mathcal{T}$ from the category **Petri** of PT nets (as objects) and simulation morphisms (as arrows) to the category **CMonCat** of strictly symmetric strict monoidal categories (as objects) and monoidal functors (as arrows). Moreover, $\mathcal{T}$ 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.

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 often be expressed as (co)limits of their simpler constituents [31].

Since the publication of [19], several studies have extended the functorial construction from the *CTph* towards the *ITph* [12,21,28]. Building on the notion of *process* presented in [15], the idea has been to take semantic models in the category of *symmetric monoidal categories*. But all the proposed constructions 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 [28] based on pseudo functors (see also [21]).

In [5], 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 $t\colon u \longrightarrow v$ is simulated by *one*, arbitrarily fixed, linear implementation $t_{\bar{u},\bar{v}}\colon \bar{u} \longrightarrow \bar{v}$ 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 [5] it is shown that the construction can be conveniently expressed at the level of algebraic theories of the form (Σ, E), rather than at the level of their categories of models, i.e. of (Σ, 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

[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.

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 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 γ the swap symmetry on $c \otimes c$. (A thorough discussion and the details are given, e.g., in [27], but see also [12,21].) 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 [5]).

The above-mentioned theories can be conveniently expressed in *partial membership equational logic* (**PMEqtl**, see [18,20] 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 [7], by relying on non-free monoids of objects, and in [13], exploiting match-share categories in place of symmetric monoidal categories.

Regarding [7], 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 [17], 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 [7] 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 [13] is instead that the freely generated model of computations has too many arrows, representing spurious computations that contextual nets cannot perform.

In this paper we improve upon [13] 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 [5] 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 [13] 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 ∇ and Δ 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 $\Delta;f = f\otimes f;\Delta$ are not enforced.

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 ∇ (defining the domain and codomain of each component of ∇, stating that the unit e is trivially shared and that the component for $a\otimes b$ can be expressed in terms of the components for a and b, the last two axioms roughly establishing that sharing is associative and commutative), and the second group that of Δ. 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 connections are bidirectional, 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 ∇ and matching Δ. 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.

```
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
```

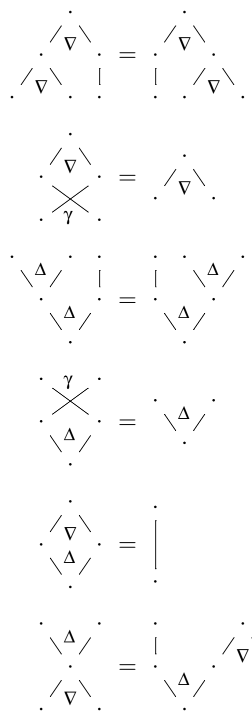


Fig. 2. Theory of match-share categories.

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 distinguishes 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.

Notation. Let R be a pre-net with read arcs. We denote by $[R]$ the pre-net with the same places as R and transitions $\{[t]\colon u \otimes w \longrightarrow v \otimes w \mid t\colon u \xrightarrow{w} v \in R\}$. Moreover, we let $\mathbf{S}([R])$ denote the free symmetric monoidal category generated by $[R]$ and let $\mathbf{MS}([R])$ denote the free match-share category generated by $[R]$.

Definition 1. *The symmetric monoidal functor $\mathcal{E}\colon \mathbf{S}([R]) \to \mathbf{MS}([R])$ is defined on generators by:*

$$\mathcal{E}(a) \overset{\text{def}}{=} a \qquad \textit{(for any place } a \in R\textit{)}$$

$$\mathcal{E}([t]) \overset{\text{def}}{=} (u \otimes \nabla_w);([t] \otimes w);(v \otimes \Delta_w) \qquad \textit{(for any transition } t\colon u \xrightarrow{w} v \in R\textit{).}$$

```
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. [13]). *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 $\mathbf{MS}([R])$, while 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 elements 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`. The 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 9 in Appendix A).

The second step is to refine the theory `MSCAT` into a theory `RCOMP` 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 (i.e., `pre(_)`, `post(_)`, and `ctx(_)`). Note that these operators, unlike those for domain and codomain (i.e., `d(_)` and `c(_)`), 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 membership axioms state that the sort `Rarrow` is closed under monoidal and sequential composition and that it contains all the symmetries. The main novel ingredient is the operator `mk(_)`, which models the embedding $\mathcal{E}$ described above, namely `mk`$(t) = [t]$, for any transition t, as expressed by the last equation of the theory. The presence of `mk(_)` is also technically convenient to prove the main correspondence results.

```
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.

The third step is to express the adjunction between the class of programs and that of models. This task is accomplished by the signature morphism RV in Figure 5, which embeds homonym sorts and operators and maps the sort Monoid of RAUT to the sort Object of RCOMP. It is easy to verify that all axioms in RAUT are respected by RV:

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

By Proposition 2 and because of the properties of theory morphisms [18], 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 $\mathcal{F}_{\mathtt{RV}}$ ensures that the elements of sort Arrow of $\mathcal{F}_{\mathtt{RV}}(R)$ are built by composing objects, transitions $t \in R$, symmetries 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. For the following definition, 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* `Rarrow`(R) *denote the lluf sub-category 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.*

Lemma 3. *The category* `Rarrow`(R) *is symmetric monoidal.*

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

Proof. 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. 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 : `Rtrans` are present that could further quotient out the elements of sort `Arrow`.

Let us consider the match-share functor $\mathcal{S}\colon \mathbf{MS}([R]) \to \mathcal{F}_{\mathtt{RV}}(R)$ sending $[t]$ to $\mathtt{mk}(t)$ (and being the identity otherwise) which is well-defined by initiality of $\mathbf{MS}([R])$. 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* `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 $\mathcal{Q}\colon \mathbf{S}([R]) \to$ `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 `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. □

Note that the categories $\mathcal{E}(\mathbf{S}([R]))$ and `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 [3], membership predicates allow automated verification in languages like Maude [9], then the construction RV answers to the ambiguity of $\mathcal{E}$.

Note that for the arrows in `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 `Rarrow`(R) into their consumed, read and produced parts. We show this below.

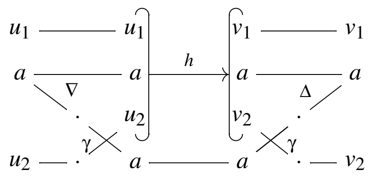


Fig. 6. A read object a for the arrow h.

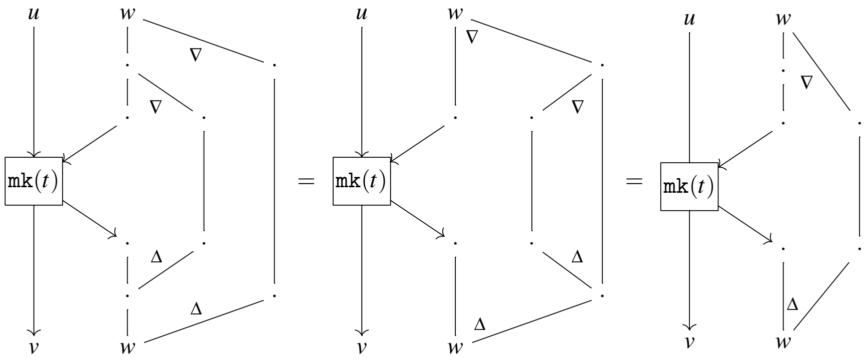


Fig. 7. The proof of Lemma 4, graphically.

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.*

The 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 $\mathtt{post}(h)$.*

For h pure, 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. $\widehat{h} = \gamma(w, u); h; \gamma(v, w)$.

Corollary 1 (From Lemma 4). *Any arrow $h \in \mathtt{Rtrans}(R)$ is pure.*

Lemma 5. *Let $h \in \mathtt{Rarrow}(R)$ be pure, with $\mathtt{pre}(h) = u$, $\mathtt{post}(h) = v$ and $\mathtt{ctx}(h) = w$. Then $\widehat{h} \in \mathtt{Rarrow}(R)$ and $\widehat{h} = (\nabla(w) \otimes u); (w \otimes \widehat{h}); (\Delta(w) \otimes v)$.*

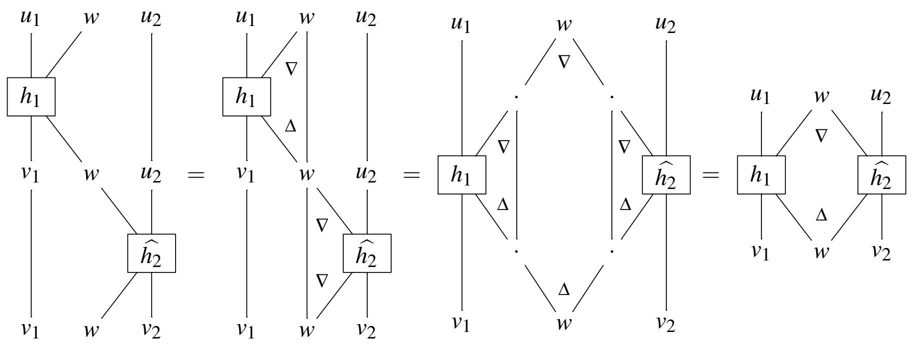


Fig. 8. The proof of Proposition 3, graphically.

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* $h_1, h_2 \in \texttt{Rarrow}(R)$ *be pure arrows, with* $\texttt{pre}(h_i) = u_i$, $\texttt{post}(h_i) = v_i$ *and* $\texttt{ctx}(h_i) = w$ *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 5 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 (valid for pure arrows):

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

– 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 expression where $\widehat{h_2}$ precedes h_1 is analogous. □

5 Conclusion

Previous approaches to extending the "Petri nets are monoids" semantics 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. The theory

RCOMP enucleates the fundamental algebraic principles on which the non-trivial operation of *reading without consuming* is based on. The functorial construction presented in this paper has been reconciled with unfolding semantics in [1]. Moreover, as equational reasoning in **PMEqtl** is supported by the rewriting logic language Maude [9], the theory RCOMP offers a mathematical basis for the analysis and optimization of concurrent computations in systems with many-readers access policies to shared resources (e.g., for the applications of contextual nets in [11, 23, 29, 10]).

We conclude by mentioning that a non-initial match-share category of abstract models for nets with read arcs has been used in [4], based on categories of (co)spans in **Set**. However, the models in [4] do not retain all the information about the concurrent computations of the net: they just keep track of which resources have been read throughout the computation and thus can be concurrently accessed from the environment.

Acknowledgment

We warmfully thank Paolo Baldan for many interesting discussions on the semantics of contextual nets. We thank the anonymous referees for their criticisms and suggestions that helped us in preparing the final version.

References

1. P. Baldan, R. Bruni, and U. Montanari. Pre-nets, read arcs and unfolding: a functorial presentation. *Proc. WADT 2002*, *LNCS*. Springer, 2003. To appear.
2. P. Baldan, A. Corradini, and U. Montanari. Contextual Petri nets, asymmetric event structures, and processes. *Inform. and Comput.*, 171(1):1–49, 2001.
3. A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theoret. Comput. Sci.*, 236:35–132, 2000.
4. R. Bruni and F. Gadducci. Some algebraic laws for spans (and their connections with multirelations). *Proc. RelMiS 2001*, *ENTCS* 44.3, Elsevier, 2001.
5. R. Bruni, J. Meseguer, U. Montanari, and V. Sassone. Functorial models for petri nets. *Inform. and Comput.*, 170(2):207–236, 2001.
6. R. Bruni, J. Meseguer, U. Montanari, and V. Sassone. Functorial models for contextual pre-nets. Technical Report TR-02-09, Computer Science Department, University of Pisa, 2002.
7. R. Bruni and V. Sassone. Two algebraic process semantics for contextual nets. *Advances in Petri Nets: Unifying Petri Nets*, *LNCS* 2128, pp. 427–456. Springer, 2001.
8. S. Christensen and N.D. Hansen. Coloured petri nets extended with place capacities, test arcs and inhibitor arcs. *Proc. ICATPN'93*, *LNCS* 691, pp. 186–205. Springer, 1993.
9. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. Maude: Specification and programming in rewriting logic. *Th. Comput. Sci.*, 285:187–243, 2002.
10. F. Crazzolara and G. Winskel. Events in security protocols. *Proc. CCS'01*, pp. 96–105. ACM, 2001.
11. N. De Francesco, U. Montanari, and G. Ristori. Modeling concurrent accesses to shared data via Petri nets. *Programming Concepts, Methods and Calculi*, IFIP Transactions A-56, pp. 403–422. North Holland, 1994.
12. P. Degano, J. Meseguer, and U. Montanari. Axiomatizing the algebra of net computations and processes. *Acta Inform.*, 33(7):641–667, 1996.

13. F. Gadducci and U. Montanari. Axioms for contextual net processes. *Proc. ICALP'98*, *LNCS* 1443, pp. 296–308. Springer, 1996.
14. R.J. van Glabbeek and G.D. Plotkin. Configuration structures. *Proc. LICS'95*, pp. 199–209. IEEE Computer Society Press, 1995.
15. U. Goltz and W. Reisig. The non-sequential behaviour of Petri nets. *Inform. and Comput.*, 57:125–147, 1983.
16. R. Janicki and M. Koutny. Semantics of inhibitor nets. *Inf. and Comput.*, 123(1):1–16, 1995.
17. J. Meseguer. Rewriting logic as a semantic framework for concurrency: A progress report. *Proc. CONCUR'96*, *LNCS* 1119, pp. 331–372. Springer, 1996.
18. J. Meseguer. Membership algebra as a logical framework for equational specification. *Proc. WADT'97*, *LNCS* 1376, pp. 18–61. Springer, 1998.
19. J. Meseguer and U. Montanari. Petri nets are monoids. *Inf. and Comp.* 88(2):105–155, 1990.
20. J. Meseguer and U. Montanari. Mapping tile logic into rewriting logic. *Proc. WADT'97*, *LNCS* 1376, pp. 62–91. Springer, 1998.
21. J. Meseguer, U. Montanari, and V. Sassone. Representation theorems for Petri nets. *Foundations of Computer Science: Potential - Theory - Cognition, to Wilfried Brauer on the occasion of his sixtieth birthday*, *LNCS* 1337, pp. 239–249. Springer, 1997.
22. J. Meseguer, P.C. Ölveczky, and M.-O. Stehr. Rewriting logic as a unifying framework for Petri nets. *Advances in Petri Nets: Unifying Petri Nets*, *LNCS* 2128, pp. 250–303. Springer, 2001.
23. U. Montanari and F. Rossi. Contextual occurrence nets and concurrent constraint programming. *Proc. Dagstuhl Seminar on Graph Transformations in Computer Science*, *LNCS* 776, pp. 280–295. Springer, 1994.
24. U. Montanari and F. Rossi. Contextual nets. *Acta Inform.*, 32:545–596, 1995.
25. C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für Instrumentelle Mathematik, Bonn, 1962.
26. W. Reisig. *Petri Nets: An Introduction*. EATCS Monographs. Springer, 1985.
27. V. Sassone. An axiomatization of the algebra of Petri net concatenable processes. *Theoret. Comput. Sci.*, 170(1-2):277–296, 1996.
28. V. Sassone. An axiomatization of the category of Petri net computations. *Math. Struct. in Comput. Sci.*, 8(2):117–151, 1998.
29. W. Vogler. Efficiency of asynchronous systems and read arcs in Petri nets. *Proc. ICALP'97*, *LNCS* 1256, pp. 538–548. Springer, 1997.
30. W. Vogler. Partial order semantics and read arcs. *Proc. MFCS'97*, *LNCS* 1295, pp. 508–517. Springer, 1997.
31. G. Winskel and M. Nielsen. Models for concurrency. *Handbook of Logic in Computer Science*. Oxford University Press, 1995.

```
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
```

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

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. 9. The theories `CAT`, `MON`, and `MONCAT`.

```
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. 10. The theory `SMONCAT`.

A Theories in Partial Membership Equational Logic

The theory of categories `CAT` is defined in Figure 9. It has sorts `Object` and `Arrow` with `Object` $<$ `Arrow`. There are two unary total operations `d`(_) and `c`(_), for *domain* and *codomain*, and a binary composition _;_ defined iff the codomain of the first argument is equal to the domain of the second argument. By convention, functions with given domain and codomain are total on that domain and codomain. It is easy to check that a model of `CAT` is a category (in which objects coincide with identity arrows), and that `CAT`-homomorphisms are just functors (cf. [20] for the details).

The theory `MON` of monoids is even simpler (Figure 9). It has a unique sort `Monoid` and two total operators: the associative tensor $\_ \otimes \_$ and the unit element `e`, which is the identity for $\_ \otimes \_$. Then, by exploiting the tensor product of theories $\bigotimes$ defined in [20], the theory of monoidal categories can be obtained by combining the theories `MON` and `CAT` as illustrated in Figure 9. Note that the tensor product construction `MON`$\bigotimes$`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 `MON`$\bigotimes$`CAT` are generated by combining the axioms of `MON` and `CAT` (see the appendix of [5] for details). 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 10, by adding the symmetric natural transformation $\gamma(\_,\_)$.