Rewriting Logic as a Unifying Framework for Petri Nets

Mark-Oliver Stehr, José Meseguer, and Peter Csaba Ölveczky

SRI International, Menlo Park, CA 94025, USA

Abstract. We propose rewriting logic as a unifying framework for a wide range of Petri nets models. We treat in detail place/transition nets and important extensions of the basic model by individual tokens, test arcs, and time. Based on the idea that "Petri nets are monoids" suggested by Meseguer and Montanari we define a rewriting semantics that maps place/transition nets into rewriting logic specifications. We furthermore generalize this result to a general form of algebraic net specifications subsuming also colored Petri nets as a special case. The soundness and completeness results we state relate the commutative process semantics of Petri nets proposed by Best and Devillers to the model-theoretic semantics of rewriting logic in the sense of natural isomorphisms between suitable functors. In addition we show how place/transition nets with test arcs and timed Petri nets can be equipped with a rewriting semantics and discuss how other extensions can be treated along similar lines. Beyond the conceptual unification of quite different kinds of Petri nets within a single framework, the rewriting semantics can provide a guide for future extensions of Petri nets and help to cope with the growing diversity of models in this field. On the practical side, a major application of the rewriting semantics is its use as a logical and operational representation of Petri net models for formal verification and for the efficient execution and analysis using a rewriting engine such as Maude, which also allows us to specify different execution and analysis strategies in the same rewriting logic language by means of reflection.

1 Introduction

This paper attempts to contribute to the general goal of unifying Petri net models by studying in detail the unification of a wide range of such models within rewriting logic [48], which is used as a logical and semantic framework. Specifically, we show how place/transition nets, nets with test arcs, algebraic net specifications, colored Petri nets, and timed Petri nets can all be naturally represented within rewriting logic. Our work extends in substantial ways previous work on the rewriting logic representation of place/transition nets [48], nets with test arcs [50], algebraic net specifications [70], and timed Petri nets [63].

The representations in question associate a rewrite specification to each net in a given class of Petri net models in such a way that concurrent computations in the original net naturally coincide with concurrent computations in the associated rewrite specification. That is, we exhibit appropriate bijections between Petri net computations and rewriting logic computations, viewed as equivalence classes of proofs, that is, as elements of the free model associated to the corresponding rewrite specification [48].

Furthermore, for certain classes of nets, namely place/transition nets and a general form of algebraic net specifications, which subsume the well-known class of colored Petri nets, we show that the representation maps into rewriting logic are *functorial*; that is, that they map in a functorial way net morphisms to rewrite specification morphisms. In addition, such functorial representations can be further extended to the level of *semantic models*, yielding *semantic equivalence theorems* (in the form of natural isomorphisms of functors) between well-known semantic models for the given class of Petri nets and the free models of the corresponding rewrite theories or, more precisely, models obtained from such free models by forgetting some structure.

As we further explain in the body of the paper, this work, including the abovementioned functorial semantics and the semantic equivalences, generalizes in some ways, and complements in others, a substantial body of work initiated by the second author in joint work with Ugo Montanari under the motto "Petri nets are monoids" [53, 45, 46, 56, 21, 54, 55, 57, 12, 13], in which categorical models are naturally associated as semantic models to Petri nets, and are shown to be equivalent to well-known "true concurrency" models. Our work is also related to linear logic representations of Petri nets [45, 46, 4, 11, 10, 26]. All this is not surprising, since, as explained in [48], both the categorical place/transition net models of [53] and the linear logic representations of place/transition nets inspired rewriting logic as a generalization of both formalisms. But, as shown in this paper, the extra algebraic expressiveness of rewriting logic is very useful to model in a simple and natural way not only place/transition nets, but also *high-level nets*, such as algebraic net specifications, colored Petri nets, and timed Petri nets.

Our proposed unification of Petri net models is not only of conceptual interest. Given that, under reasonable assumptions, rewrite theories can be executed, the representation maps that we propose provide a uniform operational semantics in terms of efficient logical deduction. Furthermore, using a rewriting logic language implementation such as Maude [19, 18], or the Real-Time Maude tool in the timed case [62, 61], it is possible to use the results of this paper to create execution environments for different classes of Petri nets. In addition, because of Maude's reflective capabilities [17], the Petri nets thus represented cannot only be executed, but they can also be formally analyzed and model checked by means of *rewriting strategies* that explore and analyze at the metalevel the different rewriting computations of a given rewrite specification.

The general way of representing Petri nets within rewriting logic that we propose is by no means limited to the net classes explicitly discussed in this paper. We believe that it can be similarly applied to other important classes of nets that we cannot discuss in detail due to space limitations. We briefly address how similar representations could be defined for other Petri net classes, such as colored Petri nets based on (higher-order) programming languages [39], nets with macroplaces [2, 3], nets with FIFO places [30, 40, 29, 27], object-oriented variants of Petri nets [68, 44], and object nets [73, 74, 28, 75] where nets are viewed as token objects.

We conclude this introduction with a brief overview of the paper: After introducing rewriting logic together with the underlying membership equational logic in the following section, we introduce in Section 3 a category of place/transition nets together with a functor that associates the process semantics of Best and Devillers [7] with each place/transition net. We then define the rewriting specification associated with a place/transition net and we establish a semantic connection in terms of a natural isomorphism at the level of symmetric monoidal categories. We conclude the section on place/transition nets by showing how test arcs can be incorporated using a slightly richer state space that satisfies certain symmetries. In Section 4 we generalize the rewriting semantics for place/transition nets to algebraic net specifications, which we view as colored net specifications over membership equational logic. As it is the case for rewriting logic, the concept of colored net specifications is quite general, since it is parameterized over an underlying logic. However, for the sake of concreteness we only deal with rewriting logic and colored net specifications over membership equational logic in this paper. As in the previous section we relate the Best-Devillers process semantics and the model-theoretic semantics obtained via rewriting logic in terms of a natural isomorphism. In Section 5 we deal with timed Petri nets, an extension of place/transition nets by a notion of real time. The model we use is closely related to the model of interval timed colored Petri nets proposed by van der Aalst [1], but for the purpose of a simpler exposition we deal with the corresponding uncolored model and focus on the essential real-time aspects. Finally, in Section 6 we conclude by discussing how our approach can be generalized or extended to the other models of Petri nets like those mentioned before.

2 Preliminaries

A *finite multiset* over a set S is a function m from S to $\mathbb{N}$ such that its support $\mathcal{S}(m) = \{s \in S \mid m(s) > 0\}$ is finite. We denote by $S^{\oplus}$ the set of finite multisets over S, by $\emptyset_S$ the empty multiset over S (we usually omit S if it is clear from the context), and we use the standard definitions of multiset membership $\in$, multiset inclusion $\sqsubseteq$, multiset union $\oplus$, and multiset difference $-$. Sometimes we write x instead of the singleton multiset containing x.

A *list* of length n over a set S is a function l from the interval $[1, n]$ of $\mathbb{N}$ to S. We denote by $\mathcal{L}(S)$ the set of lists of arbitrary length over S. Concatenation of lists u and v is written as uv. Sometimes we write x instead of the singleton list containing x. If x is a variable ranging over elements, we often use the variable $\bar{x}$ to range over lists of such elements.

Often we implicitly lift functions $f : X \to Y$ to sets $f : \mathcal{P}(X) \to \mathcal{P}(Y)$, finite multisets $f : X^{\oplus} \to Y^{\oplus}$, and lists $f : \mathcal{L}(X) \to \mathcal{L}(Y)$ in the natural homomorphic way. Given a finite set S we sometimes assume a *canonical enumeration* of S, i.e. a list $\bar{x}$ of n distinct elements such that $S = \{\bar{x}_1, \ldots, \bar{x}_n\}$ which is fixed thoughout the paper. In order to ensure the existence of a canonical enumeration of certain sets we could assume that all their elements are drawn from a single total order that we do not make explicit in this paper.

2.1 Membership Equational Logic

Membership equational logic (MEL) [9, 51] is a many-sorted logic with subsorts and overloading of function symbols. It can express partiality very directly by defining membership in a sort by means of membership equational conditions. In accordance with the terminology introduced in the references above we refer to the types of the logic as *kinds*, and we view the *sorts* for each kind as unary predicates. The atomic sentences are *equalities* $M = N$ for terms M,N of the same kind, and *memberships* $M : s$ for M a term and s a sort, both of the same kind. Sentences of MEL are universally quantified Horn clauses on the atoms.

Definition 1. A *memberhip equational signature* Ω consists of a set of *kinds* K_Ω, a set of sorts S_Ω, a function $\pi_\Omega : S_\Omega \to K_\Omega$ that associates to each sort its kind, and a family $(OP_\Omega^{\bar{k},k})_{\bar{k} \in K_\Omega{}^*, k \in K_\Omega}$ of *operator symbols* such that the following *overloading restriction* holds: If $OP_\Omega^{\bar{k},k} \cap OP_\Omega^{\bar{k}',k'} \neq \emptyset$ then $\bar{k} = \bar{k}'$ implies $k = k'$. Instead of $o \in OP_\Omega^{\bar{k},k}$ we simply write $o : \bar{k} \to k$. If $\bar{k}$ is empty we write $o : \to k$, and o is called a *constant symbol*, otherwise o is called a *function symbol.*

Given membership equational signatures Ω and Ω' a *membership equational signature morphism* $H : \Omega \to \Omega'$ consists of functions $H_K : K_\Omega \to K_{\Omega'}$, $H_S : S_\Omega \to S_{\Omega'}$ and $H_{OP} : OP_\Omega \to OP_{\Omega'}$ such that (1) $H_K(\pi_\Omega(s)) = \pi_{\Omega'}(H_S(s))$ for each sort $s \in S_\Omega$, and (2) $f : \bar{k} \to k$ in Ω implies $H_{OP}(f) : H_K(\bar{k}) \to H_K(k)$ in Ω'. We usually omit the indices of H if there is no danger of confusion. Membership equational signatures together with their morphisms form a category **MESign**.

A *kinded variable set* is a family $(X_k)_{k \in K}$ of pairwise disjoint sets which are also disjoint from the operator symbols in OP_Ω. Given a kinded variable set X, the kinded set of *Ω-terms* over X, written $\mathbf{T}_\Omega(X) = (\mathbf{T}_\Omega(X)_k)_{k \in K}$, is inductively defined as follows: (1) each variable $x \in X_k$ is in $\mathbf{T}_\Omega(X)_k$; (2) each constant symbol c with $c : \to k$ is in $\mathbf{T}_\Omega(X)_k$ for $k \in K$; (3) each *function application* of the form $f(\bar{M}_1, \ldots, \bar{M}_n)$ is in $\mathbf{T}_\Omega(X)_k$ for $f : \bar{k} \to k$ and $\bar{M}_1 \in \mathbf{T}_\Omega(X)_{\bar{k}_1}, \ldots, \bar{M}_n \in \mathbf{T}_\Omega(X)_{\bar{k}_n}$ where $\bar{k} = \bar{k}_1 \ldots \bar{k}_n$. If X is the empty variable set the terms above are called *ground terms* and we write $\mathbf{T}_\Omega$ and $\mathbf{T}_{\Omega,k}$ instead of $\mathbf{T}_\Omega(X)$ and $\mathbf{T}_\Omega(X)_k$, respectively.

We define *atomic Ω-formulae* over X as either (1) *Ω-memberships* over X of the form $M : s$ for $M \in \mathbf{T}_\Omega(X)_{\pi(s)}$, or (2) *$\Omega$-equations* over X of the form $M = N$ for $M, N \in \mathbf{T}_\Omega(X)_k$ for some kind k. Furthermore, *Ω-conditions* over

X are of the form $\bar{\phi}_1 \wedge \ldots \wedge \bar{\phi}_n$, where $\bar{\phi}_1, \ldots, \bar{\phi}_n$ are atomic formulae over X. Given an Ω-condition $\bar{\phi}_1 \wedge \ldots \wedge \bar{\phi}_n$ over X an *Ω-axiom* can be either (1) a *membership axiom* of the form $\forall\ X\ .\ M : s$ `if` $\bar{\phi}_1 \wedge \ldots \wedge \bar{\phi}_n$, where $M : s$ is an Ω-membership over X, or (2) an *equational axiom* of the form $\forall\ X\ .\ M = N$ `if` $\bar{\phi}_1 \wedge \ldots \wedge \bar{\phi}_n$, where $M = N$ is an Ω-equation over X. We usually omit the quantifier if X is empty.

A *membership equational theory (MET)* $\mathcal{T}$ consists of a signature $\Omega_{\mathcal{T}}$ and a set of $\Omega_{\mathcal{T}}$-axioms $E_{\mathcal{T}}$.

The *algebraic semantics* of membership equational logic is a standard model-theoretic one [9, 51]. *Models* of a membership equational theory are suitable algebras satisfying the axioms.

Definition 2. Let Ω be a signature. An *Ω-algebra A* consists of a *kind interpretation* $[\![k]\!]_A$ for each $k \in K$, a *sort interpretation* $[\![s]\!]_A \subseteq [\![k]\!]_A$ for each $s \in \pi^{-1}(k)$, an *operator interpretation* $[\![o_{\bar{k},k}]\!]_A$ for each $o : \bar{k} \to k$ such that $[\![c]\!]_A \in [\![k]\!]_A$ for $c :\to k$ and $[\![f]\!]_A \in [\![\bar{k}]\!]_A \to [\![k]\!]_A$ for $f : \bar{k} \to k$ where $[\![\bar{k}]\!]_A = [\![\bar{k}_1]\!]_A \times \ldots \times [\![\bar{k}_n]\!]_A$ if $\bar{k} = \bar{k}_1 \ldots \bar{k}_n$. For better readability we often write $[\![c]\!]_A$ and $[\![f]\!]_A$ instead of $[\![c_k]\!]_A$ and $[\![f_{\bar{k},k}]\!]_A$ assuming that the subscripts are clear from the context. To simplify some constructions we assume in this paper without loss of generality that $[\![k]\!]_A \cap [\![k']\!]_A = \emptyset$ for all kinds $k \neq k'$.

Let A,B be Ω-algebras. A *Ω-morphism*, written $h : A \to B$, is a kinded function $h = (h_k)_{k \in K}$ such that $h_k : [\![k]\!]_A \to [\![k]\!]_B$ for all $k \in K$ and the following conditions hold: (1) $h_k([\![s]\!]_A) \subseteq [\![s]\!]_B$ for $s \in \pi^{-1}(k)$; (2) $h_k([\![c_k]\!]_A) = [\![c_k]\!]_B$ for $c : \to k$; and (3) $h_k([\![f_{\bar{k},k}]\!]_A(\bar{a}_1, \ldots, \bar{a}_n)) = [\![f_{\bar{k},k}]\!]_B(h_{\bar{k}_1}(\bar{a}_1), \ldots, h_{\bar{k}_n}(\bar{a}_n))$ for $f : \bar{k} \to k$ with $\bar{k} = \bar{k}_1 \ldots \bar{k}_n$ and $\bar{a}_i \in [\![\bar{k}_i]\!]_A$. Ω-algebras together with Ω-morphisms constitute a category $\mathbf{Mod}(\Omega)$.

Definition 3. Let A be an Ω-algebra. An *assignment* $\beta : X \to A$ is a kinded function $\beta = (\beta_k)_{k \in K}$ associating to each $x \in X_k$ an element $\beta_k(x) \in [\![k]\!]_A$. It is extended to terms over X as follows: (1) $\beta_k(c) = [\![c_k]\!]_A$ for $c : \to k$; and (2) $\beta_k(f(\bar{M}_1, \ldots, \bar{M}_n)) = [\![f_{\bar{k},k}]\!]_A(\beta_{\bar{k}_1}(\bar{M}_1), \ldots, \beta_{\bar{k}_n}(\bar{M}_n))$ for $f : \bar{k} \to k$ and $\bar{M}_i \in \mathbf{T}_\Omega(X)_{\bar{k}_i}$ where $\bar{k} = \bar{k}_1 \ldots \bar{k}_n$. Instead of $\beta_k(M)$ for $M \in \mathbf{T}_\Omega(X)_k$ we also use the notation $\beta(M)$ or $[\![M]\!]_{A,\beta}$.

Let A be an Ω-algebra, let $\beta : X \to A$ be an assignment, and let $M, N \in \mathbf{T}_\Omega(X)_k$. We define validity of formulae starting with atomic formulae: an Ω-membership $M : s$ over X is *valid* under β iff $[\![M]\!]_{A,\beta} \in [\![s]\!]_A$; and an Ω-equation $M = N$ over X is *valid* under β iff $[\![M]\!]_{A,\beta} = [\![M]\!]_{A,\beta}$. We write $A, \beta \models \phi$ iff an atomic formula ϕ is valid under β. Furthermore, an Ω-condition $\bar{\phi}_1 \wedge \ldots \wedge \bar{\phi}_n$ over X is *valid* under β iff $A, \beta \models \bar{\phi}_i$ for each $i \in \{1 \ldots n\}$, in which case we also write $A, \beta \models \bar{\phi}_1 \wedge \ldots \wedge \bar{\phi}_n$. An Ω-axiom $\forall\ X\ .\ \phi$ `if` $\bar{\phi}_1 \wedge \ldots \wedge \bar{\phi}_n$ is *valid* iff for each assignment $\beta : X \to A$ we have $A, \beta \models \phi$ whenever $A, \beta \models \bar{\phi}_1 \wedge \ldots \wedge \bar{\phi}_n$. We also write $A \models \forall\ X\ .\ \phi$ `if` $\bar{\phi}_1 \wedge \ldots \wedge \bar{\phi}_n$ in this case. Given a set E of Ω-axioms we write $A \models E$ iff $A \models \psi$ for each $\psi \in E$. Given a MET $\mathcal{T}$ we say that A is a *$\mathcal{T}$-algebra* iff $A \models E_{\mathcal{T}}$. We write $\mathcal{T} \models \psi$ iff $A \models \psi$ for each $\mathcal{T}$-algebra

A and given a set E of Ω-axioms we write $\mathcal{T} \models E$ iff $\mathcal{T} \models \psi$ for each $\psi \in E$. We furthermore say that M and N are *E-equivalent* iff $[\![M]\!]_{A,\beta} = [\![N]\!]_{A,\beta}$ for all Ω-algebras A satisfying $A \models E$ and assignments $\beta : X \to A$.

Given METs $\mathcal{T}$ and $\mathcal{T}'$, a *MET morphism* $H : \mathcal{T} \to \mathcal{T}'$ is a membership equational signature morphism $H : \Omega_{\mathcal{T}} \to \Omega_{\mathcal{T}'}$ such that $\mathcal{T}' \models H(E_{\mathcal{T}})$, where H is lifted to terms and axioms in the natural homomorphic way. We say that $\mathcal{T}$ is a *subtheory* of $\mathcal{T}'$, written $\mathcal{T} \hookrightarrow \mathcal{T}'$, iff there is a MET morphism $J : \mathcal{T} \hookrightarrow \mathcal{T}'$ that is an inclusion.

METs together with their morphisms form a category **MET**, and given a MET $\mathcal{T}$ the class of $\mathcal{T}$-algebras together with their Ω-morphisms constitutes a full subcategory of $\mathbf{Mod}(\Omega)$ denoted by $\mathbf{Mod}(\mathcal{T})$. Each MET morphism $H : \mathcal{T} \to \mathcal{T}'$ induces an obvious forgetful functor $\mathbf{Mod}(H) : \mathbf{Mod}(\mathcal{T}') \to \mathbf{Mod}(\mathcal{T})$ that we also write as $\mathbf{U}_H$. In fact, we have a contravariant functor $\mathbf{Mod} : \mathbf{MET} \to \mathbf{Cat}^{\mathrm{op}}$. Given an inclusion $I : \mathcal{T} \hookrightarrow \mathcal{T}'$ and a $\mathcal{T}'$-algebra A we also write $A|\mathcal{T}$ instead of $\mathbf{U}_I(A)$.

METs have initial and free models [9, 51]. In fact, given a MET $\mathcal{T}'$ there exists an initial $\mathcal{T}'$-algebra, written $\mathbf{I}(\mathcal{T}')$. More generally, given a MET morphism $H : \mathcal{T} \to \mathcal{T}'$ between METs $\mathcal{T}$ and $\mathcal{T}'$ there exists a free functor $\mathbf{F}_H : \mathbf{Mod}(\mathcal{T}) \to \mathbf{Mod}(\mathcal{T}')$, i.e. a functor that is left adjoint to $\mathbf{U}_H$. In the following we write η_H and ϵ_H for unit and counit, respectively, of this adjunction, i.e., we have natural transformations $\eta_H(A) : A \to \mathbf{U}_H(\mathbf{F}_H(A))$ for $\mathcal{T}$-algebras A and $\epsilon_H(A') : \mathbf{F}_H(\mathbf{U}_H(A')) \to A'$ for $\mathcal{T}'$-algebras A'.

In contrast to an entirely loose or entirely initial semantics of membership equational theories, in practice a mixed specification style is used, where certain subtheories are intended to be equipped with initial interpretations or certain subtheories are interpreted freely over their parameter specifications. To make such restrictions on the models explicit in the specification we enrich a membership equational theory by initiality and freeness constraints [23, 35], and refer to these enriched theories as *membership equational logic specifications (MES)*.

From a model-theoretic point of view, constraints are axioms that are treated in full analogy to membership or equational axioms, i.e., as sentences that have to be valid in all models. Hence, the models of a MES are algebras which satisfy all the given initiality and freeness constraints. Given a MES model, a model of a subspecification is obtained by its associated forgetful functor $\mathbf{U}_K$ for K the corresponding subspecification inclusion. In particular, this means that a model induces a unique interpretation for each subspecification, which is the justification for the condition on ϵ below. The notion of constraint we use here is a special case of the notion proposed in [35], where initiality constraints are seen as a special case of freeness constraints.

Definition 4. Let $J : \mathcal{T}'' \hookrightarrow \mathcal{T}'$ and $I : \mathcal{T}' \hookrightarrow \mathcal{T}$ be MET inclusions. A *constraint* for $\mathcal{T}$ can take one of the following two forms: (1) $\mathcal{T}'$ `initial` or (2) $\mathcal{T}'$ `free over` $\mathcal{T}''$. A *membership equational specification (MES)* $\mathcal{S}$ is a MET $\mathcal{T}_{\mathcal{S}}$ together with a set $C_{\mathcal{S}}$ of constraints for $\mathcal{T}_{\mathcal{S}}$.

Let A be a $\mathcal{T}$-algebra. We define *validity* of a constraint as follows: (1) the constraint $\mathcal{T}'$ `initial` is *valid* iff the unique morphism from $\mathbf{I}(\mathcal{T}')$ to $A|\mathcal{T}'$ is an isomorphism, and (2) the constraint $\mathcal{T}'$ `free over` $\mathcal{T}''$ is *valid* iff $\epsilon_J(A|\mathcal{T}') : \mathbf{F}_J(A|\mathcal{T}'') \to A|\mathcal{T}'$ is an isomorphism. Given a MES $\mathcal{S}$ with an underlying MET $\mathcal{T}_\mathcal{S}$, an $\mathcal{S}$-*algebra* is a $\mathcal{T}_\mathcal{S}$-algebra A such that each constraint in $C_\mathcal{S}$ is valid in A.

In complete analogy to METs we define: Given MESs $\mathcal{S}$ and $\mathcal{S}'$, a *MES morphism* $H : \mathcal{S} \to \mathcal{S}'$ is a morphism $H : \mathcal{T} \to \mathcal{T}'$ such that $\mathcal{S}' \models H(C_\mathcal{S})$, i.e. the constraints $H(C_\mathcal{S})$ are valid in all $\mathcal{S}'$-algebras, where H is lifted to constraints in the natural way. $\mathcal{S}$ is a *subspecification* of $\mathcal{S}'$, written $\mathcal{S} \hookrightarrow \mathcal{S}'$, iff there is a MES morphism $J : \mathcal{S} \hookrightarrow \mathcal{S}'$ that is an inclusion. MESs together with their morphisms form a category **MES** and the category of $\mathcal{S}$-algebras $\mathbf{Mod}(\mathcal{S})$ is the full subcategory of $\mathbf{Mod}(\mathcal{T}_\mathcal{S})$ that contains only $\mathcal{S}$-algebras. Each MES morphism $H : \mathcal{S} \to \mathcal{S}'$ induces an obvious forgetful functor $\mathbf{Mod}(H) : \mathbf{Mod}(\mathcal{S}') \to \mathbf{Mod}(\mathcal{S})$ that we also write as $\mathbf{U}_H$. Again, we have a contravariant functor $\mathbf{Mod} : \mathbf{MES} \to \mathbf{Cat}^{\mathrm{op}}$ that generalizes $\mathbf{Mod} : \mathbf{MET} \to \mathbf{Cat}^{\mathrm{op}}$. Given an inclusion $I : \mathcal{S}' \hookrightarrow \mathcal{S}$ and an $\mathcal{S}$-algebra A we also write $A|\mathcal{S}'$ instead of $\mathbf{U}_I(A)$.

Furthermore, we introduce interpreted specifications together with a general notion of morphism that reflects a transformation of the specification as well as a transformation of the algebras possibly associated with different specifications.

Definition 5. An *interpreted MES* $(\mathcal{S}, A)$ consists of a MES $\mathcal{S}$ and a $\mathcal{S}$-algebra A. The category **IMES** of interpreted MES is given by the Grothendiek construction $\Sigma(\mathbf{Mod})$ where $\mathbf{Mod} : \mathbf{MES} \to \mathbf{Cat}^{\mathrm{op}}$. Recall that a morphism $(H, h) : (\mathcal{S}, A) \to (\mathcal{S}', A')$ in $\Sigma(\mathbf{Mod})$ consists of morphisms $H : \mathcal{S} \to \mathcal{S}'$ and $h : A \to \mathbf{U}_H(A')$ satisfying the conditions of the Grothendiek construction [72].

Given a MES $\mathcal{S}$, the *operational semantics* [9], that can be used to efficiently execute a specification under certain assumptions, is explained using a refinement of $\mathcal{S}$, namely by viewing $E_\mathcal{S}$ as composed of a set $E^S_\mathcal{S}$ of *structural axioms* and a set $E^C_\mathcal{S}$ of *computational axioms*, i.e., $E = E^S_\mathcal{S} \cup E^C_\mathcal{S}$. Assuming that the computational axioms in $E^C_\mathcal{S}$ satisfy the variable restriction explained below the equational axioms in $E^C_\mathcal{S}$ can be seen as reduction rules that operate modulo the equational theory induced by $E^S_\mathcal{S}$. Identifying $E^S_\mathcal{S}$-equivalent terms, we write $M \Rightarrow M'$ to express that M can be *reduced* to M' by applying an equation in $E^C_\mathcal{S}$ to a subterm of M. The *variable restriction* requires that all variables occurring in the righthand side or in the condition of an equational axiom also appear in the lefthand side, and for membership axioms that all variables occurring in the condition also appear in the conclusion. A MET $\mathcal{S}$ is said to be *executable* iff the variable restriction[1] is satisfied for all axioms in $E^C_\mathcal{S}$ and the following conditions hold after identifying $E^S_\mathcal{S}$-equivalent terms: the equations in $E^C_\mathcal{S}$ are confluent, equational and membership axioms in $E^C_\mathcal{S}$ are terminating, equational axioms in $E^C_\mathcal{S}$ are sort-decreasing and satisfy the regularity condition. For formal details of

[1] In its most recent version Maude imposes an even weaker restriction for executability due to the admissibility of conditions with *matching equations* [20].

these conditions we refer to [9]. In particular, these conditions imply that each term M has a unique normal form w.r.t. $\Rightarrow$ which is denoted by $\mathrm{NF}(M)$.

2.2 Rewriting Logic

In the simplified setting of [48] a rewrite specification $\mathcal{R}$ consists of a single-sorted signature $\Omega_{\mathcal{R}}$, a set $E_{\mathcal{R}}$ of equations over $\Omega_{\mathcal{R}}$, and a set $R_{\mathcal{R}}$ of labelled rewrite rules of the form $\forall\ X\ .\ l : M \to N$ `if` $\bar{\phi}_1 \wedge \ldots \wedge \bar{\phi}_n$, where l is a label, M and N are $\Omega_{\mathcal{R}}$-terms, and $\bar{\phi}_1 \wedge \ldots \wedge \bar{\phi}_n$ is a $\Omega_{\mathcal{R}}$-condition[2] over the variable set X. The rewrite rules in $R_{\mathcal{R}}$ are applied *modulo* the equations $E_{\mathcal{R}}$. *Rewriting logic (RWL)* has rules of deduction to infer all rewrites, i.e., those sentences of the form $P : M \to N$ that are valid in a given rewrite specification [48]. A *rewrite* $P : M \to N$ means that the term M *rewrites* to the term N modulo $E_{\mathcal{R}}$, and this rewrite is witnessed by the *proof term* P. Apart from general (concurrent) rewrites $P : M \to N$ that are generated from identity and atomic rewrites by parallel and sequential composition, rewriting logic classifies its most basic rewrites as follows: a *one-step (concurrent) rewrite* is generated by parallel composition from identity and atomic rewrites and contains at least one atomic rewrite, and a *one-step sequential rewrite* is a one-step rewrite containing exactly one atomic rewrite.

From a more general point of view, rewriting logic is parameterized by the choice of its underlying equational logic, which can be single-sorted, many-sorted, order-sorted and so on. In the design of the Maude language [19, 18], *membership equational logic* has been chosen as the underlying equational logic. To introduce rewriting logic over membership equational logic, abbreviated as $\mathrm{RWL}_{\mathrm{MEL}}$ or just RWL, we assume an underlying MES $\mathcal{S}_{\mathcal{R}}$ with a distinguished *data subspecification* $\mathcal{S}^D_{\mathcal{R}}$. The data subspecification specifies the static data part of the system, whereas the remaining part of $\mathcal{S}_{\mathcal{R}}$ specifies the *state space* by introducing the *rewrite kinds*, i.e., kinds not already contained in $\mathcal{S}^D_{\mathcal{R}}$ whose terms correspond to states and therefore can be rewritten, together with their algebraic structure, which characterizes the possibilities of parallel composition. In the context of this paper the state space is always specified in a purely equational way.

Definition 6. A *rewrite specification (RWS)* $\mathcal{R}$ consists of a MES $\mathcal{S}_{\mathcal{R}}$ with a distinguished *data subspecification* $\mathcal{S}^D_{\mathcal{R}}$, a set of *labels* $L_{\mathcal{R}}$, and a set of *rules* $R_{\mathcal{R}}$ of the form $\forall\ X\ .\ l : M \to N$ `if` $\bar{\phi}_1 \wedge \ldots \wedge \bar{\phi}_n$ where $l \in L_{\mathcal{R}}$, $\bar{\phi}_1 \wedge \ldots \wedge \bar{\phi}_n$ is a $\mathcal{S}_{\mathcal{R}}$-condition over X, and $M, N \in \mathbf{T}_{\mathcal{R}}(X)_k$ in $\mathcal{S}_{\mathcal{R}}$ for a rewrite kind k. To simplify the exposition we identify either $E_{\mathcal{R}}$-equivalent or $E^S_{\mathcal{R}}$-equivalent terms in the context of a RWS $\mathcal{R}$ whenever we are concerned with the algebraic semantics or the operational semantics, respecively.

Given two RWSs $\mathcal{R}$ and $\mathcal{R}'$, a RWS morphism $H : \mathcal{R} \to \mathcal{R}'$ consists of a MES morphism $H_{\mathcal{S}} : \mathcal{S}_{\mathcal{R}} \to \mathcal{S}_{\mathcal{R}'}$ and a function $H_L : L_{\mathcal{R}} \to L_{\mathcal{R}'}$ such that $H_{\mathcal{S}}$ has a

[2] Rewriting logic as presented in [48] admits rewrites in conditions of rules, but we do not exploit this possibility in the present paper.

restriction $H_D : \mathcal{S}^D_{\mathcal{R}} \to \mathcal{S}^D_{\mathcal{R}'}$ to the data subspecification and for each rule $r \in R_{\mathcal{R}}$ there is a rule in $R_{\mathcal{R}'}$ that is $E_{\mathcal{R}'}$-equivalent to $H(r)$ up to a renaming of the variables, where H is lifted to rules in the obvious homomorphic way. RWSs together with their morphisms form a category that is denoted by **RWS**.

The *algebraic semantics* of rewriting logic is defined as follows. A model of a rewrite specification (RWS) $\mathcal{R}$ is a model A of the underlying MES $\mathcal{S}_{\mathcal{R}}$ together with an enriched categorical structure for each set $[\![k]\!]_A$, where k is a rewrite kind. The interpretation of $\_ : \_ \to \_$, which can be regarded as a ternary predicate, is given by the arrows of the category. Sequential composition of rewrite proofs is interpreted by arrow composition, and parallel composition operators are interpreted by enriching the category with an algebraic structure as it has been specified for the rewrite kinds in $\mathcal{S}_{\mathcal{R}}$. In order to be a model, the category has to satisfy a number of natural requirements, namely, functoriality w.r.t. the algebraic structure that is relevant for the rewrite kinds, the equations in $\mathcal{S}_{\mathcal{R}}$ that are relevant for the rewrite kinds lifted to arrows, and for each rule in $\mathcal{R}$ the so-called exchange and decomposition laws. For a detailed description of these requirements we refer to [48]. The models of a RWS we consider in this paper are freely generated over models of the data subspecification $\mathcal{S}^D_{\mathcal{R}}$. In the important case where $\mathcal{S}^D_{\mathcal{R}}$ is interpreted initially, we obtain precisely the initial model described in [48]. A more precise definition of the algebraic semantics of rewriting logic will be given in Sections 3.2 and 4.4 for the particular forms of underlying specifications that are relevant for Petri nets.

The *operational semantics* of RWSs extends the operational semantics of MESs by applying computational equations $E^C_{\mathcal{R}}$ *and* rewrite rules $R_{\mathcal{R}}$ modulo the structural equations $E^S_{\mathcal{R}}$. In this way we can achieve the effect of rewriting modulo $E_{\mathcal{R}}$ provided that a suitable coherence requirement between equations and rules is satisfied. In particular, we say that a RWS is *weakly executable* iff the underlying MES is executable, and the equations in $E^C_{\mathcal{R}}$ are coherent with the rules in $R_{\mathcal{R}}$ modulo $E^S_{\mathcal{R}}$. Identifying terms that are $E^S_{\mathcal{R}}$-equivalent and identifying proof terms that are equivalent in the sense of [48], *coherence* means that if $P : M \to N$ then there is a term N' such that $\mathrm{NF}(P) : \mathrm{NF}(M) \to N'$ and $N' \Rightarrow^* \mathrm{NF}(N)$ (this is stronger than coherence in [78] since we take proofs into account). A RWS is *strongly executable* iff additionally the *variable restriction for rules* is satisfied, i.e., all variables occurring in the righthand side or in the condition of a rule also appear in the lefthand side. In this case matching is sufficient for finding instantiations for the variables, whereas in the case of weak executability a strategy is needed to take care of this.

3 Place/Transition Nets

Place/transition nets (PTNs) are a model of concurreny in which behaviour is governed by local state changes in a distributed state space. The global distributed state of the system is represented by a *marking*, which assigns a number of indistinguishable *tokens* to each *place*. State changes that may occur in the system are specified by *transitions*. Each transition can only affect the part of the marking that is local to the transition, i.e., present in the places the transition is connected to. More precisely, a local state change corresponds to the atomic occurrence of a transition which removes tokens from its *input places* and produces tokens on its *output places*. The number of tokens that are transported by an arc is specified by its *inscription*.

As an example consider the PTN modeling an instance of the well-known banker's problem depicted in Fig. 1, which models the situation of a bank loaning money to (in this case two) clients. As usual, places and transitions are drawn as circles and rectangles, respectively. The flow relation and the weight function are given by arrows and their inscriptions. An additional initial marking is specified by place inscriptions. The money available for clients is modeled by the number of tokens in the place `BANK`. Furthermore, each client n has an individual credit limit modeled by a place `CLAIM-`n. The fact that client n requests and receives money is modeled by a transition `GRANT-`n and we assume that after exhausting the credit limit client n returns all the money via the transiton `RETURN-`n.

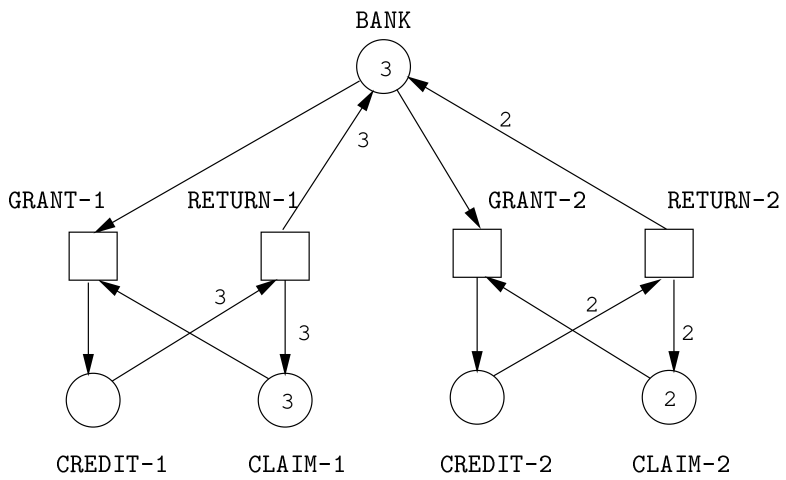

Fig. 1. Banker's problem with two clients

We now give formal definitions of basic nets and define a PTN as a particular form of an inscribed net. Instead of just finite nets we admit infinite nets, but we restrict our attention to nets with transitions that can affect only a finite part of the marking (locality principle) so that each transition can be represented in a finitary way.

Definition 7. A *net* N consists of a set of *places* P_N, a set of *transitions* T_N disjoint from P_N, and a *flow relation* $F_N \subseteq (P_N \times T_N) \cup (T_N \times P_N)$ such that ${}^\bullet t = \{p \mid p\ F_N\ t\}$ and $t^\bullet = \{p \mid p\ F_N\ t\}$ are finite for each $t \in T_N$ (local finiteness). A net is *finite* iff the sets P_N and T_N are finite. Given nets N and N', a *net morphism* $H : N \to N'$ consists of functions $H_P : P_N \to P_{N'}$ and $H_T : T_N \to T_{N'}$ such that $H_P({}^\bullet t) = {}^\bullet H_T(t)$ and $H_P(t^\bullet) = H_T(t)^\bullet$. Nets together with their morphisms form a category **Net**.

A place/transition net is essentially a net with arcs inscribed by natural numbers.

Definition 8. A *place/transition net (PTN)* $\mathcal{N}$ consists of: (1) a net $N_\mathcal{N}$ and (2) an *arc inscription* $W_\mathcal{N} : F_\mathcal{N} \to \mathbb{N}$. $W_\mathcal{N}$ is extended to $W_\mathcal{N} : (P_\mathcal{N} \times T_\mathcal{N}) \cup (T_\mathcal{N} \times P_\mathcal{N}) \to \mathbb{N}$ in such a way that $(x, y) \notin F_\mathcal{N}$ implies $W_\mathcal{N}(x, y) = 0$. Given PTNs $\mathcal{N}$ and $\mathcal{N}'$, a *PTN morphism* $H : \mathcal{N} \to \mathcal{N}'$ is a net morphism $H : N_\mathcal{N} \to N_{\mathcal{N}'}$ such that:

1. $W_{\mathcal{N}'}(p', t') = W_\mathcal{N}(p_1, t) + \ldots + W_\mathcal{N}(p_n, t)$
   for all $p' \in P_{\mathcal{N}'}$, $t' \in T_{\mathcal{N}'}$, $t \in H^{-1}(t')$,
   and $\{p_1, \ldots, p_n\} = H^{-1}(p') \cap {}^\bullet t$ with distinct p_i, and
2. $W_{\mathcal{N}'}(t', p') = W_\mathcal{N}(t, p_1) + \ldots + W_\mathcal{N}(t, p_n)$
   for all $p' \in P_{\mathcal{N}'}$, $t' \in T_{\mathcal{N}'}$, $t \in H^{-1}(t')$,
   and $\{p_1, \ldots, p_n\} = H^{-1}(p') \cap t^\bullet$ with distinct p_i.

PTNs together with their morphisms form a category **PTN**. Each net N can be conceived as a PTN $\mathcal{N}$ with $N_\mathcal{N} = N$ and $W_\mathcal{N}(x, y) = 1$ iff $x \; F_N \; y$.

The notion of net morphism we use here is more restrictive than the (topological) net morphisms used in [64] and close to, but slightly stronger than, the (algebraic) net morphisms used in [53]. The justification for our definition is that net morphisms should be morphisms in the sense of [64] and should preserve the behaviour in the strongest reasonable sense. Given a net morphism $H : \mathcal{N} \to \mathcal{N}'$ the intention is that the behaviour of $\mathcal{N}$ is subsumed by the behaviour of $\mathcal{N}'$, although $\mathcal{N}'$ may exhibit a richer behaviour. In this paper we focus on a description of behaviour by Best-Devillers processes in a way that generalizes the well-known step semantics. Indeed, not only the interleaving semantics but also the step semantics and the process semantics can be regarded as labelled transition systems where the states are markings and the labels are steps or processes, respectively. In the case of Best-Devillers processes, the labelled transition system is equipped with additional algebraic structure which will be made explicit by regarding the transition system as a symmetric monoidal category.

Definition 9. Let $\mathcal{N}$ be a PTN. A *marking* is a multiset of places. A *(concurrent) step* is a nonempty finite multiset of transitions. The *set of markings* and the *set of steps* are denoted by $\mathcal{M}_\mathcal{N}$ and $\mathcal{ST}_\mathcal{N}$, respectively. We define *preset* and *postset functions* $\partial_0, \partial_1 : T_\mathcal{N} \to \mathcal{M}_\mathcal{N}$ by $\partial_0(t)(p) = W(p, t)$ and $\partial_1(t)(p) = W(t, p)$, respectively. The *(concurrent) step semantics* of a place/transition net $\mathcal{N}$ is given by the labelled transition system which has $\mathcal{M}_\mathcal{N}$ as its set of states, $\mathcal{ST}_\mathcal{N}$ as its set of labels and a transition relation $\to \;\subseteq \mathcal{M}_\mathcal{N} \times \mathcal{ST}_\mathcal{N} \times \mathcal{M}_\mathcal{N}$

defined by $m_1 \xrightarrow{e} m_2$ iff there is a marking m such that, for all $p \in P_{\mathcal{N}}$,

$$m_1(p) = m(p) + \partial_0(e)(p) \text{ and}$$
$$m_2(p) = m(p) + \partial_1(e)(p) .$$

Writing the occurrence rule in the way given above makes it evident that the occurrence of an action replaces its preset by its postset, whereas the remainder of the marking, here denoted by m, is not involved in this process. This is an important fact that will be made formally explicit in the process semantics that we review subsequently in a somewhat informal style. For details we refer to [21] and [7].

Definition 10. An *occurrence net* $\mathcal{N}$ is a net such that $F_{\mathcal{N}}$ is acyclic and $|^\bullet p|$, $|p^\bullet| \leq 1$ for each $t \in P_{\mathcal{N}}$. Given an occurrence net $\mathcal{N}$, $F_{\mathcal{N}}$ induces a partial order $(<) = {F_{\mathcal{N}}}^+$ on $P_{\mathcal{N}} \cup T_{\mathcal{N}}$ and its minimal and maximal elements are denoted by $Max(\mathcal{N})$ and $Min(\mathcal{N})$, respectively.

Let $\mathcal{N}$ be a PTN. Then a *finite process* $\mathcal{P}$ of $\mathcal{N}$ with *origin* marking m_1 and *destination* marking m_2 consists of a finite occurrence net $N_{\mathcal{P}}$ and a PTN morphism $L_{\mathcal{P}} : N_{\mathcal{P}} \to \mathcal{N}$ (where $N_{\mathcal{P}}$ is viewed as a PTN) such that $L_{\mathcal{P}}(Min(N_{\mathcal{P}})) = m_1$ and $L_{\mathcal{P}}(Max(N_{\mathcal{P}})) = m_2$. Given finite processes $\mathcal{P}$ and $\mathcal{P}'$ the *parallel composition* of $\mathcal{P}$ and $\mathcal{P}'$ is defined as the disjoint union of the underlying nets and label functions. Given processes $\mathcal{P}$ and $\mathcal{P}'$ such that the destination of $\mathcal{P}$ is equal to the origin of $\mathcal{P}'$, a *sequential composition* of $\mathcal{P}$ and $\mathcal{P}'$ is obtained by disjoint union (as above) pairwise identifying maximal places of $\mathcal{P}$ with minimal places of $\mathcal{P}'$, where every two places to be identified must have the same label. Notice that in general the result of sequential composition is not unique [21].

Intuitively, a process of a PTN is generated by "temporal unfolding" starting from a marking that becomes the origin of the process. Observe that for a given finite process $\mathcal{P}$ of $\mathcal{N}$, not only $Min(\mathcal{P})$ and $Max(\mathcal{P})$ but each snapshot (S-cut in the sense of [8]) of $\mathcal{P}$ corresponds to a marking of $\mathcal{N}$ by virtue of $L_{\mathcal{P}}$. The ambiguity of the result of sequential composition is caused by a snapshot corresponding to a marking with several identical tokens in some place, say p. Consider a transition in $\mathcal{N}$ that removes one token from p. A single firing of this transition gives rise to two different processes, since identical tokens are represented by different places in the process net. An obvious solution to avoid this ambiguity is to restrict our attention to *safe processes*, i.e., processes that take place in the safe part of the state space where such situations do not occur. A marking m is said to be a *safe* marking iff all markings m' reachable from m in the step semantics satisfy $m'(p) \leq 1$ for all $p \in P$. A process is said to be *safe* iff its origin is safe. Safe processes coincide with the classical notion of processes if we consider 1-safe PTNs which are equivalent to contact-free elementary net systems [8, 64]. Our definition of safe processes is restrictive enough to ensure that the class of safe finite processes is always closed under sequential composition, a property that is not shared by the subclass of finite processes with the weaker property that all markings m corresponding to snapshots (S-cuts) satisfy $m(p) \leq 1$ for each $p \in P$.

Definition 11. A *(strict) monoidal category (MC)* $\mathbf{C}$ is a category equipped with a monoidal operation $\_ \otimes_{\mathbf{C}} \_$ and an identity object $id_{\mathbf{C}}$ such that $\_ \otimes_{\mathbf{C}} \_$ is an associative bifunctor with left and right identity $id_{\mathbf{C}}$. A monidal category morphism $h : \mathbf{C} \to \mathbf{C}'$ is a functor that preserves $\_ \otimes \_$ and id, i.e., $h(u \otimes_{\mathbf{C}} v) = h(u) \otimes_{\mathbf{C}'} h(v)$ and $h(id_{\mathbf{C}}) = id_{\mathbf{C}'}$. If in addition $\_ \otimes_{\mathbf{C}} \_$ is commutative, then we say that $\mathbf{C}$ is a *(strictly) symmetric (strict) monoidal category (SMC)*. The category of SMCs is denoted by **SMC**.

A variation of an SMC is a *partial SMC* $\mathbf{C}$ where $\_ \otimes_{\mathbf{C}} \_$ is a partial functor and each equation in the definition of SMCs is only required to be satisfied iff both sides are defined. The category of partial SMCs is denoted by **PSMC**. Clearly, **SMC** is a subcategory of **PSMC**.

Definition 12. The *safe process semantics* $\mathbf{SP}(\mathcal{N})$ of a PTN $\mathcal{N}$ is given by a partial SMC that has safe markings as objects and safe processes as arrows. Arrow composition is given by sequential composition, the partial monoidal operation is given by parallel composition, and the identity for an object m is given by the finite process without transitions with origin m and destination m. **SP** can be extended to a functor $\mathbf{SP} : \mathbf{SPTN} \to \mathbf{PSMC}$, where **SPTN** is the subcategory of **PTN** obtained by restricting morphisms to safe PTN morphisms. Here, a PTN morphism $H : \mathcal{N} \to \mathcal{N}'$ is *safe* iff it maps each safe marking in $\mathcal{N}$ to a safe marking in $\mathcal{N}'$. Now **SP** lifts each safe PTN morphism $H : \mathcal{N} \to \mathcal{N}'$ to a functor $\mathbf{SP}(H) : \mathbf{SP}(\mathcal{N}) \to \mathbf{SP}(\mathcal{N}')$ defined in the obvious way.

If we restrict our attention to safe markings there is a close correspondence between the step semantics and the process semantics: Each step sequence, i.e., each computation w.r.t. the step semantics, generates a unique process, and a process determines a set of step sequences that contains the original one. As a consequence processes are more abstract than step sequences. A similar correspondence exists for the interleaving semantics, i.e., if we restrict steps to single transitions. Both correspondences are investigated in [7]. On the other hand, the authors of [7] observe that step sequences and processes become incomparable when we admit markings that are not safe, which means that the natural view of processes as an abstraction of step sequences does not hold anymore. In order to recover this correspondence a more abstract notion of process is needed, and in fact Best-Devillers processes [7], which became also known as commutative processes [53, 21], provide such a notion. In contrast to processes which adhere to the *individual token philosophy*,[3] Best-Devillers processes share with step sequences the *collective token philosophy*, meaning that identical tokens on a place in the system are not distinguished in the process. This allows us to define an

[3] A functorial semantics following the individual token philosophy has recently been given in [14] by using pre-nets, a refinement of PTNs.

operation of sequential composition that has a unique result whenever sequential composition is possible. The following definition of Best-Devillers processes is equivalent to the definition given in [7], except for the fact that [7] does not make explicit the algebraic and categorical structure.

Definition 13. Let $\mathcal{P}$ and $\mathcal{P}'$ be finite processes and let $p_1, p_2 \in P_{\mathcal{P}}$ with $L_{\mathcal{P}}(p_1) = L_{\mathcal{P}}(p_2)$. We define a predicate $\mathrm{swap}(\mathcal{P}, \mathcal{P}', p_1, p_2)$ which holds iff $P_{\mathcal{P}'} = P_{\mathcal{P}}$, $T_{\mathcal{P}'} = T_{\mathcal{P}}$, $L_{\mathcal{P}'} = L_{\mathcal{P}}$ and:

1. $t\ F_{\mathcal{P}'}\ p \Leftrightarrow t\ F_{\mathcal{P}}\ p$,
2. $p\ F_{\mathcal{P}'}\ t \Leftrightarrow p\ F_{\mathcal{P}}\ t$ if $p \neq p_1$ and $p \neq p_2$,
3. $p_1\ F_{\mathcal{P}'}\ t \Leftrightarrow p_2\ F_{\mathcal{P}}\ t$,
4. $p_2\ F_{\mathcal{P}'}\ t \Leftrightarrow p_1\ F_{\mathcal{P}}\ t$.

We define an *equivalence* on finite processes as the smallest equivalence relation that contains $(\mathcal{P}, \mathcal{P}')$ if there are $p_1, p_2 \in P_{\mathcal{P}}$ such that $L_{\mathcal{P}}(p_1) = L_{\mathcal{P}}(p_2)$ and $\mathrm{swap}(\mathcal{P}, \mathcal{P}', p_1, p_2)$ holds. The equivalence classes are called *Best-Devillers processes*.

The notions of *origin*, *destination*, *parallel* and *sequential composition* of processes are lifted to Best-Devillers processes in the obvious way. At this level the result of sequential composition becomes unique, since all potentially different results obtained by composing two processes fall into the same equivalence class.

Definition 14. The *Best-Devillers process semantics* $\mathbf{BDP}(\mathcal{N})$ of a PTN $\mathcal{N}$ is given by an SMC that has markings as objects and Best-Devillers processes as arrows. Arrow composition is given by sequential composition, the monoidal operation is given by parallel composition, and the identity for an object m is given by the Best-Devillers process without transitions with origin m and destination m. $\mathbf{BDP}$ can be extended to a functor $\mathbf{BDP} : \mathbf{PTN} \to \mathbf{SMC}$ that sends each PTN morphism $H : \mathcal{N} \to \mathcal{N}'$ to a functor $\mathbf{BDP}(H) : \mathbf{BDP}(\mathcal{N}) \to \mathbf{BDP}(\mathcal{N}')$.

The above definition is also equivalent to the one given in [21], although we define Best-Devillers processes as a quotient of (classical) processes as in [7] rather than as a quotient of *concatenable processes* as in [21]. Concatenable processes are a slight refinement of finite (classical) processes: a concatenable process is a finite process together with a total ordering of $\{p \in Min(\mathcal{N}) \mid L(p) = p'\}$ for each place p' in the origin and a total ordering of $\{p \in Max(\mathcal{N}) \mid L(p) = p'\}$ for each place p' in the destination. Using this refined notion of process the obvious definition of sequential composition, where places are only identified if they have the same position in this order, yields a unique result, which allows us to view the class of concatenable processes as a category.

Since a safe process is only equivalent to itself, it corresponds to a Best-Devillers process given by a singleton equivalence class. Hence each safe process can be regarded as a Best-Devillers process giving rise to an injection $\boldsymbol{\iota}(\mathcal{N}) : \mathbf{SP}(\mathcal{N}) \to \mathbf{BDP}(\mathcal{N})$. Actually we can state the following stronger

Remark 1. **SP** : **SPTN** $\rightarrow$ **PSMC** is a subfunctor of **BDP** : **SPTN** $\rightarrow$ **PSMC** (the obvious restriction of **BDP** : **PTN** $\rightarrow$ **SMC**) as witnessed by $\boldsymbol{\iota}$: **SP** $\rightarrow$ **BDP** which is in fact a natural transformation.

We take this remark as a justification for focusing primarily on the Best-Devillers processes in the following, keeping in mind that classical safe processes form an important subcategory. In the context of nets with individual tokens we shall give some additional arguments for the relevance of this subcategory.

3.1 Rewriting Semantics: An Example

Rewriting logic can provide a direct semantics of PTNs following the motto "Petri nets are monoids" advocated in [53]. In fact, the categorical semantics presented in that work and also the relation between PTNs and linear logic explained in [46] inspired the development of rewriting logic.

The PTN of the banker's problem can be represented by the following RWS given in Maude syntax [19, 18], which consists of a MES specification and a set of rewrite rules. As usual in Maude, the rewrite kind `[Marking]` is implicitly introduced by introducing a sort `Marking` of this kind.[4]

```
sort Marking .

op empty : -> Marking .
op __ : Marking Marking -> Marking [assoc comm id: empty] .

ops BANK CREDIT-1 CREDIT-2 CLAIM-1 CLAIM-2 : -> Marking .

rl [GRANT-1]  : BANK CLAIM-1 => CREDIT-1 .

rl [RETURN-1] : CREDIT-1 CREDIT-1 CREDIT-1 =>
                BANK BANK BANK CLAIM-1 CLAIM-1 CLAIM-1 .

rl [GRANT-2]  : BANK CLAIM-2 => CREDIT-2 .

rl [RETURN-2] : CREDIT-2 CREDIT-2 =>
                BANK BANK CLAIM-2 CLAIM-2 .
```

Here we have applied the translation of PTNs into rewriting logic suggested in [48], which is closely related to the translation of PTNs into linear logic [46]. A marking is represented as an element of the finite multiset sort `Marking`. The constant `empty` represents the empty marking and `__` is the corresponding multiset union operator. Associativity, commutativity and identity laws are specified

[4] In fact, here and in the rest of the paper `Marking` and `[Marking]` can be identified, since the latter does not contain any additional (error) elements (cf. [9, 51]).

as structural equations by the operator attributes in square brackets. For each place p there is a constant p, called *token constructor*, representing a single token residing in that place. In fact, under the initial semantics `Marking` is a multiset sort over tokens generated by these token constructors. For each transition t there is a rule, called *transition rule*, labelled by t and stating that its preset marking may be replaced by its postset marking.

As clearly demonstrated by the use of rewrite rules in the above RWS, there is an important difference between the reduction rules induced by computational equations of a MES and the rewrite rules of a RWS: The relation induced by one-step rewrites is in general neither terminating nor confluent, although there may be situations where this is the case. Only terminating systems where for each initial state there is a unique final state can be described by terminating and confluent rewrite rules. Hence this generalization is a practical necessity to represent general system models. For instance, the PTN model of the banker's problem has not only infinite executions but also finite ones due to the possibility of deadlock. Therefore, the transition system is neither terminating nor confluent in this case.

In order to control the execution of a RWS the user can specify a strategy which successively selects rewrite rules and initiates rewriting steps. For instance, in the case of the banker's example it is possible to define an execution strategy that avoids states which are necessarily leading to a deadlock such that the banker stays always in the "safe" part of the state space. In applications such as net execution and analysis the choice of a strategy will be guided by the need to explore the behaviour of the system under certain conditions. Strategies are well-supported by the Maude engine via reflection [19, 18], i.e. the capability to represent rewrite specifications as objects and control their execution at the meta-level, which makes Maude a suitable tool not only for executing place-transition nets but also for analyzing such nets using strategies for (partial) state-space exploration and model checking.

3.2 Rewriting Semantics in the General Case

The rewriting semantics that has been explained in terms of the banker's example in the previous section can be conceived as a functor from the category **PTN** of place/transitions nets to the category **SMRWS** of symmetric monoidal RWSs (SMRWSs) that will be introduced next. The characteristic feature of SMRWSs is that their underlying specification has a single rewrite kind `[Marking]` that is specified to be a free commutative monoid over a set of constants. The definition of SMRWSs given below is quite restrictive, but is sufficient for the rewriting semantics of PTNs. In Section 4.4 SMRWSs will be generalized to provide a rewriting semantics for nets with individual tokens.

Definition 15. A RWS $\mathcal{R}$ is a *symmetric monoidal RWS (SMRWS)* iff the following conditions are satisfied:

1. $\mathcal{S}^D_{\mathcal{R}}$ is empty.

2. $\mathcal{S}_{\mathcal{R}}$ contains precisely the following:
   (a) a kind `[Marking]` together with operator symbols

   $$\texttt{empty} : \to \texttt{[Marking]}, \quad \_\_ : \texttt{[Marking] [Marking]} \to \texttt{[Marking]};$$

   (b) any number of operator symbols of the general form

   $$p : \to \texttt{[Marking]};$$

   (c) the parallel composition axioms

   $$\forall\, u, v, w : \texttt{[Marking]}\ .\ u\ (v\ w) = (u\ v)\ w,$$
   $$\forall\, u, v : \texttt{[Marking]}\ .\ u\ v = v\ u,$$
   $$\forall\, u : \texttt{[Marking]}\ .\ \texttt{empty}\ u = u.$$

3. Rules in $R_{\mathcal{R}}$ do not have conditions and do not contain any variables.

Given two SMRWSs $\mathcal{R}$ and $\mathcal{R}'$, a SMRWS morphism $H : \mathcal{R} \to \mathcal{R}'$ is a RWS morphism that preserves `[Marking]`, `empty` and $\_\_$. SMRWSs together with their morphisms form a subcategory of **RWS** denoted **SMRWS**.

In order to obtain a precise definition of the initial model-theoretic semantics $\mathbf{I}(\mathcal{R})$ of a SMRWS $\mathcal{R}$, it is convenient to define the model-theoretic semantics of $\mathcal{R}$ by means of a MES $\mathbf{E}(\mathcal{R})$ which has a standard model-theoretic semantics in terms of $\mathbf{E}(\mathcal{R})$-algebras. Having done that, we then define $\mathbf{I}(\mathcal{R})$ as $\mathbf{I}(\mathbf{E}(\mathcal{R}))$, i.e., as the initial model of $\mathbf{E}(\mathcal{R})$.

Definition 16. The *membership equational presentation* of a SMRWS $\mathcal{R}$ is a MES $\mathbf{E}(\mathcal{R})$ that extends $\mathcal{S}_{\mathcal{R}}$, the underlying MES of $\mathcal{R}$, by the following:

1. a new kind `[RawProc]` together with new operator symbols called *proof constructors*

   $$\texttt{id} : \texttt{[Marking]} \to \texttt{[RawProc]},$$
   $$\_\_ : \texttt{[RawProc] [RawProc]} \to \texttt{[RawProc]},$$
   $$\_;\_ : \texttt{[RawProc] [RawProc]} \to \texttt{[RawProc]};$$

2. a new operator symbol called *atomic proof constructor*

   $$t :\to \texttt{[RawProc]}$$

   for each rule $t : M \to N$ in $R_{\mathcal{R}}$;
3. a kind `[Proc]` with a sort `Proc` and an operator symbol

   $$\_ : \_ \to \_\ :\ \texttt{[RawProc] [Marking] [Marking]} \to \texttt{[Proc]};$$

4. a membership axiom

$$t : M \to N$$

for each rule $t : M \to N$ in $R_{\mathcal{R}}$, where we introduce the notation

$$P : M \to N \quad \text{as a shorthand for} \quad (P : M \to N) : \texttt{Proc};$$

5. membership axioms corresponding to the standard inference rules of rewriting logic, namely:
   (a) *identity*:

   $$\texttt{id}(u) : u \to u$$

   (b) *composition*:

   $$\alpha;\beta : u_1 \to u_3 \ \texttt{if} \ \alpha : u_1 \to u_2 \ \wedge \ \beta : u_2 \to u_3$$

   (c) *compatibility* of parallel composition:

   $$\alpha_1 \ \alpha_2 : u_1 \ u_2 \to u_1' \ u_2' \ \texttt{if} \ \alpha_1 : u_1 \to u_1' \ \wedge \ \alpha_2 : u_2 \to u_2'$$

6. equational axioms corresponding to the standard rewriting logic axioms, namely:
   (a) *identity*:

   $$\texttt{id}(u);\alpha = \alpha \ \texttt{if} \ \alpha : u \to u'$$
   $$\alpha;\texttt{id}(u') = \alpha \ \texttt{if} \ \alpha : u \to u'$$

   (b) *associativity*:

   $$\alpha;(\beta;\gamma) = (\alpha;\beta);\gamma$$
   $$\texttt{if} \ \alpha : u_1 \to u_2 \ \wedge \ \beta : u_2 \to u_3 \ \wedge \ \gamma : u_3 \to u_4$$

   (c) *functoriality* of the parallel composition operator:

   $$\texttt{id}(u_1) \ \texttt{id}(u_2) = \texttt{id}(u_1 \ u_2)$$
   $$(\alpha_1;\beta_1)(\alpha_2;\beta_2) = (\alpha_1 \ \alpha_2);(\beta_1 \ \beta_2)$$
   $$\texttt{if} \ \alpha_1 : u_1 \to v_1 \ \wedge \ \beta_1 : v_1 \to w_1 \ \wedge$$
   $$\alpha_2 : u_2 \to v_2 \ \wedge \ \beta_2 : v_2 \to w_2$$

   (d) *inherited equations* for the parallel composition operator:

   $$\alpha_1 \ (\alpha_2 \ \alpha_3) = (\alpha_1 \ \alpha_2) \ \alpha_3$$
   $$\texttt{if} \ \alpha_1 : u_1 \to u_1' \ \wedge \ \alpha_2 : u_2 \to u_2' \ \wedge \ \alpha_3 : u_3 \to u_3'$$
   $$\alpha_1 \ \alpha_2 = \alpha_2 \ \alpha_1$$
   $$\texttt{if} \ \alpha_1 : u_1 \to u_1' \ \wedge \ \alpha_2 : u_2 \to u_2'$$
   $$\texttt{id}(\texttt{empty}) \ \alpha = \alpha \ \texttt{if} \ \alpha : u \to u'$$

For better readability we leave universal quantifiers implicit: $u, u', v, w, u_i, u'_i, v_i, w_i$ are distinct variables of kind `[Marking]` and $\alpha, \beta, \gamma, \alpha_i, \beta_i, \gamma_i$ are distinct variables of kind `[RawProc]`.

$\mathbf{E}$ can be extended to a functor $\mathbf{E} : \mathbf{SMRWS} \to \mathbf{MES}$ in the obvious way. Furthermore, composing $\mathbf{E} : \mathbf{SMRWS} \to \mathbf{MES}$ with the functor $\mathbf{Mod} : \mathbf{MES} \to \mathbf{Cat}^{\mathrm{op}}$ we obtain $\mathbf{Mod} \circ \mathbf{E} : \mathbf{SMRWS} \to \mathbf{Cat}^{\mathrm{op}}$ which is also denoted $\mathbf{Mod} : \mathbf{SMRWS} \to \mathbf{Cat}^{\mathrm{op}}$. As usual we write $\mathbf{U}_H$ for $\mathbf{Mod}(H)$ given a SMRWS morphism H.

In this paper we are not interested in the entire algebraic structure of SMRWS models. Instead, our first goal is to relate two different semantics of PTNs, namely, the Best-Devillers process semantics and the rewriting semantics of Definition 14, in terms of SMCs. In other words, the category $\mathbf{SMC}$ will serve as a common basis and suitable level of abstraction to compare different descriptions. Below, the initial models of SMRWSs, that are defined in terms of a functor $\mathbf{I}$, will be uniformly mapped into the same domain via a forgetful functor $\mathbf{V}$.

Definition 17. Let $\Sigma(\mathbf{Mod})$ be the Grothendiek construction for the functor $\mathbf{Mod} : \mathbf{SMRWS} \to \mathbf{Cat}^{\mathrm{op}}$ and let $\boldsymbol{\pi}_1 : \Sigma(\mathbf{Mod}) \to \mathbf{SMRWS}$ be the obvious projection functor that sends $(\mathcal{R}, A)$ to $\mathcal{R}$. Given a SMRWS $\mathcal{R}$ we define $\mathbf{I}(\mathcal{R})$ as $\mathbf{I}(\mathbf{E}(\mathcal{R}))$ and $\Sigma\mathbf{I}(\mathcal{R})$ as $(\mathcal{R}, \mathbf{I}(\mathcal{R}))$. Given a SMRWS morphism $H : \mathcal{R} \to \mathcal{R}'$ we define $\Sigma\mathbf{I}(H)$ as the morphism $(H, \mathbf{I}(H)) : (\mathcal{R}, \mathbf{I}(\mathcal{R})) \to (\mathcal{R}', \mathbf{I}(\mathcal{R}'))$ with $\mathbf{I}(H)$ the unique morphism $\mathbf{I}(H) : \mathbf{I}(\mathcal{R}) \to \mathbf{U}_H(\mathbf{I}(\mathcal{R}'))$ guaranteed by the fact that $\mathbf{I}(\mathcal{R})$ and $\mathbf{U}_H(\mathbf{I}(\mathcal{R}'))$ are objects in $\mathbf{Mod}(\mathcal{R})$ with the former being initial. In this way we have defined a functor $\Sigma\mathbf{I} : \mathbf{SMRWS} \to \Sigma(\mathbf{Mod})$ that is left adjoint to $\boldsymbol{\pi}_1$.

Let $\mathbf{V} : \Sigma(\mathbf{Mod}) \to \mathbf{SMC}$ be the forgetful functor which sends $(\mathcal{R}, \hat{A})$ to the SMC defined as follows: The sets of objects and arrows are $[\![\texttt{[Marking]}]\!]_{\hat{A}}$ and $[\![\texttt{Proc}]\!]_{\hat{A}}$, respectively. Arrow composition is $[\![\_;\_]\!]_{\hat{A}}$ and identities are $[\![\texttt{id}]\!]_{\hat{A}}(m)$ for $m \in [\![\texttt{[Marking]}]\!]_{\hat{A}}$. The monoidal operation and its identity are given by $[\![\_\,\_]\!]_{\hat{A}}$ and $[\![\texttt{empty}]\!]_{\hat{A}}$, respectively. Given a morphism $(H, h) : (\mathcal{R}, \hat{A}) \to (\mathcal{R}', \hat{A}')$ in $\Sigma(\mathbf{Mod})$ we define $\mathbf{V}(H, h)$ as the SMC morphism given by the obvious restriction of h.

The rewriting semantics of PTNs is then defined as follows:

Definition 18. Given a PTN $\mathcal{N}$ the *rewriting semantics* of $\mathcal{N}$ is the smallest SMRWS $\mathbf{R}(\mathcal{N})$ such that:

1. $\mathcal{S}_{\mathbf{R}(\mathcal{N})}$ contains a *token constructor*

   $p : \to$ `[Marking]`

   for each place $p \in P_{\mathcal{N}}$;

2. $\mathbf{R}(\mathcal{N})$ has a label t and a rule called a *transition rule*, namely,

$$t : \underbrace{p_1 \dots p_1}_{W(p_1,t)} \;\cdots\; \underbrace{p_m \dots p_m}_{W(p_m,t)} \rightarrow \underbrace{p_1 \dots p_1}_{W(t,p_1)} \;\cdots\; \underbrace{p_m \dots p_m}_{W(t,p_m)}$$

for each transition $t \in T_{\mathcal{N}}$ assuming $P_{\mathcal{N}} = \{p_1, \dots, p_m\}$ with distinct p_i.

$\mathbf{R}$ can be extended to a functor $\mathbf{R} : \mathbf{PTN} \rightarrow \mathbf{SMRWS}$ that maps each PTN morphism $H : \mathcal{N} \rightarrow \mathcal{N}'$ to the unique SMRWS morphism $G : \mathbf{R}(\mathcal{N}) \rightarrow \mathbf{R}(\mathcal{N}')$ with $G_L(t) = H(t)$ for each $t \in T_{\mathcal{N}}$ and $G_S(p) = H(p)$ for each $p \in P_{\mathcal{N}}$.

The main result in this section states that for a PTN $\mathcal{N}$ the Best-Devillers process semantics $\mathbf{BDP}(\mathcal{N})$ coincides with the initial semantics of $\mathbf{R}(\mathcal{N})$ in the strongest possible categorical sense of a natural isomorphism.

In fact, this theorem is closely related to and can be proved using a result in [21] (Theorem 27), which states that the monoidal category $\mathcal{CP}(\mathcal{N})$ of concatenable processes and a monoidal category $\mathcal{P}(\mathcal{N})$ defined by an inductive equational definition are isomorphic. Both $\mathcal{CP}(\mathcal{N})$ and $\mathcal{P}(\mathcal{N})$ are not symmetric, but they still enjoy certain symmetries. For an exact definition of $\mathcal{CP}(\mathcal{N})$ and $\mathcal{P}(\mathcal{N})$ we refer to [21].

The difference between Theorem 27 in [21] and Theorem 1 below is that: (1) Theorem 1 is about Best-Devillers processes which are more abstract than concatenable processes, (2) it uses rewriting logic instead of giving a direct inductive equational definition, and (3) it states a natural isomorphism instead of just an isomorphism, that is, we use not only categories in the small, but we also aim at a systematic categorical treatment in the large.

Theorem 1. There is a natural isomorphism $\hat{\tau} : \mathbf{BDP} \rightarrow \mathbf{V} \circ \Sigma\mathbf{I} \circ \mathbf{R}$ between the functors $\mathbf{BDP} : \mathbf{PTN} \rightarrow \mathbf{SMC}$ and $\mathbf{V} \circ \Sigma\mathbf{I} \circ \mathbf{R} : \mathbf{PTN} \rightarrow \mathbf{SMC}$ (with $\mathbf{R} : \mathbf{PTN} \rightarrow \mathbf{SMRWS}$ and $\mathbf{V} \circ \Sigma\mathbf{I} : \mathbf{SMRWS} \rightarrow \mathbf{SMC}$).

In particular, the previous theorem entails that for each individual PTN we have precisely characterized Best-Devillers processes in rewriting logic via $\mathbf{R}$ as stated by the corollary below. As a byproduct we have obtained a corresponding characterization in membership equational logic via $\mathbf{E}$.

Corollary 1. The rewrite specification $\mathbf{R}(\mathcal{N})$ provides a sound and complete axiomatization of the Best-Devillers processes of the PTN $\mathcal{N}$.

Again, this is closely related to Corollary 33 in [21], which states that the presentation of an SMC denoted by $\mathcal{T}(\mathcal{N})$ provides a complete and sound axiomatization of Best-Devillers processes. Similar to the category $\mathcal{P}(\mathcal{N})$ mentioned before, $\mathcal{T}(\mathcal{N})$ is given by a direct inductive equational definition, whereas here we use the SMRWS $\mathbf{R}(\mathcal{N})$ to express the same category. In other words we use rewriting logic to equip the presentation of $\mathcal{T}(\mathcal{N})$ itself with a first-class formal status.

3.3 Petri Nets with Test Arcs

In this section we illustrate how the techniques for giving a rewriting logic semantics to place/transition nets can be extended to deal with the important class of place/transition nets with *test arcs* [16, 59, 79, 15]. Petri nets have been equipped with test arcs (also called *read arcs*, or *positive contexts* in contextual nets [59]) to naturally model cases where a certain resource may be *read without being consumed* by a transition, such as in a database system where multiple users are allowed to simultaneously read the same piece of data. In contrast to ordinary arcs, several test arcs are allowed to access the same token in the same concurrent step, but a token accessed by a test arc may not be accessed by an ordinary arc in the same step.[5] Test arcs cannot change the marking of a place.

Formally, a place/transition net with test arcs $\mathcal{N}$ is a place/transition net together with a set of *test arcs* $TA_{\mathcal{N}} \subseteq P_{\mathcal{N}} \times T_{\mathcal{N}}$. We define the *context function* $\partial_{TA} : T_{\mathcal{N}}^{\oplus} \to \mathcal{M}$ on finite multisets e of transitions by $\partial_{TA}(e)(p) = 1$ if there is a transition $t \in e$ with $(p, t) \in TA_{\mathcal{N}}$, and by $\partial_{TA}(e)(p) = 0$ otherwise. The *step semantics* of a place/transition net with test arcs is defined as for place/transition nets (see Section 3) with the modification that for $m_1 \xrightarrow{e} m_2$ to hold we require additionally that, for each place $p \in P_{\mathcal{N}}$, $\partial_{TA}(e)(p) \leq m(p)$.

We propose a rewriting semantics for a place/transition net with test arcs, defined in terms of a rewrite specification $\mathbf{R}(\mathcal{N})$ similar to the one in Definition 18, but specifying tokens by means of a kind `[Place]` and two operators $\texttt{[\_]}, \langle \_ \rangle$: `[Place]` $\to$ `[Marking]` so that a token residing at place p is represented by the term $\texttt{[}p\texttt{]}$. An occurrence of $\texttt{[}p\texttt{]}$ may not be shared by more than one rewrite at the same time; to allow simultaneous rewrites with *read-only* access to a token at place p, we consider a token $\texttt{[}p\texttt{]}$ to be equivalent to an arbitrary number of read-only tokens of the form $\langle p \rangle$. This can be accomplished, using a technique described in [50], by adding to our specification $\mathbf{R}(\mathcal{N})$ an operator $\{\_ \mid \_\}$: `[Marking] [Nat]` $\to$ `[Marking]` and two "copying" axioms[6]

$$\texttt{[}p\texttt{]} = \{p \mid 0\} \quad \text{and} \quad \{p \mid n\} = \{p \mid n+1\}\,\langle p \rangle,$$

where p and n are variables ranging, respectively, over `[Place]` and `[Nat]`.

A transition t which consumes the tokens $a_1, \ldots, a_n$, produces the tokens $b_1, \ldots, b_m$, and "reads" the tokens $c_1, \ldots, c_k$, is modeled by a rewrite rule

$$t : \texttt{[}a_1\texttt{]} \ldots \texttt{[}a_n\texttt{]}\ \langle c_1 \rangle \ldots \langle c_k \rangle \longrightarrow \texttt{[}b_1\texttt{]} \ldots \texttt{[}b_m\texttt{]}\ \langle c_1 \rangle \ldots \langle c_k \rangle.$$

The database example in Figure 2, taken from [16], where multiple users may

[5] This last restriction is omitted in some definitions of Petri nets with test arcs (see e.g. [79]).

[6] The counting of the read-only copies and their read-only use guarantee that all the copies must have been "folded back together" in order for the original token to be engaged in a transition that consumes the token.

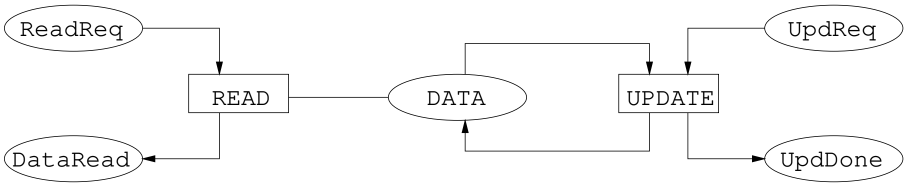

Fig. 2. Small database example using test arcs.

read some data simultaneously, but where only one at a time is allowed to update the data, is, therefore, modeled in rewriting logic by the following rules:

$$\begin{array}{ll} \texttt{READ}: & \texttt{[ReadReq]}\ \langle\texttt{Data}\rangle \longrightarrow \texttt{[DataRead]}\ \langle\texttt{Data}\rangle \\ \texttt{UPDATE}: & \texttt{[UpdReq]}\ \texttt{[Data]} \longrightarrow \texttt{[UpdDone]}\ \texttt{[Data]}. \end{array}$$

Let $\mathbf{R}(\mathcal{N})$ be the rewrite specification representing a place/transition net with test arcs $\mathcal{N}$ as explained above, and for any marking m in $\mathcal{N}$, let $m^\sharp$ denote the term of kind `[Marking]` which contains exactly $m(p)$ occurrences of the term $\texttt{[}p\texttt{]}$ for each place p in $\mathcal{N}$. Then, there is a step $m_1 \xrightarrow{e} m_2$ in $\mathcal{N}$ iff there is a one-step concurrent rewrite $\alpha : m_1^\sharp \longrightarrow m_2^\sharp$ in $\mathbf{R}(\mathcal{N})$, where, in addition, the step e can be extracted from the proof α. Furthermore, as in Definition 17 we can define a functor that associates with $\mathbf{R}(\mathcal{N})$ a symmetric monoidal category determined by the initial semantics. This provides a categorical semantics for all the concurrent computations of the net $\mathcal{N}$ that is closely related to the one recently proposed by Bruni and Sassone in [15].

4 High-Level Petri Nets

We use the term *high-level Petri nets* to refer to a range of extensions of PTNs by individual tokens, a line of research that has been initiated by the introduction of *predicate/transition nets* in [33, 34, 32]. High-level Petri nets make use of an underlying formalism, such as first-order logic in the case of predicate/transition nets, to describe the information that is associated with each token and its transformation. *Colored nets*[7] introduced in [38] are another quite general model of this kind with a more set-theoretic flavour. They generalize PTNs in such a way that tokens can be arbitrary set-theoretic objects. Quite different from, but closely related to, colored nets are high-level Petri nets that use an algebraic specification language as an underlying formalism [76, 6, 77, 67, 65, 66, 22, 5]. In this paper we subsume such approaches under the general notion of *algebraic net specifications*, parameterized over an underlying equational

[7] In fact, the nets introduced in [38] are called *colored Petri nets (CPNs)*, but this name has later been used for the more syntactic version introduced in [39], which is also the sense for which we would like to reserve this term (see below).

specification language. The main feature that algebraic net specifications have in common with predicate/transition nets is that an algebraic net specification does not necessarily specify a single colored net, but instead denotes a *class* of colored nets that satisfy the specification. In the following we first define colored nets, and then we introduce algebraic net specifications over MEL, a straightforward generalization of algebraic net specifications over many-sorted equational-logic (MSA). Both, algebraic net specifications and rewriting logic are specification formalisms that admit a variety of models. From an even more general point of view that is only briefly sketched in this paper, one can define colored net specifications parameterized over an underlying logic. In fact, predicate/transition nets can essentially be regarded as colored net specifications over first-order logic. From this more general point of view we restrict our attention in this paper to the particular class of colored net specifications over MEL, that we also call algebraic net specifications (over MEL), to establish a systematic connection to rewriting logic (over MEL). Later, in Section 6, we will discuss how other high-level Petri net extensions can be covered as generalizations or variants of our approach.

4.1 Colored Nets and Colored Net Specifications

Algebraic net specifications will be introduced later as a formal specification language for colored nets. In the following we define the most general set-theoretic version of colored nets [38]. We also give a suitable notion of colored net morphism and we use **CN** to abbreviate the resulting category of colored nets.

Colored nets are nets with places, transitions, and arcs inscribed with additional information given by functions C and W. The color set $C(p)$ of a place p is the set of possible objects p can carry. The color set $C(t)$ of a transition t can be seen as a set of modes in which t may occur. The arc inscription W defines a multiset of objects ("colored" tokens) that are transported by an arc when the associated transition occurs. In fact, this multiset may depend on the mode in which the transition occurs, which is why $W(p,t)$ and $W(t,p)$ take the form of functions in the definition below.

Definition 19. A *colored net (CN)* $\mathcal{N}$ consists of:

1. a finite net $N_{\mathcal{N}}$;
2. a set of *color sets* $CS_{\mathcal{N}}$;
3. a *color function* $C_{\mathcal{N}} : P_{\mathcal{N}} \cup T_{\mathcal{N}} \to CS_{\mathcal{N}}$; and
4. an *arc inscription* $W_{\mathcal{N}}$ on $F_{\mathcal{N}}$ such that

$$W_{\mathcal{N}}(p,t) : C_{\mathcal{N}}(t) \to C_{\mathcal{N}}(p)^{\oplus}, \text{ and } \; W_{\mathcal{N}}(t,p) : C_{\mathcal{N}}(t) \to C_{\mathcal{N}}(p)^{\oplus}.$$

$W_{\mathcal{N}}$ is extended to a function on $(P_{\mathcal{N}} \times T_{\mathcal{N}}) \cup (T_{\mathcal{N}} \times P_{\mathcal{N}})$ in such a way that $(p,t) \notin F_{\mathcal{N}}$ implies $W_{\mathcal{N}}(p,t)(b) = \emptyset$ and $(t,p) \notin F_{\mathcal{N}}$ implies $W_{\mathcal{N}}(t,p)(b) = \emptyset$ for each $b \in C_{\mathcal{N}}(t)$.

Let $\mathcal{N}$ and $\mathcal{N}'$ be CNs. A *CN morphism* $H : \mathcal{N} \to \mathcal{N}'$ consists of a net morphism $H_N : N_{\mathcal{N}} \to N_{\mathcal{N}'}$, and functions $H_x : C_{\mathcal{N}}(x) \to C_{\mathcal{N}'}(H_N(x))$ for each $x \in P_{\mathcal{N}} \cup T_{\mathcal{N}}$ such that:

1. $W_{\mathcal{N}'}(p', t')(H_t(b)) = H_{p_1}(W_{\mathcal{N}}(p_1, t)(b)) \oplus \ldots \oplus H_{p_n}(W_{\mathcal{N}}(p_n, t)(b))$ for all $p' \in P_{\mathcal{N}'}$, $t' \in T_{\mathcal{N}'}$, $t \in H_N^{-1}(t')$, $b \in C(t)$, and $\{p_1, \ldots, p_n\} = H_N^{-1}(p') \cap {}^{\bullet}t$ with distinct p_i;
2. $W_{\mathcal{N}'}(t', p')(H_t(b)) = H_{p_1}(W_{\mathcal{N}}(t, p_1)(b)) \oplus \ldots \oplus H_{p_n}(W_{\mathcal{N}}(t, p_n)(b))$ for all $p' \in P_{\mathcal{N}'}$, $t' \in T_{\mathcal{N}'}$, $t \in H_N^{-1}(t')$, $b \in C(t)$, and $\{p_1, \ldots, p_n\} = H_N^{-1}(p') \cap t^{\bullet}$ with distinct p_i.

CNs together with their morphisms form a category denoted by **CN**.

CNs generalize PTNs. The two dual objects of generalization are places and transitions. PTNs arise as the special case in which $C(x)$ is a singleton set for each $x \in P \cup T$. This gives rise to an obvious inclusion functor $\boldsymbol{\iota} : \mathbf{PTN} \to \mathbf{CN}$.

Although CNs can be seen as a generalization of PTNs, there is a more fundamental justification for introducing CNs, namely, that a CN is just a convenient abbreviation for a typically rather complex PTN [39, 32]. Indeed, this connection can be exploited to lift low-level concepts such as markings, safe processes, and Best-Devillers processes to the higher level. This is achieved by the following flattening functor $(\_)^{\flat} : \mathbf{CN} \to \mathbf{PTN}$ which associates to each CN the PTN obtained by "spatial unfolding." We call this operation *flattening* to clearly distinguish it from "temporal unfolding" which generates the processes of a PTN as we defined them earlier.

Definition 20. Given a CN $\mathcal{N}$, we define the *flattening* $\mathcal{N}^{\flat}$ of $\mathcal{N}$ as the unique PTN that satisfies:

1. $P_{\mathcal{N}^{\flat}} = \{(p, c) \mid p \in P_{\mathcal{N}}, c \in C_{\mathcal{N}}(p)\}$;
2. $T_{\mathcal{N}^{\flat}} = \{(t, b) \mid t \in T_{\mathcal{N}}, b \in C_{\mathcal{N}}(t)\}$;
3. $W_{\mathcal{N}^{\flat}}((p, c), (t, b)) = W_{\mathcal{N}}(p, t)(b)(c)$; and
4. $W_{\mathcal{N}^{\flat}}((t, b), (p, c)) = W_{\mathcal{N}}(t, p)(b)(c)$

for $p \in P_{\mathcal{N}}, c \in C_{\mathcal{N}}(p), t \in T_{\mathcal{N}}, b \in C_{\mathcal{N}}(t)$.

Flattening is extended to a functor $(\_)^{\flat} : \mathbf{CN} \to \mathbf{PTN}$ as follows: Given a CN morphism $H : \mathcal{N} \to \mathcal{N}'$, the PTN morphism $H^{\flat} : \mathcal{N}^{\flat} \to \mathcal{N}'^{\flat}$ is given by

1. $H^{\flat}((p, c)) = (H_N(p), H_p(c))$,
2. $H^{\flat}((t, b)) = (H_N(t), H_t(b))$

for $p \in P_{\mathcal{N}}$, $c \in C_{\mathcal{N}}(p)$, $t \in T_{\mathcal{N}}$, and $b \in C_{\mathcal{N}}(t)$.

It is important to point out that although we have defined the notion of a colored net, we have not yet introduced a notion of finite specification of colored nets. This is unsatisfactory if we want to reason about colored net specifications instead of just reasoning about colored nets. It is also unsatisfactory if we want to apply tools for execution, analysis and verification of colored nets, since such tools rely on a finitary, formal specification. Although a formal inscription language can be obtained by a formalization of set theory, such an enterprise is cumbersome and is of little help when we are interested in effective net execution and analysis. Also, the direct use of formalized set theory for specification and verification purposes is not very convenient and could be compared with the use of a low-level programming language.

Colored Petri nets, a more syntactic, finitary version of colored nets based on an underlying programming language, are proposed in [39]. A remarkable point is that this definition leaves open the particular choice of the underlying programming language. We use $\mathbf{CPN}_{\mathcal{L}}$ to abbreviate the class of colored Petri nets over a programming language $\mathcal{L}$. A quite well known instance of this definition is $\mathbf{CPN}_{\mathrm{ML}}$, the class supported by the execution and analysis tool Design/CPN [39] that employs the functional programming language ML. Appart from their operational flavor, the essential characteristic of colored Petri nets is that each colored Petri net denotes a single well-defined colored net in the above sense. A more logic-oriented view of colored nets (which emphasizes classes of models) is given by colored net specifications that are introduced subsequently.

As a useful concept, we informally introduce *colored net specifications (CNS)* which capture the essential idea shared by predicate/transition nets and algebraic net specifications, namely, that they denote an entire class of colored nets instead of just a single one. In fact, there is a general concept of CNSs that is parameterized by an underlying logic. A logic has a deductive system and a model-theoretic semantics, a concept that can be formalized by general logics [47] which contain institutions [35] as the model-theoretic component. We denote by $\mathbf{CNS}_{\mathcal{L}}$ the class of colored net specifications over the underlying logic $\mathcal{L}$. Possible candidates for $\mathcal{L}$ include equational logics such as many-sorted equational logic (MSA), order-sorted equational logic (OSA), or membership equational logic (MEL). We refer to CNSs over such equational logics also as *algebraic net specifications (ANS)*, and we denote by $\mathbf{ANS}_{\mathcal{L}}$ the class of algebraic net specifications over $\mathcal{L}$. Obviously, there are other possible choices for the underlying logic, such as full first-order logic (as in predicate/transition nets), a version of higher-order logic, or a higher-order algebraic specification language (as in [37]).

4.2 Algebraic Net Specifications

In the following we use the term *algebraic net specification (ANS)* to specifically refer to ANSs over MEL, since MEL is sufficiently expressive to cover other commonly used algebraic specification languages such as MSA and OSA [51]. The use of MEL is particularly attractive, because it is weak enough to admit initial models. Indeed, under the initial semantics (which can be internally specified using constraints in the data subspecification) an ANS denotes a unique

CN. Another benefit of the use of membership equational logic is that, under the restrictions mentioned in Section 2.1, it comes with a natural operational semantics (which is actually implemented in the Maude engine) so that it can be used directly as a programming language or, more generally, as a metalanguage to specify the logical and operational semantics of other specification or programming languages. As a consequence, colored Petri nets in $\mathbf{CPN}_{\mathcal{L}}$ which use $\mathcal{L}$ as a programming language can be seen as a special case of algebraic net specifications in $\mathbf{ANS}_{\mathrm{MEL}}$ if the semantics of $\mathcal{L}$ can be specified in MEL.

Due to the fact that MEL generalizes MSA in an obvious way, ANSs over MEL are a straightforward generalization of ANSs over MSA, i.e., many-sorted algebraic net specifications. Disregarding the issue of the underlying specification language, the definition we give below is equivalent to the one in [41, 43], generalizing [65] by so-called *flexible arcs*, which transport variable multisets of tokens in the sense that the number of tokens transported by an arc is not fixed but can depend on the mode in which the associated transition occurs. Later, in Section 4.3 we will illustrate by means of an example how an executable subset of the specification language can be used to obtain executable specifications of net models.

An ANS presupposes an underlying specification that has a multiset kind for each place domain. Hence we introduce a generic notion of multiset specification first.

Definition 21. A *MES of finite multisets* over a kind k consists of:

1. a MET having kinds k and $\texttt{[FMS}_k\texttt{]}$ with operator symbols

   $\texttt{empty}_k : \texttt{[FMS}_k\texttt{]}$,
   $\texttt{single} : k \to \texttt{[FMS}_k\texttt{]}$,
   $\_\,\_ : \texttt{[FMS}_k\texttt{]}\ \texttt{[FMS}_k\texttt{]} \to \texttt{[FMS}_k\texttt{]}$;

   equational axioms

   $\forall\ a, b, c : \texttt{[FMS}_k\texttt{]}\ .\ a\ (b\ c) = (a\ b)\ c$,
   $\forall\ a, b : \texttt{[FMS}_k\texttt{]}\ .\ a\ b = b\ a$,
   $\forall\ a : \texttt{[FMS}_k\texttt{]}\ .\ \texttt{empty}_k\ a = a$;
2. and a constraint stating that this theory is free over k.

To simplify notation we write M instead of $\texttt{single}(M)$. To further simplify the exposition we assume without loss of generality that $[\![\texttt{[FMS}_k\texttt{]}]\!] = [\![k]\!]^{\oplus}$, i.e., $\texttt{[FMS}_k\texttt{]}$ is interpreted in the standard way, and the operator symbols are interpreted accordingly.

The subsequent definition of algebraic net specifications should be regarded as an instance of CNSs over a logic $\mathcal{L}$ choosing MEL for $\mathcal{L}$. In fact, the only requirements that $\mathcal{L}$ has to meet is that it has a notion of type and that it is expressive enough to axiomatize multisets.

Definition 22. An *algebraic net specification (ANS)* $\mathcal{N}$ consists of:

1. a MES $\mathcal{S}_{\mathcal{N}}$;
2. a finite net $N_{\mathcal{N}}$;
3. a *place declaration*, i.e., a function $D_{\mathcal{N}} : P_{\mathcal{N}} \to K_{\mathcal{S}_{\mathcal{N}}}$ assigning a kind $D_{\mathcal{N}}(p)$ to each place $p \in P_{\mathcal{N}}$ such that $\mathcal{S}_{\mathcal{N}}$ includes a MES of finite multisets over $D_{\mathcal{N}}(p)$;
4. a *variable declaration*, i.e., a function $V_{\mathcal{N}}$ on $T_{\mathcal{N}}$ associating to each transition $t \in T_{\mathcal{N}}$ a kinded variable set $V_{\mathcal{N}}(t)$;
5. an *arc inscription*, i.e., a function $W_{\mathcal{N}}$ on $F_{\mathcal{N}}$ such that for $p \in P_{\mathcal{N}}$, $t \in T_{\mathcal{N}}$,
   (a) $(p, t) \in F_{\mathcal{N}}$ implies $W_{\mathcal{N}}(p, t) \in \mathbf{T}_{\mathcal{S}_{\mathcal{N}}}(V_{\mathcal{N}}(t))_{[\mathtt{FMS}_{D_{\mathcal{N}}(p)}]}$ and
   (b) $(t, p) \in F_{\mathcal{N}}$ implies $W_{\mathcal{N}}(t, p) \in \mathbf{T}_{\mathcal{S}_{\mathcal{N}}}(V_{\mathcal{N}}(t))_{[\mathtt{FMS}_{D_{\mathcal{N}}(p)}]}$;
6. a *guard definition*, i.e., a function $G_{\mathcal{N}}$ on $T_{\mathcal{N}}$ with $G_{\mathcal{N}}(t)$ being an $\mathcal{S}_{\mathcal{N}}$-condition over $V_{\mathcal{N}}(t)$.

$W_{\mathcal{N}}$ is extended to a function on $(P_{\mathcal{N}} \times T_{\mathcal{N}}) \cup (T_{\mathcal{N}} \times P_{\mathcal{N}})$ such that $(p, t) \notin F_{\mathcal{N}}$ implies $W_{\mathcal{N}}(p, t) = \mathtt{empty}_{D_{\mathcal{N}}(p)}$ and $(t, p) \notin F_{\mathcal{N}}$ implies $W_{\mathcal{N}}(t, p) = \mathtt{empty}_{D_{\mathcal{N}}(p)}$ for $p \in P_{\mathcal{N}}$ and $t \in T_{\mathcal{N}}$.

Let $\mathcal{N}$ and $\mathcal{N}'$ be ANSs. An *ANS morphism* $H : \mathcal{N} \to \mathcal{N}'$ consists of a MES morphism $H_{\mathcal{S}} : \mathcal{S}_{\mathcal{N}} \to \mathcal{S}_{\mathcal{N}'}$ of the underlying MESs, a net morphism $H_N : N_{\mathcal{N}} \to N_{\mathcal{N}'}$, and a function $H_V^t : V_{\mathcal{N}}(t) \to V_{\mathcal{N}'}(t)$ for each $t \in T_{\mathcal{N}}$ such that $x \in V_{\mathcal{N}}(t)_k$ implies $H_V^t(x) \in V_{\mathcal{N}'}(t)_{H_{\mathcal{S}}(k)}$ for $k \in K_{\mathcal{S}_{\mathcal{N}}}$ and the following conditions are satisfied:

1. $H_{\mathcal{S}}(D_{\mathcal{N}}(p)) = D_{\mathcal{N}'}(H_N(p))$ for each $p \in P_{\mathcal{N}}$;
2. $\mathcal{S}_{\mathcal{N}'} \models \forall\ V_{\mathcal{N}'}(t)\ .\ H_{\mathcal{S}}^t(G_{\mathcal{N}}(t)) \Rightarrow G_{\mathcal{N}'}(H_N(t))$ for each $t \in T_{\mathcal{N}}$;
3. $\mathcal{S}_{\mathcal{N}'} \models \forall\ V_{\mathcal{N}'}(t)\ .\ H_{\mathcal{S}}^t(G_{\mathcal{N}}(t)) \Rightarrow$
   $$W_{\mathcal{N}'}(p', t') = H_{\mathcal{S}}^t(W_{\mathcal{N}}(p_1, t)) \dots H_{\mathcal{S}}^t(W_{\mathcal{N}}(p_n, t))$$
   for all $p' \in P_{\mathcal{N}'}$, $t' \in T_{\mathcal{N}'}$, $t \in H_N^{-1}(t')$,
   and $\{p_1, \dots, p_n\} = H_N^{-1}(p') \cap {}^{\bullet}t$ with distinct p_i;
4. $\mathcal{S}_{\mathcal{N}'} \models \forall\ V_{\mathcal{N}'}(t)\ .\ H_{\mathcal{S}}^t(G_{\mathcal{N}}(t)) \Rightarrow$
   $$W_{\mathcal{N}'}(t', p') = H_{\mathcal{S}}^t(W_{\mathcal{N}}(t, p_1)) \dots H_{\mathcal{S}}^t(W_{\mathcal{N}}(t, p_n))$$
   for all $p' \in P_{\mathcal{N}'}$, $t' \in T_{\mathcal{N}'}$, $t \in H_N^{-1}(t')$,
   and $\{p_1, \dots, p_n\} = H_N^{-1}(p') \cap t^{\bullet}$ with distinct p_i;

where $H_{\mathcal{S}}^t : \mathbf{T}_{\mathcal{S}_{\mathcal{N}}}(V_{\mathcal{N}}(t)) \to \mathbf{T}_{\mathcal{S}_{\mathcal{N}'}}(V_{\mathcal{N}'}(t))$ is the common extension of $H_{\mathcal{S}}$ and H_V^t to terms. We assume for the above definition that validity $\models$ has been extended to first-order formulae in the standard way.

ANSs together with their morphisms form a category **ANS**.

A typical ANS admits several colored nets as models. Since we want to state our results for an arbitrary but fixed model we also consider interpreted ANSs, i.e., ANSs together with distinguished data models. We furthermore equip interpreted ANS with a notion of morphism that allows us to express simultaneous transformations at the level of the ANSs and at the level of the data models.

Definition 23. An *interpreted ANS* $\mathcal{IN} = (\mathcal{N}, A)$ consists of an ANS $\mathcal{N}$ and a $\mathcal{S}_{\mathcal{N}}$-algebra A. An *interpreted ANS morphism* $(H, h) : (\mathcal{N}, A) \to (\mathcal{N}', A')$ consists of an ANS morphism $H : \mathcal{N} \to \mathcal{N}'$ and an interpreted MES morphism $(H_{\mathcal{S}}, h) : (\mathcal{S}_{\mathcal{N}}, A) \to (\mathcal{S}_{\mathcal{N}'}, A')$. Interpreted ANSs together with their morphisms form a category **IANS**.

Interpreted ANSs are considerably richer than CNs, since they contain their specification together with a model equipped with a corresponding algebraic structure. In this sense they are similar to concrete predicate/transition nets [33, 34, 32] and algebraic high-level nets [25]. In fact, interpreted ANS, concrete predicate/transition nets [31] and algebraic-high-level nets [25] can be regarded as instances of a general notion of *interpreted CNSs*.[8] The transition from interpreted ANSs to CNs can be described by a forgetful functor as follows.

Definition 24. Given an interpreted ANS $(\mathcal{N}, A)$, the *CN semantics* of $(\mathcal{N}, A)$ is given by the CN $\mathbf{CN}(\mathcal{N}, A)$ defined as follows:

1. the underlying net $N_{\mathbf{CN}(\mathcal{N},A)}$ is precisely $N_{\mathcal{N}}$;
2. the color function $C_{\mathbf{CN}(\mathcal{N},A)}$ is defined by
   $C_{\mathbf{CN}(\mathcal{N},A)}(p) = [\![D_{\mathcal{N}}(p)]\!]_A$ for $p \in P_{\mathcal{N}}$ and
   $C_{\mathbf{CN}(\mathcal{N},A)}(t) = B_{\mathcal{N},A}(t)$ for $t \in T_{\mathcal{N}}$,
   where $B_{\mathcal{N},A}(t)$ is the set of *valid bindings* of $t \in T_{\mathcal{N}}$, i.e.,
   the set of assignments $\beta : V_{\mathcal{N}}(t) \to A$ satisfying $G_{\mathcal{N}}(t)$;
3. the set of color sets $CS_{\mathbf{CN}(\mathcal{N},A)}$ is the smallest set that contains all $C_{\mathbf{CN}(\mathcal{N},A)}(x)$ for $x \in P_{\mathcal{N}} \cup T_{\mathcal{N}}$; and
4. the arc inscription $W_{\mathbf{CN}(\mathcal{N},A)}$ is defined by
   $W_{\mathbf{CN}(\mathcal{N},A)}(p, t)(\beta) = [\![W_{\mathcal{N}}(p, t)]\!]_{A,\beta}$ and
   $W_{\mathbf{CN}(\mathcal{N},A)}(t, p)(\beta) = [\![W_{\mathcal{N}}(t, p)]\!]_{A,\beta}$
   for $p \in P_{\mathcal{N}}$, $t \in T_{\mathcal{N}}$ and assignments $\beta : V_{\mathcal{N}}(t) \to A$.

CN is extended to a functor $\mathbf{CN} : \mathbf{IANS} \to \mathbf{CN}$ that maps each morphism $(H, h) : (\mathcal{N}, A) \to (\mathcal{N}', A')$ to the morphism $G : \mathbf{CN}(\mathcal{N}, A) \to \mathbf{CN}(\mathcal{N}', A')$ satisfying $G_N = H_N$ and $G_x = h_{D_{\mathcal{N}}(x)}$ for $x \in P_{\mathcal{N}} \cup T_{\mathcal{N}}$.

We lift the flattening functor $(\_)^\flat : \mathbf{CN} \to \mathbf{PTN}$ to interpreted ANS, denoting also by $(\_)^\flat : \mathbf{IANS} \to \mathbf{PTN}$ the composition $(\_)^\flat \circ \mathbf{CN}$. Using flattening we furthermore lift $\mathbf{BDP} : \mathbf{PTN} \to \mathbf{SMC}$ by defining $\mathbf{BDP} : \mathbf{IANS} \to \mathbf{SMC}$ as $\mathbf{BDP} \circ (\_)^\flat$.

4.3 A Case Study

In the following we generalize the rewriting semantics from PTNs to ANSs. Before dealing with the general case we try to convey the main ideas using

[8] To be precise, arc inscriptions have to be restricted, since flexible arcs are not available in predicate/transition nets and algebraic high-level nets.

a distributed network algorithm as a running example, and we show how the rewriting semantics is obtained in this particular but typical case.

An algorithm which admits a very natural presentation as an algebraic net specification is the well-known echo algorithm, also called PIF algorithm (where PIF stands for propagation of information with feedback). The algebraic net model we use here has been developed and verified in [42].

Given a network of agents with bidirectional channels the echo problem can be informally described as follows. A distinguished agent initiates the transmission of a piece of information which should be propagated (possibly using other agents) to all agents participating in the network. After that the initiator should receive feedback about the succesful completion of this task, i.e., that each agent has received the information transmitted.

A possible solution to this problem is modeled by the algebraic net specification described below. To focus on the algorithm itself, the model abstracts from the concrete information that is transmitted. This information can be easily added by refining the messages without major changes to the algorithm.

We assume that the agents can be distinguished in terms of their identifiers, which are modeled by a sort `Id`. The network of agents is represented as a directed multigraph, i.e., as a finite multiset of (directed) channels, where each channel is a pair of agent identifiers. In the specification fragment below, `Pair` is the sort of pairs of identifiers and `FMS-Pair` is the sort of finite multisets over such pairs. Finite multisets are equationally axiomatized as discussed before. The obvious initiality and freeness constraints for `Id`, `FMS-Id`, `Pair`, and `FMS-Pair` can be specified using (parameterized) functional modules in Maude [19, 18], but for the sake of brevity we omit the details here.

```
sort Id FMS-Id
     Pair FMS-Pair .

op (_,_) : Id Id -> Pair .

op empty-Id : -> FMS-Id .
op single : Id -> FMS-Id .
op __ : FMS-Id FMS-Id -> FMS-Id
   [assoc comm id: empty-Id] .

op empty-Pair : -> FMS-Pair .
op single : Pair -> FMS-Pair .
op __ : FMS-Pair FMS-Pair -> FMS-Pair
   [assoc comm id: empty-Pair] .

var x y x' y' : Id .
var fmsp fmsp' : FMS-Pair .
var p p' : Pair .
```

To work with a concrete example we assume agent identifiers and a network as specified below. Actually, the algorithm is parametric in the choice of agent identifiers and in the network topology, the only assumptions being that there is a distinguished initiator and that the network is a strongly connected network with bidirectional channels. Again this parameterization could be made explicit in Maude by viewing the entire specification as a parameterized module which can be instantiated, for instance, by the following choices for `Id` and `network`.[9]

```
ops i a b c d e : -> Id .

op  sym : Id Id -> FMS-Pair .
eq  sym(x,y) = ((x,y) (y,x)) .

op  network : -> FMS-Pair .
eq  network = (sym(i,a) sym(i,b) sym(e,b) sym(e,d)
               sym(c,d) sym(c,i) sym(c,a) sym(a,b)) .
```

Now we equationally specify three auxiliary functions operating on finite multisets of pairs. The first one `_-_` removes one occurrence of a given pair from a multiset of pairs. The other functions `out` and `in` will be used with `network` as a first argument: `out(network,x)` denotes the multiset of messages to be sent to neighours of `x` and, correspondingly, `in(network,x)` denotes the multiset of messages to be received from neighbours of `x`.

```
op  _-_ : FMS-Pair Pair -> FMS-Pair .
eq  empty-Pair - p = empty-Pair .
eq  (p fmsp) - p = fmsp .
ceq (p' fmsp) - p = (p' (fmsp - p)) if p =/= p' .

op  in : FMS-Pair Id -> FMS-Pair .
eq  in(empty-Pair,y') = empty-Pair .
eq  in(((x,y) fmsp),y) = ((x,y) in(fmsp,y)) .
ceq in(((x,y) fmsp),y') = in(fmsp,y') if y =/= y' .

op  out : FMS-Pair Id -> FMS-Pair .
eq  out(empty-Pair,x') = empty-Pair .
eq  out(((x,y) fmsp),x) = ((x,y) out(fmsp,x)) .
ceq out(((x,y) fmsp),x') = out(fmsp,x') if x =/= x' .
```

This concludes the MES. We are now ready to define the ANS on top of it. Its inscribed net is depicted in Fig. 3. In the center we have a message pool `MESSAGES` modeling messages in transit. The net elements at the top model the

[9] If we were interested in (abstract) formal verification rather than (concrete) execution we would leave open the interpretation of `Id` and `network` and in this way obtain an ANS admitting a rich variety of quite different models.

activity of the initiating agent `i`, which is initially in a state `QUIET`, whereas the net elements at the bottom model the activities of all the remaining agents which are initially `UNINFORMED`. More precisely, the activities of initiators and non-initiators are the following:

- After the initiator `i` sends out a message to all its neighbours (transition `ISEND`) it will remain in the `WAITING` state until it receives an acknowledgement message from all its neighbours. If this happens, it will go into the `TERMINATED` state (transition `IRECEIVE`), i.e., the initiator has locally detected that all agents have received a message.
- After a non-initiator `x` receives a message from an agent `y`, it sends messages to all its neighbours except for `y` (transition `SEND`), and goes into a `PENDING` state, where it remembers that it is pending after receiving a message from `y`. As soon it receives acknowledgement messages from all neigbours except for `y` it goes into the `ACCEPTED` state (transition `RECEIVE`).

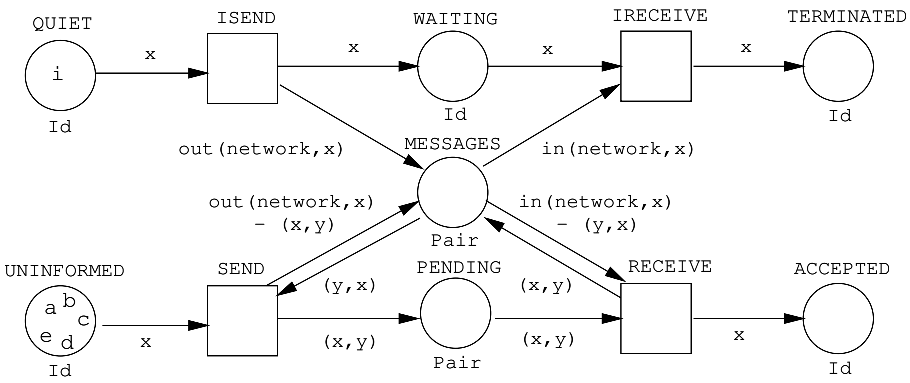

Fig. 3. Echo Algorithm

The initial marking specificaton m_0 for our concrete choice of the network is given by the terms inside places. It is

$$
\begin{array}{rlrl}
m_0(\texttt{QUIET}) &= \texttt{i} & m_0(\texttt{UNINFORMED}) &= \texttt{a b c d e} \\
m_0(\texttt{WAITING}) &= \texttt{empty-Id} & m_0(\texttt{PENDING}) &= \texttt{empty-Pair} \\
m_0(\texttt{TERMINATED}) &= \texttt{empty-Id} & m_0(\texttt{ACCEPTED}) &= \texttt{empty-Id}
\end{array}
$$

In Section 3 we have already discussed a rewriting semantics for the PTN of the banker's problem. Using the echo algorithm we will demonstrate how the rewriting semantics generalizes to ANSs. It is worth mentioning that our semantics is designed to cope with flexible arcs as the ones connected with the place `MESSAGES` in the echo algorithm.

The rewriting semantics associated to a RWS extends but does not modify the underlying MES of the net, the advantage being that properties established for the equational logic specification are preserved and their proofs remain valid.

As in the PTN case we represent a marking as an element of the kind `[Marking]`, which is equipped with a monoidal structure via the operations `empty` and `__`. For each place p we have a *token constructor*, also written as p, representing the fact that a single token resides in place p. A difference with respect to the PTN rewriting semantics is that tokens carry data, which is reflected in the fact that token constructors are functions instead of being constants. For instance, a token `MESSAGES`(*msg*) represents a token carrying the data *msg* residing in the place `MESSAGES`. So the token constructor can be seen as a function *tagging* a data object with information about the place in which it is currently located.

```
sort Marking .

op empty : -> Marking .
op __ : Marking Marking -> Marking
   [assoc comm id: empty] .

ops MESSAGES PENDING : Pair -> Marking .
ops QUIET WAITING TERMINATED UNINFORMED ACCEPTED : Id -> Marking .
```

When formulating the transition rule for `ISEND` we are faced with the problem of how to translate the flexible arc between `ISEND` and `MESSAGES` appropriately. Of course we would like to express that the multiset `out`(`network`, `x`) is added to the place `MESSAGES`, but this presupposes an interpretation of places as containers of objects which is different from our current one, where tokens are tagged objects "mixed up in a soup together with other tokens."

An elegant solution is the linear extension of `MESSAGES` to multisets. For this purpose we *generalize* the token constructor `MESSAGES` which has been declared above to

```
op MESSAGES : FMS-Pair -> Marking .
```

and we add two equations expressing linearity of `MESSAGES`, which will also be called *place linearity equations*:

```
seq MESSAGES(empty-Pair) = empty .
seq MESSAGES(fmsp fmsp') = MESSAGES(fmsp) MESSAGES(fmsp') .
```

The place linearity equations express the equivalence of different ways of looking at the same marking of an ANS. So, as indicated by the keyword `seq`, from a high-level specification point of view it is reasonable to assign them to the class of structural equations expressing symmetries of the state representation.

For reasons of uniformity we generalize the remaining token constructors correspondingly and we impose corresponding place linearity equations that we omit here.

Now the translation of transitions into rewrite rules can be done in full analogy with the rewriting semantics for PTN. Each transition is represented as a rewrite rule, also called a *transition rule*, replacing its preset marking by its postset marking. If the transition has a guard, then that guard becomes a condition of the rewrite rule. In this way we obtain the following rules:

```
rl  [ISEND]:    QUIET(x) =>
                WAITING(x) MESSAGES(out(network,x))) .

rl  [IRECEIVE]: WAITING(x) MESSAGES(in(network,x)) =>
                TERMINATED(x) .

rl  [SEND]:     UNINFORMED(x) MESSAGES((y,x)) =>
                PENDING((x,y)) MESSAGES(out(network,x)-(x,y)) .

rl  [RECEIVE]:  PENDING((x,y)) MESSAGES(in(network,x)-(y,x)) =>
                ACCEPTED(x) MESSAGES((x,y)) .
```

According to our initial explanation a place can be seen as the tag of an object which indicates the place the token resides in. This is what we call the *tagged-object view*. The place linearity equations suggest a complementary view which is encountered more often in the context of Petri nets: a place is simply a container of objects. We call this the *place-as-container view*. The place linearity equations express our intention to consider both views as equivalent.

4.4 Rewriting Semantics in the General Case

Generalizing the above example, we now define for an arbitrary ANS its associated rewriting semantics. We also show in which sense the rewriting semantics is equivalent to the Best-Devillers process semantics of ANSs, which we have defined by lifting the Best-Devillers process semantics of PTNs to ANSs via the flattening construction. First we generalize symmetric monoidal RWSs (SMRWSs) to extended symmetric monoidal RWSs (ESMRWSs), which will serve as a suitable domain for the rewriting semantics. Notice that in ESMRWSs the data subspecification is not required to be empty. A second difference w.r.t. SMRWSs is that token constructors are extended to multisets and place linearity equations are added.

Definition 25. A RWS $\mathcal{R}$ is an *extended symmetric monoidal RWS (ESMRWS)* iff the the following conditions are satisfied:

1. $\mathcal{S}_{\mathcal{R}}$ extends $\mathcal{S}^{D}_{\mathcal{R}}$ precisely by:

(a) a new kind `[Marking]` and new operator symbols

$$\texttt{empty} : \texttt{[Marking]}, \qquad \_\_ : \texttt{[Marking]}\ \texttt{[Marking]} \to \texttt{[Marking]};$$

(b) any number of new operator symbols of the general form

$$p : [\texttt{FMS}_k] \to \texttt{[Marking]},$$

where k is a kind in $\mathcal{S}^D_{\mathcal{R}}$ such that $\mathcal{S}^D_{\mathcal{R}}$ includes a MES of finite multisets over k;

(c) the axioms for parallel composition

$$\forall\ u, v, w : \texttt{[Marking]}\ .\ u\ (v\ w) = (u\ v)\ w,$$
$$\forall\ u, v : \texttt{[Marking]}\ .\ u\ v = v\ u,$$
$$\forall\ u : \texttt{[Marking]}\ .\ \texttt{empty}\ u = u; \quad \text{and}$$

(d) the *place linearity equations*

$$p(\texttt{empty}_k) = \texttt{empty},$$
$$\forall\ a, b : [\texttt{FMS}_k]\ .\ p(a\ b) = p(a)\ p(b)$$

for each operator $p : [\texttt{FMS}_k] \to \texttt{[Marking]}$ introduced above.

2. Rules in $R_{\mathcal{R}}$ contain only variables with kinds in $\mathcal{S}^D_{\mathcal{R}}$ and have $\mathcal{S}^D_{\mathcal{R}}$-conditions.

Given two ESMRWSs $\mathcal{R}$ and $\mathcal{R}'$, an ESMRWS morphism $H : \mathcal{R} \to \mathcal{R}'$ is a RWS morphism that preserves `[Marking]`, `empty` and $\_\_$. ESMRWSs together with their morphisms form a subcategory of **RWS** that is denoted by **ESMRWS**.

Definition 26. The *membership equational presentation* of an ESMRWS $\mathcal{R}$ is a MES $\mathbf{E}(\mathcal{R})$ that extends $\mathcal{S}_{\mathcal{R}}$, the underlying MES of $\mathcal{R}$, as explained in Definition 16, but modifying items 2 and 4 we have:

2. a new operator symbol called *atomic proof constructor*

$$t : \bar{k} \to \texttt{[RawProc]},$$

4. a membership axiom

$$\forall\ X\ .\ t(\bar{x}) : M \to N\ \ \texttt{if}\ \ \bar{\phi}_1 \wedge \ldots \wedge \bar{\phi}_n$$

for each rule $\forall\ X\ .\ t : M \to N\ \ \texttt{if}\ \ \bar{\phi}_1 \wedge \ldots \wedge \bar{\phi}_n$ in $R_{\mathcal{R}}$, assuming that $\bar{x} : \bar{k}$ is a canonical enumeration of the variables X.

As in Definition 16, $\mathbf{E}$ can be extended to a functor $\mathbf{E} : \mathbf{ESMRWS} \to \mathbf{MES}$ in the obvious way.

Definition 27. An *interpreted ESMRWS* $(\mathcal{R}, A)$ consists of an ESMRWS $\mathcal{R}$ and a $\mathcal{S}^D_{\mathcal{R}}$-model A. An interpreted ESMRWS morphism $(H, h) : (\mathcal{R}, A) \to (\mathcal{R}', A')$ consists of an ESMRWS morphism $H : \mathcal{R} \to \mathcal{R}'$ and an interpreted MES morphism $(H_D, h) : (\mathcal{S}^D_{\mathcal{R}}, A) \to (\mathcal{S}^D_{\mathcal{R}'}, A')$. Interpreted ESMRWSs together with their morphisms form a category **IESMRWS**.

Definition 28. For an interpreted ESMRWS $(\mathcal{R}, A)$ we define $\mathbf{Mod}(\mathcal{R}, A)$ as the subcategory of $\mathbf{Mod}(\mathcal{R})$ (i.e. $\mathbf{Mod}(\mathbf{E}(\mathcal{R}))$), with objects being $\mathcal{R}$-algebras (i.e. $\mathbf{E}(\mathcal{R})$-algebras) $\hat{A}$ satisfying $\hat{A}|\mathcal{S}^D_{\mathcal{R}} = A$. In fact, this gives rise to a functor $\mathbf{Mod} : \mathbf{IESMRWS} \to \mathbf{Cat}^{\mathrm{op}}$, and again we write $\mathbf{U}_H$ for $\mathbf{Mod}(H)$ given a ESMRWS morphism H.

Lemma 1 (Protection Lemma).

Let $(\mathcal{R}, A)$ be an interpreted ESMRWS and consider the obvious inclusion $K : \mathcal{S}^D_{\mathcal{R}} \hookrightarrow \mathbf{E}(\mathcal{R})$. Then $\eta_K(A) : A \to \mathbf{U}_K(\mathbf{F}_K(A))$ is an isomorphism.

To simplify the exposition assume that the free functor $\mathbf{F}_K$ has been defined in such a way that $\eta_K(A)$ becomes the identity and therefore $\mathbf{U}_K(\mathbf{F}_K(A)) = A$ for all $(\mathcal{R}, A)$ and K as above. The protection lemma ensures that this is possible without loss of generality.

$$\begin{tikzcd}
\mathcal{S}^D_{\mathcal{R}} \arrow[r, "K"] \arrow[d, "\mathbf{E}(H_D)"'] & \mathbf{E}(\mathcal{R}) \arrow[d, "\mathbf{E}(H)"] \\
\mathcal{S}^D_{\mathcal{R}} \arrow[r, "K'"] & \mathbf{E}(\mathcal{R}')
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathbf{Mod}(\mathcal{S}^D_{\mathcal{R}}) \arrow[r, shift left, "\mathbf{F}_K"] & \mathbf{Mod}(\mathcal{R}) \arrow[l, shift left, "\mathbf{U}_K"] \\
\mathbf{Mod}(\mathcal{S}^D_{\mathcal{R}}) \arrow[u, "\mathbf{U}_{H_D}"] \arrow[r, shift left, "\mathbf{F}_{K'}"] & \mathbf{Mod}(\mathcal{R}') \arrow[l, shift left, "\mathbf{U}_{K'}"] \arrow[u, "\mathbf{U}_H"']
\end{tikzcd}$$

Fig. 4. Morphisms in Definition 29

Definition 29. Let $\Sigma(\mathbf{Mod})$ be the Grothendiek construction for the functor $\mathbf{Mod} : \mathbf{IESMRWS} \to \mathbf{Cat}^{\mathrm{op}}$ and let $\boldsymbol{\pi}_1 : \Sigma(\mathbf{Mod}) \to \mathbf{IESMRWS}$ be the obvious projection functor that sends $((\mathcal{R}, A), \hat{A})$ to $(\mathcal{R}, A)$. Furthermore, let $(\mathcal{R}, A)$ and $(\mathcal{R}', A')$ be interpreted ESMRWSs and let $K : \mathcal{S}^D_{\mathcal{R}} \hookrightarrow \mathbf{E}(\mathcal{R})$ and $K' : \mathcal{S}^D_{\mathcal{R}'} \hookrightarrow \mathbf{E}(\mathcal{R}')$ be the obvious inclusions (cf. Fig. 4). We then define $\mathbf{F}(\mathcal{R}, A)$ as $\mathbf{F}_K(A)$ and $\Sigma\mathbf{F}(\mathcal{R}, A)$ as $((\mathcal{R}, A), \mathbf{F}(\mathcal{R}, A))$. Given an interpreted ESMRWS morphism $(H, h) : (\mathcal{R}, A) \to (\mathcal{R}', A')$ with $H : \mathcal{R} \to \mathcal{R}'$ and $h : A \to \mathbf{U}_{H_D}(A')$ we define $\Sigma\mathbf{F}(H, h)$ as the morphism $((H, h), \mathbf{F}(H) \circ \mathbf{F}_K(h)) : ((\mathcal{R}, A), \mathbf{F}_K(A)) \to ((\mathcal{R}', A'), \mathbf{F}_{K'}(A'))$ where $\mathbf{F}_K(h) : \mathbf{F}_K(A) \to \mathbf{F}_K(\mathbf{U}_{H_D}(A'))$ and $\mathbf{F}(H)$ is the unique morphism $\mathbf{F}(H) : \mathbf{F}_K(\mathbf{U}_{H_D}(A')) \to \mathbf{U}_H(\mathbf{F}_{K'}(A'))$ guranteed by the fact that $\mathbf{F}_K(\mathbf{U}_{H_D}(A'))$ and $\mathbf{U}_H(\mathbf{F}_{K'}(A'))$ are objects in $\mathbf{Mod}(\mathcal{R}, \mathbf{U}_{H_D}(A'))$, since using Lemma 1 we find $\mathbf{U}_K(\mathbf{F}_K(\mathbf{U}_{H_D}(A'))) = \mathbf{U}_{H_D}(A')$ and $\mathbf{U}_K(\mathbf{U}_H(\mathbf{F}_{K'}(A')))$ $= \mathbf{U}_{H_D}(\mathbf{U}_{K'}(\mathbf{F}_{K'}(A'))) = \mathbf{U}_{H_D}(A')$, and by the fact that $\mathbf{F}_K(\mathbf{U}_{H_D}(A'))$ is initial. In this way we have defined a functor $\Sigma\mathbf{F} : \mathbf{IESMRWS} \to \Sigma(\mathbf{Mod})$ that is left adjoint to $\boldsymbol{\pi}_1$.

Furthermore, let $\mathbf{V} : \Sigma(\mathbf{Mod}) \to \mathbf{SMC}$ be the forgetful functor, which sends $((\mathcal{R}, A), \hat{A})$ to a SMC, defined as in Def. 17.

Definition 30. Given an ANS $\mathcal{N}$, the *rewriting semantics* of $\mathcal{N}$ is the smallest ESMRWS $\mathbf{R}(\mathcal{N})$ with an underlying data specification $\mathcal{S}^D_{\mathbf{R}(\mathcal{N})} = \mathcal{S}_{\mathcal{N}}$ such that:

1. $\mathcal{S}_{\mathbf{R}(\mathcal{N})}$ contains a *token constructor*

   $$p : [\mathtt{FMS}_{D_{\mathcal{N}}(p)}] \to [\mathtt{Marking}]$$

   for each place $p \in P_{\mathcal{N}}$; and
2. $\mathbf{R}(\mathcal{N})$ has a label t and a rule called *transition rule*, namely,

   $$\begin{aligned} \forall\ V_{\mathcal{N}}(t)\ .\ t : (p_1(\,W_{\mathcal{N}}(p_1,t))\ \ldots\ p_m(\,W_{\mathcal{N}}(p_m,t))) \to \\ (p_1(\,W_{\mathcal{N}}(t,p_1))\ \ldots\ p_m(\,W_{\mathcal{N}}(t,p_m)))\ \mathtt{if}\ \ G_{\mathcal{N}}(t) \end{aligned}$$

   for each transition $t \in T_{\mathcal{N}}$, assuming $P_{\mathcal{N}} = \{p_1, \ldots, p_m\}$ with distinct p_i.

$\mathbf{R}$ can be extended to a functor $\mathbf{R} : \mathbf{ANS} \to \mathbf{ESMRWS}$ that maps each ANS morphism $H : \mathcal{N} \to \mathcal{N}'$ to the unique ESMRWS morphism $G : \mathbf{R}(\mathcal{N}) \to \mathbf{R}(\mathcal{N}')$ with $G_{\mathcal{S}}(p) = H_N(p)$ for each $p \in P_{\mathcal{N}}$ and $G_L(t) = H_N(t)$ for each $t \in T_{\mathcal{N}}$.

The functor $\mathbf{R} : \mathbf{ANS} \to \mathbf{ESMRWS}$ is naturally extended to a functor $\mathbf{R} : \mathbf{IANS} \to \mathbf{IESMRWS}$ sending each interpreted ANS $(\mathcal{N}, A)$ to the interpreted ESMRWS $(\mathbf{R}(\mathcal{N}), A)$. Furthermore, $\mathbf{R}$ sends each interpreted ANS morphism $(H, h) : (\mathcal{N}, A) \to (\mathcal{N}', A')$ to the interpreted ESMRWS morphism $(\mathbf{R}(H), h) : \mathbf{R}(\mathcal{N}, A) \to \mathbf{R}(\mathcal{N}', A')$.

Definition 31. Given an interpreted ESMRWS $(\mathcal{R}, A)$ we define the *flattening* of $(\mathcal{R}, A)$ as the smallest SMRWS $(\mathcal{R}, A)^\flat$ satisfying the following conditions:

1. For each operator $p : [\mathtt{FMS}_k] \to [\mathtt{Marking}]$ in $\mathcal{S}_{\mathcal{R}}$ and for each $a \in [\![k]\!]_A$ there is a constant $p^a : \to [\mathtt{Marking}]$ in $\mathcal{S}_{(\mathcal{R},A)^\flat}$.
2. For each rule $\forall\ X\ .\ t : M \to N$ $\mathtt{if}$ $\bar{\phi}_1 \wedge \ldots \wedge \bar{\phi}_n$ in $R_{\mathcal{R}}$ and for each assignment $\beta : X \to A$ with $A, \beta \models \bar{\phi}_1 \wedge \ldots \wedge \bar{\phi}_n$ we define functions σ and σ_p for each operator p as above by

   $$\sigma(\mathtt{empty}) = \mathtt{empty},\ \sigma(p(M)) = \sigma_p([\![M]\!]_{A,\beta})),\ \sigma(M\ N) = \sigma(M)\ \sigma(N),$$
   $$\sigma_p([\![\_\_]\!]([\![\mathtt{single}]\!](a_1), \ldots, [\![\mathtt{single}]\!](a_m))) = p^{a_1} \ldots p^{a_m}$$

   ($[\![\_\_]\!]$ is naturally extended to an arbitrary number of arguments) and we have a rule

   $$t^{\beta(\bar{x})} : \sigma(M) \to \sigma(N)$$

   to $R_{(\mathcal{R},A)^\flat}$, assuming that $\bar{x} : \bar{k}$ is a canonical enumeration of X.

$(\_)^\flat$ is extended to a functor $(\_)^\flat : \mathbf{IESMRWS} \to \mathbf{SMRWS}$ as follows: $(\_)^\flat$ sends each interpreted ESMRWS morphism $(H, h) : (\mathcal{R}, A) \to (\mathcal{R}', A')$ with $H : \mathcal{R} \to \mathcal{R}'$ and $h : A \to \mathbf{U}_{H_D}(A')$ to a SMRWS morphism $(H, h)^\flat : (\mathcal{R}, A)^\flat \to (\mathcal{R}', A')^\flat$ defined such that $(H, h)^\flat(p^a) = H_{\mathcal{S}}(p)^{h(a)}$ and $(H, h)^\flat(t^{\bar{a}}) = H_L(t)^{h(\bar{a})}$ for $\bar{a} = \beta(\bar{x})$ and all p, t, a, β, $\bar{x}$ as above.

The theorem and the corollary below are stated in complete analogy to the corresponding results for PTNs. Indeed the former results can be seen as special cases of the latter via an inclusion $\iota : \mathbf{PTN} \to \mathbf{ANS}$ which is the counterpart of $\iota : \mathbf{PTN} \to \mathbf{CN}$ on the specification level. However, for the proof we exploit the opposite direction, namely that Theorem 2 can be reduced to Theorem 1 via the flattening constructions introduced earlier. This can be done by a combination of commutative diagrams using the following two lemmas.

The first lemma essentially states that the rewriting semantics is compatible with flattening. Notice the overloading of $\mathbf{R}$ and $(\_)^\flat$.

Lemma 2. There is a natural isomorphism $\boldsymbol{\sigma} : (\_)^\flat \circ \mathbf{R} \to \mathbf{R} \circ (\_)^\flat$ between the functors $(\_)^\flat \circ \mathbf{R} : \mathbf{IANS} \to \mathbf{SMRWS}$ (with $\mathbf{R} : \mathbf{IANS} \to \mathbf{IESMRWS}$ and $(\_)^\flat : \mathbf{IESMRWS} \to \mathbf{SMRWS}$) and $\mathbf{R} \circ (\_)^\flat : \mathbf{IANS} \to \mathbf{SMRWS}$ (with $(\_)^\flat : \mathbf{IANS} \to \mathbf{PTN}$ and $\mathbf{R} : \mathbf{PTN} \to \mathbf{SMRWS}$).

The second lemma expresses that flattening preserves models at the level of abstraction given by SMCs.

Lemma 3. There is a natural isomorphism $\boldsymbol{\rho} : \mathbf{V} \circ \Sigma\mathbf{F} \to \mathbf{V} \circ \Sigma\mathbf{I} \circ (\_)^\flat$ between the functors $\mathbf{V} \circ \Sigma\mathbf{F} : \mathbf{IESMRWS} \to \mathbf{SMC}$ and $\mathbf{V} \circ \Sigma\mathbf{I} \circ (\_)^\flat : \mathbf{IESMRWS} \to \mathbf{SMC}$ (with $(\_)^\flat : \mathbf{IESMRWS} \to \mathbf{SMRWS}$ and $\mathbf{V} \circ \Sigma\mathbf{I} : \mathbf{SMRWS} \to \mathbf{SMC}$).

Now the main result follows from Lemma 2, Lemma 3 and Theorem 1:

$$
\begin{tikzcd}
\mathbf{ANS} \arrow[r, "\mathbf{R}"] & \mathbf{ESMRWS} & & \\
\mathbf{IANS} \arrow[u, "\pi_1"] \arrow[r, "\mathbf{R}"] \arrow[d, "\mathbf{CN}"] \arrow[dd, bend left, "(\_)^\flat"] \arrow[ddd, bend right, "\mathbf{BDP}"'] & \mathbf{IESMRWS} \arrow[u, "\pi_1"'] \arrow[r, "\Sigma\mathbf{F}"] \arrow[rr, bend right, "\mathbf{V}\circ\Sigma\mathbf{F}"'] \arrow[dd, "(\_)^\flat"] & \Sigma(\mathbf{Mod}) \arrow[r, "\mathbf{V}"] & \mathbf{SMC} \arrow[dd, equal] \\
\mathbf{CN} \arrow[d, "(\_)^\flat"] \arrow[r, phantom, "\cong\ \text{Lemma 2}"] & {} \arrow[rr, phantom, "\cong\ \text{Lemma 3}"] & & {} \\
\mathbf{PTN} \arrow[r, "\mathbf{R}"] \arrow[d, "\mathbf{BDP}"] & \mathbf{SMRWS} \arrow[r, "\Sigma\mathbf{I}"] \arrow[rr, bend right, "\mathbf{V}\circ\Sigma\mathbf{I}"'] \arrow[d, phantom, "\cong\ \text{Theorem 1}"] & \Sigma(\mathbf{Mod}) \arrow[r, "\mathbf{V}"] & \mathbf{SMC} \\
\mathbf{SMC} \arrow[rrru, equal, bend right] & {} & &
\end{tikzcd}
$$

Fig. 5. Proof of Theorem 2

Theorem 2. There is a natural isomorphism $\widetilde{\boldsymbol{\tau}} : \mathbf{BDP} \to \mathbf{V} \circ \Sigma\mathbf{F} \circ \mathbf{R}$ between the functors $\mathbf{BDP} : \mathbf{IANS} \to \mathbf{SMC}$ and $\mathbf{V} \circ \Sigma\mathbf{F} \circ \mathbf{R} : \mathbf{IANS} \to \mathbf{SMC}$ (with $\mathbf{R} : \mathbf{IANS} \to \mathbf{IESMRWS}$ and $\mathbf{V} \circ \Sigma\mathbf{F} : \mathbf{IESMRWS} \to \mathbf{SMC}$).

Proof By composition of natural isomorphisms (see Fig. 5). $\square$

In analogy to Corollary 1 we obtain:

Corollary 2. The interpreted RWS $\mathbf{R}(\mathcal{N}, A)$ provides a sound and complete axiomatization of the Best-Devillers processes of the interpreted ANS $(\mathcal{N}, A)$.

Remember that the models we consider here do not only contain Best-Devillers processes. They also contain safe processes as an important special case. Safe processes are not only a special case of the classical notion of process in Petri net theory, but they seem to be sufficient in practice as witnessed by [66], which presents a methodology for modeling and verification of distributed algorithms based on a version of ANSs that only admits safe processes.

Another related issue, namely the gap between the individual token philosophy and the collective token philosophy which clearly exists at the level of PTNs seems to become less relevant at the level of CNs, because of the increase of expressivity. We argue that interpreting CNs under the collective token philosophy is not only simpler and less dependent on the structure of the state space but also sufficient in principle, since by a suitable transformation of the CN we can equip tokens with unique identities in such a way that each original process corresponds to a safe process of the resulting CN.[10] As we discussed earlier, individual and collective token philosophies coincide for safe processes. Non-safe processes of the resulting CN are not considered any more. In this sense, the individual token philosophy can be seen as a special case of the collective token philosophy. Beyond that it may well be adequate for certain applications to mix the individual and the collective token views in the same system model, and indeed this is possible with the approach that we propose, namely by adopting the collective token semantics as a framework semantics and equipping tokens with additional identity attributes whenever needed for modeling purposes. Indeed this view reveals that individual and collective tokens semantics are just two extreme levels of abstraction and there are many intermediate levels that can be covered in this way. A good example of a very similar experience giving support to this point of view is the work [58] on a partial order semantics for object-oriented systems that, although typical of the individual token philosophy, is shown to be isomorphic to the rewriting semantics typical of the collective token philosophy, thanks to the unique identities of objects and messages.

[10] One policy to maintain unique identities is to encode the local history, i.e. the information about all events in the past cone of a token, in the identity of the token itself, and to ensure locally that the identities of the tokens produced by a transition are distinct. Of course, it is easy to imagine interesting classes of nets, e.g. object-oriented versions of high-level nets, where tokens are already equipped with unique identities so that this transformation is not needed at all.

4.5 Execution of Colored Net Specifications

First of all we lift the notion of executability from rewriting logic to net specifications. We say that a net specification is *weakly/strongly executable* iff its rewriting semantics is weakly/strongly executable. To actually execute a specification it is necessary to have an implementation of a matching algorithm for all combinations of structural equations used in the specification. A typical rewrite engine such as Maude supports matching modulo all combinations of the laws of associativity, commutativity and identity (ACU) [19, 18]. Since the place linearity equations belong to a class of equations that are typically not supported by standard rewrite engines we distinguish in the following between direct execution using ACUL-matching (L stands for linearity) and an alternative approach, namely execution via ACU-matching, which makes use of a simple semantics-preserving translation that can achieve executability without structural linearity equations.

Direct Execution via ACUL-Matching. It is easy to verify that the underlying MES in our example is already in executable form when the place linearity equations are seen as structural equations. Still the rewriting specification is not coherent and, as a consequence, the net specification is not executable as given.

A subterm of the form `in(network,x)` which occurs in the lefthand side of the rewrite axiom `IRECEIVE` can be reduced using the equations for `in`, so that the rewrite axiom is not applicable anymore. An obvious solution is to replace the arc inscription `in(network,x)` of the transition `IRECEIVE` by a variable `fmsmsg` and to add the guard `fmsmsg == in(network,x)` to this transition. A corresponding modification of the net specification has to be carried out for the transition `RECEIVE`. In the rewriting semantics these changes are reflected by the modified rewrite rules given below.

```
var fmsmsg : FMS-Message .

crl [IRECEIVE]: WAITING(x) MESSAGES(fmsmsg) =>
                TERMINATED(x)
    if fmsmsg == in(network,x) .

crl [RECEIVE]:  PENDING((x,y)) MESSAGES(fmsmsg) =>
                ACCEPTED(x) MESSAGES((x,y))
    if fmsmsg == in(network,x)-(y,x) .
```

After this simple semantics-preserving transformation the rewrite specification is indeed coherent and therefore strongly executable. To execute the RWS it is sufficient to use rewriting modulo associativity, commutativity, identity and linearity for the representation of markings.

Execution using ACU-Matching. We show in the following that, given an executable ANS such as the one we have just obtained, there is an alternative approach to net execution by regarding the place linearity equations as computational equations instead of as structural equations. Of course, from the viewpoint of the abstract algebraic semantics nothing will change. An immediate consequence is, however, that the net specification can be executed using a standard rewriting engine such as Maude, without the need for a new matching algorithm.

The first step is to regard the place linearity equations as reduction rules, i.e.,

```
eq  MESSAGES(empty-Pair) = empty .
ceq MESSAGES(fmsp fmsp') = (MESSAGES(fmsp) MESSAGES(fmsp'))
    if fmsp =/= empty-Pair and fmsp' =/= empty-Pair .
```

After applying this modification to all place linearity equations the reduction rules are terminating (the condition avoids potential non-terminating computations) and confluent, yielding an executable equational part of the specification.

However, as a consequence of the use of place-linearity equations as reduction rules instead of as structural equations, the rewrite specification is not coherent anymore, because of the rules for `IRECEIVE` and `RECEIVE` and the new equations above. Again, we can carry out a simple semantics-preserving translation by introducing a variable `mmsg` ranging over markings containing only tokens on `MESSAGES` and satisfying the equality condition `mmsg == MESSAGES(fmsmsg)`. By introducing the inverse `inv-MESSAGES` of `MESSAGES` this condition becomes `inv-MESSAGES(mmsg) == fmsmsg`. Therefore, `inv-MESSAGES(mmsg)` gives us access to the flexible arc inscription `fmsmsg`. As a result we replace these two rules by the following, which make the specification coherent and, hence, strongly executable:

```
sorts empty MESSAGES-Marking Marking .
subsorts empty < MESSAGES-Marking < Marking .

vars mmsg,mmsg' : MESSAGES-Marking .

op  empty : -> empty .
op  __ : Marking Marking -> Marking
    [assoc comm id: empty] .
op  __ : MESSAGES-Marking MESSAGES-Marking -> MESSAGES-Marking
    [assoc comm id: empty] .
op  __ : empty empty -> empty
    [assoc comm id: empty] .

op MESSAGES : FMS-Pair -> MESSAGES-Marking .

op  inv-MESSAGES : MESSAGES-Marking -> FMS-Pair .
```

```
eq  inv-MESSAGES(empty) = empty-Pair .
eq  inv-MESSAGES(MESSAGES(fmsp)) = fmsp .
ceq inv-MESSAGES(mmsg mmsg') =
    (inv-MESSAGES(mmsg) inv-MESSAGES(mmsg'))
    if mmsg =/= empty and mmsg' =/= empty .

crl [IRECEIVE]: WAITING(x) mmsg =>
                TERMINATED(x)
    if inv-MESSAGES(mmsg) == in(network,x) .

crl [RECEIVE]:  PENDING((x,y)) mmsg =>
                ACCEPTED(x) MESSAGES((x,y))
    if inv-MESSAGES(mmsg) == in(network,x)-(y,x) .
```

It should be clear from this example how the general translation works. It takes the form of a conservative theory transformation from the original RWS of an ANS executable by ACUL matching to a logically equivalent RWS executable by ACU matching. The transformation can be applied to any executable ANS satisfying the mild condition that flexible arcs are inscribed by variables, as it is the case in the executable version of the echo algorithm.[11]

Even though the resulting RWS is strongly executable, a strategy to execute the specification or to partially explore the state space can be useful, because of the highly nondeterminstic nature of the algorithm. A strategy of this kind can be seen as restricting the possible rewrites leading to a subcategory of the original category of all rewrites. If the RWS is only weakly executable, as in the example discussed in [70], the strategy can play an additional role, namely to find suitable instantiations for the variables that cannot be determined by matching.

5 Timed Petri Nets

This section illustrates how an important class of *timed* Petri nets can be given a rewriting logic semantics. Petri nets have been extended to model real-time systems in different ways (see e. g. [1, 60, 36]). Three of the most frequently used time extensions are the following [60], from which other timed versions of Petri nets can be obtained either as special cases or by combining the extensions:

1. Each *transition* t has an associated time interval $[l_t, u_t]$. A transition fires as soon as it can, but the resulting tokens are delayed, that is, when a transition t fires, the resulting tokens are produced after some time delay $r \in [l_t, u_t]$.
2. Each *place* p has a duration d_p. A token at place p cannot participate in a transition until it has been at p for at least time d_p.

[11] A more general transformation is possible if we use conditions with matching equations, a feature supported by the most recent version of Maude [20].

3. Each transition t is associated with a time interval $[l_t, u_t]$, and the transition t cannot fire before it has been continuously enabled for at least time l_t. Also, the transition t cannot have been enabled continuously for more than time u_t without being taken.

We will not treat the third case in this paper. We will instead cover the first two cases as special cases of the *interval timed colored Petri net* (ITCPN) model proposed by van der Aalst [1]. ITCPNs appear in the context of colored nets, but to simplify the exposition and focus on real-time features, we abstract from the colors of the tokens and instead have atomic tokens (with timestamps).

5.1 Interval Timed Petri Nets

We define a new model called *interval timed Petri nets* (ITPNs). Our model is similar to the interval timed colored Petri net model proposed in [1], but with two differences: (1) ITPNs ignore the coloring of the tokens, and (2) ITPNs have a notion of concurrent firing of multisets of transitions.

An ITPN is a PTN where the outgoing arcs are inscribed by time intervals denoting the range of possible firing delays of the produced tokens. The set TI of all time intervals, in a time domain $Time$, is the set $TI = \{ [r_1, r_2] \mid r_1, r_2 \in Time \wedge r_1 \leq r_2 \}$.

Definition 1. *An* interval timed Petri net *(ITPN) $\mathcal{N}$ is a PTN together with a* delay inscription *$D_{\mathcal{N}} : F_{\mathcal{N}} \cap (T_{\mathcal{N}} \times P_{\mathcal{N}}) \to TI^{\oplus}$ verifying $|D_{\mathcal{N}}(t,p)| = W_{\mathcal{N}}(t,p)$. The preset function $\partial_0 : T_{\mathcal{N}} \to P_{\mathcal{N}}^{\oplus}$ is defined, as for PTNs, by $\partial_0(t)(p) = W_{\mathcal{N}}(p,t)$, and the postset function $\partial_1^* : T_{\mathcal{N}} \to (P_{\mathcal{N}} \times TI)^{\oplus}$, where each resulting token is equipped with its delay interval, is defined by $\partial_1^*(t)(p, \Delta) = D_{\mathcal{N}}(t,p)(\Delta)$.*

The ITPN in Fig. 6 models a setting where each process performs transition `a`, followed by transition `b` within time 5 to 10, again followed by transition `a` time 4 to 8 thereafter, and so on. Furthermore, each process *forks* when performing transition `a` (this is modeled by having two arcs from a to q).

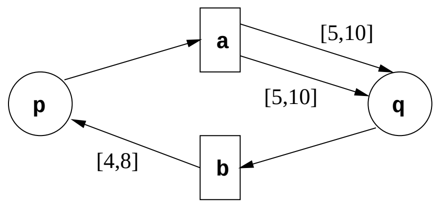

Fig. 6. An interval timed Petri net.

In the ITPN model, as in the ITCPN model, we attach a *timestamp* to each token. This timestamp indicates the time when a token becomes available. The *enabling time* of a transition is the maximum timestamp of the tokens to be consumed. Transitions are *eager* to fire (i.e., they fire as soon as possible), therefore the transition with the smallest enabling time will fire first. Firing is an atomic action, producing tokens with a timestamp equal to the firing time plus some *firing delay* specified by the delay inscription.

In the following, let $\mathcal{N}$ be an ITPN. The set of *markings* $\mathcal{M}_\mathcal{N} = (P_\mathcal{N} \times \mathit{Time})^\oplus$ is the set of all finite multisets of pairs (p, r) representing the presence of a token at place p with timestamp r. The function $\mathit{places} : \mathcal{M}_\mathcal{N} \to P_\mathcal{N}^\oplus$ which removes the timestamps from a marking is defined by $\mathit{places}(m)(p) = \Sigma_{(p,r)\in\mathcal{S}(m)}\, m(p, r)$. The function $\mathit{max} : (\mathcal{M}_\mathcal{N} - \{\emptyset\}) \to \mathit{Time}$ which finds the maximal timestamp in a non-empty marking is given by $\mathit{max}(m) = \max\{r \in \mathit{Time} \mid \exists p \,.\, (p, r) \in m\}$. The earliest enabling time of any transition in a marking is given by a function $\mathit{EET} : \mathcal{M}_\mathcal{N} \to \mathit{Time} \cup \{\infty\}$ defined by $\mathit{EET}(m) = \min\{\mathit{max}(m') \mid \exists t \in T_\mathcal{N}, m' \in \mathcal{M}_\mathcal{N} \,.\, m' \sqsubseteq m \wedge \mathit{places}(m') = \partial_0(t)\}$ with $\min(\emptyset) = \infty$. The function $\hat{+} : (P_\mathcal{N} \times \mathit{TI})^\oplus \times \mathit{Time} \to (P_\mathcal{N} \times \mathit{TI})^\oplus$ adds a delay to all the intervals in a multiset and is defined by $(m \hat{+} r)(p, [r_1 + r, r_2 + r]) = \Sigma_{(p,[r_1,r_2])\in\mathcal{S}(m)}\, m(p, [r_1, r_2])$, and $(m \hat{+} r)(p, [r', r'']) = 0$ if $r' < r$. Finally, to relate multisets of tokens with timestamps with multisets of tokens with time intervals, we define the specialization relation $\lhd \subseteq (P_\mathcal{N} \times \mathit{Time})^\oplus \times (P_\mathcal{N} \times \mathit{TI})^\oplus$, where $m \lhd m'$ holds if and only if each token in m corresponds to one token in m', such that they are in the same place and the timestamp of the token in m is in the interval of the corresponding token in m'. That is, $m \lhd m'$ if and only if either $(m = \emptyset \wedge m' = \emptyset)$ or $\exists (p, r) \in m,\ (p, [r_1, r_2]) \in m' \,.\, r_1 \le r \le r_2 \wedge (m - (p, r)) \lhd (m' - (p, [r_1, r_2]))$.

An ITPN makes computational progress by applying transitions, thereby consuming and producing multisets of timestamped tokens. A nonempty finite multiset of transitions firing at the same time constitutes a *(concurrent) step*. The *(concurrent) step semantics* of an ITPN $\mathcal{N}$ is given by the labelled transition system which has $\mathcal{M}_\mathcal{N}$ as states, $\mathcal{ST}_\mathcal{N} = T_\mathcal{N}^\oplus - \{\emptyset\}$ as steps, and where the transition relation $\to\ \subseteq \mathcal{M}_\mathcal{N} \times \mathcal{ST}_\mathcal{N} \times \mathcal{M}_\mathcal{N}$ is defined inductively by the following rules:

$$\frac{\mathit{places}(m) = \partial_0(t) \qquad \mathit{max}(m) = \mathit{EET}(m) \qquad m' \lhd \partial_1^*(t) \hat{+} \mathit{EET}(m)}{m \xrightarrow{t} m'}$$

$$\frac{m \xrightarrow{e} m' \qquad m'' \in \mathcal{M}_\mathcal{N} \qquad \mathit{EET}(m \oplus m'') = \mathit{EET}(m)}{(m \oplus m'') \xrightarrow{e} (m' \oplus m'')}$$

$$\frac{m_1 \xrightarrow{e_1} m_1' \qquad m_2 \xrightarrow{e_2} m_2' \qquad \mathit{EET}(m_1 \oplus m_2) = \mathit{EET}(m_1) = \mathit{EET}(m_2)}{(m_1 \oplus m_2) \xrightarrow{e_1 \oplus e_2} (m_1' \oplus m_2')}.$$

A *step sequence* in $\mathcal{N}$ is a (finite or infinite) sequence

$$\varsigma : \quad m_0 \stackrel{e_1}{\to} m_1 \stackrel{e_2}{\to} m_2 \stackrel{e_3}{\to} \cdots ,$$

where each step e_i represents the simultaneous firing of its transitions at time $EET(m_{i-1})$. The set of all step sequences of an ITPN $\mathcal{N}$ is denoted $\mathcal{C}_{\mathcal{N}}^{\infty}$.

Timed Petri nets of type (1) described above, where transitions have durations, can be seen as a special case of ITPNs as follows. A transition t with time interval $[l_t, u_t]$ which consumes the tokens m and produces the tokens m' can be simulated in an ITPN by adding a new place p_t, and having a transition t_1 which consumes the tokens m and produces one token at place p_t in some time in the interval $[l_t, u_t]$, and another transition t_2 which consumes one token from p_t and produces the tokens m' in zero time. Timed Petri nets of type (2), where each place has a duration d_p, corresponds to the special case of ITPNs where each token produced at place p has firing delay d_p.

5.2 Representing ITPNs in rewriting logic

Representing real-time theories in rewriting logic. We have proposed in [63] a framework for modeling real-time and hybrid systems in rewriting logic by means of *real-time rewrite theories*, and have shown that a number of well-known models of real-time and hybrid systems can naturally be specified as such theories. Essentially, a real-time rewrite theory should include a sort *Time* and an operator $\{\_\}$ which encloses the global state of the system and is used to ensure that time advances uniformly in all parts of a system. In addition to ordinary rewrite rules modeling instantaneous change in a system, a real-time rewrite theory may contain *tick rules* of the form $l : \; \{t\} \xrightarrow{\tau_l} \{t'\}$ **if** C, which model time elapse in a system, and where the term τ_l of sort *Time* denotes the duration of the rule. The total time elapse $\tau(\alpha)$ of a rewrite proof $\alpha : \{u\} \longrightarrow \{u'\}$ is defined as the sum of the time elapsed in each tick rule application in α. Even though it is useful to highlight the real-time aspects of a system using real-time rewrite theories, we have shown that, by adding an explicit clock, such theories are reducible to ordinary rewrite theories in a way that preserves all their expected properties.

In real-time systems, some actions are *eager*, that is, their application should take precedence over the application of time-advancing tick rules. We divide the rules of a real-time rewrite theory into *eager* and *lazy* rules, and define the *admissible rewrites* [63] to be the subset of all rewrites satisfying the additional requirement that a lazy rule may only be applied when no eager rule is applicable. The Real-Time Maude language and tool [61, 62] supports the specification and analysis of real-time rewrite theories, including the possibility to define eager and lazy rules.

Specifying ITPNs as real-time rewrite theories. The rewriting logic semantics of interval timed Petri nets is based on the rewriting logic semantics of untimed place/transition nets given in Section 3.2. For the sake of simplicity of the rewriting logic representation of ITPNs, we choose not to carry the timestamps in the tokens at all times. Instead, a term `[`p`]` of sort `VisibleMarking` represents a occurrence of a token at place p that is "visible", i.e., available for consumption. A token that *will be* visible at place p in time r is represented by the term `dly(`p, r`)`, which has the sort `DelayedMarking` whenever $r \neq 0$.[12] A token with delay 0 is visible, i.e., `dly(`$p, 0$`)` $=$ `[`p`]`. The sort `Marking` is a supersort of the sorts `VisibleMarking` and `DelayedMarking`, and denotes multisets of these two forms of tokens, where multiset union is represented by juxtaposition. The function `mte` takes a term of sort `DelayedMarking` and returns the time that can elapse until the next delayed token becomes visible. The function `delta` models the effect of the passage of time on delayed tokens by decreasing their delays according to the time elapsed.

The rewriting logic semantics of an ITPN $\mathcal{N}$ with $P_{\mathcal{N}} = \{p_1, \ldots, p_n\}$ is given by a real-time rewrite specification $\mathbf{R}(\mathcal{N})$, where the underlying MES contains an axiomatization of the sort `Time` of the time domain [63], a sort `TimeInf` for the time domain extended with ∞, together with the functions `+`, $\leq$, `min`, and $\dot{-}$ ("monus"), and the following declarations and axioms:

```
sorts Place EmptyMarking VisibleMarking DelayedMarking Marking System .
subsorts EmptyMarking < VisibleMarking DelayedMarking < Marking .

ops p1 ... pn : -> Place .
op [_] : Place -> VisibleMarking .
op dly : Place Time -> Marking .
op empty : -> EmptyMarking .
op __ : Marking Marking -> Marking [assoc comm id: empty] .
op __ : DelayedMarking DelayedMarking -> DelayedMarking
                                   [assoc comm id: empty] .
op __ : VisibleMarking VisibleMarking -> VisibleMarking
                                   [assoc comm id: empty] .
op __ :  EmptyMarking EmptyMarking -> EmptyMarking
                                   [assoc comm id: empty] .
op {_} : Marking -> System .
op delta : DelayedMarking Time -> Marking .
op mte : DelayedMarking -> TimeInf .

vars DM DM' : DelayedMarking .  var VM : VisibleMarking .
var P : Place .  vars X Y : Time .
cmb dly(P, X) : DelayedMarking if X =/= 0 .
eq dly(P, 0) = [P] .
eq delta(empty, X) = empty .
eq delta(dly(P, X), Y) = dly(P, X ∸ Y) .
ceq delta(DM DM', X) = delta(DM, X) delta(DM', X)
```

[12] We will see later that no interesting information about time is lost by this simplification, since the time when a firing of a transition occurs can always be extracted from the proof term.

```
       if DM =/= empty and DT' =/= empty .
eq mte(empty) = ∞ .
ceq mte(dly(P, X)) = X if X =/= 0 .
ceq mte(DM DM') = min(mte(DM), mte(DM'))
       if DM =/= empty and DM' =/= empty .
```

The rewrite semantics of an ITPN $\mathcal{N}$ is a real-time rewrite theory $\mathbf{R}(\mathcal{N})$ whose signature Ω and axioms E define the sort `Marking` and the functions `delta` and `mte`. The set of rules of $\mathbf{R}(\mathcal{N})$ consists of a *lazy* tick rule modeling time elapse and, for each transition t in $T_{\mathcal{N}}$, an *eager* rule

$$
\begin{aligned}
t : \; & \overbrace{\texttt{[}p_1\texttt{]} \;\ldots\; \texttt{[}p_1\texttt{]}}^{W(p_1,t)} \;\ldots\; \overbrace{\texttt{[}p_n\texttt{]} \;\ldots\; \texttt{[}p_n\texttt{]}}^{W(p_n,t)} \longrightarrow \\
& \underbrace{\texttt{dly(}p_1, x_{1,1}\texttt{)} \;\ldots \texttt{dly(}p_1, x_{1,W(t,p_1)}\texttt{)}}_{W(t,p_1)} \;\ldots\; \underbrace{\texttt{dly(}p_n, x_{n,1}\texttt{)} \;\ldots \texttt{dly(}p_n, x_{n,W(t,p_n)}\texttt{)}}_{W(t,p_n)} \\
& \textbf{if } (l_{1,1} \leq x_{1,1} \leq u_{1,1}) \wedge \ldots \wedge (l_{1,W(t,p_1)} \leq x_{1,W(t,p_1)} \leq u_{1,W(t,p_1)}) \wedge \ldots \\
& \wedge (l_{n,1} \leq x_{n,1} \leq u_{n,1}) \wedge \ldots \wedge (l_{n,W(t,p_n)} \leq x_{n,W(t,p_n)} \leq u_{n,W(t,p_n)})
\end{aligned}
$$

where $P_{\mathcal{N}} = \{p_1, \ldots, p_n\}$, with p_i distinct, $D(t, p_i)$ is the multiset $\{[l_{i,1}, u_{i,1}], \ldots, [l_{i,W(t,p_i)}, u_{i,W(t,p_i)}]\}$ for each $p_i \in P_{\mathcal{N}}$, and the $x_{i,j}$'s are distinct variables of sort *Time*. The following lazy tick rule advances time until the first delayed token becomes visible:

$$\texttt{tick} : \; \{\texttt{VM DM}\} \overset{\texttt{mte(DM)}}{\longrightarrow} \{\texttt{VM delta(DM,mte(DM))}\} \textbf{ if } \texttt{mte(DM)} \neq \infty.$$

For example, the translation of the ITPN in Fig. 6 contains the above tick rule and the following two instantaneous eager rules:

$$
\begin{aligned}
&\texttt{a} : \; \texttt{[p]} \longrightarrow \texttt{dly(q,X)}\;\; \texttt{dly(q,Y)} \textbf{ if } (5 \leq \texttt{X} \leq 10) \wedge (5 \leq \texttt{Y} \leq 10) \\
&\texttt{b} : \; \texttt{[q]} \longrightarrow \texttt{dly(p,X)} \textbf{ if } 4 \leq \texttt{X} \leq 8.
\end{aligned}
$$

The tick rule only needs to compute the time until the next delayed token becomes visible and advances time by that amount. After such a tick, the tick rule is again enabled but, due to its being lazy, it will not be applied if the new visible token(s) enable some transition(s) (whose firing in turn could immediately trigger further instantaneous transitions).

Since a step $m \xrightarrow{e} m'$ of an ITPN does not depend on the firing delays of the individual transitions taken in the step, two *one-step* rewrites $\{\alpha\} : u \longrightarrow v$ and $\{\beta\} : u \longrightarrow v$ should be considered equal — in the sense that we add the equivalence $t(r_1, \ldots, r_n) = t(r'_1, \ldots, r'_n)$, for each $t \in T_{\mathcal{N}}$, as a further equality identifying rewrite proofs — if the multisets of rule labels in α and β are the same. A *timed computation* in $\mathbf{R}(\mathcal{N})$ is a finite or infinite sequence

$$\tilde{\varsigma} : \; u_0 \xrightarrow{\gamma_1;\delta_1} u_1 \xrightarrow{\gamma_2;\delta_2} u_2 \xrightarrow{\gamma_3;\delta_3} \cdots$$

with admissible rewrite proofs $\gamma_i;\delta_i : u_{i-1} \longrightarrow u_i$ in $\mathbf{R}(\mathcal{N})$, such that each γ_i corresponds either to the identity proof or to a sequence of tick applications, each δ_i corresponds to a one-step concurrent rewrite using instantaneous rules, and u_0 is a term $\{w_0\}$ with w_0 a term of sort `Marking`. The set of timed computations in $\mathbf{R}(\mathcal{N})$ is denoted $\mathcal{C}(\mathbf{R}(\mathcal{N}))$. It follows from the factorization property of proofs in rewriting logic [48] that each non-identity admissible ground rewrite $\alpha : \{w\} \longrightarrow \{w'\}$ in $\mathbf{R}(\mathcal{N})$, is equivalent to a rewrite $\gamma;\delta$, such that γ can be rearranged as a finite timed computation, and δ corresponds to the identity proof or to a sequence of tick applications. Furthermore, each (infinite) computation of $\mathbf{R}(\mathcal{N})$, consisting of admissible rewrites involving ground terms of sort `System`, which contains an infinite number of applications of instantaneous rules, can be rearranged as a timed computation.

The fact that ITPNs are faithfully represented in their rewriting logic semantics is made precise in the theorem below, which can be used as the basis of a method to execute and analyze ITPNs in a tool such as Real-Time Maude [61, 62].

Theorem 3. Let $\mathcal{N}$ be an ITPN. Then, there is a bijective function $\tilde{(\_)} : \mathcal{C}^{\infty}_{\mathcal{N}} \to \mathcal{C}(\mathbf{R}(\mathcal{N}))$ taking a step sequence of the form ς to an timed computation of the form $\tilde{\varsigma}$ (see above for ς and $\tilde{\varsigma}$) such that:

- Each u_i is a term equivalent to a term of the form $\{w_i\}$, which consists of $m_i(p,r)$ occurrences of the term `dly`$(p, r \dot{-} \tau(\gamma_1;\delta_1;\ldots;\gamma_i;\delta_i))$, for all p and r (recall that `dly(`p`, 0)` is equivalent to `[`p`]`).
- The transitions fire at the same time in ς and $\tilde{\varsigma}$, that is, $\tau(\gamma_1;\delta_1;\ldots;\gamma_{i+1}) = EET(m_i)$.
- The transitions taken (concurrently) in each step are the same. That is, each δ_i is equivalent to a proof term of the form $\{\varepsilon_i\}$, where ε_i is a term containing, for each $t \in T_{\mathcal{N}}$, exactly $e_i(t)$ occurrences of proof terms of the form $t(r_1, \ldots, r_n)$.

6 Conclusions

In this paper we have explained in detail how rewriting logic can be used as a semantic framework in which a wide range of Petri net models can be naturally unified. Specifically, we have explored how place/transition nets, nets with test arcs, algebraic net specifications, colored Petri nets, and timed Petri nets can all be naturally expressed in rewriting logic, and how well-known semantic models often coincide with (in the sense of being naturally isomorphic to) the natural semantic models associated to the rewriting logic representations of the given nets. Space limitations do not allow us to explain in detail how other classes of Petri nets could similarly be treated. However, we sketch below a number of extensions of the ideas presented here that could deal with some of these.

A question that deserves some discussion is how colored Petri nets based on higher-order programming languages such as ML can be formally represented and, furthermore, how can they be related to the approach to ANSs presented in this paper. One possible answer is to translate each colored Petri net over a possibly higher-order language $\mathcal{L}$ into an ANS with an initial semantics. This reduces the problem to finding a translation of $\mathcal{L}$ into membership equational logic. The main problem with embedding a higher-order language into a first-order framework is the treatment of bound variables and there are different solutions. Recently, we have developped CINNI [69, 71], a new calculus of names and explicit substitutions, to solve this problem in a systematic way, and we have applied it to obtain executable embeddings of languages such as the lambda calculus and Abadi and Cardelli's object calculus into membership equational logic.

The step from higher-order programming languages to higher-order specification languages can be regarded in some instances as a move from typed lambda calculi to higher-order logics. The use of a specification language with higher-order capabilities seems to be not only attractive for enhancing the modeling and abstraction capabilities, but it can also provide a framework for extensions of algebraic specifications by initiality and freeness constraints [35] or first-order axioms such as those used in [66]. Recent experience with representing an entire family of pure type systems in rewriting logic [71] indicates that, using rewriting logic as a metalanguage, typed lambda calculi and higher-order logics can be naturally expressed. By viewing membership equational logic as a sublogic of a higher-order logic the approach presented in this paper, including the important aspect of executability, naturally extends to the higher-order case.

Apart from generalizations of the underlying specification or programming language, there is another potential source of Petri net generalizations, namely, the structure of the state space. Instead of considering a flat state space as in ordinary Petri nets we could choose a hierarchical one, or we could consider extensions such as *macroplaces* [2, 3] that can be seen as combining several places into a single one from the viewpoint of certain transitions. Also, we could consider different kinds of places. For instance, we could distinguish ordinary high-level places that carry a multiset of tokens from places that are organized as a queue or as a stack. The former idea has been studied in the literature in terms of *FIFO-nets* [30, 40, 29, 27]. Rewriting logic seems to be a suitable formalism to represent and unify such variations of Petri nets, since the state space can be specified by an equational theory that is entirely user-definable. A related approach that allows some freedom in the choice of the state space algebra and specializes to different low-level and high-level Petri net classes is presented in [24]. The approach of rewriting logic has the advantage of being more general, in the sense that it goes beyond Petri-net-like models and hence provides a bridge to formalisms that are quite different from ordinary Petri nets.

Yet another interesting generalization of Petri nets are different variants of *object Petri nets* [73, 74, 28, 75], where tokens can themselves be nets with their own dynamic behaviour. A quite different line of research is the integration of object-oriented techniques with Petri nets. As a result there are a number of variants of object-oriented Petri nets [68, 44], where the tokens are objects according to standard object-oriented terminology. As a unifying generalization of both approaches we propose a notion of *active token nets*. In contrast to object Petri nets, tokens can not only be nets but arbitrary objects with an internal dynamic behaviour. In constrast to *object-oriented approaches to Petri nets*, active tokens are not static but dynamic entities. In particular, they can evolve concurrently with the overall system behaviour, and they can also interact or communicate with each other. It might appear that the complexity of such models is beyond the scope of a rigorous formal treatment. However, a closer look reveals that the approach to Petri nets via rewriting logic is closer to the ideas described above than it might appear at the first sight. In fact, our approach can be easily generalized to active token nets by essentially replacing the underlying MES of a net by a RWS. So far we have employed rewrite rules only to represent transitions of the net. In order to describe tokens with internal activity we could use rewrite rules that transform individual tokens. To capture group activity such as interaction (which corresponds to synchronous communication) and asynchronous communication we have to add rewrite rules that operate on a group of tokens. One possible realization is to view tokens as objects in the sense of the rewriting logic approach to concurrent object-oriented programming [49], where rewrite rules operate on a multiset of objects that are interrelated by object references. As we have already pointed out, recent work on partial-order semantics for object-oriented systems specified in rewriting logic in this manner [58] is in fact very close to the safe process semantics of high-level Petri nets.

In our view, the unification of Petri net models within the rewriting logic framework is useful not only for conceptual reasons, but also for purposes of execution, formal analysis, and formal reasoning about Petri net specifications. Using the reflective and metalanguage capabilities of Maude, it is possible to build execution environments for Petri net specifications where the language description provided by the user and the user interaction could all take place at the Petri net level with which the user is familiar. Similarly, the Real-Time Maude tool [62] could offer corresponding capabilities for executing and analyzing timed Petri net models. Indeed, there is already a substantial body of experience and case studies on the Maude executable specification and formal analysis of consurrent and distributed systems in general (see [52]) and of Petri nets in particular [70, 63], suggesting that the Maude-based development of execution and formal analysis environments for Petri nets is a clear and promising practical application of the conceptual unification of Petri net models suggested in this paper.

Acknowledgements. Support by DARPA through Rome Laboratories Contract F30602-C-0312 and NASA through Contract NAS2-98073, by Office of Naval Research Contract N00014-99-C-0198, and by National Science Foundation Grant CCR-9900334 is gratefully acknowledged. Part of this work is based on earlier work conducted by the first author at University of Hamburg, Germany and in the scope of the European Community project MATCH (CHRX-CT94-0452). Furthermore, we would like to thank

Roberto Bruni, Narciso Martí-Oliet, Rüdiger Valk and the referees for their constructive criticism and many helpful suggestions.

References

1. W. M. P. van der Aalst. Interval timed coloured Petri nets and their analysis. In M. A. Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 453–472. Springer-Verlag, 1993.
2. N. A. Anisimov. An algebra of regular macronets for formal specification of communication protocols. *Computers and Artificial Intelligence*, 10(6):541–560, 1991.
3. N. A. Anisimov, K. Kishinski, A. Miloslavski, and P. A. Postupalski. Macroplaces in high level Petri nets: Application for design inbound call center. In *Proceedings of the Int. Conference on Information System Analysis and Synthesis (ISAS'96), Orlando, FL, USA*, pages 153–160, July 1996.
4. A. Asperti. A logic for concurrency. unpublished manuscript, November 1987.
5. E. Battiston, F. De Cindio, and G. Mauri. OBJSA nets: a class of high-level nets having objects as domains. In G. Rozenberg, editor, *Advances in Petri Nets*, volume 340 of *Lecture Notes in Computer Science*. Springer-Verlag, 1988.
6. B. Berthomieu, N. Choquet, C. Colin, B. Loyer, J. M. Martin, and A. Mauboussin. Abstract data nets: Combining Petri nets and abstract data types for high level specifications of distributed systems. In *Proc. of the Seventh Workshop on Applications and Theory of Petri Nets, Oxford, UK*, pages 25–48, 1986.
7. E. Best and R. Devillers. Sequential and concurrent behaviour in Petri net theory. *Theoretical Computer Science*, 55:87–136, 1987.
8. E. Best and C. Fernandez. *Nonsequential Processes—A Petri Net View*, volume 13 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1988.
9. A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theoretical Computer Science*, 236:35–132, 2000.
10. C. Brown. Relating Petri nets to formulae of linear logic. Technical Report ECS-LFCS-89-87, Laboratory of Foundations of Computer Science, University of Edinburgh, June 1989.
11. C. Brown and D. Gurr. A categorical linear framework for Petri nets. In *Proc. Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 208–218, June 1990.
12. R. Bruni, J. Meseguer, U. Montanari, and V. Sassone. A comparison of Petri net semantics under the collective token philosophy. In J. Hsiang and A. Ohori, editors, *Proceedings of ASIAN'98, 4th Asian Computing Science Conference*, volume 1538 of *Lecture Notes in Computer Science*, pages 225–244. Springer-Verlag, 1998.
13. R. Bruni, J. Meseguer, U. Montanari, and V. Sassone. Functorial semantics for Petri nets under the individual token philosophy. In *Proc. Category Theory and Computer Science, Edinburgh, Scottland, September 1999*, volume 29 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1999. http://www.elsevier.nl/locate/entcs/volume29.html.
14. R. Bruni, J. Meseguer, U. Montanari, and V. Sassone. Functorial semantics for petri nets under the individual token philosophy. In *Electronic Notes in Theoretical Computer Science: Proceedings of CTCS'99, 8th Conference on Category Theory and Computer Science*, volume 29, pages 1–19. Elsevier Science, 1999.
15. R. Bruni and V. Sassone. Algebraic models for contextual nets. In U. Montanari, J. D. P. Rolim, and E. Welzl, editors, *Automata, Languages and Programming. Proceedings 2000.*, volume 1853. Springer-Verlag, 2000.

16. S. Christensen and N. D. Hansen. Coloured Petri nets extended with place capacities, test arcs and inhibitor arcs. In M. A. Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, 1993.
17. M. Clavel. Reflection in general logics and in rewriting logic, with applications to the Maude language. Ph.D. Thesis, University of Navarre, 1998.
18. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. A tutorial on Maude. SRI International, March 2000, http://maude.csl.sri.com.
19. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. *Maude: Specification and Programming in Rewriting Logic*. Computer Science Laboratory, SRI International, Menlo Park, 1999. http://maude.csl.sri.com.
20. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. Towards Maude 2.0. In K. Futatsugi, editor, *Third International Workshop on Rewriting Logic and its Applications (WRLA'2000), Kanazawa, Japan, September 18 - 20, 2000*, volume 36 of *Electronic Notes in Theoretical Computer Science*, pages 297 – 318. Elsevier, 2000. http://www.elsevier.nl/locate/entcs/volume36.html.
21. P. Degano, J. Meseguer, and U. Montanari. Axiomizing the algebra of net computations and processes. *Acta Informatica*, 33:641–667, 1996.
22. C. Dimitrovici, U. Hummert, and L. Petrucci. Semantics, composition and net properties of algebraic high-level nets. In G. Rozenberg, editor, *Advances in Petri Nets 1991*, volume 524 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.
23. F. Durán and J. Meseguer. Structured theories and institutions. In M. Hofmann, G. Rosolini, and D. Pavlović, editors, *Proceedings of CTCS'99, 8th Conference on Category Theory and Computer Science, Edinburgh, Scotland, U.K., September 10-12, 1999*, volume 29, pages 71–90. Elsevier, 1999. http://www.elsevier.nl/locate/entcs/volume29.html.
24. H. Ehrig and J. Padberg. Uniform approach to Petri nets. In C. Freksa, M. Jantzen, and R. Valk, editors, *Foundations of Computer Science: Potential – Theory – Cognition*, volume 1337 of *Lecture Notes in Computer Science*, pages 219–231. Springer-Verlag, August 1997.
25. H. Ehrig, J. Padberg, and L. Ribeiro. Algebraic high-level nets: Petri nets revisited. In *Recent Trends in Data Type Specification*, volume 785 of *Springer-Verlag*, pages 188–206, 1994.
26. U. Engberg and G. Winskel. Petri nets as models of linear logic. In A. Arnold, editor, *CAAP'90*, volume 431 of *Lecture Notes in Computer Science*, pages 147–161. Springer-Verlag, 1990.
27. J. Fanchon. FIFO-net models for processes with asynchronous communication. In G. Rozenberg, editor, *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 152–178. Springer-Verlag, 1992.
28. B. Farwer. A linear logic view of object Petri nets. *Fundamenta Informaticae*, 37(3):225–246, 1999.
29. A. Finkel and A. Choquet. FIFO nets without order deadlock. *Acta Informatica*, 25(1):15–36, 1988.
30. A. Finkel and G. Memmi. FIFO nets: New model of parallel computation. In *6th GI-Conference on Theoretical Computer Science, Dortmund*, volume 145 of *Lecture Notes in Computer Science*, pages 111–121. Springer-Verlag, 1982.
31. H. J. Genrich. Equivalence transformation of PrT-nets. In G. Rozenberg, editor, *Advances in Petri Nets 1989*, volume 424, pages 179–208. Springer-Verlag, 1990.

32. H. J. Genrich. Predicate/transition nets. In *High-Level Petri Nets: Theory and Practice*, pages 3–43. Springer-Verlag, 1991.
33. H. J. Genrich and K. Lautenbach. The analysis of distributed systems by means of predicate/transition-nets. In G. Kahn, editor, *Semantics of Concurrent Computation*, volume 70 of *Lecture Notes in Computer Science*, pages 123–146, Berlin, 1979. Springer-Verlag.
34. H. J. Genrich and K. Lautenbach. System modelling with high-level Petri nets. *Theoretical Computer Science*, 13:109–136, 1981.
35. J. Goguen and R. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.
36. H. M. Hanisch. Analysis of place/transition nets with timed arcs and its application to batch process control. In M. A. Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 282–299. Springer-Verlag, 1993.
37. K. Hoffmann. Run time modification of algebraic high level nets and algebraic higher order nets using folding and unfolding construction. In G. Hommel, editor, *Communication-Based Systems, Proceedings of the 3rd International Workshop held at the TU Berlin, Germany, 31 March – 1 April 2000*, pages 55–72. Kluwer Academic Publishers, 2000.
38. K. Jensen. Coloured Petri nets and the invariant-method. *Theoretical Computer Science*, pages 317–336, 1981.
39. K. Jensen. *Coloured Petri Nets, Basic Concepts, Analysis Methods and Practical Use.*, volume 1 of *EATCS monographs on theoretical computer science*. Springer-Verlag, 1992.
40. E. Kettunen, E. Montonen, and T. Tuuliniemi. Comparison of Pr-net based channel models. In *Proc. of the 12th IMACS World Conf.*, volume 3, pages 479–482, 1988.
41. E. Kindler and W. Reisig. Algebraic system nets for modelling distributed algorithms. *Petri Net Newsletter*, (51):16–31, December 1996.
42. E. Kindler, W. Reisig, H. Völzer, and R. Walter. Petri net based verification of distributed algorithms: An example. *Formal Aspects of Computing*, 9:409–424, 1997.
43. E. Kindler and H. Völzer. Flexibility in algebraic nets. In J. Desel and M. Silva, editors, *Application and Theory of Petri Nets 1998, 19th International Conference, ICATPN'98, Lisbon, Portugal, June 1998, Proceedings*, volume 1420 of *Lecture Notes in Computer Science*, pages 345–384. Springer-Verlag, 1998.
44. C. A. Lakos. From coloured Petri nets to object Petri nets. In M. Diaz G. De Michelis, editor, *Application and Theory of Petri Nets*, volume 935 of *Lecture Notes in Computer Science*, pages 278–297, Berlin, 1995. Springer-Verlag.
45. N. Martí-Oliet and J. Meseguer. From Petri nets to linear logic. *Mathematical Structures in Computer Science*, 1:69–101, 1991.
46. N. Martí-Oliet and J. Meseguer. From Petri nets to linear logic through categories: A survey. *International Journal of Foundations of Computer Science*, 2(4):297–399, 1991.
47. J. Meseguer. General logics. In H.-D. Ebbinghaus et al., editors, *Proceedings, Logic Colloquium, 1987*, pages 275–329. North-Holland, 1989.
48. J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96:73–155, 1992.
49. J. Meseguer. A logical theory of concurrent objects and its realization in the Maude language. In G. Agha, P. Wegner, and A. Yonezawa, editors, *Research Directions in Concurrent Object-Oriented Programming*. MIT Press, 1993.

50. J. Meseguer. Rewriting logic as a semantic framework for concurrency: a progress report. In U. Montanari and V. Sassone, editors, *Proc. Concur'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 331–372. Springer-Verlag, 1996.
51. J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *Recent Trends in Algebraic Development Techniques, 12th International Workshop, WADT '97, Tarquinia, Italy, June 3-7, 1997, Selected Papers*, volume 1376 of *Lecture Notes in Computer Science*, pages 18 – 61. Springer-Verlag, 1998.
52. J. Meseguer. Rewriting logic and Maude: a wide-spectrum semantic framework for object-based distributed systems. In S. Smith and C.L. Talcott, editors, *Formal Methods for Open Object-based Distributed Systems, FMOODS 2000*, pages 89–117. Kluwer, 2000.
53. J. Meseguer and U. Montanari. Petri nets are monoids. *Information and Computation*, 88(2):105–155, October 1990.
54. J. Meseguer, U. Montanari, and V. Sassone. On the semantics of Petri nets. In W.R. Cleaveland, editor, *Proceedings of the Concur'92 Conference, Stony Brook, New York, August 1992*, volume 630 of *Lecture Notes in Computer Science*, pages 286–301. Springer-Verlag, 1992.
55. J. Meseguer, U. Montanari, and V. Sassone. On the model of computation of place/transition Petri nets. In *Proceedings 15th International Conference on Application and Theory of Petri Nets*, volume 815 of *Lecture Notes in Computer Science*, pages 16–38. Springer-Verlag, 1994.
56. J. Meseguer, U. Montanari, and V. Sassone. Process versus unfolding semantics for place/transition Petri nets. *Theoretical Computer Science*, 153(1–2):171–210, 1996.
57. J. Meseguer, U. Montanari, and V. Sassone. Representation theorems for Petri nets. In C. Freska, M. Jantzen, and R. Valk, editors, *Foundations of Computer Science: Potential, Theory, Cognition*, volume 1337 of *Lecture Notes in Computer Science*, pages 239–249. Springer-Verlag, 1997.
58. J. Meseguer and C. Talcott. A partial order event model for concurrent objects. In *Proc. CONCUR'99, Eindhoven, The Netherlands, August 1999*, volume 1664 of *Lecture Notes in Computer Science*, pages 415–430. Springer-Verlag, 1999.
59. U. Montanari and F. Rossi. Contextual nets. *Acta Informatica*, 32:545–596, 1995.
60. S. Morasca, M. Pezzè, and M. Trubian. Timed high-level nets. *The Journal of Real-Time Systems*, 3:165–189, 1991.
61. P. C. Ölveczky. *Specification and Analysis of Real-Time and Hybrid Systems in Rewriting Logic*. PhD thesis, University of Bergen, 2000. Available at http://maude.csl.sri.com/papers.
62. P. C. Ölveczky and J. Meseguer. Real-Time Maude: A tool for simulating and analyzing real-time and hybrid systems. In *Third International Workshop on Rewriting Logic and its Applications*, 2000. To appear in *Electronic Notes in Theoretical Computer Science*.
63. P. C. Ölveczky and J. Meseguer. Specification of real-time and hybrid systems in rewriting logic. To appear in *Theoretical Computer Science*. Available at http://maude.csl.sri.com/papers, September 2000.
64. C. A. Petri. Nets, time and space. *Theoretical Computer Science*, 153(1–2):3–48, 1996.
65. W. Reisig. Petri nets and algebraic specifications. *Theoretical Computer Science*, 80:1–34, 1991.
66. W. Reisig. *Elements of Distributed Algorithms: Modeling and Analysis with Petri Nets*. Springer-Verlag, 1998.

67. W. Reisig and J. Vautherin. An algebraic approach to high level Petri nets. In *Proceedings of the Eighth European Workshop on Application and Theory of Petri Nets*, pages 51–72. Universidad de Zaragoza (Spain), 1987.
68. C. Sibertin-Blanc. Cooperative nets. In R. Valette, editor, *Application and Theory of Petri Nets*, volume 815 of *Lecture Notes in Computer Science*, pages 471–490, Berlin, 1994. Springer-Verlag.
69. M.-O. Stehr. CINNI - A Generic Calculus of Explicit Substitutions and its Application to lambda-, sigma- and pi-calculi. In K. Futatsugi, editor, *Third International Workshop on Rewriting Logic and its Applications (WRLA'2000), Kanazawa, Japan, September 18 - 20, 2000*, volume 36 of *Electronic Notes in Theoretical Computer Science*, pages 71 – 92. Elsevier, 2000. http://www.elsevier.nl/locate/entcs/volume36.html.
70. M.-O. Stehr. A rewriting semantics for algebraic nets. In C. Girault and R. Valk, editors, *Petri Nets for Systems Engineering – A Guide to Modelling, Verification, and Applications*. Springer-Verlag, 2001. To appear.
71. M.-O. Stehr and J. Meseguer. Pure type systems in rewriting logic. In *Proc. of LFM'99: Workshop on Logical Frameworks and Meta-languages, Paris, France, September 28, 1999.*
72. A. Tarlecki, R. M. Burstall, and J. A. Goguen. Some fundamental algebraic tools for the semantics of computation, III: indexed categories. *Theoretical Computer Science*, 91:239–264, 1991.
73. R. Valk. Petri nets as dynamical objects. In *Workshop Proc. 16th International Conf. on Application and Theory of Petri Nets, Torino, Italy*, June 1995.
74. R. Valk. Petri nets as token objects: An introduction to elementary object nets. In J. Desel and M. Silva, editors, *Proceedings of the 19th International Conference on Application and Theory of Petri Nets, Lissabon, June 22-26, 1998*, volume 1420 of *Lecture Notes in Computer Science*, pages 1 – 25. Springer-Verlag, 1998.
75. R. Valk. Relating Different Semantics for Object Petri Nets. Technical report, FBI-HH-B-266/00, Fachbereich Informatik, Universität Hamburg, 2000.
76. J. Vautherin. *Un Modele Algebrique, Base sur les Reseaux de Petri, pour l'Etude des Systemes Paralleles.* These de Docteur Ingenieur, Univ. de Paris-Sud, Centre d'Orsay, June 1985.
77. J. Vautherin. Parallel systems specifications with coloured Petri nets and algebraic specifications. *Lecture Notes in Computer Science: Advances in Petri Nets 1987*, 266:293–308, 1987.
78. P. Viry. Rewriting: An effective model of concurrency. In C. Halatsis, D. Maritsas, G. Philokyprou, and S. Theodoridis, editors, *PARLE'94 – Parallel Architectures and Languages Europe, 6th Int. PARLE Conf. Athes, Greece, July 1994, Proceedings.*, volume 817 of *Lecture Notes in Computer Science*, pages 648–660. Springer-Verlag, 1994.
79. W. Vogler. Partial order semantics and test arcs. In *Proc. MFCS'97*, volume 1295 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.