Transactions and Zero-Safe Nets[⋆]

Roberto Bruni and Ugo Montanari

Dipartimento di Informatica, Università di Pisa, Italia
`{bruni,ugo}@di.unipi.it`

Abstract When employing Petri nets to model distributed systems, one must be aware that the basic activities of each component can vary in duration and can involve smaller internal activities, i.e., that transitions are conceptually refined into *transactions.* We present an approach to the modeling of transactions based on *zero-safe nets*. They extend ordinary PT nets with a simple mechanism for transition synchronization. We show that the net theory developed under the two most widely adopted semantic interpretations (*collective token* and *individual token* philosophies) can be uniformly adapted to zero-safe nets. In particular, we show that each zero-safe net has two associated PT nets that represent the abstract counterparts of the modeled system according to these two philosophies. We show several applications of the framework, a distributed interpreter for ZS nets based on classical net unfolding (here extended with a *commit* rule) and discuss some extensions to other net flavours to show that the concept of *zero place* provides a unifying notion of transaction for several different kinds of Petri nets.

Introduction

A distributed system can be viewed as a collection of several components that evolve concurrently, by performing local actions, but that can also exchange information, e.g., according to suitable communication protocols. Operational models for distributed systems are often defined using suitable labeled transition systems. *Place/transition Petri nets* [41,43] (abbreviated as PT *nets*) can be viewed as particular structured transition systems, where the additional algebraic structure (i.e., monoidal composition of states and runs) offers a suitable basis for expressing the concurrency of local actions. In fact PT nets have been extensively used both as a foundational model for concurrent computations and as a specification language, due to their well studied theory, a simple graphical presentation and several supporting tools.

When designing large and complex systems via PT nets, the more convenient approach is to start by outlining a very abstract model and then to refine each transition (that might represent a complex activity of the system) into a net that offers a more precise representation of the associated activity. For example, communication protocols for passing and retrieving values cannot ignore that agent synchronization is built on finer actions (e.g., for sending data requests and acknowledgments). Moreover, such actions

---

[⋆] Research supported by CNR Integrated Project *Progettazione e Verifica di Sistemi Eterogenei Connessi mediante Reti*; by TMR Project *GETGRATS*; by Esprit Working Group *APPLIGRAPH*; and by MURST project *TOSCa: Tipi, Ordine Superiore e Concorrenza.*

must be executed according to certain local/global strategies that must be completed before the interaction is closed. Hence the abstract transition is seen, at the refined level, as a distributed computation (that we call a *transaction*) which succeeds only if all the involved component accomplish their tasks. In particular the *commit* of the transaction synchronizes all the terminal operations of local tasks. For the refinement to be correct, we must assume that the transaction is executed *atomically*, as if it were a transition. Thus, the execution strategy can be only partially distributed, since certain local choices must be globally coordinated. However, this is also the case in ordinary (non free-choice) PT nets. In fact, let us consider a generic interpreter for PT nets, where each transition synchronizes the consumption and production of its pre- and postset. This assumption requires that a local activity can influence the behavior of other transitions: Before executing any transition t, the interpreter must lock all the distributed resources that t will consume and this must be done atomically; otherwise a different transition t' could lock some of the resources needed by t. Therefore the interpreter can afford only a certain degree of distribution. This originates what can be called 'place synchronization.'

Several approaches have appeared in the literature that present different refinement techniques for top-down design of a concurrent system (e.g., *Petri Box calculus* [7,6] and *rule-based refinement* [39]). Many references to the subject can be found in [8,26]. Typically at each step a single transition (say t) of the actual net N is refined into a suitable subnet M, yielding the net $N[t \to M]$. This approach is somehow related to the notion of *general net morphism* proposed by Petri, that can be used to map the refined net into its abstract representative by collapsing the structure of M into the transition t. In general some constraints must be assumed on the net M for its behavior to be consistent with that of t, as e.g. in [49,48,47]. Our approach is slightly different, because all transitions of the abstract net are refined by runs of 'the same' *zero-safe net*.

Zero-safe nets (ZS nets) have been introduced in [13] to provide a basic synchronization mechanism for transitions as a built-in feature. In fact, PT nets allow for 'place synchronization' only, whereas 'transition synchronization' is an essential feature to write modular and expressive programs, and to model systems equipped with synchronization primitives (to achieve modularity in defining the net associated to the synchronous composition of two programs, the translations presented in the literature involve complex, and often ad hoc, constructions [50,28,38,22,30,7]).

Besides transitions and ordinary places (here called *stable* places), ZS nets include a distinguished set of *zero* places for modeling idealized resources that remain invisible to external observers, whilst *stable markings*, which just consist of tokens in stable places, define the observable states. Any operational step of a ZS net starts at some stable marking, evolves through hidden states (i.e., markings with some tokens in zero places, called *non-stable markings*) and eventually leads to a stable marking. All the stable tokens produced during a certain step are released together only at the end of the step, i.e., they are 'frozen' until the commit is executed. The synchronization of transitions can thus be performed via zero tokens. The toy example in Figure 1 illustrates this basic mechanism. First, note that we extend the standard graphical representation for nets — in which boxes stand for transitions, circles for places, dots for tokens, and directed weighted arcs describe the flow relation with unary weights omitted — by using smaller

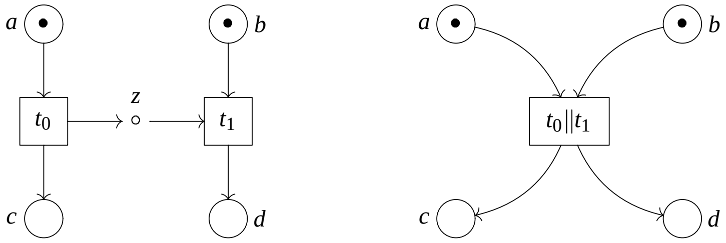

Figure 1. A ZS net and its abstract counterpart.

circles to represent zero places. In the refined model (Figure 1, left), the initial marking $\{a,b\}$ is stable and enables the transition t_0 whose firing produces a 'frozen' stable token in c and a zero token in z. Hence, after the firing of t_0 we reach a non-stable marking. But now t_1 is enabled and its firing consumes the stable token in b and the zero token in z, and produces a frozen token in d. Since the reached marking is stable, the transaction is closed and frozen tokens are released. At the abstract level, we are not interested in observing the hidden intermediate state $\{b,c,z\}$. In fact we just consider stable places plus the atomic activity that takes $\{a,b\}$ and produces $\{c,d\}$ (Figure 1, right). Pursuing this view, a 'refined' ZS net and an 'abstract' PT net model the same system. The latter, where only stable places are considered, offers the synchronized view, which abstracts away the production and consumption of zero tokens.

In this paper we survey the operational and abstract semantics of ZS nets, together with several applications to the modeling of distributed systems. It is worth remarking that both the operational semantics of ZS nets and the construction of their abstract PT nets are characterized as two universal constructions, following the so-called '*Petri nets are monoids*' approach [32]. More precisely, the former can be characterized as an *adjunction* and the latter as a *coreflection*. The universal properties of the two constructions witness that they are the 'optimal' choices. In particular, by expressing the abstract semantics via coreflection we fully justify the choice of abstract PT nets as canonical representatives.

We stress that the synchronization mechanism of ZS nets can favor a uniform approach to concurrent language translation. For instance, in the case of CCS-like process algebras, the parallel composition of two nets modeling communicating processes involves the combinatorial analysis of all the admissible synchronizations, whereas we have shown in [17] that using zero places for modeling communication channels, the parallel composition of two nets can just merge the common channels. As an original contribution, here we show how to model distributed choices in a compositional way and discuss how the basic concept of 'zero place' can be exploited in other net flavours, still preserving some distinguishing features of the approach.

For what concerns ZS nets implementation, one has to specify the computational machinery for performing only correct transactions, recovering deadlocks and treating infinite low-level computations. We illustrate our proposal in [17] for equipping ZS nets with such a distributed operational tool and then extend it to deal with *read arcs* in contextual (zero-safe) nets.

Since the notion of *zero safe* place is to some extent orthogonal w.r.t. the different kinds of Petri nets considered in the literature (e.g., contextual, coloured, timed, probabilistic) we think that it can provide a unifying basis for developing a theory of concurrent transactions in Petri nets. Note that we employ the terminology ‘transaction’ with a meaning analogous to the one it finds in databases: a (sort of) program that when applied to a consistent state still leads to a consistent state, though not necessarily the consistency of the state is preserved by all steps in the program.

Origin and structure of the paper. The operational and abstract semantics of ZS nets according to the two more widely adopted net philosophies (called *collective token* and *individual token*) have been presented in [13,14] together with the associated universal constructions. A comparison between the two approaches has been discussed in [17], in the Ph.D. Thesis of the first author [10], and in the tutorial overview [15]. The distributed interpreter for ZS nets has been proposed in [16]. The modeling of distributed *don’t know* choice and the extensions of the zero safe approach to other net flavours (e.g., read arcs) have not appeared elsewhere.

In Section 1 we recall PT nets and their semantics. Section 2 illustrates ZS nets and their operational and abstract semantics and uses two examples to motivate the usage of zero places. In Section 3 we give a compositional representation of a simple process algebra equipped with action prefix, parallel composition, restriction and *don’t know* nondeterministic choice. The distributed interpreter for ZS nets is defined in Section 4. We conclude in Section 5 by extending the ZS net formalism to deal with read arcs. For detailed proofs of most results we refer to [17,16,10].

Acknowledgements. We thank Paolo Baldan for several interesting discussions on contextual nets and for his careful reading of a preliminary version of this work. We also thank José Meseguer and the anonymous referees for their suggestions that helped us in improving the presentation of the material.

1 Place/transition Petri nets

Definition 1 (Net). *A* net *is a triple* $N = (S_N, T_N, F_N)$*, where* S_N *is the set of* places $a, a', \ldots$*,* T_N *is the set of* transitions $t, t', \ldots$ *(with* $S_N \cap T_N = \varnothing$*), and* $F_N \subseteq (S_N \times T_N) \cup (T_N \times S_N)$ *is called the* flow relation. *The elements of the flow relation are called* arcs, *and we write* $x\, F_N\, y$ *for* $(x,y) \in F_N$*.*

We shall denote $S_N \cup T_N$ by N when no confusion can arise. Subscripts will be omitted if they are obvious from the context. For $x \in N$, the set ${}^{\bullet}x = \{y \in N \mid y\, F\, x\}$ is called the *preset* of x, and the set $x^{\bullet} = \{y \in N \mid x\, F\, y\}$ is called the *postset* of x. We only consider nets such that for any transition t, ${}^{\bullet}t \neq \varnothing$. Moreover, let ${}^{\circ}N = \{x \in N \mid {}^{\bullet}x = \varnothing\}$ and $N^{\circ} = \{x \in N \mid x^{\bullet} = \varnothing\}$ denote the sets of *initial* and *final elements* of N respectively. A place a is *isolated* if ${}^{\bullet}a \cup a^{\bullet} = \varnothing$.

Definition 2 (PT net). *A* marked place/transition Petri net *(PT net) is a tuple* $N = (S,T,F,W,u_{\mathrm{in}})$ *such that* (S,T,F) *is a net, the function* $W{:}F \to \mathbb{N}$ *assigns a positive* weight *to each arc in* F*, and the finite multiset* $u_{\mathrm{in}}{:}S \to \mathbb{N}$ *is the* initial marking *of* N*.*

We find convenient to view F as a function $F{:}(S\times T)\cup(T\times S)\to\{0,1\}$, with $x\,F\,y \iff F(x,y)\neq 0$. Then, for PT nets we replace $\{0,1\}$ by $\mathbb{N}$ and abandon W. Thus, the flow relation becomes a *multiset relation* $F{:}(S\times T)\cup(T\times S)\to\mathbb{N}$.

A *marking* $u{:}S\to\mathbb{N}$ is a finite multiset of places. It can be written either as $u = \{n_1a_1,\ldots,n_ka_k\}$ where each n_i dictates the number of occurrences (*tokens*) of the place a_i in u, i.e., $n_i = u(a_i)$ (if $n_i = 0$ then the n_ia_i is omitted), or as the formal sum $u = \bigoplus_{a_i\in S} n_ia_i$ denoting an element of the free commutative monoid $S^{\oplus}$ on the set of places S (the monoidal operation is defined by $(\bigoplus_i n_ia_i)\oplus(\bigoplus_i m_ia_i) = (\bigoplus_i(n_i+m_i)a_i)$ with 0 as the neutral element). The monoid $(\mu(S),\cup,\varnothing)$ of finite multisets on S (with multiset union as monoidal operation and the empty multiset as unit) is isomorphic to $S^{\oplus}$.

For any transition $t\in T$, let $\mathrm{pre}(t)$ and $\mathrm{post}(t)$ be the multisets over S such that $\mathrm{pre}(t)(a) = F(a,t)$ and $\mathrm{post}(t)(a) = F(t,a)$, for all $a\in S$. A PT net can be equivalently defined as the (marked) graph $(S^{\oplus},T,\mathrm{pre},\mathrm{post},u_{\mathrm{in}})$, with nodes in the monoid $S^{\oplus}$ and edges in T, where $\mathrm{pre}(\_),\mathrm{post}(\_){:}T\to S^{\oplus}$ define the source and target of transitions, respectively. As usual we write $t{:}u\to v$ for a transition t with $\mathrm{pre}(t) = u$ and $\mathrm{post}(t) = v$. This definition emphasizes the algebraic structure of PT nets and allows us to define a category of nets by considering the obvious homomorphisms preserving such structure.

Definition 3 (Category Petri). *A* PT net morphism $h{:}N\to N'$ *is a pair of functions* $h = (f{:}T\to T', g{:}S^{\oplus}\to S'^{\oplus})$ *with* g *a monoid homomorphism and with* $g(\mathrm{pre}(t)) = \mathrm{pre}(f(t))$ *and* $g(\mathrm{post}(t)) = \mathrm{post}(f(t))$ *for each* $t\in T$*. That is,* h *is a graph morphism whose node component* g *is a monoid homomorphism. (For marked nets, morphisms must also preserve initial markings, i.e.,* $g(u_{\mathrm{in}}) = u'_{\mathrm{in}}$*.) The category* **Petri** *has (unmarked)* PT *nets as objects and* PT *net morphisms as arrows.*

Definition 4 (Firing). *Given a* PT *net* N*, let* u *and* u' *be markings of* N*. A transition* $t\in T$ *is* enabled *at* u *if* $\mathrm{pre}(t)(a)\leq u(a)$*, for all* $a\in S$*. Moreover, we say that* u *evolves to* u' *under the* firing *of* t*, written* $u\,[t\rangle\,u'$*, if* t *is enabled at* u *and* $u'(a) = u(a) - \mathrm{pre}(t)(a) + \mathrm{post}(t)(a)$*, for all* $a\in S$*.*

A *firing sequence* from u_0 to u_n is a sequence of markings and transitions such that $u_0\,[t_1\rangle\,u_1\ldots u_{n-1}\,[t_n\rangle\,u_n$. Besides firings and firing sequences, *steps* and *step sequences* are usually introduced.

Definition 5 (Step). *Given a* PT *net* N*, we say that a multiset* $X{:}T\to\mathbb{N}$ *is enabled at* u *if* $\sum_{t\in T} X(t)\cdot\mathrm{pre}(t)(a)\leq u(a)$ *for all* $a\in S$*.*

Moreover, we say that u *evolves to* u' *under the* step X*, written* $u\,[X\rangle\,u'$*, if* X *is enabled at* u *and* $u'(a) = u(a) + \sum_{t\in T} X(t)\cdot(\mathrm{post}(t)(a) - \mathrm{pre}(t)(a))$ *for all* $a\in S$*.*

Given a marking u of N, we denote by $[u\rangle$ the set of all the markings that are reachable from u via some firing sequence. The *reachable markings* of the net $N = (S,T,F,u_{\mathrm{in}})$ are the elements of the set $[u_{\mathrm{in}}\rangle$.

identities	**generators**	**parallel composition**	**basic step**	**sequential composition**
$\dfrac{u \in S^{\oplus}}{u \Rightarrow_N u}$	$\dfrac{t\colon u \to v \in T}{u \Rightarrow_N v}$	$\dfrac{u \Rightarrow_N v,\ u' \Rightarrow_N v'}{u \oplus u' \Rightarrow_N v \oplus v'}$	$\dfrac{u \Rightarrow_N v}{u \Rightarrow_N^* v}$	$\dfrac{u \Rightarrow_N^* v,\ v \Rightarrow_N w}{u \Rightarrow_N^* w}$

Table 1. The inference rules for $\_ \Rightarrow_N \_$ and $\_ \Rightarrow_N^* \_$.

The dynamics of a net can be expressed by the one-step relation $\_ \Rightarrow_N \_$ defined by the three leftmost inference rules in Table 1: identities represent idle resources, generators represent the firing of a transition within the minimal marking that can enable it, and parallel composition provides concurrent execution of generators and idle steps. Then, it is obvious that $u \Rightarrow_N v \iff \exists (X\colon T \to \mathbb{N}).u\ [X\rangle\ v$.

The extension of this approach to computations $u_0 \Rightarrow u_1 \Rightarrow \cdots \Rightarrow u_n$ is not straightforward. Indeed, concurrent semantics must consider as equivalent all the computations where the same *concurrent* events are executed in different orders, and we cannot leave out of consideration the distinction between *collective* and *individual token philosophies* (noticed e.g., in [27], but see also [11,12]).

The simplest approach relies on the collective token philosophy (*CTph*), where semantics does not distinguish among tokens which are available at the same place, because any such token is regarded to be *operationally* equivalent to all the others. A major drawback of this approach is that it leaves out of consideration the fact that operationally equivalent resources may have different origins and histories, carrying different *causality* information. Instead, according to the individual token philosophy (*ITph*), causal dependencies are central to net dynamics. As a consequence, only the computations that refer to isomorphic *Goltz-Reisig processes* [29] can be identified, and causality information is fully maintained (the *CTph* relies instead on the *commutative processes* of Best and Devillers [5]). If one is simply interested in 'reachability' matters, then the distinction between the *CTph* and *ITph* is irrelevant, and the obvious two rightmost rules in Table 1 can be introduced (transitive closure). Otherwise, suitable *proof terms* for computations can be introduced and axiomatized to faithfully recover the two different philosophies. In this sense, Best-Devillers and Goltz-Reisig processes can be seen as *concurrent computation strategies* for *CTph* (resp. *ITph*) and can be shown to correspond to equivalence classes of proof terms modulo natural algebraic axiomatizations [23].

Commutative processes can be characterized by quotienting step sequences.

Definition 6 (Diamond transformation). *Given a* PT *net N, let*

$$s = u_0\ [t_1\rangle\ u_1 \cdots u_{i-1}\ [t_i\rangle\ u_i\ [t_{i+1}\rangle\ u_{i+1} \cdots u_{n-1}\ [t_n\rangle\ u_n$$

be a step sequence of N, where t_i and t_{i+1} are concurrently enabled by u_{i-1}, in the sense that $(\mathrm{pre}(t_i) \cup \mathrm{pre}(t_{i+1}))(a) \leq u_{i-1}(a)$ for any $a \in S_N$. Let s' be the firing sequence obtained from s by firing t_i and t_{i+1} in the reverse order, i.e.,

$$s' = u_0\ [t_1\rangle\ u_1 \cdots u_{i-1}\ [t_{i+1}\rangle\ u_i'\ [t_i\rangle\ u_{i+1} \cdots u_{n-1}\ [t_n\rangle\ u_n.$$

Then, the sequence s' is called a diamond transformation *of s.*

Since in step sequences transitions can be fired concurrently, we let the step sequence $u_0\ [X_1\rangle\ u_1 \cdots u_{i-1}\ [X_i \cup X_{i+1}\rangle\ u_{i+1} \cdots u_{n-1}\ [X_n\rangle\ u_n$ be a diamond transformation of $u_0\ [X_1\rangle\ u_1 \cdots u_{i-1}\ [X_i\rangle\ u_i\ [X_{i+1}\rangle\ u_{i+1} \cdots u_{n-1}\ [X_n\rangle\ u_n$ with X_i and X_{i+1} concurrently enabled by u_{i-1} (and vice versa).

Diamond transformations define a symmetric relation whose reflexive and transitive closure gives the right equivalence w.r.t. the *CTph* interpretation.

The notion of (causal) *process* is due to Goltz and Reisig [29] and gives a more precise account of causal dependencies between firings and tokens.

Definition 7 (Occurrence net). *A net K is a (*deterministic*)* occurrence net *if (1) for all $a \in S_K$, $|{}^\bullet a| \leq 1 \wedge |a^\bullet| \leq 1$ and (2) F_K^* is acyclic.*[1]

Definition 8 (Process). *A* process *for a* PT *net N is a net morphism $P{:}K \to N$, from an occurrence net K to N, such that $P(S_K) \subseteq S_N$, $P(T_K) \subseteq T_N$, ${}^\circ K \subseteq S_K$, and for all $t \in T_K$, $a \in S_N$, $F_N(a,P(t)) = \left|P^{-1}(a) \cap {}^\bullet t\right|$ and $F_N(P(t),a) = \left|P^{-1}(a) \cap t^\bullet\right|$.*

Two processes P and P' of N are *isomorphic* and thus identified if there exists a net isomorphism $\psi{:}K_P \to K_{P'}$ such that $\psi;P' = P$. As usual we denote the set of *origins* (i.e., minimal or initial places) and *destinations* (i.e., final or maximal places) by $O(K) = {}^\circ K$ and $D(K) = K^\circ \cap S_K$, respectively. For concatenating causal computations, the notion of *concatenable process* has been introduced in [23]. Concatenable processes are obtained from processes by imposing a total ordering on the origins that are instances of the same place and, similarly, on the destinations. The orderings are defined by means of label-indexed ordering functions. Given a set S with a labeling function $l{:}S \to S'$, a *label-indexed ordering function* for l is a family $\beta = \{\beta_a\}_{a \in S'}$ of bijections, where $\beta_a{:}l^{-1}(a) \to \{1,\ldots,\left|l^{-1}(a)\right|\}$. Thus, for $x,y \in l^{-1}(a)$ we let $x \sqsubseteq y \iff \beta_a(x) \leq \beta_a(y)$.

Definition 9 (Concatenable process). *A* concatenable process *for a* PT *net N is a triple $C = (P,\ {}^\circ\ell, \ell^\circ)$ where $P{:}K \to N$ is a process, and ${}^\circ\ell$, ℓ° are label-indexed ordering functions for the function P restricted to $O(K)$ and $D(K)$ respectively.*

Two concatenable processes C and C' are *isomorphic* if P_C and $P_{C'}$ are isomorphic via a morphism that preserves all the orderings. A partial binary operation $\_;\_$ (associative up to isomorphism and with identities) of concatenation of concatenable processes (whence their names) can be easily defined: we take as source (target) the image through P of the initial (maximal) places of K_P; then the composition of $C = (P,\ {}^\circ\ell, \ell^\circ)$ and $C' = (P',\ {}^\circ\ell', \ell'^\circ)$ is realized by merging, when it is possible, the maximal places of K_P with the initial places of $K_{P'}$ according to their labeling and ordering functions. Concatenable processes admit also a monoidal *parallel* composition $\_ \otimes \_$ (commutative up to a natural isomorphism), which can be represented by putting two processes side by side. We refer the interested reader to [23] for the formal definitions of $C;C'$ and $C \otimes C'$, which make the concatenable processes of a PT net N be the arrows of a symmetric monoidal category $\mathcal{CP}(N)$ (whose objects are the markings of N). The symmetries of $\mathcal{CP}(N)$ are given by concatenable processes with empty set of transitions (token permutation is expressed by different orderings ${}^\circ\ell$ and ℓ°).

---

[1] F^* denotes the reflexive and transitive closure of relation F.

$$\frac{u \in S^{\oplus}}{id_u : u \to u} \qquad \frac{t : u \to v \in T}{t : u \to v} \qquad \frac{\alpha : u \to v,\ \beta : u' \to v'}{\alpha \otimes \beta : u \oplus u' \to v \oplus v'} \qquad \frac{\alpha : u \to v,\ \beta : v \to w}{\alpha ; \beta : u \to w}$$

Table 2.

neutral: $id_{\emptyset} \oplus \alpha = \alpha,$

commutativity: $\alpha \oplus \beta = \beta \oplus \alpha,$

associativity: $(\alpha \oplus \beta) \oplus \alpha' = \alpha \oplus (\beta \oplus \alpha'), \qquad (\alpha;\beta);\alpha' = \alpha;(\beta;\alpha'),$

identities: $\alpha; id_u = id_v; \alpha = \alpha, \qquad id_u \oplus id_v = id_{u \oplus v},$

functoriality: $(\alpha;\beta) \oplus (\alpha';\beta') = (\alpha \oplus \alpha');(\beta \oplus \beta').$

Table 3.

1.1 Petri nets are monoids

Several interesting aspects of net theory can be profitably developed within category theory, see, e.g., [52,32,9,24,40,35]. We focus on the so-called 'Petri nets are monoids' approach initiated in [32] (see also [23,33,45,34,46,12]). The idea is to extend (part of) the algebraic structure of states to the level of proof terms associated to the rules in Table 1 in such a way to capture the basic laws of concurrent and causal computations. The proof terms we consider are inductively defined in Table 2. In [32,23] it is shown that axiomatic equivalences on such proof terms can precisely characterize several standard semantic constructions. In particular, commutative processes can be characterized by lifting the multiset structure of states to the level of computations in a functorial way, yielding a strictly symmetric monoidal category $\mathcal{T}(N)$ (it is called 'strictly symmetric' because the monoidal operation is commutative). For each net N, the category $\mathcal{T}(N)$ has markings of N as objects, and proof terms modulo the axioms in Table 3 as arrows. Abusing the notation, in Table 3 the parallel composition of arrows is denoted by $\oplus$, instead of $\otimes$, to emphasize that it is commutative and can be viewed as multiset union. The functoriality law is the analogous of diamond transformation. Denoting by **CMonCat** the category of strictly symmetric monoidal categories (as objects) and monoidal functors (as arrows), $\mathcal{T}(\_)$ extends to a functor from **Petri** to **CMonCat**.

Proposition 1 (cf. [32]). *The presentation of $\mathcal{T}(N)$ given above precisely characterizes the algebra of commutative processes of N, i.e., the arrows in $\mathcal{T}(N)$ are in bijection with the commutative processes of N.*

Under the *ITph*, for analogous results to hold, one must resort to symmetric monoidal categories, where parallel composition is commutative only up to a natural isomorphism. In fact, suitable auxiliary arrows called *symmetries* are present (see Table 4) that can model the possible reorganizations of minimal and maximal places of a process. We recall here the definition of the category $\mathcal{P}(N)$ introduced in [23] and finitely axiomatized in [45].

Definition 10. *Let N be a* PT *net. The category $\mathcal{P}(N)$ is the monoidal quotient of the free symmetric monoidal category $\mathcal{F}(N)$ generated by N, modulo the two axioms: (i)*

$$\frac{u, u' \in S^{\oplus}}{\gamma_{u,u'} : u \oplus u' \to u' \oplus u}$$

Table 4.

neutral:	$id_\varnothing \otimes \alpha = \alpha \otimes id_\varnothing = \alpha$,	
associativity:	$(\alpha \otimes \beta) \otimes \alpha' = \alpha \otimes (\beta \otimes \alpha')$,	$(\alpha;\beta);\alpha' = \alpha;(\beta;\alpha')$,
identities:	$\alpha; id_u = id_v; \alpha = \alpha$,	$id_u \otimes id_v = id_{u \oplus v}$,
functoriality:	$(\alpha;\beta) \otimes (\alpha';\beta') = (\alpha \otimes \alpha');(\beta \otimes \beta')$,	
naturality:	$(\alpha \otimes \alpha');\gamma_{v,v'} = \gamma_{u,u'};(\alpha' \otimes \alpha)$,	
coherence:	$\gamma_{u,v \oplus v'} = (\gamma_{u,v} \otimes id_{v'});(id_v \otimes \gamma_{u,v'})$,	$\gamma_{u,v};\gamma_{v,u} = id_{u \oplus v}$.

Table 5.

$\gamma_{a,b} = id_a \otimes id_b$, *if* $a, b \in S$, *and* $a \neq b$; *and (ii)* $s;t;s' = t$, *if* $t \in T$ *and* s, s' *are symmetries (where* $\gamma_{\_,\_}$, $id_\_$, $\_ \otimes \_$, *and* $\_;\_$ *are, resp., the symmetry isomorphism, the identities, the tensor product, and the composition of* $\mathcal{F}(N)$*).*

We remark that in $\mathcal{F}(N)$ the tensor product is not commutative and the symmetries satisfy the naturality axiom and the MacLane coherence axioms [31]. For the reader's convenience, the axioms of $\mathcal{F}(N)$ are recalled in Table 5.

Proposition 2. *The presentation of* $\mathcal{P}(N)$ *given above precisely characterizes the algebra of concatenable processes of the* PT *net* N.

The constructions $\mathcal{T}(N)$ and $\mathcal{P}(N)$ provide a useful syntax that can be used for denoting commutative processes and concatenable processes, respectively.

2 Zero-safe nets

We recall the notion of *safety* in PT nets.

Definition 11 (*n*-safe net). *A place is n-*safe *if it contains at most n tokens in any reachable marking. A net is n-safe if all its places are n-safe.*

Thus, the adjective '0-safe' for nets means that all places cannot contain any token in all reachable markings. We use the terminology *zero-safe net* — using the word 'zero' instead of the digit '0' — to mean that the net contains special places, called *zero places*, whose role is that of coordinating the atomic execution of several transitions, which, from an abstract viewpoint, will appear as synchronized. However no new interaction mechanism is needed, and the coordination of the transitions participating in a step is handled by the ordinary token-pushing rules of nets, assuming late delivery of stable tokens (postponed to the end of the transaction). These places are 'zero-safe' in the sense that they cannot contain any token in any *observable* state.

underlying	**horizontal composition**	**commit**
$\dfrac{u\oplus x \Rightarrow_{N_B} v\oplus y,\ u,v\in L^{\oplus},\ x,y\in Z^{\oplus}}{(u,x)\rightrightarrows_B (v,y)}$	$\dfrac{(u,x)\rightrightarrows_B (v,y),\ (u',y)\rightrightarrows_B (v',y')}{(u\oplus u',x)\rightrightarrows_B (v\oplus v',y')}$	$\dfrac{(u,0)\rightrightarrows_B (v,0)}{u\Rrightarrow_B v}$

Table 6. The inference rules for $\_\Rrightarrow_B\_$.

Definition 12 (ZS net). *A* zero-safe net *(*ZS *net) is a tuple $B=(S_B,\ T_B,\ F_B,\ u_B,\ Z_B)$ where $N_B=(S_B,\ T_B,\ F_B,\ u_B)$ is the* underlying PT net *of B and the set $Z_B\subseteq S_B$ is the set of* zero places*. The places in $L_B=S_B\setminus Z_B$ are called* stable places*. A* stable marking *is a multiset of stable places, and the initial marking u_B must be stable.*

Stable markings describe *observable* states, whereas the presence of one or more zero tokens in a given marking makes it be *unobservable*. We call *stable tokens* and *zero tokens* the tokens that respectively belong to stable places and to zero places. Since $S^{\oplus}$ is a free commutative monoid, it is isomorphic to the cartesian product $L^{\oplus}\times Z^{\oplus}$ and we can write $t\colon(u,x)\to(v,y)$ for a transition t with $\mathrm{pre}(t)=u\oplus x$ and $\mathrm{post}(t)=v\oplus y$, where u and v are stable multisets and x and y are multisets over Z. In a way similar to PT nets, ZS nets can also be seen as suitable graphs, yielding the following category.

Definition 13 (Category dZPetri). *A* ZS net morphism *between two* ZS *nets B and B' is a* PT *net morphism $(f,g)\colon N_B\to N_{B'}$ where g preserves the partitioning of places (i.e., $g(a)\in L_{B'}^{\oplus}$ if $a\in L_B$ and $g(a)\in Z_{B'}^{\oplus}$ if $a\in Z_B$) and satisfies the additional condition of mapping zero places into pairwise disjoint (nonempty) zero markings (i.e., for all $z\neq z'\in Z_B$, if $g(z)=n_1a_1\oplus\cdots\oplus n_ka_k$ and $g(z')=m_1b_1\oplus\cdots\oplus m_lb_l$ then we have that $a_i\neq b_j$ for $i=1,\ldots,k$ and $j=1,\ldots,l$), which is called the* disjoint image *property. The category* **dZPetri** *has* ZS *nets as objects and* ZS *net morphisms as arrows.*

Since $S^{\oplus}$ is equivalent to $L^{\oplus}\times Z^{\oplus}$, ZS net morphisms become triples of the form $h=(f,g_L,g_Z)$, where both g_L and g_Z are monoid homomorphisms on the free commutative monoids of stable and zero places, respectively.

Proposition 3. *The category* **Petri** *is a full subcategory of* **dZPetri**.

As for PT nets, we can define the behavior of ZS nets by means of a step relation $\_\Rrightarrow_B\_$, defined by the inference rules in Table 6. An auxiliary relation $\_\rightrightarrows_B\_$ is introduced for modeling *transaction segments*. We can take advantage of the step relation $\_\Rightarrow_{N_B}\_$ of the underlying net for concurrently executing several transitions (rule **underlying**). The rule **horizontal composition** acts as parallel composition for stable resources and as sequential composition for zero places. We call it 'horizontal' because we prefer to view it as a synchronization mechanism rather than as the ordinary sequential composition of computations, which flows vertically from top to bottom. The rule **commit** selects the transaction segments that correspond to acceptable steps: They must start from a stable marking and end up in a stable marking. As a particular instance of the horizontal composition of two transaction segments $(u,0)\rightrightarrows_B(v,0)$ and $(u',0)\rightrightarrows_B(v',0)$, we can derive their parallel composition $(u\oplus u',0)\rightrightarrows_B(v\oplus v',0)$.

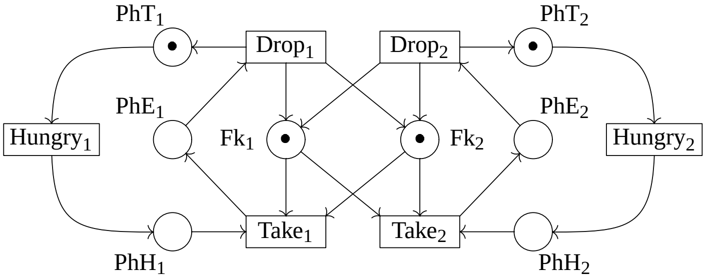

Figure 2. An abstract view for (two) dining philosophers.

2.1 Introductory example: Dining philosophers.

A simple example that illustrates the coordination role played by zero places relies on the modeling of the well-known ‘dining philosophers’ problem: There are n philosophers (with $n \geq 2$) sitting on a round table; each having a plate in front with some food on it; between each couple of plates there is a fork, with a total of n forks on the table; each philosopher cyclically thinks and eats, but to eat he needs both the fork on the left and that on the right of his plate; after eating a few mouthfuls, the philosopher puts the forks back on the table and starts thinking again.

The PT net for the case $n = 2$ is illustrated in Figure 2. A token in one of the places PhH_i, PhE_i, and PhT_i, for $1 \leq i \leq 2$, means that the ith philosopher is hungry, is eating, and is thinking, respectively. A token in the place Fk_i means that the ith fork is on the table. The transitions Take_i, Drop_i, and Hungry_i represent that the ith philosopher takes the forks and starts eating, finishes eating and drops the forks, feels his stomach hungry and prepares to eat, respectively. Note that Take_i requires both forks and thus cannot be fired if the other philosopher is eating. The initial marking of the net is $\{\mathrm{PhT}_1, \mathrm{PhT}_2, \mathrm{Fk}_1, \mathrm{Fk}_2\}$ (i.e., both philosophers are thinking and both forks are on the table).

Of course, this model does not tell how the philosophers access the ‘resources’ needed to eat, whereas the action Take_i is not trivial and requires some atomic mechanism for getting the forks. At a more refined level, for example, the strategy for executing the action Take_i could be specified as ‘take the ith fork (if possible), then the $((i\,\mathbf{mod}\,2)+1)$th fork (if possible) and eat,’ hence it is not difficult to imagine a deadlock where each philosopher takes one fork and cannot continue, since conflict arises. The fact is that the coordination mechanism is hidden inside transitions whose granularity is too coarse.

The situation is completely different if one wants to model the system using *free choice* nets,[2] where all decisions are local to each place. To see this, let us concentrate

---

[2] We recall that a net is *free choice* if for any transitions t_1 and t_2 whose presets are not disjoint, then the presets of t_1 and t_2 consist of exactly one place, or equivalently, a net is free choice if for any place s in the preset of two or more transitions, then the preset of any such transition is exactly $\{s\}$.

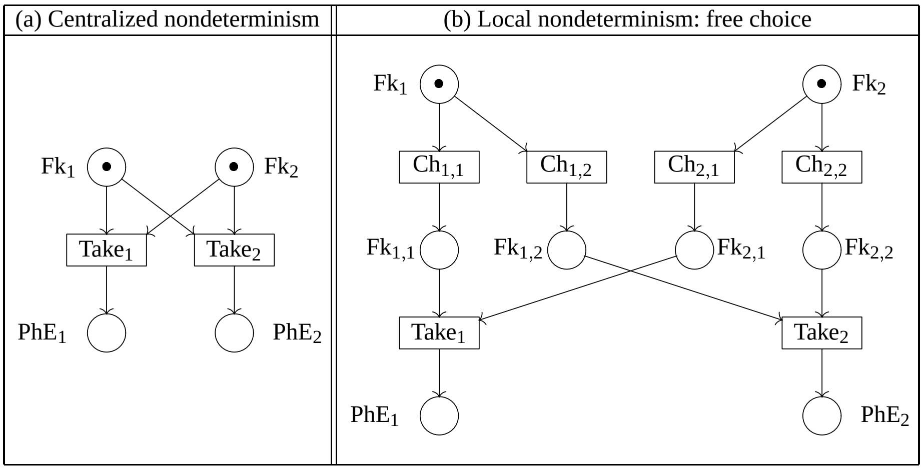

Figure 3. Global vs (completely) local choices.

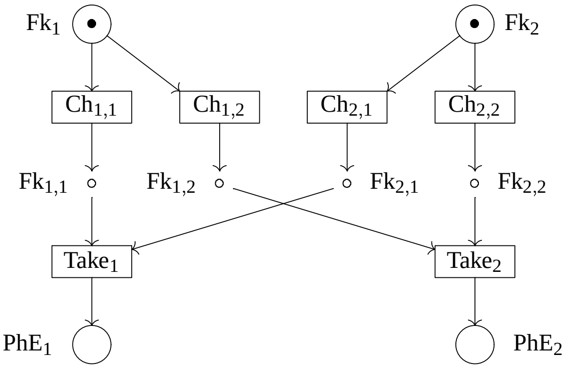

Figure 4. Atomic free choice.

our attention to a subpart of the net in Figure 2, depicted in Figure 3(a), which will suffice to illustrate the point. We can translate any net into a free choice net by adding special transitions that perform the local decisions required. For example, the free choice net in Figure 3(b) corresponds to the net in Figure 3(a), but models a system where two decisions can take place independently: One decision concerns the assignment of the first fork either to the first or the second philosopher, the other decision concerns the assignment of the second fork. Then, it might happen that the first fork is assigned to the first philosopher ($\mathrm{Ch}_{1,1}$) and the second fork is assigned to the second philosopher ($\mathrm{Ch}_{2,2}$), and in such case the translated net deadlocks and none of the Take_i actions can occur. Thus, the translated net admits computations not allowed in the abstract system of Figure 3(a).

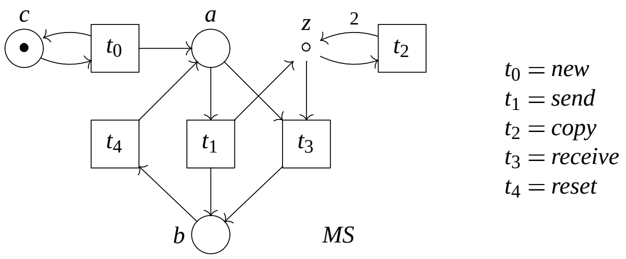

Figure 5. The ZS net *MS* representing a multicasting system.

Zero-safe nets overcome this deadlock problem by executing only certain atomic transactions, where tokens produced in low-level resources are also consumed. In our example, the invisible resources consist of places $\mathrm{Fk}_{i,j}$ for $1 \leq i,j \leq 2$, that can be interpreted as zero-places. In this way the computation performing $\mathrm{Ch}_{1,1}$ and $\mathrm{Ch}_{2,2}$ is forbidden, because it stops in an invisible state, i.e., a state that contains zero tokens. Figure 4 represents the low-level model as a ZS net. (Recall that smaller circles stand for zero-places.)

2.2 *CTph* vs. *ITph*: The multicasting system example

At an abstract level, the system modeled via a ZS net B can be equivalently described via a PT net $\mathcal{E}(B)$ such that $S_{\mathcal{E}(B)} = L_B = S_B \setminus Z_B$ and $(\_ \Rightarrow_{\mathcal{E}(B)} \_) = (\_ \Rrightarrow_B \_)$. Among the several PT nets that satisfy the above conditions we would like to choose the optimal one: Informally the transitions of such net should represent the proofs of transaction steps $u \Rrightarrow_B v$ taken up to concurrent equivalence and such that they cannot be decomposed into smaller transaction proofs. When these two conditions are satisfied, the concurrent kernel of the possible behaviors has been identified, and all the steps can be generated by it.

We have seen in Section 1 that when dealing with concurrency, there is a real dichotomy between the *CTph* and the *ITph*. According to the *CTph*, all those firing sequences obtained by repeatedly permuting pairs of (adjacent) concurrently enabled firings are identified. We call *abstract stable transactions* the resulting equivalence classes of ZS net behaviours. However, acting in this way, causal dependencies on zero tokens are lost, and the class of computations captured by abstract nets may be too abstract for some applications. According to the *ITph*, instead, causal dependencies are a central aspect. As a consequence, only the transactions which refer to isomorphic Goltz-Reisig processes are identified, and we call *connected transactions* the induced equivalence classes. To illustrate these concepts, we recall the 'multicasting' example, taken from [14]. The ZS net *MS* depicted in Figure 5 is designed to model a *multicasting system*: As in a broadcasting system, an agent can simultaneously send the same message to an unlimited number of receivers, but here the receivers are not necessarily all the remaining agents.

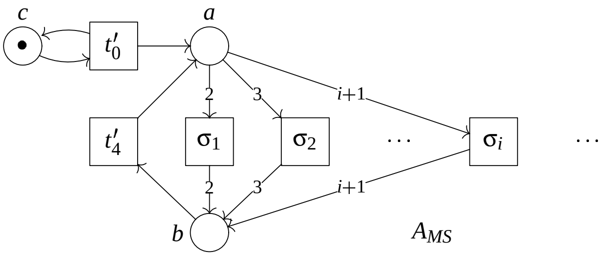

Figure 6. The abstract net for the multicasting system under the *CTph*.

Each token in place *a* represents a different *active* agent that is ready to communicate, while tokens in *b* represent *inactive* agents. The zero place *z* models a buffer where tokens are messages (e.g., data, values). The transition *new* permits creating fresh agents. Each firing of *send* opens a one-to-many communication: A message is put in the buffer *z* and the agent which started the communication is frozen in *b* until the end of the current transaction. Each time the transition *copy* fires, a new copy of a message is created. To complete a transaction, as many firings of *receive* are needed as the number of copies created by *copy* plus one. Each firing of *receive* synchronizes an active agent with a copy of the message and then freezes the agent. At the end of a session, all the suspended agents are moved into place *b*. The transition *reset* activates an inactive agent. The graph corresponding to the ZS net *MS* has the following set of arcs: $T_{MS} = \{t_0\colon(c,0) \to (a \oplus c,0),\ t_1\colon(a,0) \to (b,z),\ t_2\colon(0,z) \to (0,2z),\ t_3\colon(a,z) \to (b,0),\ t_4\colon(b,0) \to (a,0)\}$.

In Figure 6 we see the infinite abstract PT net A_{MS} for the refined ZS net *MS*, according to the *CTph* (see Definition 17). As it will be explained later, the *abstract* net A_{MS} comes equipped with a *refinement morphism* ε^C_{MS} to the refined net *MS*. The refinement morphism maps each place of A_{MS} into the homonymous stable place of *MS* and defines a bijection between the transitions of A_{MS} and the abstract stable transactions of *MS*: The transition σ_n of A_{MS} represents a one-to-*n* transmission. By contrast, under the *ITph*, different copy policies[3] for a one-to-*n* transmission may be distinguished. The infinite *causal abstract* PT net I_{MS} corresponding to the refined ZS net *MS* under the *ITph* (see Definition 24) is displayed in Figure 7. It comes equipped with a *causal refinement morphism* ε^I_{MS} to the refined net *MS*. Such morphism maps each place of I_{MS} into the homonymous stable place of *MS*, and defines a bijection between the transitions of I_{MS} and the connected transactions of *MS*. We assume that the generic transition σ^k_n corresponds to the one-to-*n* transmission that follows the *k*-th codified copy policy (we denote by c_n the number of different copy policies associated to the one-to-*n* transmission).

Zero places can be used to coordinate and synchronize in a single transaction any number of transitions of the refined net. Thus it may well happen that the refined net

---

[3] We call *copy policy* any strategy (e.g., sequential, with maximal parallelism) for making copies of the messages in the buffer *z*.

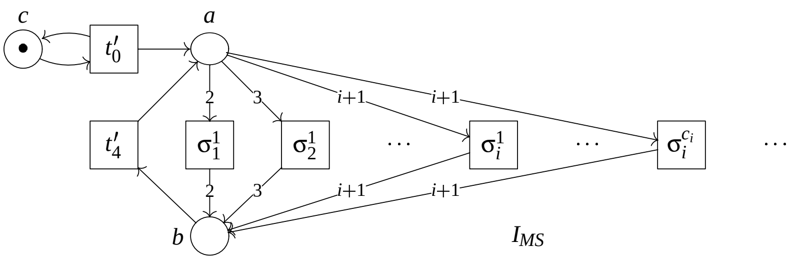

Figure 7. The *causal* abstract net for the multicasting system under the *ITph*.

is finite while the abstract net is infinite. This is the case for this example, in which communication events can involve any number of receivers.

2.3 Collective token approach

Operational semantics under the *CTph*. A *stable step* of a ZS net B may involve the execution of several transitions. At the beginning, the state must contain enough stable tokens to concurrently enable the stable presets of all these transitions. As the computation progresses, the firings can only consume the stable tokens that were also available at the beginning of the computation and the zero tokens that have been produced by some fired transition. A stable step whose intermediate markings are all nonstable and which consumes all the available stable tokens is called a *stable transaction*.

Definition 14 (Stable step and stable transaction). *Let B be a* ZS *net. A firing sequence $s = u_0\ [t_1\rangle\ u_1 \dots u_{n-1}\ [t_n\rangle\ u_n$ of the underlying net N_B is a* stable step *of B if:*

- $\sum_{i=1}^{n} \mathrm{pre}(t_i)(a) \leq u_0(a)$ *for all $a \in S_B \setminus Z_B$ (concurrent enabling);*
- *u_0 and u_n are stable markings of B (stable fairness).*

We write $u_0\{[s\rangle u_n$ to denote the stable step s, and $O(s)$ and $D(s)$ to denote the u_0 and u_n respectively. A stable step s is a stable transaction *if in addition:*

- *the markings $u_1, \dots, u_{n-1}$ are not stable (atomicity);*
- $\sum_{i=1}^{n} \mathrm{pre}(t_i)(a) = u_0(a)$ *for all $a \in S_B \setminus Z_B$ (perfect enabling).*

A *stable step sequence* is a sequence of stable steps $u_0\{[s_1\rangle u_1 \dots u_{n-1}\{[s_n\rangle u_n$. We then say that u_n is *reachable* from u_0. We recall that stable tokens produced during the transaction become operative in the system only after the commit.

Example 1. Consider the ZS net *MS* of Figure 5.

The firing sequence $\{2a\}\ [t_1\rangle\ \{a,b,z\}\ [t_4\rangle\ \{2a,z\}\ [t_3\rangle\ \{a,b\}$ is not a stable step, because the concurrent enabling condition is not satisfied.

The sequence $\{4a\}\ [t_1\rangle\ \{3a,b,z\}\ [t_2\rangle\ \{3a,b,2z\}\ [t_3\rangle\ \{2a,2b,z\}\ [t_3\rangle\ \{a,3b\}$ is a stable step but not a stable transaction, because the perfect enabling condition is not satisfied.

The firing sequence $s' = \{2a,b\}\ [t_1\rangle\ \{a,2b,z\}\ [t_3\rangle\ \{3b\}\ [t_4\rangle\ \{a,2b\}$ is a stable step but not a stable transaction, because the atomicity constraint is not satisfied.

The firing sequence $s'' = \{2a,b\}\ [t_1\rangle\ \{a,2b,z\}\ [t_4\rangle\ \{2a,b,z\}\ [t_3\rangle\ \{a,2b\}$ is a stable transaction (compare it with the first sequence of this example).

To obtain a more satisfactory notion of stable step (transaction) in the concurrent setting of *CTph*, we can then consider commutative processes.

Definition 15 (Abstract sequence). *Equivalence classes of sequences (w.r.t. diamond transformation) are called* abstract sequences *and are ranged over by* σ*. The abstract sequence of* s *is written* $[\![s]\!]$*. We also write* $\mathrm{pre}([\![s]\!]) = O(s)$ *and* $\mathrm{post}([\![s]\!]) = D(s)$ *to denote respectively the origins and the destinations of* $[\![s]\!]$*.*

Definition 16 (Abstract stable step and abstract transaction). *Given a* ZS *net* B*, an* abstract stable step *is an abstract sequence* $[\![s]\!]$ *of the underlying net* N_B*, where* s *is a stable step. An* abstract stable transaction *is an abstract sequence of* N_B *that contains only stable transactions of* B*. We denote by* Υ_B *the set of all abstract stable transactions of* B*.*

The equivalence induced by diamond transformation preserves stable steps (because the diamond transformation preserves the properties of concurrent enabling and of stable fairness required by Definition 14) but does not preserve stable transactions. Generally speaking, the problem is that two stable transactions that are concurrently enabled could be interleaved in such a way that the resulting sequence is a stable transaction. Of course, such transaction cannot be considered as a representative of an atomic activity of the system, because it can be expressed in terms of two concurrent sub-activities. Therefore, we take as representatives of abstract stable transactions all those stable transactions whose equivalence classes contain only stable transactions.

Abstract semantics under the *CTph*. It is now possible to define abstract representatives of those systems modeled by ZS nets in terms of PT nets whose transitions are abstract stable transactions.

Definition 17 (Abstract net). *Given a* ZS *net* $B = (S_B, T_B, F_B, u_B, Z_B)$*, its* abstract net *is the net* $A_B = (S_B \setminus Z_B, \Upsilon_B, F, u_B)$*, with* $F(a,\sigma) = \mathrm{pre}(\sigma)(a)$ *and* $F(\sigma,a) = \mathrm{post}(\sigma)(a)$ *for all* $a \in S_B \setminus Z_B$ *and* $\sigma \in \Upsilon_B$*.*

Example 2. Consider the following firing sequences of the underlying net N_{MS} of the ZS net MS in Figure 5: $s_{new} = \{c\}\ [t_0\rangle\ \{a,c\}$, $s_{res} = \{b\}\ [t_4\rangle\ \{a\}$, $s_1 = \{2a\}\ [t_1\rangle\ \{a,b,z\}\ [t_3\rangle\ \{2b\}$, $s_2 = \{3a\}\ [t_1\rangle\ \{2a,b,z\}\ [t_2\rangle\ \{2a,b,2z\}\ [t_3\rangle\ \{a,2b,z\}\ [t_3\rangle\ \{3b\}, \ldots,$

$$s_i = \{(i+1)a\}\ [t_1\rangle\ \{ia,b,z\}\underbrace{[t_2\rangle \cdots [t_2\rangle}_{i-1}\{ia,b,iz\}\underbrace{[t_3\rangle \cdots [t_3\rangle}_{i}\{(i+1)b\}, \cdots$$

We have $\Upsilon_{MS} = \{t'_0, t'_4, \sigma_1, \ldots, \sigma_i, \ldots\}$ with $t'_0 = [\![s_{new}]\!]$, $t'_4 = [\![s_{res}]\!]$ and $\sigma_i = [\![s_i]\!]$, for $i \geq 1$. The abstract net A_{MS} of MS is shown in Figure 6. It consists of three places and infinitely many transitions: One transition for creating a new active process, one for reactivating a process after a synchronization, and one for each possible multicasting involving a different number of receivers.

Proposition 4. *The reachable markings of* A_B *and of* B *are the same.*

Universal Constructions in the *CTph*. We recast the operational and abstract (*CTph*) semantics of ZS nets in a categorical framework via two universal constructions. The first construction starts from the category **dZPetri** (where ZS nets are seen as programs) and exhibits an adjunction to a category **HCatZPetri** consisting of machines equipped with suitable operations on states and transitions (e.g., parallel composition and a special kind of sequential composition, called *horizontal*). This adjunction corresponds to the operational semantics of ZS nets, in the sense that the transitions of the machine $\mathscr{Z}[B]$ associated to a ZS net B are exactly the abstract stable steps of B. Moreover, abstract stable transactions can be characterized algebraically as special transitions of $\mathscr{Z}[B]$, called *prime arrows*. The second construction starts from a different category **ZSN** of ZS nets (strictly related to **HCatZPetri**), having the ordinary category **Petri** of PT nets as a subcategory, and yields a coreflection that recovers the abstract net construction in Definition 17. We remark that **ZSN** allows one to map transitions of a machine into prime arrows of another machine, yielding a very general notion of 'implementation morphism.'

Definition 18 (Category HCatZPetri). *A* ZS graph

$$H = ((L \cup Z)^{\oplus}, (T, \oplus, 0, id, \cdot), \mathrm{pre}, \mathrm{post})$$

is both a ZS *net and a reflexive Petri commutative monoid.*[4] *In addition, it is equipped with a partial function* $\_ \cdot \_$*, called* horizontal composition*, such that:*

$$\frac{\alpha\colon (u,x) \to (v,y),\ \beta\colon (u',y) \to (v',z)}{\alpha \cdot \beta\colon (u \oplus u', x) \to (v \oplus v', z)}. \qquad (1)$$

Horizontal composition is associative and has identities $id_{(0,x)}\colon (0,x) \to (0,x)$ *for any* $x \in Z^{\oplus}$*. Moreover, the commutative monoidal operator* $\_ \oplus \_$ *is functorial w.r.t. horizontal composition. A* ZS graph morphism $h = (f, g_L, g_Z)\colon H \to H'$ *between two* ZS *graphs* H *and* H' *is both a* ZS *net morphism and a reflexive Petri monoid morphism such that* $f(\alpha \cdot \beta) = f(\alpha) \cdot f(\beta)$. ZS *graphs (as objects) and their morphisms (as arrows) form the category* **HCatZPetri**.

Horizontal composition acts as a sequential composition on zero places and as a parallel composition on stable places.

Proposition 5. *If* $\alpha\colon (u,0) \to (v,0)$ *and* $\alpha'\colon (u',0) \to (v',0)$ *are two transitions of a* ZS *graph then* $\alpha \cdot \alpha' = \alpha \oplus \alpha'$.

Theorem 1. *Let* $\mathscr{U}$:**HCatZPetri** $\to$ **dZPetri** *be the functor which forgets about the additional structure on transitions, i.e.,*

$$\mathscr{U}[((L \cup Z)^{\oplus}, (T, \oplus, 0, id, \cdot), \mathrm{pre}, \mathrm{post})] = (L^{\oplus} \times Z^{\oplus}, T, \mathrm{pre}, \mathrm{post}).$$

The functor $\mathscr{U}$ *has a left adjoint* $\mathscr{Z}$:**dZPetri** $\to$ **HCatZPetri**.

---

[4] A *reflexive Petri commutative monoid* is a Petri net together with a function $id\colon S^{\oplus} \to T$, where the set of transitions is a commutative monoid $(T, \oplus, 0)$ and pre, post and id are monoid homomorphisms, with $\mathrm{pre}(id(x)) = \mathrm{post}(id(x)) = x$.

$$\frac{(u,x)\in L_B^{\oplus}\times Z_B^{\oplus}}{id_{(u,x)}\colon(u,x)\to(u,x)\in\mathscr{Z}[B]}\qquad\frac{\alpha\colon(u,x)\to(v,y),\ \beta\colon(u',x')\to(v',y')\in\mathscr{Z}[B]}{\alpha\oplus\beta\colon(u\oplus u',x\oplus x')\to(v\oplus v',y\oplus y')\in\mathscr{Z}[B]}$$

$$\frac{t\colon(u,x)\to(v,y)\in T_B}{t\colon(u,x)\to(v,y)\in\mathscr{Z}[B]}\qquad\frac{\alpha\colon(u,x)\to(v,y),\ \beta\colon(u',y)\to(v',z)\in\mathscr{Z}[B]}{\alpha\cdot\beta\colon(u\oplus u',x)\to(v\oplus v',z)\in\mathscr{Z}[B]}$$

Table 7. Free construction of $\mathscr{Z}[B]$.

The functor $\mathscr{Z}$: **dZPetri** → **HCatZPetri** maps a ZS net B into the ZS graph which is defined by the inference rules in Table 7 modulo suitable axioms: Transitions form a commutative monoid (with $\oplus$ and $id_{(0,0)}$); the horizontal composition $\_\cdot\_$ is associative and has identities $id_{(0,x)}$; finally, the monoidal operator $\_\oplus\_$ is functorial w.r.t. horizontal composition and identities.

Example 3. Let MS be the ZS net defined in Section 2.2. The arrow $t_1\cdot t_3\in\mathscr{Z}[MS]$ has source $(2a,0)$ and target $(2b,0)$, while $(t_1\oplus id_{(a,0)})\cdot(id_{(b,0)}\oplus t_3)$ goes from $(3a\oplus b,0)$ to $(a\oplus 3b,0)$. As another example, the following expressions all denote the same arrow (i.e., the one-to-three communication):

$$\begin{aligned}t_1\cdot t_2\cdot(t_2\oplus t_3)\cdot(t_3\oplus t_3) &= t_1\cdot t_2\cdot(t_2\oplus id_{(0,z)})\cdot(id_{(0,2z)}\oplus t_3)\cdot(t_3\oplus t_3)\\ &= t_1\cdot t_2\cdot(t_2\oplus id_{(0,z)})\cdot(t_3\oplus t_3\oplus t_3)\\ &= t_1\cdot t_2\cdot(t_2\oplus id_{(0,z)})\cdot(t_3\oplus id_{(0,2z)})\cdot(t_3\oplus id_{(0,z)})\cdot t_3.\end{aligned}$$

Definition 19 (Prime arrow). *An arrow $\alpha\colon(u,0)\to(v,0)$ of a* ZS *graph H is* prime *if it cannot be expressed as the monoidal composition of nontrivial arrows (i.e., $\alpha=\beta\otimes\gamma$ implies that $\beta=id_{(0,0)}$ or $\gamma=id_{(0,0)}$).*

The following theorem defines the correspondence between the algebraic and operational semantics of ZS nets.

Theorem 2. *Given a* ZS *net B, there is a bijection between arrows $\alpha\colon(u,0)\to(v,0)$ in $\mathscr{Z}[B]$ and abstract stable steps of B. Moreover, if such an arrow is prime then the corresponding abstract stable step is an abstract stable transaction.*

Example 4. The prime arrows in $\mathscr{Z}[MS]$ are $\tau_0=t_0$, $\tau_4=t_4$, $\alpha_1=t_1\cdot t_3$, $\alpha_2=t_1\cdot t_2\cdot(t_3\oplus id_{(0,z)})\cdot t_3$, ..., $\alpha_i=t_1\cdot t_2\cdot(t_2\oplus t_3)\cdot\ldots\cdot(t_2\oplus t_3)\cdot(t_3\oplus t_3)$, and so on, where the expression $(t_2\otimes t_3)$ appears exactly $i-2$ times in α_i.

To characterize the abstract semantics, we introduce a category **ZSN** of ZS nets, where the morphisms may map a transition into a transaction.

Definition 20. *An* abstract transition *of a* ZS *net B is either a prime arrow of $\mathscr{Z}[B]$ or a transition of B (seen as an arrow in $\mathscr{Z}[B]$).*

Definition 21 (Category ZSN). *Given two* ZS *nets B and B', a* refinement morphism *$h{:}B\to B'$ is a* ZS *net morphism $(f,g_L,g_Z){:}B\to\mathscr{U}[\mathscr{Z}[B']]$ such that the function f*

maps transitions into abstract transitions. The category **ZSN** *has* ZS *nets as objects and refinement morphisms as arrows. The composition between two refinement morphisms* $h{:}B \to B'$ *and* $h'{:}B' \to B''$ *is defined as the* ZS *net morphism* $\mathscr{U}[\widetilde{h'}] \circ h{:}B \to \mathscr{U}[\mathscr{Z}[B'']]$*, where* $\widetilde{h'}{:}\mathscr{Z}[B'] \to \mathscr{Z}[B'']$ *is the unique extension of* h' *to a morphism in* **HCatZPetri***.*

Theorem 3. *The Category* **Petri** *is embedded into* **ZSN** fully *and* faithfully *as a coreflective subcategory and the right adjoint functor* $\mathscr{A}[\_]$ *is such that* $\mathscr{A}[B] = A_B$ *for any* ZS *net B. Furthermore, the counit component* ε^C_B *maps transitions of the abstract net into appropriate abstract transactions.*

The universal property of the coreflection witnesses that A_B is the PT net that better approximates the abstract *CTph* behaviour of B.

2.4 Individual token approach

In this section, the basic activities of ZS nets are defined accordingly to the *ITph*. This choice has a great impact on the resulting notion of transaction.

Operational semantics under the *ITph*. In the *ITph*, a marking can be thought of as an indexed (over the places) collection of ordered sequences of tokens and each firing must exactly specify *which* tokens are consumed.

In [14], inspired by [44], we presented a *stack based approach* to the implementation of *ITph* states. The idea was to choose a canonical interpretation of the tokens that have to be consumed and produced in a firing and to introduce *permutation firings* with the task of rearranging ordered tokens: A marking is represented as a collection of stacks, one for each place and thus the extraction and the insertion of tokens follow the LIFO policy. However, permutation firings can modify the token positions in the stacks. Causal firings were essentially introduced as a concrete means to describe the token flow, providing an intuitive grasp of the underlying mechanism. In this presentation, we prefer to resort to the more compact algebraic notation given in Section 1.1, that precisely denotes concatenable processes. For the interested reader, causal firing sequences can be thought of as arrows in $\mathcal{P}(N_B)$ having the form

$$\omega = s_0; (t_1 \otimes id_{u_1}); s_1; (t_2 \otimes id_{u_2}); s_2; \ldots ; (t_n \otimes id_{u_n}); s_n,$$

where the s_i are permutations, the t_i are transitions, and the u_i are suitable markings. In the following, we shall keep the terminology of causal firing sequences for such arrows. For ω a causal firing sequence, we denote by $pr(\omega)$ the concatenable process it represents (considered up to isomorphism).

Example 5. Let N_{MS} be the underlying net of the ZS net MS defined in Figure 5 (i.e., in N_{MS} we do not distinguish between stable and zero places). The concatenable processes associated to the sequences

$$\begin{aligned}
\omega &= (t_0 \otimes id_b); (t_4 \otimes id_{a\oplus c}); (t_1 \otimes id_{a\oplus c}); (t_3 \otimes id_{b\oplus c}){:}\, b \oplus c \to 2b \\
\omega' &= (t_4 \otimes id_c); (t_0 \otimes id_a); (t_1 \otimes id_{a\oplus c}); (t_3 \otimes id_{b\oplus c}){:}\, b \oplus c \to 2b \\
\omega'' &= (t_0 \otimes id_b); (t_4 \otimes id_{a\oplus c}); (\gamma_{a,a} \otimes id_c); (t_1 \otimes id_{a\oplus c}); (t_3 \otimes id_{b\oplus c}){:}\, b \oplus c \to 2b
\end{aligned}$$

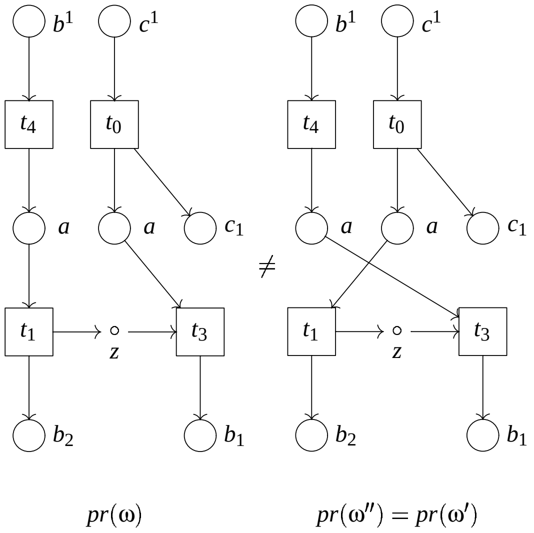

Figure 8. The concatenable processes for ω, ω', and ω'' of Example 5.

are presented in Figure 8. We use the standard notation that labels the places and transitions of the occurrence net K with their images in N. A superscript for each initial place and a subscript for each final place denote, respectively, the value of ${}^\circ\ell$ and ℓ°.

Before continuing, let us introduce some terminology that will be used in defining the *ITph* semantics of ZS nets. A process is *full* if it does not contain idle (i.e., isolated) places. A process is *active* if it includes at least one transition, *inactive* otherwise. An active process is *decomposable into parallel activities* if it is the parallel composition of two (or more) active processes. If such a decomposition does not exist, then the process is called *connected*. A connected process may involve idle places, but it does not admit globally disjoint activities (i.e., the adjective refers to activities and not to states). Finally, the set of *evolution places* (that represent resources which are first produced and then consumed) of a process C is the set $E_C = \{P(a) \mid a \in K,\ |{}^\bullet a| = |a^\bullet| = 1\}$.

To forget about the ordering functions of origins and destinations we can quotient concatenable processes modulo the underlying Goltz-Reisig processes.

Definition 22. *Let N be a net. Two causal firing sequences* ω *and* ω' *are* causally equivalent, *written* $\omega \approx \omega'$ *if* $pr(\omega) = (P{:}K \to N,\ {}^\circ\ell, \ell^\circ)$ *and* $pr(\omega') = (P'{:}K' \to N,\ {}^\circ\ell', \ell'^\circ)$ *with process P isomorphic to P′. The equivalence class of* ω *is denoted by* $[\![\omega]\!]_\approx$*. We use* ξ *to range over equivalence classes. Since the relation* $\approx$ *respects the initial and final marking, we extend the notation letting* $O(\xi) = O(\omega)$ *and* $D(\xi) = D(\omega)$*, for* $\xi = [\![\omega]\!]_\approx$.

In the *ITph*, state changes are given in terms of *connected steps*, which may involve the concurrent execution and synchronization of several transitions. A connected transaction is a connected step such that no intermediate marking is stable, and which consumes all the available stable tokens of the starting state.

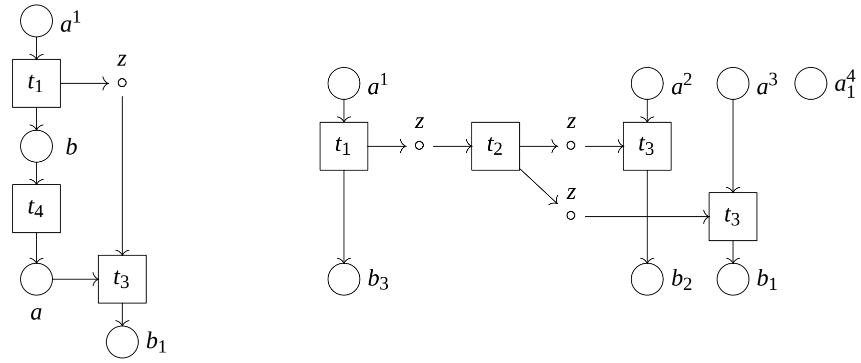

Figure 9. The concatenable processes $pr(\omega_1)$ (left) and $pr(\omega_2)$ (right).

Definition 23 (Connected step and transaction). *Given a* ZS *net B, let ω be a causal firing sequence of the underlying* PT *net N_B. The equivalence class $\xi = [\![\omega]\!]_\approx$ is a* connected step *of B, written $O(\xi)[\![\xi\rangle D(\xi)$, if:*

- *$O(\omega)$ and $D(\omega)$ are stable markings (stable fairness);*
- *$E_{pr(\omega)} \subseteq Z_B$ (atomicity).*

Furthermore, the connected step ξ is a connected transaction *of B if:*

- *$pr(\omega)$ is connected;*
- *$pr(\omega)$ is full.*

We denote by Ξ_B (ranged by δ) the set of connected transactions of B.

A *connected step sequence* is a sequence $u_0[\![\xi_1\rangle u_1 \ldots u_{n-1}[\![\xi_n\rangle u_n$ of connected steps, and we then say that u_n is reachable from u_0. Connected steps differ from stable steps in that they allow for a finer causal relationship among events. Fullness ensures the absence of idle resources in connected transactions. Note that all conditions in Definition 23 impose constraints only over the Goltz-Reisig process associated with $pr(\omega)$.

Example 6. Let us consider the ZS net *MS* in Figure 5 and the causal firing sequences

$$\begin{aligned}
\omega_1 &= t_1; (t_4 \otimes id_z); t_3 : a \to b \\
\omega_2 &= (t_1 \otimes id_{3a}); (t_2 \otimes id_{3a\oplus b}); (t_3 \otimes id_{2a\oplus b\oplus z}); (t_3 \otimes id_{a\oplus 2b}) : 4a \to a \oplus 3b \\
\omega_3 &= (t_1 \otimes id_{a\oplus c}); (t_3 \otimes id_{b\oplus c}); (t_0 \otimes id_{2b}) : 2a \oplus c \to a \oplus 2b \oplus c.
\end{aligned}$$

The equivalence class $[\![\omega_1]\!]_\approx$ is not a connected step since the 'atomicity' requirement is not fulfilled (Figure 9, left). The equivalence class $[\![\omega_2]\!]_\approx$ is a connected step but not a connected transaction since the associated process is connected but not full (Figure 9, right). Likewise, $[\![\omega_3]\!]_\approx$ is a connected step but not a connected transaction since the associated process is not connected (Figure 10). The equivalence class of the causal firing sequence $(t_1 \otimes id_{4a}); (t_2 \otimes id_{4a\oplus b}); (t_2 \otimes id_{4a\oplus b\oplus z}); (t_2 \otimes id_{4a\oplus b\oplus 2z}); (t_3 \otimes id_{3a\oplus b\oplus 3z}); (t_3 \otimes id_{2a\oplus 2b\oplus 2z}); (t_3 \otimes id_{a\oplus 3b\oplus z}); (t_3 \otimes id_{4b}) : 5a \to 5b$ is a connected transaction.

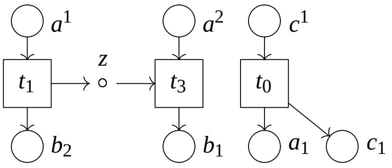

Figure 10. The concatenable process $pr(\omega_3)$.

Abstract semantics under the *ITph*. In the *ITph* based approach it is also possible to define an abstract view of the systems modeled via ZS nets. Since transactions rewrite multisets of stable tokens, PT nets are again a natural candidate for the abstract representation.

Definition 24 (Causal abstract net). *Let B be a* ZS *net. The net* $I_B = (S_B \setminus Z_B, \Xi_B, F, u_B)$, *with* $F(a,\delta) = \mathrm{pre}(\delta)(a)$ *and* $F(\delta,a) = \mathrm{post}(\delta)(a)$ *for all* $a \in S_B \setminus Z_B$ *and* $\delta \in \Xi_B$, *is the* causal abstract net *of B.*

Proposition 6. *The reachable markings of* I_B *and of B are the same.*

Example 7. Let *MS* be the ZS net in Figure 5. Its causal abstract net I_{MS} is shown in Figure 7. Transition t'_0 is the basic activity which creates a new communicating process and it corresponds to $[\![t_0]\!]_\approx$. Similarly $t'_4 = [\![t_4]\!]_\approx$. Each σ_i^k describes a different one-to-*i* communication. The index *k* identifies the copy policy under consideration. For each *i*, we denote by c_i the number of different copy policies for the communication one-to-*i* and we have a bijective correspondence among copy policies and the *complete* binary trees[5] with exactly *i* leaves.

Universal Constructions in the *ITph*. In this section, analogously to what has been done for the *CTph*, we present the categorical constructions that characterize the operational and abstract semantics of ZS nets under the *ITph*. The first adjunction goes from **dZPetri** to a category **ZSCGraph** of more structured models, called ZS *causal graphs*, equipped not only with parallel and horizontal compositions as in **HCatZPetri**, but also with a family of *swappings* playing the role of zero token permutations. Again, the connected transactions are characterized as prime arrows of ZS causal graphs. The second construction starts from a category **ZSC** of ZS nets and more complex morphisms, having the ordinary category **Petri** of PT nets as a subcategory, and yields a coreflection that recovers exactly the construction of the causal abstract net.

Definition 25 (Category ZSCGraph). *A* ZS causal graph

$$E = ((L \cup Z)^{\oplus}, (T, \otimes, 0, id, *, e_{\_,\_}), \mathrm{pre}, \mathrm{post})$$

---

[5] We recall that a binary tree is *complete* if any internal node has exactly two children and we do not distinguish between *left* and *right* children.

$$\frac{(u,x) \in L_B^{\oplus} \times Z_B^{\oplus}}{id_{(u,x)}\colon (u,x) \to (u,x) \in \mathscr{CG}[B]} \qquad \frac{z,z' \in Z_B}{d_{z,z'}\colon (0,z\oplus z') \to (0,z'\oplus z) \in \mathscr{CG}[B]}$$

$$\frac{t\colon (u,x) \to (v,y) \in T_B}{t\colon (u,x) \to (v,y) \in \mathscr{CG}[B]} \qquad \frac{\alpha\colon (u,x) \to (v,y),\ \beta\colon (u',y) \to (v',z) \in \mathscr{CG}[B]}{\alpha * \beta\colon (u\oplus u',x) \to (v\oplus v',z) \in \mathscr{CG}[B]}$$

$$\frac{\alpha\colon (u,x) \to (v,y),\ \beta\colon (u',x') \to (v',y') \in \mathscr{CG}[B]}{\alpha \otimes \beta\colon (u\oplus u',x\oplus x') \to (v\oplus v',y\oplus y') \in \mathscr{CG}[B]}$$

Table 8. Free construction of $\mathscr{CG}[B]$.

---

is both a ZS *net and a reflexive Petri monoid. In addition, it comes equipped with a partial function* $\_ * \_$ *called* horizontal composition,

$$\frac{\alpha\colon (u,x) \to (v,y),\ \beta\colon (u',y) \to (v',y')}{\alpha * \beta\colon (u\oplus u',x) \to (v\oplus v',y')},$$

and a family of horizontal swappings, $\{e_{x,y}\colon (0,x\oplus y) \to (0,y\oplus x)\}_{x,y\in Z^{\oplus}}$. *Horizontal composition is associative and has identities* $id_{(0,x)}$ *for all* $x \in Z^{\oplus}$. *The monoidal operator* $\_ \otimes \_$ *is functorial w.r.t. horizontal composition. The (horizontal)* naturality *axiom,* $e_{x,x'} * (\beta\otimes\alpha) = (\alpha\otimes\beta) * e_{y,y'}$ *holds for any* $\alpha\colon (u,x) \to (v,y)$ *and* $\beta\colon (u',x') \to (v',y')$. *Furthermore, the* coherence *axioms* $e_{x,y} * e_{y,x} = id_{(0,x\oplus y)}$ *and* $e_{x,y\oplus y'} = (e_{x,y} \otimes id_{(0,y')}) * (id_{(0,y)} \otimes e_{x,y'})$ *must be satisfied. A morphism h between two* ZS *causal graphs E and E′ is a* ZS *net monoidal morphism which in addition respects horizontal composition and swappings. This defines the category* **ZSCGraph**.

Again, horizontal composition is the key feature of the approach: It avoids the construction of steps which reuse stable tokens.

Proposition 7. *If* $\alpha\colon (u,0) \to (v,0)$ *and* $\alpha'\colon (u',0) \to (v',0)$ *are two transitions of a* ZS *causal graph then* $\alpha\otimes\alpha' = \alpha'\otimes\alpha$ *and* $\alpha * \alpha' = \alpha\otimes\alpha'$.

The next theorem defines the algebraic semantics of ZS nets by means of a universal property.

Theorem 4. *The obvious forgetful functor* $\mathscr{U}$: **ZSCGraph** → **dZPetri** *has a left adjoint* $\mathscr{CG}$: **dZPetri** → **ZSCGraph**.

The functor $\mathscr{CG}$ maps a ZS net B into the ZS causal graph $\mathscr{CG}[B]$ whose arrows are generated by the inference rules in Table 8 modulo suitable axioms (see [17] for details). The ZS causal graph $\mathscr{CG}[B]$ is still too concrete w.r.t. the operational (*ITph*) semantics of ZS nets. More precisely we need two more axioms.

Definition 26. *Given a* ZS *net B, we denote by* $\mathscr{CG}[B]/\Psi$ *the quotient of the free* ZS *causal graph* $\mathscr{CG}[B]$ *generated by B in* **ZSCGraph** *modulo the axioms*

$$d_{z,z'} = id_{(0,z\oplus z')},\ \text{if } z \neq z' \in Z_B, \tag{2}$$

$$d * t * d' = t,\ \text{if } t \in T_B \text{ and } d,d' \text{ are swappings}. \tag{3}$$

The quotient $\mathscr{C}\mathscr{G}[B]/\Psi$ is such that for any $k{:}\mathscr{C}\mathscr{G}[B] \to E \in$ **ZSCGraph** which respects axioms (2) and (3), there exists a unique arrow $k_\Psi{:}\mathscr{C}\mathscr{G}[B]/\Psi \to E$ such that $k_\Psi \circ Q_\Psi = k$ in **ZSCGraph**, where $Q_\Psi{:}\mathscr{C}\mathscr{G}[B] \to \mathscr{C}\mathscr{G}[B]/\Psi$ is the obvious ZS causal graph morphism associated to the (least) congruence generated by the imposed axiomatization.

Proposition 8. *For any morphism $h{:}B \to B'$ in* **dZPetri** *there is a unique extension $\hat{h}{:}\mathscr{C}\mathscr{G}[B]/\Psi \to \mathscr{C}\mathscr{G}[B']/\Psi$ of h in* **ZSCGraph***.*

Example 8. Let MS be the ZS net defined in Section 2.2. The arrow $t_1 * t_3 \in \mathscr{C}\mathscr{G}[MS]/\Psi$ has source $(2a,0)$ and target $(2b,0)$, while $(t_1 \otimes id_{(a,0)}) * (id_{(b,0)} \otimes t_3)$ goes from $(3a \oplus b,0)$ to $(a \oplus 3b,0)$. As another example, all the following expressions denote the same arrow:

$$\begin{aligned}
t_1 * t_2 * (t_2 \otimes t_3) * (t_3 \otimes t_3) &= t_1 * t_2 * (t_2 \otimes id_{(0,z)}) * (t_3 \otimes t_3 \otimes t_3)\\
&= t_1 * t_2 * d_{z,z} * (t_2 \otimes id_{(0,z)}) * (t_3 \otimes t_3 \otimes t_3)\\
&= t_1 * t_2 * (id_{(0,z)} \otimes t_2) * (t_3 \otimes t_3 \otimes t_3)\\
&= t_1 * t_2 * (t_3 \otimes t_2) * (t_3 \otimes t_3).
\end{aligned}$$

To give the expected correspondence between algebraic and operational semantics we reuse in the current setting the notion of prime arrows.

Theorem 5. *Given a* ZS *net B, there is a one-to-one correspondence between arrows $\alpha{:}(u,0) \to (v,0) \in \mathscr{C}\mathscr{G}[B]/\Psi$ and the connected steps of B. Moreover, if such an arrow is prime (and is not an identity) then the corresponding connected step is a connected transaction.*

Example 9. In our running example, some prime arrows of $\mathscr{C}\mathscr{G}[MS]$ are t_0, $t_1 * t_3$, and $t_1 * t_2 * (t_2 \otimes t_2) * (t_3 \otimes t_2 \otimes t_3 \otimes t_3) * (t_3 \otimes t_3)$, while the arrow $(t_1 \otimes t_1) * d_{z,z} * (t_2 \otimes t_3) * (t_3 \otimes t_3)$ is not prime.

To recover the abstract semantics of ZS nets in the *ITph*, we define a category **ZSC** whose objects are ZS nets and whose morphisms allow for the refinement of a transition into an abstract connected transaction.

Definition 27. *Given a* ZS *net B, a* causal abstract transition *of $\mathscr{C}\mathscr{G}[B]/\Psi$ is either a prime arrow of $\mathscr{C}\mathscr{G}[B]/\Psi$ or a transition of B (seen as arrow).*

Definition 28 (Category ZSC). *Given two* ZS *net B and B', a* causal refinement morphism *$h{:}B \to B'$ is a* ZS *net morphism $h = (f, g_L, g_Z)$ from B to (the image through the forgetful functor of) $\mathscr{C}\mathscr{G}[B']/\Psi$, such that function f maps transitions into causal abstract transitions. The category* **ZSC** *has* ZS *nets as objects and causal refinement morphisms as arrows, with composition defined similarly to that in* **ZSN***.*

Theorem 6. *Category* **Petri** *is embedded in* **ZSC** *fully and faithfully as a coreflective subcategory and the right adjoint functor $\mathscr{I}[\_]$ is such that $\mathscr{I}[B] = I_B$ for any* ZS *net B. Furthermore, the counit component of the coreflection ε^I_B maps each transition of the abstract net into the appropriate connected transaction.*

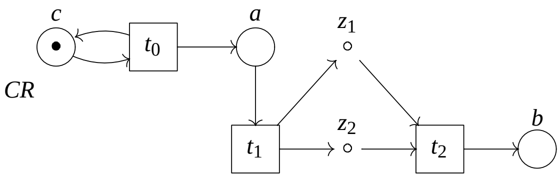

Figure 11.

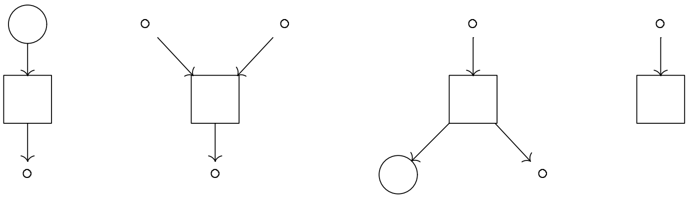

Figure 12.

2.5 One more example

We remark that the impact of different philosophies on the modeled system is considerable. This has been already suggested by the multicasting system example, but there are many other examples where the dichotomy is immediate. Let us consider the ZS net *CR* in Figure 11. Then, according to the *CTph* the abstract net A_{CR} has only two transitions that correspond to t_0 and $t_1 \cdot t_2$, whereas, according to the *ITph* the causal abstract net I_{CR} has infinitely many transitions: t_0, $t_1 * t_2$, $(t_1 \otimes t_1) * d_{z,3z} * (t_2 \otimes t_2)$, and so on. Note the analogy between I_{CR} and the abstract net A_{MS} of the multicasting system example.

We end this section by observing that all PT nets can be constructed using ZS nets whose transitions have four fixed 'shapes.' The needed components are illustrated in Figure 12. We leave to the reader, as an easy task, to combine these shapes for building a generic transition, though of course several other sets of building blocks could have been chosen.

3 Translation of languages with synchronization primitives

In this section we give some general hints for modeling CCS-like communication via ZS nets. The idea is to represent each channel by a pair of zero places, one for input and one for output, and to model each input (output) action on a channel with a transition that produces a token on the input (output) zero place associated to that channel. A special transition, also associated to the channel, is enabled by a token in the input and a token in the output zero place. If the channel is restricted, this is the only transition that can consume those tokens, thus synchronizing the input and output actions that produced the two tokens. If the channel is not restricted, two additional transitions can consume

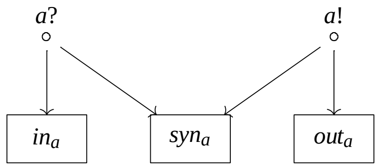

Figure 13. The ZS net Z_a.

$$\frac{}{\lambda.p \xrightarrow{\lambda} p} \qquad \frac{p \xrightarrow{\mu} q}{p|r \xrightarrow{\mu} q|r} \qquad \frac{p \xrightarrow{\lambda} q,\ p' \xrightarrow{\bar{\lambda}} q'}{p|p' \xrightarrow{\tau} q|q'} \qquad \frac{p \xrightarrow{\mu} q}{r|p \xrightarrow{\mu} r|q} \qquad \frac{p \xrightarrow{\mu} q \quad \mu \notin \{a,\bar{a}\}}{p\backslash a \xrightarrow{\mu} q\backslash a}$$

Table 9. SOS rules for the simple process algebra SPA.

---

the tokens separately. Thus, for every channel name a we define a ZS net Z_a consisting of two zero places $a!$ and $a?$, and three transitions in_a, syn_a, and out_a (see Figure 13). For a set $\mathbf{A} = \{a_1,...,a_n\}$ of channel names, we denote by $Z(\mathbf{A})$ the ZS net obtained as the disjoint union of $Z_{a_1},\ldots,Z_{a_n}$.

Definition 29 (Interfaced net). *Given a set* $\mathbf{A} = \{a_1,...,a_n\}$ *of channel names, an* $\mathbf{A}$-interfaced net *is a triple* $\langle B,\mathbf{A},P\rangle$*, where* B *is a* ZS *net — in our translation the initial marking will always be a set — and* P *is an injective mapping from* $Z(\mathbf{A})$ *to* B*, which preserves the* ZS *net structure. The set* $\mathbf{A}$ *is called the* interface *of the net.*

Two $\mathbf{A}$-interfaced nets $\langle B,\mathbf{A},P\rangle$ and $\langle B',\mathbf{A},P'\rangle$ are *isomorphic* if there exists a ZS net isomorphism ψ from B to B' that 'preserves interfaces' (in the sense that it must preserve the injective images of $Z(\mathbf{A})$).

The *simple process algebra* (SPA) considered in [17] is equipped with the operations of inaction *nil*, input and output action prefix $a.\_$ and $\bar{a}.\_$, parallel composition $\_|\_$, and restriction $\_\backslash a$, whose associated SOS rules are given in Table 9. (We let μ range over input (a), output ($\bar{a}$) and silent (τ) actions, and let λ range over input/output actions.) We will show later how to deal with distributed nondeterministic sum.

Each agent p is modeled by an $fv(p)$-interfaced net $[\![p]\!]_{\mathrm{ZS}}$, where the set $fv(p)$ is the set of the free (i.e., non-restricted) channel names in p. The definition of $[\![p]\!]_{\mathrm{ZS}}$ is given by initiality (i.e., it is the unique SPA-algebra homomorphism from the term algebra), and thus it is enough to define the corresponding operations on interfaced nets.

Inaction. The inactive net *nil* is a $\varnothing$-interfaced net $\langle B,\varnothing,\varnothing\rangle$, where B consists of a single place that contains one token in the initial marking.

Action prefix. The interfaced net $a.\langle B,\mathbf{A}\cup\{a\},P\rangle$ is given by adding a new stable place b and a new transition t to B. The initial marking consists of a token in b. The transition t takes a token in b and produces the initial marking of B plus a token in the zero place $P(a?)$. If the name a is not contained in the interface of the given net, then also a copy of Z_a has to be added, and the injective mapping P is extended in the obvious way. A similar construction is defined for an output action prefix $\overline{a}.p$ (we substitute $a!$ for $a?$ in the postset of the new transition t).

Parallel composition. We let $\langle B_1, \mathbf{A_1}, P_1\rangle | \langle B_2, \mathbf{A_2}, P_2\rangle = \langle B, \mathbf{A_1} \cup \mathbf{A_2}, P\rangle$, with B given by the union of B_1 and B_2 where only $P_1(Z(\mathbf{A_1} \cap \mathbf{A_2}))$ and $P_2(Z(\mathbf{A_1} \cap \mathbf{A_2}))$ are identified, and with the mapping P given by the union of P_1 and P_2. The initial marking of B is the union of the initial markings of B_1 and B_2.

Restriction. If a does not appear in the interface, then $\langle B, \mathbf{A}, P\rangle \backslash a = \langle B, \mathbf{A}, P\rangle$. Otherwise, $\langle B, \mathbf{A} \cup \{a\}, P\rangle \backslash a = \langle B', \mathbf{A}, P'\rangle$, with $B' = B \setminus \{P(in_a), P(out_a)\}$ and P' is P restricted to $Z(\mathbf{A})$.

The image of $Z(fv(p))$ in B (via P) plays the role of the interface, since it is the only part of the net $[\![p]\!]_{\mathrm{ZS}}$ that is modified by the construction defined above: It can be increased (as in the case of action prefix), it can be merged with another interface (as in the case of parallel composition) and it can also be restricted (as in the case of the restriction operator). It is worth noting that for each agent p, with $[\![p]\!]_{\mathrm{ZS}} = \langle B, \mathbf{A}, P\rangle$, we have $A_B = I_B$.

The relation between SPA agents and their associated interfaced nets can be formalized by adding a labeling function ϕ from the transitions of abstract nets to the set of actions.

Definition 30 (Labels of transactions). *Let p be an agent. For each (connected) transaction ξ of $[\![p]\!]_{\mathrm{ZS}}$, we define*

$$\phi(\xi) \overset{\text{def}}{=} \begin{cases} a_i \ \textit{if}\ a_i \in fv(p) \ \textit{and}\ P(in_{a_i}) \ \textit{is fired in}\ \xi \\ \overline{a_i} \ \textit{if}\ a_i \in fv(p) \ \textit{and}\ P(out_{a_i}) \ \textit{is fired in}\ \xi \\ \tau \ \ \textit{otherwise} \end{cases}$$

The definition of labels is not ambiguous, because each transaction ξ of $[\![p]\!]_{\mathrm{ZS}}$ contains at most one firing of transitions in $P(Z(fv(p)))$.

Definition 31 (Bisimilarity between agents and markings). *Let p be an agent, let N be a net whose transitions are labeled by ϕ over the set of actions, and let u be a marking of N. We say that p is* bisimilar *to u in N if there exists a relation $\sim$ between agents and markings of N such that $p \sim u$, and: (1) for each transition $p \xrightarrow{\mu} p'$ there exists a firing $u\,[t\rangle\, u'$ of N such that $\phi(t) = \mu$ and $p' \sim u'$; (2) for each firing $u\,[t\rangle\, u'$ of N with $\phi(t) = \mu$ there exists a transition $p \xrightarrow{\mu} p'$ such that $p' \sim u'$.*

Proposition 9. *Let p be an agent, and $[\![p]\!]_{\mathrm{ZS}} = \langle B, \mathbf{A}, P\rangle$, then p is bisimilar to the initial marking of the abstract net A_B.*

A comparison with other net semantics presented in the literature for CCS-like algebras is out of the scope of this presentation. We just remark the linearity of our encoding and that it provides a reasonable concurrent semantics for SPA agents, as formalized by Proposition 9 above.

don’t know		**don’t care**	
$\dfrac{p \xrightarrow{\lambda} q}{p+r \xrightarrow{\lambda} q}$	$\dfrac{p \xrightarrow{\lambda} q}{r+p \xrightarrow{\lambda} q}$	$\dfrac{}{p+r \xrightarrow{\tau} p}$	$\dfrac{}{r+p \xrightarrow{\tau} p}$

Table 10. Two kinds of nondeterministic choice.

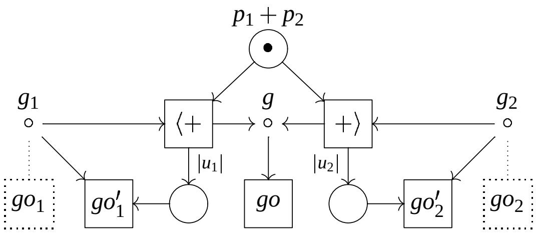

Figure 14.

Restricted names have only local scopes, and agents that differ only for local names (i.e., agents that can be obtained one from the other by α-conversion) can be considered equivalent. We use the symbol $\_ \equiv_\alpha \_$ to denote such equivalence (e.g., we have $(a_1.a_2.nil|\overline{a}_2.nil)\backslash a_2 \equiv_\alpha (a_1.a_3.nil|\overline{a}_3.nil)\backslash a_3$ for any $a_3 \neq a_1$). It is worth noting that our translation supports α-conversion.

Proposition 10. *If* $p \equiv_\alpha q$, *then* $[\![p]\!]_{\mathrm{ZS}}$ *and* $[\![q]\!]_{\mathrm{ZS}}$ *are isomorphic.*

3.1 Distributed sum

Usually, one can distinguish between two kinds of nondeterminism: *don’t know* and *don’t care*. In the former, an alternative is selected via a sort of ‘lookahead’ (e.g., only if the associated subprocess can move), whereas the latter is ‘blind.’ The difference between the two is evident just by looking at the SOS rules in Table 10, as ‘don’t care choice’ is modeled via axioms. In the next we rely on don’t know choice, which is more complicate to deal with.

To model don’t know choice in distributed implementations of CCS-like languages, the classical approach is to make a cross product of the initial markings of the subcomponents in such a way that when one thread r in one component moves, then all the threads in the other component will never be enabled since r consumes some of their premises. Of course this is an expensive construction that adds a lot of auxiliary structure causing state explosion. Using ZS nets the ‘interface’ approach described above can be exploited for accommodating a more compact solution (though the classical one is still possible).

The idea is that besides channel (zero) places, also a ‘generic action’ zero place, say g, is added that receives tokens from all transitions generated by the action prefix

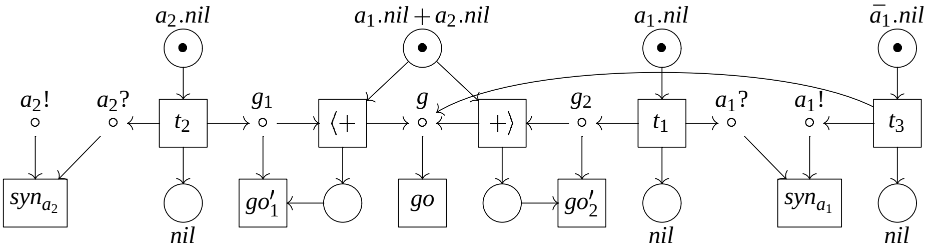

Figure 15. The interfaced net for the process $((a_1.nil + a_2.nil)|\bar{a}_1.nil)\backslash a_1\backslash a_2$.

construction. Using this place we can establish whether or not a thread can evolve. Normally, a transition *go* can consume exactly one token from *g* and produce nothing. (Both *g* and *go* are part of any interface). Since we are interested in catching 'top level' actions, every time a process is prefixed by an action we remove from the interface *g* and *go* and add new instances of them which become connected to the prefixed action only. The general situation is that we have two such interfaced nets and we want to model their nondeterministic sum. The first step is to replace the two *go* transitions by transitions that are in some way controlled by the choice-point. The relevant part of the construction is illustrated in Figure 14; for the rest we assume that the two 'argument' nets are put in parallel, merging their interfaces except that for *g* and *go* components (denoted by g_i and go_i in figure) that are carried out of the interface, while fresh *g* and *go* are inserted in the composed net. We call u_1 and u_2 the markings of the two argument nets, that form, together with a token in the place '$p_1 + p_2$' the initial marking of the composed net. To see how the construction behaves, suppose that p_1 can perform a certain action, then a zero token appears in g_1 that can be consumed only by firing '$\langle+$' since go_1 has been deleted and go'_1 is not yet enabled. The firing of '$\langle+$' consumes the only stable token in $p_1 + p_2$ and hence, all threads in the second net cannot complete any transaction (because tokens in g_2 can never be consumed). We have decided to put $|u_1|$ tokens in the stable place that rule go'_1, so that when other top level threads of p_1 will be able to fire, then enough tokens will be available to close all transactions asynchronously. Propositions 9 and 10 are still valid for this extended framework.

Example 10. The interfaced net for the agent $((a_1.nil + a_2.nil)|\bar{a}_1.nil)\backslash a_1\backslash a_2$ is presented in Figure 15. Note that the presence of the arc from t_3 to *g* is motivated by the fact that *g* (and *go*) are part of the interfaces of the two nets associated with $a_1.nil + a_2.nil$ and $\bar{a}_1.nil$ and thus *g* becomes shared when the two agents are composed in parallel. If t_2 tries to fire, then the zero token in $a_2?$ cannot be consumed. If t_1 tries to fire, then the only possibility is that also t_3 fires producing a token in $a_1!$ so that also syn_{a_1} is enabled and the token in $a_1?$ can be consumed. If this is the case, then one has still to consume the zero tokens in *g* and g_2 produced by the firings of t_3 and t_1, respectively. Thus, $+\rangle$ must be fired that produces another zero token in *g*. Then, transition *go* can be fired twice to conclude the transaction. This transaction corresponds to the transition $((a_1.nil + a_2.nil)|\bar{a}_1.nil)\backslash a_1\backslash a_2 \xrightarrow{\tau} (nil|nil)\backslash a_1\backslash a_2$, and is the only possible one.

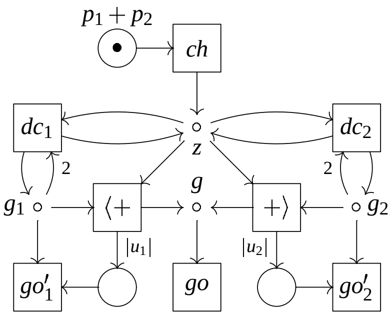

Figure 16.

Note that since '$\langle +$' and '$+\rangle$' take one token only from g_1 and g_2 respectively, then τ-moves cannot force the choice (because synchronizations produce two tokens in g_1 or g_2). This was also the reason for writing λ and not μ in the rules for don't know choice in Table 10. To deal with this possibility, it suffices to add two variants of '$\langle +$' and '$+\rangle$' that consume two tokens from g_1 and g_2 respectively. More generally, one might want to synchronize any number of threads as triggers of the same left/right choice. This can be easily done by augmenting the composed nets with one additional zero place and three transitions, as illustrated in Figure 16. The transitions dc_i have the duty of sequentially decrementing the number of tokens in g_i (takes two and puts one back) until only one token is left that can be consumed by go'_i, thus synchronizing the choice with all threads of the ith component that tried to move. Since only one token is present in the place $p_1 + p_2$, then only one token can be produced in z and the dc_i preserve this invariant under firing. Therefore, only one transition between $\langle +$ and $+\rangle$ can fire in the transaction. As a consequence, it is not possible that both dc_1 and dc_2 are fired in the same transaction, as otherwise a zero token would remain in one of the zero places g_i. Note that the *CTph* and the *ITph* can yield different abstract nets, as the latter distinguishes between the different ways for dc_i to consume the tokens in g_i. One may argue that there is a centralized choice point and that therefore two threads of p_1 cannot force concurrently the 'left choice' but must be synchronized on it. We think that this is by no means a limitation of the approach, as e.g. the classical solution requires a 'token synchronization' between each thread of one component and all other threads of the other components, via a conflict resolution similar to the one illustrated in the dining philosophers example (here the tokens in the cross product of the two initial markings of subnets are 'forks' and the threads are the 'philosophers' that want to eat).

4 An Operational Definition for Transactions

The issues that we want to address in this section regard ZS nets implementation. The problem is that the operational semantics relies on some sort of meta-definition, where one computes on the underlying net, builds transaction segments, and then can discard 'bad' behaviors and accept the 'good' ones, acting as a filter. This means that there are important questions which can be asked for any actual interpreter — Is backtracking necessary? Is the implementation correct? And complete? Does a more efficient implementation exist? We try to answer these questions (see the Conclusions) by defining a machinery for computing on ZS nets. The idea is to adapt the classical net unfolding to pursue concurrently all the nondeterministic runs of the ZS net under inspection, in such a way that 'commit' stable states are recognized and generated.

Whether one is interested in distinguishing between different concurrent proofs or is just interested in the step relation $\_ \Rrightarrow_B \_$ is an important issue. In particular, given a ZS net B and a stable marking u we address the problem of computing in a distributed and efficient fashion the set of markings that can be reached from u via an atomic transaction step, i.e., the set $\{v \mid u \Rrightarrow_B v\}$ (that is invariant under the *CTph* and *ITph*). The solution relies on a modification of the interpreter for unfolding PT nets [37,51,33], which is extended with a commit rule enforcing the synchronous termination of transactions.

4.1 PT Net Unfolding

The unfolding of a net gives a constructive way to generate all its possible computations, offering a satisfactory mathematical description of the interplay between nondeterminism and causality (and concurrency). In fact the unfolding construction allows for bridging the gap between PT nets and *prime algebraic domains*. The obvious references to this approach are [37,51,33], but we suggest also the interesting overview [25]. It is worth remarking that our presentation is slightly different from usual ones (but reminiscent of [51]), since it is presented as the least net generated by suitable inference rules, rather than by making explicit the chain of finite nets that approximate it.

The construction provides a distributed interpreter for PT nets. We remark that the unfolding applies only to marked nets, i.e., it requires an initial marking.

Starting from a net N, the unfolding produces a nondeterministic occurrence net $\mathcal{U}(N)$ (an acyclic net, where transition pre- and post-markings are sets instead of multisets and where each place has at most one entering arc), together with a mapping from $\mathcal{U}(N)$ to N that tells which places and transitions of the unfolding are instances of the same element of N. Hence the places of $\mathcal{U}(N)$ represent the tokens and the transitions (called *events*) the occurrences of transitions in all possible runs. For this kind of nets the notions of *causally dependent*, of *conflicting* and of *concurrent* elements can be straightforwardly defined and are represented by the binary relations $\_ \preceq \_$, $\_\#\_$ and $\mathbf{co}(\_,\_)$, respectively. Formally, the relation $\_ \preceq \_$ is the transitive and reflexive closure of the *immediate precedence* relation $\_ \prec_0 \_$ defined as $\prec_0 \overset{\text{def}}{=} \{(a,t) \mid a \in \text{pre}(t)\} \cup \{(t,a) \mid a \in \text{post}(t)\}$, while the binary conflict relation is defined as the minimal symmetric relation that contains $\_\#_0\_$ (defined by $t_1 \#_0 t_2 \overset{\text{def}}{\Leftrightarrow} t_1 \neq t_2 \ \wedge \ \text{pre}(t_1) \cap \text{pre}(t_2) \neq \varnothing$), and that is hereditary with respect to $\_ \preceq \_$. Since the conflict relation must be irreflexive, then $\_ \preceq \_$ and $\_\#\_$ have empty intersection. The concurrency relation is defined by letting $\mathbf{co}(o_1,o_2)$ if it is not the case that ($o_1 \prec o_2$ or $o_2 \prec o_1$ or $o_1 \# o_2$). In particular, the relation $\mathbf{co}$ is usually extended to sets of elements by writing $\mathbf{co}(X)$ if for all $o_1, o_2 \in X$ we have $\mathbf{co}(o_1,o_2)$.

More concretely, the places of $\mathcal{U}(N)$ have the form $\langle a,n,H \rangle$, where $a \in S_N$, n is a positive natural number that is used to distinguish different tokens with the same history,

$$\frac{u_{\mathrm{in}}(a) = n,\ 1 \leq k \leq n}{\langle a,k,\varnothing\rangle \in S_{\mathcal{U}(N)}}$$

$$\frac{t{:}u \to \bigoplus_{j\in J} n_j b_j \in T_N,\ \Theta = \{\langle a_i,k_i,H_i\rangle \mid i \in I\} \subseteq S_{\mathcal{U}(N)},\ \mathbf{co}(\Theta),\ u = \bigoplus_{i\in I} a_i}{e = \langle t,\Theta\rangle \in T_{\mathcal{U}(N)},\ \Upsilon = \{\langle b_j,m,\{e\}\rangle \mid j \in J,\ 1 \leq m \leq n_j\} \subseteq S_{\mathcal{U}(N)},\ \mathrm{pre}(e) = \Theta,\ \mathrm{post}(e) = \Upsilon}$$

Table 11. The unfolding $\mathcal{U}(N)$.

---

and H is the history of the place under inspection and therefore either consists of just one event (the one that produced the token) or is empty (if the token is in the initial marking). Analogously, a generic transition of $\mathcal{U}(N)$ has the form $\langle t,H\rangle$ with $t \in T_N$, since each transition is completely identified by its history H, which in this case consists of the set of consumed tokens. The set H cannot be empty since transitions with empty preset are not allowed. The net $\mathcal{U}(N)$ is defined as the minimal net generated by the rules in Table 11.

We now give a computational interpretation of such rules. The first rule defines the initial marking of $\mathcal{U}(N)$. The second rule is the core of the unfolding: It searches for a set Θ of concurrent tokens that enables a transition t of N, atomically locks them, fires the event e (that is an occurrence of t), and produces some fresh tokens Υ according to $\mathrm{post}(t)$. Notice that the condition $\mathbf{co}(\Theta)$ depends only on the histories H_i for $i \in I$, and therefore cannot be altered by successive firings. In fact, as in *memoizing* for logic programming, or more generally in *dynamic programming*, the history is completely encoded in the tokens, so that it is not necessary to compute it at every firing. Also, note that histories retain concurrent information rather than just sequential, therefore each token/event is generated exactly once (though it can be later referred to many times). Moreover, several occurrences of the second rule can be applied concurrently and therefore the unfolding can be implemented as a distributed algorithm.

4.2 ZS Net Unfolding

The unfolding of the underlying net N_B does not yield a faithful representation of the behavior of B. In fact, we must forbid the consumption of stable resources that were not inserted in the starting marking. Moreover, we must be able to apply the commit when the transaction step has consumed all the zero tokens produced so far.

The net $\mathcal{U}(B)$ is defined as the minimal net generated by the rules in Table 12. Together with the unfolding net we compute a set of (reachable) stable markings $\mathscr{R}(B,u)$ for the initial (stable) marking u of the unfolding.

The first two rules define the unfolding, which remains similar to the classical algorithm, except for the fact that stable tokens in the postset of the fired transition are not released to the system. In fact, while the set Θ must contain enough tokens to provide both the stable and the zero resources needed by t (as expressed by the condition $u \oplus x = \bigoplus_{i\in I} s_i$), the tokens that are produced by the occurrence of t applied to Θ (i.e., tokens in the set $\Upsilon = \mathrm{post}(e)$) just match the zero place component $\bigoplus_{j\in J} n_j z_j$ of $\mathrm{post}(t)$ and not the stable place component v (it is not released until a commit related to e will

$$\frac{u(a) = n,\ 1 \leq k \leq n}{\langle a,k,\varnothing\rangle \in S_{\mathcal{U}(B)}}$$

$$\frac{t\colon (u,x) \to (v, \bigoplus_{j\in J} n_j z_j) \in T_B,\ \Theta = \{\langle s_i,k_i,H_i\rangle \mid i \in I\} \subseteq S_{\mathcal{U}(B)},\ \mathbf{co}(\Theta),\ u \oplus x = \bigoplus_{i\in I} s_i}{e = \langle t,\Theta\rangle \in T_{\mathcal{U}(B)},\ \Upsilon = \{\langle z_j,m,\{e\}\rangle \mid j \in J,\ 1 \leq m \leq n_j\} \subseteq S_{\mathcal{U}(B)},\ \mathrm{pre}(e) = \Theta,\ \mathrm{post}(e) = \Upsilon}$$

$$\frac{}{u \in \mathscr{R}(B,u)} \qquad \frac{\Gamma \subseteq T_{\mathcal{U}(B)},\ \mathbf{co}(\Gamma),\ \mathbf{ZProd}(\Gamma) = \mathbf{ZCons}(\Gamma)}{u \ominus \mathbf{SCons}(\Gamma) \oplus \mathbf{SProd}(\Gamma) \in \mathscr{R}(B,u)}$$

Table 12. The unfolding $\mathcal{U}(B)$.

occur). The third rule is obvious. The fourth rule defines the commit of a transaction step.

To shorten the notation, we introduce the following functions that, given an event e, return the set of zero tokens respectively consumed and produced by the ancestors of e (and by e itself), i.e., we let

$$\mathbf{ZCons}(e) \stackrel{\text{def}}{=} \textstyle\bigcup_{\langle t,\Theta\rangle \preceq e}\{\langle z,k,H\rangle \in \Theta \mid z \in Z_B\}, \qquad \mathbf{ZProd}(e) \stackrel{\text{def}}{=} \textstyle\bigcup_{e' \preceq e} \mathrm{post}(e'),$$

where $\mathbf{ZCons}(e)$ is the set of zero tokens that have been consumed by some $e' \preceq e$; similarly $\mathbf{ZProd}(e)$ represents the set of zero tokens that have been produced by some $e' \preceq e$ (note that for any $\langle z,k,H\rangle \in \mathbf{ZCons}(e)$ we have $\langle z,k,H\rangle \preceq e$, while $\mathbf{ZProd}(e)$ can also contain tokens that are concurrent with e or produced by e). We remark that $\mathbf{ZCons}(e) \subseteq \mathbf{ZProd}(e)$, because the marking u is stable and therefore it does not contain zero tokens with empty histories. For stable places the situation is different, since we are just interested in knowing how many tokens have been consumed and will be produced for each place by the antecedents of e, thus:

$$\mathbf{SCons}(e) \stackrel{\text{def}}{=} \bigoplus_{\langle t:(u,x)\to(v,y),\Theta\rangle \preceq e} u, \qquad \mathbf{SProd}(e) \stackrel{\text{def}}{=} \bigoplus_{\langle t:(u,x)\to(v,y),\Theta\rangle \preceq e} v.$$

The four functions that we have defined are extended to sets of events in the obvious way. We remark that while $\mathbf{ZCons}(\_)$ and $\mathbf{ZProd}(\_)$ return sets (of zero places in the unfolding net), the functions $\mathbf{SCons}(\_)$ and $\mathbf{SProd}(\_)$ return multisets (of stable places in the original net).

The fourth rule takes a set Γ of concurrent events and checks that any zero token produced by their antecedents is consumed by an antecedent of some event in Γ. The latter condition can be conveniently expressed as the equality $\mathbf{ZProd}(\Gamma) = \mathbf{ZCons}(\Gamma)$. In fact, if a certain token o is in $\mathbf{ZProd}(\Gamma)$, then the condition states that there exists at least an event $e \in \Gamma$ and a unique[6] $e' \preceq e$ such that $o \in \mathrm{pre}(e)$. If these premises are satisfied, then the rule extends $\mathscr{R}(B,u)$ with the multiset obtained by subtracting from u the stable resources consumed by all the antecedents of events in Γ, but adding those

---

[6] Otherwise a conflict would arise.

that would have been produced during the step. This rule defines a commit, since it synchronizes local commits, as the following result shows.

Proposition 11. *If* $\Gamma \subseteq T_{\mathcal{U}(B)}$ *such that* $\mathbf{co}(\Gamma)$ *and* $\mathbf{ZProd}(\Gamma) = \mathbf{ZCons}(\Gamma)$*, then for any* $e = \langle t, \Theta \rangle \in \Gamma$ *we have that* t *does not produce any zero token.*

Note the analogy between the 'commit' rule that takes a set of concurrent events and the 'unfolding' rule that takes a set of concurrent tokens: This is to some extent related to our view of ZS nets as a formalism for expressing transition synchronization rather that just token synchronization.

The resulting algorithm is as much distributed as the classical one when applied to the abstract net of B. In fact all the useful relations are defined by just looking at the history of the elements in the premises, which, under the atomicity assumption reduce to the stable preset of the abstract step. To improve efficiency, the sets $\mathbf{ZProd}(e)$, $\mathbf{ZCons}(e)$, $\mathbf{SProd}(e)$ and $\mathbf{SCons}(e)$ could be also encoded in e more directly, although they can be easily calculated from the history component. The main result can be formulated as the following theorem.

Theorem 7. $\mathscr{R}(B,u) = \{v \mid u \Rrightarrow_B v\}$.

Since the unfolding encodes the proof of the transaction step (via the history components), it is possible to use the same scheme for computing the abstract net (whatever philosophy is preferred). However, for doing this efficiently, we must be able to recognize isomorphic processes. For example, note that given a certain computed process, any renaming of the stable tokens in the initial marking (i.e., any permutation of tokens in the same place) yields a different but isomorphic process that is also calculated during the unfolding. To solve this problem, we can either try to avoid having several isomorphic processes in the unfolding by some clever construction, or check at commit-time if the freshly computed transaction is isomorphic to some transaction already computed.

Since a ZS net B can contain cycles that produce an unbounded number of zero tokens, the unfolding can become infinite. So an important question concerns the decidability of $\mathscr{R}(B,u)$. In a private communication to the authors [20], Nadia Busi proved that such set is indeed decidable. Roughly speaking the idea is to simulate the behaviors of B by a PT net with exactly one inhibitor arc,[7] for which the reachability problem has been solved in [42]. The set $\mathscr{R}(B,u)$ is recursively enumerated by the inference system, and if it is infinite we cannot do any improvement. But when $\mathscr{R}(B,u)$ is finite, it would be desirable to find some condition for halting the execution of the algorithm, that otherwise could continue computing transaction segments that cannot be completed. Finding some general condition for halting the unfolding of ZS is an open problem that we leave for future investigations.

---

[7] Nets with inhibitor arcs, also called with negative arcs, have been introduced in [1,2] for modeling systems where the presence of certain resources can inhibit the firing of some transitions. We recall that the reachability problem is undecidable for the class of nets with two or more inhibitor arcs.

5 Zero-safe nets and read arcs

We now show how to extend the zero-safe net paradigm with read arcs, in the style of contextual nets [36]. The idea is to model transitions that can read certain tokens without consuming them, so that multiple readings on the same token can take place concurrently (using ordinary PT nets, the naive way of modeling readings via self-loops[8] is not appropriate because the accesses to read tokens are sequentialized).

5.1 Contextual nets

Definition 32. *A marked* contextual net *(*c-net*) is a tuple* $\mathsf{N} = (S,T,F,\mathsf{C},u_{\text{in}})$*, where* $N_{\mathsf{N}} = (S,T,F,u_{\text{in}})$ *is the underlying* PT *net and* $\mathsf{C}\colon S \times T \to \mathbb{N}$ *is the* context relation.

We denote by $\text{ctx}(t)$ the multiset of places defined by $\text{ctx}(t)(a) = \mathsf{C}(a,t)$ for all $a \in S$ and by $\lfloor u \rfloor$ the underlying set of places of a multiset u (i.e., $\lfloor u \rfloor = \{a \mid u(a) > 0\}$). Informally, the minimum amount of resources that a transition t requires to be enabled is $\text{pre}(t) \oplus \lfloor \text{ctx}(t) \rfloor$: The tokens in $\text{pre}(t)$ are fetched, while those in $\lfloor \text{ctx}(t) \rfloor$ are just read, and other transitions can access them, concurrently with t. The minimum requirement involves $\lfloor \text{ctx}(t) \rfloor$ and not $\text{ctx}(t)$ because the same token can be read more than once. However, t can also read different tokens from the same place, up to the maximum established by $\text{ctx}(t)$. For $t \in T$ with $\text{pre}(t) = u$, $\text{post}(t) = v$ and $\text{ctx}(t) = w$, we write $t\colon u \xrightarrow{w} v$. In the following, we shall overload the symbol $\subseteq$ to denote multiset inclusion.

Definition 33. *Let* u *and* v *be markings of a c-net* N *and let* X *be a finite multiset of transitions of* N*. We say that* X *is enabled at* u *if* $\lfloor \bigoplus_{t\in T} \text{ctx}(t) \rfloor \oplus \bigoplus_{t\in T} X(t) \cdot \text{pre}(t) \subseteq u$*. Moreover, we say that* u *evolves to* v *via* X*, written* $u\,[X\rangle\,v$ *if* X *is enabled at* u *and* $u\,[X\rangle\,v$ *is a step of the underlying* PT *net* N_{N}*.*

Note that if u has enough tokens to satisfy also the ‘context’ of X, then v is obtained from u by removing $\bigoplus_{t\in T} X(t) \cdot \text{pre}(t)$ and then adding $\bigoplus_{t\in T} X(t) \cdot \text{post}(t)$. The step relation can be equivalently defined by the inference rules in Table 13, that carry also information about the context used in the step. The meaning of $u \overset{w}{\Longrightarrow}_{\mathsf{N}} v$ is that from the marking u there is a step that leads to v reading w — note that there must exist two markings u_1 and v_1 such that $u = u_1 \oplus w$ and $v = v_1 \oplus w$. Idle tokens are seen as part of the context of a step. Transitions yield basic steps, where only the minimal context is required. When building larger steps, any part of the contexts of the two substeps can be shared. For example, from $w \overset{w}{\Longrightarrow}_{\mathsf{N}} w$ and from the step $u \oplus \lfloor w \rfloor \overset{\lfloor w \rfloor}{\Longrightarrow}_{\mathsf{N}} v \oplus \lfloor w \rfloor$ associated to $t\colon u \xrightarrow{w} v$, we obtain $u \oplus w \overset{w}{\Longrightarrow}_{\mathsf{N}} v \oplus w$, because $\lfloor w \rfloor \subseteq w$, and therefore $\lfloor w \rfloor$ can be shared.

For sequential composition of steps we have several alternatives: (1) to forget about all the information on context; (2) to arbitrarily forget about part of the context; (3) to define the context of the composed sequence in a canonical way.

The three cases are illustrated in Table 14, where $u \cap v$ denotes the multiset of places such that $(u \cap v)(a) = \min(u(a), v(a))$, for all $a \in S$. In particular, the third set of rules

---

[8] Given a net N, a self-loop consists of two arcs $(a,t),(t,a) \in F_N$ for a place $a \in S_N$ and a transition $t \in T_N$.

identities	generators	parallel composition
$\dfrac{u \in S^{\oplus}}{u \overset{u}{\Longrightarrow}_{\mathsf{N}} u}$	$\dfrac{t{:}\,u \xrightarrow{w} v \in T}{u \oplus \lfloor w \rfloor \overset{\lfloor w \rfloor}{\Longrightarrow}_{\mathsf{N}} v \oplus \lfloor w \rfloor}$	$\dfrac{u_1 \oplus w \overset{w_1 \oplus w}{\Longrightarrow}_{\mathsf{N}} v_1 \oplus w,\ u_2 \oplus w \overset{w_2 \oplus w}{\Longrightarrow}_{\mathsf{N}} v_2 \oplus w}{u_1 \oplus u_2 \oplus w \overset{w_1 \oplus w_2 \oplus w}{\Longrightarrow}_{\mathsf{N}} v_1 \oplus v_2 \oplus w}$

Table 13. The inference rules for $\_ \overset{\_}{\Longrightarrow}_{\mathsf{N}} \_$.

	basic step	sequential composition
(1)	$\dfrac{u \overset{w}{\Longrightarrow}_{\mathsf{N}} v}{u \Longrightarrow^{*}_{\mathsf{N}} v}$	$\dfrac{u \Longrightarrow^{*}_{\mathsf{N}} v,\ v \overset{w}{\Longrightarrow}_{\mathsf{N}} v'}{u \Longrightarrow^{*}_{\mathsf{N}} v'}$
(2)	$\dfrac{u \overset{w_1 \oplus w}{\Longrightarrow}_{\mathsf{N}} v}{u \overset{w}{\Longrightarrow}{}^{*}_{\mathsf{N}} v}$	$\dfrac{u \overset{w}{\Longrightarrow}{}^{*}_{\mathsf{N}} v,\ v \overset{w_1 \oplus w}{\Longrightarrow}_{\mathsf{N}} v'}{u \overset{w}{\Longrightarrow}{}^{*}_{\mathsf{N}} v'}$
(3)	$\dfrac{u \overset{w}{\Longrightarrow}_{\mathsf{N}} v}{u \overset{w}{\Longrightarrow}{}^{*}_{\mathsf{N}} v}$	$\dfrac{u_1 \overset{w_1}{\Longrightarrow}{}^{*}_{\mathsf{N}} v_1,\ v_1 \overset{w_2}{\Longrightarrow}_{\mathsf{N}} v_2,\ w = w_1 \cap w_2}{u_1 \overset{w}{\Longrightarrow}{}^{*}_{\mathsf{N}} v_2}$

Table 14. Three sets of inference rules for $\_ \overset{\_}{\Longrightarrow}{}^{*}_{\mathsf{N}} \_$.

keeps track of the maximal possible context of a sequence. The three definitions lead to the same set of reachable markings. Though (2) and (3) are similar, in principle the former is more appropriate for the *ITph* (because the shared context is not necessarily the maximal one), while the latter can deal well with the *CTph* (cf. the 'maximum sharing hypothesis' of [18,19]).

In a way analogous to PT nets, step sequences for c-nets can be considered up to diamond transformation, originating commutative contextual processes. Instead, for accommodating causal dependencies, (causal) contextual processes are introduced.

Definition 34. *A* deterministic occurrence c-net *is a finite, acyclic (w.r.t. the preorder in which t precedes t' if either* $\mathrm{post}(t) \cap (\mathrm{pre}(t') \cup \mathrm{ctx}(t')) \neq \varnothing$ *or* $\mathrm{ctx}(t) \cap \mathrm{pre}(t') \neq \varnothing$*) c-net* O *such that: (1) for all $t \in T$,* $\mathrm{pre}(t)$ *and* $\mathrm{post}(t)$ *are sets (not multisets) and (2) for all* $t_0 \neq t_1 \in T$, $\mathrm{pre}(t_0) \cap \mathrm{pre}(t_1) = \mathrm{post}(t_0) \cap \mathrm{post}(t_1) = \varnothing$.

The dependencies between events in an occurrence c-net can be of two kinds: 'causal' and 'temporal.' When $\mathrm{post}(t) \cap (\mathrm{pre}(t') \cup \mathrm{ctx}(t')) \neq \varnothing$, then t *causes* t', because t produces a token that is necessary for enabling t'. When $\mathrm{ctx}(t) \cap \mathrm{pre}(t') \neq \varnothing$, then t cannot happen after t', because t' consumes (part of) the context needed by t and since we are describing a deterministic computation where both t and t' must fire, we have that t temporally precedes t'. In [4,3] it is shown that these two notions are precisely characterized by a causal dependency relation $<$ and a relation $\nearrow$ called *asymmetric conflict*. The former is the transitive closure of the relation $\prec$ defined by: (i) if $s \in \mathrm{pre}(t)$, then $s \prec t$; (ii) if $s \in \mathrm{post}(t)$, then $t \prec s$; and (iii) if $\mathrm{post}(t) \cap \mathrm{ctx}(t') \neq \varnothing$, then $t \prec t'$. The asymmetric conflict relation is the union of the causal dependency relation together with the strict asymmetric conflict relation $\rightsquigarrow$ defined by letting $t \rightsquigarrow t'$ if $\mathrm{ctx}(t) \cap \mathrm{pre}(t') \neq \varnothing$ or $t \neq t' \ \wedge\ \mathrm{pre}(t) \cap \mathrm{pre}(t') \neq \varnothing$. The conflict relation # is then induced by $\nearrow$ and $\leq$ (the reflexive closure of $<$) via the rules below:

$$\frac{t_0 \nearrow t_1 \nearrow \ldots \nearrow t_n \nearrow t_0}{\#\{t_0, t_1, \ldots, t_n\}} \qquad \frac{\#(A \cup \{t\}) \quad t \leq t'}{\#(A \cup \{t'\})}$$

where A is a finite set of transitions.

Note that # relates finite sets of transitions and not just pairs of transitions. However, when read arcs are not present, then # is just the closure under set union of the ordinary binary conflict relation of PT nets. It follows that for each deterministic occurrence c-net O the relation $\nearrow_{\mathsf{O}}$ is acyclic and thus the net is conflict free. The last relation we shall introduce regards place concurrency. A set U of places is called concurrent, written $\mathbf{co}(U)$, if: (1) for any $a, a' \in U$, it is not the case that $a < a'$; and (2) $\nearrow$ is acyclic when restricted to the set $\Uparrow(U) = \bigcup_{a \in U} \Uparrow(a)$, where $\Uparrow(x) = \{t \in T \mid t \leq x\}$ is the set of ancestors of x.

Definition 35. *A* contextual process *(*c-process*)* P *for a c-net* N *consists of a deterministic occurrence c-net* O *together with a pair of functions* $f_{\mathsf{P}} : T_{\mathsf{O}} \to T_{\mathsf{N}}$ *and* $g_{\mathsf{P}} : S_{\mathsf{O}} \to S_{\mathsf{N}}$ *that respect sources, targets and contexts of transitions.*

5.2 Zero-safe contextual nets

We now merge the features of ZS nets with that of c-nets, by allowing the combined use of zero places and read arcs in our models.

Definition 36 (ZS c-net). *A* ZS c-net *is a tuple* $\mathsf{B} = (S, T, F, C, Z, u_{\mathrm{in}})$ *such that* $\mathsf{N}_{\mathsf{B}} = (S, T, F, C, u_{\mathrm{in}})$ *is a c-net and* $(S, T, F, Z, u_{\mathrm{in}})$ *is a* ZS *net.*

Note that zero places can be used as context, because $Z \subset S$. In defining the dynamics of ZS c-nets, we can follow two main alternatives. The crucial point is whether to forbid or not that a stable token is read (possibly many times) and then also fetched during the same transaction. While the rules **underlying** and **commit** are identical for both alternatives (see Table 15), the difference between the two is expressed by the rule for sequential composition. For simplicity, on zero tokens we consider the step relation that forgets about the information on contexts, as it is equivalent to the other possible choices from the ‘reachability’ point of view.

In order to allow consumption of previously read stable tokens in the same transaction, we need the complex rule below:

$$(\phi): \frac{(u_1 \oplus w_1 \oplus w, x) \overset{w_1 \oplus w}{\rightrightarrows}_{\mathsf{B}} (v_1 \oplus w_1 \oplus w, y),\ (u_2 \oplus w_1 \oplus w, y) \overset{w}{\rightrightarrows}_{\mathsf{B}} (v_2 \oplus w, y')}{(u_1 \oplus u_2 \oplus w_1 \oplus w, x) \overset{w}{\rightrightarrows}_{\mathsf{B}} (v_1 \oplus v_2 \oplus w, y')}.$$

underlying	**commit**
$\dfrac{u \oplus x \overset{w \oplus z}{\Longrightarrow}_{\mathsf{N_B}} v \oplus y,\ u,v,w \in L^{\oplus},\ x,y,z \in Z^{\oplus}}{(u,x) \overset{w}{\rightrightarrows}_{\mathsf{B}} (v,y)}$	$\dfrac{(u,0) \overset{w}{\rightrightarrows}_{\mathsf{B}} (v,0)}{u \overset{w}{\Rrightarrow}_{\mathsf{B}} v}$

Table 15. The two common inference rules for $\_ \overset{-}{\Rrightarrow}_{\mathsf{B}} \_$.

The idea is that the second step can consume the tokens in w_1 that the first step reads. The context w is instead shared between the two steps.

For *not* allowing consumption of previously read stable tokens in the same transaction, it suffices to introduce the simpler rule

$$(\psi)\ \frac{(u_1 \oplus w, x) \overset{w}{\rightrightarrows}_{\mathsf{B}} (v_1 \oplus w, y),\ (u_2 \oplus w, y) \overset{w}{\rightrightarrows}_{\mathsf{B}} (v_2 \oplus w, y')}{(u_1 \oplus u_2 \oplus w, x) \overset{w}{\rightrightarrows}_{\mathsf{B}} (v_1 \oplus v_2 \oplus w, y')}.$$

The rule sequentializes on zero tokens, while composing in parallel on stable tokens (sharing the whole context w of the two substeps). We recall that w can contain idle tokens, and therefore, given any two steps $(u_1, x) \overset{w_1 \oplus w}{\rightrightarrows}_{\mathsf{B}} (v_1, y)$ and $(u_2, y) \overset{w_2 \oplus w}{\rightrightarrows}_{\mathsf{B}} (v_2, y')$ we can always 'complement' such steps with markings w_2 and w_1 respectively, to obtain $(u_1 \oplus w_2, x) \overset{w_1 \oplus w_2 \oplus w}{\rightrightarrows}_{\mathsf{B}} (v_1 \oplus w_2, y)$ and $(u_2 \oplus w_1, y) \overset{w_1 \oplus w_2 \oplus w}{\rightrightarrows}_{\mathsf{B}} (v_2 \oplus w_1, y')$, so that the rule for horizontal composition can be applied.

To distinguish between the two interpretations, we write either $u \overset{w}{\Rrightarrow}_{\mathsf{B},\phi} v$ or $u \overset{w}{\Rrightarrow}_{\mathsf{B},\psi} v$, depending on which rule among ϕ and ψ is considered. Of course $u \overset{w}{\Rrightarrow}_{\mathsf{B},\psi} v$ implies $u \overset{w}{\Rrightarrow}_{\mathsf{B},\phi} v$ but not vice versa.

Though the operational and abstract semantics can be defined either according to the *CTph* or to the *ITph*, we prefer to follow the latter interpretation only: in this way it is possible to distinguish the places that are used as context during the transaction and the abstract counterpart is still a c-net, whereas the *CTph* might introduce some confusion.

For the *ITph* the transactions correspond to full and connected deterministic c-processes of the underlying c-net such that the origins and destinations are stable and the evolution places are zero places. Thus, the context of a transaction is the set of places that are both minimal and maximal (transactions do not contain isolated places but can contain places that are simply read). The abstract net associated to a ZS c-net B is a c-net that has the stable places of B as places, the transactions of B as transitions with preset, postset and context defined in the obvious way (if we allow to first read and then consume a stable token in the same transaction, then the token is put in the preset of the corresponding abstract transition, not in the context).

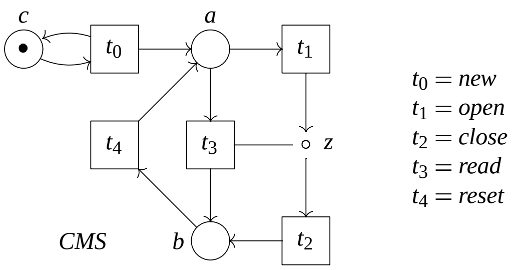

Figure 17. The ZS c-net *CMS* for multicasting.

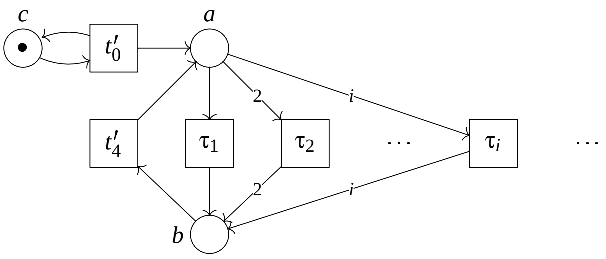

Figure 18. The abstract c-net for the multicasting system *CMS*.

Example 11 (Multicasting revisited). To illustrate the use of read arcs in the zero-safe framework, we show an improved specification of the multicasting system. Let us consider the ZS c-net *CMS* in Figure 17 (as usual, read arcs are depicted as undirected lines). The idea is that copying can be avoided, as all receivers can simply ‘read’ the same copy. Thus, a firing of t_1 opens the session producing the message in the buffer z, then many receivers can concurrently read the information by firing t_3 and then the session is closed by a firing of t_2 that removes the message from the buffer. Of course, many multicasting sessions can take place concurrently. At the abstract level the system is then represented by the c-net in Figure 18. Though this time also the empty transmission τ_1 is possible, the analogy with the abstract net A_{MS} of the multicasting system presented in Section 2 is evident (τ_i represents the one-to-$(i-1)$ transmission).

5.3 A distributed contextual interpreter

We conclude by showing that the interpreter of Section 4 can be modified to deal with contexts by considering the unfolding of ZS nets proposed in [4]. As a matter of notation, we write $t\colon(u,x) \xrightarrow{(v,y)} (u',x')$ for a transition t with preset $u \oplus x$, context $v \oplus y$ and postset $u' \oplus x'$, with $u,v,u' \in (S \setminus Z)^{\oplus}$ and $x,y,x' \in Z^{\oplus}$. The rules for dealing with the case where stable tokens can be first read and then also consumed in the same transaction are illustrated in Table 16. Note that, to store the context accessed, events are encoded as triples rather than as couples.

$$\frac{u(a) = n,\ 1 \leq k \leq n}{\langle a,k,\varnothing\rangle \in S_{\mathcal{U}(\mathsf{B})}}$$

$$\frac{\begin{array}{c} t\colon (u,x) \xrightarrow{(v,y)} (w,\bigoplus_{j\in J} n_j z_j) \in T_{\mathsf{B}},\ \Theta = \{\langle s_i,k_i,H_i\rangle \mid i\in I\} \subseteq S_{\mathcal{U}(\mathsf{B})},\ \Phi = \{\langle s'_i,k'_i,H'_i\rangle \mid i \in I'\} \subseteq S_{\mathcal{U}(\mathsf{B})}, \\ \mathbf{co}(\Theta\cup\Phi),\ \Theta\cap\Phi = \varnothing,\ u\oplus x = \bigoplus_{i\in I} s_i,\ \lfloor v\oplus y\rfloor \subseteq \bigoplus_{i\in I'} s'_i \subseteq v\oplus y \end{array}}{e = \langle t,\Theta,\Phi\rangle \in T_{\mathcal{U}(\mathsf{B})},\ \Upsilon = \{\langle z_j,m,\{e\}\rangle \mid j\in J,\ 1\leq m\leq n_j\} \subseteq S_{\mathcal{U}(\mathsf{B})},\ \mathrm{pre}(e) = \Theta,\ \mathrm{post}(e) = \Upsilon,\ \mathrm{ctx}(e) = \Phi}$$

$$\frac{}{u \in \mathscr{R}_{\phi}(\mathsf{B},u)} \qquad \frac{\Gamma \subseteq T_{\mathcal{U}(\mathsf{B})},\ \nearrow_{\Uparrow(\Gamma)} \text{ is acyclic},\ <_{\Gamma} = \varnothing,\ \mathbf{ZProd}(\Gamma) = \mathbf{ZCons}(\Gamma)}{u \ominus \mathbf{SCons}(\Gamma) \oplus \mathbf{SProd}(\Gamma) \in \mathscr{R}_{\phi}(\mathsf{B},u)}$$

Table 16. The unfolding $\mathcal{U}(\mathsf{B})$ with commit according to (ϕ).

The main differences w.r.t. the interpreter of Section 4 concern the firing rule and the commit rule. In fact, here the execution of a transition has to keep track of the context, differentiating it from the fetched tokens. For this purpose, we have supplied the set Φ. Note that we do not record multiplicities of readings, as they are not important to establish the correctness of a transaction; hence we just check that the context contains enough tokens (more than $\lfloor v\oplus y\rfloor$), but less than the maximum allowed ($v\oplus y$). However, if one is interested in computing the associated processes, then also multiplicities should be considered: They should be assigned to the s'_i so as to exactly match $v\oplus y$. Of course, for the transition to fire, both the context and the preset must be concurrently available and disjoint. When the premises are satisfied, then the event e and the zero places in Υ are inserted in the unfolding. For the commit, we cannot just assume that transitions in Γ are concurrent, as the following example demonstrates.

Example 12. Let us consider the ZS c-net in Figure 19 (reminiscent of *CMS* in Figure 17). There are two admissible transactions: The first is given by a firing of t_1 followed by a firing of t_2 that consumes the token produced in z by t_1. The second consists of a firing of t_1, followed by a firing of t_3 that reads the token in z, and then by a firing of t_2. The unfolding progressively introduces the following tokens and events:

- $s_1 = \langle a,1,\varnothing\rangle$, $s_2 = \langle a,2,\varnothing\rangle$;
- $e_1 = \langle t_1,\{s_1\},\varnothing\rangle$, $z_1 = \langle z,1,\{e_1\}\rangle$;
- $e_2 = \langle t_1,\{s_2\},\varnothing\rangle$, $z_2 = \langle z,1,\{e_2\}\rangle$;
- $e_3 = \langle t_3,\{s_1\},\{z_2\}\rangle$ and $e_4 = \langle t_3,\{s_2\},\{z_1\}\rangle$ (note that, e.g., $\langle t_3,\{s_1\},\{z_1\}\rangle$ cannot be introduced because $s_1 < z_1$);
- $e_5 = \langle t_2,\{z_1\},\varnothing\rangle$ and $e_6 = \langle t_2,\{z_2\},\varnothing\rangle$.

If we require that the commit is given by concurrent transitions only, then the only admissible Γ are $\{e_5\}$, $\{e_6\}$ and $\{e_5,e_6\}$. From these sets we cannot derive any information about the occurrence of e_3 and e_4. However, there are asymmetric conflicts between e_3 and e_6 and between e_4 and e_5, and therefore these events are not completely unrelated. So the question is, e.g., ‘how can we distinguish between the deterministic c-process that involves e_1 and e_5 only from the one that involves also e_4?’ Our answer

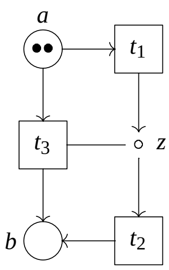

Figure 19.

$$\frac{}{u \in \mathscr{R}_\psi(\mathsf{B},u)} \qquad \frac{\Gamma \subseteq T_{\mathcal{U}(\mathsf{B})},\ \nearrow_{\Uparrow(\Gamma)} \text{ acyclic},\ <_\Gamma = \varnothing,\ \mathbf{ZProd}(\Gamma) = \mathbf{ZCons}(\Gamma),\ \mathbf{Spre}(\Gamma) \cap \mathbf{Sctx}(\Gamma) = \varnothing}{u \ominus \mathbf{SCons}(\Gamma) \oplus \mathbf{SProd}(\Gamma) \in \mathscr{R}_\psi(\mathsf{B},u)}$$

Table 17. The commit of the unfolding according to (ψ).

---

amounts to take Γ as consisting of ‘compatible’ (but not necessarily concurrent) events that are not causally dependent. This is expressed by requiring the asymmetric conflict relation to be acyclic (when restricted to the ancestors of events in Γ) and the intersection between $<$ and $\Gamma \times \Gamma$ to be empty. Under these assumptions, the commit can happen under any of the following Γ: $\Gamma_1 = \{e_5\}$, $\Gamma_2 = \{e_6\}$, $\Gamma_3 = \{e_3, e_6\}$, $\Gamma_4 = \{e_4, e_5\}$, and $\Gamma_5 = \{e_5, e_6\}$.

Since stable contexts are left unchanged by the transaction, then the marking inserted after the commit just computes the tokens consumed and those produced. Note that all conditions can be verified ‘locally’, just by looking at the encoded history of the chosen premises (Θ and Φ for the firing rule and Γ for the commit rule). It might happen that a stable token is first read and also consumed before the end of the transaction; this makes clear the difference between read arcs and self-loops, as in the second case the stable token cannot be reused in the same transaction. It can be verified that the set $\mathscr{R}_\phi(\mathsf{B},u)$ computed in this way is exactly the set of markings that are reachable in one step from u.

Theorem 8. $\mathscr{R}_\phi(\mathsf{B},u) = \{v \mid u \overset{w}{\Rrightarrow}_{\mathsf{B},\phi} v\}$.

The obvious alternative is to forbid stable tokens to be read and consumed in the same transaction, according to the rule (ψ). In this case, the rules for computing $\mathscr{R}_\phi(\mathsf{B},u)$ must be changed as shown in Table 17, where **Spre**(_) and **Sctx**(_) are the pointwise extensions of the functions:

$$\mathbf{Spre}(e) \overset{\text{def}}{=} \textstyle\bigcup_{\langle t,\Theta,\Phi\rangle \preceq e} \{\langle a,k,H\rangle \in \Theta \mid a \in L_{\mathsf{B}}\},$$

$$\mathbf{Sctx}(e) \overset{\text{def}}{=} \textstyle\bigcup_{\langle t,\Theta,\Phi\rangle \preceq e} \{\langle a,k,H\rangle \in \Phi \mid a \in L_{\mathsf{B}}\}.$$

$$\begin{tikzcd}
\mathbf{HCatZPetri} \arrow[r, shift right, "\mathcal{U}_{\mathrm{C}}"'] \arrow[r, phantom, "\perp"] & \mathbf{dZPetri} \arrow[l, shift right, "\mathscr{Z}"'] \arrow[r, shift left, "\mathcal{CG}"] \arrow[r, phantom, "\perp"] & \mathbf{ZSCGraph} \arrow[l, shift left, "\mathcal{U}_{\mathrm{I}}"]
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathbf{ZSN} \arrow[r, shift right, "\mathcal{A}"'] \arrow[r, phantom, "\perp"] & \mathbf{Petri} \arrow[l, hook, shift right] \arrow[r, hook, shift left] \arrow[r, phantom, "\perp"] & \mathbf{ZSC} \arrow[l, shift left, "I"]
\end{tikzcd}$$

Figure 20. Operational and abstract semantics of zero-safe nets.

Note that **Spre**$(\_)$ and **Sctx**$(\_)$ return sets of stable places in the unfolding, and we have **SCons**$(e) = \bigoplus_{\langle a,k,H\rangle \in \mathbf{Spre}(e)} a$.

Theorem 9. $\mathscr{R}_\psi(\mathsf{B},u) = \{v \mid u \overset{w}{\Rrightarrow}_{\mathsf{B},\psi} v\}$.

Since both interpreters are based on the same unfolding net $\mathcal{U}(\mathsf{B})$, it is evident from the two different commit rules that $\mathscr{R}_\psi(\mathsf{B},u) \subseteq \mathscr{R}_\phi(\mathsf{B},u)$, as $\mathscr{R}_\psi(\mathsf{B},u)$ has an additional premise.

Conclusions

We have proposed the framework of zero-safe nets as a basis for modeling and implementing distributed transactions. In fact, ZS nets can provide both the refined view of the systems where actions have finer grain and an abstract view where transactions are seen just as transitions of an ordinary PT net. Working at the level of ZS nets allows one to keep smaller the size of the system description (for example the abstract net can have an infinite number of transitions also when the refined net is finite).

After surveying the operational and abstract semantics of the framework, we have shown how to encode many features of concurrent systems, as e.g. distributed choice, in a compositional manner and how to combine zero places with read arcs. It is worth remarking that the construction of transactions can be defined, in the language of category theory, as an adjunction, i.e., it is a free construction and thus preserves several net composition operations (defined as colimits in the category of ZS nets). The construction of the abstract net defines a coreflection, whose universal properties confirm that it is the optimal such construction. These constructions can be pursued according to either the *CTph* or the *ITph*. The two approaches yield the same step relation but different abstract nets. The categorical semantics (summarized by the four adjunctions in Figure 20) recovers the operational and abstract semantics of ZS nets, introducing an algebraic characterization of the whole framework.

We have also illustrated a distributed interpreter for computing on (ordinary and contextual) ZS nets. We want to remark that the resulting implementation does not violate the locality assumptions, since it is completely analogous to the widely accepted implementation for PT nets. This interpreter satisfactorily answers the questions formulated at the beginning of Section 4: Backtracking is not necessary, correctness is given by Theorem 7, and completeness is ensured by our inference system. We are confident that formal halting criteria can be found for expressive classes of ZS nets.

As future work, we plan to extend the concept of ‘zero place’ to other net flavours, as e.g., coloured and timed nets, nets with inhibitor arcs, and probabilistic nets. In fact we conjecture that our basic mechanism for expressing ‘transition synchronization’ can be helpful also in these richer models for a compositional modeling of systems and for describing execution protocols. Another ongoing line of research concerns the study of hierarchical ZS nets, where one can have different levels of abstraction. Finally, it would be interesting to apply contextual ZS nets to the modeling and study of serializability in transaction systems, in a way which is somehow analogous to the research conducted in [44,21].

References

1. T. Agerwala. A complete model for representing the coordination of asynchronous processes. Hopkins Computer Research Report 32, John Hopkins University, 1974.
2. T. Agerwala and M. Flynn. Comments on capabilities, limitations and “correctness” of Petri nets. *Computer Architecture News*, 4(2):81–86, 1973.
3. P. Baldan. *Modelling concurrent computations: From contextual Petri nets to graph grammars*. PhD thesis, Computer Science Department, University of Pisa, 2000. Published as Technical Report TD-1/00.
4. P. Baldan, A. Corradini, and U. Montanari. An event structure semantics for P/T contextual nets: Asymmetric event structures. In M. Nivat, editor, *Proceedings of FoSSaCS’98, 1st International Conference on Foundations of Software Science and Computation Structures*, volume 1378 of *Lect. Notes in Comput. Sci.*, pages 63–80. Springer Verlag, 1999.
5. E. Best and R. Devillers. Sequential and concurrent behaviour in Petri net theory. *Theoret. Comput. Sci.*, 55:87–136, 1987.
6. E. Best, R. Devillers, and J. Esparza. General refinement and recursion for the Petri Box calculus. In P. Enjalbert, A. Finkel, and K.W. Wagner, editors, *Proceedings STACS’93*, volume 665 of *Lect. Notes in Comput. Sci.*, pages 130–140. Springer Verlag, 1993.
7. E. Best, R. Devillers, and J. Hall. The Box calculus: A new causal algebra with multi-label communication. In G. Rozenberg, editor, *Advances in Petri Nets’92*, volume 609 of *Lect. Notes in Comput. Sci.*, pages 21–69. Springer Verlag, 1992.
8. W. Brauer, R. Gold, and W. Vogler. A survey of behaviour and equivalence preserving refinements of Petri nets. In G. Rozenberg, editor, *Advances in Petri Nets’90*, volume 483 of *Lect. Notes in Comput. Sci.*, pages 1–46. Springer Verlag, 1991.
9. C. Brown and D. Gurr. A categorical linear framework for Petri nets. In *Proceedings of LICS’90, 5th Annual IEEE Symposium on Logic in Computer Science*, pages 208–218. IEEE Computer Society Press, 1990.
10. R. Bruni. *Tile Logic for Synchronized Rewriting of Concurrent Systems*. PhD thesis, Computer Science Department, University of Pisa, 1999. Published as Technical Report TD-1/99.
11. 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 *Lect. Notes in Comput. Sci.*, pages 225–244. Springer Verlag, 1998.
12. R. Bruni, J. Meseguer, U. Montanari, and V. Sassone. Functorial semantics for petri nets under the individual token philosophy. In M. Hofmann, G. Rosolini, and D. Pavlovic, editors, *Proceedings of CTCS’99, 8th Category Theory and Computer Science*, volume 29 of *Elect. Notes in Th. Comput. Sci.* Elsevier Science, 1999.
13. R. Bruni and U. Montanari. Zero-safe nets, or transition synchronization made simple. In C. Palamidessi and J. Parrow, editors, *Proceedings EXPRESS’97, 4th workshop on Expressiveness in Concurrency*, volume 7 of *Elect. Notes in Th. Comput. Sci.* Elsevier Science, 1997.

14. R. Bruni and U. Montanari. Zero-safe nets: The individual token approach. In F. Parisi-Presicce, editor, *WADT'97, 12th workshop on Recent Trends in Algebraic Development Techniques*, volume 1376 of *Lect. Notes in Comput. Sci.*, pages 122–140. Springer Verlag, 1998.
15. R. Bruni and U. Montanari. Zero-safe nets: Composing nets via transition synchronization. In H. Weber, H. Ehrig, and W. Reisig, editors, *Int. Colloquium on Petri Net Technologies for Modelling Communication Based Systems*, pages 43–80. Fraunhofer Gesellschaft ISST, 1999.
16. R. Bruni and U. Montanari. Executing transactions in zero-safe nets. In M. Nielsen and D. Simpson, editors, *Proceedings of ICATPN2000, 21st Int. Conf. on Application and Theory of Petri Nets*, volume 1825 of *Lect. Notes in Comput. Sci.*, pages 83–102. Springer Verlag, 2000.
17. R. Bruni and U. Montanari. Zero-safe nets: Comparing the collective and individual token approaches. *Inform. and Comput.*, 156:46–89, 2000.
18. R. Bruni and V. Sassone. Algebraic models for contextual nets. In U. Montanari, J.D.P. Rolim, and E. Welzl, editors, *Proceedings of ICALP2000, 27th Int. Coll. on Automata, Languages and Programming*, volume 1853 of *Lect. Notes in Comput. Sci.*, pages 175–186. Springer Verlag, 2000.
19. R. Bruni and V. Sassone. Two algebraic process semantics for contextual nets. This Volume.
20. N. Busi. On zero safe nets, April 1999. Private communication.
21. N. De Francesco, U. Montanari, and G. Ristori. Modeling concurrent accesses to shared data via Petri nets. In E.-R. Olderog, editor, *Programming Concepts, Methods and Calculi*, IFIP Transactions A-56, pages 403–422. North Holland, 1994.
22. P. Degano, R. De Nicola, and U. Montanari. A distributed operational semantics for CCS based on condition/event systems. *Acta Inform.*, 26(1-2):59–91, 1988.
23. P. Degano, J. Meseguer, and U. Montanari. Axiomatizing the algebra of net computations and processes. *Acta Inform.*, 33(7):641–667, 1996.
24. H. Ehrig and J. Padberg. Uniform approach to Petri nets. In C. Freska, M. Jantzen, and R. Valk, editors, *Proceedings Foundations of Computer Science: Potential-Theory-Cognition*, volume 1337 of *Lect. Notes in Comput. Sci.*, pages 219–231. Springer Verlag, 1997.
25. R.J. van Glabbeek. Petri nets, configuration structures and higher dimensional automata. In J.C.M. Baeten and S. Mauw, editors, *Proceedings CONCUR'99*, volume 1664 of *Lect. Notes in Comput. Sci.*, pages 21–27. Springer Verlag, 1999.
26. R.J. van Glabbeek and U. Goltz. Refinement of actions and equivalence notions for concurrent systems. Hildesheimer Informatik Bericht 6/98, Institut fuer Informatik, Universitaet Hildesheim, 1998.
27. R.J. van Glabbeek and G.D. Plotkin. Configuration structures. In D. Kozen, editor, *Proceedings of LICS'95, 10th Annual IEEE Symposium on Logics in Computer Science*, pages 199–209. IEEE Computer Society Press, 1995.
28. R.J. van Glabbeek and F. Vaandrager. Petri net models for algebraic theories of concurrency. In J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, editors, *Proceedings PARLE*, volume 259 of *Lect. Notes in Comput. Sci.*, pages 224–242. Springer Verlag, 1987.
29. U. Goltz and W. Reisig. The non-sequential behaviour of Petri nets. *Inform. and Comput.*, 57:125–147, 1983.
30. R. Gorrieri and U. Montanari. On the implementation of concurrent calculi into net calculi: Two case studies. *Theoret. Comput. Sci.*, 141(1-2):195–252, 1995.
31. S. MacLane. *Categories for the Working Mathematician.* Springer Verlag, 1971.
32. J. Meseguer and U. Montanari. Petri nets are monoids. *Inform. and Comput.*, 88(2):105–155, 1990.
33. J. Meseguer, U. Montanari, and V. Sassone. Process versus unfolding semantics for place/transition Petri nets. *Theoret. Comput. Sci.*, 153(1-2):171–210, 1996.

34. J. Meseguer, U. Montanari, and V. Sassone. Representation theorems for Petri nets. In Ch. Freksa, M. Jantzen, and R. Valk, editors, *Foundations of Computer Science: Potential - Theory - Cognition, to Wilfried Brauer on the occasion of his sixtieth birthday*, volume 1337 of *Lect. Notes in Comput. Sci.*, pages 239–249. Springer Verlag, 1997.
35. J. Meseguer, P.C. Ölveczky, and M.-O. Stehr. Rewriting logic as a unifying framework for Petri nets. This Volume.
36. U. Montanari and F. Rossi. Contextual nets. *Acta Inform.*, 32:545–596, 1995.
37. M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains, part I. *Theoret. Comput. Sci.*, 13:85–108, 1981.
38. E.R. Olderog. Operational Petri net semantics for CCSP. In G. Rozenberg, editor, *Advances in Petri Nets'87*, volume 266 of *Lect. Notes in Comput. Sci.*, pages 196–223. Springer Verlag, 1987.
39. J. Padberg. *Abstract Petri nets: Uniform approach and rule-based refinement*. PhD thesis, Technische Universität Berlin, 1996.
40. J. Padberg. Classification of Petri nets using adjoint functors. In *EATCS Bulletin*, volume 66, pages 85–91. European Association for Theoretical Computer Science, 1998.
41. C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für Instrumentelle Mathematik, Bonn, 1962.
42. K. Reinhardt. Reachability in Petri nets with inhibitor arcs. Technical Report WSI-96-30, Wilhelm Schickard Institut für Informatik, Universität Tübingen, 1996.
43. W. Reisig. *Petri Nets: An Introduction*. EATCS Monographs on Theoretical Computer Science. Springer Verlag, 1985.
44. G. Ristori. *Modelling Systems with Shared Resources via Petri Nets*. PhD thesis, Computer Science Department, University of Pisa, 1994. Published as Technical Report TD-5/94.
45. V. Sassone. An axiomatization of the algebra of Petri net concatenable processes. *Theoret. Comput. Sci.*, 170(1-2):277–296, 1996.
46. V. Sassone. An axiomatization of the category of Petri net computations. *Math. Struct. in Comput. Sci.*, 8(2):117–151, 1998.
47. I. Suzuki and T. Murata. A method for stepwise refinement and abstraction of Petri nets. *J. Comput. and System Sci.*, 27:51–76, 1983.
48. R. Valette. Analysis of Petri nets by stepwise refinement. *J. Comput. and System Sci.*, 18:35–46, 1979.
49. W. Vogler. Behaviour preserving refinements of Petri nets. In G. Tinhofer and G. Schmidt, editors, *Proceedings 12th International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 246 of *Lect. Notes in Comput. Sci.*, pages 82–93. Springer Verlag, 1987.
50. G. Winskel. Event structure semantics of CCS and related languages. In M. Nielsen and E. Meineche Schmidt, editors, *Proceedings ICALP'82*, volume 140 of *Lect. Notes in Comput. Sci.*, pages 561–567. Springer Verlag, 1982.
51. G. Winskel. Event structures. In W. Brauer, editor, *Proceedings Advanced Course on Petri Nets*, volume 255 of *Lect. Notes in Comput. Sci.*, pages 325–392. Springer Verlag, 1987.
52. G. Winskel. Petri nets, algebras, morphisms and compositionality. *Inform. and Comput.*, 72:197–238, 1987.