# Zero-Safe Nets: Composing Nets via Transition Synchronization*

Roberto Bruni and Ugo Montanari
Dipartimento di Informatica, Università di Pisa, Italy
{bruni,ugo}@di.unipi.it

**Abstract**

*Zero-safe nets* have been introduced to extend classical Petri nets with a primitive notion of *transition synchronization*. To this aim, besides ordinary places, called *stable*, zero-safe nets are equipped with *zero places*, which cannot contain any token in a stable marking . An evolution between two stable markings is called *transaction* and can be a complex computation that involves zero places, with the restriction that no stable token generated in a transaction can be reused in the same transaction. The abstract counterpart of a generic zero-safe net $B$ consists of an ordinary PT net whose places are the stable places of $B$, and whose transitions are the transactions of $B$. The two nets offer the refined and the abstract model of the same system, where the former can be much smaller than the latter, because of the transition synchronization mechanism. Depending on the chosen approach — *collective* vs *individual token philosophy* — two notions of transaction may be defined, each leading to different operational and abstract models. We survey the two approaches, discussing their main properties and showing several applications.

## Introduction

*Place/transition Petri nets* [26] (also PT nets) are unanimously regarded as one of the most evocative models of concurrency. As a matter of fact,

*Research supported by CNR Integrated Project *Metodi e Strumenti per la Progettazione e la Verifica di Sistemi Eterogenei Connessi mediante Reti di Comunicazione*; by Esprit Working Groups *CONFER2* and *COORDINA*; and by MURST project *Tecniche Formali per Sistemi Software*.

this model offers a basic concurrent framework that has often been used as a semantic foundation for system analysis and for the interpretation of many concurrent languages [30, 14, 24, 11, 16, 3]. However, though net transitions allow for *token* synchronization, the basic net model does not offer any synchronization mechanism among transitions, while this feature is essential to write modular and expressive programs, and to model systems equipped with synchronization primitives. In fact, complex constructions are required in all the above translations for defining the net associated to the synchronous composition of two programs.

*Zero-safe nets* (also ZS nets) extend PT nets along this direction exploiting a simple but very general notion of transition synchronization as a built-in feature. ZS nets are based on the notion of *zero* places. The leading idea is that tokens in a zero place (called *zero tokens*) denote internal states of the system, which are "invisible" at an abstract level. In addition to zero places, a distinguished set of *stable* places is also present. *Stable markings* (consisting only of *stable tokens*, i.e., tokens in stable places) define the "observable" states of the system. All the other markings (i.e., those involving zero tokens) are called *non-stable*. Any evolution of a ZS net must start at some observable marking, then can evolve through hidden states and eventually terminates into a new observable state. It follows that all the zero tokens that are produced during the evolution must be consumed, thus enforcing the firing of some transitions able to consume them. On the contrary, our basic assumption on stable tokens is that if they are produced in some hidden state, then they cannot be used until an observable marking is reached. This means that fresh stable tokens are "frozen" along the evolution, and finally are released by some *commit* that concludes the step.

We can associate to each ZS net a PT net that represents the same system but at a different level of abstraction: The ZS net specifies how every transition of its abstract counterpart is actually refined as a different coordinated computation (called *transaction*). The places of the "abstract" PT net are the stable places of the "refined" ZS net. The transitions of the PT net are the (basic) transactions of the ZS net; hence the PT net offers the synchronized view, which corresponds to the abstraction from the hidden mechanism that controls the token flow in zero places. We remark that in our setting the term "transaction" denotes a certain activity of the system that might be composed by many, possibly concurrent, coordinated sub-activities. Moreover, we require that at the right level of abstraction, where transactions are considered as atomic activities, all the intermediate states (except the initial and the final states) must be unobservable.

A toy example that illustrates the basic concepts of our model is given by the ZS net in Figure 1. First, notice that we extend the standard graphical

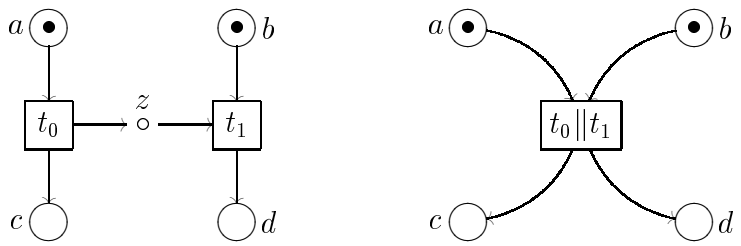


Figure 1: A ZS net and its abstract counterpart.

representation for nets — boxes stand for transitions, circles for places, dots for tokens, and oriented arcs describe the flow relation — by using smaller circles to represent zero places. In the refined model (Figure 1, on the left), the initial marking 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 it can consume both the stable token in $b$ and the zero token in $z$, producing a token in $d$. Then, the reached marking is stable and the transaction is closed. At the abstract level, we are not interested in observing the hidden state consisting of the stable token in $b$, the frozen stable token in $c$ and the zero token in $z$. In fact we just consider stable places plus the atomic activity that takes one token from $a$ and one token from $b$ and produces one token in $c$ and one token in $d$ (Figure 1, on the right).

### PT net semantics

The study of ZS net semantics is of course strictly related to PT net semantics. Most of the semantics for PT nets that have been presented in the literature can be classified as either *process-oriented* semantics or *algebraic* semantics or *unfolding* semantics.

The process-oriented view describes the deterministic computations a PT net can perform, together with some information about the causal dependencies among the executed events. To this aim several different notions of *processes* have been proposed and classified accordingly to the concrete information provided/needed by their underlying structure [2, 15, 12, 21].

The algebraic semantics for PT nets focus instead both on the algebraic structure of the computation space of a single net, and on the global structure of the class of all nets, providing useful operations that can be used to combine nets (e.g., parallel and nondeterministic compositions). Algebraic semantics are often carried out by means of category theory techniques, which

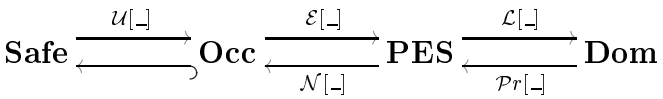


Figure 2: From safe nets to coherent finitary prime algebraic domains.

allow for an abstract formal framework where many notions can be straightforwardly formulated and many constructions find an adequate (universal) characterization [4, 32, 18, 12, 21, 20]. Categories can be used "in the large," when referring to the category of nets and morphisms between them to characterize their global properties (e.g., composition as universal construction), or "in the small," by noticing that the space of computation of each PT net possesses a category structure itself. The latter idea has been exploited, e.g., in [18, 12, 21] to characterize several notion of processes as arrows of suitable (symmetric) monoidal categories, all relying on the main observation that the distributivity of tensor product and arrow composition in monoidal categories expresses the equivalence between concurrent computations.

The unfolding semantics is more oriented to a denotational interpretation for nets, where the interplay between concurrency and nondeterminism plays the central rôle. As for the algebraic semantics, category theory (in the large) is widely employed in ulfonding semantics to justify the adequacy of the selected assignment of denotations. In particular, the notion of *adjunction* between the category of nets and the category of denotations ensures that the assignment given by the *adjoint functor* is the best "possible." This approach has been first developed by Glynn Winskel in [31] for the subclass of *safe* nets — building on previous work [23] — and then extended to PT nets by José Meseguer, Ugo Montanari and Vladimiro Sassone [19].

In particular, in [31] it is shown the existence of the chain of *coreflections* (a particularly nice form of adjunction) in Figure 2 that connects the category **Dom** of *coherent finitary prime algebraic domains* [28] and the category **Safe** of *safe nets*. We recall that **PES** is the category of *prime event structures* (with binary conflict relation), which is equivalent to **Dom**, and **Occ** is the category of (nondeterministic) *occurrence nets*.

In [19], the chain in Figure 2 has been extended from **Occ** to the general category **PTNets** of PT nets via an intermediate category **DecOcc** of *decorated occurrence nets* — a special kind of occurrence nets, where the places belonging to the post-set of the same transition are partitioned into *families*, and partitions are respected by morphisms — and the two adjunctions in Figure 3. Therefore, the adjunction between **Dom** and **PTNets** can be

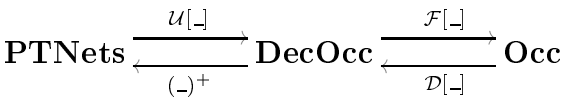


Figure 3: From (marked) PT nets to occurrence nets.

obtained by merging the two chains.

We remark that the unfolding applies only to *marked nets*, i.e., it is always built with respect to an initial marking, whereas both process and algebraic semantics are usually developed for *unmarked nets* (but they can be easily extended to marked nets). We refer to [20] for a more detailed description of the relationship between the three views. We also mention that on top of each of them, various notions of *bisimulation* can be defined (by assuming a labeling of net transitions).

In this paper, apart from the classical "token game," we will mainly rely on the (deterministic) process approach. For the algebraic account of the operational and abstract semantics of ZS nets, here summarized in Section 4, we refer to [10]. Some proposal for the unfolding semantics will be discussed in Section 5.

### Collective and individual token philosophies

Since the concurrent semantics of an operational model considers as equivalent all the computations where the same concurrent events are executed in different orders, it is likely to identify those transactions which are equivalent from a concurrent viewpoint, in such a way that the actual order of execution of concurrent transitions in the refined net is invisible in the abstract net. Doing this, a real dichotomy runs on the distinction between *collective* and *individual token philosophies* (noticed e.g., in [13], but see also [6, 7] where it is explained how the two approaches can influence the *categorical*, *logical* and *behavioral* semantics of ordinary PT nets).

The simplest approach relies on the *collective token philosophy* (*CTph*), where net semantics should not distinguish among different instances (i.e., tokens) of the idealized resources (i.e., places) that rule the basics of net behaviour. Of course, this is true only if any such instance is *operationally* equivalent to all the others. According to this school of thought, 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. As a major drawback, this approach disregards that operationally equivalent resources may have

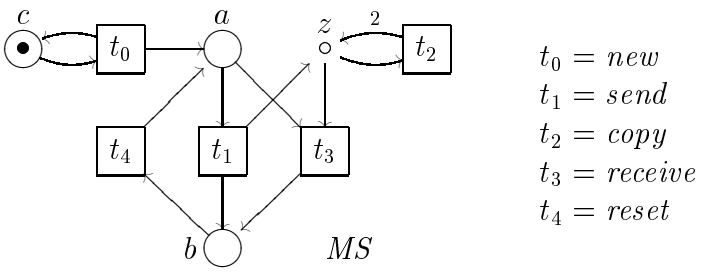


Figure 4: The ZS net *MS* representing a multicasting system.

different origins and histories, and may, therefore, carry different *causality* information (e.g., selecting one instance of a resource rather than another, may be as different as being or not causally dependent on some previous event). Therefore, causal dependencies on zero tokens are lost and the class of computations captured by abstract nets may be too abstract for many applications.

An alternative approach is the *individual token philosophy* (*ITph*). According to the *ITph*, causal dependencies are a central aspect in the dynamic evolution of a net. As a consequence, only the transactions which refer to isomorphic Goltz-Reisig processes [15] are identified, and we call *connected transactions* the induced equivalence classes. In this case, the actual order of execution of concurrent transitions in the refined net is invisible in the abstract net, but all the causal dependencies are preserved.

### The multicasting system example

To better illustrate the concepts so far discussed, we describe a simple but enlightening example, taken from [10]. We start by considering the ZS net *MS* pictured in Figure 4. The net *MS* 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, and thus, several one-to-many communications can take place concurrently.

Each token in place $a$ represents a different *active* agent (i.e., an agent which is ready to communicate), while the tokens in place $b$ are *inactive* agents. The zero place $z$ models a buffer with tokens as messages.

The transition *new* permits to create an unlimited number of 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 suspended

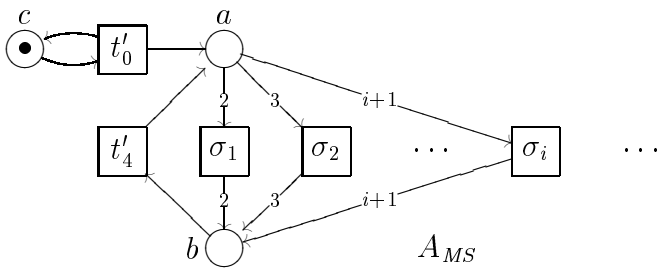


Figure 5: The abstract net for the multicasting system under the *CTph*.

until the end of the current transaction. Each time the transition *copy* fires, a new copy of the same 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 suspends the agent. At the end of a session, all the suspended agents are moved into place $b$. The transition *reset* activates an inactive agent.

We call *copy policy* any strategy for making copies of the messages in the buffer $z$. For instance, the *sequential one-to-n copy policy* acts as follows: Once the message is produced, a firing of *copy* creates two copies of the message in the buffer, one of them is retrieved by the firing of *receive*, the remaining message is duplicated again by a second firing of *copy*, one of the two messages is retrieved by a second agent via a second firing of *receive*, and so on, until the $n$-th agent consumes the last copy of the message, i.e., the firing sequence starts with *send*, then repeats $n-1$ times *copy* followed by *receive*, and is concluded by the $n$-th *receive*. Another copy policy, is the *parallel one-to-n copy policy*, which begins with *send*, then creates all the needed copies of the message by firing *copy* with the maximal parallelism allowed (e.g., for $n = 8$, first *copy* fires, then two instances of *copy* fire concurrently, producing four copies of the message, then four instances of *copy* fire concurrently), and at last, the $n$ agents retrieve the $n$ copies concurrently (by firing $n$ instances of *receive*).

According to the *CTph*, transactions are distinguished only if they differ for the number of involved agents, whereas if the *ITph* is adopted, then different copy policies involving the same number of agents can be distinguished. In Figure 5 we see the infinite abstract PT net $A_{MS}$ for the refined ZS net $MS$, according to the *CTph* (see Definition 2.6). As it will be explained later, the *abstract* net $A_{MS}$ comes equipped with a *refinement morphism* $\epsilon^C_{MS}$ 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 equivalence classes under the $\mathsf{CTph}$ of) the transactions of $MS$. The transition $\sigma_n$ of $A_{MS}$ represents a one-to-$n$ transmission, i.e., it represents any transaction which corresponds to a one-to-$n$ transmission. For example, provided that we have reached a marking with at least five active agents, the sequential copy one-to-four transmission $sc_4$ defined as

$$sc_4 \quad = \quad \textit{send-copy-receive-copy-receive-copy-receive-receive}$$

and the parallel copy one-to-four transmission $bc_4$ informally written as

$$bc_4 \quad = \quad \textit{send-copy-}\{\textit{copy,copy}\}\textit{-}\{\textit{receive,receive,receive,receive}\}$$

are in the same equivalence class. In fact, it is possible to repeatedly swap concurrently enabled consecutive transitions in $sc_4$ to derive the transaction

$$\textit{send-copy-copy-copy-receive-receive-receive-receive},$$

as well as any other linearization of $bc_4$.

On the contrary, if we take the $\mathsf{ITph}$, then different copy policies may be distinguished. The infinite *causal abstract* PT net $I_{MS}$ corresponding to the refined ZS net $MS$ under the $\mathsf{ITph}$ (see Definition 2.12) is displayed in Figure 6. Also the net $I_{MS}$ comes equipped with a *causal refinement morphism* $\epsilon^I_{MS}$ 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 equivalence classes under the $\mathsf{ITph}$ of) the transactions of $MS$. We assume that the generic transition $\sigma^k_n$ 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). As an example transitions $\sigma^1_4$ and $\sigma^2_4$ denote the equivalence classes of $sc_4$ and $bc_4$ (the two are no longer equivalent).

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 is finite while the abstract net is infinite. This is the case for our example, where a message can be delivered to an unlimited number of receivers.

### On process algebra encodings

The synchronization mechanism of ZS net can favor a uniform approach to concurrent languages translation. For instance, in the case of process algebras

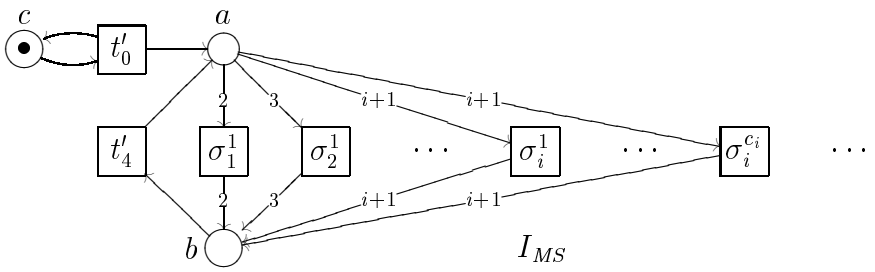


Figure 6: The *causal* abstract net for the multicasting system under the *ITph*.

as CCS [22], the parallel composition of two nets modeling communicating processes involves the combinatorial analysis of all the admissible synchronizations, whereas if zero places were used to model communication channels, then the parallel composition of two nets would just merge the common channels. We will present in Section 3 a compositional representation of a simple process algebra equipped with action prefix, parallel composition and restriction operators. In particular, each agent will be mapped in a ZS net (with *interface*) whose abstract net is able to (bi)simulate the behaviour of the agent. Moreover, the hiding of local names to the external world is handled very well, because agents that are equivalent up to $\alpha$-conversion yield isomorphic nets.

Our results are also supported by a present research by Philippe Darondeau and David Harlet concerning a ZS net model for (the boolean component of) the synchronous language SIGNAL [1]. This fact could be almost surprising, since nets are well suited for modeling asynchronous systems rather than synchronous. However the basic mechanism of ZS nets seems powerful enough to capture the expressiveness of (boolean) SIGNAL primitives.

### Structure of the paper

After recalling in Section 1 some basic definitions of net theory, we present ZS nets and their operational and abstract semantics throughout Section 2, where the simpler *CTph* approach is analyzed first. The operational semantics for ZS nets in the *CTph* is defined in Section 2.1.1, while Section 2.1.2 concerns the related abstract semantics. In this context, the basic evolutions of ZS nets are equivalence classes of ordinary firing sequences induced by diamond transformation, and are called *abstract stable transactions*. To illustrate the *ITph* versions, in Section 2.2.1 we introduce the notion of *causal firing sequence*, which allows for a concise representation of concatenable processes, and has a suggestive implementation on a machine whose states

are collections of token stacks. Then, we define the evolution of a ZS net in terms of the equivalence classes of causal firing sequences induced by isomorphic underlying processes. As for the *CTph* based approach, in Section 2.2.2, ordinary PT nets are defined as the abstract counterparts of ZS nets.

In Section 3, we show a simple application of ZS nets in the modeling of CCS-like process algebras. In Section 4 we summarize the universal properties of abstract nets and give an account of how the operational and abstract semantics can be recovered using the algebraic/categorical approach.

In Section 5 we sketch some preliminary ideas concerning the unfolding of ZS nets, which could suggest a deep insight in the implementation of distributed transactions.

**Related work.** ZS nets have been introduced in [8], where their semantics is discussed under the *CTph*. The *ITph* approach has been developed in [9]. A comparison between the two approaches and several applications have been discussed in [10] and in the PhD Thesis of first author [5]. A discussion on related approaches to net refinement/abstraction can be found in [10, 5].

# 1 PT nets

DEFINITION 1.1 (NET)
*A* net $N$ *is a triple* $N = (S_N, T_N, F_N)$*, where* $S_N$ *is the (nonempty) set of* places $a, a', \ldots$*,* $T_N$ *is the set of* transitions $t, t', \ldots$ *(with* $S_N \cap T_N = \emptyset$*), 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* $F_N$ *are called* arcs *and we write* $xF_N y$ *for* $(x, y) \in F_N$*.*

We will denote $S_N \cup T_N$ by $N$ whenever no confusion arises, and subscripts will be omitted if they are obvious from the context. For $x \in N$, the set ${}^\bullet x = \{y \in N \mid yFx\}$ is called the *pre-set* of $x$, and the set $x^\bullet = \{y \in N \mid xFy\}$ is called the *post-set* of $x$. We only consider nets such that any transition $t$ has non-empty pre-set. Moreover, let ${}^\circ N = \{x \in N \mid {}^\bullet x = \emptyset\}$ and $N^\circ = \{x \in N \mid x^\bullet = \emptyset\}$ denote the sets of *initial* and *final elements* of $N$ respectively. A place $a$ is said to be *isolated* if ${}^\bullet a \cup a^\bullet = \emptyset$.

DEFINITION 1.2 (PT NET)
*A* marked place/transition Petri net *(also* PT *net) is a 5-tuple*

$$N = (S, T, F, W, u_{\text{in}})$$

*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* $u_{\text{in}} : S \to \mathbb{N}$ *is the* initial marking *of* $N$*.*

It is convenient to interpret the relation $F$ as a function

$$F : ((S \times T) \cup (T \times S)) \to \{0, 1\},$$

assuming $xFy \iff F(x, y) \neq 0$. In fact, for nets with weighted arrows we may safely replace $\{0, 1\}$ by $\mathbb{N}$ and abandon $W$. Thus, the relation $F$ 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 $u = \{n_1 a_1, ..., n_k a_k\}$ 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 $n_i a_i$ is omitted), or as a formal sum $u = \bigoplus_{a_i \in S} n_i a_i$ denoting an element of the free commutative monoid $S^{\oplus}$ on the set of places $S$ (the order of summands is immaterial and the monoidal composition is defined by taking $(\bigoplus_i n_i a_i) \oplus (\bigoplus_i m_i a_i) = (\bigoplus_i (n_i + m_i) a_i)$ and 0 as the neutral element).

For any transition $t \in T$, let $pre(t)$ and $post(t)$ be the multisets over $S$ such that $pre(t)(a) = F(a, t)$ and $post(t)(a) = F(t, a)$, for all $a \in S$.

Definition 1.3 (Enabling, firing and firing sequence)
*Given a* PT *net $N$, let $u$ and $u'$ be markings of $N$. Then a transition $t \in T_N$ is* enabled *at $u$ if $pre(t)(a) \leq u(a)$, for all $a \in S_N$.*

*Moreover, we say that $u$ evolves to $u'$ under the* firing *of $t$, written $u[t\rangle u'$, if and only if $t$ is enabled at $u$ and $u'(a) = u(a) - pre(t)(a) + post(t)(a)$, for all $a \in S$.*

*A* firing sequence *from $u_0$ to $u_n$ is a sequence of markings and firings such that $u_0[t_1\rangle u_1 ... u_{n-1}[t_n\rangle u_n$.*

Given a marking $u$ of $N$ the set $[u\rangle$ of its *reachable markings* is the smallest set of markings such that $u \in [u\rangle$, and moreover, for any $u' \in [u\rangle$ such that $u'[t\rangle u''$ for some transition $t$, then $u'' \in [u\rangle$ (i.e., it is closed w.r.t. firings).

Definition 1.4 (Reachable markings of $N$)
*The* reachable markings *of the net $N = (S, T, F, u_{\text{in}})$ are the elements of the set $[u_{\text{in}}\rangle$, i.e., the markings that are reachable from the initial marking of $N$.*

Besides firings and firing sequences, *steps* and *step sequences* are usually introduced. A step allows for the simultaneous execution of several concurrently enabled transitions, i.e., the execution of a multiset of transitions.

A multiset $X : T \to \mathbb{N}$ is enabled at $u$ if for all $a \in S_N$

$$\sum_{t \in T} X(t) \cdot pre(t)(a) \leq u(a).$$

We say that $u$ evolves to $u'$ under the *step* $X$, written $u[X\rangle u'$, if and only if $X$ is enabled at $u$ and for all $a \in S$

$$u'(a) = u(a) + \sum_{t \in T} X(t) \cdot (post(t)(a) - pre(t)(a)).$$

The notion of *process* has been introduced by Ursula Goltz and Wolfgang Reisig [15] to deal with a more informative semantic account of the causal relationships between firings and instances of resources.

Definition 1.5 (Occurrence net and process)
*A net $K$ is a (*deterministic*)* occurrence net *if for all $a \in S_K$, $|{}^\bullet a| \leq 1 \wedge |a^\bullet| \leq 1$ and $F_K^*$ is acyclic,*[1] *(i.e., for all $x, y \in K$, $xF_K^*y \wedge yF_K^*x \Longrightarrow x = y$).*

*A (Goltz-Reisig)* process *for a* PT *net $N$ is a mapping $P : K \to N$, 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 $t \in T_K$, $a \in S_N$,*

$$F_N(a, P(t)) = \left|P^{-1}(a) \cap {}^\bullet t\right| \ \wedge \ F_N(P(t), a) = \left|P^{-1}(a) \cap t^\bullet\right|.$$

Two processes $P$ and $P'$ of $N$ are *isomorphic* and thus identified if there exists an 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.

*Concatenable processes* [12] are obtained from processes by imposing a total ordering on the origins that are instances of the same place and, similarly, on the destinations.

Definition 1.6 (Concatenable process)
*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*

$$\beta_a : l^{-1}(a) \to \{1, \ldots, \left|l^{-1}(a)\right|\}.$$

*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 for $N$, and ${}^\circ\ell$, $\ell^\circ$ are label-indexed ordering functions for the labeling 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

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

initial (maximal) places of $K_P$; then the composition of $C = (P, \ell, \ell^\circ)$ and $C' = (P', \ell', \ell'^\circ)$ 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 so to match these places one-to-one. Concatenable processes admit also a monoidal *parallel* composition $_ \otimes _$, which can be represented by putting two processes side by side. We refer the interested reader to [12] for the formal definitions of $C; C'$ and $C \otimes C'$, which make the concatenable processes of a PT net $N$ the arrows of a symmetric monoidal category $\mathcal{P}(N)$.

We conclude this introductory section by recalling the notion of *safety*.

DEFINITION 1.7 ($n$-SAFE NETS)
*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. A net is* safe *if a bound can be given for the number of tokens in each place (for all reachable markings), i.e., if there exists $n \in \mathbb{N}$ such that for all $u \in [u_{\text{in}}\rangle$, and for all $a \in S$, we have $u(a) \leq n$.*

# 2 Zero-safe nets

We augment PT nets with special places called zero places. Their rôle is to coordinate 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. Notice that, whereas in the standard terminology a $n$-safe net is a net whose places are all $n$-safe, in the case of zero-safe nets only a subset of places (the zero places) are required to satisfy a 0-safe condition w.r.t. observable markings.

DEFINITION 2.1 (ZS NET)
*A* zero-safe net *(also* ZS *net) is a 5-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 and the set $Z_B \subseteq S_B$ is the set of* zero places. *The places in $S_B \setminus Z_B$ are called* stable places. *A* stable marking *is a multiset of stable places, and the initial marking of $B$ must be stable.*

Stable markings describe *observable* states of the system, whereas the presence of one or more zero tokens in a given marking marks it as unobservable. We call *stable tokens* and *zero tokens* the tokens that respectively belong to stable places and to zero places.

## 2.1 Collective token approach

### 2.1.1 Operational semantics

A *stable step* of a ZS net $B$ may involve the execution of several transitions of the underlying PT net $N_B$ (it is actually a firing sequence of $N_B$). At the beginning, the state must contain enough stable tokens to enable the stable pre-sets of all these transitions at the same time. 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. However, no zero token can be left at the end of the computation (nor can belong to the starting configuration).

A stable step whose intermediate markings are all not stable and which consumes all the available stable tokens is called *stable transaction*. *Stable step sequences* are sequences of stable steps. In a certain sense, each stable step can be thought of as a collection of stable transactions plus a collection of idle resources; hence from stable transactions we can derive all the activities of the system.

DEFINITION 2.2 (STABLE STEP, TRANSACTION AND STEP SEQUENCE)
*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:*

- $\forall a \in S_B \setminus Z_B,\ \sum_{i=1}^{n} pre(t_i)(a) \leq u_0(a)$ *(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 markings $u_0$ and $u_n$ respectively.*

*A stable step $s$ is a* stable transaction *of $B$ if in addition:*

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

*A* stable step sequence *is a sequence $u_0\{\![s_1\rangle u_1 \dots u_{n-1}\{\![s_n\rangle u_n$. We also say that $u_n$ is* reachable from $u_0$ *and we write $u_n \in \{\![u_0\rangle$. Sometimes we will denote the set $\{\![u_B\rangle$ of* reachable (stable) markings *of $B$ by $\{\![B\rangle$.*

We recall the main assumption that stable tokens produced during the transaction become operative in the system only at the end of the transaction (i.e., after the firing of the *commit* transition $t_n$).

EXAMPLE 2.1 *Consider the* ZS *net MS of Figure 4.*

*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 since the concurrent enabling condition is not satisfied, in fact* $pre(t_1)(b) + pre(t_4)(b) + pre(t_3)(b) = 1 \not\leq 0$*).*

*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 since the perfect enabling condition is not satisfied (*$pre(t_1)(a) + pre(t_2)(a) + 2pre(t_3)(a) = 3 < 4$*).*

*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 since the atomicity constraint is not satisfied (the inner marking* $\{3b\}$ *is stable).*

*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 (as opposed to the first sequence of this example that starts from the marking* $2a$ *mimicking* $s''$*).*

The concurrent semantics of an operational model is usually defined by considering as equivalent all the computations where the same concurrent events are executed in different orders. In the case of PT nets, the simplest approach relies on the *CTph*, which identifies all the firing sequences obtained by repeatedly permuting pairs of consecutive firings which are concurrently (i.e., independently) enabled [17]. Thus, Definition 2.3 allows for a more satisfactory notion of stable step (transaction) in a concurrent setting.

DEFINITION 2.3 (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 firing sequence of* $N$*, where* $t_i$ *and* $t_{i+1}$ *are concurrently enabled by* $u_{i-1}$*, i.e.,* $pre(t_i)(a) + pre(t_{i+1})(a) \leq u_{i-1}(a)$ *for any place* $a$*. Let* $s'$ *be the firing sequence obtained by permuting the firing ordering of* $t_i$ *and* $t_{i+1}$*, i.e.,*

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

*The sequence* $s'$ *is a* diamond transformation *of* $s$*.*

Diamond transformations define a symmetric relation whose reflexive and transitive closure gives the right equivalence w.r.t. the *CTph* interpretation. Note that equivalent sequences have the same origin ($u_0$) and the same destination ($u_n$).

DEFINITION 2.4 (ABSTRACT SEQUENCE)
*Equivalence classes of sequences (w.r.t. diamond transformations) are called* abstract sequences *and are ranged over by* $\sigma$*. The abstract sequence of* $s$ *is written* $[\![s]\!]$*. We also write* $pre([\![s]\!]) = O(s)$ *and* $post([\![s]\!]) = D(s)$ *to denote respectively the origins and the destinations of* $[\![s]\!]$*.*

Definition 2.5 (Abstract stable step and 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* $\Upsilon_B$ *the set of all abstract stable transactions of* $B$.

The equivalence induced by diamond transformations preserves stable steps (because the diamond transformation preserves the properties of concurrent enabling and of stable fairness required by Definition 2.2) but not 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. Thus, it is not enough to require $s$ to be a stable transaction to make sure that $[\![s]\!]$ is an abstract stable transaction, and we need a stronger constraint, namely that all the sequences in the equivalence class are stable transactions.

Example 2.2 *As a counterexample showing that stable transactions are not preserved, consider the stable steps* $s'$ *and* $s''$ *defined in Example 2.1. It is easy to verify that* $[\![s']\!] = [\![s'']\!]$, *since* $s''$ *is obtained from* $s'$ *by a diamond transformation. However* $s''$ *is a stable transaction whereas* $s'$ *is not. Thus* $[\![s'']\!]$ *is not an abstract stable transaction.*

*Conversely, the firing sequence*

$$s = \{4a\}[t_1\rangle\{3a,b,z\}[t_2\rangle\{3a,b,2z\}[t_2\rangle\{3a,b,3z\}[t_3\rangle\{2a,2b,2z\}[t_3\rangle\{a,3b,z\}[t_3\rangle\{4b\}$$

*defines an abstract stable transaction* $[\![s]\!]$. *In fact the stable transaction*

$$\overline{s} = \{4a\}[t_1\rangle\{3a,b,z\}[t_2\rangle\{3a,b,2z\}[t_3\rangle\{2a,2b,z\}[t_2\rangle\{2a,2b,2z\}[t_3\rangle\{a,3b,z\}[t_3\rangle\{4b\}$$

*is the unique diamond transformation of* $s$ *(and vice versa).*

### 2.1.2 Abstract semantics

According to the *CTph*, since the basic execution steps of a system modeled via zs nets consist of abstract stable transactions, it is natural to define a high-level description of such a model as a net whose transitions are abstract stable transactions.

Definition 2.6 (Abstract net)
*The* abstract net *of a* zs *net* $B = (S_B, T_B, F_B, u_B, Z_B)$ *is the net* $A_B = (S_B \setminus Z_B, \Upsilon_B, F, u_B)$, *with* $F(a,\sigma) = pre(\sigma)(a)$ *and* $F(\sigma,a) = post(\sigma)(a)$

*for all $a \in S_B \setminus Z_B$ and $\sigma \in \Upsilon_B$, is the* abstract net *of $B$ (we recall that $pre(\sigma)$ and $post(\sigma)$ yield the first and last marking of any stable transaction in the equivalence class of $\sigma$, and that $\Upsilon_B$ is the set of all the abstract stable transactions of $B$).*

EXAMPLE 2.3 *Let MS be the* ZS *net of our running example and let $\{c\}$ be its initial marking. Consider the following firing sequences of the underlying net $N_{MS}$ of MS:*

$$\begin{aligned} s_{new} &= \{c\}[t_0\rangle\{a,c\}, \\ s_{res} &= \{b\}[t_4\rangle\{a\}, \\ s_1 &= \{2a\}[t_1\rangle\{a,b,z\}[t_3\rangle\{2b\}, \\ &\cdots \\ s_i &= \{(i+1)a\}[t_1\rangle\{ia,b,z\}[t_2\rangle\cdots[t_2\rangle\{ia,b,iz\}[t_3\rangle\cdots[t_3\rangle\{(i+1)b\}, \\ &\cdots \end{aligned}$$

*where $s_i$ has $i-1$ firings of $t_2$ and $i$ firings of $t_3$.*

*Thus $\Upsilon_{MS} = \{t'_0, t'_4, \sigma_1, \ldots, \sigma_i, \ldots\}$ with $t'_0 = [\![s_{new}]\!]$, $t'_4 = [\![s_{res}]\!]$ and $\sigma_i = [\![s_i]\!]$, for $i \geq 1$. The (infinite) abstract net $A_{MS}$ of MS is (partially!) depicted in Figure 5. 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 communication involving a different number of receivers.*

## 2.2 Individual token approach

In this section, the basic activities of ZS nets are defined accordingly to the *ITph*. As we will see, this choice has a great impact on the resulting notion of transaction. To better understand the difference between the two philosophies, we propose the following example.

EXAMPLE 2.4 *Let MS be the net in Figure 4, and suppose that the current marking is $\{a,b\}$. If $t_4$ fires then a new token is produced into place $a$. A firing of $t_1$ consumes a token from place $a$.*

- *In the individual token approach, it makes a difference if $t_1$ gets the token produced by $t_4$ or the one already present in $a$ (in the former case the firing of $t_1$ causally depends on that of $t_4$ while in the latter case the firings of $t_1$ and of $t_4$ are concurrent activities).*

- *In the collective token approach the two firings are always concurrent, since the initial marking enables both $t_1$ and $t_4$, i.e., the execution of $t_4$ does not modify the enabling condition of $t_1$. Thus $t_1$ and $t_4$ may fire in any order always originating equivalent computations.*

### 2.2.1 Operational semantics

In the *ITph*, a marking can be thought of as an indexed (over the places of the net) collection of ordered sequences of tokens. Moreover, the firing of a transition specifies which tokens (of each ordered sequence) are consumed and also the correspondence between each token in the reached marking with either some produced token or an idle token of the original marking. Using multisets instead of ordered sequences would make it impossible to recognize which token was produced by which firing, as it happens for the *CTph*.

**The stack based approach.** The approach we propose is very similar to the one adopted in [27]: We choose a canonical interpretation of the tokens that have to be consumed and produced in a firing and we introduce *permutation firings* with the task of rearranging the orderings of the indexed sequences of tokens. A marking is represented as a collection of stacks, one for each place. Therefore, the extraction and the insertion of tokens follow the LIFO policy. However, permutation firings are introduced to modify the token positions in the stacks. Informally, permutation firings permit to choose which tokens to consume next. We will denote the token stack associated to a place $a$ by the term $a$-stack.

Definition 2.7 (Causal firing)
*Let $N$ be a* PT *net, and $s = u[t\rangle u'$ be a firing of $N$ for some marking $u$ and transition $t$. We interpret the firing $s$ as a* causal firing *by assuming that $s$ consumes the first $pre(t)(a)$ tokens of the $a$-stack of $u$ and produces the first $post(t)(a)$ tokens of the $a$-stack of $u'$, for each place $a$.*

Definition 2.8 (Permutation firing)
*Let $N$ be a* PT *net. Given a marking $u = \{n_a a\}_{a \in S_N}$ of $N$, a* symmetry *$p$ on $u$ is a vector of* permutations *$p = \langle \pi_a \rangle_{a \in S_N}$ with $\pi_a \in \Pi(n_a)$, for all $a \in S_N$, i.e., each $\pi_a$ is a permutation of $n_a$ elements. We denote by $\Pi(u)$ the set of all symmetries on $u$. Each symmetry $p$ on $u$ induces a* permutation firing *$s = u[p\rangle u$ on the net.*

Definition 2.9 (Causal firing sequence)
*Given a* PT *net $N$, a* causal firing sequence *is a finite sequence $\omega = s_1 \cdots s_n$ of causal and permutation firings such that $s_i = u_{i-1}[x_i\rangle u_i$ with $x_i \in T_N \cup$*

$\Pi(u_{i-1})$ *for* $i = 1, \ldots, n$. *We say that* $\omega$ starts at $u_0$ *(written* $O(\omega) = u_0$*) and* ends in $u_n$ *(written* $D(\omega) = u_n$*).*

EXAMPLE 2.5 *Let* $N_{MS}$ *be the underlying net of the* ZS *net MS defined in Figure 4 (i.e., in* $N_{MS}$ *we do not distinguish between stable and zero places). A causal firing sequence for* $N_{MS}$ *is*

$$\omega = \{b, c\}[t_0\rangle\{a, b, c\}[t_4\rangle\{2a, c\}[t_1\rangle\{a, b, c, z\}[t_3\rangle\{2b, c\}.$$

*At the beginning of* $\omega$ *both the a-stack and the z-stack are empty whilst the b-stack and the c-stack contain one token each. After the firing of* $t_0$*, the token in the c-stack is replaced by a new one and a token is also inserted in the a-stack. The firing of* $t_4$ *consumes the token in the b-stack and puts a new token on top of the a-stack. The firing of* $t_1$ *consumes the token on top of the a-stack (the one produced by the firing of* $t_4$*) and produces a token in the b-stack and a token in the z-stack. The firing of* $t_3$ *consumes the token in the z-stack and also the token produced by the firing of* $t_0$ *in the a-stack; then, it inserts a token on top of the b-stack.*

*To represent the sequence in which* $t_1$ *depends on* $t_0$ *and* $t_3$ *depends on* $t_4$*, we have two possibilities. The first one is to execute* $t_0$ *after* $t_4$ *(they are concurrently enabled), thus obtaining the sequence*

$$\omega' = \{b, c\}[t_4\rangle\{a, c\}[t_0\rangle\{2a, c\}[t_1\rangle\{a, b, c, z\}[t_3\rangle\{2b, c\}.$$

*The second possibility is to reorganize the a-stack just before executing* $t_1$*. This can be done via the symmetry* $p = \langle \pi_a, \pi_b, \pi_c, \pi_z \rangle$*, where* $\pi_b$ *and* $\pi_z$ *are the empty permutations in* $\Pi(0)$*,* $\pi_a = \{1 \to 2, 2 \to 1\} \in \Pi(2)$*, and* $\pi_c$ *is the unique identity permutation in* $\Pi(1)$*. We obtain the sequence*

$$\omega'' = \{b, c\}[t_0\rangle\{a, b, c\}[t_4\rangle\{2a, c\}[p\rangle\{2a, c\}[t_1\rangle\{a, b, c, z\}[t_3\rangle\{2b, c\}.$$

Causal firing sequences precisely characterize the token flow. This is due to the implicit orders which are imposed on the markings and is strictly related to a *process* view of computations, and in particular to concatenable processes (see Definition 1.6).

It may be easily noticed that each causal firing sequence uniquely determines a concatenable process. Informally the construction associates an elementary (concatenable) process to each causal and permutation firing.

**From causal firings to processes.** Let $N$ be a PT net and $s = u[t\rangle u'$ be a causal firing, with $u = \{n_a a\}_{a \in S_N}$, $pre(t) = \{h_a a\}_{a \in S_N}$, $post(t) = \{k_a a\}_{a \in S_N}$. The concatenable process $pr(s) = (P : K \to N, \mathscr{C}, \ell^\circ)$ is defined as follows:

- $T_K = \{\tilde{t}\},\ P(\tilde{t}) = t$;
- $S_K = \{\tilde{a}_i \mid a \in S_N,\ 1 \leq i \leq n_a + k_a\}$, with $P(\tilde{a}_i) = a$;
- ${}^\bullet\tilde{t} = \{\tilde{a}_i \mid a \in S_N,\ 1 \leq i \leq h_a\}, \tilde{t}^\bullet = \{\tilde{a}_i \mid a \in S_N,\ h_a{+}1 \leq i \leq h_a{+}k_a\}$. Therefore
  - $O(K) = \{\tilde{a}_i \in S_K \mid i \leq h_a \vee i \geq h_a + k_a + 1\}$ and
  - $D(K) = \{\tilde{a}_i \in S_K \mid i \geq h_a + 1\}$;
- $\forall \tilde{a}_i \in O(K),\ \mathscr{C}_a(\tilde{a}_i) = \begin{cases} i & \text{if } 1 \leq i \leq h_a \\ i - k_a & \text{if } h_a + 1 + k_a \leq i \leq n_a + k_a; \end{cases}$
- $\forall \tilde{a}_i \in D(K),\ \ell^\circ_a(\tilde{a}_i) = i - h_a$.

A brief explanation is necessary: the occurrence net $K$ contains a unique transition $\tilde{t}$ (which is mapped in the transition $t$ of $N$), and a place for each consumed, produced, and idle token of $s$. For each place $a \in S_N$ we need exactly $n_a + k_a$ different places in $S_K$. We denote the generic $i$-th place associated to place $a$ by $\tilde{a}_i$. The set $\{\tilde{a}_i \mid a \in S_N,\ 1 \leq i \leq h_a\}$ represents the tokens which are consumed by the causal firing of $t$, i.e., the first $h_a$ tokens of each $a$-stack in the starting state. The set $\{\tilde{a}_i \mid a \in S_N,\ h_a + 1 \leq i \leq h_a + k_a\}$ represents the tokens which are produced by the causal firing of $t$, i.e., the first $k_a$ tokens of each $a$-stack in the ending state. The set $\{\tilde{a}_i \mid a \in S_N,\ h_a + k_a + 1 \leq i \leq n_a + k_a\}$ contains the remaining idle tokens, i.e., the last $n_a - h_a$ tokens of each $a$-stack of both $u$ and $u'$. Functions $\mathscr{C}$ and $\ell^\circ$ are defined accordingly to these assumptions.

**From permutation firings to processes.** Let $N$ be a PT net and $s = u[p\rangle u'$ be a permutation firing, with $u = \{n_a a\}_{a \in S_N}$ and $p = \langle \pi_a \rangle_{a \in S_N}$. The concatenable process $pr(s) = (P : K \to N, \mathscr{C}, \ell^\circ)$ is defined as follows:

- $T_K = \emptyset$ (it follows that $O(K) = D(K) = S_K$);
- $S_K = \{\tilde{a}_i \mid a \in S_N,\ 1 \leq i \leq n_a\}$, with $P(\tilde{a}_i) = a$;
- $\forall \tilde{a}_i \in O(K),\ \mathscr{C}_a(\tilde{a}_i) = i$;
- $\forall \tilde{a}_i \in D(K),\ \ell^\circ_a(\tilde{a}_i) = \pi_a(i)$.

In this case the set of transitions is empty and all the tokens remain idle. The generic place $\tilde{a}_i$ of $k$ denotes the instance of place $a$ which corresponds to the $i$-th token (from the top) of the $a$-stack of the starting state. The reorganization induced by the permutation firing is realized by differentiating the functions $\mathscr{C}$ and $\ell^\circ$.

**From causal firing sequences to processes.** The concatenable process associated to a (finite) causal firing sequence is given by the sequential composition of the concatenable processes associated to each step of the given sequence. In what follows we denote by $pr(\omega)$ the concatenable process associated to the causal firing sequence $\omega$ (up to isomorphism).

EXAMPLE 2.6 *The concatenable processes derived from the sequences $\omega$, $\omega'$ and $\omega''$ of Example 2.5 are presented in Figure 7. We use the standard notation that labels the places and transitions of the occurrence net $K$ with their images in $N$. A superscript for any initial place and a subscript for any final place denote respectively the value of $^\circ\ell$ and $\ell^\circ$. The construction of $pr(\omega'')$ is explained in detail by making explicit the concatenable processes associated to each firing component of $\omega''$.*

**Terminology.** 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 disjoint activities. The resources which are first produced and then consumed (i.e., the inner places) are called *evolution places*.

More formally, a concatenable process $C = (P : K \to N, {}^\circ\ell, \ell^\circ)$ is *connected* if and only if the set of transitions of $K$ is non-empty and moreover, for each pair $(t, t')$ of transitions of $K$ there exists an undirected path (through the arcs of the flow relation) connecting $t$ and $t'$. Process $C$ is *full* if it does not contain idle (i.e., isolated) places. Finally, the set of *evolution places* of process $C$ is the set $E_C = \{P(a) \mid a \in K,\ |{}^\bullet a| = |a^\bullet| = 1\}$.

**Connected transactions.** A causal firing sequence is essentially a "linearization" of a (concatenable) process and more than one sequence may correspond to the same (up to isomorphism) concatenable process. In fact, sequences differing in the order in which concurrent firings are executed or in the way equivalent symmetries are performed are equivalent. For example, repeatedly swapping two tokens (of the same place) for an odd number of times is equivalent to apply the swap just once.

We will consider the equivalence over sequences induced by isomorphic processes. Moreover, for the kind of sequences under consideration, the label-indexed ordering functions of origins and destinations are no longer important, so we define the equivalence on the underlying Goltz-Reisig processes, rather than on the concatenable processes.

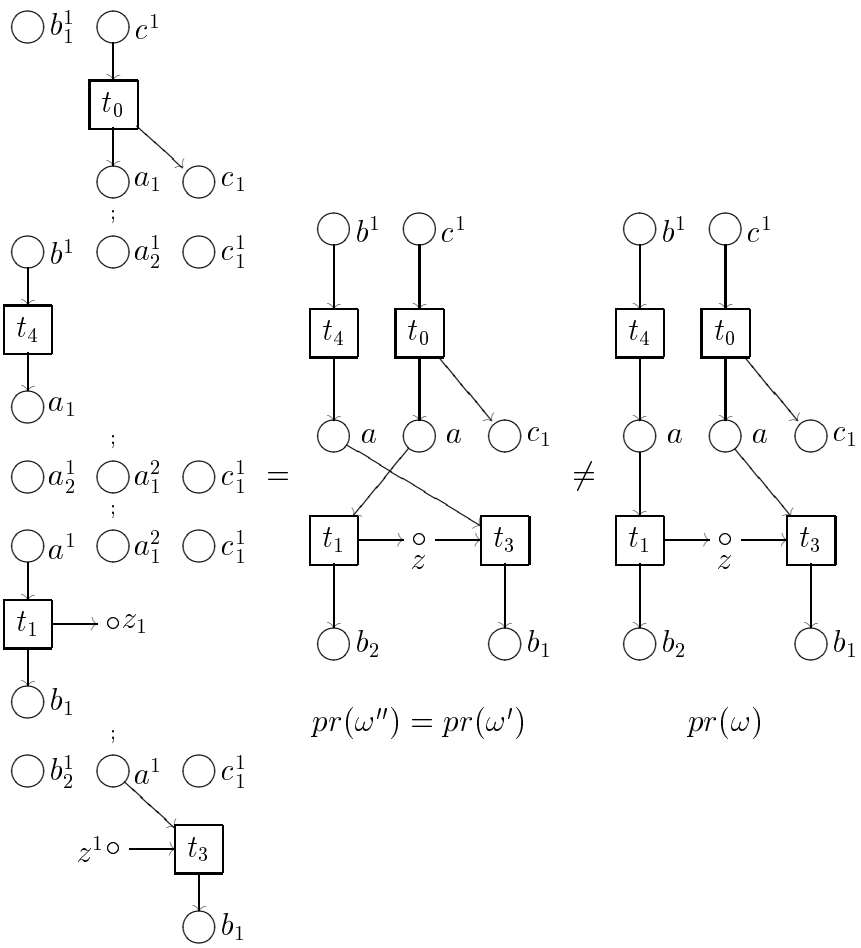


Figure 7: The concatenable processes for $\omega''$, $\omega'$, and $\omega$ of Example 2.5.

Definition 2.10 (Equivalent causal firings)
*Let $N$ be a net and $\omega, \omega'$ be two causal firing sequences. We say that $\omega$ and $\omega'$ are* causally equivalent, *written $\omega \approx \omega'$ iff $pr(\omega) = (P : K \to N, \ell, \ell^\circ)$ and $pr(\omega') = (P' : K' \to N, \ell', \ell'^\circ)$ with process $P$ isomorphic to $P'$. The equivalence class of $\omega$ is denoted by $[\![\omega]\!]_\approx$. We use $\xi$ to range over equivalence classes. Since 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.

Definition 2.11 (Connected step and transaction)
*Given a* zs *net $B$, let $\omega = s_1 \cdots s_n$ 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).*

*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 say that $u_n$ is reachable from $u_0$.*

*Furthermore, the connected step $\xi$ is a* connected transaction *of $B$ if:*

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

*We denote by $\Xi_B$ (ranged by $\delta$) the set of connected transactions of $B$.*

As for the *CTph* approach, a connected step may be applied if the starting state contains enough stable tokens to enable all the transitions independently, and no token may be left on zero places at the end of the step (nor may be found on them at the beginning of the step). This means that all the zero tokens which are produced are also consumed in the same step, and defines the synchronization mechanism. 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. Connected steps differ from stable steps in that they allow for a finer causal relationship among firings. Let us point out that all conditions in Definition 2.11 impose constraints only over the Goltz-Reisig process associated with $pr(\omega)$, thus the proof of the following proposition is straightforward.

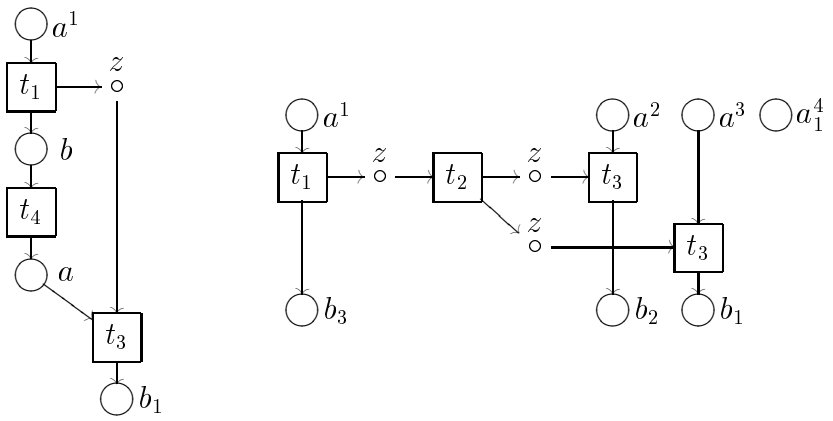


Figure 8: The concatenable processes $pr(\omega_1)$ (left) and $pr(\omega_2)$ (right).

PROPOSITION 2.1
*Conditions "stable fairness," "atomicity," "connectedness," and "fullness" (see Definition 2.11) are preserved by the equivalence* $\approx$.

The fact that connected transactions denote basic computations follows immediately from the connectedness of the associated processes (e.g., a transaction cannot be decomposed in two disjoint activities, because they must interact by definition of connectedness). Fullness ensures the absence of idle resources.

EXAMPLE 2.7 *Let us consider the* ZS *net MS of Figure 4.*

*The equivalence class of the causal firing sequence*

$$\omega_1 = \{a\}[t_1\rangle\{b, z\}[t_4\rangle\{a, z\}[t_3\rangle\{b\}$$

*is not a connected step since the "atomicity" requirement is not fulfilled. More precisely, this sequence defines a communication between an agent and itself, which is forbidden (see Figure 8, the net on the left).*

*The equivalence class of the causal firing sequence*

$$\omega_2 = \{4a\}[t_1\rangle\{3a, b, z\}[t_2\rangle\{3a, b, 2z\}[t_3\rangle\{2a, 2b, z\}[t_3\rangle\{a, 3b\}$$

*is a connected step but not a connected transaction since the associated process is not full (see Figure 8, the net on the right).*

*The equivalence class of the causal firing sequence*

$$\omega_3 = \{2a, c\}[t_1\rangle\{a, b, c, z\}[t_3\rangle\{2b, c\}[t_0\rangle\{a, 2b, c\}$$

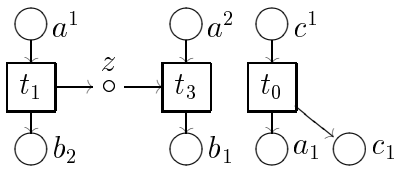


Figure 9: The concatenable process $pr(\omega_3)$.

*is a connected step but not a connected transaction since the associated process is not connected (see Figure 9).*

*The equivalence class of the causal firing sequence*

$$\{5a\}[t_1\rangle\{\cdots\}[t_2\rangle\{\cdots\}[t_2\rangle\{\cdots\}[t_2\rangle\{\cdots\}[t_3\rangle\{\cdots\}[t_3\rangle\{\cdots\}[t_3\rangle\{\cdots\}[t_3\rangle\{5b\}$$

*(where the obvious intermediate markings have been omitted), is a connected transaction.*

### 2.2.2 Abstract semantics

Also in the *ITph* based approach it is possible to define an abstract view of the systems modeled via zero-safe nets. As for the *CTph*, since transactions rewrite multisets of stable tokens, it is natural to choose a net as a candidate for the abstraction. Furthermore, since the ordering of tokens in the pre-set (post-set) of a transition is useless we should abstract from it. This is already achieved because we consider equivalence classes of causal firing sequences which denote different Goltz-Reisig processes.

When restricted to connected steps, this equivalence intuitively corresponds to limiting the symmetries of permutation firings to be vectors of permutations over zero places only, with the assumption that the stable tokens which are produced in a transaction are not reused during the same transaction. The last statement was also the basis for the *CTph* approach.

EXAMPLE 2.8 *Consider the* ZS *net MS in Figure 4, and let $s$ denote the permutation firing $\{2a\}[p\rangle\{2a\}$, where $p \in \Pi(2a)$ is the symmetry which swaps the two tokens in the place $a$, let $s' = \{2b\}[p'\rangle\{2b\}$, where $p' \in \Pi(2b)$ is the symmetry which swaps the two tokens in the place $b$, and let $\omega = \{2a\}[t_1\rangle\{a, b, z\}[t_3\rangle\{2b\}$. Clearly the causal sequences $\omega$, $s\omega$, and $\omega s'$ define the same connected transaction $\xi = [\![\omega]\!]_{\approx}$, but $pr(s\omega) \neq pr(\omega) \neq pr(\omega s')$. If we represent the connected transaction $\xi$ as a transition $t$ of a net, then its pre-set (as well as its post-set) is an unordered multiset. This means that*

*when $t$ fires it is impossible to distinguish among the tokens in $a$ and in $b$ that it produces and consumes. It follows that it makes no sense to have many different transitions to represent behaviours that we cannot reproduce at the abstract level. Thus we are forced to identify $pr(s\omega)$, $pr(\omega)$, $pr(\omega s')$, and also $pr(s\omega s')$.*

DEFINITION 2.12 (CAUSAL ABSTRACT NET)
*Let $B = (S_B, T_B, F_B, u_B, Z_B)$ be a* ZS *net. The net $I_B = (S_B \setminus Z_B, \Xi_B, F, u_B)$, with $F(a,\delta) = pre(\delta)(a)$ and $F(\delta,a) = post(\delta)(a)$, is the* causal abstract net *of $B$ (we recall that $pre(\delta)$ and $post(\delta)$ denote respectively the multisets $O(\delta)$ and $D(\delta)$, and that $\Xi_B$ is the set of all the connected transactions of $B$).*

We conclude this section by examining the causal abstract net of our multicasting system.

EXAMPLE 2.9 *Let MS be the zero-safe net defined in Figure 4. Its causal abstract net $I_{MS}$ is (partially!) depicted in Figure 6. We now comment on the possible connected transactions. Transition $t'_0$ is the basic activity which creates a new communicating process and it corresponds to $[\![\{c\}[t_0\rangle\{a,c\}]\!]_{\approx}$. Similarly $t'_4$ is the equivalence class of the firing $\{b\}[t_4\rangle\{a\}$. Each $\sigma_i^k$ 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 policy for the communication one-to-i.*

*Essentially, a generic one-to-i communication can be described as follows: a firing of $t_1$ initiates the communication, then the system executes as many firings of $t_2$ as the number of copies of the message that are needed (i.e., $i-1$ since a message is already present in the buffer), and finally $i$ firings of $t_3$ synchronize the remaining $i$ messages with $i$ active processes. The individual token philosophy distinguishes among tokens created by different firings in the same place. In this way we have a one-to-one correspondence among copy policies and the* complete *binary trees*[2] *with exactly $i$ leaves. For any $i \in \mathbb{N}$ $(i \neq 0)$, the total number of copy policies $c_i$ can be defined recursively as below (see [10]):*

$$c_i = \begin{cases} 1 & \text{if } \ i = 1 \\ \sum_{j=1}^{h} c_j \cdot c_{i-j} & \text{if } \ \exists h \in \mathbb{N}, i = 2h+1 \\ \frac{c_h \cdot (c_h+1)}{2} + \sum_{j=1}^{h-1} c_j \cdot c_{n-j} & \text{if } \ \exists h \in \mathbb{N}, i = 2h \end{cases}$$

[2]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.

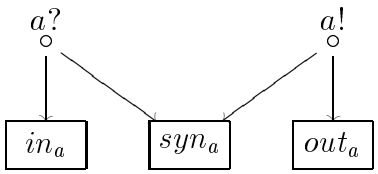


Figure 10: The ZS net $Z_a$.

# 3 Concurrent language translation

In this section we give some general hints for modeling CCS-like communication mechanism by 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 tokens. If the channel is not restricted, two additional transitions can consume the tokens separately.

More precisely, 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 $out_a$ as depicted in Figure 10. We denote by $Z(\{a_1, ..., a_n\})$ the ZS net obtained as the disjoint union of $Z_{a_1}, \ldots, Z_{a_n}$, where $\{a_1, ..., a_n\}$ is any set of channel names.

DEFINITION 3.1 (INTERFACED NET)
*A* $\{a_1, ..., a_n\}$-interfaced net *is a triple* $\langle B, \{a_1, ..., a_n\}, P\rangle$*, where* $B$ *is a* ZS *net — in our translation the initial marking will always be a set — and* $P$ *is an injective mapping from* $Z(\{a_1, ..., a_n\})$ *to* $B$*, which preserves the* ZS *net structure. The set of names* $\{a_1, ..., a_n\}$ *is also called the* interface *of the net.*

We consider a *simple process algebra* (SPA) equipped with the operations of inaction $nil$, input and output action prefix $a._-$ and $\bar{a}._-$, parallel composition $_|_$, and restriction $_\backslash a$, i.e., the agents of the process algebra SPA are defined by the grammar

$$p ::= nil \;\mid\; a.p \;\mid\; \bar{a}.p \;\mid\; p\backslash a \;\mid\; p|p.$$

Other operations, such as nondeterministic sum and recursion, could also be added, with limitations similar to those described in the literature for the existing approaches. The operational semantics for SPA agents is given in

$$\frac{}{a.p \xrightarrow{a} p} \qquad \frac{}{\bar{a}.p \xrightarrow{\bar{a}} p} \qquad \frac{p \xrightarrow{\mu} q}{p\backslash a \xrightarrow{\mu} q\backslash a}\ \mu \notin \{a, \bar{a}\}$$

$$\frac{p \xrightarrow{\mu} q}{p|r \xrightarrow{\mu} q|r} \qquad \frac{p \xrightarrow{\lambda} q,\ p' \xrightarrow{\bar{\lambda}} q'}{p|p' \xrightarrow{\tau} q|q'} \qquad \frac{p \xrightarrow{\mu} q}{r|p \xrightarrow{\mu} r|q}$$

Table 1: SOS rules for the simple process algebra SPA.

Table 1 using the SOS style [25]. We let $\mu$ range over input ($a$), output ($\bar{a}$) and silent ($\tau$) actions and let $\lambda$ range over input/output actions.

Each agent $p$ is modeled by an $fv(p)$-interfaced net $[\![p]\!]_{\text{zs}}$, where the set $fv(p)$ is the set of the free (i.e., non-restricted) channel names in $p$. The definition of $[\![p]\!]_{\text{zs}}$ 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 $\emptyset$-interfaced net $\langle B, \emptyset, \emptyset \rangle$, where $B$ consists of a single place that contains one token in the initial marking.

**Action prefix.** The interfaced net $a.\langle B, A \cup \{a\}, P \rangle$ is given by adding a new stable place $b$ and a new transition $t$ to $B$. The initial marking consists of a token in $b$. 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 $\bar{a}.p$ (we substitute $a!$ for $a?$ in the post-set of the new transition $t$).

**Parallel composition.** We let $\langle B_1, A_1, P_1 \rangle | \langle B_2, A_2, P_2 \rangle = \langle B, A_1 \cup A_2, P \rangle$, with $B$ given by the union of $B_1$ and $B_2$ where only $P_1(Z(A_1 \cap A_2))$ and $P_2(Z(A_1 \cap A_2))$ 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.** We define $\langle B, A \cup \{a\}, P \rangle \backslash a = \langle B', A, P' \rangle$, with $B' = B \setminus \{P(in_a), P(out_a)\}$ and $P'$ is $P$ restricted to $Z(A)$. If $a$ does not appear in the interface, then $\langle B, A, P \rangle \backslash a = \langle B, A, P \rangle$

The image of $Z(fv(p))$ in $B$ (via $P$) plays the rôle of the interface, since it is the only part of the net $[\![p]\!]_{\text{zs}}$ 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).

Since each transition of the interfaced nets associated to the agents consumes and produces at most one token in each zero place and there is no transitions from zero places to zero places, it follows that the corresponding abstract nets obtained under the *CTph* and the *ITph* coincide.

PROPOSITION 3.1
*For each agent $p$, with $[\![p]\!]_{\text{zs}} = \langle B, A, P\rangle$, we have $A_B = I_B$.*

For the proofs of the previous and the following propositions we refer to the PhD Thesis of first author [5].

The relation between SPA agents and their associated interfaced nets can be formalized by adding a labeling function $\phi$ from the transitions of abstract nets to the set of actions. We first need the following proposition.

PROPOSITION 3.2
*Let $p$ be an agent, then each (connected) transaction $\xi$ of $[\![p]\!]_{\text{zs}}$ contains at most one firing of transitions in $P(Z(fv(p)))$.*

DEFINITION 3.2 (LABELS OF TRANSACTIONS)
*Let $p$ be an agent. For each (connected) transaction $\xi$ of $[\![p]\!]_{\text{zs}}$, we define*

$$\phi(\xi) = \begin{cases} a_i & \text{if } a_i \in fv(p) \text{ and } P(in_{a_i}) \text{ is fired in } \xi \\ \bar{a}_i & \text{if } a_i \in fv(p) \text{ and } P(out_{a_i}) \text{ is fired in } \xi \\ \tau & \text{otherwise} \end{cases}$$

EXAMPLE 3.1 *The interfaced net for the agent $(a_1.a_2.nil|\bar{a}_2.nil)\backslash a_2$ is presented in Figure 11, together with its (labeled) abstract net.*

It is out of the scope of this presentation to investigate the relationships between the net semantics we propose for SPA agents and those already known in the literature for CCS-like algebras. We just remark that our translation is very linear, and that it provides a reasonable concurrent semantics for the simple class of agents considered here. This last assertion is supported by Propositions 3.3 and 3.4.

DEFINITION 3.3 (BISIMILARITY BETWEEN AGENTS AND MARKINGS)
*Let $p$ be an agent, let $N$ be a net whose transitions are labeled by $\phi$ over the set of actions, and let $u$ be a marking of $N$. We say that $p$ is* bisimilar *to*

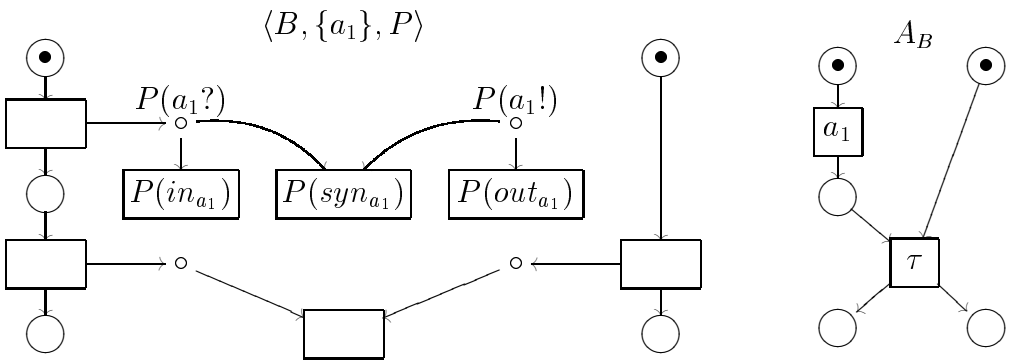


Figure 11: An interfaced net (left) and its abstract net (right).

*$u$ in $N$ if there exists a relation $\sim$ between agents and markings of $N$ such that $p \sim u$, and:*

1. *for each transition $p \xrightarrow{a} p'$ there exists a firing $u[t\rangle u'$ of $N$ such that $\phi(t) = a$ and $p' \sim u'$;*
2. *for each firing $u[t\rangle u'$ of $N$ with $\phi(t) = a$ there exists a transition $p \xrightarrow{a} p'$ such that $p' \sim u'$.*

PROPOSITION 3.3
*Let $p$ be an agent, and $[\![p]\!]_{\text{zs}} = \langle B, A, P\rangle$, then $p$ is bisimilar to the initial marking of the abstract net $A_B$.*

The restriction operator allows hiding local names w.r.t. some external observer. Therefore, a restricted name has only a local scope, and agents that differ only for the local names (i.e., agents that can be obtained one from the other by $\alpha$-conversion) are usually considered equivalent. We use the symbol $_ \equiv_\alpha _$ to denote such equivalence. For example, we have

$$(a_1.a_2.nil|\bar{a}_2.nil)\backslash a_2 \equiv_\alpha (a_1.a_3.nil|\bar{a}_3.nil)\backslash a_3$$

for any $a_3 \neq a_1$. Proposition 3.4 shows that our translation deals very well with $\alpha$-conversion.

DEFINITION 3.4
*Two $A$-interfaced nets $\langle B, A, P\rangle$ and $\langle B', A, P'\rangle$ are* isomorphic *if there exists a* ZS *net isomorphism $\psi$ from $B$ to $B'$ that respects interfaces, i.e., such that $\psi(P(x)) = P'(x)$ for each element $x \in Z(A)$ (either place or transition).*

PROPOSITION 3.4
*If $p$ and $q$ are agents such that $p \equiv_\alpha q$, then $[\![p]\!]_{\text{zs}}$ and $[\![q]\!]_{\text{zs}}$ are isomorphic.*

# 4 Universal constructions

In [10] we have presented both the *CTph* and *ITph* categorical semantics for ZS nets. For the *CTph* approach, our categorical models are based on monoidal graphs equipped with an operation of horizontal composition, and for the *ITph* approach, we have introduced an additional collection of special transitions, called *swappings*, to represent the permutations of tokens which are all present at the same time into the same place, following the style of [12]. The categorical semantics (summarized by the four adjunctions in Figure 12) recover the operational and abstract semantics presented in this paper, introducing an algebraic characterization of the whole framework.

The first construction starts from a category **dZPetri** (where zero-safe nets are considered as programs) and exhibits an adjunction to a category **HCatZPetri** consisting of some kind of machines equipped with suitable operations defined over states and transitions (e.g., parallel composition and a special kind of sequential composition, called *horizontal*). This adjunction corresponds to the token-pushing semantics of ZS nets defined in Section 2.1, in the sense that the transitions of the machine $\mathcal{Z}[B]$ associated to a zero-safe net $B$ are exactly the abstract stable steps of $B$. Moreover, the notion of abstract stable transaction exactly corresponds to the algebraic characterization of special transitions of $\mathcal{Z}[B]$, 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 2.6. We remark that **ZSN** allows one to map transitions of a machine into prime arrows of another machine (i.e., transactions), yielding a very general notion of "implementation morphism." It is worth noticing that the counit component $\epsilon^C_B$ of the coreflection maps transitions of the abstract net $A_B$ into appropriate abstract transactions of $B$.

The third construction is an adjunction from **dZPetri** to a category **ZSCGraph** of more structured models, equipped not only with parallel and horizontal compositions as in **HCatZPetri**, but also with a family of *swappings* playing the rôle of zero token permutations. This adjunction is strictly related to the operational semantics of ZS nets as defined in Section 2.2 (again, the connected transactions are characterized as prime arrows).

Our fourth 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 in Definition 2.12 and whose counit component $\epsilon^I_B$ maps transitions of the causal abstract net $I_B$ into appropriate connected transactions of $B$.

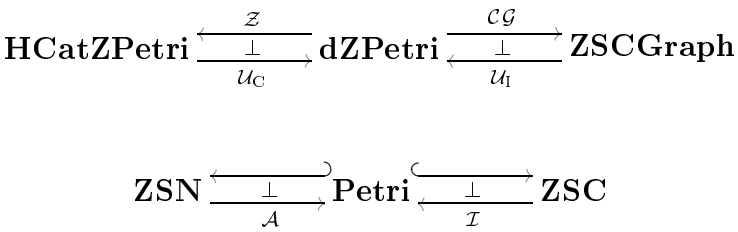


Figure 12: Operational and abstract semantics of zero-safe nets.

The universal property of coreflection confirms that the abstract counterparts $A_B$ and $I_B$ of a ZS net $B$ are the PT nets that better approximate the behaviour of $B$ according to the $\mathsf{CTph}$ and the $\mathsf{ITph}$ interpretation respectively. Moreover, since left (right) adjoints preserve colimits (limits), it follows that several constructions on the refined model are preserved by the operational and abstract semantics. For instance, the algebraic model of the *pushout* of two nets — which is often useful for combining two nets merging some of their places — is the pushout of their semantics.

# 5 Some ideas on ZS nets unfolding

The unfolding gives a constructive way to generate all the possible computations of a net. The construction is, by its nature, closer to the $\mathsf{ITph}$ rather than to the $\mathsf{CTph}$, but it is not straightforward as there are many difficulties for systems where transactions are distributed. For example, in concurrent constraint programming, concurrent agents can add constraints to their local stores, but the consistency of the global store (i.e., the validation of local computations) is checked only at given instants and acts as a global commit. In our framework this would mean to perform many disjoint computations without any restriction on zero tokens, and then check if their composition can define a transaction, which is clearly a too optimistic approach.

At a very abstract level, an implementation can be described as follows. A transaction can be started independently by the components that want to interact, i.e., the various components may initiate mutually disjoint transaction pieces. For example we can think that every time a transaction starts, a fresh "color" (i.e., a unique *key*) is assigned to it. We assume the color *white* to denote non-frozen stable tokens. The firing of a transition that consumes *white* tokens starts a transaction, e.g. colored in *col*, and all the produced resources (zero and frozen stable) carry the same color *col*. If no zero token

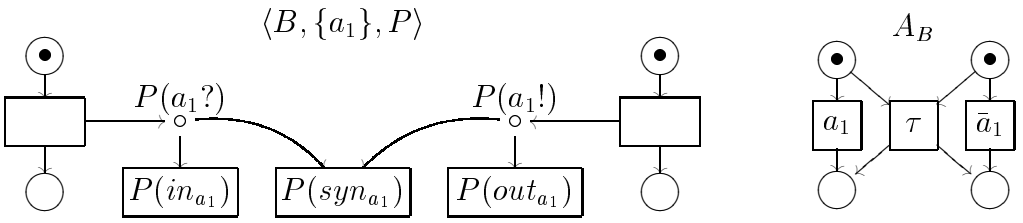


Figure 13: The interfaced net $[\![a_1.nil|\bar{a}_1.nil]\!]_{\text{zs}}$ and its abstract net.

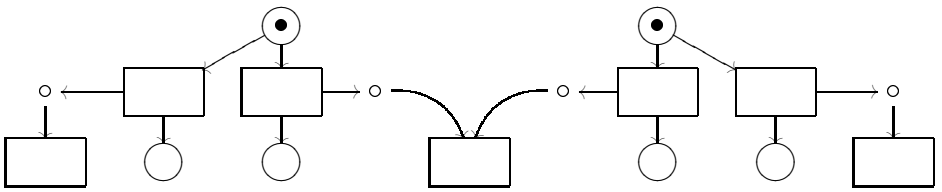

Figure 14:

is produced, then the transition defines a transaction itself and therefore the produced stable tokens can be released and colored in *white*. When a transition synchronizes (consumes) zero tokens of different colors, say $col_1$,...,$col_n$ (we remind that only *white* stable tokens can be consumed), then some global mechanism must take control and paint in the same color every element colored in $col_1$,...,$col_n$ (it can be either a fresh color or a unique combination of the different colors or a certain color $col_i$ chosen according to a suitable ordering on colors). If a transition consumes all the zero tokens of $col_1$,...,$col_n$, and produces only stable tokens, it acts as a commit action and produces *white* stable tokens. In this case also the stable tokens colored by $col_1$,...,$col_n$ must become *white*. This emphasizes the fact that the same instance of a resource can be possibly released by many different commit actions, which is counterintuitive for an approach based on events. One possibility would be to look at resources of the same color as an atomic entity, but this would disregard their distributed nature and would reduce the concurrency degree of the model (after the synchronization, locally concurrent actions can only be interleaved).

The problem is that one wants to keep the control of computation flow as much distributed as possible, but in this way a local "commit" that closes the transaction must be able to act as a global action. This is reflected in ZS nets by assuming that the stable tokens produced by the transitions are frozen until the transaction ends and then are released all together. This means

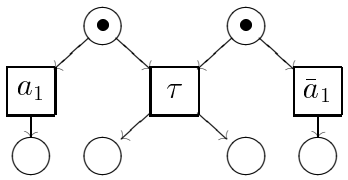


Figure 15:

that, in the ordinary unfolding, stable resources can also depend on some side action able to synchronize via zero places with the transition that produced the resources. For example, let us consider the interfaced net $[\![a_1.nil|\bar{a}_1.nil]\!]_{\text{zs}}$ in Figure 13 (see Section 3). It coincides with its ordinary unfolding, whereas one might expect that different instances of stable places are reached when different synchronizations are performed (see Figure 14, where, in some sense, conflicts on zero places are moved to the level of stable places), as it is the case in the unfolding of the associated abstract net (see Figure 15, which also represents the abstract net of the unfolding in Figure 14).

When considering the unfolding of the underlying net $N_B$ of a zs net $B$, a first problem that might arise is that stable tokens must not be used before the transaction is closed, so to avoid situations as the one illustrated in Figure 8 (where the token in $b$ is used to close the transaction that produced it). This situation is very hard to handle, but notice that if we consider the abstraction of such unfolding, then situations of that kind are discharged, although forward conflicts may arise in the resulting net.

The unfolding of abstract nets is of course a good candidate for building the unfolding of zs nets, but when refining the events to zs net transactions it might happen that different instances of the same resource consumed by the same event are confused, because a transition is refined by a process (origins and destinations are not ordered, see Example 2.8) and not by a (decorated) concatenable process, which would preserve the causal relationships.

One possibility to solve this problem could be to consider *pre-nets* rather than nets. The difference is that the states of pre-nets are strings (i.e., elements of the free monoid over places) rather than multisets (i.e., elements of the free commutative monoid over places). Pre-nets have been introduced by José Meseguer, Vladimiro Sassone, and the authors in [7], where it is shown that they fit well in the *ITph*, better than PT nets. In particular, pre-nets can be equipped with a satisfactory functorial semantics. Moreover, the semantics of the PT net associated to a pre-net by abstracting strings to multisets can be recovered from the semantics of the pre-net (i.e., any two pre-nets associated to the same PT net have exactly the same semantic

model). We conjecture that in the framework of pre-nets the refinement of the unfolding could be correctly defined, but we leave this as future work.

# Conclusion

In this paper we have presented zero-safe nets as a refined model of ordinary place/transition Petri nets. ZS nets include an additional simple mechanism for the synchronization of transition firings. We have given concurrent operational and abstract semantics for zero-safe nets under both the *CTph* and the *ITph*. The differences between the two approaches have been discussed on the basis of a multicasting system representation. We have shown that, independently from the chosen philosophy, the ZS net *MS* modeling the multicasting system yields abstract PT nets with infinitely many transitions. The distinction between the two abstract nets has been characterized in terms of the feasible one-to-many communications: The *ITph* approach can faithfully model different copy policies, whereas the *CTph* approach can only record the number of agents involved in a communication. We have also shown that CCS-like process algebras can be easily interpreted by representing the communication channels as zero places, in the style of the multicasting example.

Our results mainly concern process semantics, algebraic semantics and their connections. A satisfactory unfolding semantics seems far from being trivial, and deserves a deeper investigation. Some hinted solutions have been proposed in Section 5.

**Acknowledgements**

We want to thank Paolo Baldan and Daniela Giorgetti for the careful reading of the paper and precious comments.

# References

[1] A. Benveniste, P. Le Guernic, and C. Jacquemot, Synchronous programming with events and relations: The Signal language and its semantics, *Science of Computer Programming* **16**, 103–149 (1991).

[2] E. Best and R. Devillers, Sequential and concurrent behaviour in Petri net theory, *Theoretical Computer Science* **55**(1), 87–136 (1987).

[3] E. Best, R. Devillers, and J. Hall, The box calculus: A new causal algebra with multi-label communication, in: G. Rozenberg, Ed., *Advances in Petri Nets 1992*, *LNCS* **609**, 21–69, Springer-Verlag (1992).

[4] C. Brown and D. Gurr, A categorical linear framework for Petri nets, in *Proceedings 5th LICS Symposium*, 208–218, IEEE Computer Society Press (1990).

[5] R. Bruni, *Tile Logic for Synchronized Rewriting of Concurrent Systems*, PhD Thesis TD-99-01, Department of Computer Science, University of Pisa (1999).

[6] 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, Eds., *Proceedings 4th ASIAN'98*, *LNCS* **1538**, 225–244, Springer-Verlag (1998).

[7] R. Bruni, J. Meseguer, U. Montanari, and V. Sassone, Functorial semantics for Petri nets under the individual token philosophy, in: M. Hofmann, Ed., *Proceedings 8th CTCS'99*, *ENTCS*, Elsevier, to appear.

[8] R. Bruni and U. Montanari, Zero-safe nets, or transition synchronization made simple, in: C. Palamidessi and J. Parrow, Eds., *Proceedings EXPRESS'97*, *ENTCS* **7**, 19 pages, Elsevier (1997).

[9] R. Bruni and U. Montanari, Zero-safe nets: The individual token approach, in: F. Parisi-Presicce, Ed., *Proceedings 12th WADT'97*, *LNCS* **1376**, 122–140, Springer-Verlag (1998).

[10] R. Bruni and U. Montanari, Zero-safe nets: Comparing the collective and the individual token approaches, *Information and Computation*, to appear.

[11] P. Degano, R. De Nicola, and U. Montanari, A distributed operational semantics for CCS based on condition/event systems, *Acta Informatica* **26**(1-2), 59–91 (1988).

[12] P. Degano, J. Meseguer, and U. Montanari, Axiomatizing the algebra of net computations and processes, *Acta Informatica* **33**(7), 641–667 (1996).

[13] R.J. van Glabbeek and G.D. Plotkin, Configuration structures, in: *Proceedings 10th LICS Symposium*, 199–209, IEEE Computer Society Press (1995).

[14] 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, Eds., *Proceedings of PARLE*, *LNCS* **259**, 224–242, Springer-Verlag (1987).

[15] U. Goltz and W. Reisig, The non-sequential behaviour of Petri nets, *Information and Computation* **57**, 125–147 (1983).

[16] R. Gorrieri and U. Montanari, On the implementation of concurrent calculi into net calculi: Two case studies, *Theoretical Computer Science* **141**(1-2), 195–252 (1995).

[17] A. Mazurkiewicz, Trace theory, in: W. Brauer, W. Reisig, and G. Rozenberg, Eds., *Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986*, *LNCS* **255**, 279–324, Springer-Verlag (1987).

[18] J. Meseguer and U. Montanari, Petri nets are monoids, *Information and Computation* **88**, 105–154 (1990).

[19] J. Meseguer, U. Montanari, and V. Sassone, On the semantics of Petri nets, in: W.R. Cleaveland, Ed., *Proceedings CONCUR'92*, *LNCS* **630**, 286–301, Springer-Verlag (1992).

[20] J. Meseguer, U. Montanari, and V. Sassone, On the Semantics of Place/Transition Petri Nets, *Mathematical Structures in Computer Science* **7**(4), 359-397 (1997).

[21] J. Meseguer, U. Montanari, and V. Sassone, Representation theorems for Petri nets, in: W. Brauer, C. Freksa, M. Jantzen, and R. Valk, Eds., *Foundations of Computer Science*, *LNCS* **1337**, 239–249, Springer-Verlag (1997).

[22] R. Milner, *Communication and Concurrency*, Prentice-Hall (1989).

[23] M. Nielsen, G. Plotkin, and G. Winskel, Petri nets, event structures and domains, part I, *Theoretical Computer Science* **13**, 85–108 (1981).

[24] E.R. Olderog, Operational Petri net semantics for CCSP, in: G. Rozenberg, Ed., *Advances in Petri Nets 1987*, *LNCS* **266**, 196–223, Springer-Verlag (1987).

[25] G. Plotkin, A structural approach to operational semantics, Technical Report DAIMI FN-19, Computer Science Department, Aarhus University (1981).

[26] W. Reisig, *Petri Nets*, Springer-Verlag (1985).

[27] G. Ristori, *Modelling Systems with Shared Resources via Petri Nets*, PhD thesis TD-94-05, Department of Computer Science, University of Pisa (1994).

[28] D. Scott, Outline of a mathematical theory of computation, in: *Proceedings 4th Annual Princeton Conference on Information Science and Systems*, 169–176 (1970).

[29] V. Sassone, An axiomatization of the algebra of Petri net concatenable processes, *Theoretical Computer Science* **170**(1-2), 277–296 (1996).

[30] G. Winskel, Event structure semantics of CCS and related languages, in: M. Nielsen and E. Meineche Schmidt, Eds., *Proceedings ICALP'82*, *LNCS* **140**, 561–567, Springer-Verlag (1982).

[31] G. Winskel, Event structures, in: W. Brauer, Ed., *Proceedings Advanced Course on Petri Nets*, *LNCS* **255**, 325–392, Springer-Verlag (1987).

[32] G. Winskel, Petri nets, algebras, morphisms and compositionality, *Information and Computation* **72**, 197–238 (1987).