Non-Atomic Transition Firing in Contextual Nets*

Thomas Chatain[1], Stefan Haar[1], Maciej Koutny[2], and Stefan Schwoon[1]

[1] INRIA and LSV, CNRS and ENS Cachan, France
[2] University of Newcastle-upon-Tyne, UK

Abstract. The firing rule for Petri nets assumes instantaneous and simultaneous consumption and creation of tokens. In the context of ordinary Petri nets, this poses no particular problem because of the system's asynchronicity, even if token creation occurs later than token consumption in the firing. With read arcs, the situation changes, and several different choices of semantics are possible. The step semantics introduced by Janicki and Koutny can be seen as imposing a two-phase firing scheme: first, the presence of the required tokens is checked, then consumption and production of tokens happens. Pursuing this approach further, we develop a more general framework based on explicitly splitting the phases of firing, allowing to synthesize coherent steps. This turns out to define a more general non-atomic semantics, which has important potential for safety as it allows to detect errors that were missed by the previous semantics. Then we study the characterization of partial-order processes feasible under one or the other semantics.

1 Introduction

There are some aspects of concurrent behaviour that cannot be modeled by sequences of actions nor by partial orders alone (c.f. [9, 11]). An example is the 'earlier than or simultaneous' (that is, 'not later than') relationship [11], for which neither sequences nor partial orders are expressive enough. Consider, for example, a priority system with three actions: a, b, and c such that c has higher priority than both a and b. Initially, a and b can be executed simultaneously, while c is blocked. Moreover, completing a or b permanently enables action c. Using sequences of actions, we cannot capture the execution where a and b are executed in the same run of the system, as both (ab) and (ba) would violate the priority constraint. However, a sequence $(\{a, b\})$, representing a step in which actions a and b are executed simultaneously, faithfully reflects a possible scenario in which both a and b are executed. Consider now a modified system in which executing a no longer enables action c. In such a case, two executions involving the actions a and b are possible, namely $(\{a, b\})$ and (ab). Now, this behavior is *not* reflected by the partial order in which a and b are concurrent; for in that case, sequence (ba) would emerge as a valid system behaviour, which it is *not*

* This work is partially supported by the UK EPSRC project UNCOVER.

according to the above specification. To cover such cases, [12] used structures richer than causal partial orders and, in this particular case, introduced the notion of a 'weak causality' between a and b, meaning that 'a can be earlier than or simultaneous with b', but 'not later than b'. In the resulting model, causality (partial order) is augmented with weak causality leading to *stratified order structures* [8, 10, 12], which extend the standard causal partial orders if the underlying concurrent system does not exhibit features like priorities in the above example. Stratified order structures have been successfully applied to model, e.g., inhibitor and priority systems and asynchronous races (see, e.g. [12, 14, 16]). Extensions of the standard partial order model of concurrency to cover features such as priorities as well as inhibitor and read arcs in the elementary net systems are systematically discussed in [17].

Let us turn now to the more specific model class of Petri nets. Many distributed systems allow read-only access to some data. These non-destructive accesses can be done concurrently by several components of the system. In order to model these read-only accesses with Petri nets, a classical method is to design a loop in which some transition consumes and rewrites a token on the same place. Nevertheless this technique is not satisfactory when one is dealing with causal semantics because the consumption of the token artificially enforces an order on the events accessing the same data.

In order to solve this problem, read arcs were added to Petri nets [5, 22]. This extension is now quite commonly used, and partial order semantics were proposed for this new model [3, 4, 31, 28]. In the same vein, inhibitor arcs were also introduced [12, 5]. Their expressive power is similar to the one of read arcs in the case of bounded nets. Finite complete prefixes of Petri nets with read arcs (also called contextual Petri nets) were first defined in the restricted case of *read-persistent nets* [29], and later in the general case [32]. Efficient procedures exist for the computation and analysis of finite complete prefixes for safe Petri nets with read arcs [2, 25].

In the present paper, we push the analysis of contextual Petri nets further in the direction of collective, or non-atomic, firing of several transitions jointly, in one *step*, where a step is seen here as a set of transitions (or multi-set in the case of non safe nets). Giving a semantics that allows this is not problematic in ordinary Petri nets; a step is enabled iff the current marking is bigger than the sum of all presets of its transitions, both seen as vectors whose dimension is the number of places. With read arcs, the situation changes, and several different choices of step semantics are possible. The one introduced in [12] can be seen as imposing a two-phase firing scheme: first, the presence of the required tokens is checked, then consumption and production of tokens happens. Here, we develop a more general framework based on explicitly splitting the phases of firing, allowing to synthesize coherent steps. This turns out to define a more general non-atomic semantics. We will recall the fundamentals of Contextual Petri nets in Section 2, and develop the non-atomic sequential semantics in Section 3. In Section 4, we continue with the study of non-sequential, partial order semantics with non-atomic firing; finally, Section 5 concludes.

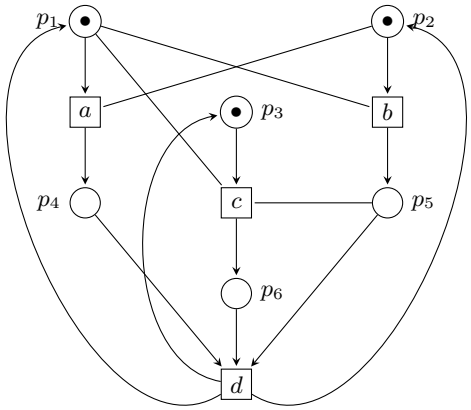

Fig. 1. A contextual Petri net.

2 Contextual Petri Nets

2.1 Definition

We consider only *safe* contextual Petri nets (PNs), i.e. PNs where there is never more than one token in a place. We discuss the general case in Section 5.

Definition 1 (Contextual Petri Net (PN)). *A* contextual Petri net *is a tuple* $(P, T, pre, cont, post, M_0)$ *where* P *and* T *are finite sets of* places *and* transitions *respectively, pre and post map each transition* $t \in T$ *to its (nonempty)* preset *denoted* $^\bullet t \stackrel{\text{def}}{=} pre(t) \subseteq P$*, its (possibly empty)* context *denoted* $\underline{t} \stackrel{\text{def}}{=} cont(t) \subseteq P \setminus {}^\bullet t$ *and its (possibly empty)* postset *denoted* $t^\bullet \stackrel{\text{def}}{=} post(t) \subseteq P$*;* $M_0 \subseteq P$ *is the* initial marking*.*

We usually denote $^\bullet\underline{t} \stackrel{\text{def}}{=} {}^\bullet t \cup \underline{t}$. For simplicity, we assume that for any transition t, its context is disjoint from its preset and postset.

A contextual Petri net is represented as a graph with two types of nodes: places (circles) and transitions (rectangles). Presets are represented by arrows from places to transitions, postsets by arrows from transitions to places, and contexts by undirected edges between places and transitions. The initial marking is represented by tokens in places. Figure 1 shows an example of a contextual Petri net. The transition a, for instance, has p_1 in its preset, p_2 in its context and p_4 in its postset.

2.2 Atomic Semantics

A *marking* of a safe contextual Petri net is a set $M \subseteq P$ of marked places. A Petri net starts in its *initial marking* M_0. A transition $t \in T$ is *enabled* in a

marking M if all the places of its preset and context are marked, i.e. ${}^\bullet t \cup \underline{t} \subseteq M$. Then t can *fire* from M, leading to the marking $M' \stackrel{\text{def}}{=} (M \setminus {}^\bullet t) \cup t^\bullet$.

Again, we consider only *safe* contextual Petri nets, that is we assume that if a transition $t \in T$ is enabled in a marking M, then $(M \setminus {}^\bullet t) \cap t^\bullet = \emptyset$.

Definition 2 (Atomic semantics, a-run). *We call* firing sequence of N under the atomic semantics*, or* a-run*, any sequence* $\sigma \stackrel{\text{def}}{=} (t_1 \dots t_n)$ *of transitions for which there exist markings* $M_1, \dots, M_n$ *such that for all* $i \in \{1, \dots, n\}$*, firing* t_i *from* M_{i-1} *is possible and leads to* M_i*.*

For instance, the net in Figure 1 has two possible firing sequences: (a) and (bc). However, it is never possible to fire d because that would require to fire both a and b first, and firing one of a, b disables the other.

3 Non-atomic Semantics

In this section, we discuss two semantics for concurrent firing of multiple transitions. One is the well-known *step semantics* [11], in which multiple transitions can fire simultaneously. This is typically the case of a and b in the net of Figure 1, which are enabled simultaneously and have disjoint presets, but cannot fire together according to the atomic semantics. The step semantics can be interpreted as first checking whether all members of a set of transitions can fire, and then firing them either simultaneously or one by one, in any order. We then introduce a new, so-called *interval semantics*, which allows a more liberal choice of checking and firing transitions in a set.

We present the semantics under the assumption that the underlying net is safe even under these two semantics, which allow more possibilities than the atomic one.

3.1 Step Semantics

We first recall the step semantics [11].

Definition 3 (Step semantics, s-run). *Let* N *be a PN. We call* s-run *of* N *any sequence* $\sigma \stackrel{\text{def}}{=} (T_1 \dots T_n)$ *of* sets *of transitions for which there exist markings* $M_1, \dots, M_n$ *such that for all* $i \in \{1, \dots, n\}$*,*

- *every* $t \in T_i$ *is enabled in* M_{i-1}*,*
- *the presets of the transitions in* T_i *are disjoint, and*
- $M_i = (M_{i-1} \setminus \bigcup_{t \in T_i} {}^\bullet t) \cup \bigcup_{t \in T_i} t^\bullet$*.*

In the example of Figure 1, the step semantics allows one to fire a and b in one step since they are both enabled in the initial state and ${}^\bullet a \cap {}^\bullet b = \emptyset$. This gives the s-run $(\{a, b\})$ in addition to the others which were already possible under the atomic semantics; for instance the a-run involving b followed by c, (denoted (bc) for the atomic semantics), is simply rewritten as the s-run $(\{b\}\{c\})$ under the

step semantics. However, transition d remains dead since none of these s-runs contains all of a, b, and c.

The intuitive model underlying the step semantics is that all the transitions in the step can first check, in any order, whether they are enabled and not in conflict with one another. Once the checks have been performed, they can all fire, again in any order. Put differently, if we denote the checking phase of a transition t by t^- and its firing phase by t^+, then every step consists of any permutation of the actions of type t^- (for all transitions t in the step), followed by any permutation of the actions t^+. The notion introduced in Definition 4 formalizes this intuition.

Definition 4 ($\mathrm{s}^{\pm}$-run). *For every s-run $(T_1 \dots T_n)$ of a contextual Petri net N, every concatenation $u_1^-.u_1^+.\cdots.u_n^-.u_n^+$ of sequences u_i^- and u_i^+, is a* $\mathrm{s}^{\pm}$-run *of N, where every u_i^- is a permutation of the set $\{t^- \mid t \in T_i\}$ and every u_i^+ is a permutation of the set $\{t^+ \mid t \in T_i\}$ (remember that T_i is a set of transitions of N).*

For example, the s-run $(\{b\}\{c\})$ yields the $\mathrm{s}^{\pm}$-run $(b^-b^+c^-c^+)$ and the s-run $(\{a,b\})$ yields four $\mathrm{s}^{\pm}$-runs: $(a^-b^-a^+b^+)$, $(a^-b^-b^+a^+)$, $(b^-a^-a^+b^+)$ and $(b^-a^-b^+a^+)$.

3.2 Splitting Transitions for Understanding Steps

Definition 4 formalizes a new semantics of PNs, in which the firing of a transition does not happen atomically, but in two steps, the checking of the pre-conditions and the actual execution. In this section, we generalize this idea.

The left-hand side of Figure 2 shows a part of the net in Figure 1, which consists of transition a with its preset $\{p_1\}$, context $\{p_2\}$, and postset $\{p_4\}$. The construction on the right-hand side of 2 illustrates the idea of splitting firing transitions into two phases:

- Every transition t is split into t^- and t^+.
- Every place p is duplicated to p^c (meaning token in p available for consumption) and p^r (meaning token in p available for reading).

Similar ideas about splitting transitions can be found in several works, for instance in [27].

Intuitively, if we apply this construction to all transitions from Figure 1, then the $\mathrm{s}^{\pm}$-runs of that net correspond to a-runs of the newly constructed net. The following Definition 5 provides the precise details of the construction.

Definition 5 ($\boldsymbol{split(N)}$). *For every contextual Petri net $N = (P, T, pre, cont, post, M_0)$, we define the contextual Petri net $split(N) \stackrel{\text{def}}{=} (P', T', pre', cont', post', M_0')$ where*

- *T' contains two copies, denoted t^- and t^+ of every transition $t \in T$.*
- *P' contains two copies, denoted p^c and p^r of every place $p \in P$, plus one place p_t per transition $t \in T$.*

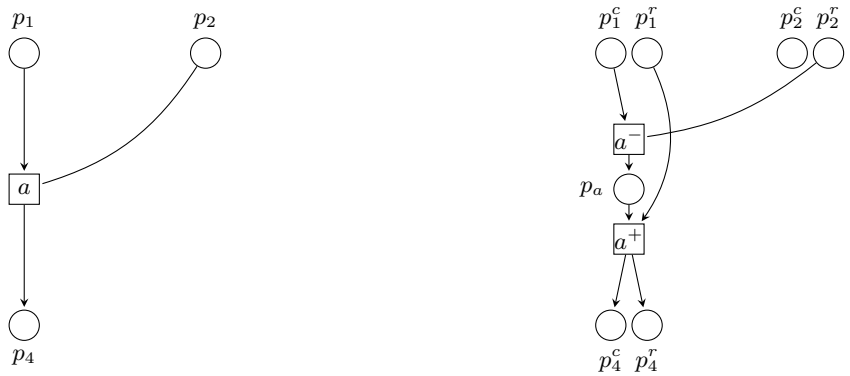

Fig. 2. The splitting of transition a (left) into a^- and a^+ (right).

- ${}^\bullet t^- \stackrel{\text{def}}{=} \{p^c \mid p \in {}^\bullet t\}$
- $\underline{t^-} \stackrel{\text{def}}{=} \{p^r \mid p \in \underline{t}\}$
- $t^{-\bullet} \stackrel{\text{def}}{=} \{p_t\}$
- ${}^\bullet t^+ \stackrel{\text{def}}{=} \{p^r \mid p \in {}^\bullet t\} \cup \{p_t\}$
- $\underline{t^+} \stackrel{\text{def}}{=} \emptyset$
- $t^{+\bullet} \stackrel{\text{def}}{=} \{p^c \mid p \in t^\bullet\} \cup \{p^r \mid p \in t^\bullet\}\}$
- $M_0' \stackrel{\text{def}}{=} \{p^c \mid p \in M_0\} \cup \{p^r \mid p \in M_0\}$

We now formally prove the intuition mentioned above:

Lemma 1. *Every $s^\pm$-run $\sigma^\pm$ of N is a a-run of $split(N)$. Moreover $\sigma^\pm$ reaches the marking $\{p^c \mid p \in M\} \cup \{p^r \mid p \in M\}$, where M is the marking of N reached after the s-run σ from which $\sigma^\pm$ is obtained.*

Proof. We proceed by induction on the length of σ. The case $\sigma = ()$ is trivial. Now, let $\sigma^\pm = u_1^-.u_1^+.\cdots.u_n^-.u_n^+$ be a $s^\pm$-run obtained from a s-run $\sigma = (T_1 \dots T_n)$, assume the property true for $u_1^-.u_1^+.\cdots.u_{n-1}^-.u_{n-1}^+$ and denote M_{n-1} the marking reached after $(T_1 \dots T_{n-1})$. By induction hypothesis, $u_1^-.u_1^+.\cdots.u_{n-1}^-.u_{n-1}^+$ reaches the marking $\{p^c \mid p \in M_{n-1}\} \cup \{p^r \mid p \in M_{n-1}\}$ of $split(N)$. The fact that T_n is a valid step from M_{n-1} implies that $\bigcup_{t \in T_n} {}^\bullet \underline{t} \subseteq M_{n-1}$ and that the presets of the transitions in T_n are disjoint. This allows one to fire all the t^-, $t \in T_n$ in any order and reach the marking $\{p^c \mid p \in M_{n-1} \setminus \bigcup_{t \in T_n} {}^\bullet t\} \cup \{p^r \mid p \in M_{n-1}\} \cup \{p_t \mid t \in T_n\}$ of $split(N)$. Now the t^+, $t \in T_n$, are all enabled and their presets are disjoint. They can in turn be fired in any order, reaching the desired marking of $split(N)$. □

Note that the converse of Lemma 1 does not hold. For instance, for the net N from Figure 1, the net $split(N)$ admits the a-run $a^-b^-b^+c^-c^+a^+$, which is not an $s^\pm$-run of N.

3.3 Interval Semantics

We have seen that the construction $split(N)$ admits firing sequences that cannot be mapped back to executions under either the atomic or the step semantics. In this section, we shall introduce a new, so-called *interval semantics*, which is more general than the step semantics, and whose interpretation on a net N does correspond to the feasible executions in $split(N)$.

Definition 6 (Interval semantics, i-run). *Every a-run of $split(N)$ is called* i-run *of N, or run of N under the interval semantics.*

Coming back to the example of Figure 1, transition d can fire under the interval semantics, for instance after the i-run $a^-b^-b^+c^-c^+a^+d^-d^+$ where transitions b and c complete the firing during the period in which a fires. Under the atomic semantics, a and b are in conflict, which prevents d from firing. Under the step semantics, a and b can fire in the same step, but then c cannot fire. Under the interval semantics, d can also fire.

Recall that we introduced t^- and t^+ to represent different phases during the execution of transition t. An obvious question is whether the new semantics can lead to runs in which a transition 'gets stuck' during its execution. The following Lemma 2 affirms that this is not the case: once t^- is fired, nothing can hinder t^+ from firing, too.

Definition 7 (complete i-run). *An i-run is* complete *if every t^- is matched by a t^+.*

Lemma 2. *Every i-run can be completed: for every i-run σ, there exists a suffix μ which matches all the unmatched t^-, and such that $\sigma\mu$ is an i-run.*

Proof. As long as a t^- is unmatched, ${}^\bullet t^+$ remains included in the marking: no other transition consumes these tokens. Hence it suffices to fire all the t^+ corresponding to the unmatched t^-, in any order. □

3.4 Comparison of Sequential Semantics

This section provides a brief summary and comparison of the previously discussed semantics. To simplify the comparison, we first need a technical definition that allows to represent atomic runs in a form comparable to i-runs. The following Definition 8 simply makes explicit the assumption that the firing of a transition is atomic: in terms of i-runs, every t^- is immediately followed by the corresponding t^+.

Definition 8 ($\mathrm{a}^\pm$-run). *For every a-run $(t_1 \dots t_n)$, the sequence $t_1^- t_1^+ \dots t_n^- t_n^+$ is called an* $\mathrm{a}^\pm$-run.

We can now turn to comparing the different sequential semantics based on (complete) i-runs. It is immediate that every $\mathrm{a}^\pm$-run of a contextual Petri net N

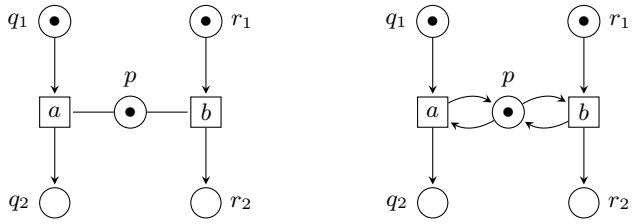

Fig. 3. Example illustrating the effect of read-arcs on partial-order semantics.

is an $s^{\pm}$-run of N. Also, by Lemma 1, every $s^{\pm}$-run is an i-run, and by Lemma 2, every i-run can be made complete.

Let N be a PN, and denote by $Atomic_N$ the set of its $a^{\pm}$-runs, by $Step_N$ the set of its $s^{\pm}$-runs, and by $Interval_N$ the set of its i-runs. Then we have the following relation:

$$Atomic_N \subseteq Step_N \subseteq Interval_N$$

Note that, in general, the subset inclusions are strict, as we have seen in previous examples. The strictness holds even when N does not contain any read arcs. E.g., if N contains three enabled transitions a, b, c, whose presets are all disjoint, then $a^-b^-c^-a^+b^+c^+ \in Step_N \setminus Atomic_N$ and $a^-b^-a^+c^-b^+c^+ \in Interval_N \setminus Step_N$. However, the set of reachable markings remains the same under all three semantics when no read arcs are present.

4 Non-Atomicity and Partial Order Semantics

Consider a PN N with read arcs such as in the left part of Figure 3. It is easy to see that if one replaces a read arc in a net by a pair of arrows forming a loop (see, e.g., the net N' in the right-hand side of Figure 3), then any a-run of N remains an a-run of N', and vice versa, and that both nets have the same reachable markings. However, one of the reasons why read arcs have attracted the attention of the Petri net community is that they change the step semantics of the net. E.g., both nets admit the s-runs $(\{a\}\{b\})$ and $(\{b\}\{a\})$, but N additionally admits the s-run $(\{a, b\})$. The splitting operation provided in the previous section preserves this difference: $split(N')$ admits the two s-runs $(a^-a^+b^-b^+)$ and $(b^-b^+a^-a^+)$, while $split(N)$ admits additional runs, e.g. $(a^-b^-a^+b^+)$.

We first present processes as partial-order semantics for nets under atomic semantics. These definitions are standard [3, 4, 31, 28]. Then, as well as for the sequential semantics, we define the partial-order semantics of a net N under the non-atomic semantics by applying partial-order atomic semantics to $split(N)$. This gives processes where every transition firing is split into two events e^- and e^+. These processes give sufficiently detailed information to understand how a scenario can or cannot be fired under non-atomic semantics.

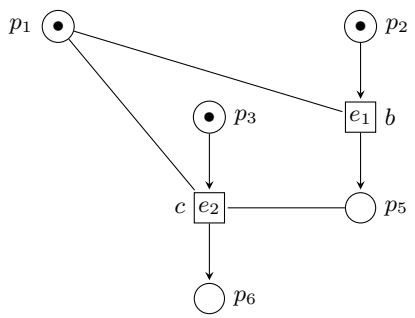

Fig. 4. A process representing the a-run (bc) of the contextual Petri net of Figure 1. Technically, the condition labeled p_1 is coded as $(\bot, p_1)$, the event e_1 (labeled b) is coded as $(\{(\bot, p_2)\}, \{(\bot, p_1)\}, b)$ and e_2 as $(\{(\bot, p_3)\}, \{(\bot, p_1), (e_1, p_5)\}, c)$.

Anyway, in the end, we propose an abstract view of the processes where the e^- and e^+ are abstracted back to a single event e. These abstract processes strictly generalize the processes of the original net under atomic semantics. We characterize the conditions under which an abstract process is feasible under any of the atomic, step or interval semantics.

4.1 Processes Under Atomic Semantics

Processes are a way to represent an execution of a Petri net so that the actions (called events) are not totally ordered like in firing sequences, but only partially ordered by weak (or conditional) and strong (or unconditional) *causality* relations which indicate the dependencies between events due to creation, consumption and reading of tokens.

An execution of a Petri net N is represented as a labeled Petri net where every transition, called *event* and labeled by a transition t of N, stands for an occurrence of t, and every place, called *condition* and labeled by a place p of N, refers to a token produced by an event in place p or to a token of the initial marking. The arcs represent the creation and consumption of tokens.

Figure 4 shows a process representing the a-run (bc) of the contextual Petri net of Figure 1.

Because fresh conditions are created for the tokens created by each event, every condition has either no input arc (if it is an initial condition) or a single input arc, coming from the event that created the token. Symmetrically, each place has no more than one output arc since a token can be consumed by only one event in an execution.

We will define the mapping Π from the a-runs of a safe Petri net to their partial order representation as processes. We use a canonical coding like in [6]. This coding is illustrated in Figure 4.

Each process will be a set E of *events*. Every event e is itself a triple $({}^\bullet e, \underline{e}, \tau(e))$ that codes an occurrence of the transition $\tau(e)$ in the process. ${}^\bullet e$

and $\underline{e}$ are sets of pairs $b \stackrel{\text{def}}{=} ({}^\bullet b, \pi(b))$. Such a pair is called a *condition* and refers to the token that has been created by the event ${}^\bullet b$ in the place $\pi(b)$. We say that the event $e \stackrel{\text{def}}{=} ({}^\bullet e, \underline{e}, \tau(e))$ *consumes* the conditions in ${}^\bullet e$ and *reads* the conditions in $\underline{e}$. It also *creates* the set $\{(e,p) \mid p \in \tau(e)^\bullet\}$ of conditions, which we denote $e^\bullet$. A virtual initial event $\bot$ is used as ${}^\bullet b$ for initial conditions. By convention $\bot^\bullet \stackrel{\text{def}}{=} \{\bot\} \times M_0$.

To summarize the coding of the processes, it is convenient to define a set D_N, such that all the events that appear in the processes of a contextual Petri net N, are elements of D_N.

Definition 9 (D_N). *We define D_N as the smallest set satisfying:*
for all $B_1, B_2 \subseteq \bigcup_{e \in D_N \cup \{\bot\}} e^\bullet$ such that $\pi_{|B_1 \cup B_2}$ is injective,
for all $t \in T$,
$\pi(B_1) = {}^\bullet t \wedge \pi(B_2) = \underline{t} \implies (B_1, B_2, t) \in D_N$.
Notice that this inductive definition is initialized by the fact that $\bot \in D_N \cup \{\bot\}$.

We need a last notion before defining the mapping Π from a-runs to processes: the set of conditions that remain at the end of a set E of events (meaning that they have been created by an event of E, and no event of E has consumed them) is $\uparrow(E) \stackrel{\text{def}}{=} \bigcup_{e \in E \cup \{\bot\}} e^\bullet \setminus \bigcup_{e \in E} {}^\bullet e$. Because N is safe, the restriction of π to $\uparrow(E)$ will be injective when E is a process and $\pi(\uparrow(E))$ will be the marking reached at the end of E.

Definition 10. *The function Π that maps each firing sequence $(t_1 \dots t_n)$ to a process is defined as follows:*

- $\Pi(\epsilon) \stackrel{\text{def}}{=} \emptyset$
- $\Pi((t_1 \dots t_{n+1})) \stackrel{\text{def}}{=} E \cup \{e\}$, *where*
  - $E \stackrel{\text{def}}{=} \Pi((t_1 \dots t_n))$ *and*
  - *the event* $e \stackrel{\text{def}}{=} (\pi^{-1}_{|\uparrow(E)}({}^\bullet t_{n+1}), \pi^{-1}_{|\uparrow(E)}(\underline{t_{n+1}}), t_{n+1})$ *represents the last firing of the sequence.*

Causality. We define the relation $\to$ on the events as: $e \to e' \stackrel{\text{def}}{\iff} e^\bullet \cap {}^\bullet \underline{e'} \neq \emptyset$. The reflexive transitive closure $\to^*$ of $\to$ is called the *unconditional* or *strong causality* relation.

If two events e and f are causally related ($e \to^* f$), then:

- e occurs in every process where f occurs, and
- if a process contains e and f, then e occurs before f.

Because of the read arcs, two events e and f may satisfy the second item even without being in strong causal relation. This happens when e reads a condition that is consumed by f. This phenomenon is captured by the relation $\rightsquigarrow$ defined as $e \rightsquigarrow f \stackrel{\text{def}}{\iff} \underline{e} \cap {}^\bullet f \neq \emptyset$. Combining $\to$ and $\rightsquigarrow$, we get the *conditional* or *weak* causality, denoted $\nearrow$, and defined as $e \nearrow f \stackrel{\text{def}}{\iff} (e \to f) \vee (e \rightsquigarrow f)$.

For every event e, we denote $\lceil e \rceil \stackrel{\text{def}}{=} \{f \in E \mid f \to^* e\}$, and for all set E of events, $\lceil E \rceil \stackrel{\text{def}}{=} \bigcup_{e \in E} \lceil e \rceil$.

Branching processes, conflicts, unfoldings. Each process represents one execution of the net. One often uses the partial order representation to represent also *sets* of executions. This is done simply by superimposing several processes and merging their common prefixes; technically, this operation is nothing but the set union of the processes. The result is called a *branching process*. The most obvious difference between branching processes and processes is that branching processes may contain two distinct events e and e' which have a common precondition (${}^\bullet e \cap {}^\bullet e' \neq \emptyset$). This is called a *conflict* and implies that e and e' never occur together in the same process, since, in a process each condition corresponds to a precise occurrence of a token in a place, created by an event and possibly consumed by another one.

For branching processes of contextual Petri nets, another source of incompatibilities between events needs to be considered: contrary to the strong causality relation $\rightarrow$, the weak causality relation $\nearrow$ may have some cycles. The events involved in such cycle are incompatible because, when an event is added to a process, it is never the predecessor by $\nearrow$ of an older event. This situation arises between the events representing an occurrence of a and an occurrence of b from the initial marking of the net of Figure 1, and corresponds to the fact that the firing of one disables the other in the atomic semantics.

The maximal branching process, obtained by superimposing all the processes of a net N, is called the *unfolding* of N.

4.2 Processes Under Non-Atomic Semantics

In Section 3, we have defined non-atomic sequential semantics of a contextual Petri net N using the construction $split(N)$: every (complete) run σ of $split(N)$ under atomic semantics is interpreted as a run of N under non-atomic semantics.

We can now move very naturally to partial order non-atomic semantics.

Definition 11 ((Complete) split process). *For every (complete) i-run σ of a contextual Petri net N, $\Pi(\sigma)$ is called a (complete)* split process *of N.*

Figure 5 represents a split process of the contextual Petri net N of Figure 1. Weak and strong causality relations in the split process show precisely what are the interleavings of the a^-, a^+, b^-, b^+, c^-, c^+ which make possible a scenario where a, b and c occur.

This representation as split process has the interest of showing a very detailed view of the execution of a contextual Petri net under non-atomic semantics. We propose now a more abstract representation with only one event per transition firing. This representation generalizes the partial order semantics under atomic semantics, in the sense that every process under the atomic semantics is an abstraction of a split process.

The intuition behind the abstraction is the following. We remark that in a complete split process E of N, every event $e^- \in E$ representing an occurrence of a transition t^- of $split(N)$ creates a unique condition b corresponding to a

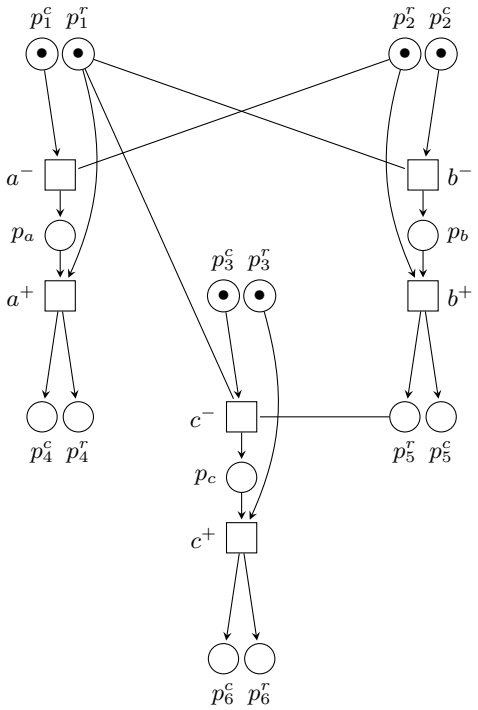

Fig. 5. A process of the splitting of the contextual Petri net N of Figure 1. Transitions a^- and b^- are concurrent, as well as a^+ and c^+. Hence the process represents the 4 i-runs $(a^-b^-b^+c^-c^+a^+)$, $(a^-b^-b^+c^-a^+c^+)$, $(b^-a^-b^+c^-c^+a^+)$ and $(b^-a^-b^+c^-a^+c^+)$. After this process, transition d (split to d^-d^+) becomes fireable.

token in p_t, and this condition is consumed by a unique event e^+ representing the occurrence of t^+.

The abstraction merges e^- and e^+ and deletes b. It also merges the two copies of the created tokens (occurrences of p^c and p^r).

Figure 6 shows the abstraction of the complete split process of Figure 5.

Definition 12 (Abstract processes, *abstr*, α). *We define the abstraction of a complete split process E as:*

$$abstr(E) \stackrel{\text{def}}{=} \{\alpha(e^+) \mid e^+ \in E^+\}$$

where E^+ is the set of events of E representing the occurrence of a transition t^+ of split(N), and α is defined inductively by:

- $\alpha(\bot) \stackrel{\text{def}}{=} \bot$ *and*
- $\alpha(e^+) \stackrel{\text{def}}{=} (\{(\alpha(f^+), p) \mid (f^+, p^c) \in {}^\bullet e^-\},$
  $\{(\alpha(f^+), p) \mid (f^+, p^r) \in \underline{e^-}\},$
  $t)$

  where t is the transition of N such that e^+ represents an occurrence of t^+.

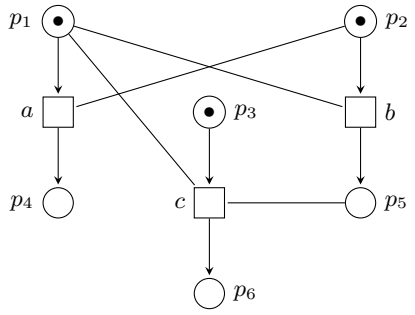

Fig. 6. The abstraction of the split process of Figure 5. This abstract process is feasible only under the interval semantics. With the atomic semantics, a and b cannot fire together. With the step semantics, they can fire 'simultaneously'. But this consumes the token in p_1 which is required to enable c.

We call $abstr(E)$ *an* abstract process *of* N *under interval semantics.*

Notice that the elements in $abstr(E)$ are members of the set D_N (Definition 9); this means they have the same shape as the events that occur in the processes of N under atomic semantics.

What is more: abstractions of processes of $\mathrm{a}^{\pm}$-runs coincide with processes of a-runs.

Theorem 1. *For every a-run* $(t_1 \dots t_n)$*, the process of* $(t_1 \dots t_n)$ *is also the abstraction of the split process of the* $a^{\pm}$*-run* $(t_1^- t_1^+ \dots t_n^- t_n^+)$*:*

$$abstr(\Pi((t_1^- t_1^+ \dots t_n^- t_n^+))) = \Pi((t_1 \dots t_n))\,.$$

Proof. We first remark that the final conditions $\uparrow(abstr(E))$ of the abstraction of a complete split process E are the $(\alpha(f^+), p)$ with $(f^+, p^c) \in \uparrow(E)$ (or equivalently $(f^+, p^r) \in \uparrow(E)$): by definition of the abstraction, the conditions in $abstr(E)$ are the $(\alpha(f^+), p)$ with $p \in t^\bullet$ and $t^+ = \tau(f^+)$; and those that are final in $abstr(E)$ are those that are not consumed by any other $\alpha(e^+)$, which (by definition of $\alpha(e^+)$ and because E is complete) is equivalent to saying that (f^+, p^c) is not consumed by any e^- of E.

Then we prove the theorem by induction on the size of the a-run, using the inductive definitions of Π and α. The occurrences of t^- and t^+ at the end of the $\mathrm{a}^{\pm}$-run are represented in $\Pi((t_1^- t_1^+ \dots t_n^- t_n^+))$ by two events e^- and e^+, and α maps precisely e^+ to the event e of $\Pi((t_1 \dots t_n))$ which represents the firing of t_n: the conditions in ${}^\bullet e^-$ being by definition final conditions of $\Pi((t_1^- t_1^+ \dots t_{n-1}^- t_{n-1}^+))$, the $(\alpha(f^+), p)$ which occur in the definition of $\alpha(e^+)$ are the final conditions of $\Pi((t_1 \dots t_{n-1}))$ which occur in the definition of the event e in $\Pi((t_1 \dots t_n))$. □

A direct consequence of this theorem is that every process of N under the atomic semantics, is also an abstract process of N.

We can also notice that the map α, and by consequence the abstraction *abstr* itself, are injective.

An important thing for the following is how the abstraction preserves the weak and strong causality relations.

Lemma 3. *We have explained when defining the abstraction that every event representing an occurrence of a t^- in a split process of N is followed in the process by a uniquely defined event representing the occurrence of the corresponding t^+. We use the notation e^- and e^+ to identify this correspondence. Moreover this pair of events appears in the abstraction as the event $\alpha(e^+)$, which we denote e. Similarly, for every pair of conditions $(e^+, p^c), (e^+, p^c)$, merged in the abstraction to the condition $(\alpha(e^+), p)$, we denote b^c and b^r and b.*

Using these notations, we have the following properties:

- *for every e, $e^- \to e^+$*
- *for every e and f, $e \to f \iff e^+ \to f^-$.*
- *$\rightsquigarrow$ occurs in a split process only between an e^- and an f^+. It is preserved by the abstraction: $e \rightsquigarrow f \iff e^- \rightsquigarrow f^+$.*

Proof.

- The causality $e^- \to e^+$ simply comes from the condition representing the token created by e^- in place p_t, and consumed by e^+.
- $e \to f$ implies that there exists a condition $b \in e^\bullet \cap {}^\bullet\underline{f}$. If $b \in e^\bullet \cap {}^\bullet f$, then $b^c \in e^{+\bullet} \cap {}^\bullet f^-$; the other case is $b \in e^\bullet \cap \underline{f}$ and implies $b^r \in e^{+\bullet} \cap {}^\bullet f^-$. In both cases, we have $e^{+\bullet} \cap {}^\bullet f^- \neq \emptyset$, and then $e^+ \to f^-$.
- The only case where a condition is read by an event and consumed by another in a split process is when the condition (call it b^r) represents a token in a p^r, and $b^r \in \underline{e^-} \cap {}^\bullet f^+$ for some e and f. This appears in the abstraction as a place $b \in \underline{e} \cap {}^\bullet f$. □

4.3 Characterization of Abstract Processes Feasible with the (Atomic, Step, Interval) Semantics

We propose a direct characterization of the abstract processes without using split processes. We have already remarked that abstract processes are subsets of D_N; now we give the conditions under which a subset of D_N is an abstract process for one or the other semantics.

Theorem 2 (Atomic semantics). *Every set $E \subseteq D_N$ of events is an abstract process of N under the atomic semantics iff*

- *$\lceil E \rceil \subseteq E$ (i.e. E is causally closed),*
- *for every $e, e' \in E$, $e \neq e' \implies {}^\bullet e \cap {}^\bullet e' = \emptyset$ (i.e. E contains no conflict), and*
- *the restriction $\nearrow_{|E}$ of $\nearrow$ to E is acyclic (i.e. its transitive closure ${\nearrow_{|E}}^+$ is irreflexive).*

Proof. Since abstract processes of N coincide with processes under the atomic semantics (see Theorem 1), this amounts to expressing the conditions under which a set $E \subseteq D_N$ of events is a process of N under the atomic semantics, which is a classical result of previous works about partial order semantics of contextual nets [3, 4, 31, 28].

Briefly, the idea is that, by the inductive definition of the events of a process (see Definition 10), all the processes are causally closed and contain no conflicts. Also, when an event e is added to a process (again like in Definition 10), it has no successor by $\nearrow$ in the process. The consequence is that the processes contain no cycle of $\nearrow$.

Conversely, every set E of events that satisfies our three conditions is a process: it suffices to take the events of E in a sequence $(e_1 \dots e_{|E|})$ compatible with the weak causality relation (i.e. such that $e_i \nearrow e_j$ implies $i < j$). This is possible because $\nearrow$ is acyclic. We get that the sequence $\sigma \stackrel{\text{def}}{=} (\pi(e_1) \dots \pi(e_{|E|}))$ is a a-run of N and $\Pi(\sigma) = E$. □

Theorem 3 (Step semantics). *Every set $E \subseteq D_N$ of events is an abstract process of N under the step semantics iff*

- $\lceil E \rceil \subseteq E$,
- *for every* $e, e' \in E$, $e \neq e' \implies {}^\bullet e \cap {}^\bullet e' = \emptyset$, *and*
- *the composition* ${\nearrow_{|E}}^+ \rightarrow_{|E}$ *of relations* ${\nearrow_{|E}}^+$ *and* $\rightarrow_{|E}$ *is irreflexive.*

Proof. Here the principle is the one developed for stratified order structures [8, 10, 12]. Consider a step to be executed after an $\mathrm{s}^{\pm}$-run represented by a split process E'. The events corresponding to all the t^- are added first to the split process (first layer), and then the events corresponding to the t^+ (second layer). In the split process, the (weak and strong) causal dependencies involving the new events go only from the events of E' to the new events and from the first layer to the second layer. After abstraction, the two layers are merged into a single one, among which only weak causal dependencies may exist and may even have cycles (like the events labeled a and b in Figure 6). But no causal dependency exists from the new events to the old ones, which implies that ${\nearrow_{|E}}^+ \rightarrow_{|E}$ is irreflexive on the abstract process E.

Conversely, every set E of events that satisfies our three conditions is an abstract process of N under the step semantics: the fact that ${\nearrow_{|E}}^+ \rightarrow_{|E}$ be irreflexive allows one to partition the events of E into sets E_i such that only $\rightsquigarrow$ dependencies are possible between the events of an E_i, and the other causal dependencies go only from an E_i to an E_j with $i < j$. The sequence of E_i gives a sequence of steps whose split process is mapped to E by the abstraction. □

Theorem 4 (Interval semantics). *Every set $E \subseteq D_N$ of events is an abstract process of N under the interval semantics iff*

- $\lceil E \rceil \subseteq E$,
- *for every* $e, e' \in E$, $e \neq e' \implies {}^\bullet e \cap {}^\bullet e' = \emptyset$, *and*
- $\nearrow_{|E} \rightarrow_{|E}{}^+$ *is acyclic (i.e.* $(\nearrow_{|E} \rightarrow_{|E}{}^+)^+$ *is irreflexive).*

Proof. E is an abstract process of N under the interval semantics iff it is the abstraction of a process E' of $split(N)$ under the atomic semantics. The abstraction mapping *abstr* being injective, it defines a unique candidate for E'. One checks easily that $E' \subseteq D_{split(N)}$. We know that E' is a process of $split(N)$ iff it satisfies the conditions recalled in Theorem 2. It remains to show that they are equivalent to the conditions of the present theorem applied to $E = abstr(E')$. The equivalence of the conditions about the causal pasts ($\lceil E' \rceil \subseteq E'$ iff $\lceil E \rceil \subseteq E$) and about absence of conflicts are straightforward. The more interesting point is the correspondence between the acyclicity conditions: $\nearrow_{|E} \rightarrow_{|E}{}^{+}$ is acyclic iff $\nearrow_{|E'}$ is acyclic. This point derives from the properties of preservation of weak and strong causality by abstraction given in Lemma 3: they give immediately that every cycle for $\nearrow_{|E} \rightarrow_{|E}{}^{+}$ yields a cycle in $\nearrow_{|E'}$. The converse also holds because $\rightsquigarrow$ appears in E only between an e^- and an f^+, as $e^- \rightsquigarrow f^+$. Hence, if this weak causality dependency is concatenated with another one, giving $e^- \rightsquigarrow f^+ \nearrow g$, then the causal dependency $f^+ \nearrow g$ must be strong: $f^+ \rightarrow g$. In the end, this implies that every cycle of the $\nearrow_{|E'}$ relation provides a cycle of the $\nearrow_{|E'} \rightarrow_{|E'}{}^{+}$ relation. By Lemma 3, this cycle yields a cycle for $\nearrow_{|E} \rightarrow_{|E}{}^{+}$ in the abstracted process. □

Summary. As a summary, we just want to confirm, at the level of abstract processes, our intuition that the interval semantics is more permissive than the step semantics, which is in turn more permissive than the atomic semantics. Namely, we compare the conditions about cycles of causality dependencies that appear in the three theorems above. It is true that if $\nearrow_{|E} \rightarrow_{|E}{}^{+}$ has a cycle, the cycle contains at least one strong causality dependency $e \rightarrow f$ and we have $f \nearrow_{|E}{}^{+} \rightarrow_{|E} f$. It is also true that if $f \nearrow_{|E}{}^{+} \rightarrow_{|E} f$, then $f \nearrow_{|E}{}^{+} f$.

When there is no read arc, anyway, they all collapse (at the level of abstract or split processes).

Theorem 5. *For every Petri net N without read arcs (i.e. $\underline{t} = \emptyset$ for all t), the abstract processes under the three semantics coincide.*

Proof. In this case the weak and strong causality relations coincide, and, because the strong causality relation is acyclic by construction of the events, the conditions of acyclicity in the three previous theorems are all automatically verified. □

4.4 The End

One could now wonder what happens if $split(N)$ be itself interpreted under step or interval semantics. The answer is: nothing; and this gives an end to our story! By its structure, $split(N)$ has the property that the three semantics generate the same abstract processes (and the same split processes too, since abstraction is injective).

Theorem 6. *For every contextual occurrence net N, every abstract process of $split(N)$ under the interval semantics is also an abstract process of $split(N)$ under the step and atomic semantics.*

Proof. We show that for every set of events $E \subseteq D_{split(N)}$, if $\nearrow_{|E}$ has a cycle, then $\nearrow_{|E} \rightarrow_{|E}{}^{+}$ has a cycle too. By Lemma 3, $\rightsquigarrow$ appears in E only between an e^- and an f^+, as $e^- \rightsquigarrow f^+$. Hence, if this weak causality dependency is concatenated with another one, giving $e^- \rightsquigarrow f^+ \nearrow g$, then the causal dependency $f^+ \nearrow g$ must be strong: $f^+ \rightarrow g$. In the end, this implies that every cycle of the $\nearrow_{|E}$ relation provides a cycle of the $\nearrow_{|E} \rightarrow_{|E}{}^{+}$ relation. □

5 Discussion

We have shown, within a general framework obtained by an adequate splitting of transitions, how a novel non-atomic firing semantics emerges for contextual nets, and studied the resulting concurrent processes, which provide a deeper insight into complex dynamics of distributed systems.

A key motivation for the research presented in this paper comes from concurrent behaviours as exhibited by systems with a semantics that cannot be captured by sequences of actions (i.e., the atomic semantics). While the step semantics of, e.g., [8, 10, 15, 26] provides an expressive operational semantics, it still does not represent the most general case. It was argued in [30], and analysed in detail in [11], that the most general observational semantics can be represented by the interval semantics. Invariant structures for such a semantics have been proposed in [10, 19], and analysed in detail in [13]. A calculus for temporal reasoning about interval semantics was introduced in [1] where thirteen basic relations between time intervals that are qualitative rather than quantitative (no exact numeric spans are represented) were investigated. These relations and the operations on them form an interval algebra for which several distinct subalgebras of different expressiveness and tractability have since been investigated, e.g., in [23, 21].

An example of recent application of concurrency semantics based on step sequences was the paper [7] which investigated the behaviour of GALS (Globally Asynchronous Locally Synchronous) systems in the context of VLSI circuits. The specification of a system was given in the form of a Petri net N, and the aim was to re-design the system to optimize signal management, by grouping together concurrent events. More precisely, by looking at the concurrent reachability graph of N (i.e., one based on the step semantics), one aims at discovering events that appear in 'bundles', so that they all can be executed in a single clock tick (in effect, pruning the concurrent reachability graph). The resulting bundling is envisaged to reduce signal management, reducing the cost of scheduling and control, and improving system performance. The paper proposes a method that derives a combination of bundles that represents the temporal activities the designer requires. Careful selection of bundles is essential so that the pruned behaviour of the fully asynchronous model still exhibits some characteristics of its parent and is persistent. *Step semantics* and *step persistence*

are hence important features that will guarantee true persistent behaviour for mixed synchronous-asynchronous (GALS) models.

An interesting question from a practical point of view is how to construct abstract processes resp. the abstract unfolding automatically. One possibility requiring little work would be to translate a net N into $split(N)$, then unfold it with the tool CUNF, which efficiently generates the unfolding for the atomic semantics [24]. The resulting unfolding could then be transformed into an abstract unfolding by merging pairs of conditions labeled p^c, p^r and pairs of events labeled t^-, t^+, respectively; the pairs to merge are identified uniquely by definition of the abstraction. A more intriguing question is whether the abstract unfolding can also be generated directly from the net N. A starting point are the results presented in Section 4.3, which characterize the events that belong to the unfolding. However, checking those conditions directly would be inefficient. Existing unfolding tools compute, e.g., a *concurrency relation* that allows to identify possible events more quickly, see, e.g., [2]. Transferring these results does not seem straightforward, and moreover, the issue of how to compute a finite *marking-complete prefix* of the unfolding would require attention. These questions promise to be interesting future work.

We have restricted ourselves to safe nets for technical simplicity and hence readability. However, there are no major obstacles for extending our work to non safe contextual nets. Unfoldings can be defined easily for the very large class of *semi-weighted nets* [20, 3]. Simply, we lose uniqueness of the process representing a firing sequence, which prevents us from using our function Π (Definition 10). More importantly, in split processes of safe nets, for every condition (f^+, p^r) consumed by an event e^+, the corresponding (f^+, p^c) is consumed by e^-. For non safe nets, an e^+ may consume another condition labeled p^r, created by another event, say f'. This would induce 'superfluous' causality between f' and e^+. Taking this into account would make Lemma 3 and the following more tedious.

Future work should also include more general non-atomic semantics, in particular for boolean nets [18].

References

1. J. F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.
2. P. Baldan, A. Bruni, A. Corradini, B. König, C. Rodríguez, and S. Schwoon. Efficient unfolding of contextual petri nets. *Theor. Comput. Sci.*, 449:2–22, 2012.
3. P. Baldan, A. Corradini, and U. Montanari. Contextual Petri nets, asymmetric event structures, and processes. *Information and Computation*, 171(1):1–49, 2001.
4. N. Busi and G. M. Pinna. Non sequential semantics for contextual P/T nets. In *Application and Theory of Petri Nets*, volume 1091 of *Lecture Notes in Computer Science*, pages 113–132. Springer, 1996.
5. S. Christensen and N. D. Hansen. Coloured Petri nets extended with place capacities, test arcs and inhibitor arcs. In *Application and Theory of Petri Nets*, volume 691 of *Lecture Notes in Computer Science*, pages 186–205. Springer, 1993.

6. J. Engelfriet. Branching processes of Petri nets. *Acta Informatica*, 28(6):575–591, 1991.
7. J. Fernandes, M. Koutny, M. Pietkiewicz-Koutny, D. Sokolov, and A. Yakovlev. Step persistence in the design of GALS systems. In *Application and Theory of Petri Nets and Concurrency. Proceedings*, volume 7927 of *Lecture Notes in Computer Science*, pages 190–209. Springer, 2013.
8. H. Gaifman and V. R. Pratt. Partial order models of concurrency and the computation of functions. In *Proceedings, Symposium on Logic in Computer Science*, pages 72–85. IEEE Computer Society, 1987.
9. R. Janicki. Relational structures model of concurrency. *Acta Inf.*, 45(4):279–320, 2008.
10. R. Janicki and M. Koutny. Invariants and paradigms of concurrency theory. In *PARLE 1991*, volume 506 of *Lecture Notes in Computer Science*, pages 59–74. Springer, 1991.
11. R. Janicki and M. Koutny. Structure of concurrency. *Theoretical Computer Science*, 112(1):5–52, 1993.
12. R. Janicki and M. Koutny. Semantics of inhibitor nets. *Inf. Comput.*, 123(1):1–16, 1995.
13. R. Janicki and M. Koutny. Fundamentals of modelling concurrency using discrete relational structures. *Acta Inf.*, 34:367–388, 1997.
14. G. Juhás, R. Lorenz, and S. Mauser. Synchronous + concurrent + sequential = earlier than + not later than. In *Sixth International Conference on Application of Concurrency to System Design (ACSD 2006)*, pages 261–272. IEEE Computer Society, 2006.
15. G. Juhás, R. Lorenz, and S. Mauser. Causal semantics of algebraic Petri nets distinguishing concurrency and synchronicity. *Fundam. Inform.*, 86(3):255–298, 2008.
16. H. C. M. Kleijn and M. Koutny. Process semantics of general inhibitor nets. *Inf. Comput.*, 190(1):18–69, 2004.
17. J. Kleijn and M. Koutny. Causality in extensions of petri nets. *T. Petri Nets and Other Models of Concurrency*, 7:225–254, 2013.
18. J. Kleijn, M. Koutny, M. Pietkiewicz-Koutny, and G. Rozenberg. Step semantics of boolean nets. *Acta Informatica*, 50(1):15–39, 2013.
19. L. Lamport. The mutual exclusion problem: part I - a theory of interprocess communication. *J. ACM*, 33(2):313–326, 1986.
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. D. D. Monica, V. Goranko, A. Montanari, and G. Sciavicco. Expressiveness of the interval logics of allen's relations on the class of all linear orders: Complete classification. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 845–850. IJCAI/AAAI, 2011.
22. U. Montanari and F. Rossi. Contextual nets. *Acta Inf.*, 32(6):545–596, 1995.
23. B. Nebel and H. Bürckert. Reasoning about temporal relations: A maximal tractable subclass of allen's interval algebra. *J. ACM*, 42(1):43–66, 1995.
24. C. Rodríguez. *Verification Based on Unfoldings of Petri Nets with Read Arcs*. PhD thesis, Laboratoire Spécification et Vérification, ENS Cachan, France, Dec. 2013.
25. C. Rodríguez and S. Schwoon. Verification of petri nets with read arcs. In *CONCUR 2012 - Concurrency Theory - 23rd International Conference. Proceedings*, volume 7454 of *Lecture Notes in Computer Science*, pages 471–485. Springer, 2012.
26. W. Vogler. A generalization of trace theory. *RAIRO Informatique théorique et applications*, 25(2):147–156, 1991.

27. W. Vogler. Fairness and partial order semantics. *Inf. Process. Lett.*, 55(1):33–39, 1995.
28. W. Vogler. Partial order semantics and read arcs. *Theoretical Computer Science*, 286(1):33–63, 2002.
29. W. Vogler, A. L. Semenov, and A. Yakovlev. Unfolding and finite prefix for nets with read arcs. In *CONCUR*, volume 1466 of *LNCS*, pages 501–516, 1998.
30. N. Wiener. A contribution to the theory of relative position. *Proc. of the Cambridge Philosophical Society*, 33(2):313–326, 1914.
31. J. Winkowski. Processes of contextual nets and their characteristics. *Fundamenta Informaticae*, 36(1), 1998.
32. J. Winkowski. Reachability in contextual nets. *Fundamenta Informaticae*, 51(1-2):235–250, 2002.