Concurrency and Probability: Removing Confusion, Compositionally

Roberto Bruni  
Dipartimento di Informatica  
Università di Pisa  
Pisa, Italy  
bruni@di.unipi.it

Hernán Melgratti  
Departamento de Computación  
Universidad de Buenos Aires & Conicet  
Buenos Aires, Argentina  
hmelgra@dc.uba.ar

Ugo Montanari  
Dipartimento di Informatica  
Università di Pisa  
Pisa, Italy  
ugo@di.unipi.it

Abstract

Assigning a satisfactory truly concurrent semantics to Petri nets with confusion and distributed decisions is a long standing problem, especially if one wants to resolve decisions by drawing from some probability distribution. Here we propose a general solution based on a recursive, static decomposition of (occurrence) nets in loci of decision, called structural branching cells (s-cells). Each s-cell exposes a set of alternatives, called transactions. Our solution transforms a given Petri net into another net whose transitions are the transactions of the s-cells and whose places are those of the original net, with some auxiliary structure for bookkeeping. The resulting net is confusion-free, and thus conflicting alternatives can be equipped with probabilistic choices, while nonintersecting alternatives are purely concurrent and their probability distributions are independent. The validity of the construction is witnessed by a tight correspondence with the recursively stopped configurations of Abbes and Benveniste. Some advantages of our approach are that: i) s-cells are defined statically and locally in a compositional way; ii) our resulting nets faithfully account for concurrency.

CCS Concepts • Theory of computation → Parallel computing models; Probabilistic computation;

Keywords Petri nets, confusion, dynamic nets, persistent places, OR causality

ACM Reference Format:

1 Introduction

Concurrency theory and practice provide a useful abstraction for the design and use of a variety of systems. Concurrent computations (also processes), as defined in many models, are equivalence classes of execution sequences, called traces, where the order of concurrent (i.e., independent) events is inessential. A key notion in concurrent models is conflict (also known as choices or decisions). Basically, two events are in conflict when they cannot occur in the same execution. The interplay between concurrency and conflicts introduces a phenomenon in which the execution of an event can be influenced by the occurrence of another concurrent (and hence independent) event. Such situation, known as confusion, naturally arises in concurrent and distributed systems and is intrinsic to problems involving mutual exclusion [Smith 1996]. When inter-leaving semantics is considered, the problem is less compelling, however it has been recognised and studied from the beginning of net research [Rozenberg and Engelfriet 1998], and to address it in a general and acceptable way can be considered as a long-standing open problem for concurrency theory.

To illustrate confusion, we rely on Petri nets [Petri 1962; Reisig 2013], which are a basic, well understood model of concurrency. The simplest example of (asymmetric) confusion is the net in Fig. 1a. The net has two traces involving the concurrent events a and b; namely σ1 = a; b and σ2 = b; a. Both traces define the same concurrent execution. Contrasting, σ1 and σ2 are associated with completely different behaviours of the system as far as the resolution of choices is concerned. In fact, the system makes two choices while executing σ1: firstly, it chooses a over d, which enables c; secondly, b is selected over c. Differently, the system makes just one choice in σ2: since c is not enabled, b is executed without any choice; after that, the system chooses a over d. As illustrated by this example, the choices made by two different traces of the same concurrent computation may differ depending on the order in which concurrent events occur.

The fundamental problem behind confusion relates to the description of distributed, global choices. Such problem becomes essential when choices are driven by probabilistic distributions and one wants to assign probabilities to executions, as it is the case with probabilistic, concurrent models. Consider again Fig. 1a and assume that a is chosen over d with probability p_a while b is chosen over c with probability p_b. When treated as independent choices, the

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure1.png}
\caption{Some nets (top) and their event structures (bottom)}
\end{figure}
traces $\sigma_1$ has probability $p_a \cdot p_b$, while $\sigma_2$ has probability $1 \cdot p_a = p_a$.
Hence, two traces of the same concurrent computation, which are deemed equivalent, will be assigned different probabilities.

Different solutions have been proposed in the literature for adding probabilities to Petri nets [Bouillard et al. 2009; Dugan et al. 1984; Eisenträut et al. 2013; Haar 2002; Katoen et al. 1993; Kudlek 2005; Marsan et al. 1984; Molloy 1985]. To avoid confusion, most of them replace nondeterminism with probability only in part, or disregard concurrency, or introduce time dependent stochastic distributions, thus giving up the time and speed independence features typical of truly concurrent models. Confusion-free probabilistic models have been studied in [Varacca et al. 2006], but this class, which subsumes free-choice nets, is usually considered quite restrictive. More generally, the distributability of decisions has been studied, e.g., in [Katoen and Peled 2013; van Glabbeek et al. 2013], but while the results in [van Glabbeek et al. 2013] apply to some restricted classes of nets, the approach in [Katoen and Peled 2013] requires nets to be decorated with agents and produces distributed models with both nondeterminism and probability, where concurrency depends on the scheduling of agents.

A substantial advance has been contributed by Abbes and Benveniste (AB) [Abbes and Benveniste 2005, 2006, 2008]. They consider prime event structures and provide a branching cell decomposition that establishes the order in which choices are resolved (see Section 4.2). Intuitively, the event structure in Fig. 1a has the three branching cells outlined in Fig. 2. First a decision between $a$ and $d$ must be taken (Fig. 2a): if $a$ is executed, then a subsequent branching cell $\{b, c\}$ is enabled (Fig. 2b); otherwise (i.e., if $d$ is chosen) the branching cell $\{b\}$ is enabled (Fig. 2c). In this approach, the trace $\sigma_2 = b; a$ is not admissible, because the branching cell $\{b\}$ does not exist in the original decomposition (Fig. 2a): it appears after the choice of $d$ over $a$. Branching cells are equipped with independent probability distributions and the probability assigned to a concurrent execution is given by the product of the probabilities assigned by its branching cells. Notably, the sum of the probabilities of maximal configurations is 1. Every decomposition of a configuration yields an execution sequence compatible with that configuration. Unfortunately, certain sequences of events, legal w.r.t. the configuration, are not executable according to AB.

**Problem statement** The question addressed in this paper is a foundational one: can concurrency and general probabilistic distributions coexist in Petri nets? If so, under which circumstances? By coexistence we mean that all the following issues must be addressed:

1. **Speed independence**: Truly concurrent semantics usually requires computation to be time-independent and also independent from the relative speed of processes. In this sense, while attaching rates of stochastic distributions to transitions is perfectly fine with interleaving semantics, they are not suited when truly concurrent semantics is considered.

2. **Schedule independence**: Concurrent events must be driven by independent probability distributions.

3. **Probabilistic computation**: Nondeterministic choices must be replaceable by probabilistic choices. This means that whenever two transitions are enabled, the choice to fire one instead of the other is either inessential (because they are concurrent) or is driven by a probability distribution.

4. **Complete concurrency**: It must be possible to establish a bijective correspondence between equivalence classes of firing sequences and a suitable set of concurrent processes.

5. **Sanity check #1**: All firing sequences of the same process carry the same probability, i.e., the probability of a concurrent computation is independent from the order of execution.

6. **Sanity check #2**: The sum of the probabilities assigned to all possible processes must be 1.

In this paper we provide a positive answer for finite occurrence nets: given any such net we show how to define loci of decisions, called structural branching cells (s-cells), and construct another net where independent probability distributions can be assigned to concurrent events. This means that each s-cell can be assigned to a distributed random agent and that any concurrent computation is independent from the scheduling of agents.

**Overview of the approach** Following the rationale behind AB’s approach, a net is transformed into another one that postpones the execution of choices that can be affected by pending decisions. According to this intuition, the net in Fig. 1a is transformed into another one that delays the execution of $b$ until all its potential alternatives (i.e., $c$) are enabled or definitively excluded. In this sense, $b$ should never be executed before the decision between $a$ and $d$ is taken, because $c$ could still be enabled (if $d$ is chosen). As a practical situation, imagine that $a$ and $d$ are the choices of your partner to either come to town ($a$) or go to the sea ($d$) and that you can go to the theatre alone ($b$), which is always an option, or go together with him/her ($c$), which is possible only when he/she is in town and accepts the invitation. Of course you better postpone the decision until you know if your partner is in town or not. This behaviour is faithfully represented, e.g., by the confusion-free net in Fig. 1b, where two variants of $b$ are made explicit: $b_1$ (your partner is in town) and $b_2$ (your partner is not in town). The new place $\neg c$ represents the fact that $c$ will never be enabled. Now, from the concurrency point of view, there is a single process that comprises both $a$ and $b_1$ (with a cause of $b_1$), whose overall probability is the product of the probability of choosing $a$ over $d$ by the probability of choosing $b_1$ over $c$. The other two processes comprise, respectively, $d$ and $b_2$ (with a cause of $b_2$) and $a$ and $c$ (with a cause of $c$). As the net is confusion-free all criteria in the desiderata are met.

The general situation is more involved because: i) there can be several ways to disable the same transition; ii) resolving a choice may require to execute several transitions at once. Consider the net in Fig. 3a: i) $c$ is discarded as soon as $d$ or $f$ fires; and ii) when both $a$ and $e$ are fired we can choose to execute $c$ alone or both $b$ and $g$. Likewise the previous example, we may expect to transform the net as in Fig. 3b. Again, the place $\neg c$ represents the permanent disabling of $c$. This way a probability distribution can drive the choice between $c$ and (the joint execution of) $b_g$, whereas $b$ and $g$ (if enabled) can fire concurrently when $\neg c$ is marked.

A few things are worth remarking: i) a token in $\neg c$ can be needed several times (e.g., to fire $b$ and $g$), hence tokens should be read but...
not consumed from \( \sim c \) (whence the double headed arcs from \( \sim c \) to \( b \) and \( g \), called self-loops); ii) several tokens can appear in the place \( \sim c \) (by firing both \( d \) and \( f \)). These facts have severe repercussions on the concurrent semantics of the net. Suppose the trace \( d; f; b \) is observed. Does \( b \) causally depend on the token generated from \( d \) or from \( f \) (or from both)? Moreover, consider the trace \( d; e; c; b; g \), in which \( b \) takes and releases a token in \( \sim c \). Does \( g \) causally depend on \( b \) (due to such self-loop)? This last question can be solved by replacing self-loops with read arcs [Montanari and Rossi 1995], so that the firing of \( b \) does not alter the content of \( \sim c \) and thus no causal dependency arises between \( b \) and \( g \). Nevertheless, if process semantics or event semantics is considered, then we should explode all possible combinations of causal dependencies, thus introducing a new, undesired kind of nondeterminism. In reality, we should not expect any causal dependency between \( b \) and \( g \), while both have OR dependencies on \( d \) and \( f \).

To account for OR dependencies, we exploit the notion of persistence: tokens in a persistent place have infinite weight and are collective. Namely, once a token reaches a persistent place, it cannot be removed and if two tokens reach the same persistent place they are indistinguishable. Such networks are a variant of ordinary P/T nets and have been studied in [Crazzolara and Winskel 2005]. In the example, we can declare \( \sim c \) to be a persistent place and replace self-loops/read arcs on \( \sim c \) with ordinary outgoing arcs (see Fig. 3c).

Nicely we are able to introduce a process semantics for nets with persistent places that satisfies complete concurrency.

The place \( \sim c \) in the examples above is just used to sketch the general idea: our transformation introduces persistent places like \( \overline{3} \) to express that a token will never appear in the regular place 3.

**Contribution.** In this paper we show how to systematically derive confusion-free nets (with persistence) from any (finite, occurrence) Petri net and equip them with probabilistic distributions and concurrent semantics in the vein of AB’s construction.

Technically, our approach is based on a structurally recursive decomposition of the original net in s-cells. A simple kind of Asperti-Busi’s dynamic nets is used as an intermediate model to structure the coding. While not strictly necessary, the intermediate step emphasises the hierarchical nature of the construction. The second part is a general flattening step independent of our special case. Our definition is purely local (to s-cells), static and compositional, whereas AB’s is dynamic and global (i.e., it requires the entire PES).

Using nets with persistence, we compile the execution strategy of nets with confusion in a statically defined, confusion-free, operational model. The advantage is that the concurrency within a process of the obtained p-net is consistent with execution, i.e., all linearizations of a persistent process are executable.

**Structure of the paper** After fixing notation in Section 2, our solution to the confusion problem consists of the following steps: (i) we define s-cells in a compositional way (Section 3.1); (ii) from s-cells decomposition and the use of dynamic nets, we derive a confusion-free net with persistence (Section 3.2); (iii) we prove the correspondence with AB’s approach (Section 4); (iv) we define a new notion of process that accounts for OR causal dependencies and satisfies complete concurrency (Section 5); and (v) we show how to assign probabilistic distributions to s-cells (Section 6).

## 2 Preliminaries

### 2.1 Notation

We let \( \mathbb{N} \) be the set of natural numbers, \( \mathbb{N}_0 = \mathbb{N} \cup \{0\} \) and \( 2 = \{0, 1\} \). We write \( U^2 \) for the set of functions from \( S \) to \( U \); hence a subset of \( S \) is an element of \( 2^S \), a multiset \( m \) over \( S \) is an element of \( \mathbb{N}^S \), and a bag \( b \) over \( S \) is an element of \( \mathbb{N}_0^S \). By overloading the notation, union, difference and inclusion of sets, multisets and bags are all denoted by the same symbols: \( \cup, \setminus \) and \( \subseteq \), respectively. In the case of bags, the difference \( b \setminus m \) is defined only when the second argument is a multiset, with the convention that \( (b \setminus m)(s) = \infty \) if \( b(s) = \infty \).

Similarly, \( (b \cup b')(s) = \infty \) if \( b(s) < \infty \) or \( b'(s) = \infty \). A set can be seen as a multiset or a bag whose elements have unary multiplicity. Membership is denoted by \( \in \): for a multiset \( m \) (or a bag \( b \)), we write \( s \in m \) for \( m(s) \neq 0 \) (\( b(s) \neq 0 \)). Given a relation \( R \subseteq S \times S \), we let \( R^* \) be its transitive closure and \( R^+ \) be its reflexive and transitive closure.

We say that \( R \) is acyclic if \( \forall s \in S. (s, s) \notin R^* \).

### 2.2 Petri Nets, confusion and free-choiceness

A **net structure** \( N \) (also Petri net) [Petri 1962; Reisig 2013] is a tuple \((P, T, F)\) where: \( P \) is the set of places, \( T \) is the set of transitions, and \( F \subseteq (P \times T) \cup (T \times P) \) is the flow relation. For \( x \in P \cup T \), we denote by \( ^* x = \{ y \mid (y, x) \in F \} \) and \( ^* x = \{ z \mid (x, z) \in F \} \) its **pre-set** and **post-set**, respectively. We assume that \( P \) and \( T \) are disjoint and non-empty and that \( ^* t \) and \( ^* t^* \) are non empty for every \( t \in T \). We write \( t : X \rightarrow Y \) for \( t \in T \) with \( X = ^* t \) and \( Y = t^* \).

\( \text{Figure 3. Running example} \)
A marking is a multiset \( m \in \mathbb{N}^P \). We say that \( p \) is marked at \( m \) if \( p \in m \). We write \((N, m)\) for the net \( N \) marked by \( m \). We write \( m_0 \) for the initial marking of the net, if any.

Graphically, a Petri net is a directed graph whose nodes are the places and transitions and whose set of arcs is \( F \). Places are drawn as circles and transitions as rectangles. The marking \( m \) is represented by inserting \( m(p) \) tokens in each place \( p \in m \) (see Fig. 1).

A transition \( t \) is enabled at the marking \( m \), written \( m \overset{t}{\rightarrow} m' \), if \( *t \subseteq m \). The execution of a transition \( t \) enabled at \( m \), called firing, is written \( m \rightarrow m'' \) with \( m'' = (m \setminus *t) \cup *\). A firing sequence from \( m \) to \( m' \) is a finite sequence of firings \( m = m_0 \overset{t_1}{\rightarrow} \cdots \overset{t_n}{\rightarrow} m_n = m' \), abbreviated to \( m \overset{t_1 \cdots t_n}{\rightarrow} m' \) or just \( m \rightarrow^* m' \). Moreover, it is maximal if no transition is enabled at \( m' \). We write \( m \overset{t_1 \cdots t_n}{\rightarrow} \) if there is \( m' \) such that \( m \overset{t_1 \cdots t_n}{\rightarrow} m' \). We say that \( m' \) is reachable from \( m \) if \( m \rightarrow^* m' \). The set of markings reachable from \( m \) is written \( [m] \). A marked net \((N, m_0)\) has confusion iff there exists a reachable marking \( m \) and transitions \( t, u \) such that:

1. (i) \( t, u \) are in direct conflict if \( *t \cap *u = \emptyset \). A net is called free-choice if for all transitions \( t, u \) we have either \( *t = \emptyset \) or \( t \cap *u = \emptyset \), i.e., if a transition \( t \) is enabled then all its conflicting alternatives are also enabled. Note that free-choice is purely structural. Confusion-freeness considers instead the dynamics of the net. A safe marked net \((N, m_0)\) has confusion iff there exists a reachable marking \( m \) and transitions \( t, u \) such that:

2. (i) \( t, u \) are enabled at \( m \), (ii) \( t \cap *u = \emptyset \setminus u \cap *v \), (iii) \( *v \cap u = \emptyset \setminus t \) (symmetric case); or
3. (i) \( t \), \( u \) are enabled at \( m \), (ii) \( u \) is not enabled at \( m \) but it becomes enabled after the firing of \( t \), and (iii) \( *t \cap *v = \emptyset \) and \( *v \cap *u = \emptyset \) (asymmetric case).

In case 1, \( t \) and \( u \) are concurrently enabled but the firing of one disables an alternative (\( u \)) to the other. In case 2, the firing of \( t \) enables an alternative to \( u \). An example of symmetric confusion is given by \( m = \{2, 3, 8\}, t = b, u = c \) and \( u = g \) in Fig. 3a, while for the asymmetric case take \( m = \{1, 2\}, t = a, v = b \) and \( u = c \) in Fig. 1a. A net is confusion-free when it has no confusion.

### 2.3 Deterministic Nonsequential Processes

A deterministic nonsequential process (or just process) [Goltz and Reisig 1983] represents the equivalence class of all firing sequences of a net that only differ in the order in which concurrent firings are executed. It is given as a mapping \( \pi : D \rightarrow N \) from a deterministic occurrence net \( D \) to \( N \) (preserving pre- and post-sets), where a deterministic occurrence net is such that: (1) the flow relation is acyclic, (2) there are no backward conflicts (\( \forall p \in P, |p| \leq 1 \)), and (3) there are no forward conflicts (\( \forall p \in P, |p^\prime| \leq 1 \)). We let \( D = \{p \mid |p = 0\} \) and \( D^\pi = \{p \mid p^\prime = 0\} \) be the sets of initial and final places of \( D \), respectively (with \( \pi(D) \) the initial marking of \( N \)). When \( N \) is an acyclic safe net, the mapping \( \pi : D \rightarrow N \) is just an injective graph homomorphism: without loss of generality, we name the nodes in \( D \) as their images in \( N \) and let \( \pi \) be the identity. The firing sequences of a processes \( D \) are its maximal firing sequences starting from the marking \( \pi D \). A process of \( N \) is maximal if its firing sequences are maximal in \( N \).

For example, take the net in Fig. 1a. The equivalence class of the firing sequences \( m_0 \overset{a b}{\rightarrow} \) and \( m_0 \overset{b a}{\rightarrow} \) is the maximal process \( D \) with places \( \{1, 2, 3, 4\} \) and transitions \( \{a : 1 \rightarrow 3, b : 2 \rightarrow 4\} \), where \( \pi D = \{1, 2\} \) and \( D^\pi = \{3, 4\} \).

Given an acyclic net we let \( \leq = F^\pi \) be the (reflexive) causality relation and say that two transitions \( t_1 \) and \( t_2 \) are in immediate conflict, written \( t_1 \#_\pi t_2 \) if \( t_1 \neq t_2 \) and \( t_1 \cap t_2 \neq 0 \). The conflict relation \( \# \) is defined by letting \( xy \) if there are \( t_1, t_2 \in T \) such that \( (t_1, x), (t_2, y) \in F^{t_1} \) and \( t_1 \#_\pi t_2 \). Then, a nondeterministic occurrence net (or just occurrence net) is a net \( D = (P, T, F) \) such that: (1) the flow relation is acyclic, (2) there are no backward conflicts (\( \forall p \in P, |p| \leq 1 \)), and (3) there are no self-conflicts (\( \forall t \in T, t \# t \)). The unfolding \( \mathcal{U}(N) \) of a safe Petri net \( N \) is an occurrence net that accounts for all (finite and infinite) runs of \( N \): its transitions model all the possible instances of transitions in \( N \) and its places model all the tokens that can be created in any run. Our construction takes a finite occurrence net as input, which can be, e.g., the (truncated) unfolding of any safe net.

### 2.4 Nets With Persistence

Nets with persistency \((p, nets)\) [Cazzola and Winskel 2005] partition the set of places into regular places \( P \) (ranged by \( p, q, ... \)) and persistent places \( P \) (ranged by \( p, q, ... \)). We use \( s \) to range over \( S = PUP \) and write a \( p \)-net as a tuple \((S, T, F)\). Intuitively, persistent places guarantee some sort of monotonicity about the knowledge of the system. Technically, this is realised by letting states be bags of places \( b \in N_{\text{Bag}}^P \) instead of multisets, with the constraint that \( b(p) \in N \) for any regular place \( p \in P \) and \( b(p) \in [0, \infty) \) for any persistent place \( p \in P \). To guarantee that this property is preserved by firing sequences, we assume that the post-set \( \pi^* \) of a transition \( t \) is the bag such that \( \pi^*(p) = 1 \) if \((t, p) \in F \) (as usual); \( \pi^*(p) = \infty \) if \((t, p) \in F \); and \( \pi^*(s) = 0 \) if \((t, s) \notin F \). We say that a transition \( t \) is persistent if it is attached to persistent places only (i.e. if \( \pi^* \cap \pi \subseteq P \)).

The notions of enabling, firing, firing sequence and reachability extend in the obvious way to \( p \)-nets (when markings are replaced by bags). For example, a transition \( t \) is enabled at the bag \( b \), written \( b \overset{t}{\rightarrow} \), if \( *t \subseteq b \), and the firing of an enabled transitions is written \( b \overset{t}{\rightarrow} b' \) with \( b' = (b \setminus *t) \cup *t \).

A firing sequence is stuttering if it has multiple occurrences of a persistent transition. Since firing a persistent transition \( t \) multiple times is inessential, we consider non-stuttering firing sequences. (Alternatively, we can add a marked regular place \( p_t \) to the preset of each persistent transition \( t \), so \( t \) fires at most once.)

A marked \( p \)-net \((N, b_0)\) is \( 1-\infty \)-safe if each reachable bag \( b \in b_{\infty} \) is such that \( b(p) \in 2 \) for all \( p \in P \) and \( b(p) \in [0, \infty) \) for all \( p \in P \). Note that in \( 1-\infty \)-safe nets the amount of information conveyed by any reachable bag is finite, as each place is associated with one bit of information (marked or unmarked). Graphically, persistent places are represented by circles with double border (and they are either empty or contain a single token). See Fig. 3c for an example.

The notion of confusion extends to \( p \)-nets, by checking direct conflicts w.r.t. regular places only.
The definition above is a domain equation for the set of dynamic p-nets over the set of places $\preceq$: the set $\text{dn}(S)$ is the least fixed point of the equation. The simplest elements in $\text{dn}(S)$ are pairs $(b, b')$ with $b, b' \in \mathbb{N}_0^p$ (with $b(p) \in \mathbb{N}$ for any $p \in P$ and $b(p) \in \{0, \infty\}$ for any $p \in P$). Nets $(T, b)$ are defined recursively; indeed any element $t \models (S, N) \in T$ stands for a transition with preset $S$ and postset $N$, which is another element of $\text{dn}(S)$. An ordinary transition from $b$ to $b'$ has thus the form $(b, (b, b'))$. We write $S \rightarrow N$ for the transition $t \models (S, N)$, $\ast = S$ for its preset, and $\ast = N \in \text{dn}(S)$ for its postset. For $N = (T, b)$ we say that $T$ is the set of top transitions of $N$. All the other transitions are called dynamic.

The firing rule rewrites a dynamic p-net $(T, b)$ to another one. The firing of a transition $t = S \rightarrow (T', b') \in T$ consumes the preset $S$ and releases both the transitions $T' \setminus t$ and the tokens in $b'$. Formally, if $t = S \rightarrow (T', b') \in T$ and $S \subseteq b$ then $(T, b) \xrightarrow{t} (T \cup T', (b \setminus S) \cup b')$.

The notion of $1$-$\infty$-safe dynamic p-net is defined analogously to p-nets by considering the bags of reachable states $(T, b)$.

A sample of a dynamic net is shown in Fig. 4a, whose only dynamic transition, which is activated by $t_3$, is depicted with dashed border. The arrow between $t_3$ and $b$ denotes the fact that $b$ is activated dynamically by the firing of $t_3$. Same $S \rightarrow \{t: 2 \rightarrow 4\}$, $(S, 5)$.

We show that any dynamic p-net can be encoded as a (flat) p-net. Our encoding resembles the one in [Asperti and Busi 2009], but it is simpler because we do not need to handle place creation. Intuitively, we release any transition $t$ immediately but we add a persistent place $p_t$ to its preset, to enable $t$ dynamically ($p_t$ is initially empty iff $t$ is not a top transition). Given a set $T$ of transitions, $b_T$ is the bag such that $b_T(p_t) = \infty$ if $t \in T$ and $b_T(t) = 0$ otherwise.

For $N = (T, b) \in \text{dn}(S)$, we let $\text{T}(N) = T \cup \bigcup_{t \in T} \text{T}(t^*)$ be the set of all (possibly nested) transitions appearing in $N$. From Definition 2.1 it follows that $\text{T}(N)$ is finite and well-defined.

**Definition 2.2** (From dynamic to static). Given $N = (T, b) \in \text{dn}(S)$, the corresponding p-net $\{N\}$ is defined as $\{N\} = (S \cup P_{\text{T}(N)}, \text{T}(N), F, b \cup b_T)$, where

- $P_{\text{T}(N)} = \{p_t | t \in \text{T}(N)\};$
- $F$ is such that for any $t = S \rightarrow (T', b') \in \text{T}(N)$ then $t : \ast \cup p_t \rightarrow b' \cup T'$.

The transitions of $\{N\}$ are those from $N$ (set $\text{T}(N)$). Any place of $N$ is also a place of $\{N\}$ (set $S$). In addition, there is one persistent place $p_t$ for each $t \in \text{T}(N)$ (set $P_{\text{T}(N)}$). The initial marking of $\{N\}$ is that of $N$, i.e., $b_T$ together with the persistent tokens that enable the top transitions of $N$ (i.e., $b_T$). Adding $b_T$ is convenient for the statement in Proposition 2.4, but we could safely remove $P_T \subseteq P_{\text{T}(N)}$ (and $b_T$) from the flat p-net without any consequence.

**Example 2.3.** The dynamic p-net $N$ in Fig. 4a is encoded as the p-net $\{N\}$ in Fig. 4b, which has as many transitions as $N$, but the preset of every transition contains an additional persistent place (depicted in grey) to indicate transition’s availability. All the new places but $p_t$ are marked because the corresponding transitions are initially available. Contrastingly, $p_t$ is unmarked because the corresponding transition becomes available after the firing of $t$.

The following result shows that all computations of a dynamic p-net can be mimicked by the corresponding p-net and vice versa. Hence, the encoding preserves also 1-safety over regular places.

**Proposition 2.4.** Let $N = (T, b) \in \text{dn}(S)$. Then,

1. $N \xrightarrow{t} N'$ implies $\{N\} \xrightarrow{t} \{N'\}$
2. Moreover, $\langle N \rangle \xrightarrow{t} N'$ implies there exists $N''$ such that $N \xrightarrow{t} N''$ and $N' = \{N''\}$.

**Corollary 2.5.** $\{N\}$ is 1-$\infty$-safe iff $N$ is 1-safe.

### 3 From Petri Nets to Dynamic P-Nets

In this section we show that any (finite, acyclic) net $N$ can be associated with a confusion-free, dynamic p-net $\{N\}$ by suitably encoding loci of decision. The mapping builds on the structural cell decomposition introduced below.

#### 3.1 Structural Branching Cells

A structural branching cell represents a statically determined locus of choice, where the firing of some transitions is considered against all the possible conflicting alternatives. To each transition $t$ we assign an s-cell $[t]$. This is achieved by taking the equivalence class of $t$ w.r.t. the equivalence relation $\equiv$ induced by the least preorder $\subseteq$ that includes immediate conflict $\#_0$ and causality $\prec$. For convenience, each s-cell $[t]$ also includes the places in the preset of the transitions in $[t]$, i.e., we let the relation $\text{Pre}^{-1}$ be also included in $\subseteq$, with $\text{Pre} = F \cap (P \times T)$. This way, if $(p, t) \in F$ then $p \subseteq t$ because $p \leq t$ and $t \not\in \subseteq$ because $(p, t) \in \text{Pre}^{-1}$. Formally, we let $\subseteq$ be the transitive closure of the relation $\#_0 \cup \subseteq \cup \text{Pre}^{-1}$. Since $\#_0$ is subsumed by the transitive closure of the relation $\subseteq \cup \text{Pre}^{-1}$, we equivalently set $\subseteq = (\subseteq \cup \text{Pre}^{-1})^*$. Then, we let $\subseteq = \{(x, y) | x \subseteq y \wedge y \subseteq x\}$.

**Definition 3.1 (S-cells).** Let $N = (P, T, F)$ be a finite, nondeterministic occurrence net. The set $\text{bc}(N)$ of s-cells is the set of equivalence classes of $\equiv$, i.e., $\text{bc}(N) = \{[t] \subseteq N | t \in T\}$.

We let $\subseteq$ range over s-cells. By definition it follows that for all $C, C' \in \text{bc}(N)$, if $C \cap C' \neq \emptyset$ then $C = C'$. For any s-cell $C$, we denote by $N_C$ the sub-net of $N$ whose elements are in $C \cup \bigcup_{t \in C} \text{T}(t^*)$. Abusing the notation, we denote by $C$ the set of all the initial places in $N_C$ and by $C^+$ the set of all the final places in $N_C$.

**Definition 3.2 (Transactions).** Let $C \subseteq \text{bc}(N)$. Then, a transaction $\theta$ of $C$, written $\theta : C$, is a maximal (deterministic) process of $N_C$.

Since the set of transitions in a transaction $\theta$ uniquely determines the corresponding process in $N_C$, we write a transaction $\theta$ simply as the set of its transitions.

**Example 3.3.** The net $N$ in Fig. 3a has the three s-cells shown in Fig. 5a, whose transactions are listed in Fig. 5b. For $C_1$ and $C_2$, each transition defines a transaction; $C_3$ has one transaction associated with $c$ and one with (the concurrent firing of) $b$ and $g$.

The following operation $\odot$ is instrumental for the definition of our encoding and stands for the removal of a minimal place of a net and all the elements that causally depend on it. Formally, $N \odot p$ is the least set that satisfies the rules (where $\odot$ has higher precedence over set difference):

$$
\begin{align*}
q &\in N \setminus \{p\} & t \in N \quad \uparrow \leq N \odot p && t \in N \odot p \\
q &\in N \odot p & t \in N \odot p && q \in t^*
\end{align*}
$$

**Example 3.4.** Consider the s-cells in Fig. 5a. The net $N_{C_1} \odot 1$ is empty because every node in $N_{C_1}$ causally depends on 1. Similarly, $N_{C_2} \odot 7$ is empty. The cases for $C_3$ are in Figs. 5c–5e.
3.2 Encoding s-cells as confusion-free dynamic nets

Intuitively, the proposed encoding works by explicitly representing the fact that a place will not be marked in a computation. We denote with \( \overline{p} \) the place that models such “negative” information about the regular place \( p \) and let \( \overline{P} = \{ \overline{p} | p \in P \} \). The encoding uses negative information to recursively decompose s-cells under the assumption that some of their minimal places will stay empty.

**Definition 3.5** (From s-cells to dynamic p-nets). Let \( N = (P, T, F, m) \) be a marked occurrence net. Its dynamic p-net \( [N] \in DN(P \cup \overline{P}) \) is defined as \( [N] = (T_{\text{pos}} \cup T_{\text{neg}}, m) \), where:

\[
\begin{align*}
T_{\text{pos}} &= \{ c \in C | \theta \in \text{bc}(N) \} \\
T_{\text{neg}} &= \{ \overline{p} \in \overline{P} | \overline{p} \in \text{bc}(N) \}
\end{align*}
\]

For any s-cell \( C \) of \( N \) and transaction \( \theta : C \), the encoding generates a transition \( t_{\theta, C} : (c^C \rightarrow (0, \theta^C_2 \cup \overline{C^\theta} \cup \overline{F^\theta})) \) to \( T_{\text{pos}} \) to mimic the atomic execution of \( \theta \). Despite \( \theta \) may be strictly included in \( C^\theta \), we set \( C^\theta \) as the preset of \( t_{\theta, C} \) to ensure that the execution of \( \theta \) only starts when the whole s-cell \( C \) is enabled. Each transition \( t_{\theta, C} \in T_{\text{pos}} \) is a transition of an ordinary Petri net because its postset consists of (i) the final places of \( \theta \) and (ii) the negative versions of the places in \( C^\theta \setminus \theta^C \). A token in \( \overline{p} \in \overline{C^\theta} \setminus \overline{F^\theta} \) represents the fact that the corresponding ordinary place \( p \in C^\theta \) will not be marked because it depends on discarded transitions (not in \( \theta \)).

Negative information is propagated by the transitions in \( T_{\text{neg}} \). For each cell \( C \) and place \( p \in C^\theta \), there exists one dynamic transition \( t_{p, C} : (\overline{p} \rightarrow (T^C \setminus (NC \cup \overline{C}^\theta \cup \overline{F}^\theta))) \) whose preset is just \( \overline{p} \) and whose postset is defined in terms of the subnet \( NC \cap p \). The postset of \( t_{p, C} \) accounts for two effects of propagation: (i) the generation of the negative tokens for all maximal places of \( C \) that causally depend on \( p \), i.e., for the negative places associated with the ones in \( C^\theta \) that are not in \( NC \cup p \); and (ii) the activation of all transitions \( T' \) obtained by encoding \( NC \cap p \), i.e., the behaviour of the branching cell \( C \) after the token in the minimal place \( p \) is excluded. We remark that the bag \( b \) in \( (T', b) \) is always empty, because i) \( NC \) is unmarked and, consequently, \( NC \cup p \) is unmarked, and ii) the initial marking of \([N]\) corresponds to the initial marking of \( N \).

**Example 3.6.** Consider the net \( N \) and its s-cells in Fig. 5a. Then, \([N] = (T, b)\) is defined such that \( b \) is the initial marking of \( N \), i.e., \( b = (1, 2, 7) \), and \( T \) has the transitions shown in Fig. 6.

First consider the s-cell \( C_1 \). \( T_{\text{pos}} \) contains one transition for each transaction in \( C_1 \), namely \( t_a \) for \( \theta_a : C_1 \) and \( t_d \) for \( \theta_d : C_1 \). Both \( t_a \) and \( t_d \) have \( ^{\circ \circ}C_1 = \{1\} \) as preset. By definition of \( T_{\text{pos}} \), both transitions have empty sets of transitions in their postsets. Additionally, \( t_{a, C_1}^\theta \) produces tokens in \( \theta_a^C = \{3\} \) (positive) and \( t_{a, C_1}^\theta \) produces tokens in \( \theta_a^C = \{3\} \) (negative), while \( t_{d, C_1}^\theta \) produces tokens in \( \theta_d^C = \{6\} \) (negative), while \( t_{d, C_1}^\theta \) produces tokens in \( \theta_d^C = \{3\} \). Finally, \( t_1 \in T_{\text{neg}} \) propagates negative tokens for the unique place in \( ^{\circ \circ}C_1 = \{1\}. \) Since \( NC_1 \cap 1 = \emptyset \) and \( \theta_1 \) is the empty net, \( [N_1, \cap 1] = (0, 0) \). Hence, \( t_1 \) produces negative tokens for all maximal places of \( C_1 \), i.e., \( \{3, 6\} \). For the s-cell \( C_2 \) we analogously obtain the transitions \( t_e, t_f, t_g \).
the places in $\mathcal{C}_3 = \{2, 3, 8\}$. The transition $t_3$ has $t_3 = \{3\}$ as its preset and its postset is obtained from $\mathcal{C}_3 \cap 3$, which has two (sub) s-cells $\mathcal{C}_3$ and $\mathcal{C}_g$ (see Fig. 5d). The transitions $t_4$ and $t'_4$ arise from $\mathcal{C}_b$, and $t_5$ and $t'_5$ from $\mathcal{C}_g$. Hence, $t_4 = \{(t_5, t'_5, t_4, t_5', \emptyset)\}$ because $[\mathcal{N}_b \cap 3] = \{(t_5, t'_5, t_4, t_5', \emptyset)\}$ and $\mathcal{C}_g \setminus \mathcal{C}_3 \cap 3 = \{3\}$. Similarly, we derive $t_5$ from $\mathcal{N}_c \cap 2$ and $t_8$ from $\mathcal{N}_c \cap 8$.

We now highlight some features of the encoded net. First, the set of top transitions is free-choice: positive and negative transitions have disjoint presets and the presets of any two positive transitions either coincide (if they arise from the same s-cell) or are disjoint. Recursively, this property holds at any level of nesting. Hence, the only source of potential confusion is due to the combination of top transitions and those activated dynamically, e.g., $t_b$ and either $t_{bg}$ or $t_c$. However, $t_b$ is activated only when either $\mathfrak{3}$ or $\mathfrak{f}$ are marked, while $t_{bg} \cap t_c = \{2, 3, 8\}$. Then, confusion is avoided if $p$ and $\overline{p}$ can never be marked in the same execution (Lemma 3.7).

The net $[N]$ is shown in Fig. 7, where the places $\{1, 2, 7\}$ and the transitions $\{t_1, t_2, t_3, t'_3\}$ are omitted because superseded by the initial marking $\{1, 2, 7\}$.

We remark that the same dynamic transition can be released by the firing of different transitions (e.g., $t_b$ by $t_1$ and $t_3$) and possibly several times in the same computation. Similarly, the same negative information can be generated multiple times. However, this duplication has no effect, since we handle persistent tokens. For instance, the firing sequence $t_4; t'_4; t_3; t_b$ releases two copies of $t_b$ and marks $\mathfrak{3}$ twice. This is essential for reachability, but has interesting consequences w.r.t. causal dependencies (see Section 5).

We now show that the encoding generates confusion-free nets. We start by stating a useful property of the encoding that ensures that an execution cannot generate tokens in both $p$ and $\overline{p}$.

**Lemma 3.7** (Negative and positive tokens are in exclusion). If $[[N]] \rightarrow^* (T, b)$ and $\overline{p} \in b$ then $(T, b) \not\rightarrow^* (T', b')$ implies that $p \notin b'$.

We now observe from Def. 3.5 that for any transition $t \in [N] \in \mathcal{DN}(P \cup \overline{P})$ it holds that either $t \subseteq P$ or $t \subseteq \overline{P}$. The next result says that whenever there exist two transitions $t$ and $t'$ that have different but overlapping presets, at least one of them is disabled by the presence of a negative token in the marking $b$.

**Lemma 3.8** (Nested rules do not collide). Let $[[N]] \in \mathcal{DN}(P \cup \overline{P})$. If $[[N]] \rightarrow^* (T, b)$ then for all $t, t' \in T$ s.t. $t \not= t'$ and $t \cap t' \cap P \neq \emptyset$ it holds that there is $p \in P \cap (t \cup t')$ such that $\overline{p} \in b$.

The main result states that $\mathcal{E}$ generates confusion-free nets.

**Theorem 3.9.** Let $[[N]] \in \mathcal{DN}(P \cup \overline{P})$. If $[[N]] \rightarrow^* (T, b) \rightarrow (T, b)$ then either $t = t'$ or $t \cap t' = \emptyset$.

**Corollary 3.10.** Any net $[[N]] \in \mathcal{DN}(P \cup \overline{P})$ is confusion-free.

Finally, we can combine the encoding $\mathcal{E}$ with $\mathcal{E}_N$ (from Section 2.5) to obtain a (flat) 1-oo-safe, confusion-free, p-net $[[N]]$, that we call the *uniform net* of $N$. By Proposition 2.4 we get that the uniform net $[[N]]$ is also confusion-free by construction.

**Corollary 3.11.** Any p-net $[[N]]$ is confusion-free.

### 4 Static vs Dynamic cell decomposition

As mentioned in the Introduction, Abbes and Benveniste proposed a way to remove confusion by dynamically decomposing prime event structures. In Sections 4.1 and 4.2 we recall the basics of the AB’s approach as introduced in Abbes and Benveniste [2005, 2006, 2008]. Then, we show that there is an operational correspondence between AB decomposition and s-cells introduced in Section 3.1.

#### 4.1 Prime Event Structures

A prime event structure (also PES) [Nielsen et al. 1981; Winskel 1987] is a triple $\mathcal{E} = (E, \leq, \#)$ where: $E$ is the set of events; the causality relation $\leq$ is a partial order on events; the conflict relation $\#$ is a symmetric, irreflexive relation on events such that conflicts are inherited by causality, i.e., $e_1, e_2, e_3 \in E. e_1 \# e_2 \leq e_3 \Rightarrow e_1 \# e_3$.

The PES $\mathcal{E}_N$ associated with a net $N$ can be formalised using category theory as a chain of universal constructions, called coreflections. Hence, for each PES $\mathcal{E}$, there is a standard, unique (up to isomorphism) nondeterministic occurrence net $\mathcal{N}_c$ that yields $\mathcal{E}$ and thus we can freely move from one setting to the other.

Consider the nets in Figs. 1a and 3a. The corresponding PESs are shown below each net. Events are in bijective correspondence with the transitions of the nets. Strict causality is depicted by arrows and immediate conflict by curly lines.

Given an event $e$, its downward closure $[e] = \{e' \in E. e' \leq e\}$ is the set of causes of $e$. As usual, we assume that $[e]$ is finite for any $e$. Given $B \subseteq E$, we say that $B$ is downward closed if $\forall e \in B. e \leq B$ and that $B$ is conflict-free if $\forall e, e' \in E. e \# e' \Rightarrow e \not= e'$. We let the immediate conflict relation $\#_0$ be defined on events by letting $e \#_0 e' $ iff $([e] \times [e']) = \emptyset$. As usual, two events are in immediate conflict if they are in conflict but their causes are compatible.

#### 4.2 Abbes and Benveniste’s Branching Cells

In the following we assume that a (finite) PES $\mathcal{E} = (E, \leq, \#)$ is given. A prefix $B \subseteq E$ is any downward-closed set of events (possibly with conflicts). Any prefix $B$ induces an event structure $\mathcal{E}_B = (B, \leq_B, \#_B)$ where $\leq_B$ and $\#_B$ are the restrictions of $\leq$ and $\#$ to the events in $B$. A stopping prefix is a prefix $B$ that is closed under immediate conflicts, i.e., $\forall e \in B, e' \in E. e \#_B e' \Rightarrow e' \in B$. Intuitively, a stopping prefix is a prefix whose (immediate) choices are all available. It is initial if the only stopping prefix strictly included in $B$ is $\emptyset$. We assume that any $e \in E$ is contained in a finite stopping prefix.

A configuration $\nu \subseteq \mathcal{E}$ is any set of events that is downward closed and conflict-free. Intuitively, a configuration represents (the state reached after executing) a concurrent but deterministic computation of $\mathcal{E}$. Configurations are ordered by inclusion and we denote by $\nu_\mathcal{E}$ the set of finite configurations of $\mathcal{E}$ and by $\Omega_\mathcal{E}$ the set of maximal configurations of $\mathcal{E}$.
The future of a configuration \( v \), written \( E^v \), is the set of events that can be executed after \( v \), i.e., \( E^v = \{ e \in E \mid \forall e' \in v \}(e \in e') \). We write \( E^v \) for the event structure induced by \( E^v \). We assume that any finite configuration enables only finitely many events, i.e., the set of minimal elements in \( E^v \) w.r.t. \( \leq \) is finite for any \( v \in V_E \).

A configuration \( v \) is stopped if there is a stopping prefix \( B \) with \( v \in \Omega_B \) and \( v \) is recursively stopped if there is a finite sequence of configurations \( 0 = v_0 \subseteq \ldots \subseteq v_\ell = v \) such that for any \( t \in [0, n] \) the set \( v_{t+1} \setminus v_t \) is a finite stopped configuration of \( E^{v_t} \) for \( v_t \in E \).

A branching cell is any initial stopping prefix of the future \( E^v \) of a finite recursively stopped configuration \( v \). Intuitively, a branching cell is a minimal subset of events closed under immediate conflict. We remark that branching cells are determined by considering the whole (future of the) event structure \( E \) and they are recursively computed as \( E \) is executed. Remarkably, every maximal configuration has a branching cell decomposition.

**Example 4.1.** Consider the PES \( E_N \) in Fig. 3a and its maximal configuration \( v = [a, e, b, g] \). We show that \( v \) is recursively stopped by exhibiting a branching cell decomposition. The initial stopping prefixes of \( E_N = e^{(a)} \) are shown in Fig. 8a. There are two possibilities for choosing \( v_1 \subseteq v \) and \( v_1 \) recursively stopped: either \( v_1 = [a] \) or \( v_1 = [e] \). When \( v_1 = [a] \), the choices for \( v_2 \) are determined by the stopping prefixes of \( E^{[a]}_N \) (see Fig. 8b) and the only possibility is \( v_2 = [a, e] \). From \( e^{[a, e]}_N \) in Fig. 8c, we take \( v_3 = v \). Note that \( [a, e, b] \) is not recursively stopped because \( [b] \) is not maximal in the stopping prefix of \( E^{[a, e]}_N \) (see Fig. 8c). Finally, note that the branching cells \( E^{[a]}_N \) (Fig. 8b) and \( E^{[a, e]}_N \) (Fig. 8d) correspond to different choices in \( E^{[a]}_N \) and thus have different stopping prefixes.

### 4.3 Relating s-cells and AB’s decomposition

The recursively stopped configurations of a net \( N \) characterise all the allowed executions of \( N \). Hence, we formally link the recursively stopped configurations of \( E_N \) with the computations of the uniformed net \( ([N]) \). For technical convenience, we first show that the recursively stopped configurations of \( E_N \) are in one-to-one correspondence with the computations of the dynamic net \([N]\). Then, the desired correspondence is obtained by using Proposition 2.4 to relate the computations of a dynamic net and its associated p-net.

We rely on the auxiliary map \([\|\|]\) that links transitions in \([N]\) with events in \( E_N \). Specifically, \([\|\|]\) associates each transition \( t \) of \([N]\) with the set \([\|t\|]\) of transitions of \( N \) (also events in \( E_N \)) that are encoded by \( t \). Formally,

\[
[|t|] = \begin{cases} 
ev(\theta) & \text{if } t = t_\theta \in T_{\text{pos}} \\ 0 & \text{if } t \in T_{\text{neg}} \end{cases}
\]

where \( ev(\theta) \) is the set of transitions in \( \theta \).

**Example 4.2.** Consider the net \( N \) in Fig. 5a which is encoded as the dynamic p-net in Fig. 6. The auxiliary mapping \([\|\|]\) is as follows:

\[
\begin{align*}
\|[tbg]| & = [b, g] & \|[tc]| & = [c] & \|[tg]| & = [g] \\
\|[tl]| & = 0 & \text{if } t \in \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9\} \\
\end{align*}
\]

A transition \( t_{\theta, C} \) of \( [N] \) associated with a transaction \( \theta : C \) of \( N \) is mapped to the transitions of \( \theta \). For instance, \( t_a \) is mapped to \([a]\), which is the only transition in \( \theta_a \). Differently, transitions that propagate negative information, i.e., \( t \in \{t_1, t_7, t_2, t_3, t_4, t_5, t_6\} \), are mapped to \( 0 \) because they do not encode any transition of \( N \).

In what follows we write \( M \Rightarrow M' \) for a possibly empty firing sequence \( M \xrightarrow{t_1 \ldots t_n} M' \) such that \(|t_i| \neq 0 \) for all \( i \in [1, n] \). If \(|t_i| = 0 \), we write \( M \xrightarrow{t} M' \) if \( M \Rightarrow M_0 \Rightarrow M_1 \Rightarrow M' \) for some \( M_0, M_1 \). Moreover, we write \( M \xrightarrow{t} M \) if there exist \( M_1, \ldots, M_n \) such that \( M \Rightarrow M_1 \Rightarrow \cdots \Rightarrow M_n \Rightarrow M \).

The following result states that the computations of any dynamic p-net produced by \([\|\|]\) are in one-to-one correspondence with the recursively stopped configurations of Abbes and Benveniste.

**Lemma 4.3.** Let \( N \) be an occurrence net.

1. If \( [N] \xrightarrow{t_1 \ldots t_n} \), then \( v = \bigcup_{i \leq n} [|t_i|] \) is recursively stopped in \( E_N \) and \([|t_i|]\) is a valid decomposition of \( v \).

2. If \( v \) is recursively stopped in \( E_N \), then for any valid decomposition \((v_i)_{1 \leq i \leq n}\) there exists \( [N] \xrightarrow{t_1 \ldots t_n} \) such that \([|t_i|] = v_i \).

**Example 4.4.** Consider the branching cell decomposition for \( v = [a, e, b, g] \in E_N \) discussed in Ex. 4.1. Then, the net \([N] \) in Ex. 3.6 can mimic that decomposition with the following computation:

\[
\begin{align*}
(T, (1, 2, 7)) & \xrightarrow{t_3} (T, (2, 3, 7, 6)) \xrightarrow{t_6} (T, (2, 3, 8, 6, 9)) \\
& \xrightarrow{t_9} (T, (4, 10, 5, 6, 9))
\end{align*}
\]

with \( v_1 = [t_a] = [a], v_2 = [t_6] = [e], \) and \( v_3 = [t_{bg}] = [b, g] \). From Lemma 4.3 and Proposition 2.4 we obtain the next result.

**Theorem 4.5 (Correspondence).** Let \( N \) be an occurrence net.

1. If \( ([N]) \xrightarrow{t_1 \ldots t_n} \), then \( v = \bigcup_{i \leq n} [|t_i|] \) is recursively stopped in \( E_N \) and \( ([|t_i|]\) is a valid decomposition of \( v \).

2. If \( v \) is recursively stopped in \( E_N \), then for any valid decomposition \((v_i)_{1 \leq i \leq n}\) there exists \( ([N]) \xrightarrow{t_1 \ldots t_n} \) such that \([|t_i|] = v_i \).

By (1) above, any computation of \([N]\) corresponds to a recursively stopped configuration of \( E_N \), i.e., a process of \( N \). By (2), every execution of \( N \) that can be decomposed in terms of AB’s branching cells is preserved by \([N]\), because any recursively stopped configuration of \( E_N \) is mimicked by \([N]\).

### 5 Concurrency of the Uniformed Net

In this section we study the amount of concurrency still present in the uniformed net \([N]\). Here, we extend the notion of a process to the case of \( 1-\infty \)-safe p-nets and we show that all the legal firing sequences of a process of the uniformed net \([N]\) are executable.

The notion of deterministic occurrence net is extended to p-nets by slightly changing the definitions of conflict and causal dependency: (i) two transitions are not in conflict when all shared places are persistent, (ii) a persistent place can have more than one immediate cause in its preset, which introduces OR-dependencies.

**Definition 5.1 (Persistent process).** An occurrence p-net \( O = (P \cup P, T, F) \) is an acyclic p-net such that \(|p^*| \leq 1 \) and \(|p| \leq 1 \) for any \( p \in P \) (but not necessarily for those in \( P \)).

A persistent process for \( N \) is an occurrence p-net \( O \) together with a net morphism \( \pi : O \rightarrow N \) that preserves presets and postsets and the distinction between regular and persistent places. Without loss of generality, when \( N \) is acyclic, we assume that \( O \) is a subnet of \( N \) (with the same initial marking) and \( \pi \) is the identity.

In an ordinary occurrence net, the causes of an item \( x \) are all its predecessors. In p-nets, the alternative sets of causes of an item \( x \)
are given by a formula $\Phi(x)$ of the propositional calculus without negation, where the basic propositions are the transitions of the occurrence net. If we represent such a formula as a sum of products, it corresponds to a set of collections, i.e., a set of sets of transitions. Different collections correspond to alternative causal dependencies, while transitions within a collection are all the causes of that alternative and true represents the empty collection. Such a formula $\Phi(x)$ represents a monotone boolean function, which expresses, as a function of the occurrences of past transitions, if $x$ has enough causes. It is known that such formulas, based on positive literals only, have a unique DNF (sum of products) form, given by the set of prime implicants. In fact, every prime implicant is also essential [Wegener 1987]. We define $\Phi(x)$ by well-founded recursion:

$$\Phi(x) = \begin{cases} 
\text{true} & \text{if } x \in P \cup P \land x = \emptyset \\
\lor_{i \in \Theta(x)} (t \land \Phi(t)) & \text{if } x \in P \lor P \land x \neq \emptyset \\
\land_{i \in \Theta(x)} \Phi(i) & \text{if } x \in T 
\end{cases}$$

Ordinary deterministic processes satisfy complete concurrency: each process determines a partial ordering of its transitions, such that the executable sequences of transitions are exactly the linearizations of the partial order. More formally, after executing any firing sequence $\sigma$ of the process, a transition $t$ is enabled if and only if all its predecessors in the partial order (namely its causes) already appear in $\sigma$. In the present setting a similar property holds.

**Definition 5.2** (Legal firing sequence). A sequence of transitions $t_1; \cdots; t_n$ of a persistent process is legal if for all $k \in [1, n]$ we have that $\land_{i=1}^{k-1} t_i$ implies $\Phi(t_k)$.

It is immediate to notice that if the set of persistent places is empty ($P = \emptyset$) then the notion of persistent process is the ordinary one; $\Phi(x)$ is just the conjunction of the causes of $x$ and a sequence is legal if it is a linearization of the process.

**Theorem 5.3** (Complete Concurrency). Let $\sigma = t_1; \cdots; t_n$ with $n \geq 0$ be a, possibly empty, firing sequence of a persistent process, and $t$ a transition not in $\sigma$. The following conditions are all equivalent: (i) $t$ is enabled after $\sigma$; (ii) there is a collection of causes of $t$ which appears in $\sigma$; (iii) $\land_{i=1}^{n} t_i$ implies $\Phi(t)$.

**Corollary 5.4**. Given a persistent process, a sequence is legal iff it is a firing sequence.

**Example 5.5**. Fig. 9 shows a process for the net $[[N]]$ of our running example (see $N$ in Fig. 3a and $[[N]]$ in Fig. 7). The process accounts for the firing of the transitions $d, f, b$ in $N$. Despite they look as concurrent events in $N$, the persistent place $p_b$ introduces some causal dependencies. In fact, we have: $\Phi(t_d) = \Phi(t_f) = \text{true}$, $\Phi(t_b) = t_d \land t_f$ and $\Phi(t_d) = (t_b \land t_d) \lor (t_b \land t_f)$, thus $t_b$ can be fired only after either $t_d$ or $t_f$ (or both).

### 6 Probabilistic Nets

We can now outline our methodology to assign probabilities to the concurrent runs of a Petri net, also in the presence of confusion.

![Figure 8. AB’s branching decomposition](image)

**Figure 8. AB’s branching decomposition (running example)**

![Figure 9. A process for $[[N]]$ (running example)](image)

**Figure 9. A process for $[[N]]$ (running example)**

Given a net $N$, we apply $s$-cell decomposition from Section 3.1, and then we assign probability distributions to the transactions available in each cell $C$ (and recursively to the $s$-cell decomposition of $N$). Let $P_C : \{\emptyset \mid \emptyset : C\} \to [0,1]$ denote the probability distribution function of the $s$-cell $C$ (such that $\sum_{\emptyset \in C} P_C(\emptyset) = 1$). Such probability distributions are defined locally and transferred automatically to the transitions in $P_{\text{pos}}$ of the dynamic p-net $[[N]]$ defined in Section 3, in such a way that $P(t_{\emptyset,C}) = P_C(\emptyset)$. Each negative transitions in $T_{\text{neg}}$ has probability 1 because no choice is associated with it. Since the uniformed net $[[N]]$ has the same transitions of $[N]$, the probability distribution can be carried over $[[N]]$ (thanks to Proposition 2.4).

A simple way to define $P_C$ is by assigning probability distributions to the arcs leaving the same place of the original net. Then, given a transaction $\emptyset : C$, we can set $Q_C(\emptyset)$ be the product of the probability associated with the arcs of $N$ entering the transitions in $\emptyset$. Of course, in general it can happen that $\sum_{\emptyset \in C} Q_C(\emptyset) < 1$, as not all combinations are feasible. However, it is always possible to normalise the quantities of feasible assignments by setting $P_C(\emptyset) = \frac{Q_C(\emptyset)}{\sum_{\emptyset \in C} Q_C(\emptyset)}$ for any transaction $\emptyset : C$.

**Example 6.1**. Suppose that in our running example we assign uniform distributions to all arcs leaving a place. From simple calculation we have $P_C(t_{\emptyset,d}) = P_C(t_{\emptyset,b}) = \frac{1}{2}$ for the first cell, $P_C(t_{\emptyset,d}) = P_C(t_{\emptyset,f}) = \frac{1}{2}$ for the second cell, $P_C(t_{\emptyset,c}) = P_C(t_{\emptyset,b}) = \frac{1}{2}$ for the third cell. The transactions of nested cells are uniquely defined and thus have all probability 1.

Given a firing sequence $t_1; \cdots; t_n$ we can set $P(t_1; \cdots; t_n) = \prod_{i=1}^{n} P(t_i)$. Hence firing sequences that differ in the order in which transitions are fired are assigned the same probability. Thanks to Theorem 5.3, we can consider maximal persistent processes and set $P(O) = \prod_{t \in O} P(t)$. In fact any maximal firing sequence in $O$ includes all transitions of $O$. It follows from Theorem 4.3 that any maximal configuration has a corresponding maximal process (and viceversa) and since Abbes and Benveniste proved that the sum of
the probabilities assigned to maximal configurations is 1, the same holds for maximal persistent processes.

**Example 6.2.** Suppose the distributions assigned in Example 6.1. Then, the persistent process in Fig. 9 has probability: \( P(O) = P(t_2); \)
\( P(t_4) \cdot P(t_5) \cdot P(t_6) \cdot P(t_8) = \frac{1}{4} \cdot \frac{1}{2} \cdot 1 \cdot 1 \cdot 1 \cdot 1 = \frac{1}{4}. \)

7 Conclusion and Future Work

AB’s branching cells are a sort of interpreter (or scheduler) for evaluating \( L \) algebraic formulae and we conjecture it is feasible only if the net is safe and its behavior has some regularity: the same s-cell can be executed several times in a computation but every instance is restarted without tokens left from previous rounds. The causal AND/OR-dependencies share some similarities also with the work on connecters and Petri nets with boundaries [Brunei et al. 2013] that we would like to formalize. We also want to investigate the connection between our s-cell structure and Bayesian networks, so to make forward and backward reasoning techniques available in our setting.

**Acknowledgments**

The idea of structural cells emerged, with a different goal, in collaboration with Lorenzo Galeotti after his MS thesis. We thank Glynn Winskel, Holger Hermanns, Joost-Pieter Katoen for giving us insightful references. The second author has been partially supported by CONICET grant PIP 1120130100148CO. The third author carried on part of the work while attending a Program on Logical Structures in Computation at Simon’s Institute, Berkeley, 2016.

**References**


