COMPUTING SCIENCE

A New Type of Behaviour-Preserving Transition Insertions in Unfolding Prefixes

Victor Khomenko

TECHNICAL REPORT SERIES

No. CS-TR-1189 **February, 2010**

TECHNICAL REPORT SERIES

No. CS-TR-1189 February, 2010

A New Type of Behaviour-Preserving Transition Insertions in Unfolding Prefixes

V. Khomenko

Abstract

In this paper, a new kind of behaviour-preserving transition insertions in Petri nets is proposed, and a method for computing the useful and valid transition insertions using a complete unfolding prefix of the Petri net is developed. Moreover, as several transformations often have to be applied one after the other, the developed theory allows one to avoid (expensive) re-unfolding after each transformation, and instead use local modifications on the existing complete prefix to obtain a complete prefix of the modified net.

Printed and published by the University of Newcastle upon Tyne,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

KHOMENKO, V

A New Type of Behaviour-Preserving Transition Insertions in Unfolding Prefixes
[By] V. Khomenko

Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Science, 2010.

(University of Newcastle upon Tyne, Computing Science, Technical Report Series, No. CS-TR-1189)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE
Computing Science. Technical Report Series. CS-TR-1189

Abstract

In this paper, a new kind of behaviour-preserving transition insertions in Petri nets is proposed, and a method for computing the useful and valid transition insertions using a complete unfolding prefix of the Petri net is developed. Moreover, as several transformations often have to be applied one after the other, the developed theory allows one to avoid (expensive) re-unfolding after each transformation, and instead use local modifications on the existing complete prefix to obtain a complete prefix of the modified net.

About the author

Victor obtained an MSc with distinction in Computer Science, Applied Mathematics and Teaching of Mathematics and Computer Science in 1998 from Kiev Taras Shevchenko University, and PhD in Computing Science in 2003 from University of Newcastle upon Tyne. He is a Program Committee Chair for the International Conference on Application of Concurrency to System Design (ACSD'10). He also organised the Workshop on UnFOlding and partial order techniques (UFO'07) and Workshop on BALSA Re-Synthesis (RESYN'09). From September 2005 Victor is a Royal Academy of Engineering/EPSRC Post-doctoral Research Fellow, working on the Design and Verification of Asynchronous Circuits (DAVAC) project. His **Interest include** model checking of Petri nets, Petri net unfolding techniques, verification and synthesis of self-timed (asynchronous) circuits.

Suggested keywords

TRANSITION INSERTIONS
TRANSFORMATIONS
PETRI NETS
PETRI NET UNFOLDINGS
STGS
ASYNCHRONOUS CIRCUITS

A New Type of Behaviour-Preserving Transition Insertions in Unfolding Prefixes

Victor Khomenko

***Abstract*—In this paper, a new kind of behaviour-preserving transition insertions in Petri nets is proposed, and a method for computing the useful and valid transition insertions using a complete unfolding prefix of the Petri net is developed. Moreover, as several transformations often have to be applied one after the other, the developed theory allows one to avoid (expensive) re-unfolding after each transformation, and instead use local modifications on the existing complete prefix to obtain a complete prefix of the modified net.**

***Index Terms*—Transition insertions, transformations, Petri nets, Petri net unfoldings, STGs, asynchronous circuits.**

I. Introduction

MANY design methods based on Petri nets modify the original specification by behaviour-preserving insertion of new transitions. For example, in the design flow for asynchronous circuits based on Signal Transition Graphs (STGs) [1]–[4] — a class of labelled Petri nets widely used for specifying the behaviour of asynchronous control circuits — transition insertions are used at two different stages: for resolving state encoding conflicts and for logic decomposition of gates. In the discussion below, though a particular motivating application is envisaged, viz. synthesis of asynchronous circuits from STG specifications, the developed techniques and algorithms are not specific to this application domain and suitable for generic Petri nets (e.g. one can envisage applications to action refinement).

This paper focuses primarily on *SB-preserving* transformations, i.e. ones preserving safeness and behaviour (in the sense that the original and the transformed STGs are weakly bisimilar, provided that the newly inserted transitions are considered silent) of the Petri net.

In previous work [5] several types of transition insertions were introduced, and they turned out to be successful in certain applications, e.g. for resolution of state encoding conflicts [6]. However, for other applications, in particular for logic decomposition of asynchronous circuits, those types of transition insertions appear to be insufficient. In this paper, the framework developed in [5] is extended with another type of transition insertions, called *generalised transition insertions (GTIs),* and it is demonstrated how GTIs can be employed in practical applications. In particular:

- It is shown how the validity of a GTI can be checked using a complete unfolding prefix of the Petri net.
- As several insertions often have to be applied one after the other, a theory that allows one to avoid (expensive) re-unfolding the Petri net after each insertion, and instead use local modifications on the existing complete prefix to obtain a complete prefix of the modified net is developed. This has an additional advantage, viz. the produced prefix is similar to the original one, which is useful for visualisation and allows one to transfer some information (like encoding conflicts in asynchronous circuit design) from the original prefix to the modified one, rather than having to re-compute it from scratch.
- A method allowing one to avoid enumerating all GTIs and compute only potentially useful ones is developed; note that unlike the transition insertions proposed in [5], whose number is relatively small, there are exponentially many GTIs in the size of the Petri net, and so limiting their number is very important in practice.

V. Khomenko is a Royal Academy of Engineering/EPSRC Post-Doctoral Research Fellow. He is affiliated with School of Computing Science, Newcastle University, UK. E-mail: Victor.Khomenko@ncl.ac.uk.

This research was supported by the Royal Academy of Engineering/EPSRC post-doctoral research fellowship EP/C53400X/1 (DAVAC) and EPSRC grant EP/G037809/1 (VERDAD).

II. Basic Notions

In this section, basic definitions concerning Petri nets are presented first, and then notions related to net unfoldings and their canonical prefixes are recalled (see also [7]–[11]).

Petri nets

A *net* is a triple $N \stackrel{\text{df}}{=} (P, T, F)$ such that P and T are disjoint sets of respectively *places* and *transitions*, and $F \subseteq (P \times T) \cup (T \times P)$ is a *flow relation*. A *marking* of N is a multiset M of places, i.e. $M : P \to \mathbb{N} = \{0, 1, 2, \ldots\}$. The standard rules about drawing nets are adopted in this paper, viz. places are represented as circles, transitions as boxes, the flow relation by arcs, and markings are shown by placing tokens within circles. As usual, ${}^\bullet z \stackrel{\text{df}}{=} \{y \mid (y, z) \in F\}$ and $z^\bullet \stackrel{\text{df}}{=} \{y \mid (z, y) \in F\}$ denote the *preset* and *postset* of $z \in P \cup T$, and ${}^\bullet Z \stackrel{\text{df}}{=} \bigcup_{z \in Z} {}^\bullet z$ and $Z^\bullet \stackrel{\text{df}}{=} \bigcup_{z \in Z} z^\bullet$, for all $Z \subseteq P \cup T$. It is assumed that ${}^\bullet t \neq \emptyset$, for every $t \in T$. N is *finite* if $P \cup T$ is finite, and *infinite* otherwise.

A *net system* is a tuple $\Sigma \stackrel{\text{df}}{=} (P_\Sigma, T_\Sigma, F_\Sigma, M_\Sigma)$ where $(P_\Sigma, T_\Sigma, F_\Sigma)$ is a finite net and M_Σ is an *initial* marking. A

transition $t \in T_\Sigma$ is *enabled* at a marking M, denoted $M[t\rangle$, if, for every $p \in {}^\bullet t$, $M(p) \geq 1$. Such a transition can be *executed* or *fired*, leading to the marking $M' \stackrel{\text{df}}{=} M - {}^\bullet t + t^\bullet$, where '$-$' and '$+$' stand for the multiset difference and sum respectively. This is denoted by $M[t\rangle M'$. A finite or infinite sequence $\sigma = t_1 t_2 t_3 \ldots$ of transitions is an *execution* from a marking M, denoted $M[\sigma\rangle$, if either σ is an empty sequence or $M[t_1\rangle M'$ and $\sigma' = t_2 t_3 \ldots$ is an execution from M'. Moreover, σ is an execution of Σ if $M_\Sigma[\sigma\rangle$.

The set of *reachable* markings of Σ is the smallest (w.r.t. $\subset$) set containing M_Σ and such that if M is reachable and $M[t\rangle M'$ (for some $t \in T_\Sigma$) then M' is reachable. A transition is *dead* if no reachable marking enables it. A transition is *live* if from every reachable marking M there is an execution containing it. (Note that being live is a stronger property than being non-dead.)

A net system Σ is *k-bounded* if, for every reachable marking M and every place $p \in P_\Sigma$, $M(p) \leq k$, *safe* if it is 1-bounded, and *bounded* if it is k-bounded for some $k \in \mathbb{N}$. For a finite Σ, the set of its reachable markings is finite iff it is bounded.

Branching processes and canonical prefixes

A *finite and complete unfolding prefix* of a Petri net Σ is a finite acyclic net which implicitly represents all the reachable states of Σ together with transitions enabled at those states. Intuitively, it can be obtained through *unfolding* Σ, by successive firings of transitions, under the following assumptions: (a) for each new firing a fresh transition (called an *event*) is generated; (b) for each newly produced token a fresh place (called a *condition*) is generated. The unfolding is infinite whenever Σ has an infinite run; however, if Σ has finitely many reachable states then the unfolding eventually starts to repeat itself and can be truncated (by identifying a set E_{cut} of *cut-off* events beyond which the prefix is not generated) without loss of information, yielding a finite and complete prefix. Due to its structural properties (such as acyclicity), the reachable markings of Σ can be represented using *configurations* of any of its complete prefixes. Intuitively, a configuration is a partial-order execution, i.e. an execution where the order of firing of some of the events (viz. concurrent ones) is not important.

Efficient algorithms exist for building finite and complete prefixes [8], [9], which ensure that the number of non-cut-off events in the resulting prefix never exceeds the number of reachable states of Σ. In fact, complete prefixes are often exponentially smaller than the corresponding state graphs, especially for highly concurrent Petri nets, because they represent concurrency directly rather than by multidimensional 'diamonds' as it is done in state graphs. For example, if the original Petri net consists of 100 transitions which can fire once in parallel, the state graph will be a 100-dimensional hypercube with 2^{100} vertices, whereas the complete prefix will coincide with the net itself. The experimental results in [9] demonstrate that high levels of compression can indeed be achieved in practice.

Formally, two nodes (places or transitions), y and y', of a net $N = (P, T, F)$ are *in conflict*, denoted by $y \# y'$, if there are distinct transitions $t, t' \in T$ such that ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$ and (t, y) and (t', y') are in the reflexive transitive closure of the flow relation F, denoted by $\preceq$. A node y is in *self-conflict* if $y \# y$.

An *occurrence net* is a net $ON \stackrel{\text{df}}{=} (B, E, G)$, where B is the set of *conditions* (places) and E is the set of *events* (transitions), satisfying the following: ON is acyclic (i.e. $\preceq$ is a partial order); for every $b \in B$, $|{}^\bullet b| \leq 1$; no node $y \in B \cup E$ is in self-conflict; and there are finitely many y' such that $y' \prec y$, where $\prec$ denotes the transitive closure of G. $Min(ON)$ will denote the set of minimal (w.r.t. $\prec$) elements of $B \cup E$. The relation $\prec$ is the *causality relation*. Two nodes are *concurrent*, denoted $y \parallel y'$, if neither $y \# y'$ nor $y \preceq y'$ nor $y' \preceq y$.

A *homomorphism* from an occurrence net ON to a net system Σ is a mapping $h : B \cup E \to P_\Sigma \cup T_\Sigma$ such that: $h(B) \subseteq P_\Sigma$ and $h(E) \subseteq T_\Sigma$ (conditions are mapped to places, and events to transitions); for all $e \in E$, the restriction of h to ${}^\bullet e$ is a bijection between ${}^\bullet e$ and ${}^\bullet h(e)$ and the restriction of h to $e^\bullet$ is a bijection between $e^\bullet$ and $h(e)^\bullet$ (transition environments are preserved); the restriction of h to $Min(ON)$ is a bijection between $Min(ON)$ and M_Σ (minimal conditions correspond to the initial marking); and for all $e, f \in E$, if ${}^\bullet e = {}^\bullet f$ and $h(e) = h(f)$ then $e = f$ (there is no redundancy). A *branching process* of Σ is a tuple $\pi_\Sigma \stackrel{\text{df}}{=} (B_{\pi_\Sigma}, E_{\pi_\Sigma}, G_{\pi_\Sigma}, h_{\pi_\Sigma})$ such that $(B_{\pi_\Sigma}, E_{\pi_\Sigma}, G_{\pi_\Sigma})$ is an occurrence net and h_{π_Σ} is a homomorphism from it to Σ. If a node $x \in B_{\pi_\Sigma} \cup E_{\pi_\Sigma}$ is such that $h_{\pi_\Sigma}(x) = y \in P_\Sigma \cup T_\Sigma$, then x is often referred to as being *y-labelled* or as an *instance of* y.

A branching process $\pi'_\Sigma = (B_{\pi'_\Sigma}, E_{\pi'_\Sigma}, G_{\pi'_\Sigma}, h_{\pi'_\Sigma})$ of Σ is a *prefix* of π_Σ, denoted $\pi'_\Sigma \sqsubseteq \pi_\Sigma$, if $(B_{\pi'_\Sigma}, E_{\pi'_\Sigma}, G_{\pi'_\Sigma})$ is a subnet of $(B_{\pi_\Sigma}, E_{\pi_\Sigma}, G_{\pi_\Sigma})$ containing all minimal elements and such that: if $e \in E_{\pi'_\Sigma}$ and $(b, e) \in G_{\pi_\Sigma}$ or $(e, b) \in G_{\pi_\Sigma}$ then $b \in B_{\pi'_\Sigma}$; if $b \in B_{\pi'_\Sigma}$ and $(e, b) \in G_{\pi_\Sigma}$ then $e \in E_{\pi'_\Sigma}$; and $h_{\pi'_\Sigma}$ is the restriction of h_{π_Σ} to $B_{\pi'_\Sigma} \cup E_{\pi'_\Sigma}$. For each Σ there exists a unique (up to isomorphism) maximal (w.r.t. $\sqsubseteq$) branching process Unf_Σ, called the *unfolding* of Σ. To simplify the notation, h_Σ is used instead of h_{π_Σ}; this is justified since $h_{\pi_\Sigma}(x)$ is the same in any branching process of Σ containing x, in particular, one can always refer to Unf_Σ.

An example of a safe net system and two of its branching processes is shown in Fig. 1, where the homomorphism h_Σ is indicated by the labels of the nodes. The branching process in Fig. 1(b) is a prefix of that in Fig. 1(c).

Configurations and cuts: A *configuration* of a branching process π_Σ is a finite set of events $C \subseteq E_{\pi_\Sigma}$ such that there are no $e, f \in C$ for which $e \# f$, and for every $e \in C$, $f \prec e$ implies $f \in C$. For every event $e \in E_{\pi_\Sigma}$, the configuration $[e]_\Sigma \stackrel{\text{df}}{=} \{f \mid f \preceq e\}$ is called the *local configuration* of e.

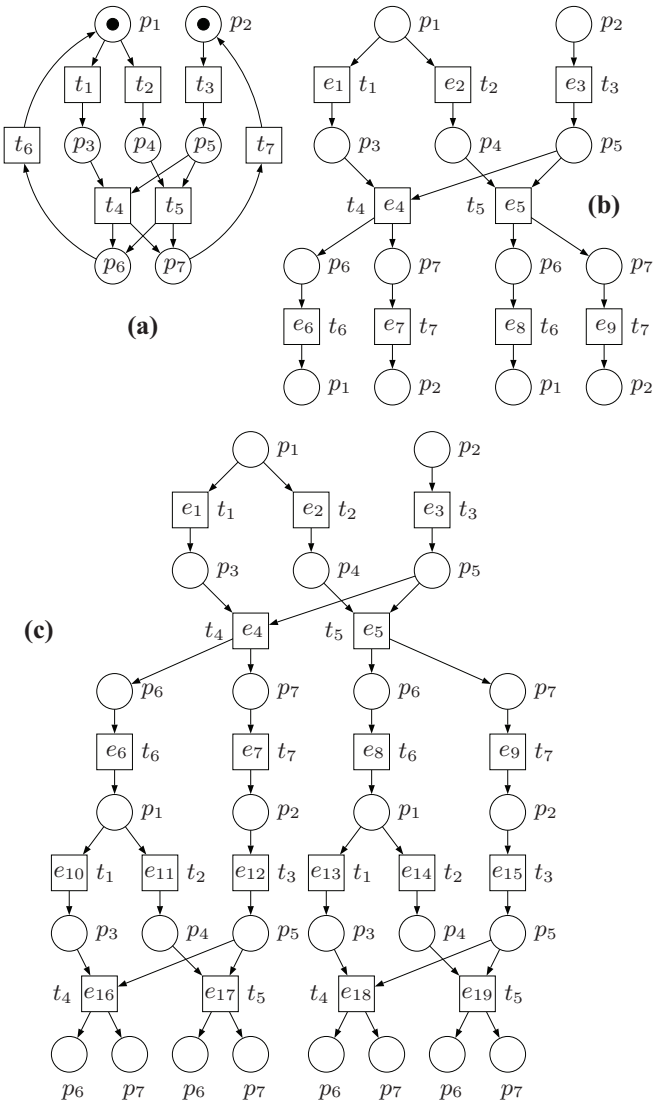


Fig. 1. A net system **(a)** and two of its branching processes **(b,c)**.

Moreover, for a set of events E', $C \oplus E'$ denotes that $C \cup E'$ is a configuration and $C \cap E' = \emptyset$. Such a set is a *suffix* of C, and the configuration $C \oplus E'$ is an *extension* of C. For singleton suffices, $C \oplus e$ is used instead of $C \oplus \{e\}$. For a transition $t \in T_\Sigma$ and a configuration C of π_Σ, $\#_t C$ denotes the number of instances of t in C.

A set of events E' is *downward-closed* if all causal predecessors of the events in E' also belong to E'. Such a set *induces* a unique branching process π_Σ whose events are exactly the events in E', and whose conditions are the conditions incident to the events in E' together with the causally minimal conditions.

A set of conditions B' such that for all distinct $b, b' \in B'$, $b \parallel b'$, is called a *co-set*. A *cut* is a maximal (w.r.t. $\subset$) co-set. Every marking reachable from $Min(ON)$ is a cut. If C is a configuration of π_Σ then the set

$$Cut_\Sigma(C) \stackrel{\text{df}}{=} \Big(Min(ON) \cup C^\bullet\Big) \setminus {}^\bullet C$$

is a cut; moreover, the multiset of places $Mrk_\Sigma(C) \stackrel{\text{df}}{=} h_\Sigma(Cut_\Sigma(C))$ is a reachable marking of Σ, called the *final marking* of C, and $Mrk^p_\Sigma(C)$ denotes the number of tokens $Mrk_\Sigma(C)$ puts on a place p. A marking M of Σ is *represented* in π_Σ if there is a configuration C of π_Σ such that $M = Mrk_\Sigma(C)$. Every marking represented in π_Σ is reachable in the original net system Σ, and every reachable marking of Σ is represented in Unf_Σ.

Note that the notations $[\cdot]_\Sigma$, $Cut_\Sigma(\cdot)$ and $Mrk_\Sigma(\cdot)$ differ from the conventional ones by the presence of subscripts. This is useful in the envisaged settings since the existing unfolding prefix is modified when the original Petri net is modified, i.e. the same event e may belong to the prefixes of two different Petri nets, viz. the original and modified ones, and the subscript is needed to distinguish between them. That Σ rather than π_Σ is used as the subscripts is justified in the view of the fact that the denoted objects are the same in any branching process of Σ containing the necessary events; in particular, one can always refer to Unf_Σ.

Cutting context: There exist different methods of truncating Petri net unfoldings. The differences are related to the kind of information about the original unfolding one wants to preserve in the prefix, as well as to the choice between using either only local configurations (which can improve the running time of an algorithm), or all configurations (which can result in a smaller prefix) for truncating the prefix.

In order to cope with different variants of the technique for truncating unfoldings, the abstract parametric model developed in [10] will be used. The main idea behind it is to speak about configurations of Unf_Σ rather than reachable markings of Σ. In this model, the first parameter determines the information one intends to preserve in the prefix (in the standard case, this is the set of reachable markings). Formally, this information corresponds to the equivalence classes of some equivalence relation $\approx$ on the configurations of Unf_Σ. The other parameters are more technical: they specify the circumstances under which an event can be designated as a cut-off event.

Definition 1 (Cutting context)**.** *A triple*

$$\Theta \stackrel{\text{df}}{=} \Big(\approx, \lhd, \{\mathcal{C}_e\}_{e \in E_{Unf_\Sigma}}\Big)$$

is a cutting context *if:*

1) *$\approx$ is an equivalence relation on the configurations of Unf_Σ.*
2) *$\lhd$, called an* adequate *order, is a strict well-founded partial order on the configurations of Unf_Σ refining $\subset$, i.e. $C' \subset C''$ implies $C' \lhd C''$.*
3) *$\approx$ and $\lhd$ are* preserved by finite extensions*, i.e. for every pair of configurations $C' \approx C''$, and for every finite suffix E'' of C'', there exists a finite suffix E' of C' such that*
   a) *$C' \oplus E' \approx C'' \oplus E''$, and*
   b) *if $C' \lhd C''$ then $C' \oplus E' \lhd C'' \oplus E''$.*

4) For each event e of Unf_Σ, $\mathcal{C}_e$ is a set of configurations of Unf_Σ. $\Diamond$

The main idea behind the adequate order is to specify which configurations will be preserved in the complete prefix; it turns out that all $\lhd$-minimal configurations in each equivalence class of $\approx$ will be preserved. The last parameter is needed to specify for each event e of Unf_Σ the set of configurations $\mathcal{C}_e$ which can be used as the corresponding configurations to declare e a cut-off event. For example, $\mathcal{C}_e$ may contain all configurations of Unf_Σ, or, as usually the case in practice, only the local ones.

For convenience, the domain of $\lhd$ is extended to the events of Unf_Σ as follows: $e \lhd f$ iff $[e]_\Sigma \lhd [f]_\Sigma$. Clearly, $\lhd$ is a well-founded partial order on the set of events. Hence, one can use Noetherian induction (see [12]) for definitions and proofs, i.e. it suffices to define or prove something for an event under the assumption that it has already been defined or proven for all its $\lhd$-predecessors.

In this paper it is assumed that the first component of the cutting context is the equivalence of final markings, defined as $C' \approx_{mar} C''$ iff $Mrk_\Sigma(C') = Mrk_\Sigma(C'')$. The first time the unfolding prefix is built, some fixed cutting context, e.g. the cutting context

$$\Theta_{ERV} \stackrel{\text{df}}{=} \left(\approx_{mar}, \lhd_{erv}, \left\{\mathcal{C}_e\right\}_{e \in E_{Unf_\Sigma}}\right)$$

corresponding to the framework proposed by Esparza, Römer and Vogler in [8], can be used. Here $\lhd_{erv}$ the total adequate order for safe Petri nets proposed in [8] and, for each $e \in E_{Unf_\Sigma}$, $\mathcal{C}_e$ comprises the local configurations of Unf_Σ. However, as transformations are applied to Σ and then mirrored in the corresponding unfolding prefix, *all* components of the cutting context will change (the equivalence relation also changes, albeit for the trivial reason that the modified net has additional places).

Completeness of branching processes: The following definition introduces the notion of completeness of a branching process.

Definition 2 (Completeness)**.** *A branching process π_Σ is* complete w.r.t. a set E_{cut} *of its events if the following hold:*

1) *If C is a configuration of Unf_Σ then there is a configuration C' of π_Σ such that $C' \cap E_{cut} = \emptyset$ and $C \approx C'$.*
2) *If C is a configuration of π_Σ such that $C \cap E_{cut} = \emptyset$, and e is an event of Unf_Σ such that $C \oplus e$ is a configuration of Unf_Σ, then $C \oplus e$ is a configuration of π_Σ.*

A branching process π_Σ is complete *if it is complete w.r.t. some set E_{cut}.* $\Diamond$

Note that π_Σ remains complete after removing all causal successors of the events in E_{cut}, and so, w.l.o.g., one can assume that E_{cut} contains only causally maximal events of π_Σ. Note also that this definition depends only on the equivalence $\approx$, and not on the other components of the cutting context.

Canonical prefix: Now one can define *static* cut-off events, without reference to any unfolding algorithm (hence the term 'static'), together with *feasible* events, which are precisely those events whose causal predecessors are not cut-off events, and as such must be included in the prefix determined by the static cut-off events.

Definition 3 (Feasible and cut-off events)**.** *The set of* feasible *events, denoted by $fsble_\Theta$, and the set of* static cut-off *events, denoted by cut_Θ, are two sets of events of Unf_Σ defined inductively, in the following way:*

1) *An event e is* feasible *if $([e]_\Sigma \setminus \{e\}) \cap cut_\Theta = \emptyset$.*
2) *An event e is a* static cut-off *if it is feasible, and there is a configuration $C \in \mathcal{C}_e$ such that $C \subseteq fsble_\Theta \setminus cut_\Theta$, $C \approx [e]_\Sigma$, and $C \lhd [e]_\Sigma$. In what follows, any C satisfying these conditions will be called a* corresponding *configuration of e.* $\Diamond$

The sets $fsble_\Theta$ and cut_Θ are well-defined sets due to Noetherian induction [10].

Once the feasible events have been defined, the notion of the canonical prefix arises quite naturally, after observing that $fsble_\Theta$ is a downward-closed set of events.

Definition 4 (Canonical prefix)**.** *The branching process $Pref_\Sigma^\Theta$ induced by the set of events $fsble_\Theta$ is called the* canonical prefix *of Unf_Σ.* $\Diamond$

Note that $Pref_\Sigma^\Theta$ is uniquely determined by the cutting context Θ, hence the term 'canonical'.

Several fundamental properties of $Pref_\Sigma^\Theta$ are proven in [10]. In particular, $Pref_\Sigma^\Theta$ is always complete w.r.t. $E_{cut} = cut_\Theta$, and it is finite if $\approx$ has finitely many equivalence classes (in the case of $\approx_{mar}$ the equivalence classes correspond to the reachable markings, i.e. this condition is equivalent to the boundedness of the Petri net) and, for each $e \in E_{Unf_\Sigma}$, $\mathcal{C}_e$ contains all the local configurations of Unf_Σ. Moreover, most unfolding algorithms proposed in literature, in particular those in [8], [9], build this canonical prefix.

III. Motivating application: synthesis of asynchronous circuits from STGs

Asynchronous circuits (ACs) are circuits without clocks. This is a promising type of digital circuits, as they often have lower power consumption and electro-magnetic emission, no problems with clock skew and related subtle issues, and are fundamentally more tolerant of voltage, temperature and manufacturing process variations. The International Technology Roadmap for Semiconductors report on Design [13] predicts that 22% of the designs will be driven by handshake clocking (i.e. asynchronous) in 2013, and this percentage will raise up to 40% in 2020.

Though the listed advantages look rather attractive in the view of the current and anticipated microelectronics design challenges, correct and efficient ACs are notoriously difficult

to synthesise, and so formal methods are essential in their design. This section focuses on an important subclass of ACs, called *speed-independent (SI)* circuits; this model follows the classical Muller's approach [14] and regards each gate as an atomic evaluator of a Boolean function, with a delay element associated with its output. In the SI framework this delay is unbounded, i.e. the circuit must work correctly regardless of its gates' delays, and the wires are assumed to have negligible delays (or, alternatively, wire forks are assumed to be isochronic — in such a case the circuit is often referred to as *quasi-delay-insensitive (QDI)*; for the purposes of this paper, these two models are indistinguishable).

This application domain is introduced by means of the often used example of a simplified VME bus controller shown in Fig. 2 (see also [2, Chapter 2]). Its interface is shown in part (a) of the figure, and the specification of its desired behaviour is given by the STG in part (b) of the figure.

An STG is a labelled Petri nets with labels being of a particular kind. The idea is to associate a set of Boolean variables, referred to as *signals*, with a Petri net to represent the state of the actual digital signals (i.e. wires within a circuit). The Petri net's transitions are then labelled to represent changes in the state of these signals; a transition label either has the form a^+ to indicate a signal a goes from 0 to 1, or a^- to indicate the signal goes from 1 to 0. Thus, the underlying Petri net specifies the causal relationship between signal changes and is intended to capture the behaviour of a system. Clearly, for an STG to correctly represent a circuit one has to ensure that the labels a^+ and a^- are correctly alternated between for each signal a (the so called *consistency condition* [2]).

The following short-hand notation is used to draw STGs: the transitions are simply represented by their labels, and places with only one input and one output transition are contracted (if a contracted place contained a token, it is drawn directly on the resulting arc).

The signals of an STG are partitioned into *input*, *output* and *internal* signals; the output and internal signals are collectively referred to as *local* signals. The inputs are controlled by the environment of the system, and the outputs are controlled by the system itself and are observable by the environment (e.g. they can be inputs of other systems). Internal signals represent some auxiliary activity needed to produce outputs; like outputs, they are controlled by the system, but they are not observable by the environment.

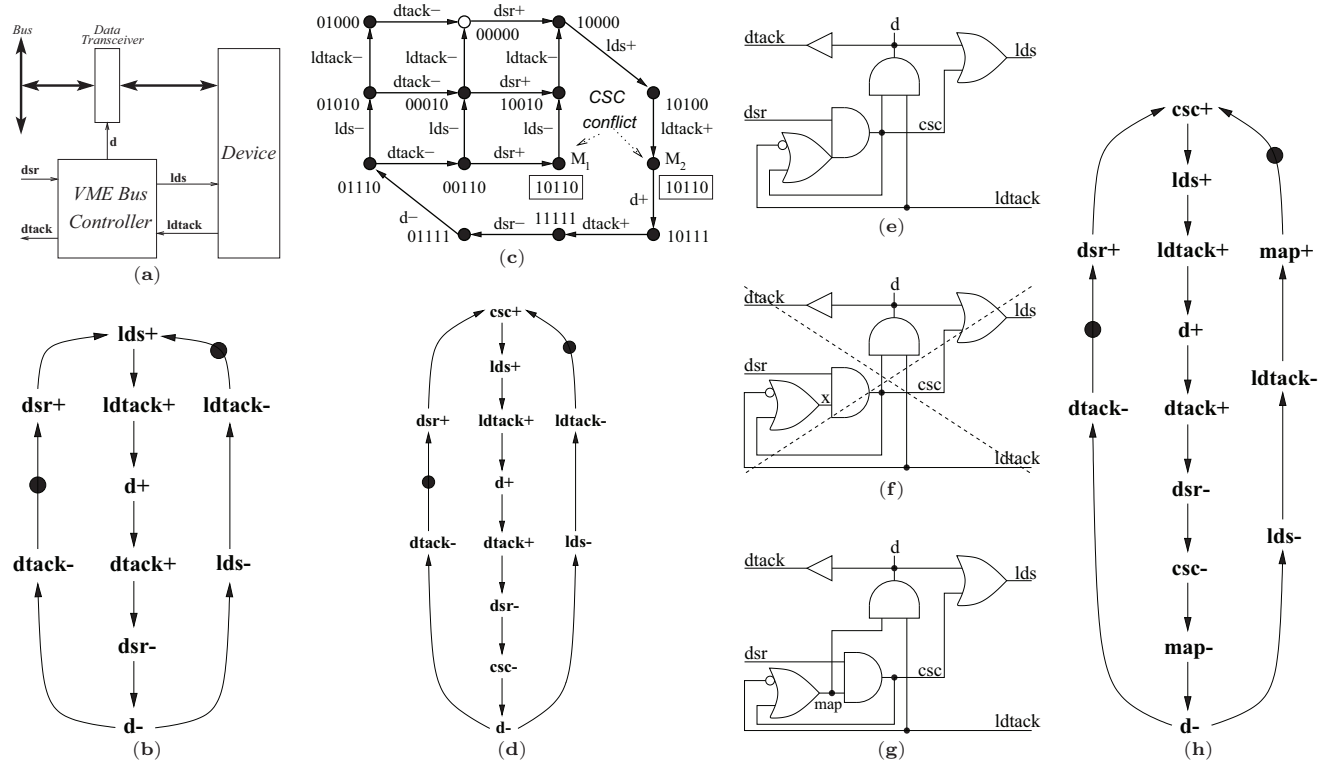
Intuitively, an STG represents a contract between the system and its environment, and is interpreted in the following way. If an input signal transition is enabled, then the environment is allowed (but is not obliged) to send this input, and vice versa, the environment is not allowed to send inputs which are not enabled. If a local transition is enabled then the system is obliged eventually to produce this signal (or it is eventually disabled by another transition), and vice versa, it is not allowed to produce local signals which are not enabled. That is, an STG specifies the behaviour of a system in the sense that the system must provide *all and only* the specified local signals, and that it must allow *at least* the specified inputs (in fact, it could optionally allow more inputs, which means that it could work in a more demanding environment).

The STG in Fig. 2(b) cannot be directly implemented as a circuit because of a so called *Complete State Coding (CSC) conflict*. Consider the state graph of this STG shown in part (c) of the figure. It has two reachable states which have the same values of all the signals but are semantically different: in one of those states the circuit has to raise signal d, whereas in the other one it has to lower signal lds. Since the actual circuit cannot 'see' the marking of the STG, but only the values of the signals, it cannot distinguish between these two states, and so cannot know what to do next. In practice, CSC conflicts are resolved by augmenting the specification with new signals and inserting their transitions at the appropriate positions in the STG. Fig. 2(d) shows how two transitions csc^+ and csc^- of a new internal signal csc can be added to the STG to resolve the CSC conflict. One can see that the values of this signal at the states which were formerly in CSC conflict are different, and so the circuit will be able to distinguish between these states by looking at the value of csc. Intuitively, adding a signal introduces additional memory into the system, which helps it to trace the current state. Note that inserting the two new transitions has to be done in a way that preserves the observable behaviour of the system; hence at this stage of design flow the theory of behaviour-preserving transition insertions has to be applied.

Once the CSC conflicts have been resolved, the so called complex-gate implementation of the STG can be automatically synthesised; for the STG in Fig. 2(d) it is shown in Fig. 2(e). The produced circuit is SI under the atomic gate assumption. Unfortunately, the complexity of individual complex-gates can be high, and a particular gate may not be present in the library of available gates. For example, if the gate library contains only one- and two-input gates, the problem of decomposing the complex-gate implementing csc in Fig. 2(e) into smaller gates arises.

It should be noted that logic decomposition of ACs is considerably more complicated than the corresponding problem in synchronous circuits. In the traditional synchronous case the problem can be formulated on a multi-level combinational Boolean network, which should be mapped to a given gate library by applying the conventional Boolean methods (in particular, algebraic or Boolean division). During this process, the existing algorithms try to minimise some cost function that takes into account the estimated area and/or delay (sometimes other metrics such as power consumption are also used).

When moving to ACs, several levels of complexity are added to the described setup. First of all, the problem can no longer be formulated as a combinational optimisation, and one has to deal with a sequential circuit. Second, it is no



inputs: $dsr, ldtack$ **outputs:** $d, dtack, lds$ **internal:** csc, map, x

Fig. 2. VME bus controller (read cycle): interface (a), the STG specification (b), the state graph showing an encoding conflict (c), the STG with an additional internal signal csc resolving the encoding conflict (d), a complex-gate implementation (e), a naïve logic decomposition exhibiting hazards (f), a correct logic decomposition with multiway acknowledgement (g), the corresponding STG with new internal signal map (h).

longer possible to break up a complex-gate into several smaller ones, together computing the same function, as a composition of several gates can no longer be regarded as atomic and hazards can easily be introduced in this way. (Note that in synchronous circuits such hazards also occur, but are filtered out by the clock.) In fact, some STGs which are implementable by complex-gates may be not implementable in an SI way in some fixed gate library (e.g. one comprising only one- and two-input gates).

In this example, it turns out that the naïve logic decomposition shown in Fig. 2(f) is not SI (even though it would be acceptable in the synchronous framework). Indeed, consider the following sequence of events:

$$⓪\ dsr^+\ csc^+\ lds^+\ ldtack^+\ d^+\ dtack^+\ dsr^-$$
$$csc^-\ ①\ d^-\ dtack^-\ dsr^+\ ②\ csc^+\ d^+$$

At the initial state marked ⓪, $x = 1$ and all the other signals are 0. At the state marked ①, x^- becomes enabled. Since an SI circuit should work regardless of the gate delays, one has to allow that the gates implementing x and lds may be relatively slow, and so the rest of the shown sequence is feasible. When the state marked ② is reached, csc^+ becomes enabled — something not expected in the STG in Fig. 2(d). Though csc is an internal signal which is not observable by the environment, this malfunction can propagate to an observable output by producing an unexpected d^+. Note that the difference between the complex-gate implementation in Fig. 2(e) and this hazardous implementation is that in the latter one has to allow the gate implementing x to have an arbitrary delay, while in the former it is 'inside' a complex-gate which is assumed to be atomic (and thus have no internal delays).

A correct decomposition into two-input gates is shown in Fig. 2(g). Note that in contrast to the described hazardous solution, the new internal signal map is *acknowledged* by two gates (see [15], [16] for the concept of acknowledgement). This illustrates the concept of *multiway acknowledgement*, when different transitions of a signal can be acknowledged by different gates; e.g. in this example map^+ is acknowledged by csc^+ and map^- by d^- (as opposed to the simple case of *local* acknowledgement, where the newly inserted signal is acknowledged only by the gate being decomposed; unlike the synchronous case, where the local acknowledgement is sufficient for decomposition, and the multiway one is used only for optimisation purposes, in the asynchronous case it is quite common that a complex-gate is not decomposable using local acknowledgements only, but is decomposable using multiway ones).

This transformation is rather not obvious on the circuit level (as the behaviour of the environment cannot be deduced from

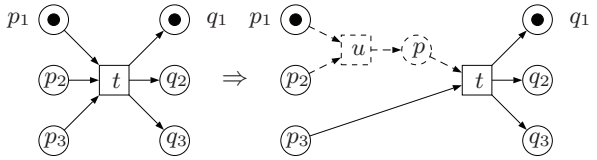
the circuit representation); the corresponding STG shown in Fig. 2(g) is much clearer. It was obtained by augmenting the STG with a new signal map and inserting its transitions map^+ and map^- into the STG in such a way that the implementation of map would be $[map] = \overline{ldtack} \vee csc$. Again, the insertion of new transitions has to be performed in a way that preserves the observable behaviour of the system; hence at this stage of design flow the theory of behaviour-preserving transition insertions has to be applied.

IV. Previous work

In this section the transition insertions presented in [5] are briefly described. The theory developed in [5] allows one to check the validity of such transformations using the canonical unfolding prefix, and to avoid re-unfolding the modified Petri net and instead use local modifications on the existing prefix to obtain the canonical prefix of the modified net.

Sequential pre-insertion

A sequential pre-insertion is essentially a generalised transition splitting, and is defined as follows. Given a transition t and a set of places $S \subseteq {}^\bullet t$, the sequential pre-insertion $S \wr t$ is the transformation inserting a new transition u (with an additional place) 'splitting off' the places in S from t. The picture below illustrates the sequential pre-insertion $\{p_1, p_2\} \wr t$.

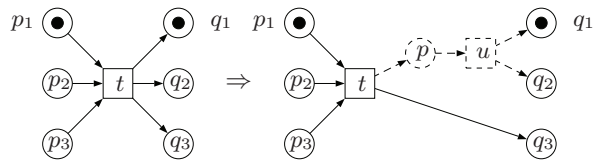


If $S = {}^\bullet t$ then the notation $\wr t$ is used instead of $S \wr t$.

Sequential pre-insertions always preserve safeness and traces (i.e. firing sequences with the newly inserted transition removed). However, in general, the behaviour is not preserved, and so a sequential pre-insertion is not guaranteed to be SB-preserving (in fact, it can introduce deadlocks). Given an unfolding prefix, it is quite easy to check whether a pre-insertion is SB-preserving [5].

Sequential post-insertion

Similarly to sequential pre-insertion, sequential post-insertion is also a generalisation of transition splitting, and is defined as follows. Given a transition t and a set of places $S \subseteq t^\bullet$, the sequential post-insertion $t \wr S$ is the transformation inserting a new transition u (with an additional place) 'splitting off' the places in S from t. The picture below illustrates the sequential post-insertion $t \wr \{q_1, q_2\}$.

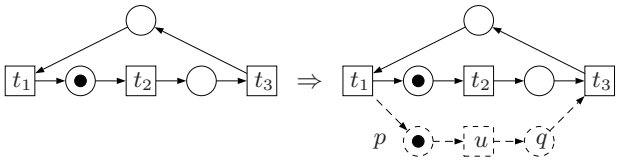


If $S = t^\bullet$ then the notation $t\wr$ is used instead of $t \wr S$.

Sequential post-insertions are always SB-preserving.

Concurrent insertion

Concurrent transition insertion can be advantageous for performance, since the inserted transition can fire in parallel with the existing ones. It is defined as follows. Given two distinct transitions, t' and t'', and an $n \in \{0, 1\}$, the concurrent insertion $t' \overset{n}{\dashv\!\!\rightarrow} t''$ is the transformation inserting a new transition u (with a couple of additional places) between t' and t'', and putting n tokens in the place in its preset. The notation $t' \dashv\!\!\rightarrow t''$ will be used instead of $t' \overset{0}{\dashv\!\!\rightarrow} t''$ and $t' \overset{\bullet}{\dashv\!\!\rightarrow} t''$ instead of $t' \overset{1}{\dashv\!\!\rightarrow} t''$. The picture below illustrates the concurrent insertion $t_1 \overset{\bullet}{\dashv\!\!\rightarrow} t_3$ (note that the token in p is needed to prevent a deadlock).



In general, concurrent insertions preserve neither safeness nor behaviour. In [5], an efficient test whether a concurrent insertion is SB-preserving, working on an unfolding prefix, has been developed.

Equivalent transformations

It can happen that a sequential post-insertion $t \wr S$ yields essentially the same net as a sequential pre-insertion $S' \wr t'$, where $t \in {}^{\bullet\bullet}t'$; in particular, this happens if $S \cup S' \subseteq t^\bullet \cap {}^\bullet t'$ and $|{}^\bullet p| = |p^\bullet| = 1$ for all $p \in S \cup S'$. In such a case there is no reason to distinguish between these two transformations, e.g. one can convert a post-insertion into an equivalent pre-insertion whenever possible. Moreover, since post-insertions are always SB-preserving, there is no need to check the validity of the resulting transformation.

Commutative transformations

A pair of transformations *commute* if the result of their application does not depend on the order they are applied. (Note that a transformation can become ill-defined after applying another transformation, e.g. $t \wr \{p, q\}$ becomes ill-defined after applying $t \wr \{p\}$.) One can observe that:

- a concurrent insertion always commutes with any other transition insertion;
- a sequential pre-insertion and a sequential post-insertion always commute;
- two sequential pre-insertions $S \wr t$ and $S' \wr t'$ commute iff $t \neq t'$ or $S \cap S' = \emptyset$;
- two sequential post-insertions $t \wr S$ and $t' \wr S'$ commute iff $t \neq t'$ or $S \cap S' = \emptyset$.

It is important to note that an SB-preserving transition insertion remains SB-preserving if another commuting SB-preserving transition insertion is applied first. Hence, transformations whose validity has been checked can be cached, and after some transformation has been applied, the non-commuting transformations are removed from the cache and the new transformations that became possible in the modified STG are computed, checked for validity and added to the cache.

A *composite* transition insertion is a transformation defined as the composition of several pairwise commutative transition insertions. Clearly, if a composite transition insertion consists of SB-preserving transition insertions then it is SB-preserving, i.e. one can freely combine SB-preserving transition insertions, as long as they are pairwise commutative.

This property is useful for signal insertions in STGs: typically, several transitions of a new internal signal have to be inserted in order to preserve the consistency of the STG. For example, in Fig. 2(b) a composite transformation comprising two commuting SB-preserving insertions (adding the new transitions csc^+ and csc^-) has been applied to resolve encoding conflicts, yielding the STG in part (d) of the figure, and then another composite transformation comprising two commuting SB-preserving insertions (adding the new transitions map^+ and map^-) has been applied for logic decomposition, yielding the STG in part (h) of the figure.

Motivation for introducing another type of transition insertions

The above transformations turn out to be very useful for certain applications, e.g. resolution of encoding conflicts [6]. However, they appear insufficient for some other applications, in particular the following problem with using them for logic decomposition arises. The STG in Fig. 3 is complex-gate implementable, and the implementation for signal $prbar$ is $[prbar] = (csc2 \wedge req) \vee wsldin \vee csc1$. Suppose this Boolean expression cannot be mapped to any gate in the library, and so has to be decomposed. One of the promising decompositions is to implement the sub-expression $(csc2 \wedge req) \vee csc1$ as an additional internal signal map. The two dashed lines in Fig. 3 show where this sub-expression changes its value, and so the transitions of map should be inserted at these positions. At one of these positions the sequential pre-insertion $\wr prbar^+$ suffices, but at the other position the newly inserted transition has to wait for both req^- and $csc1^-$. In particular, the transformations shown in dotted lines would be very useful due to the local acknowledgement by $prbar$, i.e. the signal being decomposed, which means that the implementations of the other signals are not disturbed by this insertion. Unfortunately, this transformation does not correspond to any of the insertion types considered above.

Generally, one can make the following intuitive argument that the insertion types considered above are insufficient for logic decomposition. Suppose the sizes of presets and postsets

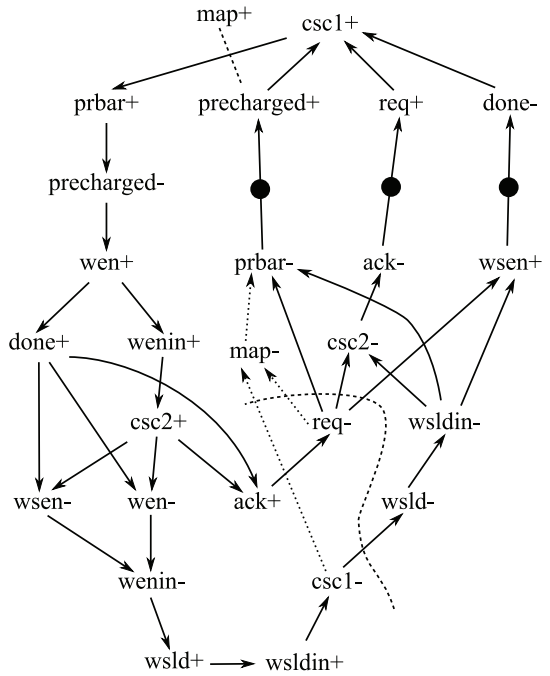


Fig. 3. The `imec-sbuf-ram-write` STG with resolved encoding conflicts. The dashed lines mark the positions where the value of $(csc2 \wedge req) \vee csc1$ changes, and a possible way of inserting the transitions of a new signal map with the implementation $[map] = (csc2 \wedge req) \vee csc1$ is shown.

of transitions in the STG are bounded by a constant (which is a reasonable assumption in practice). Then the total number of sequential pre- and post-insertions is linear in the number of transitions in the STG (as due to the assumption there is just a constant number of such insertions for each transition), and the number of concurrent insertions is quadratic (as for any ordered pair of distinct transitions t', t'' there are only two possible concurrent insertions, $t' \dashv\!\!\rightarrow t''$ and $t' \bullet\!\!\dashv\!\!\rightarrow t''$). On the other hand, Boolean functions can have exponentially many sub-expressions, and so they are likely to change their value in exponentially many positions in the STG. Hence in most cases one will not be able to find a set of transition insertions of the listed types which would cater for a particular sub-expression. Therefore, a new type of transition insertions is required, for which exponential in the size of the STG number of insertions is possible.

V. Generalised transition insertions

Before introducing a new type of transition insertions, a few auxiliary notions are required. The idea of the following definition comes from [17], though the details are different.

Definition 5 (Lock relation)**.** *Two distinct transitions t' and t'' of a Petri net are in the* lock relation, *denoted $t' \circlearrowleft t''$, if in any execution of the Petri net:*

- *the occurrences of t' and t'' alternate; and*
- *the first occurrence of t' precedes the first occurrence of t''.* $\Diamond$

It turns out that given a canonical unfolding prefix, the lock relation can be conservatively approximated.

Definition 6 (Approximated lock relation $\tilde{\circlearrowleft}$)**.** *Let t' and t'' be two distinct transitions of a safe Petri net Σ, and $Tokens(C) \stackrel{\text{df}}{=} \#_{t'}C - \#_{t''}C$ for any configuration C of Unf_Σ. Then for the given canonical prefix $Pref_\Sigma^\Theta$, the relation $\tilde{\circlearrowleft}$ is defined as follows: $t' \,\tilde{\circlearrowleft}\, t''$ iff*

1) *$Tokens([e]_\Sigma) = 1$ for each t'-labelled event e of the prefix; and*
2) *$Tokens([e]_\Sigma) = 0$ for each t''-labelled event e of the prefix; and*
3) *$Tokens([e]_\Sigma) = Tokens(C^e)$ for each cut-off event e of the prefix with the corresponding configuration C^e.*[1] $\Diamond$

Proposition 7 (Conservativeness of $\tilde{\circlearrowleft}$)**.** *For any distinct transitions t' and t'' of a safe Petri net Σ, $t' \,\tilde{\circlearrowleft}\, t''$ implies $t' \circlearrowleft t''$.*

Proof:

Claim 1: There are no two concurrent events e', e'' in Unf_Σ such that $h_\Sigma(e') = h_\Sigma(e'')$.

Follows from the safeness of Σ. □

Claim 2: If $t' \,\tilde{\circlearrowleft}\, t''$ then there are no two concurrent events $e_{t'}, e_{t''}$ in the prefix such that $h_\Sigma(e_{t'}) = t'$ and $h_\Sigma(e_{t''}) = t''$.

For the sake of contradiction, suppose $t' \,\tilde{\circlearrowleft}\, t''$ and there are two concurrent events $e_{t'}, e_{t''}$ in the prefix such that $h_\Sigma(e_{t'}) = t'$ and $h_\Sigma(e_{t''}) = t''$. Let $C \stackrel{\text{df}}{=} [e_{t'}]_\Sigma \cap [e_{t''}]_\Sigma$. Since $e_{t'} \parallel e_{t''}$, any event in $[e_{t'}]_\Sigma \setminus C$ is concurrent to any event in $[e_{t''}]_\Sigma \setminus C$. Hence, due to Claim 1, there are no instances of t'' in $[e_{t'}]_\Sigma \setminus C$ and there are no instances of t' in $[e_{t''}]_\Sigma \setminus C$. Therefore,

$Tokens([e_{t'}]_\Sigma) = Tokens(C) + \#_{t'}([e_{t'}]_\Sigma \setminus C) > Tokens(C)$

and

$Tokens([e_{t''}]_\Sigma) = Tokens(C) - \#_{t''}([e_{t''}]_\Sigma \setminus C) < Tokens(C)$,

i.e.

$$Tokens([e_{t'}]_\Sigma) > Tokens(C) > Tokens([e_{t''}]_\Sigma),$$

and so

$$Tokens([e_{t'}]_\Sigma) \geq Tokens([e_{t''}]_\Sigma) + 2.$$

Hence $Tokens([e_{t'}]_\Sigma) \neq 1$ or $Tokens([e_{t''}]_\Sigma) \neq 0$, contradicting $t' \,\tilde{\circlearrowleft}\, t''$. □

Claim 3: $t' \circlearrowleft t''$ iff $Tokens(C) \in \{0, 1\}$ for each configuration C of the unfolding.

Trivial. □

Any configuration C violating the condition $Tokens(C) \in \{0, 1\}$ will be called *bad*. To the contrary, suppose that $t' \,\tilde{\circlearrowleft}\, t''$ but $t' \not\circlearrowleft t''$. Then by Claim 3 there is a bad configuration $\widehat{C} \oplus \widehat{e}$ (perhaps, containing cut-off and post-cut-off events) such that $h_\Sigma(e) \in \{t', t''\}$. Since the prefix satisfies Def. 6(3), it remains canonical w.r.t. the cutting context where $\approx_{mar}$ is replaced by the following equivalence relation $\approx$:

$$C_1 \approx C_2 \text{ iff } C_1 \approx_{mar} C_2 \wedge Tokens(C_1) = Tokens(C_2).$$

Thus, due to the completeness of the prefix, any minimal w.r.t. $\lhd$ configuration C such that $C \approx \widehat{C}$ is in the prefix and contains no cut-off events (such a minimal configuration exists due to well-foundedness of $\lhd$), and there is an event e in the prefix such that $h_\Sigma(e) = h_\Sigma(\widehat{e}) \in \{t', t''\}$ and $C \oplus e$ is a bad configuration (e may be a cut-off event).

Since all the events in $C \setminus [e]_\Sigma$ are concurrent to e, by Claims 1 and 2 there are no instances of t' and t'' in $C \setminus [e]_\Sigma$, i.e. $Tokens([e]_\Sigma) = Tokens(C \oplus e)$, i.e. $[e]_\Sigma$ is a bad configuration. Since all the events of $[e]_\Sigma$ are in the prefix (e can be a cut-off event) and $[e]_\Sigma$ is bad, Def. 6(1) (if $h_\Sigma(e) = t'$) or Def. 6(2) (if $h_\Sigma(e) = t''$) is violated, contradicting $t' \,\tilde{\circlearrowleft}\, t''$. ■

Note that the inverse of this property is, in general, not true, i.e. it can happen that $t' \circlearrowleft t''$ but $t' \,\tilde{\not\circlearrowleft}\, t''$. However, this is conservative, and the result below shows that in practically important cases $\circlearrowleft$ and $\tilde{\circlearrowleft}$ coincide.

Proposition 8 (Exactness of $\tilde{\circlearrowleft}$ in the live case)**.** *For any distinct transitions t' and t'' of a safe Petri net Σ such that at least one of them is live, $t' \,\tilde{\circlearrowleft}\, t''$ iff $t' \circlearrowleft t''$.*

Proof:

Claim 1: If t' or t'' is live and C' and C'' are configurations of Unf_Σ such that $C' \approx_{mar} C''$ and $Tokens(C') \neq Tokens(C'')$ then $t' \not\circlearrowleft t''$.

Since $C' \approx_{mar} C''$ and due to the liveness of t' or t'', there are finite suffixes E' of C' and E'' of C'' such that $h_\Sigma(E') = h_\Sigma(E'')$ and both E' and E'' contain exactly one event with the label in $\{t', t''\}$. Hence, due to $Tokens(C') \neq Tokens(C'')$, either $Tokens(C' \oplus E') \notin \{0, 1\}$ or $Tokens(C'' \oplus E'') \notin \{0, 1\}$, and so $t' \not\circlearrowleft t''$ by Claim 3 in the proof of Prop. 7. □

In the view of Prop. 7, it remains to show that if t' or t'' is live then $t' \circlearrowleft t''$ implies $t' \,\tilde{\circlearrowleft}\, t''$, i.e. if $t' \,\tilde{\not\circlearrowleft}\, t''$ then $t' \not\circlearrowleft t''$.

If Def. 6(1) or Def. 6(2) does not hold then $t' \not\circlearrowleft t''$ by Claim 3 in the proof of Prop. 7. Suppose Def. 6(3) does not hold. Then the prefix contains a cut-off event e with the corresponding configuration C^e such that $Tokens([e]_\Sigma) \neq Tokens(C^e)$, i.e. $[e]_\Sigma$ and C^e satisfy the conditions of Claim 1, and so $t' \not\circlearrowleft t''$. ■

The central notion of this paper, viz. *generalised transition insertions,* is now introduced (cf. Fig. 4).

Definition 9 (Generalised transition insertion)**.** *Let $\Sigma = (P_\Sigma, T_\Sigma, F_\Sigma, M_\Sigma)$ be a Petri net and $S, D \subseteq T_\Sigma$ be two disjoint non-empty sets of transitions called respectively* sources *and* destinations*, such that for every source $s \in S$ either $s \circlearrowleft d$*

[1] In general, a cut-off event e can have multiple corresponding configurations, but only one (any) of them is stored with e when the prefix is built.

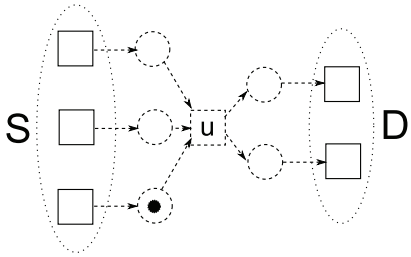


Fig. 4. Generalised transition insertion: the added nodes and arcs are shown in dotted lines; some of the places in ${}^\bullet u$ can be initially marked, depending on the lock relation between the transitions in S and D.

for all $d \in D$ (in which case s is called unmarked*) or $d \circlearrowright s$ for all $d \in D$ (in which case s is called* marked*). Then the* generalised transition insertion (GTI) $S \rightarrowtail\!\!\!\!\!\!\rightarrow D$ *is the transformation yielding the Petri net $\Sigma^u = (P_{\Sigma^u}, T_{\Sigma^u}, F_{\Sigma^u}, M_{\Sigma^u})$, where*

- $P_{\Sigma^u} \stackrel{\text{df}}{=} P_\Sigma \cup \{p_s \mid s \in S\} \cup \{p_d \mid d \in D\}$*, where p_s and p_d are new places;*
- $T_{\Sigma^u} \stackrel{\text{df}}{=} T_\Sigma \cup \{u\}$*, where u is a new transition;*
- $F_{\Sigma^u} \stackrel{\text{df}}{=} F_\Sigma \cup \{(s, p_s) \mid s \in S\} \cup \{(p_s, u) \mid s \in S\} \cup \{(u, p_d) \mid d \in D\} \cup \{(p_d, d) \mid d \in D\}$*;*
- $M_{\Sigma^u} = M_\Sigma \cup \{p_s \mid s \in S$ *and s is marked*$\}$.

A GTI is called conservative *if the relation $\tilde{\circlearrowright}$ is used instead of $\circlearrowright$ in this definition.*[2] $\Diamond$

Note that since by Prop. 7 $\tilde{\circlearrowright}$ is a stricter relation than $\circlearrowright$, conservative GTIs form a subset of all GTIs.

Proposition 10 (Validity of GTIs)**.** *Generalised transition insertions are SB-preserving.*

Proof:

Claim 1: Let Σ be a Petri net, a and b be two of its transitions, and $a \xrightarrow{\bigcirc} b$ and $a \xrightarrow{\odot} b$ be place insertions in Σ that add a new place p (which is initially marked for the latter transformation) and the arcs (a, p) and (p, b). Then for any distinct transitions t' and t'' of Σ, if $t' \circlearrowright t''$ then the place insertions $t' \xrightarrow{\bigcirc} t''$ and $t'' \xrightarrow{\odot} t'$ are SB-preserving.

Trivial. □

Let $S \rightarrowtail\!\!\!\!\!\!\rightarrow D$ be a GTI changing a Petri net Σ into Σ^u. Consider the auxiliary Petri net Σ' which is obtained from Σ by applying several place insertions as follows. For each $s \in S$ and $d \in D$, if $s \circlearrowright d$ then the place insertion $s \xrightarrow{\bigcirc} d$ is applied, and if $d \circlearrowright s$ then the place insertion $s \xrightarrow{\odot} d$ is applied (note that either $s \circlearrowright d$ or $d \circlearrowright s$ always holds due to Def. 9). Due to Claim 1, all these place insertions are SB-preserving, and so Σ' is safe and bisimilar with Σ. Now, one can observe that Σ' can be obtained from Σ^u by a *type-I secure contraction* of the inserted transition u — a transformation defined in [18, Sect. 3]. Hence, according to [18, Th. 3.5(1)] Σ' and Σ^u are weakly bisimilar (with u considered a silent transition), and so Σ and Σ^u are also bisimilar. The safeness of Σ^u follows from the safeness of Σ' due to [18, Lem. 3.3(3)]. ■

[2]Note that whether a GTI is conservative or not depends on a particular complete prefix for which $\tilde{\circlearrowright}$ was computed; however, in the settings of this paper this prefix is fixed, and so there is no need to further complicate the notation by parameterising the notion of conservativeness with a prefix.

VI. GTIs in the prefix

This section explains how to perform a GTI by locally modifying the existing prefix of the original net, avoiding thus re-unfolding. The following algorithm, given a canonical prefix and a conservative GTI (the conservativeness of which refers to the approximated lock relation $\tilde{\circlearrowright}$ computed on this prefix), builds a canonical (w.r.t. some different cutting context) prefix of the modified net.

Algorithm 11 (GTI in the prefix)**.**

1 *For each marked source $s \in S$ create a new initial p_s-labelled condition $c^{init}_{p_s}$.*

2 *For each instance e_s of each source transition $s \in S$ create a new p_s-labelled condition c_{p_s} and an arc (e_s, c_{p_s}).*

3 *For each co-set X containing no post-cut-off conditions and such that $h_\Sigma(X) = \{p_s \mid s \in S\}$:*

- **a.** *create an instance e_u of the new transition u;*
- **b.** *for all $x \in X$, create the arcs (x, e_u);*
- **c.** *for each transition $d \in D$:*
  - *i) create a new instance c_{p_d} of place p_d and the arc (e_u, c_{p_d});*
  - *ii) for each instance e_d of d such that $[{}^\bullet X] \subset [e_d]_\Sigma$ and $[e_d]_\Sigma \setminus [{}^\bullet X]_\Sigma$ does not contain instances of d other than e_d, create an arc (c_{p_d}, e_d).*

4 *For each cut-off event e with a corresponding configuration C^e, change the corresponding configuration to $\underline{\varphi}(C^e)$.* $\Diamond$

As was already mentioned, the prefix built by this algorithm is not canonical w.r.t. the cutting context

$$\Theta = \left(\approx_{mar}, \lhd, \{\mathcal{C}_e\}_{e \in E_{Unf_\Sigma}}\right)$$

used to obtain the original prefix. Hence, a different cutting context is defined below:

$$\Theta^u = \left(\approx^u_{mar}, \lhd^u, \{\mathcal{C}^u_e\}_{e \in E_{Unf_{\Sigma^u}}}\right),$$

w.r.t. which the resulting prefixes turns out to be canonical. To define Θ^u formally, some auxiliary results are required.

The proposition below explains the natural correspondence between the configurations of Unf_Σ and Unf_{Σ^u}, assuming that Σ^u is obtained from Σ by applying a GTI. It turns out that any configuration C^u of Unf_{Σ^u} corresponds to the unique configuration $\psi(C^u)$ of Unf_Σ, and any configuration C of Unf_Σ corresponds to at most two configurations of Unf_{Σ^u}, denoted $\underline{\varphi}(C)$ and $\overline{\varphi}(C)$, such that the latter is an extension of the former by a single u-labelled event.

Proposition 12 (Correspondence between configurations of Unf_Σ and Unf_{Σ^u})**.** *Let Σ^u is obtained from a safe Petri net*

Σ by applying a GTI, C be a configuration of Unf_Σ and C^u be a configuration of Unf_{Σ^u}. Then:

1) *The set $\psi(C^u) \stackrel{\text{df}}{=} \{e \in C^u \mid h_{\Sigma^u}(e) \neq u\}$ is a configuration of Unf_Σ.*
2) *There exists a unique configuration $\underline{\varphi}(C)$ of Unf_{Σ^u} none of whose causally maximal events is u-labelled and such that $\psi(\underline{\varphi}(C)) = C$. Moreover, there are at most two configurations in Unf_{Σ^u}, $\underline{\varphi}(C)$ and $\underline{\varphi}(C) \oplus e_u$, where e_u is u-labelled, such that $\psi(\underline{\varphi}(C)) = \psi(\underline{\varphi}(C \oplus e_u)) = C$. The latter configuration, if it exists, is denoted by $\overline{\varphi}(C)$; otherwise $\overline{\varphi}(C) \stackrel{\text{df}}{=} \underline{\varphi}(C)$.*
3) *Either $\underline{\varphi}(\psi(C^u)) = C^u$ or $\underline{\varphi}(\psi(C^u)) \oplus e_u = C^u$, and either $\overline{\varphi}(\psi(C^u)) = C^u$ or $\overline{\varphi}(\psi(C^u)) = C^u \oplus e_u$, for some instance e_u of u.*
4) *If C is local then $\underline{\varphi}(C)$ is local.* □

Note that in general, if C is local then $\overline{\varphi}(C)$ is not necessarily local, and if C^u is local then $\psi(C^u)$ is not necessarily local.

Definition 13 (Cutting context Θ^u)**.** *Let Σ be a Petri net and*

$$\Theta = \left(\approx_{mar}, \lhd, \{\mathcal{C}_e\}_{e \in E_{Unf_\Sigma}}\right)$$

be the cutting context with which the canonical prefix $Pref_\Sigma^\Theta$ was built. Then the cutting context Θ^u is defined as

$$\Theta^u = \left(\approx^u_{mar}, \lhd^u, \{\mathcal{C}^u_e\}_{e \in E_{Unf_{\Sigma^u}}}\right),$$

where

- *$C' \approx^u_{mar} C''$ iff $Mrk_{\Sigma^u}(C') = Mrk_{\Sigma^u}(C'')$, for any configurations C' and C'' of Unf_{Σ^u};*
- *$C' \lhd^u C''$ iff either $\psi(C') \lhd \psi(C'')$ or $\psi(C') = \psi(C'')$ and $\#_u C' < \#_u C''$, for any configurations C' and C'' of Unf_{Σ^u};*
- *$\mathcal{C}^u_e \stackrel{\text{df}}{=} \emptyset$ if $h_{\Sigma^u}(e) = u$ and $\mathcal{C}^u_e \stackrel{\text{df}}{=} \{\underline{\varphi}(C) \mid C \in \mathcal{C}_e\}$ if $h_{\Sigma^u}(e) \neq u$.* ◊

To prove that this is indeed a cutting context, it is enough to show that $\lhd^u$ is an adequate order on the configurations of Unf_{Σ^u}. (Note that $\approx^u_{mar}$ is the standard equivalence of final markings on the configurations of Unf_{Σ^u}.)

Proposition 14 (Adequacy of $\lhd^u$)**.** *Let Σ^u be the Petri net obtained from a safe Petri net Σ by a GTI $S \rightarrowtail\!\!\!\!\!\!\rightarrow D$ and $\lhd$ be an adequate order on the configurations of Unf_Σ. Then $\lhd^u$ is an adequate order on the configurations of Unf_{Σ^u}. Moreover, $\lhd^u$ is total if $\lhd$ is total.*

Proof: It is trivial to show that $\lhd^u$ is a strict well-founded order refining $\subset$. Hence, to prove that $\lhd^u$ is an adequate order it remains to show that $\lhd^u$ is preserved by finite extensions, i.e. if $C' \approx^u_{mar} C''$ and $C' \lhd^u C''$ then for any finite suffix E'' of C'', $C' \oplus E' \lhd^u C'' \oplus E''$ for some finite suffix E' of C' such that $C' \oplus E' \approx^u_{mar} C'' \oplus E''$. Moreover, it is enough to show this in the case when E'' is a singleton $\{e''\}$; the required property follows then by induction. Furthermore, one can assume that $C' \lhd^u C''$ holds due to $\psi(C') \lhd \psi(C'')$ (*), as in the alternative case $\psi(C') = \psi(C'')$ and $\#_u C' < \#_u C''$, i.e. C'' can be obtained by extending C' by a single u-labelled event, contradicting $C' \approx^u_{mar} C''$. Below two cases are considered:

$h_{\Sigma^u}(\mathbf{e}'') \neq \mathbf{u}$ Then $\psi(C'' \oplus e'') = \psi(C'') \oplus e''$. Due to (*) there exists a finite suffix $\hat{E}'$ such that $\psi(C') \oplus \hat{E}' \approx_{mar} \psi(C'') \oplus e''$ and $\psi(C') \oplus \hat{E}' \lhd \psi(C'') \oplus e''$ (due to $\lhd$ being an adequate order and hence preserved by finite extensions). Let $E' \stackrel{\text{df}}{=} \underline{\varphi}(\psi(C') \oplus \hat{E}') \setminus C'$. One can see that E' is a suffix of C' such that $C' \oplus E' \approx^u_{mar} C'' \oplus e''$ and $C' \oplus E' \lhd^u C'' \oplus e''$, as $\psi(C' \oplus E') = \psi(C') \oplus \hat{E}' \lhd \psi(C'') \oplus e'' = \psi(C'' \oplus e'')$, and so the required property holds.

$h_{\Sigma^u}(\mathbf{e}'') = \mathbf{u}$ Due to $C' \approx^u_{mar} C''$ and safeness of Σ^u, there is a unique event e' corresponding to u by which C' can be extended. Moreover, $\psi(C'' \oplus e'') = \psi(C'')$ and $\psi(C' \oplus e') = \psi(C')$ as e'' and e' are u-labelled, and so $\psi(C' \oplus e') = \psi(C') \lhd \psi(C'') = \psi(C'' \oplus e'')$ by (*), i.e. the required property holds.

Hence, $\lhd^u$ is an adequate order.

What remains to show is that if $\lhd$ is total then $\lhd^u$ is total as well. For the sake of contradiction, suppose there are two distinct configurations C' and C'' in the unfolding of Σ^u that are unordered by $\lhd^u$. Since $\lhd$ is total, this is only possible when $\psi(C') = \psi(C'')$ and $\#_u C' = \#_u C''$. The former equality implies that one of these configurations can be obtained from the other by extending it by a single u-labelled event, but this contradicts the second equality. Hence $\lhd^u$ is total whenever $\lhd$ is. ■

The following proposition states the correctness of Alg. 11, i.e. that the computed object $Pref^{alg}_{\Sigma^u}$ coincides with $Pref^{\Theta^u}_{\Sigma^u}$. Note that in this result it is essential that the GTI is conservative (with $\tilde{\circlearrowleft}$ computed for $Pref^\Theta_\Sigma$); otherwise the final markings of the cut-off events of $Pref^{alg}_{\Sigma^u}$ and their corresponding configurations might differ.

Proposition 15 (Correctness of Alg. 11)**.** *Let Σ be a safe Petri net and $Pref^\Theta_\Sigma$ be its canonical w.r.t. a cutting context Θ prefix. If the Petri net Σ^u is obtained from Σ by applying a conservative GTI $S \rightarrowtail\!\!\!\!\!\!\rightarrow D$ then the object $Pref^{alg}_{\Sigma^u}$ computed by Alg. 11 coincides with $Pref^{\Theta^u}_{\Sigma^u}$.*

Proof:

Claim 1: The object $Pref^{alg}_{\Sigma^u}$ produced by Alg. 11 is a branching process of Σ^u.

One can check that $Pref^{alg}_{\Sigma^u}$ is an occurrence net, i.e.

- it is acyclic (follows from the acyclicity of the original prefix and the fact that all the additional paths created in Step 3 are consistent with causality due to the condition $[{}^\bullet X]_\Sigma \subset [e_d]_\Sigma$);
- for every condition $b \in B$, $|{}^\bullet b| \leq 1$ (the algorithm does not change the presets of existing conditions, and the newly added conditions have presets which are either empty or singletons);

- no $x \in B \cup E$ is in self-conflict (the only possibility for violating this condition would be after completing Step 3 of the algorithm; however, (i) the new elements created at this step cannot be in self-conflict as this would imply that some elements of the co-set X are in conflict; and (ii) since the events e_d consuming the condition c_{p_d} were already in conflict in the original prefix due to the safeness of Σ, no new conflicts are created at this step, and so the causal successors of the inserted nodes are not in self-conflict);
- for each $x \in B \cup E$ there are finitely many $x' \in B \cup E$ such that $x' \prec x$ (trivial, as the produced object is a finite DAG).

Furthermore, its labelling h_{Σ^u} is a homomorphism from it to Σ^u, i.e.

- $h_{\Sigma^u}(B) \subseteq P_{\Sigma^u}$ and $h_{\Sigma^u}(E) \subseteq T_{\Sigma^u}$ (trivial);
- for all $e \in E$, the restriction of h_{Σ^u} to ${}^\bullet e$ is a bijection between ${}^\bullet e$ and $h_{\Sigma^u}({}^\bullet e)$ (the only potential problem here is that the preset of some instance e_d of some $d \in D$ at Step 3 might be left un-updated; however, this is impossible, as by Prop. 10, Σ and Σ^u are weakly bisimilar, and so one can always find a configuration $C \subset [e_d]_\Sigma$ of the original prefix $Pref^\Theta_\Sigma$ such that $[e_d]_\Sigma \setminus C$ contains no instances of d other than e_d and either $h_{\Sigma^u}(\max_\prec C) = S$ or $h_{\Sigma^u}(\max_\prec C) = \{s \in S \mid s \text{ is unmarked}\}$ and $h_{\Sigma^u}(C) \cap D = \emptyset$, i.e. C can be extended by an instance e_u of u, and the preset of e_d will be extended by one of the conditions in $e_u^\bullet$);
- for all $e \in E$, the restriction of h_{Σ^u} to $e^\bullet$ is a bijection between $e^\bullet$ and $h_{\Sigma^u}(e^\bullet)$ (trivial);
- the restriction of h_{Σ^u} to $Min(Pref^{alg}_{\Sigma^u})$ is a bijection between $Min(Pref^{alg}_{\Sigma^u})$ and M_{Σ^u}, i.e. the minimal conditions correspond to the initial marking (trivial);
- for all $e, f \in E$, if ${}^\bullet e = {}^\bullet f$ and $h_{\Sigma^u}(e) = h_{\Sigma^u}(f)$ then $e = f$, i.e. there is no redundancy (trivial).

Hence $Pref^{alg}_{\Sigma^u}$ is a branching process of Σ^u. □

Claim 2: If e is a cut-off event of $Pref^{alg}_{\Sigma^u}$ with a corresponding configuration C^e then e is causally maximal, $C^e \approx^u_{mar} [e]_{\Sigma^u}$, $C^e \lhd^u [e]_{\Sigma^u}$ and $C^e \in \mathcal{C}^u_e$. Moreover, $Pref^{alg}_{\Sigma^u}$ cannot be extended without consuming a post-cut-off condition.

The maximality of e is trivial, as the algorithm does not insert any events after cut-offs. $C^e \lhd^u [e]_{\Sigma^u}$ follows from $\psi(C^e) \lhd \psi([e]_{\Sigma^u})$ due to e being a cut-off event of $Pref^\Theta_\Sigma$. Furthermore, $C^e \in \mathcal{C}^u_e$ as $h_{\Sigma^u}(e) \neq u$ and $C^e = \underline{\varphi}(C)$ for some configuration $C \in \mathcal{C}_e$ of $Pref^\Theta_\Sigma$ (see Step 4 of the algorithm).

Since e was a cut-off event of $Pref^\Theta_\Sigma$, $\psi(C^e) \approx_{mar} [e]_\Sigma$, and so it remains to show that the final markings of C^e and $[e]_\Sigma$ coincide for the newly inserted places, i.e. $Mrk^p_{\Sigma^u}(C^e) = Mrk^p_{\Sigma^u}([e]_{\Sigma^u})$ for all $p \in \{p_s \mid s \in S\} \cup \{p_d \mid d \in D\} = {}^\bullet u \cup u^\bullet$. Due to the conservativeness of $S \rightarrowtail\!\!\!\!\!\rightarrow D$ and Def. 6(3), the following holds for each $d \in D$: for each unmarked source $s \in S$

$$\#_s[e]_\Sigma - \#_d[e]_\Sigma = \#_s\psi(C^e) - \#_d\psi(C^e), \qquad (*)$$

and for each marked source $s \in S$

$$\#_d[e]_\Sigma - \#_s[e]_\Sigma = \#_d\psi(C^e) - \#_s\psi(C^e).$$

In the latter case both sides of the equation can be multiplied by -1, and so (*) holds for all $s \in S, d \in D$. Since $u \notin S \cup D$, (*) can be re-written as

$$\#_s[e]_{\Sigma^u} - \#_d[e]_{\Sigma^u} = \#_sC^e - \#_dC^e. \qquad (**)$$

The marking equation [11] for the places p_s and p_d and the run represented by the configuration $[e]_{\Sigma^u}$ is as follows:

$$Mrk^{p_s}_{\Sigma^u}([e]_{\Sigma^u}) = M_{\Sigma^u}(p_s) + \#_s[e]_{\Sigma^u} - \#_u[e]_{\Sigma^u}$$
$$Mrk^{p_d}_{\Sigma^u}([e]_{\Sigma^u}) = \#_u[e]_{\Sigma^u} - \#_d[e]_{\Sigma^u}$$

The result of adding these equations is:

$$Mrk^{p_s}_{\Sigma^u}([e]_{\Sigma^u}) + Mrk^{p_d}_{\Sigma^u}([e]_{\Sigma^u}) = M_{\Sigma^u}(p_s) + (\#_s[e]_{\Sigma^u} - \#_d[e]_{\Sigma^u}).$$

Similarly, for the run represented by the configuration C^e, one can get

$$Mrk^{p_s}_{\Sigma^u}(C^e) + Mrk^{p_d}_{\Sigma^u}(C^e) = M_{\Sigma^u}(p_s) + (\#_sC^e - \#_dC^e),$$

and so by (**)

$$Mrk^{p_s}_{\Sigma^u}([e]_{\Sigma^u}) + Mrk^{p_d}_{\Sigma^u}([e]_{\Sigma^u}) = Mrk^{p_s}_{\Sigma^u}(C^e) + Mrk^{p_d}_{\Sigma^u}(C^e). \qquad (***)$$

For the sake of contradiction, suppose $Mrk^{p_s}_{\Sigma^u}([e]_{\Sigma^u}) \neq Mrk^{p_s}_{\Sigma^u}(C^e)$ for some $s \in S$. The following two cases are then considered:

- $Mrk^{p_s}_{\Sigma^u}([e]_{\Sigma^u}) = 0$ and $Mrk^{p_s}_{\Sigma^u}(C^e) = 1$; then by (***) and safeness of Σ^u, $Mrk^{p_d}_{\Sigma^u}([e]_{\Sigma^u}) = 1$ for all $d \in D$, i.e. some u-labelled event in $[e]_{\Sigma^u}$ is maximal; however, this is impossible, as e is the only maximal event in $[e]_{\Sigma^u}$, and it cannot be u-labelled as it was present already in $Pref^\Theta_\Sigma$.
- $Mrk^{p_s}_{\Sigma^u}([e]_{\Sigma^u}) = 1$ and $Mrk^{p_s}_{\Sigma^u}(C^e) = 0$; then by (***) and safeness of Σ^u, $Mrk^{p_d}_{\Sigma^u}(C^e) = 1$ for all $d \in D$, i.e. some u-labelled event in C^e is maximal; however, this is impossible, as C^e is the result of applying $\underline{\varphi}$ to some configuration of $Pref^\Theta_\Sigma$ (see Step 4 of the algorithm).

Hence $Mrk^{p_s}_{\Sigma^u}([e]_{\Sigma^u}) = Mrk^{p_s}_{\Sigma^u}(C^e)$ for all $s \in S$. Then, by (***), $Mrk^{p_d}_{\Sigma^u}([e]_{\Sigma^u}) = Mrk^{p_d}_{\Sigma^u}(C^e)$ for all $d \in D$, and so $C^e \approx^u_{mar} [e]_{\Sigma^u}$.

What remains to show is that $Pref^{alg}_{\Sigma^u}$ cannot be extended without consuming a post-cut-off condition. To the contrary, suppose some configuration C of $Pref^{alg}_{\Sigma^u}$ containing no cut-off events can be extended by some event e. If $h_{\Sigma^u}(e) \neq u$ then $\psi(C)$ can be extended by e as well, contradicting the completeness of $Pref^\Theta_\Sigma$. If $h_{\Sigma^u}(e) = u$ then Step 3 of the algorithm would have added e to the prefix, a contradiction. □

By Claim 1, $Pref^{alg}_{\Sigma^u}$ is a branching process, and what remains to show is that $Pref^{alg}_{\Sigma^u}$ and $Pref^{\Theta^u}_{\Sigma^u}$ coincide, i.e.

(i) e is an event of $Pref^{alg}_{\Sigma^u}$ iff e is an event of $Pref^{\Theta^u}_{\Sigma^u}$; and

(ii) e is cut-off in $Pref^{alg}_{\Sigma^u}$ iff e is cut-off in $Pref^{\Theta^u}_{\Sigma^u}$.

Since $\lhd^u$ is an adequate order (Prop. 14), it is well-founded, and so one can use Noetherian induction on $\lhd^u$. That is,

(i)&(ii) are proved assuming that (i)&(ii) holds for every $f \lhd^u e$ (note that Noetherian induction does not require the base case).

Suppose e is in one of $Pref^{alg}_{\Sigma^u}$, $Pref^{\Theta^u}_{\Sigma^u}$, but not in the other. Then, due to Claim 2 and the completeness of $Pref^{\Theta^u}_{\Sigma^u}$, e is a post-cut-off event in one of them, but not in the other, i.e. there exists an event $f \lhd^u e$ which is cut-off in one of them, but not in the other, which contradicts the induction hypothesis. Hence (i) holds.

Now suppose e is in both branching processes and it is cut-off with a corresponding configuration C^e in one of them, but not cut-off in the other. Due to the induction hypothesis, all the events in $C^e \lhd^u [e]_{\Sigma^u}$ are neither cut-off nor post-cut-off in both branching processes. In what follows, the following two cases are considered.

First, suppose e is cut-off in $Pref^{alg}_{\Sigma^u}$ but not in $Pref^{\Theta^u}_{\Sigma^u}$. Since the algorithm never declares a u-labelled event cut-off in $Pref^{alg}_{\Sigma^u}$, $h_{\Sigma^u}(e) \neq u$, and so e was a cut-off event in $Pref^{\Theta}_{\Sigma}$ with a corresponding configuration $\psi(C^e)$. By Claim 2, e satisfies the criteria of a cut-off event in $Pref^{\Theta^u}_{\Sigma^u}$ with a corresponding configuration C^e, i.e. $C^e \approx^u_{mar} [e]_{\Sigma^u}$, $C^e \lhd^u [e]_{\Sigma^u}$ and $C^e \in \mathcal{C}^u_e$, a contradiction.

Second, suppose e is cut-off in $Pref^{\Theta^u}_{\Sigma^u}$ but not in $Pref^{alg}_{\Sigma^u}$, i.e. $C^e \approx^u_{mar} [e]_{\Sigma^u}$, $C^e \lhd^u [e]_{\Sigma^u}$ and $C^e \in \mathcal{C}^u_e$. Since $h_{\Sigma^u}(e) \neq u$ (as otherwise $\mathcal{C}^u_e \stackrel{\text{df}}{=} \emptyset$ and so e cannot be cut-off in $Pref^{\Theta^u}_{\Sigma^u}$), e was a cut-off in $Pref^{\Theta}_{\Sigma}$ with a corresponding configuration $\psi(C^e)$, because

- $C^e \approx^u_{mar} [e]_{\Sigma^u}$ implies $\psi(C^e) \approx_{mar} [e]_{\Sigma}$;
- $C^e \lhd^u [e]_{\Sigma^u}$ implies either $\psi(C^e) \lhd [e]_{\Sigma}$ or $\psi(C^e) = [e]_{\Sigma}$, but the latter is impossible as $h_{\Sigma^u}(e) \neq u$;
- $\psi(C^e) \in \mathcal{C}_e$ as $h_{\Sigma^u}(e) \neq u$.

Hence, the algorithm would have left e as a cut-off in $Pref^{alg}_{\Sigma^u}$, a contradiction.

Hence (ii) also holds, which completes the proof. ■

VII. Computing useful GTIs

As was already mentioned at the end of Sect. IV, there are relatively few possible sequential and concurrent insertions, which makes it unlikely that insertions needed for logic decomposition exist. In contrast, in highly concurrent Petri nets the number of possible valid GTIs is usually exponential in the size of the net, and so there is a good chance that a suitable GTI can be found. On the flipside, though, it is no longer practical to enumerate all GTIs due to their large number. Hence, a method is needed that would allow one to reduce the number of GTIs that have to be considered, i.e. generate only potentially useful GTIs. Of course, which GTIs are useful depends on the application; however, there are some general techniques which can help here.

The purpose of this section is to demonstrate how potentially useful for logic decomposition of asynchronous circuits GTIs can be derived. The computation is performed in two steps. First, the possible sources are computed; this step is application specific, though the idea of using the incremental SAT for this is likely to be useful for other applications. Then, for a given set of sources, the possible destinations are computed; this step is relatively independent on the application.

Computing sources

In logic decomposition, given a Boolean expression implementing some local signal, its sub-expression $\mathcal{E}$ is selected, and a new internal signal is added in such a way that its implementation is $\mathcal{E}$. For this, several transitions of this signal have to be inserted into the STG. It turns out that such insertions have to be performed at the positions where $\mathcal{E}$ changes its value, cf. Fig. 3.

On the unfolding prefix, such a position can be formalised as a configuration C with each of its maximal events e labelled by a transition of some signal in the support of $\mathcal{E}$, and such that the values of $\mathcal{E}$ at the final states of $C \setminus \{e\}$ and C are different. This problem can be reduced to SAT and solved using an efficient off-the-shelf solver. Note that all such positions have to be computed, and so after one solution is returned by the solver, a new clause is added that rules out all the solutions for which the transitions corresponding to the causally maximal events of C are the same. Then the solver is executed again, giving another solution, and the process is continued until the added clauses make the SAT instance unsatisfiable. Note that many existing SAT solvers can take advantage of the similarity of this family of SAT instances (the technique that is called *incremental SAT*).

By considering all the computed configurations (more precisely, the transitions corresponding to their causally maximal events), the possible sets of transitions that can be used as sources of GTIs are obtained.

Computing destinations

The approach for computing sources described above yields several sets of transitions, and for each such a set $S \neq \emptyset$, one now has to compute all sets $D \neq \emptyset$ such that $S \rightarrowtail\!\!\!\!\!\!\rightarrow D$ is a valid GTI (only conservative GTIs are considered, which is a minor restriction in practice due to Prop. 8). Furthermore, the following minimality property is assumed: for all distinct $s, s' \in S$, $s \tilde{\not\circlearrowleft} s'$, as otherwise s can be removed from S without any change in the resulting behaviour. This property is already guaranteed for the sources produced by the above approach, as the maximal events of any configuration are concurrent with each other and hence the corresponding transitions cannot be locked; for any other approach, this property can easily be enforced by removing some transitions from S.

Given such a set S, one can compute the transitions locked with each $s \in S$:

$$L_S \stackrel{\text{df}}{=} \{d \mid \forall s \in S : s \tilde{\circlearrowleft} d \vee d \tilde{\circlearrowleft} s\}.$$

According to Def. 9, only such transitions can be in D, i.e. $D \subseteq L_S$. The binary *compatibility relation* $\bowtie$ on L_S

introduced below specifies which transitions in L_S can be in the same set D. One of the requirements is the consistent locking with sources (see Def. 9), and the other requirement is the following minimality condition similar to that formulated above for sources: for all distinct transitions d and d', $d \tilde{\not\circlearrowleft} d'$, as otherwise d' can be removed from the destinations without any change in the resulting behaviour.

Definition 16 (Compatibility relation $\bowtie$)**.** *The* compatibility relation $\bowtie$ *on the set of transitions L_S is defined as follows. Let $t', t'' \in L_S$ be a pair of distinct transitions. Then $t' \bowtie t''$ holds iff*

- *for each $s \in S$, $(t' \tilde{\circlearrowleft} s \wedge t'' \tilde{\circlearrowleft} s) \vee (s \tilde{\circlearrowleft} t' \wedge s \tilde{\circlearrowleft} t'')$; and*
- *$t' \tilde{\not\circlearrowleft} t'' \wedge t'' \tilde{\not\circlearrowleft} t'$.* $\Diamond$

The compatibility relation can be viewed as a graph with the vertex set L_S and the edges given by $\bowtie$. Now, any non-empty clique (including non-maximal ones) D in this graph forms a valid set of destinations, i.e. $S \rightarrowtail\!\!\!\!\!\!\rightarrow\!\!\!\!\!\!\rightarrow D$ is a valid conservative GTI. Hence, it is enough to enumerate all non-empty cliques and generate the corresponding GTIs.

Note that in practice further restrictions specific to the application domain can be incorporated into the definitions of L_S and $\bowtie$ in order to reduce the size of the graph. For example, in logic decomposition the newly inserted transitions always correspond to internal signals, and as such must never 'trigger' an input transition (note that the inputs are controlled by the environment, which does not 'see' internal transitions and so cannot wait for them to occur); hence all the input transitions should be removed from L_S before building the compatibility relation $\bowtie$.

Furthermore, if the number of cliques is still too large, the process of their enumeration can be stopped at any time, and one can continue to work with the curtailed set of transformations, especially if the enumeration is implemented in such a way that the most useful (e.g. smallest or largest) cliques are produced first.

A. Commutative and Equivalent transformations

This section is finished with the explanation how to check whether a transformation commutes with a GTI and whether a transformation is equivalent to a GTI (see Sect. IV for definitions). This will complete the discussion concerning the computational aspects related to GTIs.

It is rather obvious that a GTI commutes with other GTIs (including itself) and with any of the transformations described in Sect. IV. Concerning equivalence, one can observe that:

- Two GTIs $S \rightarrowtail\!\!\!\!\!\!\rightarrow\!\!\!\!\!\!\rightarrow D$ and $S' \rightarrowtail\!\!\!\!\!\!\rightarrow\!\!\!\!\!\!\rightarrow D'$ are equivalent iff $S = S'$ and $D = D'$ (note that the notion of equivalence is structural rather than behavioural.) Furthermore, one can reduce the number of behaviourally equivalent GTIs by imposing the minimality condition on their sources and destinations, as described above.
- Though a GTI $S \rightarrowtail\!\!\!\!\!\!\rightarrow\!\!\!\!\!\!\rightarrow D$ cannot be structurally equivalent to a sequential pre-insertion $S' \wr t$, it still makes sense to regard them equivalent if $S = {}^\bullet S'$ and $D = \{t\}$, as they are behaviourally equivalent in such a case.
- Similarly, though a GTI $S \rightarrowtail\!\!\!\!\!\!\rightarrow\!\!\!\!\!\!\rightarrow D$ cannot be structurally equivalent to a sequential post-insertion $t \wr S'$, it still makes sense to regard them equivalent if $S = \{t\}$ and $D = S'^\bullet$, as they are behaviourally equivalent in such a case.
- A GTI $S \rightarrowtail\!\!\!\!\!\!\rightarrow\!\!\!\!\!\!\rightarrow D$ is equivalent to a concurrent insertion $t' \overset{n}{\rightarrowtail\!\!\!\!\!\!\rightarrow} t''$ iff $S = \{t'\}$ and $D = \{t'\}$. Hence, in practice it makes sense to impose an additional constraint $|S \cup D| > 2$ to avoid generating a GTI that is equivalent to some concurrent insertion.

VIII. Conclusions

In this paper, a new type of transition insertions has been proposed, and the corresponding theory and a suit of algorithms have been developed to integrate it into the transformation framework developed in [5]. In particular, the contributions include:

- A method for computing the approximated lock relation $\tilde{\circlearrowleft}$ using a complete unfolding prefix (Def. 6). This approximation is always conservative (Prop. 7), and exact in the practical case of a live Petri net (Prop. 8).
- A new kind of transition insertion, called generalised transition insertion (Def. 9). This transformation preserves safeness and yields a weakly bisimilar Petri net (Prop. 10).
- An algorithm for efficient conversion of a given canonical prefix of a Petri net into a canonical prefix of the Petri net obtained by applying a conservative GTI (Alg. 11), avoiding thus (expensive) re-unfolding. Interestingly, a different cutting context has to be used for the resulting prefix (Prop. 15). (In general, re-unfolding the modified Petri net with the original cutting context can yield a very different prefix from that returned by Alg. 11.) As an auxiliary result, the correspondence between the configurations of the unfoldings of the original and modified Petri nets was established (Prop. 12).
  An additional advantage of this approach (besides avoiding re-unfolding) is that it yields a prefix very similar to the original one, which is useful for visualisation and allows one to transfer some information from the original prefix to the modified one (e.g. [6] used this feature, albeit for the transformations explained in Sect. IV, to transfer the yet unresolved encoding conflicts into the new prefix, avoiding thus re-computing them from scratch).
- Since the number of all possible GTIs grows exponentially with the size of the Petri net, their straightforward enumeration is impractical. Hence, a method for computing only potentially useful GTIs in the context of logic decomposition was developed (Sect. VII); however, some parts of this method (viz. computing possible destination

for the given set of sources) are relatively independent on the application domain. This method is complemented with the commutativity and equivalence results for GTIs (end of Sect. VII).

These contributions form a complete framework for efficient use of GTIs together with the transformations developed earlier (see Sect. IV).

Currently, the developed theory is applied to logic decomposition of asynchronous circuits (though it is quite generic). In future work, it would be interesting to integrate further transformations into the developed framework.

REFERENCES

[1] T.-A. Chu, "Synthesis of self-timed VLSI circuits from graph-theoretic specifications," Ph.D. dissertation, Lab. for Comp. Sci., MIT, 1987.

[2] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, *Logic Synthesis of Asynchronous Controllers and Interfaces*. Springer-Verlag, 2002.

[3] A. Yakovlev, L. Lavagno, and A. Sangiovanni-Vincentelli, "A unified signal transition graph model for asynchronous control circuit synthesis," *FMSD*, vol. 9, no. 3, pp. 139–188, 1996.

[4] L. Rosenblum and A. Yakovlev, "Signal graphs: from self-timed to timed ones," in *Proc. Int. Workshop on Timed Petri Nets*. IEEE Comp. Soc. Press, 1985, pp. 199–206.

[5] V. Khomenko, "Behaviour-preserving transition insertions in unfolding prefixes," in *Proc. ATPN'07*, ser. LNCS, vol. 4546. Springer-Verlag, 2007, pp. 204–222.

[6] ——, "Efficient automatic resolution of encoding conflicts using STG unfoldings," *IEEE Trans. VLSI Syst.*, no. 17, pp. 855–868, 2009, special section on asynchronous circuits and systems.

[7] J. Engelfriet, "Branching processes of Petri nets," *Acta Informatica*, vol. 28, pp. 575–591, 1991.

[8] J. Esparza, S. Römer, and W. Vogler, "An improvement of McMillan's unfolding algorithm," *FMSD*, vol. 20, no. 3, pp. 285–310, 2002.

[9] V. Khomenko, "Model checking based on prefixes of Petri net unfoldings," Ph.D. dissertation, School of Comp. Sci., Newcastle Univ., 2003.

[10] V. Khomenko, M. Koutny, and V. Vogler, "Canonical prefixes of Petri net unfoldings," *Acta Informatica*, no. 40, pp. 95–118, 2003.

[11] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[12] P. Cohn, *Universal Algebra*, 2nd ed. Reidel, 1981.

[13] "International Technology Roadmap for Semiconductors: Design," 2007, URL: `http://www.itrs.net/Links/2007ITRS/2007_Chapters/2007_Design.pdf`.

[14] D. Muller and W. Bartky, "A theory of asynchronous circuits," in *Proc. Int. Symp. of the Theory of Switching*, 1959, pp. 204–243.

[15] A. Martin, "The limitations to delay-insensitivity in asynchronous circuits," in *Proc. 6th MIT Conf. on Advanced Research in VLSI*. MIT Press, 1990, pp. 263–278.

[16] V. Varshavsky, Ed., *Self-Timed Control of Concurrent Processes*. Kluwer Academic Publishers, 1990, translated from Russian, published by Nauka, Moscow, 1986.

[17] P. Vanbekbergen, F. Catthoor, G. Goossens, and H. De Man, "Optimized synthesis of asynchronous control circuits from graph-theoretic specifications," in *Proc. ICCAD'90*. IEEE Comp. Soc. Press, 1990, pp. 184–187.

[18] W. Vogler and R. Wollowski, "Decomposition in asynchronous circuit design," Inst. für Informatik, Univ. Augsburg, Tech. Rep. 2002-05, 2002.