Event-based Framework for Verifying High-Level Models of Asynchronous Circuits

Alexei Semenov* and Alexandre Yakovlev†
Department of Computing Science,
University of Newcastle upon Tyne, NE1 7RU, England

University of Newcastle upon Tyne, Computing Science, Technical Report Series, No. 487, May 1994

Abstract

Obtaining a compact representation of the behaviour specified by Labelled Petri nets is an important part in verification of asynchronous circuit model at both abstract and logic synthesis levels. The paper outlines major problems of the application of the unfolding method to the bounded Petri nets. One of the possible ways of solving these problems is introduced. A new algorithm for verification of the behaviour specified by Signal Transition Graph is introduced and discussed.

Keywords asynchronous circuit design, unfolding, bounded Petri net, Labelled Petri Net, Signal Transition Graph.

1 Introduction

Asynchronous circuits have traditionally been seen as more difficult objects to design than their clocked counterparts. This attitude has recently improved due to a number of powerful techniques and tools for synthesis and verification of such circuits. They are based on a variety of modelling approaches used at different levels of abstraction. For example, component delay models (inertial, pure, chaos etc.), gate vs wire delay ratios (bounded wire/unbounded gate, delay-insensitive etc.), circuit-environment interaction modes (fundamental, input-output), transition race semantics (multiple winner, general multiple winner etc.) are just some of the factors affecting the overall circuit design process.

Verification is an important part of design and with the newly emerging hierarchical design methods, it becomes rather difficult to precisely define the scope of its application. Until recently, it has mainly been used for gate level analysis. Now it is often required to justify structural and behavioural decomposition of a circuit's abstract description, when precise wiring and signalling between the circuit and its environment has not yet been defined. At this level, the circuit can be described as an interconnection of modules, each of which is specified by its behavioural model [17]. Alternatively, even for the same structural modules, there can be several "partial view" descriptions which, when superimposed, produce the overall behavioural model of the module. Before each such module description is further refined or run through logic synthesis, the designer must check if the present level composition of either module specifications or "partial views" is correct against its higher level specification. Such dynamic model analysis is typical for the most difficult (due to its inherent irregularity) type of circuitry, control or interfacing circuits, for instance, bus adapters,

*Supported by grants from the Research Committee of the University of Newcastle upon Tyne and CVCP.
†Supported by SERC grant No. GR/J 52327

arbiters and buffers. These are the characteristic examples of system-level components that have proved most appropriate for asynchronous design approach.

Petri Nets (PNs) and their interpretations, such as Signal Transition Graphs (STGs), or other related models, such as Change Diagrams, have become a popular specification language for such designs. They have a number of advantages, among which the most important are their natural ability to capture causal, concurrency and conflict relations between discrete events, modelling clarity due to the easy graphical interpretation, syntactic likeness to timing diagrams, direct semantic links with models in traces (event sequences) and states (finite automata).

Several different formalisms have been used to verify Petri net descriptions of circuits. Here are just the most well-known examples: Trace Structures [3, 5], Symbolic Model Checking against Temporal Logic Specification [1], Event Coordination Model [7], Change Diagram unfoldings [6] and Petri Net unfoldings [10]. From the viewpoint of the fundamental semantic framework, they can be classified into two major groups: the state graph or automata graph models and event-based or pure causality models.

Due to the obvious complexity reasons, the state graph approach is often a serious obstacle for verifying Petri net models of large size. Despite their intuitive simplicity and close link with the existing logic synthesis methods (e.g., from STGs), which are mostly state graph based [9, 2], the state graph techniques often cannot compete in complexity with the causality or net unfolding approach [10]. At the same time, the use of the latter for analysis of certain classes of nets (e.g. unsafe or k-bounded nets) appears to be in practice not as efficient as desired (see Section 3 and figure 3). This is mainly caused by the fact that the pure causality models, such as net unfolding, have *overly* powerful means to distinguish between different event occurrences. One simple example is when two or more tokens arrive into the same place in a net, the semantic model based on the unfolded net (occurrence net from [11]) represents all possible alternative cases of token ordering individually, and thus produces several alternative copies of the same transition occurrence distinguished only by a particular order of tokens activating them. As a result, the model creates several (this number is proportional to the "degree of boundedness" of the place) different configurations of events all corresponding to the same state in the reachability graph of the net. Thus, from the viewpoint of the user interested in generating the semantic representation which should correspond to the reachability graph by covering it in a minimal way, such a fine modelling is not needed. Even the idea of using the truncated unfolding according to the construction method introduced in [10] does not remove this redundancy. Thus, the primary advantage of the unfoldings method (its ability to capture the reachability set without generating it explicitly [10]) is lost.

The unfolding approach applied to Change Diagrams [6] is different from McMillan's and does not create redundancy (even for k-bounded diagrams) by tacitly assuming the strict (FIFO) order between tokens generated by the cause events for effect events. However, its applicability is limited due to the original syntactic restriction of Change Diagrams – they cannot model choice and conflicts in behaviour. Another special feature of Change Diagrams, compared to ordinary uninterpreted nets (the object of [10]), is that their events are labelled with signal transitions. In this sense, Change Diagrams represent a behavioural model of the logic synthesis level. This modelling level has a specific consistency property which relates the specification to its binary encoded state graph (further used for logic implementation). Thus, the verification task has to be applied to check such a property. It was demonstrated that this check is again far more efficient using the unfolding rather than state graph. The remaining open question is whether a similar sort of consistency check could be done for the general type of STGs, capable of modelling choice and conflicts between signal transitions, in addition to concurrency and two forms (AND and OR) of causality.

The major aim of this paper is therefore twofold:

- to tackle the problem of potential semantic redundancy of McMillan's truncated unfolding that may arise for k-bounded Petri nets and modify, or better say specialise, McMillan's method; and

- to develop the technique of checking consistency of an STG (such an STG is called *valid*) using the modified truncated unfolding.

The overall organisation of the remaining sections is as follows. Section 2 provides background material for the rest of the paper. In particular, it outlines major issues of behavioural verification in a two-stage control circuit design methodology based on Labelled Petri Nets. Section 3 considers the problems of efficiency concerned with the use of the net unfolding technique for general class of nets. Section 4 modifies McMillan's technique for constructing the truncated unfolding to cater for k-bounded net analysis. Section 5 specialises the modified unfolding for analysis of STGs and their consistency checking. Section 6 contains experimental results. The full version of this paper exists as [15].

2 Preliminaries

2.1 Verification Aspects in Asynchronous Control Circuit Design

A high-level design approach based on Labelled Petri Nets (LPN) has been presented elsewhere [17]. This paper only looks at its verification aspects. An LPN is an ordinary Petri Net whose transitions are labelled with the names of abstract actions performed by the modelled control circuit module. The process of circuit design is divided into two stages: (i) *abstract* synthesis at the level of LPN models of components and (ii) *logic synthesis* at the gate level. The first stage starts with an abstract specification of the control circuit made in LPN. This specification reflects the main notion of causality, choice and conflicts the designer has about the circuit as a whole. For example, if the designed circuit is a buffer memory of some finite (say k) capacity, it is modelled as an abstract mechanism with two ports, a for writing a datum into the buffer and b for reading a datum. Such a model is presented by a simple LPN shown in figure 1(a). With the overall circuit specification, the designer begins to study potential ways of implementing this top-level description. A straightforward and often not well-thought approach could be to proceed to the logic synthesis directly, by producing the so-called *signalling expansion* of the abstract model and generating an STG corresponding to the original LPN. Such a direct way may face with certain difficulties such as solving the state coding problem for the STG or finding an area efficient implementation. The main problem here is that the initial LPN, though being structurally not so great, may "hide" its complexity in its behaviour. For example, it is easy to imagine a net for k-place buffer that, for a large value of k, may generate a large state space. Existing STG-based synthesis tools cannot cope with state-encoding for such "behaviourally dense" nets [6, 8].

A natural way is to decompose the original circuit model into a set of modules in such a way that each of these modules' behaviour is defined by an LPN whose state graph is simpler than that of the top level LPN. This process may then continue until each component is sufficiently simple to be synthesised at the logic level. For every decomposition step, the designer has to perform a *test of conformance* between the specification and its implementation. In terms of the LPN notation, this test effectively means checking that the *parallel composition* of the component LPNs is semantically equivalent to the specification LPN. For example, a two-place buffer represented on the higher level as a 2-bounded LPN can be decomposed into a pipeline interconnection of two one-place buffers, each modelled by safe LPNs. The LPN that is the parallel composition of these nets can be regarded as an implementation of the original LPN specification.

The second stage of design, at which an STG model is created from the abstract LPN description of a component, requires the following verification procedures. First, the STG or signalling expansion has to be checked for conformance to its LPN origin. This check is based on such issues as the type of signalling expansion (*handshake expansion* or simple *signal casting*, *four-phase* or *two-phase signalling* [17]) and the *renaming* mapping between the (*critical and auxiliary*) transitions of the STG and the LPN actions. Second, the STG model has to be semantically consistent

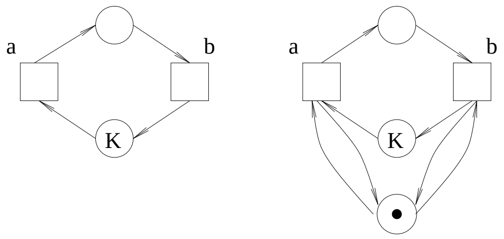


Figure 1: A LPN corresponding to pure causality (a) and interleaving semantics (b) models of k-place buffer.

with respect to all its handshake orderings and each individual binary signal transitions. The latter would guarantee that the STG can produce a state graph for subsequent synthesis.

The above analysis of the verification tasks involved in the design process clearly shows at least two types of semantic checks: the LPN conformance check and the STG consistency check. Other problems that may be required in the design are *deadlock detection*, *coverability of a subset of places* and so on.

In the following section we shall demonstrate how the choice of a particular semantic framework can affect efficiency of these checks.

2.2 Traces versus Pure Causality Modelling

An important question, which affects the choice of a particular conformance check technique, is what type of semantics this conformance has to rely upon. One standard type, used e.g. in [4], is the trace or *interleaving* semantics. This type implies conformance formally defined by the following relationship: $L(N_1) = L(N_1 \parallel N_2)\lceil A_{N_1}$ (for proof see e.g. [13]), where N_1 (N_2) is the LPN of the specification (implementation), $L(N)$ denotes the *set of traces* generated by LPN N and A_N is the alphabet of events of net N. Here, $\parallel$ denotes the parallel composition of LPNs, defined as a synchronisation between two nets on the sets of transitions labelled with the same action symbol, and $\lceil$ denotes the projection operator, which projects each trace from the given set onto the given alphabet of actions.

The way the trace approach to conformance test works is based on constructing state (reachability) graphs for the two LPNs and then applying the state graph conformity technique (and its associated verifier) as shown in [4]. This approach is quite appropriate in many cases, even though it may hide certain semantic differences between the two nets. These differences can be expressed by their finer behavioural account, the *pure causality* or *partial order* semantics. As an example, consider again a k-place buffer. It is clear that if we modify the LPN shown in figure 1(a) by adding a "mutual exclusion" place (to allow the buffer implementation based on an Up/Down counter and arbiter [16, 4, 17]), the trace semantics will not change while the pure causality one will be different. Indeed, in the case of figure 1(b) our buffer does not allow both reading and writing to be performed completely independently, e.g. in parallel, whereas the case of figure 1(a) it does. Another potential problem with the trace-based approach is the size of the state graph for for highly parallel nets. This problem affects both conformance and consistence checking. Therefore, the techniques such as net and STG unfolding, which are based on pure causality semantics, could offer a means for *semantic distinction* that the user may wish to respect in the model's behaviour. Plus they would allow *compact representation of concurrency* in its natural form. In reality, these improvements appear to be conflicting (philosophically, this is no surprise — finer descriptive means should lead to larger, not smaller, representations) and, as the following section demonstrates for the general type of nets, the direct application of the unfolding technique may not yield what is expected. We should however first introduce some (limited) formal notation used in the rest of the paper.

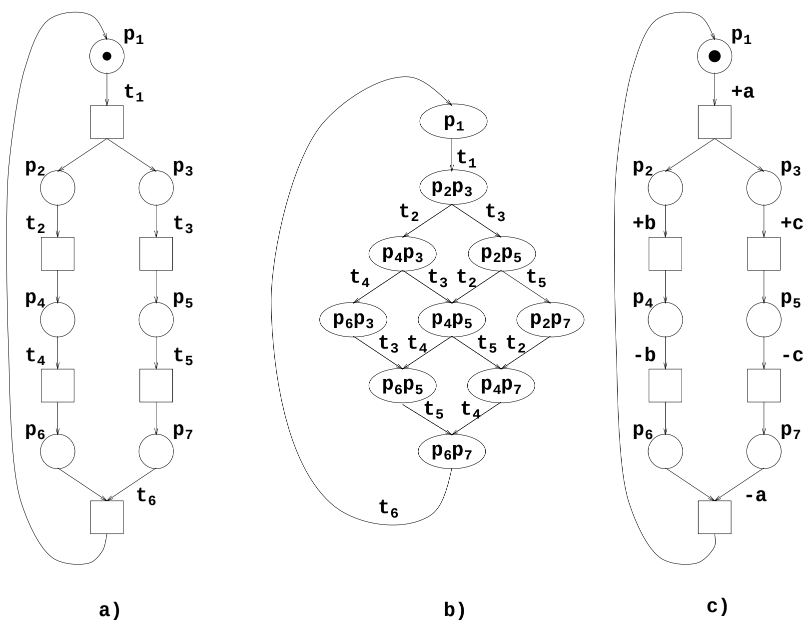


Figure 2: A PN (a), its RG (b) and an example of an STG interpretation (c) of this PN.

2.3 Basic Definitions

We assume reader's familiarity with Petri nets (PN). Briefly, PN is a graph with two types of nodes (*transitions* and *places*) and directed arcs denoting the *flow relation* between nodes. A *marking* of PN is a multiset m on places. Usually marking m is represented by tokens in places of a PN. A *marked* PN is therefore a PN and its initial marking m_0. We shall assume that any PN in this paper is a marked PN unless it is specified explicitly. A transition is *enabled* at marking m if all of its predecessors are marked. An enabled transition may *fire* yielding a new marking m' in which there is one token less in every predecessor place and one token more in every successor place. Each element has its *pre-* and *postset* defined as a possibly empty set of its predecessors and successors respectively.

A (possibly empty) set σ of transitions including all intermediate transitions which have fired between two markings m to m' is called a *firing*, or *feasible* sequence. A marking m' reachable through σ from m is denoted as $m[\sigma > m'$. A set of markings reachable from initial marking m_0 (denoted as $m[>$) is called the *reachability set* of the marked PN. It can be depicted as a graph, called *reachability graph* (RG) with nodes labelled with markings and arcs labelled with transitions. An example of a PN and its RG are shown in figure 2. A PN N is called *k-bounded* iff $m_0[>$ is finite and in any feasible sequence the number of tokens at any marking $m: \ m_0[> m$ in any place $p \in m$ is less or equal than k. A PN is called *bounded* if there exists a finite k for which the PN is k-bounded. A 1-bounded PN is called a *safe* PN. From the practical reasons of modelling finite size hardware, we shall further consider **only** bounded PNs.

A special case of LPN is called *Signal Transition Graph* (STG). Its transitions are labelled with signal changes from a finite set of signals A. Allowed changes for a signal a_i are $+a_i$ and $-a_i$. An STG has an initial state of the STG, which is a binary vector of dimension $|A|$. An example of an STG interpretation of the PN from figure 2(a) is shown in figure 2(c).

A feasible sequence $\sigma : \ m_0[\sigma > m$ of an STG is *valid* iff for every signal a: (i) the next possible change of signal a after $+a(-a)$ can only be $-a(+a)$, and (ii) the first change of signal a is consistent with the initial state of the STG, i.e.: if the value of a is 0(1) in the initial state, then only $+a(-a)$ can first happen in any feasible sequence. An STG is called *valid* iff every feasible sequence in it is valid. An STG shown in figure 2(c) is valid if the initial state is (0,0,0). If the initial state is other than (0,0,0), then this STG will be invalid.

A *single-run* STG is an STG whose RG contains no cycles. Also, a *cyclic* STG is an STG whose RG is strongly connected. Often practically useful are STGs which can be considered as combination of single-run and cyclic segments, e.g. an STG with a segment which is executed once before it enters its cyclic segment. Such STGs model the behaviour of circuits or signalling protocols with initialisation.

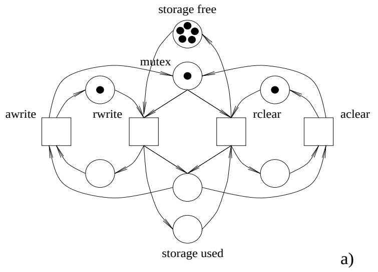
STG-based synthesis techniques using an assignment of consistent state vectors to the states in the RG have been studied in [14, 2, 9]. They were based on a single consistent *state vector* to be assigned to every state in the RG, i.e. for any two states of the RG connected with an arc, labelled with a signal transition, the state vectors can differ in only one element which corresponds to the signal involved in the transition. This (RG-based) consistency notion appears to work only for cyclic STGs. As we show below, for single-run STGs or STGs with an acyclic segment such a vector assignment may not satisfy our intuition about STG consistency.

Unlike PN, the dynamic behaviour of STG can not be defined without taking into consideration the interpretation of the marking of the underlying PN with corresponding states of signals. Indeed, it is always done for the initial state — one has to specify the initial marking of the STG along with its initial signal values. It is therefore natural to define a concept of a *full state* (FS) [15] of an STG which is a pair (marking, state vector) such that each element of the binary state vector corresponds to one and only one signal and has exactly $|A|$ elements. We thus introduce a *Full State Graph* (FSG) [15], where each node is labelled with an FS and arcs labelled with signal transitions. The FSG allows any marking of the underlying PN to be associated with *more than one* state vector as long as there is a unique node for the pair. Consistency must however still require that any two nodes of the FSG connected with an arc differ only in the value of one signal whose signal transition labels this arc. The values of element of state vectors corresponding signal a_i change from 0(1) to 1(0) when the signal transition $+a_i(-a_i)$ occurs. The construction of FSG for an STG is similar to that of the ordinary RG for a PN except that (i) two nodes are considered equal iff *both* their markings and state vectors are the same, and (ii) *consistency* is checked between the enabled signal transition and the state vector of the enabling marking.

While building the FSG it is possible to reach an FS for which one cannot produce new transitions. This may happen either because there are no transitions enabled by the FS marking or the FS state vector component is inconsistent with some signal transitions associated with the PN transition enabled by the FS marking. The former situation is known as a *deadlock*. By analogy, we shall call the latter one as a *signal deadlock with respect to the signal transition*. An example of FSG for an STG in figure 10(a) is shown in figure 10(c). It is easy to see its semantic difference with the RG (figure 10(b))

We adopt the notion a PN unfolding from [10] as a labelled occurrence net [11]. An *unfolding* built from the PN N is the maximal labelled occurrence net (up to isomorphism) in which (1) every transition is associated with a single transition of the original PN (for which this transition is a particular occurrence), (2) the pre- and postset of transition of the unfolding map on the pre- and postset of the associated transition of the original PN it is representing and (3) the initial set of places maps on the initial marking. Two transitions of unfolding t_1, t_2 are said to be in *conflict* ($t_1 \# t_2$) iff there exist two different transitions preceding t_1' and t_2' sharing their input places. For uniformity we define an *initial transition* of the unfolding, denoted as transition 0, which has initial set of places of unfolding as a postset. There is no preset for the transition 0. A very important notion [10] is *configuration* C of an unfolding, which is a subset of transitions of the unfolding complying with the conditions (a), if a transition belongs to C then all transitions preceding it also belong to C, and (b), no two transitions of C are in conflict. A *postset of configuration* C is a set of places having their predecessor included into C and none of their successors included in C. The mapping of the postset of configuration C back onto the original PN yields a multiset of places of original PN called the *final state* of C, denoted as $F_s(C)$. It was proved in [10] that for any reachable marking m there exists a configuration C such that $m = F_s(C)$ and vice versa.

A special case of configuration is *local configuration of transition* t', denoted as, $\lceil t' \rceil$ which



No. tokens	Original PN			Transformed safe PN		
	No. states	No. trans.	No. trans.*	No. states	No. trans.	No. trans.*
1	4	4	4	-	-	-
2	7	20	10	16	12	12
3	10	96	18	58	24	24
4	13	520	28	208	40	40
5	16	3260	40	754	60	60

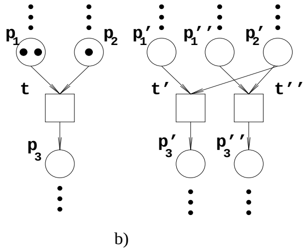


Figure 3: Storage unit and is characteristics (a) and example of redundancy (b).

includes t' and all transitions preceding t'. If in the unfolding t'_1 precedes t'_2 then $\lceil t'_1 \rceil \subseteq \lceil t'_2 \rceil$. For any transition t' in the unfolding the following is true: $\lceil 0 \rceil \subseteq \lceil t' \rceil$.

Using the notion of local configuration we can now determine the relations between transitions of unfolding. Two transitions can be called *concurrent* exactly when they are not in conflict and neither of their local configurations includes the other transition. Hence, we can say that two transitions of the original PN are strongly concurrent *iff* any of their occurrences are concurrent.

Building the unfolding is strightforward, but the unfolding can be infinite, so we need some truncation of the unfolding which will be finite yet will be as informative as the full unfolding. In [10] the *cutoff point* was defined as a transition t'' whose $\lceil t'' \rceil$ has the same final state as the $\lceil t' \rceil$ already constructed and the size of $\lceil t'' \rceil$ is strictly bigger than $\lceil t' \rceil$. The rest of the unfolding from the cutoff point is discarded as no configuration including cutoff points will produce a new marking.

3 Redundancy of Truncated Unfolding

The cutoff point condition as defined above gives rise to a truncated unfolding which fully represents the RG of a PN but attempts to avoid exploring all reachable states explicitly. There are two disadvantages of this truncated unfolding: (i) *not all* transitions of the original PN can be represented in it, and (ii) the truncated unfolding can have *redundant copies* of transitions for *unsafe* or even *safe* PNs, in the sense that some *reachable markings can be represented more than once.*

The first problem can be solved easily by adding cutoff points into the truncated unfolding. The other issue is of more concern. We can demonstrate this on the following example of the PN specification of the storage unit [4] reproduced in figure 3(a). The PN is unsafe (k-bounded though). From the column "No. trans." for this (original) PN we see that the truncated unfolding obtained according to [10] will grow exponentially with the increase of k – number of tokens in the place "storage free". On the other hand, we can transform the original net to a safe net whose behaviour would be equivalent with respect to the set of feasible traces it generates projected onto the original alphabet of labels. Such a transformed net is easily built by adding (in a "pipelined way") a set of *dummy* transitions. The number of states in the equivalent safe net will however be growing exponentially to k, which can be seen from the table in figure 3. At the same time the number of transitions in the truncated unfolding grows linearly. This analysis shows that the unfoldings in the form as introduced in [10] are unsuitable for analysis of unsafe PNs. It is also easy to find an example of a PN (e.g. FSM) in which from two transitions with equal final states and sizes of local configurations one of them can be turned into a cutoff point without loosing any reachable markings. In the next sections we introduce the modification which allows us to reduce substantially the sizes of PN behaviour representation as shown in column "No. trans*.". Yet the segment will allow us to carry out the analysis of LPNs, for example for STGs. In fact the cutoff condition introduced in [10] is a *sufficiency condition* for the truncated unfolding to represent all reachable markings. In the example in figure 3(b) it is enough to check if two local configurations have equal final states although the length of their local configuration will be equal. We show further that an attempt to find the *necessity condition* by simply ignoring the condition on sizes of

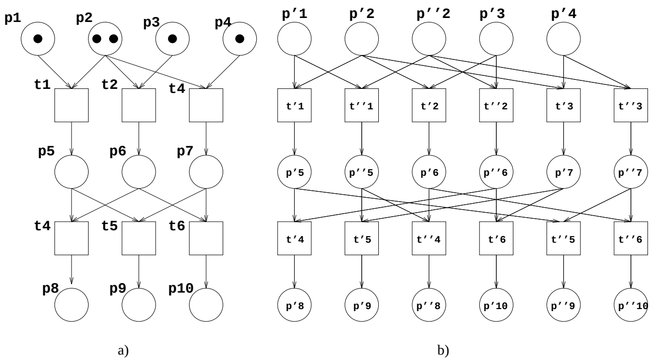


Figure 4: An PN (a) and its truncated unfolding (b) obtained using McMillan's algorithm.

local configurations is inappropriate for unsafe PNs.

There are two reasons which cause McMillan's truncated unfolding to be redundant for k-bound PNs. Firstly, its condition on the size of local configurations is *overly strong*. Namely, it leaves in the truncated unfolding the transitions whose local configuration have the same final states and the same size. We show one of the ways of relaxing this condition in the next section. Secondly, some of the local configurations have indeed their final states equal to the final states of some other already built (not necessarily local) configurations. Checking the final state of a new transition's local configuration against the final states of all existing (nonlocal) configurations would not be efficient however. This would increase complexity of the problem amounting to the construction of the RG of a PN. Thus we would rather prefer an algorithm (or better say the cutoff point condition) which could provide us with some quasi-optimal representation of a PN.

4 The Unfolding Segment

Let us consider the algorithm for unfolding the ordinary PN with the aim to obtain an *implicit* representation of all reachable markings in the form of a finite segment of the unfolding. We also want to obtain the *minimal segment*, i.e. the segment of the unfolding which contains no redundant copies of transitions even in the case of unsafe PN. A transition of the unfolding whose local configuration has a final states equal to the final state of some other configuration is called *redundant* transition. For the reasons explained in the last paragraph of the previous section we shall consider **only** the final states of local configurations. We need a new cutoff point condition which would exclude the redundant copies of transitions from further consideration *"on the fly"*, i.e. during the construction of the unfolding itself.

Consider the truncated unfolding (figure 4(a)) built for a PN using algorithm of [10]. By definition, the truncated unfolding is a PN. Let this PN be marked with initial marking m'_0 equal to the set of initial places of the unfolding. We can build the RGs for both the original PN (denote it by R) and its truncated unfolding (R') (figure 5). Note that since the truncated unfolding is acyclic, its RG R' is an acyclic graph with initial node r'_0. Each state of the RG of the truncated unfolding can be mapped onto some marking of the original net. A *representative set of enabled transitions of a local configuration* is a set of noncutoff transitions of the unfolding which map on the enabled transitions at the final state of this local configuration. Let us denote the relation between transition of truncated unfolding t'_i and state r'_j of its RG R' as $t'_i \Rightarrow r'_j$ if there exists an arc in R' terminating at r'_j and labelled with t'_i. Note that the postset of a local configuration maps onto a state of R' which has only one arc terminating at this state.

It is important to observe that *if any two states r'_1, r'_2 in R' map on one state in R, then for each state r'_3 reachable from the r'_1 there exists another state r'_4 reachable from r'_2 such that both r'_3 and r'_4 map on the same state in R.* Another useful observation is that *for any transition labelling the*

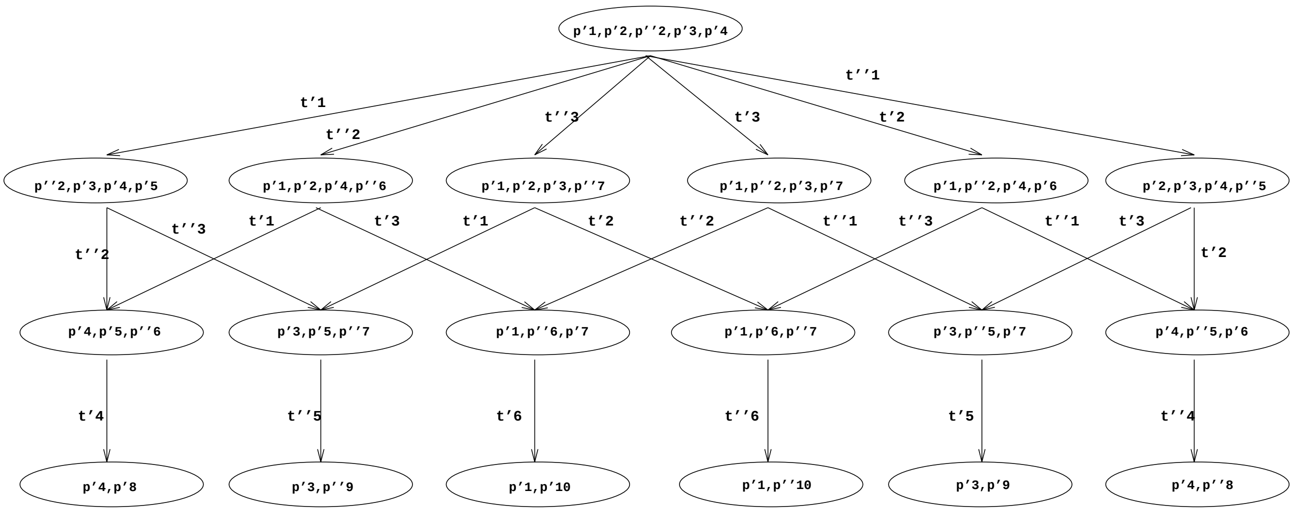


Figure 5: A reachability graph R' for the truncated unfolding.

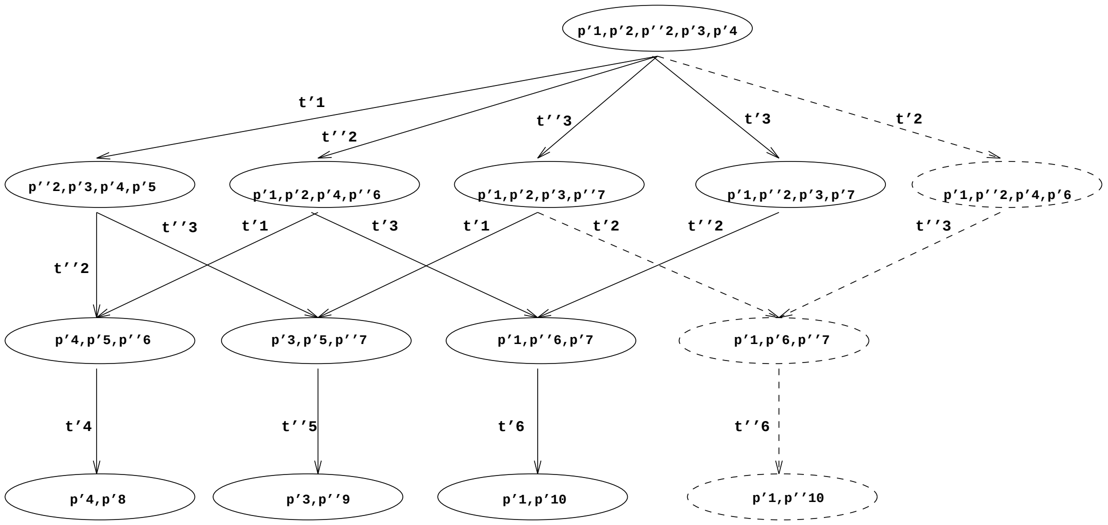
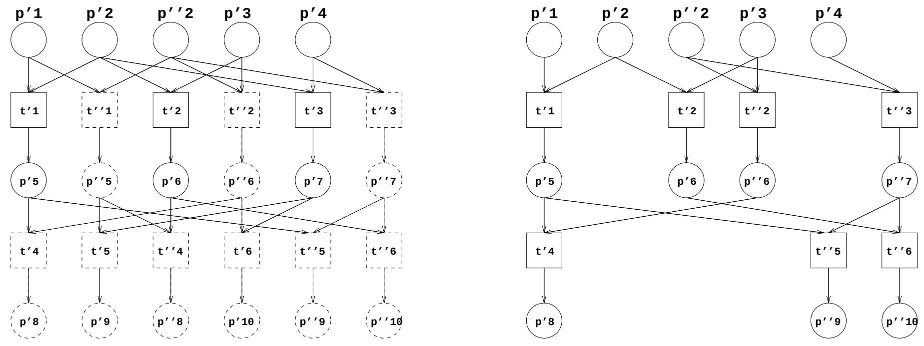


Figure 6: The modified reachability graph with arc labelled with t_1'' (and corresponding states) removed.

only arc terminating at some state all states reachable through this transition are also reachable from that state. It follows that, *if any two states map on the same state in the original RG and both have only one arc terminating at them labelled with t_1' and t_2' respectively, then we can exclude all states for which either $t_1' \Rightarrow r_i'$ or $t_2' \Rightarrow r_j'$ is true* and yet the number of states of R represented in the modified R' will still be the same. The elimination can be done excluding either of these transitions, say t_1', from the unfolding as if it is a cutoff point. Note that this time we shall decide whether *a transition is a cutoff point* **iff** *the final state of its local configuration is equal to the final state of some other already existing local configuration.*

Unfortunately, we cannot apply this procedure iteratively in unsafe PNs: once we have excluded one transition from further consideration (see figure 6) we can no longer guarantee that the above statement holds again. Excluding from the modified truncated unfolding a randomly chosen transition which satisfies our "new condition" may result in losing some of the reachable markings as shown in figure 7(a), where elements not included into the segment are shown in dashed line. However such a condition can be shown to produce a segment of unfolding which *will not be redundant for safe PNs.* On the other hand, it is clear that in the case of unsafe PNs some of the states (and transitions) can still be redundant copies (as in figure 6) and it is desirable to eliminate them from



a) b)

Figure 7: Examples of a segments.

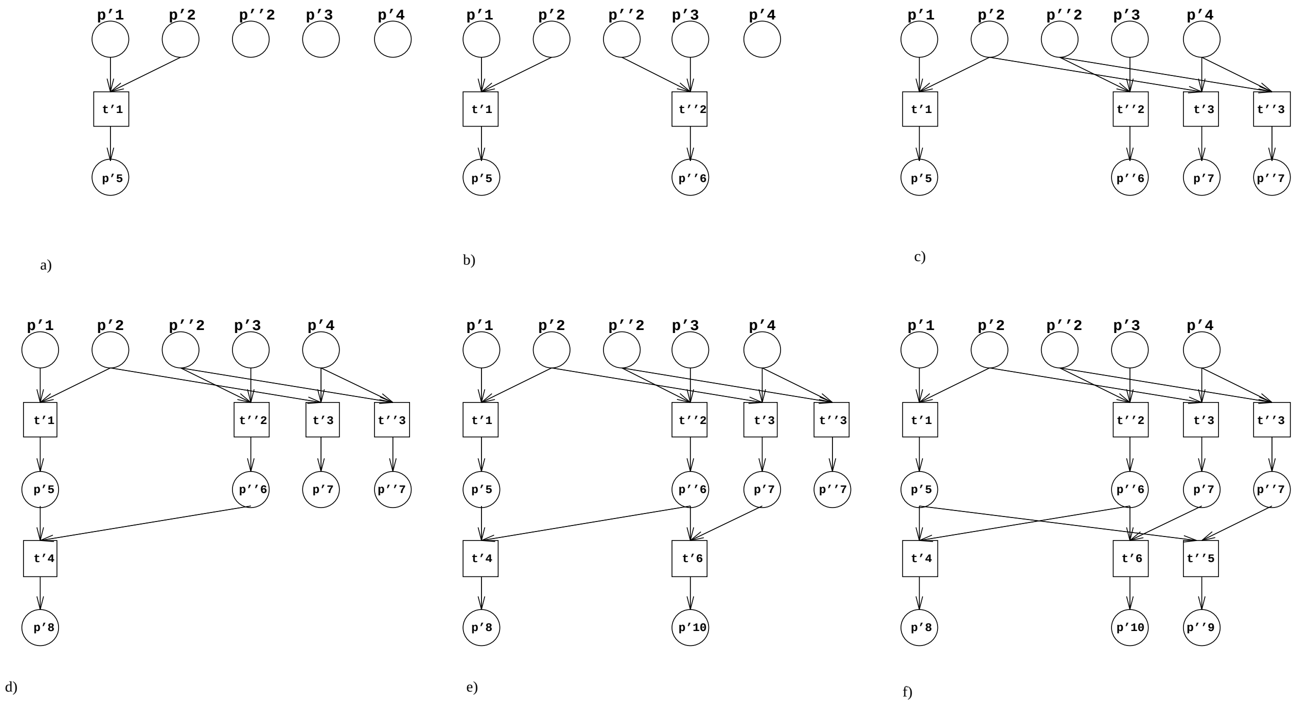


Figure 8: Unfolding a PN using *new cutoff condition*: (a) transition t_1 explored (t_1'' decided to be a cutoff point; (b) transition t_2 explored (t_2' is a cutoff point); (c) transition t_3 explored (neither t_3' nor t_3'' is a cutoff point); (d) transition t_4 explored; (e) transition t_6 explored; (f) transition t_5 explored;

further consideration.

Let us turn our attention to the representative sets. Intuitively, from *a pair of transitions in conflict whose final states and sizes of local configurations equal* we should exclude from further consideration the one which has its representative set inclusive into the other. Thereby we should guarantee that the number of reachable states of R represented in the modified R' remains intact. If the representative sets are *incompatible* then *both* transitions *must be kept* to avoid possible loss of their successors, and if the sets are *equal*, then we *can exclude either* of the transitions.

We can prune the modified RG R' excluding the arcs labelled with the transitions t'_i if for the state corresponding to the postset of $\lceil t'_i \rceil$ there exists another state being a postset of some other local configuration with equal final state and its representative set covering the representative set of $\lceil t'_i \rceil$. Thus we obtain the RG shown in figure 6 without states drawn in dashed line for the example shown in figure 4. Since we would like to prune R' "on the fly", while constructing it (strictly speaking, while the net unfolding is being constructed), we need to be able to make a decision about a newly generated transition while unfolding the PN. Note that the representative sets can be incomplete at the moment when the decision should be made. We can show however that the decision will still be correct. The key argument is that the algorithm [10] makes use of a *queue of transitions* ordered with the sizes of their local configurations. This ensures that any transition with the local configuration larger than that of a given t' will be considered *after all* the transitions with the sizes of their local configurations *less or equal* to the size of local configuration of t'. Hence we can make an assumption that *for any newly generated transition t' the unfolding built this far contains* **only** *transitions whose local configurations are less or equal than the size of* $\lceil t \rceil$. Another assumption which we can make is that *all instances of one transition with the same size of local configuration are added to the unfolding without any intermediate transitions.* Thus our **new condition of the cutoff point** will look as follows:

> **A transition t' of the unfolding is a cutoff point if there exists another transition t'' such that the final states of local configurations of t' and t'' are equal** *and* **either:**
> **(1) t'' has smaller local configuration or,**
> **(2) t' and t'' are in conflict** *and* **the representative set of the local configuration of t' is included in the representative set of local configuration of t''.**

In the actual process of unfolding this condition is applied in the following way: assume that t'_1 is a candidate for inclusion into the unfolding and t'_2 is another already built transition with the same final state of its local configuration. From t'_1 and t'_2 we choose the transition to be cutoff point with either bigger local configuration or, *only* when they are in conflict, the transition which has the representative set of its local configuration inclusive into the representative set of the local configuration of the other. Note that using this condition we may decide that the already existing transition can be marked as a cutoff point. However, this is not the case when we mark the transition, say t'_2, which has some other transition t'_i inputting from the postset of t'_2 because in that case the size of local configuration of t'_i will be bigger (at least by one transition) than the size of local configuration of t'_2; and we have to consider those which have their local configurations of the same size first (i.e. t'_1). Hence even if t'_2 has been decided to be a cutoff point and excluded from further consideration no other transition in the segment is affected as no t'_i has been added yet. From a simple example shown in figure 9 we can see that even for unsafe, but bounded PNs we shall obtain a segment where some of the redundant transitions will be excluded. Yet *all* reachable markings will be represented in this segment. Using the above condition of cutoff point we can develop an algorithm which will construct the segment of the unfolding. This segment will be *quasi-redundant* because it still will not compare the final states of local configurations with final states of configurations of the segment. Hence some of the states of the RG of original PN may be duplicated in it. However it will avoid major redundancy related to the local configurations with equal sizes and final states for most PNs.

The algorithm will be very similar to the one explained in [10], with exception to the cutoff point condition. An example of obtaining a segment for the PN shown in figure 4(a) is shown in figure 8.

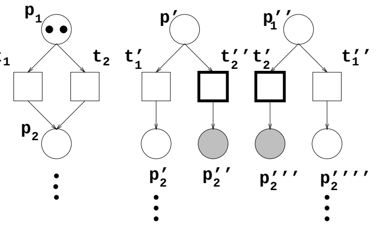


Figure 9: An example of unsafe PN and its unfolding.

The sequence in which the transitions of the original PN are being explored is: $t_1, t_2, t_3, t_4, t_6, t_5$. We can write out the representative sets of the transitions for the steps (a): for final states of local configurations of t_1' and t_1'' they are equal to $\{\}$; (b): for final states of local configurations of t_2' and t_2'' they are $\{\}$ and $\{t_1\}$ respectively; and (c): for final states of local configurations of t_3' and t_3'' they are $\{t_2\}$ and $\{t_1\}$ respectively. There is no need to write representative sets for t_4', t_5'' and t_6' simply because they are not checked. Clearly the shape of the segment depends on the order in which transitions of original net are explored. Another example (order is $t_1, t_3, t_2, t_4, t_6, t_5$) of the segment corresponding to the PN in figure 4 is shown in figure 7(b). The segment will still represent the RG of the original PN.

The obtained segment of the truncated unfolding is quasi-minimal in the sense that it represents only markings reachable from the initial marking. The size of the segment (actual number of transitions in it) is sensitive to the order in which we pick the transitions of the original PN. Clearly, if the number of transitions sequential with t_2 were less than the number of transitions sequential with t_3 in the above example then the segment obtained using the order $t_1, t_2, t_3, t_4, t_6, t_5$ would have smaller size that the segment obtained using the order $t_1, t_3, t_2, t_4, t_6, t_5$. This is the penalty which we have to pay if we want to exclude redundant transitions "on the fly" since we cannot say in advance the "posthistory" after t_2 and t_3.

Note that while building the unfolding we can also determine k-bound for each place (maximum number of independent, i.e. conflict-free and mutually unordered, occurrences of a particular place) or if the PN is (safe) bounded at all.

5 STG-segment

Consider now the algorithm for analysis of an STG. The motivation for adaptation of the above algorithm to STG analysis is that, if all events are included into the segment (i.e. the cutoff points are kept in the segment), then we can determine the *strong concurrency* relation between two signal transitions if any two occurrences of these transitions were concurrent in the segment. The validity condition of an STG can then be transformed into the checking of following conditions for each signal: (i) there should be no strongly concurrent occurrences of signal transitions in the segment; (ii) all changes of the signal should be in total alternating order; and (iii) the first change of the signal should be consistent with initial state of STG. Since the algorithm outlined above can produce a representation of the RG of an underlying PN we require additional modification of the algorithm so that we could obtain the representation of FSG.

It is clear that the marking component of the full states can be traced according to the relations as of the algorithm for segment unfolding of the PN underlying an STG. Due to potential presence of signal deadlocks the configurations may represent marking in the RG which will not be covered by the FSG. In order to be able to derive the state vector component we need the notion of the *signal state of configuration* which is binary vector with $|A|$ elements each element corresponding to one and only one signal of A. The signal state of configuration is *defined iff* for each signal all occurrences of signal transitions in this configuration are in total order with altering signs and the sign of signal transition represented by first occurrence is consistent with initial state of STG.

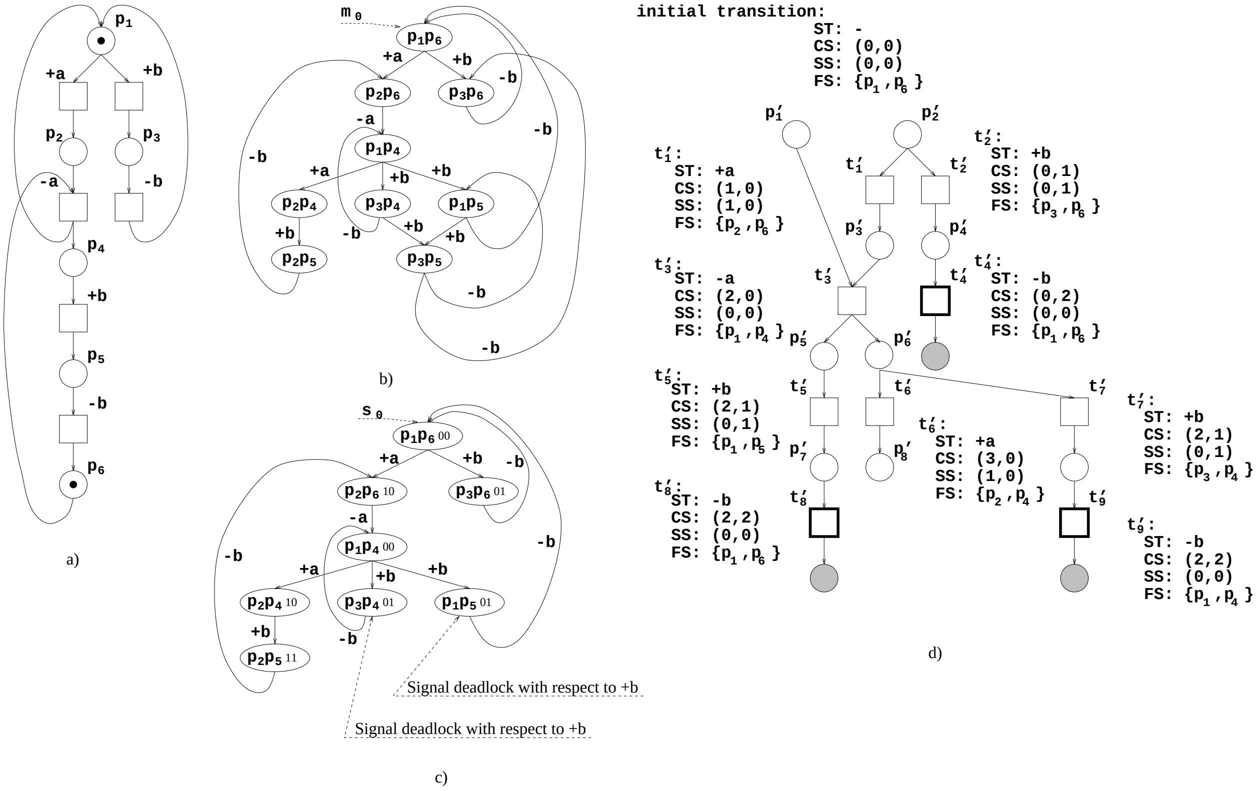


Figure 10: An example of STG (a), its RG (b), its FSG (c) and its STG-unfolding (d).

The value of this signal state for each signal is calculated as the modulo 2 sum of the number of occurrences of signal transitions produced in this configuration and the value of corresponding initial state vector. A local configuration, satisfying the above condition, also has a signal state because it is a configuration.

It is convenient to represent the number of occurrences in vector form which is called *cumulative state of configuration* (The idea of using cumulative states is similar to [6].). The cumulative state of local configuration of 0 is a 0-vector. The cumulative state for a configuration C for which its signal state is defined is then calculated as a componentwise maximum of cumulative states of local configurations comprising C. An *STG-unfolding* is then an unfolding of the underlying PN together with defined signal states for each local configuration. Obviously, if there exists a configuration with the defined signal state, then there exists an FS with the marking component equal to the final state of this configuration and the state vector component equal to its signal state.

Whenever a new transition t' is added to the unfolding the state of its local configuration can be calculated from the states of local configurations forming the minimal configuration C whose postset is the preset of t', using their cumulative states and verifying the validity conditions: (i) there are no concurrent occurrences of the transitions of one signal and (ii) the new transition is consistent with the signal state of C. The minimal configuration from which a transition t' cannot be built is called the *break point*. Discovering at least one break point while building an unfolding means that there exists a signal deadlock in the FSG and the STG is invalid.

We can now write the cutoff condition in the STG-unfolding which is much alike to the cutoff condition defined in the previous section except that we shall require that *two transitions of unfolding having equal final states of their local configurations should also have equal signal states of their local configurations*. A segment obtained using this cutoff condition is called *STG-segment*. If we have built the STG-segment we also have to check if the signal states are defined for the *final configurations* which are the maximal configurations of the segment. If for all final configurations their signal state is defined, then the STG is valid.

STG	Algorithm 1				Algorithm 2				No. states
	Time (sec.)	No. trans.	No. final states	Time* (sec.)	Time (sec.)	No. trans.	No. final states	Time* (sec.)	
NAK1	421	366	121	59	53	142	111	10	229
NAK2	892	574	227	168	155	225	199	32	984
NRA	5	56	39	2	3	48	39	1	67

Table 1: Performance of the STG verification algorithms.

An example of the STG-segment for the STG in figure 10(a) is shown in figure 10(d). Abbreviations are as follows: ST - corresponding signal transition; CS - cumulative state of the local configuration; SS - signal state of the local configuration; and FS - final state of signal configuration. There is no signal transition corresponding to the initial transition and hence it is not drawn on in the figure. We can see that the STG is invalid because the signal state of the final configuration $\{t'_1, t'_3, t'_5, t'_7, t'_8, t'_9\}$ is not defined.

Note that in order to obtain a representation of all signal transitions in the STG-segment we had to keep the cutoff points in the segment. Strictly speaking, this would introduce another redundancy: the states of the FSG which are marked as (final state, signal state) of local configurations will be duplicated. However this redundancy is unavoidable if the representation of all transitions is required.

6 Real example

Finally, we present some experimental results. We verified the STG specifications[1] for a Low Latency NAKing arbiter (NAK1), a Low Latency arbiter with a 'canonical' Mutex on In1, Out1, In2, Out2 (NAK2) and Non-reseting arbiter (ordinary net model without CSC problem) (NRA). The results can be seen in table 1.

The STG description of NAK1 has 33 transitions and 36 places, of NAK2 has 73 transitions and 77 places and NRA has 18 and 23 respectively. Algorithm 1 was implemented using the condition of the cutoff point defined as in [10] (we shall call the segment obtained using this algorithm segment 1). Algorithm 2 was implemented using our new condition of the cutoff point (we shall call the segment produced by this algorithm STG-segment). Both algorithms keep the cutoff points in the unfolding, yet the successors of cutoff points are marked so that the algorithms do not choose them while building new transitions. Note that the number of final states of local configurations explored using algorithm 2 is less in the case of NAK1 and NAK2. This is because some of these final states explored by algorithm 1 are actually duplicates of some other states which are final states of some configurations in the truncated unfolding. Observe the difference between the number of transitions in STG-segment and segment 1 for all STGs. This is because the STGs are unsafe (2-bounded). The reduction in the number of constructed transitions has resulted in time reduction (see columns "Time" for both algorithms) needed for obtaining the STG-unfolding when using algorithm 2. We have also implemented both algorithms in a time saving version where time is a trade-off for space: the local configurations were kept along with every transition. The results can be seen in table 1, columns "Time*" for both algorithms. The time saving version gives even more significant improvement in time to obtain the segment of the unfolding. This is because the local configurations are not built every time we need to check conflict or sequential relation between places or transitions. Note that the times shown above also include the "on the fly" verification of the STG validity, not only the time of building the segments.

[1]Personal communications A. Yakovlev and L. Lavagno. The general idea of NAKing arbiters can be found in[12].

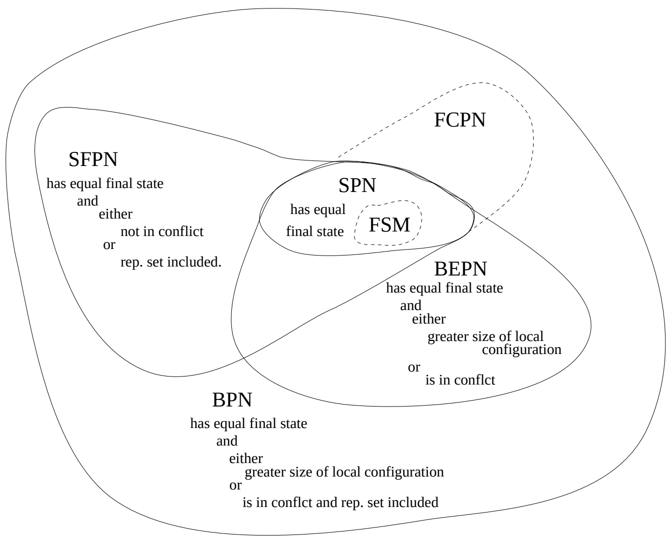


Figure 11: PNs and cutoff conditions.

7 Conclusions

We showed that the unfolding method can be applied to the STG verification at the lowest level of design although it requires modifications to represent the FSG [15]. Although the segment in the PN case and the STG-segment in the STG case are quasi-optimal we have achieved great reduction in the size of the segment. We can perform some analysis of the behaviour specified by the LPN such as deadlock detection [10]. Moreover, the segments obtained from unfolding the LPN can later be used for conformity analysis. It is clear that we can verify "strong conformity" (concurrency, conflict and order relations between transitions are exactly the same) between the LPN-specified behaviour of the higher level and composition of LPN specified behaviours of lower level. A sufficient condition for that will be that the order of (crucial) transitions in the local configurations of the unfolding segment built from the higher level LPN is preserved in the local configurations of the unfolding segment built for the parallel composition of the higher level specification and composition of lower level LPNs. The problem arises if we would like to verify the conformity in terms of interleaving semantics. One of the possible and strightforward ways is to compare the traces (i.e. all configurations of the segments) and thus verify if the sets of traces are the same. The other approach is to specialise the unfolding technique so, that the LPN conformity check will be performed again "on the fly". The basic idea is still simple — if the order of any transition firings has changed, then we must detect it.

Under the assumptions made in the section 4 we can identify the following classes of PNs and the corresponding cutoff conditions (see also diagram in figure 11):

Safe PNs (SPN); cutoff condition is as follows: a *transition t' is a cutoff point if there exists another transition in the unfolding with the same final state of local configuration.*

Balanced Enabled PNs (BEPN), i.e. PNs in which for any marking for each transition t enabled at this marking the number of tokens in the places of the preset of t is equal; the cutoff condition is as follows: a *transitions t' is a cutoff point if there exists another transition in the unfolding with the same final state of its local configuration* **and** either *has the size of its local configuration less than t'* or *(if the sizes of local configurations are equal) it is in conflict with t'*

Serialised Firings PNs (SFPN), i.e. PNs in which for any reachable marking m each transition t enabled at this marking cannot fire again from the marking $m' : m[t > m'$ unless its preset includes places from its postset; the cutoff condition is as follows: a *transition t' is a cutoff point if there exists another transition in the unfolding with the same final state of its local configuration* **and** either *it is not in conflict with t'* or *the representative set of $\lceil t' \rceil$ is included into representative* *set of the local configuration of the other transition.*

Bounded PNs (BPN); the cutoff condition is as follows: *a transition t' is a cutoff point if there exists another transition in the unfolding with the same final state of its local configuration* **and** either *has the size of local configuration less than t'* or *(if the sizes are equal) if it is in conflict with t' and the representative set of $\lceil t' \rceil$ is included into representative set of the local configuration of the other transition.*

The segment of the unfolding will not contain redundant transitions at all only if the PN is (safe) FSM. In all other cases the unfolding may contain redundant transitions because some transitions of the unfolding may have the same final state of their local configuration as a final state of some other (not local) configuration. Note also that if a PN is a SFPN *and* Free-Choice PN (FCPN) then it is an SPN.

The analysis given in section 3 shows that the unfoldings method in the original form works efficiently only to the restricted class of *safe, highly concurrent* PNs where all local configurations in the unfolding are of different size. The method given in this paper can be applied preserving the gain in its complexity domain to a wider class of PNs, i.e. bounded PNs. The complexity of this method comes close to the complexity of the RG analysis if there are many local configurations in the unfolding of a PN which have their final states of local configurations equal to the final states of already built (not local) configurations. It has also been observed that the time and complexity of building an unfolding, even using condition from section 4, is greater than for building the RG for the unsafe nets with *high "degree of unsafeness"* and *low "degree of concurrency"*. This is due to overheads on calculating the relations between occurrences of places and transitions. Nevertheless, the described method is practical for unfolding the PNs which are both *"highly concurrent"* and *"highly unsafe"*.

One of the problems in the design of asynchronous circuits using PN-based behaviour specifications is *output non-persistency* [8], i.e. the possibility of hazards and meta-stable states caused by the changes of the inputs before the outputs are stable. The problem can be resolved introducing mutual exclusion elements (a well-known arbitration mechanism) resolving the conflict between the input and output signals. The unfolding is perfectly suited to determining conflicts between different occurrences of transitions, thereby identifying *non-persistent outputs*.

8 Acknowledgements

Authors would like to thank A. Taubin and A. Kondratyev for helpful discussions at the early stage of this work, and L. Lavagno for his useful comments about the unfolding redundancy observations. We would also like to thank K.McMillan for valuable clarifications of his unfolding algorithm.

References

[1] J.R. Burch, E.M. Clarke, K.L. McMillan, Dill D.L., and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *In Proceedings of the Fifth Annual Symposium on Logic in Computer Science*, June 1990.

[2] T.A. Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-theoretic Specifications.* PhD thesis, MIT, 1987.

[3] D.L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits.* The MIT Press, Cambridge, Mass., 1988. An ACM Distinguished Dissertation 1988.

[4] D.L. Dill, S.M. Nowick, and R.F. Sproull. Specification and automatic verification of self-timed queues. *Formal Methods in System Design*, 1:29–60, 1992.

[5] J.C. Ebergen. *Translating Programs into Delay-Insensitive Circuits*, volume 56 of *CWI Tract.* CWI, Amsterdam, 1989.

[6] M. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky. *Concurrent Hardware: The Theory and Practice of Self-Timed Design*. John Wiley and Sons, London, 1993.

[7] R.P. Kurshan. Analysis of descrete event coordination. In *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness. REX Workshop Proceedings, Lecture Notes in Computer Science, 430*, pages 414–453. Springer-Verlag, New York, 1990.

[8] L. Lavagno. Personal communications.

[9] L. Lavagno and A. Sangiovanni-Vincentelli. *Algorithms for Synthesis and Testing of Asynchronous Circuits*. Kluwer Academic Publishers, Massachusetts, 1993.

[10] K.L. McMillan. Using unfolding to avoid the state explosion problem in the verification of asynchronous circuits. Manuscript, February 1993. The earlier version of this paper was presented on the 4th Workshop on Computer Aided Verification, Montreal, 1992.

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

[12] S.M. Nowick and D.L. Dill. Practicality of state-machine verification of speed-independent circuits. In *Proceedings of ICCAD'89*, Santa Clara, CA, November 1989.

[13] I. Reicher and M. Yoeli. Net-based modelling of communicating parallel processes with applications to VLSI design. Technical Report 532, Technion, Haifa, 1988.

[14] L.Ya. Rosenblum and A.V. Yakovlev. Signal graphs: from self-timed to timed ones. In *Proceedings of International Workshop on Timed Petri Nets, Torino, Italy, July 1985*, pages 199–207. IEEE Computer Society, 1985.

[15] A. Semenov and A. Yakovlev. STG verification using net unfoldings. Manuscript in preparation.

[16] I.E. Sutherland. Micropipelines. *Communications of ACM*, 32(6):720–738, 1989.

[17] A.V. Yakovlev, A.M. Koelmans, and L. Lavagno. High level modeling and design of asynchronous interface logic. *To be published in: IEEE Design and Test*, November 1994. The full version of the paper was published as Technical Report No.460, University of Newcastle upon Tyne, November 1993.

9 Appendix

We reproduce in the appendix figures of STG specified behaviour for Low Latency NAKing arbiter (NAK1), a Low Latency arbiter with a 'canonical' Mutex on In1, Out1, In2, Out2 (NAK2) verified in the section 6.

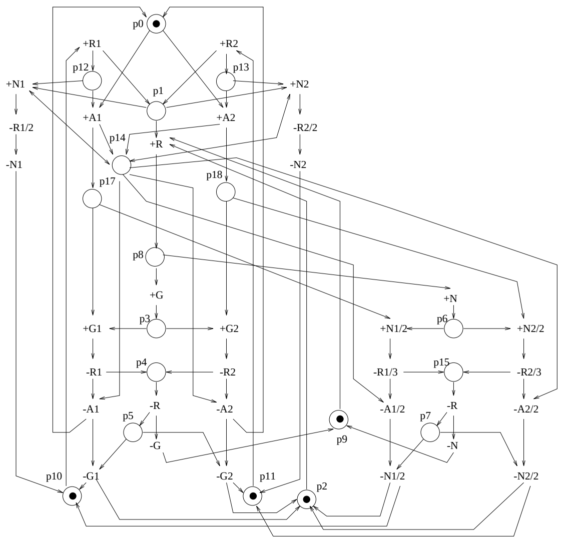


Figure 12: STG specification of Low Latency NAKing arbiter (NAK1).

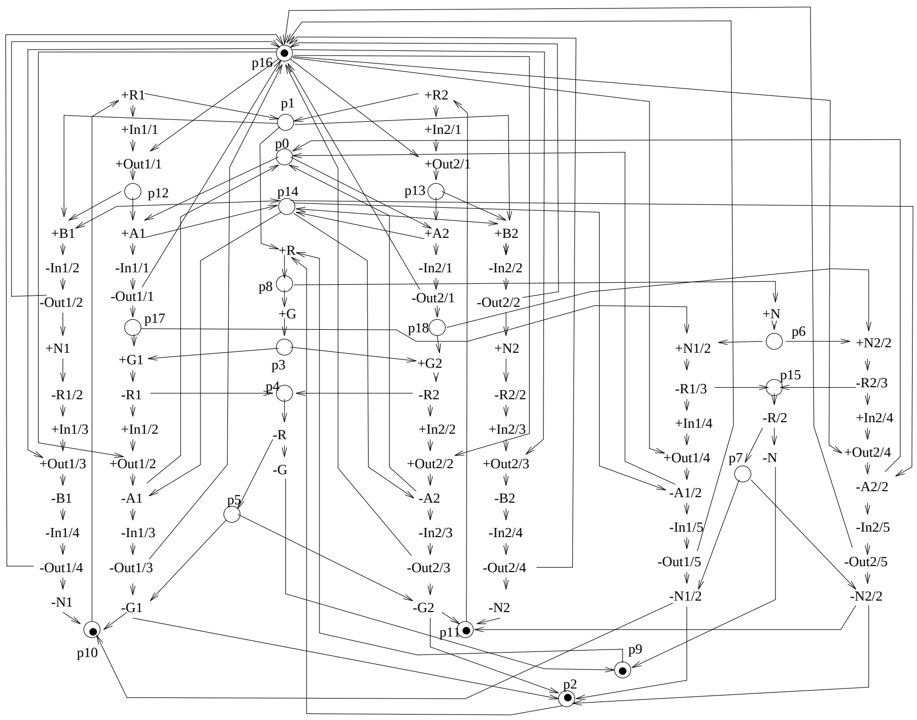


Figure 13: STG specification of Low Latency arbiter with a 'canonical' Mutex on In1, Out1, In2, Out2 (NAK2).