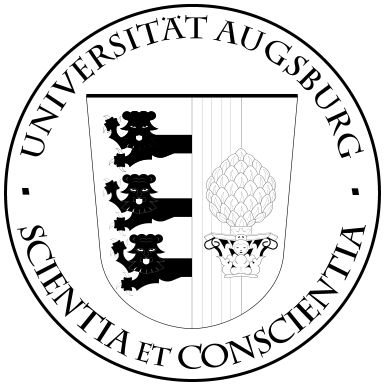
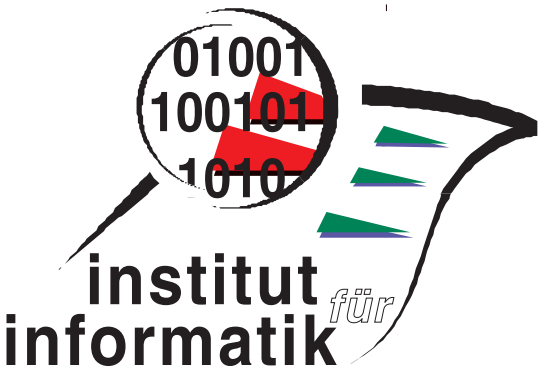
Universität Augsburg



Combining Decomposition and Unfolding for STG Synthesis

Victor Khomenko and Mark Schaefer

Report 2007-01 January 2007



Institut für Informatik
D-86135 Augsburg

Combining Decomposition and Unfolding for STG Synthesis

Victor Khomenko[1] and Mark Schaefer[2]
[1]School of Computing Science, Newcastle University, UK.
victor.khomenko@ncl.ac.uk
[2]Institute of Computer Science, University of Augsburg, Germany
mark.schaefer@informatik.uni-augsburg.de

Abstract

For synthesising efficient asynchronous circuits one has to deal with the state space explosion problem. In this paper, we present a combined approach to alleviate it, based on using Petri net unfoldings and decomposition. The experimental results show significant improvement in terms of runtime compared with other existing methods.

Keywords: Asynchronous circuit, STG, Petri net, decomposition, unfolding, state space explosion.

1 Introduction

Asynchronous circuits are a promising type of digital circuits. They 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 [ITR05] 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.

Signal Transition Graphs, or STGs [Chu87, CKK+02], are widely used for specifying the behaviour of asynchronous control circuits. They are interpreted Petri nets in which transitions are labelled with the rising and falling edges of circuit signals. An STG specifies which outputs should be performed at a given state and, at the same time, it describes assumptions about the environment, which can send an input only if it is allowed by the STG. We use the *speed-independent* model with the following properties:

- Input and outputs edges can occur in an arbitrary order.
- Wires are considered to have no delay, i.e., a signal edge is received immediately by all listeners.
- The circuit must work properly according to its formal description under arbitrary delays of each gate.

Synthesis based on STGs involves: (a) checking sufficient conditions for the implementability of the STG by a logic circuit; (b) modifying, if necessary, the initial STG to make it implementable; and (c) finding appropriate Boolean next-state functions for non-input signals.

A commonly used tool, Petrify [CKK+97], performs all these steps automatically, after first constructing the reachability graph of the initial STG specification. To gain efficiency, it uses symbolic (BDD-based [Bry86]) techniques to represent the STG's reachable state space. While this state-space based approach is relatively simple and well-studied, the issue of computational complexity for highly concurrent STGs is quite serious due to the *state space explosion* problem [Val98]; that is, even a relatively small STG can (and often does) yield a very large state space. This puts practical bounds on the size of control circuits that can be synthesised using such techniques, which are often restrictive (e.g., Petrify often fails to synthesise circuits with more that 25–30 signals), especially if the STG models are not constructed manually by a designer but rather generated automatically from high-level hardware descriptions, such as Balsa [EB02] or Tangram [Ber93].

In order to alleviate this problem, Petri net analysis techniques based on causal partial order semantics, in the form of Petri net unfoldings, were applied to circuit synthesis. Since in practice STGs usually exhibit a lot of concurrency, but have rather few choice points, their complete unfolding prefixes are often exponentially smaller than the corresponding state graphs; in fact, in many of the experiments conducted in [Kho03, KKY04] they are just slightly bigger then the original STGs themselves. Therefore, unfolding prefixes are well-suited for both visualisation of an STG's behaviour and alleviating the state space explosion problem. The papers [KKY04, KKY06, MBKY03] present a complete design flow for complex-gate logic synthesis based on Petri net unfoldings, which completely avoids generating the state graph, and hence has significant advantage both in memory consumption and in runtime, without affecting the quality of the solutions. Moreover, unfoldings are much more visual than state graphs (the latter are hard to understand due to their large sizes and the tendency to obscure causal relationships and concurrency between the events), which enhances the interaction with the user.

The unfolding-based approach can often synthesise specifications which are by orders of magnitude larger than those which can be synthesised by the state-space based techniques. However, this is still not enough for practical circuits. Hence, we combine the unfolding approach with decomposition. Intuitively, a large STG can be decomposed into several smaller ones, whose joint behaviour is the same as that of the original STG. Then these smaller components can be synthesised, one by one, using the unfolding-based approach. STG decomposition was first presented in [Chu87] for live and safe free-choice nets with injective labelling, and then generalised to STGs with arbitrary structure in [VW02, VK05].

This combined framework can cope with quite large specifications. It has been implemented using a number of tools:

PUNF — a tool for building unfolding prefixes of Petri nets [Kho03].

MPSAT — a tool for verification and synthesis of asynchronous circuits [KKY04, KKY06]; it uses unfolding prefixes built by PUNF.

DESIJ — a tool for decomposing an STG into smaller components [VW02, VK05, SVWK06]. It implements also the techniques of combining decomposition and unfolding presented in this paper and uses PUNF and MPSAT for synthesis of final components and for verification of some properties during decomposition.

2 Basic Definitions

In this section, we first present basic definitions concerning Petri nets and STGs, and then recall notions related to unfolding prefixes (see also [ERV02, Kho03, Mur89]).

2.1 Petri nets

A *net* is a triple $N \stackrel{\text{df}}{=} (P, T, W)$ such that P and T are disjoint sets of respectively *places* and *transitions*, and $W : (P \times T) \cup (T \times P) \to \mathbb{N} = \{0, 1, 2, \ldots\}$ is a *weight function*. A *marking* M of N is a multiset of places, i.e., $M : P \to \mathbb{N}$. We adopt the standard rules about drawing nets, viz. places are represented as circles, transitions as boxes, the weight function by arcs, and markings are shown by placing tokens within circles. In addition, the following short-hand notation is used: a transition can be connected directly to another transition if the place 'in the middle of the arc' has exactly one incoming and one outgoing arc (see, e.g., Figs. 1(a)). As usual, ${}^\bullet z \stackrel{\text{df}}{=} \{y \mid W(y, z) > 0\}$ and $z^\bullet \stackrel{\text{df}}{=} \{y \mid W(z, y) > 0\}$ denote the *pre-* and *postset* of $z \in P \cup T$, and we define ${}^\bullet Z \stackrel{\text{df}}{=} \bigcup_{z \in Z} {}^\bullet z$ and $Z^\bullet \stackrel{\text{df}}{=} \bigcup_{z \in Z} z^\bullet$, for all $Z \subseteq P \cup T$. We will assume 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* or *Petri net* is a tuple $\Sigma \stackrel{\text{df}}{=} (P, T, W, M_0)$ where (P, T, W) is a finite net and M_0 is an *initial* marking.

A transition $t \in T$ is *enabled* at a marking M, denoted $M[t\rangle$, if, for every $p \in {}^\bullet t$, $M(p) \geq W(p, t)$. Such a transition can be *fired*, leading to the marking M' with $M'(p) \stackrel{\text{df}}{=} M(p) - W(p, t) + W(t, p)$. We denote this by $M[t\rangle M'$. A finite or infinite sequence $\sigma = t_1 t_2 t_3 \ldots$ of transitions is a *firing sequence* of a marking M, denoted $M[\sigma\rangle$, if $M[t_1\rangle M'$ and $\sigma' = t_2 t_3 \ldots$ is a firing sequence of M'. Moreover, σ is a firing sequence of Σ if $M_0[\sigma\rangle$. If σ is finite, $M[\sigma\rangle M'$ denotes that σ is a firing sequence of M reaching the marking M'. A marking M' is *reachable from* M if $M[\sigma\rangle M'$ for some firing sequence σ. M is called *reachable* if it is reachable from M_0; $[M_0\rangle$ denotes the set of all *reachable markings* of Σ. Two transitions

t_1 and t_2 are in *(dynamic) conflict* if there is a reachable marking M, such that $M[t_1\rangle$, $M[t_2\rangle$ but for some place p, $W(p,t_1) + W(p,t_2) > M(p)$. A dynamic conflict implies a *structural conflict*, i.e. ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$.

A transition is *dead* if no reachable marking enables it. A transition is *live* if any reachable marking M enables a firing sequence containing it. (Note that being live is a stronger property than being non-dead.) A net system is called *live* if every of its transition is live; it is called *reversible* if the initial marking is reachable from every reachable marking.

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

2.2 Signal Transition Graphs

A *Signal Transition Graph (STG)* is a triple $\Gamma \stackrel{\text{df}}{=} (\Sigma, Z, \ell)$ such that Σ is a net system, Z is a finite set of signals, generating the finite alphabet $Z^\pm \stackrel{\text{df}}{=} Z \times \{+,-\}$ of *signal transition labels*, and $\ell : T \to Z^\pm \cup \{\lambda\}$ is a labelling function. The signal transition labels are of the form z^+ or z^-, and denote a transition of a signal $z \in Z$ from 0 to 1 (rising edge), or from 1 to 0 (falling edge), respectively. We will use the notation $z^\pm$ to denote a transition of signal z if we are not particularly interested in its direction. Γ inherits the operational semantics of its underlying net system Σ, including the notions of transition enabling and firing sequences, reachable markings and firing sequences.

We lift the notion of enabledness and firing to transition labels: $M[\ell(t)\rangle\rangle M'$ if $M[\sigma\rangle M'$. This is extended to sequences as usual – deleting λ-labels automatically since λ is the empty word. A sequence ω of elements of $Z^\pm$ is called a *trace of a marking* M of Γ if $M[\omega\rangle\rangle$, and a *trace* of Γ if it is a trace of M_0. The *language of* Γ is the set of all traces of Γ and denoted by $L(\Gamma)$. Γ has a *(dynamic) auto-conflict* if two transitions t_1 and t_2 with $\ell(t_1) = \ell(t_2) \neq \lambda$ can be in a dynamic conflict.

An STG may initially contain transitions labelled with λ called *dummy transitions*. They are a design simplification and describe no physical reality. Moreover, during the decomposition, certain transitions are labelled with λ at intermediate stages; this relabelling of a transition is called *lambdarising* a transition, and *delambdarising* means to change the label back to the initial value. The set of transitions labelled with a certain signal is frequently identified with the signal itself, e.g., lambdarising signal z means to change the label of all transitions labelled with $z^\pm$ to λ.

We associate with the initial marking of Γ a binary vector $v^0 \stackrel{\text{df}}{=} (v_1^0, \ldots, v_{|Z|}^0) \in \{0,1\}^{|Z|}$, where each v_i^0 corresponds to the signal $z_i \in Z$; this vector contains the initial value of each signal. Moreover, with any finite firing sequence σ of Γ we associate an integer *signal change vector* $v^\sigma \stackrel{\text{df}}{=} (v_1^\sigma, v_2^\sigma, \ldots, v_{|Z|}^\sigma) \in \mathbb{Z}^{|Z|}$, so that each v_i^σ is the difference between the number of the occurrences of z_i^+–labelled and z_i^-–labelled transitions in σ.

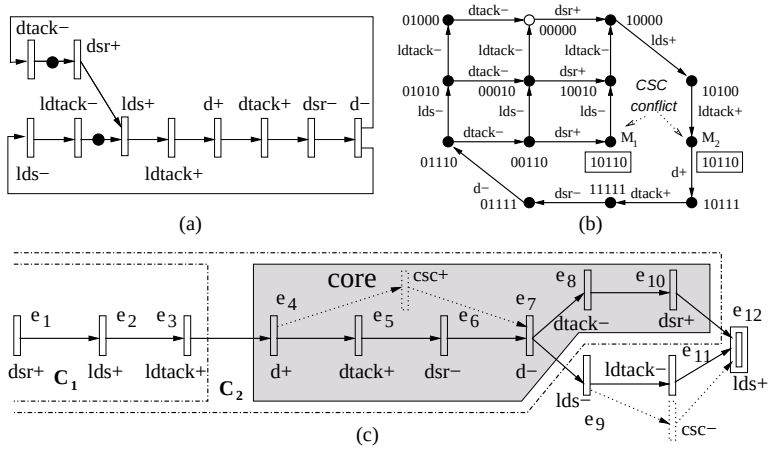
Γ is *consistent*[1] if, for every reachable marking M, all firing sequences σ from M_0 to M have the same *encoding vector* $Code(M)$ equal to $v^0 + v^\sigma$, and this vector is binary, i.e., $Code(M) \in \{0,1\}^{|Z|}$. Such a property guarantees that, for every signal $z \in Z$, the STG satisfies the following two conditions: (i) the first occurrence of z in the labelling of any firing sequence of Γ starting from M_0 has the same sign (either rising of falling); and (ii) the transitions corresponding to the rising and falling edges of z alternate in any firing sequence of Γ. In this paper it is assumed that all the STGs considered are consistent. (The consistency of an STG can easily be checked during the process of building its finite and complete prefix [Sem97].) We will denote by $Code_z(M)$ the component of $Code(M)$ corresponding to a signal $z \in Z$.

The *state graph* of Γ is a tuple $SG_\Gamma \stackrel{\text{df}}{=} (S, A, M_0, Code)$ such that: $S \stackrel{\text{df}}{=} [M_0\rangle$ is the set of *states*; $A \stackrel{\text{df}}{=} \{M \xrightarrow{\ell(t)} M' \mid M \in [M_0\rangle \wedge M[t\rangle M'\}$ is the set of *state transitions*; M_0 is the *initial state*; and $Code : S \to \{0,1\}^{|Z|}$ is the *state assignment* function, as defined above for markings.

The signals in Z are partitioned into input signals, Z_I, and output signals, Z_O (the latter may also include internal signals). Input signals are assumed to be generated by the environment, while output signals are produced by the logic gates of the circuit. For each signal $z \in Z_O$ we define

$$Out_z(M) \stackrel{\text{df}}{=} \begin{cases} 1 & \text{if } M[z^\pm\rangle\rangle \\ 0 & \text{otherwise.} \end{cases}$$

[1] This is a somewhat simplified notion of consistency; see [Sem97] for a more elaborated one, dealing also with certain pathological cases, which are not interesting in practice.



$$[\text{lds}] = \overline{\text{csc}} \cdot (\overline{\text{ldtack}} \cdot \text{dsr} + \text{lds}) + \text{d}$$
$$[\text{dtack}] = \text{d}$$
$$[\text{d}] = \text{lds} \cdot \overline{\text{csc}} \cdot \text{ldtack} + \text{d} \cdot \text{dsr}$$
$$[\text{csc}] = \text{d} + \text{lds} \cdot \text{csc}$$

(d)

inputs: *dsr*, *ldtack*; **outputs:** *lds*, *d*, *dtack*; **internal:** *csc*

Figure 1: VME bus controller: the STG for the read cycle (a), its state graph showing a CSC conflict (b), its unfolding prefix with the corresponding conflict core, and a way to resolve it by adding a new signal *csc* (c), and a complex-gate implementation (d). The signal order in binary encodings is: *dsr, dtack, lds, ldtack, d.*

Logic synthesis derives for each output signal $z \in Z_O$ a Boolean *next-state function* Nxt_z defined for every reachable state M of Γ as follows:

$$Nxt_z(M) \stackrel{\text{df}}{=} Code_z(M) \oplus Out_z(M) \ ,$$

where $\oplus$ is the 'exclusive or' operation.

The value of this function must be determined without ambiguity by the encoding of each reachable state, i.e., $Nxt_z(M)$ should be a function of $Code(M)$ rather than of M, i.e., $Nxt_z(M) = F_z(Code(M))$ for some function $F_z : \{0,1\}^{|Z|} \to \{0,1\}$ (F_z will eventually be implemented as a logic gate). To capture this, let M' and M'' be two distinct states of SG_Γ. M' and M'' are in *Complete State Coding (CSC) conflict* if $Code(M') = Code(M'')$ and $Out_z(M') \neq Out_z(M'')$ for some output signal $z \in Z_O$. Intuitively, a CSC conflict arises when semantically different reachable states of an STG have the same binary encoding. Γ satisfies the *CSC property* if no two states of SG_Γ are in *CSC* conflict. (Intuitively, this means that each output signal is implementable as a logic gate.)

An example of an STG for a data read operation in a simple VME bus controller (a standard STG benchmark, see, e.g., [CKK+02]) is shown in Figure 1(a). Part (b) of this figure shows a *CSC* conflict between two different states, M_1 and M_2, that have the same encoding, 10110, but $Nxt_d(M_1) = 0 \neq Nxt_d(M_2) = 1$ and $Nxt_{lds}(M_1) = 0 \neq Nxt_{lds}(M_2) = 1$. This means that the values of $F_d(1,0,1,1,0)$ and $F_{lds}(1,0,1,1,0)$ are ill-defined (they should be 0 according to M_1 and 1 according to M_2), and thus these signals are not implementable as logic gates.

2.3 Unfolding prefixes

A *finite and complete unfolding prefix* of an STG Γ is a finite acyclic net which implicitly represents all the reachable states of Γ together with transitions enabled at those states. Intuitively, it can be obtained through *unfolding* Γ, by successive firings of transitions, under the following assumptions: (i) for each new firing a fresh transition (called an *event*) is generated; (ii) for each newly produced token a fresh place (called a *condition*) is generated.

The unfolding is infinite whenever Γ has an infinite run; however, if Γ has finitely many reachable states then the unfolding eventually starts to repeat itself and can be truncated (by identifying a set of *cut-off* events) without loss of information, yielding a finite and complete prefix. Fig. 1(c) shows a finite and complete unfolding prefix (with the only cut-off event depicted as a double box) of the STG shown in Fig. 1(a).

Efficient algorithms exist for building such prefixes [ERV02, Kho03], which ensure that the number of non-cut-off events in a complete prefix can never exceed the number of reachable states of Γ. However, 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.

Since practical STGs usually exhibit a lot of concurrency, but have rather few choice points, their unfolding prefixes are often exponentially smaller than the corresponding state graphs; in fact, in many of the experiments conducted in [Kho03, KKY04] they were just slightly bigger then the original STGs themselves. Therefore, unfolding prefixes are well-suited for both visualisation of an STG's behaviour and alleviating the state space explosion problem.

3 Unfolding-based synthesis

Due to its structural properties (such as acyclicity), the reachable states of an STG can be represented using *configurations* of its unfolding. A configuration C is a downward-closed set of events (being downward-closed means that if $e \in C$ and f is a causal predecessor of e then $f \in C$) without *choices* (i.e., for all distinct events $e, f \in C$, there is no condition c in the unfolding such that the arcs (c, e) and (c, f) are in the unfolding). Intuitively, a configuration is a partially ordered firing sequence, i.e., a firing sequence where the order of firing of some of its events (viz. concurrent ones) is not important.

A CSC conflict can be represented in the unfolding prefix as an unordered *conflict pair* of configurations $\langle C_1, C_2 \rangle$ whose final states are in CSC conflict, as shown if Fig. 1(c). It was shown in [KKY04] that the problem of checking if there is such a conflict pair is reducible to SAT, and an efficient technique for finding all CSC conflict pairs was proposed.

Let $\langle C_1, C_2 \rangle$ be a conflict pair of configurations. The corresponding *complementary set* $\mathcal{CS}$ is defined as the symmetric set difference of C_1 and C_2. $\mathcal{CS}$ is a *core* if it cannot be represented as the union of several disjoint complementary sets. For example, the core corresponding to the conflict pair shown in Fig. 1(c) is $\{e_4, \ldots, e_8, e_{10}\}$ (note that if $C_1 \subset C_2$ then the corresponding complementary set is simply $C_2 \setminus C_1$).

One can show that every complementary set $\mathcal{CS}$ can be partitioned into $C_1 \setminus C_2$ and $C_2 \setminus C_1$, where $\langle C_1, C_2 \rangle$ is a conflict pair corresponding to $\mathcal{CS}$. Moreover, if $C_1 \subset C_2$ then one of these parts is empty, while the other is $\mathcal{CS}$ itself. An important property of complementary sets is that for each signal $z \in Z$, the differences between the numbers of z^+– and z^-–labelled events are the same in these two parts (and are 0 if $C_1 \subset C_2$). This suggests that a complementary set can be eliminated (resolving thus the corresponding encoding conflicts), e.g., by introduction of a new *internal* signal, csc^+, and insertion of its transition into this set, as these would violate the stated property. (Note that the circuit has to implement this new signal, and so for the purpose of logic synthesis it is regarded as an output, though it is ignored by the environment.) To preserve the consistency of the STG, the transition's counterpart, csc^-, must also be inserted *outside the core,* in such a way that it is neither concurrent to nor in structural conflict with csc^+. Another restriction is that an inserted signal transitions must not trigger an input signal transition (the reason is that this would impose constraints on the environment which were not present in the original STG, making it 'wait' for the newly inserted signal). Intuitively, insertion of signals introduces additional memory into the circuit, helping to trace the current state.

The core in Fig. 1(c) can be eliminated by inserting a new signal, csc^+, somewhere in the core, e.g., concurrently to e_5 and e_6 between e_4 and e_7, and by inserting its complement outside the core, e.g., concurrently to e_{11} between e_9 and e_{12}. (Note that the concurrent insertion of these two transitions avoids an increase in the latency of the circuit, where each transition is assumed to contribute a unit delay.) After transferring this signal into the STG, it satisfies the CSC property.

It is often the case that cores overlap. In order to minimise the number of performed transformations, and thus the area and latency of the circuit, it is advantageous to perform such a transformation that as many cores as possible are eliminated by it. That is, a transformation should be performed in the *intersection of several cores* whenever possible.

This idea can be implemented by means of a *height map* showing the quantitative distribution of the cores. Each event in the prefix is assigned an *altitude*, i.e., the number of cores it belongs to. (The analogy with a topographical map showing the altitudes may be helpful here.) 'Peaks' with the highest altitude are good candidates for insertion, since they correspond to the intersection of maximum number of cores. This unfolding-based method for the resolution of encoding conflicts was presented in [MBKY03].

Once the CSC conflicts are resolved, one can derive equations for logic gates of the circuit, as illustrated

in Fig. 1(d). An unfolding-based approach to this problem has been presented in [KKY06]. The main idea of this approach was to generate the truth table for each such equation as a projection of a set of reachable encodings to some chosen support, which can be accomplished with the help of the incremental SAT technique, and then applying the usual Boolean minimisation to this table.

The results in [KKY04, MBKY03, KKY06] form a complete design flow for complex-gate synthesis of asynchronous circuits based on STG unfoldings rather than state graphs, and the experimental results conducted there show that it has significant advantage both in memory consumption and in runtime, without affecting the quality of the solutions.

4 STG Decomposition

In this section, the STG decomposition algorithm of [VW02, VK05] is outlined, in order to understand the new contributions properly.

Synthesis with STG decomposition works roughly as follows. Given a consistent STG Γ, an initial partition $(In_i, Out_i)_{i \in I}$ of its signals is chosen, satisfying the following properties.

- If two output signals x_1, x_2 are in structural conflict in Γ, then they have to be in the same Out_i.
- If there are $t, t' \in T$ with $t' \in (t^\bullet)^\bullet$ (t is called *syntactical trigger of* t'), then $\ell(t') \in Out_i$ implies $\ell(t) \in In_i \cup Out_i$.

Then the algorithm decomposes Γ into component STGs, one for each set in this partition, together implementing Γ. Each component is obtained from the original STG by lambdarising the signals which are not in the corresponding partition, and then contracting the corresponding transitions (some other net reductions are also applied — see below). Then, from each component a circuit is synthesised, and these circuits together implement the original specification.

Of course, the decomposition must preserve the behaviour of the specification in some sense. In [VW02, VK05, SV05], the correctness was defined as a variation of bisimulation, tailored to the specific needs of asynchronous circuits, called *STG-bisimulation*.

Typically, the computational effort (in terms of memory consumption and runtime) needed to synthesise a circuit from an STG Γ is exponential in the size of Γ. Hence, if the components produced by the decomposition algorithm are smaller than Γ, the decomposition can be seen as successful.

We now describe the operations which the algorithm applies to an initial component until no more λ-labelled transitions remain.

Contraction of a λ-labelled transition. Transition contraction can be applied to a λ-labelled transition t if ${}^\bullet t \cap t^\bullet = \emptyset$ and for all place p $W(t,p), W(p,t) \leq 1$; it is illustrated in Figure 2. Intuitively, t is removed from the net, together with its surrounding places ${}^\bullet t \cup t^\bullet$, and the new places, corresponding to the elements of ${}^\bullet t \times t^\bullet$, are added to the net. Each new place $(p,q) \in {}^\bullet t \times t^\bullet$ inherits the connectivity of both p and q (except that t is no longer in the net), and its initial marking is the total number of tokens which were initially present in p and q. (The formal definition of transition contraction can be found in the appendix.)

The contraction is called *secure* if either $({}^\bullet t)^\bullet \subseteq \{t\}$ (*type-1 secure*) or ${}^\bullet(t^\bullet) = \{t\}$ and $M_0(p) = 0$ for some $p \in t^\bullet$ (*type-2 secure*). It is shown in [VW02, VK05] that secure contractions of λ-labelled transitions preserve the language of the STG.

Deletion of an implicit place. It is often the case that after a transition contraction implicit places (i.e., places which can be removed without changing the firing sequences of the net) are produced. Such places may prevent further transition contractions, and should be deleted before the algorithm proceeds.

Deletion of a redundant transition. There are two kinds of *redundant transitions*. First, if there are two transitions with the same label which are connected to every place in the same way, one of them can be deleted without changing the traces of the STG. Second, a λ-labelled transition t with ${}^\bullet t = t^\bullet$ can also be deleted, since its firing does not change the marking and is not visible on the level of traces; observe, that this is valid for any marking of the adjacent places.

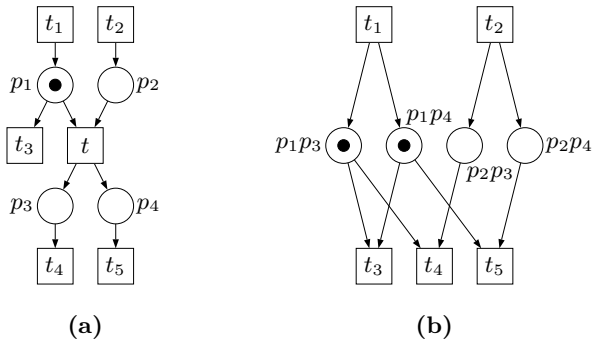


Figure 2: Transition contraction: the initial net **(a)**, and the net after contraction of t **(b)**.

Backtracking. As it was already mentioned, not every λ-labelled transition can be contracted by the decomposition algorithm. There are three reasons for this:

- The contractions is not defined (e.g., because ${}^\bullet t \cap t^\bullet \neq \emptyset$).
- The contraction is not *secure* (then the language of the STG might change).
- The contraction introduces a new auto-conflict (i.e., a new potential source of non-determinism which was not present in the specification is introduced; this is interpreted that the component has not enough information (viz. input signals) to properly produce its outputs).

If none of the described reduction operations are applicable, but the component still has some λ-labelled transitions, *backtracking* is applied, i.e., one of these λ-labelled transitions is chosen and the corresponding signal is *delambdarised*, i.e., this input is added to the initial partition and the new corresponding initial component is derived and reduced from the beginning. This cycle of reduction and backtracking is repeated until all λ-labelled transitions of the initial component can be contracted. This means that backtracking is only needed to detect these additional input signals; if they are known in advance, one can perform decomposition completely without backtracking. (In the worst case, all the lambdarised signals are delambdarised.)

The described decomposition algorithm is non-deterministic, i.e., it can apply the net reductions in any order; the result has been proven to be always correct. In [SVWK06], different ways to determinise it are described. One of them was *tree decomposition*, which greatly improves the overall efficiency of decomposition process by re-using intermediate results. Since it is the base for CSC-aware decomposition introduced below, we describe it briefly.

4.1 Tree Decomposition

In our experiments, it turned out that in most cases some initial components have many lambdarised signals in common. Therefore, the decomposition algorithm can save time by building an intermediate STG C', from which these components can be derived: instead of reducing both initial components independently, it is sufficient to generate C' only once and to proceed separately with each component afterwards, thus saving a lot of work.

Tree decomposition tries to generate a plan which minimises the total amount of work using the described idea. We introduce it by means of an example in Figure 3. Let Γ be an STG with the signal set $\{1, 2, 3, 4, 5\}$. Furthermore, let there be three components C_1, C_2, C_3, and let $\{1, 2, 3\}$, $\{2, 3, 4\}$, $\{3, 4, 5\}$ be the signals which are lambdarised initially in these components. We build a tree guiding the decomposition process, such that its leafs correspond to the final components, and every node u is labelled with the set of signals $s(u)$ to be contracted.

In (a) the initial situation is depicted. There are three independent leaves labelled with the signals which should be contracted to get the corresponding final component. A possible intermediate STG C' for C_1 and C_2 would be the STG in which signals 2 and 3 have been contracted. In (b), C' is introduced as an common intermediate result for C_1 and C_2; the signals 2 and 3 no longer have to be contracted in C_1 and C_2 (they appear in brackets) and the leaves are labelled with $\{1\}$ and $\{4\}$, respectively. In (c), a

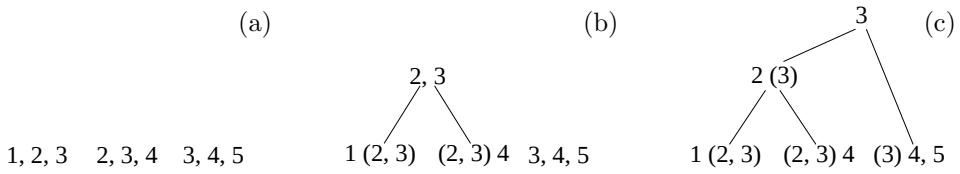


Figure 3: Building of a simple decomposition tree for three components and five signals. Leafs from the left: components C_1, C_2, C_3. (a) the initial situation; (b) two components merged; (c) the final decomposition tree.

common intermediate result for C' and C_3 with the label $\{3\}$ is added, yielding the final decomposition tree.

From this use of a decomposition tree, it is clear that in an optimal decomposition tree the sum of all $|s(u)|$ should be minimal. Decomposition trees are very similar to *preset trees* in [KK01]; there it is shown that computing an optimal preset tree is NP-complete, and a heuristic algorithm is described which performs reasonably well. We use this algorithm for the automatic calculation of decomposition trees.

The decomposition algorithm guided by such a decomposition tree traverses it in the depth-first order. It enters the root node with the initial STG Γ containing no lambdarised signals. Upon entering a node u with an STG Γ_u, the algorithm lambdarises and contracts the signals $s(u)$ in Γ_u (and performs other possible reductions) and enters each child node with its own copy of the resulting STG.[2] If u is a leaf, the resulting STG is a final component.[3]

4.2 CSC-Aware decomposition

On the basis of tree decomposition, we now introduce *CSC-aware* decomposition. Our aim is to reduce the number of CSC conflicts in the components generated by the decomposition algorithm. Ideally, if the original specification is free from CSC conflicts then this should be the case also for the components.

During its execution the algorithm has to determine if an STG has CSC conflicts. This is checked externally with the tools PUNF and MPSAT [Kho03, KKY04]. It works essentially as tree decomposition, with the following differences, cf. Figure 4. When a leaf is reached, we check whether the corresponding final component has CSC conflicts. If no, the component is saved as the final result. Otherwise, for each detected CSC core a constituting pair of firing sequences leading to the conflicting states is stored in the parent of the leaf.

When the algorithm returns to this parent node, it checks whether this CSC conflict is still present in the local intermediate STG. However, using MPSAT may be expensive at this point, as the corresponding STG is larger than the final component. Instead, we map the stored firing sequences from the final component to this STG using the *inverse projections* introduced below, and check if they still lead to states which are in a CSC conflict. For every conflict which is not destroyed, this results in a new pair of firing sequences which is propagated upwards in the tree, and so on. On the other hand, if the conflict disappears, these inverse projections are analysed as described below, and signals which helped to resolve the conflict are determined and delambdarised in the corresponding child node, and the algorithm tries to process it again. If no CSC conflicts remain in the final component (due to the delambdarised signals), it is saved as the final result.

When all pairs of firing sequences corresponding to CSC conflicts are considered, the algorithm proceeds with the next child of the current node. If there are no more children left, it goes up to the parent of the current node and deals with the corresponding firing sequences. Eventually, the algorithm reaches the root node for the last time and terminates.

This algorithm is complete, i.e., it guarantees for a specification with CSC that each component has CSC, too. This is due to the fact that a pair of firing sequences corresponding to a CSC conflict in a

[2] As an important technical improvement, the intermediate result of a component is not copied for each child. Instead, throughout the decomposition, a single STG is used, and an *undo stack* is used to restore the 'parent' STG whenever the algorithm returns to the parent node. This is much faster and uses far less memory than keeping multiple (and potentially large) STGs.

[3] There are some twists in this setting considering backtracking, which is handled a bit different in contrast to 'ordinary' decomposition; in particular, the decomposition tree can be modified during the decomposition process, cf. [SVWK06].

component can be moved up to the root node (via a sequence of inverse projections), where CSC is given initially. In practice, one can stop moving up a pair of firing sequences after several iterations and try to resolve CSC conflicts with new internal signals instead. Therefore, the algorithm is still applicable to specifications which have CSC conflicts initially.

The *inverse projection* of a firing sequence is defined as follows. Let Γ and Γ' be two STGs such that Γ' is obtained from Γ by a secure contraction of some transition t. If σ' is a firing sequence of Γ', we call a firing sequence σ of Γ an inverse projection of σ' if σ' is the projection of σ on the transitions of Γ'.

Since the contraction of t was secure, the inverse projection can be calculated easily: it is enough to fire the transitions of σ' in Γ, one by one, while possible. If, at some point, a transition of σ' cannot be fired then t is fired (it is guaranteed to be enabled in such a case). This process is continued until all the transitions of σ' are fired, yielding σ. One can see that though the inverse projection of a given firing sequence is generally not unique, the shortest inverse projection is unique, and it is precisely the one computed by the described procedure. If Γ' is obtained from Γ by a sequence of secure contractions, its firing sequences can still be inversely projected to Γ by computing a sequence of inverse projections for each individual contraction.

If Γ' has a CSC conflict, there is a corresponding pair of firing sequences (σ'_1, σ'_2) such that the corresponding signal change vectors $v^{\sigma'_1}$ and $v^{\sigma'_2}$ coincide. If the inverse projection (σ_1, σ_2) of this pair is such that $v^{\sigma_1} \neq v^{\sigma_2}$ then the corresponding conflict is likely to be destroyed by delambdarising the signal corresponding to the contracted transition.

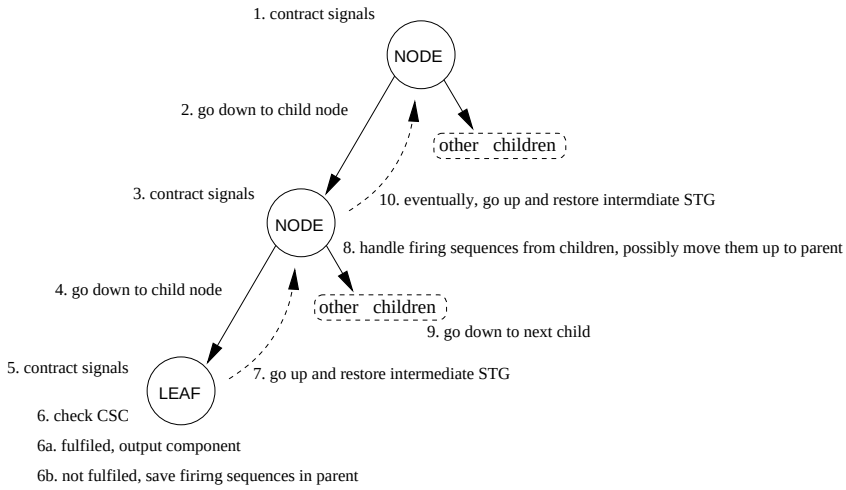


Figure 4: Outline for CSC-aware decomposition. Step 8 is repeated every time a node is entered from a child, step 9 includes contraction and detection of CSC.

5 Combining Decomposition and Unfolding Techniques

In this section we describe how our unfolding and decomposition tools can be used to combine their advantages and to compensate for each other's shortcomings. PUNF and MPSAT can perform logic synthesis, but not for very large STGs. On the other hand, DESIJ can handle very large STGs quite efficiently because it performs only local structural operations, but it has to make conservative assumptions frequently to guarantee correctness.

The strategy we adopted is as follows. While the STGs are large, only structural conservative checks are made, as it may be computationally very expensive to perform the exact tests. After some reductions have been performed, it becomes feasible to check exact reachability-like properties using PUNF and MPSAT (logic synthesis is still not feasible at this stage). Eventually, when the components are small enough, logic synthesis is performed.

While DESIJ can handle and produce non-safe nets, PUNF and MPSAT need safe nets. Therefore, we accept only safe nets as specifications (which is no serious restriction) and perform only *safeness-preserving* contractions during decomposition. They are discussed in the following subsection.

During the decomposition process the decomposition algorithm checks from time to time the following reachability-like properties:

- The decomposition algorithm should backtrack if a new dynamic auto-conflict is produced. The corresponding conservative test is the presence of a new structural auto-conflict.
- It is also helpful to remove implicit places. The corresponding conservative test looks for *redundant places* [Ber87]; they are defined by a system of linear inequalities. Checking this condition with a linear programm solver is also quite expensive, and therefore DesiJ looks only for a subset called *shortcut places* [SVJ05].
- In order to apply Mpsat, the STG must be safe. In general, a transition contraction can transform a safe STG into an non-safe (2-bounded) one. The corresponding conservative structural conditions guaranteeing that a contraction preserves safeness are developed below.

All of the mentioned dynamic properties can be checked with a reachability analysis, which can be performed by Mpsat. Since we only consider safe nets here, reachability-like properties can be expressed as Boolean expressions over the places of the net. For example, the property $p_1 \wedge p_2 \wedge \neg p_3$ holds iff some reachable marking has a token on p_1 and p_2 and no token on p_3. (Such properties can be checked by Mpsat.) Below we give Boolean expressions and the corresponding conservative tests for the properties listed above. All the necessary proofs can be found in the appendix.

Safeness-preserving contractions

A transition contraction preserves boundedness, but, in general, it can turn a safe net into a non-safe one, as well as introduce duplicate (weighted) arcs. However, since unfolding techniques are not very efficient for non-safe net, we assume that the initial STG is safe, and perform only *safeness-preserving* contractions, i.e., ones which guarantee that if the initial STG was safe then the transformed one is also safe. (Note that the transitions with duplicate (weighted) arcs must be dead in a safe Petri net, and so we can assume that the initial and all the intermediate STGs contain no such arcs.)

We now give a sufficient structural condition for a contraction being safeness-preserving. Then we will show how this can be checked with a partial reachability analysis and also how a single unfolding prefix can be used for checking if each contraction in a sequence of contractions is safeness-preserving.

Theorem 5.1 (Structural safeness-preservation)
A secure contraction of a transition t in a net Γ is safeness-preserving if

1) $|{}^\bullet t| = 1$ *or*

2) $|t^\bullet| = 1$, ${}^\bullet(t^\bullet) = \{t\}$ *and*

- *a)* Γ *is live and reversible*
  or
- *b)* $M_0(p) = 0$ *with* $t^\bullet = \{p\}$

Figure 5 shows two counterexamples: the leftmost net violates the condition that either the pre- or postset of t has to contain a single place; one can see that, the contraction of t generates an non-safe net. The net in the middle violates the condition ${}^\bullet(t^\bullet) = \{t\}$ in the second case in Theorem 5.1 (i.e., that the place in the postset of t must not have incoming arcs other than from t); the rightmost net is obtained by contracting t in the net in the middle.

In practice, the decomposition algorithm checks the condition 2*b*) which makes no assumptions about the net which are difficult to verify. This is important since there exist STGs which are neither live nor reversible, e.g., ones which have some initialisation part which is executed only once in the beginning.

If the specification is guaranteed to be live and reversible, it is also possible to use condition 2*a*); then the following lemma is needed to apply such contractions repeatedly.

Proposition 5.2
Secure transition contractions and implicit place deletions preserve liveness and reversibility.

So far, we only considered structural conditions for a contraction to be safeness-preserving; now we describe the dynamic conditions.

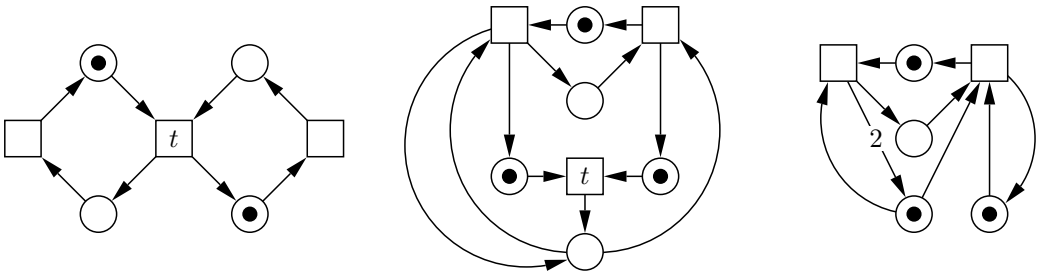


Figure 5: Examples of non-safeness-preserving contractions.

Theorem 5.3
Let Γ be a safe STG and $t \in T$ such that the contraction of t is secure. The contraction of t is safeness-preserving iff the following property does not hold:

$$\left(\bigvee_{p \in {}^\bullet t} p\right) \wedge \left(\bigvee_{p \in t^\bullet} p\right) .$$

To check these reachability properties with MPSAT one has to generate the unfolding prefix with PUNF first, which can take considerable time. It is therefore impractical to generate it for checking the safeness-preservation of a single contraction. Instead, our algorithm uses a single unfolding prefix to check if a sequence of several subsequent contractions is safeness-preserving. This technique is explained in the appendix.

Implicit Places

As it was already mentioned, the deletion of implicit places is important for the success of the decomposition. As a conservative condition, DESIJ looks for shortcut places. On the other hand, unfolding-based reachability analysis makes it possible to check exactly whether a place is implicit: a place p is implicit iff the following property does not hold:

$$\neg p \wedge \left(\bigvee_{t \in p^\bullet} \bigwedge_{q \in {}^\bullet t \setminus \{p\}} q\right) .$$

It is possible to detect all implicit place of a net with a single unfolding. Observe first, that the deletion of an implicit place cannot turn an non-implicit place into an implicit one. Indeed, suppose p_1 is implicit and deleted in Σ, yielding Σ_1, and p_2 is implicit and deleted in Σ_1, yielding Σ_2. Then $FS(\Sigma) = FS(\Sigma_1) = FS(\Sigma_2)$ by definition of implicit places, where $FS(\Sigma)$ denotes the set of all firing sequences of Σ. Suppose now that p_2 is deleted first in Σ, yielding Σ_1', and p_1 is deleted in Σ_1', yielding Σ_2 again. Then $FS(\Sigma) \subseteq FS(\Sigma_1') \subseteq FS(\Sigma_2) = FS(\Sigma)$, since deleting places can only increase the set of firing sequences. Therefore $FS(\Sigma) = FS(\Sigma_1') = FS(\Sigma_2)$, which shows that p_2 is implicit in Σ. It is therefore sufficient to iterate once over all places and to delete every implicit one.

Furthermore, the unfolding of a net in which an implicit places was deleted can be obtained from the original unfolding by deleting all occurrences of this place. For the above reachability analysis we get the same effect automatically, because deleted places will not occur in the corresponding property.

Dynamic Auto-Conflicts

A conservative test for the presence of an auto-conflict is the presence of two transitions with the same label (distinct from λ) and overlapping presets. Unfolding-based reachability analysis makes it possible to check exactly for the presence of an auto-conflict as follows.

In a safe STG distinct transitions t_1 and t_2 such that ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$ are in dynamic conflict iff the following property holds:

$$\bigwedge_{p \in {}^\bullet t_1 \cup {}^\bullet t_2} p .$$

Using this exact test can reduce the number of times the decomposition algorithm has to backtrack, which ultimately can result in the improved runtime and smaller final components.

6 Results

We applied the described combined approach to several benchmark examples with and without CSC conflicts, and compared the results with the stand-alone synthesis with MPSAT and PETRIFY. (The tool for CSC conflict resolution and decomposition described in [CC06, Car03] was not available from the authors.) In the tables, all times are given as (minutes:)seconds. The benchmarks were performed on a Pentium 4 HT with 3 GHz and 2 GB RAM.

We worked with two types of benchmarks. The first group are pipelines which have CSC initially. As expected, the new approach produces components without CSC conflicts, i.e., the signals which are necessary for preventing CSC conflicts are kept in the components (the original approach of [VW02, VK05, SVWK06] would have contracted some of them).

Our combined approach decomposes and synthesises these benchmarks (Table 1) quite quickly compared with PETRIFY (aborted after 5 minutes). However, MPSAT alone is much faster for these examples and needs less than a second for any of them. This is because these benchmarks are relatively small, with up to 257 nodes and up to 43 signals.

Benchmark	DESIJ	PETRIFY
2PP.ARB.NCH.03.CSC	1	1
2PP.ARB.NCH.06.CSC	2	14
2PP.ARB.NCH.09.CSC	4	1:54
2PP.ARB.NCH.12.CSC	10	32:55
2PP-WK.03.CSC	1	1
2PP-WK.06.CSC	2	9
2PP-WK.09.CSC	3	31
2PP-WK.12.CSC	18	24:36
3PP.ARB.NCH.03.CSC	1	4
3PP.ARB.NCH.06.CSC	3	2:14
3PP.ARB.NCH.09.CSC	7	84:17
3PP.ARB.NCH.12.CSC	22	$\geq$ 360:00
3PP-WK.03.CSC	1	1
3PP-WK.06.CSC	3	31
3PP-WK.09.CSC	7	34:08
3PP-WK.12.CSC	22	$\geq$ 360:00

Table 1: Results of the pipeline benchmarks.

The second group of benchmarks are newly generated; they are STGs derived from BALSA specifications. These kind of benchmarks was used before by [CC06]. The benchmark SEQPARTREE(21,10) from there is nearly the same as SEQPARTREE-05 here; the difference is that we did not hide the internal handshake signals. However, this is also possible for our approach and will most likely lead to further speedups, as discussed in Section 7.

These examples are generated out of two basic BALSA handshake components (see [EB02]): the 2-way *sequencer*, which performs two subsequent handshakes on its two 'child' ports when activated on its 'parent' port, and the 2-way *paralleliser*, which performs two parallel handshakes on its two 'child' ports when activated on its 'parent' port; either can be described by a simple STG. The benchmark examples SEQPARTREE-N are complete binary trees with alternating levels of sequencers and parallelisers, as illustrated in Figure 6 (N is the height of the tree), which are generated by the parallel composition of the elementary STGs corresponding to the individual sequencers and parallelisers in the tree. We also worked with other benchmarks made of handshake components (e.g., trees of parallelisers only); the results did not differ much, so we considered exemplarily only SEQPARTREE-N.

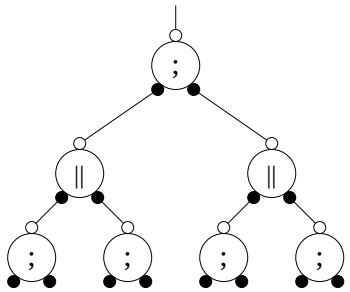


Figure 6: SeqParTree-03. Filled dots denote active handshake ports (they can start a handshake), blank nodes denote passive ones. Each port is implemented by two signals, an input and an output. If two ports are connected the parallel composition merges these four signals into two outputs.

These benchmarks have CSC conflicts initially, and Mpsat was used in the end to resolve them in each component separately. The experimental results in Table 2 show the real power of our method. The corresponding STGs are very large, and we consider it as a important achievement that the proposed combined approach could synthesise them so quickly. As one can see, an STG with more than 4000 signals is synthesised in less than 70 minutes. Petrify and Mpsat alone need more than 12 hours (aborted) for either of these benchmarks.

	Size	Signals	Combined		
Benchmark	$\|P\| - \|T\|$	$\|In\| - \|Out\|$	Deco.	Synthesis	Σ
SeqParTree-05	382 – 252	33 – 93	2	1	3
SeqParTree-06	798 – 508	65 – 189	4	2	6
SeqParTree-07	1566 – 1020	129 – 381	10	4	14
SeqParTree-08	3230 – 2044	257 – 765	48	10	57
SeqParTree-09	6302 – 4092	513 – 1533	4:55	24	5:19
SeqParTree-10	12958 – 8188	1025 – 3069	68:09	1:39	69:48

Table 2: Results of the handshake benchmarks.

In contrast to the decomposition method of [CC03,CC06] we allow components with more than output. This was utilised here: the initial partition was chosen such that each component of the decomposition corresponds to one handshake component. Other partitions of the outputs might lead to further speedups.

7 Conclusion

The purely structural decomposition approach of [VW02,VK05,SVWK06] can handle large specifications, but it does not take into account the properties of STGs related to synthesisability, such as the presence of CSC conflicts. In contrast, Mpsat can resolve CSC conflicts and perform logic synthesis, but it is inefficient for large specifications. In this paper, we demonstrated how these two methods can be combined to synthesise large STGs very efficiently.

One of the main technical contributions was to preserve the safeness of the STGs throughout the decomposition, because Mpsat can only deal with safe STGs. This is not just an implementation issue or a compensation for a missing Mpsat feature, but it is also far more efficient than working with non-safe nets, for which unfolding techniques seem to be inefficient. We also showed how dynamic properties like implicitness and auto-conflicts can be checked with unfoldings and how these checks can be combined with cheaper conservative structural conditions.

Future research is required for the calculation of the decomposition tree, the size of which is cubic in the number of signals and exceeds the memory usage for decomposition and synthesis by far. Here, heuristics are needed which explore the tradeoff between the quality of the decomposition tree and the amount of memory needed for its calculation.

Furthermore, we consider the handling of handshake based STGs as very important. Handshake circuits allow to synthesise very large specifications at the expense of a heavy overencoding of the resulting circuit, i.e., they have a lot of unnecessary state-holding elements, which increase the circuit area and latency. Decomposition can help here in the following way: instead of synthesising each handshake component separately, one can combine several such components, e.g., as it was done for SeqParTree-N, hide the internal communication signals and synthesise one circuit implementing the combination of the components using the proposed combined approach.

Acknowledgements

We would like to thank Dominic Wist for helping us with generating the benchmarks. This research was supported by DFG-projects 'STG-Dekomposition' Vo615/7-1 and Wo814/1-1, and the Royal Academy of Engineering/Epsrc grant EP/C53400X/1 (Davac).

References

[Ber87] G. Berthelot. Transformations and decompositions of nets. In W. Brauer et al., editors, *Petri Nets: Central Models and Their Properties*, Lect. Notes Comp. Sci. 254, 359–376. Springer, 1987.

[Ber93] K. v. Berkel. Handshake Circuits: an Asynchronous Architecture for VLSI Programming. *International Series on Parallel Computation*, 5, 1993.

[Bry86] R.E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35-8:677–691, 1986.

[Car03] Josep Carmona. *Structural Methods for the Synthesis of Well-Formed Concurrent Specifications*. PhD thesis, Universitat Politècnica de Catalunya, 2003.

[CC03] J. Carmona and J. Cortadella. ILP models for the synthesis of asynchronous control circuits. In *Proc. of the IEEE/ACM International Conference on Computer Aided Design*, pages 818–825, 2003.

[CC06] J. Carmona and J. Cortadella. State encoding of large asynchronous controllers. In *DAC 2006*, pages 939–944, 2006.

[Chu87] T.-A. Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications*. PhD thesis, MIT, 1987.

[CKK+97] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. PETRIFY: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Trans. Information and Systems*, E80-D, 3:315–325, 1997.

[CKK+02] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. *Logic Synthesis of Asynchronous Controllers and Interfaces*. Springer, 2002.

[EB02] D. Edwards and A. Bardsley. BALSA: an Asynchronous Hardware Synthesis Language. *The Computer Journal*, 45(1):12–18, 2002.

[ERV02] J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. *Formal Methods in System Design*, 20(3):285–310, 2002.

[ITR05] International technology roadmap for semiconductors: Design, 2005.
URL: `www.itrs.net/Links/2005ITRS/Design2005.pdf`.

[Kho03] V. Khomenko. *Model Checking Based on Prefixes of Petri Net Unfoldings*. PhD thesis, School of Computing Science, University of Newcastle upon Tyne, 2003.

[KK01] V. Khomenko and M. Koutny. Towards an efficient algorithm for unfolding Petri nets. In K.G. Larsen and M. Nielsen, editors, *CONCUR 2001*, Lect. Notes Comp. Sci. 2154, 2001.

[KKY04] V. Khomenko, M. Koutny, and A. Yakovlev. Detecting state coding conflicts in STG unfoldings using SAT. *Fundamenta Informaticae*, 62(2):1–21, 2004.

[KKY06] V. Khomenko, M. Koutny, and A. Yakovlev. Logic synthesis for asynchronous circuits based on Petri net unfoldings and incremental SAT. *Fundamenta Informaticae*, 70(1–2):49–73, 2006.

[MBKY03] A. Madalinski, A. Bystrov, V. Khomenko, and A. Yakovlev. Visualization and Resolution of Coding Conflicts in Asynchronous Circuit Design. *IEE Proceedings: Computers & Digital Techniques*, 150(5):285–293, 2003.

[Mur89] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proc. of the IEEE*, 77(4):541–580, 1989.

[Sem97] A Semenov. *Verification and Synthesis of Asynchronous Control Circuits Using Petri Net Unfolding*. PhD thesis, University of Necastle upon Tyne, 1997.

[SV05] M. Schaefer and W. Vogler. Component refinement and CSC solving for STG decomposition. In Vladimiro Sassone, editor, *FOSSACS 05*, Lect. Notes Comp. Sci. 3441, pp. 348–363. Springer, 2005.

[SVJ05] M. Schaefer, W. Vogler, and P. Jančar. Determinate STG decomposition of marked graphs. In G. Ciardo and P. Darondeau, editors, *ATPN 05*, Lect. Notes Comp. Sci. 3536, 365–384. Springer, 2005.

[SVWK06] M. Schaefer, W. Vogler, R. Wollowski, and V. Khomenko. Strategies for optimised STG decomposition. In *Proceedings of ACSD*, 2006.

[Val98] A. Valmari. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lect. Notes Comp. Sci.*, chapter The State Explosion Problem, pages 429–528. Springer-Verlag, 1998.

[VK05] W. Vogler and B. Kangsah. Improved decomposition of signal transition graphs. In *ACSD 2005*, pages 244–253, 2005.

[VW02] W. Vogler and R. Wollowski. Decomposition in asynchronous circuit design. In J. Cortadella et al., editors, *Concurrency and Hardware Design*, Lect. Notes Comp. Sci. 2549, 152 – 190. Springer, 2002.

Appendix

A The proofs

In this section we give the formal proofs of the results in this paper. Also, we provide some auxiliary definitions and results needed in these proofs. Throughout this section, we talk about net systems with labelling. Formally this is only defined for STGs, but all of the results here (and the ones cited from [VW02, VK05]) are also true for arbitrary net systems with arbitrary labelling, which is actually used for some proofs.

We start with the formal definition of a transition contraction.

Definition A.1 (Transition Contraction)
Let Γ be an STG and $t \in T$ with $\ell(t) = \lambda$. If ${}^\bullet t \cap t^\bullet = \emptyset$ and for all $p \in P$: $W(p,t), W(t,p) \leq 1$, the *contraction* of t results in the net Γ' with the same signals and:

$$\begin{array}{rcll} T' & \stackrel{\text{df}}{=} & T - \{t\} & \\ P' & \stackrel{\text{df}}{=} & \{(p,\star) \mid p \in P - ({}^\bullet t \cup t^\bullet)\} + {}^\bullet t \times t^\bullet & \\ W'((p,q),t) & \stackrel{\text{df}}{=} & W(p,t) + W(q,t) & \text{with } W(\star,t) \stackrel{\text{df}}{=} 0 \\ W'(t,(p,q)) & \stackrel{\text{df}}{=} & W(t,p) + W(t,q) & \text{with } W(t,\star) \stackrel{\text{df}}{=} 0 \\ M_0'((p_1,p_2)) & \stackrel{\text{df}}{=} & M(p_1) + M(p_2) & \text{with } M(\star) \stackrel{\text{df}}{=} 0 \\ \ell' & \stackrel{\text{df}}{=} & \ell|_{T'}\ , & \end{array}$$

where $\star \notin P$ is a new (abstract) element. $\diamond$

Definition A.2 (Simulation)
A *simulation from* Γ *to* Γ' is a relation $\mathcal{S}$ between the reachable markings of Γ and Γ' such that $(M_0, M_0') \in \mathcal{S}$ and for all $(M_1, M_2) \in \mathcal{S}$ and $M_1[t\rangle M_1'$ there is some M_2' with $M_2[\ell_1(t)\rangle\rangle M_2'$ and $(M_1', M_2') \in \mathcal{S}$. $\diamond$

If such a simulation exists, then Γ' can go on simulating all signals of Γ forever.
Below we define a similar concept.

Definition A.3 (Transition Simulation)
Let Γ and Γ' be with $T' \subseteq T$. A relation $\mathcal{S}$ between the reachable markings of Γ and Γ' is a *transition simulation* between Γ and Γ' if:

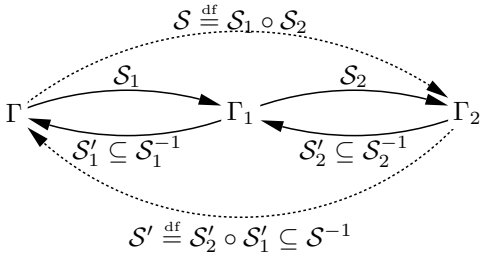
1. $(M_0, M_0') \in \mathcal{S}$;
2. $(M, M') \in \mathcal{S}$ and $M[v\rangle M_1$ with $v \in T^*$ implies $M'[v|_{T'}\rangle M_1'$ and $(M_1, M_1') \in \mathcal{S}$.

A relation $\mathcal{S}'$ between the reachable markings of Γ' and Γ is a *transition simulation* between Γ' and Γ if:

1. $(M_0', M_0) \in \mathcal{S}'$;
2. $(M', M) \in \mathcal{S}'$ and $M'[v'\rangle M_1'$ with $v' \in T'^*$ imply that $M[v\rangle M_1'$ with $v|_{T'} = v'$ and $(M_1', M_1) \in \mathcal{S}'$.$\diamond$

The following lemma is kind of folklore. We provide and prove it anyway, because it is slightly extended.

Lemma A.4 (Transitivity of Simulations)
Let Γ, Γ_1 and Γ_2 be nets. If $\mathcal{S}_1$ is a transition simulation between Γ and Γ_1, $\mathcal{S}_2$ between Γ_1 and Γ_2, $\mathcal{S}_1' \subseteq \mathcal{S}_1^{-1}$ between Γ_1 and Γ and $\mathcal{S}_2' \subseteq \mathcal{S}_2^{-1}$ between Γ_2 and Γ_1, then $\mathcal{S} \stackrel{\text{df}}{=} \mathcal{S}_1 \circ \mathcal{S}_2$ is a transition simulation between Γ and Γ_2 and $\mathcal{S}' \stackrel{\text{df}}{=} \mathcal{S}_2' \circ \mathcal{S}_1'$ is one between Γ_2 and Γ with $\mathcal{S}' \subseteq \mathcal{S}^{-1}$ (cf. the figure below).



Proof. Obviously, $T_2 \subseteq T_1 \subseteq T$. Furthermore, $(M_0, M_1) \in \mathcal{S}_1$ and $(M_1, M_2) \in \mathcal{S}_2$ implies $(M_0, M_2) \in \mathcal{S}$.

Let now $(M, M'') \in \mathcal{S}$. Hence, there is some M' such that $(M, M') \in \mathcal{S}_1$ and $(M', M'') \in \mathcal{S}_2$. Then, $M[v\rangle M_1$ implies $M'[v|_{T_1}\rangle M_1'$ with $(M_1, M_1') \in \mathcal{S}_1$; this implies $M''[v|_{T_2}\rangle M_1''$ with $(M_1', M_1'') \in \mathcal{S}_2$. Therefore, $(M_1, M_1'') \in \mathcal{S}$. With analogous reasoning, $\mathcal{S}'$ is a transition simulation between Γ_2 and Γ. Furthermore, $\mathcal{S}' = \mathcal{S}_2' \circ \mathcal{S}_1' \subseteq \mathcal{S}_2^{-1} \circ \mathcal{S}_1' \subseteq \mathcal{S}_2^{-1} \circ \mathcal{S}_1^{-1} = (\mathcal{S}_1 \circ \mathcal{S}_2)^{-1} = \mathcal{S}^{-1}$. □

Definition A.5 (Marking equality)
Let Γ' be an STG obtained from an STG Γ by the contraction of some transition. We say that a marking M of Γ and M' of Γ' satisfy the *marking equality* if for every place (p_1, p_2) of Γ': $M'((p_1, p_2)) = M(p_1) + M(p_2)$. ◇

The following proposition repeats some properties of secure transition contractions from [VW02, VK05].

Proposition A.6 (Contraction and Simulation)
Let Γ be a net with an arbitrary labelling and let Γ' be obtained from Γ by secure contraction of some transition. Then the relation $\mathcal{S} \stackrel{\text{df}}{=} \{(M, M') \mid M$ and M' satisfy the marking equality$\}$ is a simulation between Γ and Γ' and there is a simulation $\mathcal{S}' \subseteq \mathcal{S}^{-1}$ between Γ' and Γ.

We can also apply Proposition A.6 on the transition level.

Proposition A.7 (Contraction and Transition Simulation)
Let Γ be a net and let Γ' be obtained from Γ by the contraction of some transition t. Then $\mathcal{S} \stackrel{\text{df}}{=} \{(M, M') \mid$ M and M' satisfy
the marking equality$\}$ is a transition simulation between Γ and Γ' and there is a transition simulation $\mathcal{S}' \subseteq \mathcal{S}^{-1}$ between Γ' and Γ.

Proof. Follows from Proposition A.6 for the following labelling ℓ of Γ and Γ'.

$$\ell(t') = \begin{cases} t' & \text{for } t' \neq t \\ \lambda & \text{for } t' = t \end{cases}$$

□

Lemma A.8
Let Γ be a safe net and let Γ' be obtained from Γ by the secure contraction of a transition t. Then all places which are not generated by the contraction are safe (i.e. all places from $P' - {}^\bullet t \times t^\bullet$).

Proof. Let $\mathcal{S}$ be the corresponding transition simulation. Let M' be an arbitrary reachable marking with $M_0'[v'\rangle M'$. Therefore $M_0[v\rangle M$ with $v' = v|_{T'}$ and $(M, M') \in \mathcal{S}$. If $p' \in P'$ is not generated by the contraction, it is of the form $(p, \star)$ for some $p \in P$. The marking equality implies then $M'(p') = M'((p, \star)) = M(p) + M(\star) \leq 1$. □

Theorem 5.1 (Structural safeness-preservation). *A secure contraction of a transition t in a net Γ is safeness-preserving if*

1) $|{}^\bullet t| = 1$ *or*

2) $|t^\bullet| = 1$, ${}^\bullet(t^\bullet) = \{t\}$ *and*

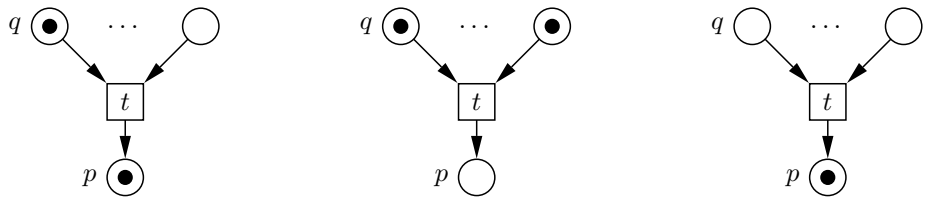
- *a)* Γ *is live and reversible*
  or
- *b)* $M_0(p) = 0$ *with* $t^\bullet = \{p\}$

Proof. Let $\mathcal{S}$ and $\mathcal{S}' \subseteq \mathcal{S}$ be the corresponding transition simulations from Proposition A.7. Observe that a transition contraction can turn a safe net in a 2-bounded one.

1) $|{}^\bullet t| = 1$: Let ${}^\bullet t = \{p\}$. If Γ is safe but Γ' not, one of the places generated during contraction is non-safe (Lemma A.8), e.g., (p, q) with $q \in t^\bullet$. Hence, a marking M' of Γ' exists with $M_0'[u\rangle M'$ with $M'((p, q)) = 2$. Proposition A.7 implies that there is a marking M of Γ with $(M', M) \in \mathcal{S}'$. Hence $M(p) + M(q) = 2$, and so due to the safeness of Γ, $M(p) = M(q) = 1$. Therefore t is enabled due to $M(p) = 1$, and firing it puts a token on q which already containes a token, contradicting the safeness of Γ.

2) $|t^\bullet| = 1$ and ${}^\bullet(t^\bullet) = \{t\}$: Observe, that the case $|t^\bullet| = 0$ is not possible, since the contraction would not be secure. When we talk about the markings M_1, M_2 and M_3 of Γ shown in the figure below, we always mean markings that look locally like these markings; this applies also to notational variations like M_1'.

Let Γ be safe and Γ' be non-safe, and let $t^\bullet = \{p\}$. Analogously to the first case, Lemma A.8 implies that there is a reachable marking M_1 of Γ such that $M_1(q) = M_1(p) = 1$ for some place $q \in {}^\bullet t$ (cf. the picture below).



M_1 M_2 M_3

If a) holds, i.e., Γ is live, there must be a reachable marking M_2 which enables t; since Γ is safe, M_2 puts exactly one token in every place in ${}^\bullet t$ and no token on p. The marking M_3 which is reachable from M_2 by firing t puts one token in p and no tokens in the places in ${}^\bullet t$. Since Γ is reversible, M_1 is reachable from M_3. If b) holds, i.e., p is initially unmarked, M_1 can only be reached via markings M_2 and M_3, since only t can put a token on p.

In both cases there are transition sequences v_1 and v_2 such that $M_0[v_1\rangle M_2[t\rangle M_3[v_2\rangle M_1$. Moreover, without loss of generality, one can assume that v_2 does not contain t; indeed, if $v_2 = v_2' t v_2''$ with v_2'' not containing t, one can build a firing sequence $M_0[v_1 t v_2'\rangle M_2'[t\rangle M_3'[v_2''\rangle M_1$ with similar properties.

Since v_2 does not contain t, the token on p cannot be removed by the transitions in v_2 because only t can put it there again. Hence $M_2[v_2\rangle M$ with $M(q) = 2$, because the firing of v_2 increases the marking of q by 1. This contradicts Γ being safe. □

Proposition 5.2. *Secure transition contractions and implicit place deletions preserve liveness and reversibility.*

Proof. The claim is obviously true for an implicit place deletion, since the resulting STG has an isomorphic state graph.

Due to Proposition A.7, there is a transition simulation $\mathcal{S}$ between Γ and Γ' and a transition simulation $\mathcal{S}' \subseteq \mathcal{S}^{-1}$ between Γ' and Γ. Let u' be a firing sequence of Γ' such that $M_0'[u'\rangle M_1'$; then $M_0[u\rangle M_1$ for some firing sequence u of Γ such that $(M_1', M_1) \in \mathcal{S}'$.

If Γ is reversible then $M_1[v\rangle M_0$ for some transition sequence v of Γ. Therefore, $M_1'[v|_{T'}\rangle M_2'$ such that $(M_0, M_2') \in \mathcal{S}$. By definition of $\mathcal{S}$, M_0 and M_0' as well as M_0 and M_2' fulfil the marking equality, i.e., the following equations hold for each place (p, q) of Γ':

$$\begin{aligned} M_0'((p,q)) &= M_0(p) + M_0(q) \\ M_2'((p,q)) &= M_0(p) + M_0(q)\ , \end{aligned}$$

i.e., $M_0' = M_2'$ and Γ' is reversible.

To show the liveness of Γ', suppose that one wants to activate some transition t' in Γ' starting from M_1' (note that $t' \neq t$ since t has been contracted). If Γ is live, one can activate t' in Γ starting from M_1 by some transition sequence w: $M_1[w\rangle M_2[t'\rangle$. Since $(M_1, M_1') \in \mathcal{S}'^{-1} \subseteq \mathcal{S}$, $M_1'[w|_{T'}\rangle M_2'[t'\rangle$, which proves the liveness of Γ'. □

Proposition 5.3. *Let Γ be a safe STG and $t \in T$ such that the contraction of t is secure. The contraction of t is safeness-preserving iff the following property does not hold:*

$$\left(\bigvee_{p \in {}^\bullet t} p\right) \wedge \left(\bigvee_{p \in t^\bullet} p\right)\ .$$

Proof. Let Γ' be the resulting STG, $\mathcal{S}$ be the transition simulation between Γ and Γ', and $\mathcal{S}' \subseteq \mathcal{S}^{-1}$ be the transition simulation between Γ' and Γ.

($\Leftarrow$) Suppose the opposite, i.e., that Γ' is safe but there is a reachable marking M of Γ fulfilling the Boolean expression. Then there are two places $p \in {}^\bullet t$ and $q \in t^\bullet$ with $M(p) = M(q) = 1$. Since the contraction is defined, $p \neq q$. Since M is reachable, $M_0[v\rangle M$ for some firing sequence v of Γ. Due to Proposition A.7, $M'_0[v|_{T'}\rangle M'$ with $(M, M') \in \mathcal{S}$. Since t was contracted, $M'((p,q)) = M(p) + M(q) = 2$, a contradiction.

($\Rightarrow$) Suppose now that Γ' is non-safe due to a place p'. Lemma A.8 implies that p' is newly generated by the contraction, i.e. $p' \equiv (p,q)$ with $p \in {}^\bullet t$ and $q \in t^\bullet$. Then, the marking equality implies $M'((p,q)) = M(p) + M(q) > 1$. Since Γ is safe, $M(p) = M(q) = 1$ and M fulfills the Boolean expression. □

To check the safeness of a sequence of contractions on a single unfolding, one has to build expressions over the original net, which are derived from the intermediate nets. For this, we have to consider the structure of places generated by the contractions.

Definition A.9 (Place Projection)
Let Γ and Γ' be STGs such that Γ' was obtained from Γ by a sequence of transition contractions. Every place p' of Γ' is a pair, where each element is another pair etc., down to the level of places from P. The function $\Phi_\Gamma^{\Gamma'}$ assigns every place of Γ' the multiset of places of Γ occurring in p'. We write $\Phi_\Gamma^{\Gamma'}(p')(p)$ to denote the number of occurrences of p within p'. $[\Phi_\Gamma^{\Gamma'}(p')]$ denotes the *support* of $\Phi_\Gamma^{\Gamma'}(p')$, i.e., the set of elements occurring in this multiset. ◇

For example, $\Phi_\Gamma^{\Gamma'}(((p_1,p_2),(p_1,p_3))) = \{2 \cdot p_1, p_2, p_3\}$ and $\Phi_\Gamma^{\Gamma'}(((p_1,p_3),\star)) = \{p_1, p_3\}$.

As an analogy to the marking equality for a single transition contraction, we show that the *extended marking equality* holds after several contractions.

Proposition A.10 (Extended marking equality)
Let Γ be an STG and let Γ' be obtained from it by a sequence of secure contractions of some transitions. Then there is a transition simulation $\mathcal{S}$ from Γ to Γ' and a transition simulation $\mathcal{S}' \subseteq \mathcal{S}^{-1}$ from Γ' to Γ such that for every $(M, M') \in \mathcal{S}$ and every place p' of Γ', $M'(p') = \sum_{p \in P} \Phi_\Gamma^{\Gamma'}(p')(p) \cdot M(p)$.

Proof. Let $\Gamma_0 \stackrel{\text{df}}{=} \Gamma$, and for $1 \leq i \leq n$, let Γ_i be the net after the i-th transition contraction, with $\Gamma_n \stackrel{\text{df}}{=} \Gamma'$. Proposition A.6 implies that there is a transition simulation $\mathcal{S}_i$ between Γ_{i-1} and Γ_i and a transition simulation $\mathcal{S}'_i \subseteq \mathcal{S}_i^{-1}$ between Γ_i and Γ_{i-1}. Recall that for every $(M, M') \in \mathcal{S}_i$, M and M' fulfil the marking equality. Let now $\mathcal{S}^i \stackrel{\text{df}}{=} \mathcal{S}_1 \circ \mathcal{S}_2 \circ \ldots \circ \mathcal{S}_i$. Repeated application of Lemma A.4 gives that for each i, $\mathcal{S}^i$ is a transition simulation between Γ and Γ_i; in particular $\mathcal{S} = \mathcal{S}^n$ is a transition simulation between Γ and Γ', and there is a transition simulation $\mathcal{S}' \subseteq \mathcal{S}^{-1}$ between Γ' and Γ.

We now show by induction that for each i, if $(M, M') \in \mathcal{S}^i$ then for every place p' of Γ_i, $M'(p') = \sum_{p \in P} \Phi_\Gamma^{\Gamma_i}(p')(p) \cdot M(p)$. For $i = 1$, this is directly implied by the marking equality for $\mathcal{S}_1 = \mathcal{S}^1$. Assume now that the claim is fulfilled for some i. Let $M_0[v\rangle M$; then $M_0^i[v|_{T_i}\rangle M'_i$ with $(M, M'_i) \in \mathcal{S}^i$ and $M_0^{i+1}[v|_{T_{i+1}}\rangle M'_{i+1}$ with $(M, M'_{i+1}) \in \mathcal{S}^{i+1}$, where M_0^i denotes the initial marking of Γ_i. Observe that $(M'_i, M'_{i+1}) \in \mathcal{S}_{i+1}$. For $(p_1, p_2) \in P_{i+1}$, we obtain

$$\begin{aligned}
\sum_{p \in P} \Phi_\Gamma^{\Gamma_{i+1}}((p_1,p_2))(p) \cdot M(p) &= \sum_{p \in P} \Phi_\Gamma^{\Gamma_i}(p_1)(p) \cdot M(p) + \sum_{p \in P} \Phi_\Gamma^{\Gamma_i}(p_2)(p) \cdot M(p) \quad \text{(def. of } \Phi\text{)} \\
&= M'_i(p_1) + M'_i(p_2) \quad \text{(induction)} \\
&= M'_{i+1}((p_1,p_2)) \quad \text{(marking equality for } \mathcal{S}_{i+1}\text{)} ,
\end{aligned}$$

and the case $i = n$ proves the claim. □

Proposition A.11
Let Γ be a safe STG and let Γ' be an STG obtained from it by a sequence of safeness-preserving transition contractions. Then the contraction of a transition t in Γ' is safeness-preserving iff the following property does not hold:

$$\left(\bigvee_{p \in \Phi_{\bullet t}} p \right) \wedge \left(\bigvee_{p \in \Phi_{t^\bullet}} p \right) ,$$

where $\Phi_{\bullet t} \stackrel{\text{df}}{=} \bigcup_{p \in {}^\bullet t}[\Phi_\Gamma^{\Gamma'}(p)]$ *and* $\Phi_{t^\bullet} \stackrel{\text{df}}{=} \bigcup_{p \in t^\bullet}[\Phi_\Gamma^{\Gamma'}(p)]$.

Proof. We will show that this equation — denoted (0) — can be fulfilled in Γ if and only if the equation from Proposition 5.3 — denoted (1) — can be fulfilled in Γ'. Let $\mathcal{S}$ be the transition simulation between Γ and Γ' and $\mathcal{S}' \subseteq \mathcal{S}^{-1}$ be the transition simulation between Γ' and Γ, whose existence is implied by Proposition A.10.

($\Leftarrow$) Suppose (0) is fulfilled for the marking M reached by some firing sequence v of Γ. Then there are two places $p \in \Phi_{\bullet t}$ and $q \in \Phi_{t^\bullet}$ with $M(p) = M(q) = 1$. Moreover, by Proposition A.10, $M_0'[v|_{T'}\rangle M'$ with $(M, M') \in \mathcal{S}$ and $M'(s) = \sum_{r \in P} \Phi_\Gamma^{\Gamma'}(s)(r) \cdot M(r)$ for each place s of Γ'. This and safeness of Γ' imply $\forall s' \in P'.p \in [\Phi_\Gamma^{\Gamma'}(s')] \Rightarrow M'(s') = 1$. In particular this is true for all places $s' \in {}^\bullet t$, hence there is a place $p' \in {}^\bullet t$ with $M'(p') = 1$ and analogously there is a place $q' \in {}^\bullet t$ with $M(q') = 1$ and therefore (1) is fulfilled for M'.

($\Rightarrow$) Suppose now that (1) is fulfilled for the marking M' reached by some firing sequence v' of Γ. Then there are two places $p' \in {}^\bullet t$ and $q' \in t^\bullet$ with $M'(p') = M'(q') = 1$. Also, there exists an firing sequence v of Γ such that $v' = v|_{T'}$ and $M_0[v\rangle M$ with $(M, M') \in \mathcal{S}'$. Moreover, by Proposition A.10, $M'(s) = \sum_{r \in P} \Phi_\Gamma^{\Gamma'}(s)(r) \cdot M(r)$ for each place s of Γ', and $\sum_{r \in P} \Phi_\Gamma^{\Gamma'}(p')(r) \cdot M(r) = 1$. Therefore, M puts a token in some place in $[\Phi_\Gamma^{\Gamma'}(p')]$. Similarly, one can show that M puts a token in some place in $[\Phi_\Gamma^{\Gamma'}(q')]$, and thus (0) is fulfilled for M. □

The last proposition makes it possible to use one unfolding to check the safeness of several subsequent contractions. This works as follows:

1. Starting with Γ, check if the first contraction is safeness-preserving using Proposition 5.3.
2. Perform the contraction resulting in a new STG.
3. To check if the next contraction is safeness-preserving in the new STG, build an expression over Γ using Proposition A.11.
4. Repeat steps 2 and 3 until all the desired contractions are performed.

Observe that all the new nets are generated and used to build an expression over the original net, thus the original unfolding prefix can be used.