Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam | 32

Dominic Wist | Mark Schaefer | Walter Vogler | Ralf Wollowski

STG Decomposition: Internal Communication for SI Implementability

Universitätsverlag Potsdam

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de/ abrufbar.

Universitätsverlag Potsdam 2010
http://info.ub.uni-potsdam.de/verlag.htm

Am Neuen Palais 10, 14469 Potsdam
Tel.: +49 (0)331 977 4623 / Fax: 3474
E-Mail: verlag@uni-potsdam.de

Die Schriftenreihe **Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam** wird herausgegeben von den Professoren des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam.

Druck: docupoint GmbH Magdeburg

ISSN 1613-5652
ISBN 978-3-86956-037-3

Zugleich online veröffentlicht auf dem Publikationsserver der Universität Potsdam:
URL http://pub.ub.uni-potsdam.de/volltexte/2010/4078/
URN urn:nbn:de:kobv:517-opus-40786
[http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-40786]

STG Decomposition: Internal Communication for SI Implementability*

Dominic Wist[1], Mark Schaefer[2], Walter Vogler[2] and Ralf Wollowski[1]

[1]Hasso-Plattner-Institut
University of Potsdam, Germany
dominic.wist@hpi.uni-potsdam.de
ralf.wollowski@hpi.uni-potsdam.de

[2]Institute of Computer Science
University of Augsburg, Germany
vogler@informatik.uni-augsburg.de
mark@markschaefer.de

Abstract

Logic synthesis of speed independent circuits based on STG decomposition is a promising approach to tackle complexity problems like state-space explosion. Unfortunately, decomposition can result in components that in isolation have irreducible CSC conflicts. Generalising earlier work, we show how to resolve such conflicts by introducing internal communication between the components.

The new algorithms are successfully applied to some benchmarks, including very complex STGs arising in the context of control resynthesis.

Keywords: STG, decomposition, speed independent, CSC, resynthesis

1 Introduction

Speed independent (SI) circuits are an important subclass of asynchronous circuits providing all their well-known advantages [vBJN99] in comparison to synchronous ones. They work correctly, i.e. they are free of hazards for some specified behaviour, regardless of their gate delays (wires are assumed to have negligible delays). *Signal Transitions Graphs (STGs)* [Chu87, Wen77] are a formalism to specify the behaviour of SI circuits. They are interpreted Petri nets in which transitions are labelled with rising and falling edges of the circuit signals.

Petrify [CKK+02] and Punf&Mpsat [KKY04] are commonly used tools for logic synthesis of SI circuits from STG *specifications*. For logic synthesis, Petrify must explore the entire state space of the STG, and so suffers from the *state-space explosion* problem. The unfolding based synthesis via Mpsat also has complexity problems when solving large SAT problems. Although logic synthesis leads to very efficient circuit implementations – compared e.g. to the synthesis based on *syntax-directed translation* [EB02] – it is only applicable to rather small specifications.

To cope with the complexity problems of logic synthesis, we suggest to decompose the STG specification into several smaller component STGs and to apply logic synthesis for each component STG [VW02, SVWK06]. In contrast to our approach, other STG

*This research was supported by the DFG-project 'STG-Dekomposition' Wo814/1-3.

decomposition techniques as the one implemented in Nutas [YM07] and the one in Moebius [CCCGV06] need specifications fulfilling the so-called CSC property (which is necessary for synthesising SI circuits). This is a serious restriction for 'real' specifications – note that non of our benchmarks satisfies CSC (see Section 5). To overcome this drawback Moebius can resolve CSC for large STGs by solving NP-complete ILP problems which limits the specification size, again.

With our STG decomposition, we follow a more scalable approach which tries to avoid expensive operations (such as resolving all encoding conflicts) on the original specification. The resulting component STGs in isolation might have irreducible CSC conflicts though (i.e. they are not resolvable), even if the original specification has none.

In [WWSV09], we proposed a solution to avoid these irreducible CSC conflicts in the component STGs by introducing new internal communication between the components. We got very promising results, but the approach only works for structural self-triggers (a special type of irreducible CSC conflicts) in combination with a very restricted specification structure. Here, we improve this approach by avoiding self-triggers for specifications having a more general structure: we give a more general algorithm how to insert an internal communication signal. We show how to limit the growth of the components, and for this we have to prove correct a generalised version of our decomposition algorithm. We also discuss how to treat repeated signal insertion in order to synthesise efficient circuits, and finally present some experimental results. Additionally, we suggest how self-trigger avoidance can be used to deal with general irreducible CSC conflicts as well.

Our approach enables the synthesis of more benchmarks than before, in particular in the important field of control resynthesis: from a behaviour description, e.g. in Balsa [EB02], a netlist is derived that describes how to connect certain handshake components (with an optimised implementation), i.e. a certain graph. Instead of connecting all these implementations to get a circuit, control resynthesis forms clusters consisting of control handshake components (like sequencer and concur components) in the graph; then for each control cluster, a complex STG modelling its interface behaviour can be derived, cf. [FC08]. Such an STG is usually too complex to apply pure logic synthesis, but decomposition-based synthesis with our tool DesiJ [SWW09] often succeeds – and with the presented improved version in many more cases.

The paper is organised as follows: in the next section we introduce the basic concepts of STGs and their decomposition, and present the generalisation of our decomposition algorithm. In Section 3, we briefly recapitulate the idea how to avoid self-triggers in component STGs by introducing internal communication between components; furthermore, we suggest how to deal with general irreducible CSC conflicts. In the following section, we present structural techniques for introducing internal communication by correctly inserting new internal signals into the original specification and decompose it anew. In Section 5, we present some experimental results also for some STGs arising in the context of control resynthesis. We draw conclusions in Section 6.

2 Basics

This section provides basic notions for Petri nets and STGs, for a more detailed explanation the reader is referred to [CKK$^+$02].

Definition 2.1 (Multiset)
A *multiset ms over a set A* is a function $ms : A \to \mathbb{N}_0$. Let ms and ms' be multisets over A; then for $a \in A$:

- $a \in ms \Leftrightarrow ms(a) > 0$
- $(ms + ms')(a) = ms(a) + ms'(a)$
- $(ms - ms')(a) = \max(0, ms(a) - ms'(a))$

A subset A' of A is considered implicitly as multiset $ms_{A'}$ with $ms_{A'}(a) = 1$ if $a \in A'$ and $ms_{A'}(a) = 0$ otherwise. △

2.1 Petri Nets

A (labelled) *Petri net* is a 6-tuple $N = (P, T, W, M_N, \Sigma, l)$ where P and T are disjoint and finite sets of *places* and *transitions*. $W : P \times T \cup T \times P \to \mathbb{N}_0$ is the *weight function* and M_N the *initial marking*, where a *marking* is a multiset of places, i.e. a function $P \to \mathbb{N}_0$ which assigns a number of *tokens* to each place. The marking of a set of places is defined as the sum of all individual markings. Σ is a set of *actions*, and $l : T \to \Sigma \cup \{\lambda\}$ is the *labelling function* where λ denotes the empty word. If necessary, we write P_N, T_N etc. for the components of N, and P', P_i etc. for the nets N', N_i etc. Analogous conventions apply later on.

A Petri net can be considered as a bipartite graph with weighted and directed edges between places and transitions. In a graphical representation, places are drawn as circles containing a number of tokens corresponding to their marking, transitions are drawn as rectangles together with their labelling, and the weight function is drawn as directed arcs xy whenever $W(x, y) \neq 0$ (and labelled with $W(x, y)$ if $W(x, y) > 1$). A place p is called *marked graph place* or *MG-place* if $\sum_{t \in T} W(t, p) = 1 = \sum_{t \in T} W(p, t)$. Unmarked MG-places are not drawn; they are implicitly given by an arc between the respective transitions; cf. Figure 1 below.

The *preset* of a place or transition x is denoted as ${}^\bullet x$ and defined by ${}^\bullet x = \{y \in P \cup T \mid W(y, x) > 0\}$, the *postset* of x is denoted as $x^\bullet$ and defined by $x^\bullet = \{y \in P \cup T \mid W(x, y) > 0\}$. These notions are extended to sets as usual. We say that there is an *arc* from each $y \in {}^\bullet x$ to x.

For a place or transition x, $N - x$ denotes the net in which x and all incident arcs are deleted; for a marking M, $M|_{P'}$ denotes its restriction to $P' \subseteq P$, and $M|_{-p}$ is shorthand for $M|_{P - \{p\}}$.

A nonempty sequence $w = x_1 x_2 \dots x_n$ of places and transitions without duplicates is a *path (of N)* if $W(x_i, x_{i+1}) > 0$ for $1 \leq i < n$. Obviously, places and transitions have to alternate on a path. With an abuse of notation we often consider a path as the set containing its elements, writing for example $p \in w$. A path w is a *marked graph path* or *MG-path* if every place of w is an MG-place. For a marking M, the *marking $M(w)$ of a path w* is defined as $M(w \cap P)$. A path w is called *non-joining (non-forking resp.)* if for every transition t on w except the first (last resp.) one, $|{}^\bullet t| \leq 1$ ($|t^\bullet| \leq 1$ resp.). It is called *non-merging (non-branching)* if for every place p on w, $|{}^\bullet p| \leq 1$ ($|p^\bullet| \leq 1$). When connecting two paths $w_1 = x_1 \dots x_n$ and $w_2 = x_n \dots x_m$ we write $w_1 \, w_2$ for $x_1 \dots x_n \dots x_m$.

A transition t is *enabled under a marking* M if $\forall p \in {}^\bullet t : M(p) \geq W(p,t)$, which is denoted by $M[t\rangle$. An enabled transition can *fire* or *occur* yielding a new marking M' – written as $M[t\rangle M'$ – if $M[t\rangle$ and $M'(p) = M(p) - W(p,t) + W(t,p)$, for all $p \in P$.

A *transition sequence* $v = t_1 \dots t_n$ is *enabled under a marking* M (yielding M') if $M[t_1\rangle\ M_1[t_2\rangle \dots\ M_{n-1}[t_n\rangle M_n = M'$, and we write $M[v\rangle$, $M[v\rangle M'$ resp.; v is called *firing sequence* if $M_N[v\rangle$. The empty transition sequence λ is enabled under every marking. M is called *reachable* if a transition sequence v with $M_N[v\rangle M$ exists, and $[M_N\rangle$ is the set of all *reachable markings*.

N is called *bounded* if, for some constant $k \in \mathbb{N}_0$, $M(p) \leq k$ for every reachable marking M and every place p ; if $k = 1$, N is called *safe*. N is bounded if and only if the set $[M_N\rangle$ of reachable markings is finite. In this paper, we are only concerned with bounded nets.

We lift the notion of enabledness to transition labels: we write $M[l(t)\rangle\rangle M'$ if $M[t\rangle M'$. This is extended to sequences as usual – deleting λ-labels automatically since λ is the empty word; i.e. $M[a\rangle\rangle M'$ means that a sequence of transitions fires, where one of them is labelled with a while the others (if any) are λ-labelled.

A net has a *dynamic conflict* if there are different transitions t_1 and t_2 such that for some reachable marking M: $M[t_1\rangle$ and $M[t_2\rangle$, but $\exists p \in P : M(p) < W(p,t_1) + W(p,t_2)$. A dynamic conflict implies a *structural conflict*, i.e. ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$. The conflict is called *auto-conflict* if $l(t_1) = l(t_2) \neq \lambda$.

The *reachability graph* RG_N of a Petri net N is an edge-labelled directed graph on the reachable markings with M_N as root; there is an edge from M to M' labelled $l(t)$ whenever $M[t\rangle M'$. RG_N can be seen as a finite automaton (where all states are accepting).[1] N is *deterministic* if its reachability graph is a deterministic automaton, i.e. if it contains no λ-labelled transitions and if for each reachable marking M and label $a \in \Sigma$ there is at most one M' with $M[a\rangle\rangle M'$.

2.2 Signal Transition Graphs and SI Implementability

A Signal Transition Graph (STG) is a tuple $N = (P, T, W, M_N, In, Out, Int, l)$, where $(P, T, W,\ M_N, Sig^\pm, l)$ is a Petri net, In, Out and Int are disjoint sets of *input*, *output* and *internal signals*, and $Sig = In \cup Out \cup Int$ is the set of all signals; *signature* refers to this partition of the signal set. $Sig^\pm = Sig \times \{+, -\}$ is the set of *signal edges* or *signal transitions*; its elements are denoted as s^+, s^- resp. instead of $(s, +)$, $(s, -)$ resp. A plus sign denotes that a signal value changes from *logical low* (written as 0) to *logical high* (written as 1), and a minus sign denotes the opposite direction. We write $s^\pm$ if it is not important or unknown which direction takes place; if such a term appears more than once in the same context, it always denotes the same direction. We define the set of *locally controlled* or just *local signals* $Loc = Out \cup Int$ which are produced by the STG and the set of all *external signals* $Ext = In \cup Out$ which are observable in the environment.

To keep the notation short, input/output/internal signal edges are just called input/output/internal edges and each output/internal edge is also called *non-input edge*; transitions labelled with these are called input/output/internal transitions or *non-input transitions* resp. If transitions are labelled with λ they do not correspond to any signal change, i.e. they are not internal transitions; they are also called *dummy-transitions*. An

[1] Recall that we only consider bounded Petri nets here, which have only finitely many reachable markings.

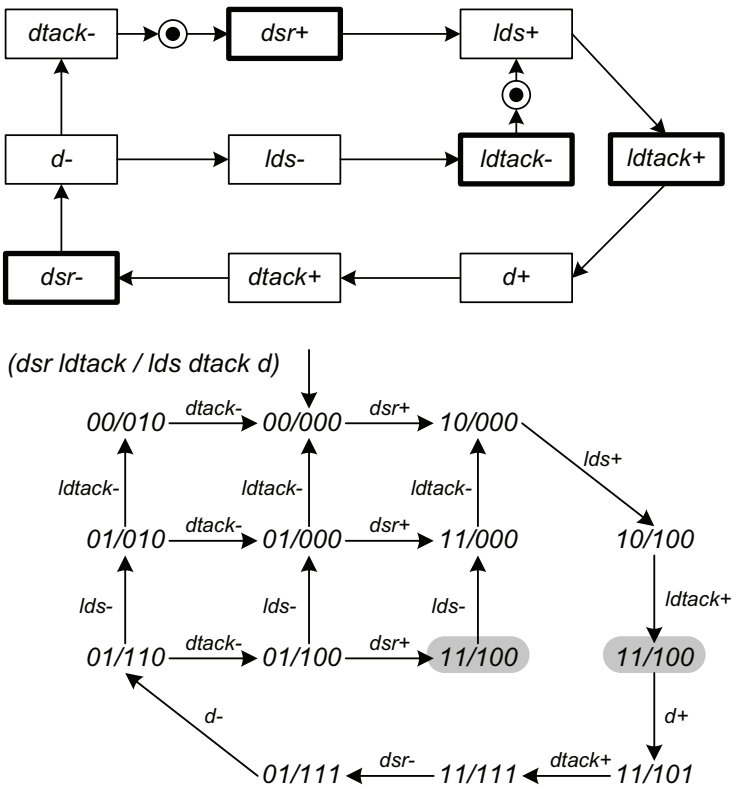

Figure 1: An STG modelling a simplified VME bus controller (top) and its state graph with a CSC conflict between the shaded states (bottom).

example of an STG is shown in Fig. 1 (cf. [CKK+02]); input transitions are drawn with a thick border.

STGs are used for specifying the behaviour of speed-independent (SI) circuits. The idea is as follows: a reachable marking of the STG roughly corresponds to a state of the intended circuit (viz. the values of its signals). If some marking activates an output (or internal) edge, the circuit must produce the same edge if it is in a corresponding state and the environment of the circuit must be ready to receive it; if some marking activates an input edge, the environment is allowed to produce it and the circuit must be ready to receive it.

For the first step from markings to circuit states, one defines the notion of *state assignment*: for an STG N, a *state vector* is a function $sv : Sig \to \{0, 1\}$, which assigns a Boolean value to each signal. A *state assignment* assigns a state vector sv_M to each marking M of $[M_N\rangle$; it must satisfy for every signal $x \in Sig$ and every pair of markings $M, M' \in [M_N\rangle$:

$$M[x^+\rangle\rangle M' \text{ implies } sv_M(x) = 0, sv_{M'}(x) = 1$$
$$M[x^-\rangle\rangle M' \text{ implies } sv_M(x) = 1, sv_{M'}(x) = 0$$
$$M[y^\pm\rangle\rangle M' \text{ for } y \neq x \;\; (M[\lambda\rangle\rangle M' \text{ resp.}) \text{ implies } sv_M(x) = sv_{M'}(x)$$

If such an assignment exists, it is uniquely defined by these properties[2], and the

[2]At least for every signal $s \in Sig$ which actually occurs, i.e. $M[s^\pm\rangle\rangle$ for some reachable marking M.

reachability graph and the underlying STG are *consistent*. From an *inconsistent* STG, one cannot synthesise a circuit, and in this paper we assume that all STGs are consistent. The *state graph* of an STG is its reachability graph where each marking is annotated with its state vector; cf. Fig. 1(bottom) where only the state vectors are shown.

We now explain the important concept of *Complete State Coding (CSC)*. If there is a state assignment, N has *CSC* if any two reachable markings M_1 and M_2 with the same state vector (i.e. $sv_{M_1} = sv_{M_2}$) enable the same output and internal signals. Otherwise, N has a *CSC conflict*, cf. e.g. Fig. 1(bottom), and no circuit can be synthesised directly. If CSC is violated, one tries to achieve it by inserting internal signals such that the state vectors of M_1 and M_2 differ and the external behaviour of the STG is unchanged; thus the internal signal insertion must be *input proper* [SV07], i.e. no input edge must be delayed by any internal edge. If CSC cannot be achieved by an input proper signal insertion, the conflict is called *irreducible*. *(Dynamic) self-triggers* are a special type of irreducible CSC conflicts characterised by a transition sequence $M_1[t_1t_2\rangle M_2$, where t_1, t_2 are labelled with the same input signal, but complementary edges, and M_2 does not activate the same local signal edges as M_1. We define a *structural self-trigger* as two transitions t and t' which are labelled with complementary edges of the same input signal satisfying $t \in {}^\bullet({}^\bullet t')$, and call t the *entry* and t' the *exit transition*; a structural self-trigger is necessary for a dynamic one.

Output persistency guarantees the robustness (hazard-freedom) of the desired SI circuit, i.e. when a signal disables another one, then both signals must be inputs; thus, each activated non-input edge will eventually happen. A circuit is called *speed-independent (SI)* if it is output persistent in all behaviours under a given environment. So, the intended circuit will work correctly under arbitrary delays of gates (while the signal propagation is considered instantaneous). We can lift the notion of output persistency to the level of state graphs and STGs as well, see [CKK+02].

From the state graph of a bounded, consistent and output persitent STG satisfying CSC one can derive an SI circuit, i.e. a boolean function for each output or internal signal. This function has to be mapped to Boolean gates. Since this synthesis process needs a representation of the state graph, it suffers from the state-space-explosion problem; there are synthesis tools e.g. PETRIFY.

2.3 STG Operations

Now, we present a number of operations that are important for decomposition. A decomposition of an STG N is correct if the parallel composition of its components matches the behaviour of N. To define this formally, we have to introduce *parallel composition*, and for this we have to consider the distinction between input and output signals.

The notion of parallel composition is that the composed systems run in parallel and synchronise on common signals – corresponding to circuits that are connected on the wires corresponding to these signals. Since a system controls its outputs, we cannot allow a signal to be an output of more than one component; input signals, on the other hand, can be shared. An output signal of a component may be an input of other components, and in any case it is an output of the composition. Internal signals of one component must not be used by the others; this is no serious restriction and can always be achieved by a suitable renaming of the respective signals.[3]

[3] A composition can also be ill-defined due to what e.g. Ebergen [Ebe92] calls *computation interference*;

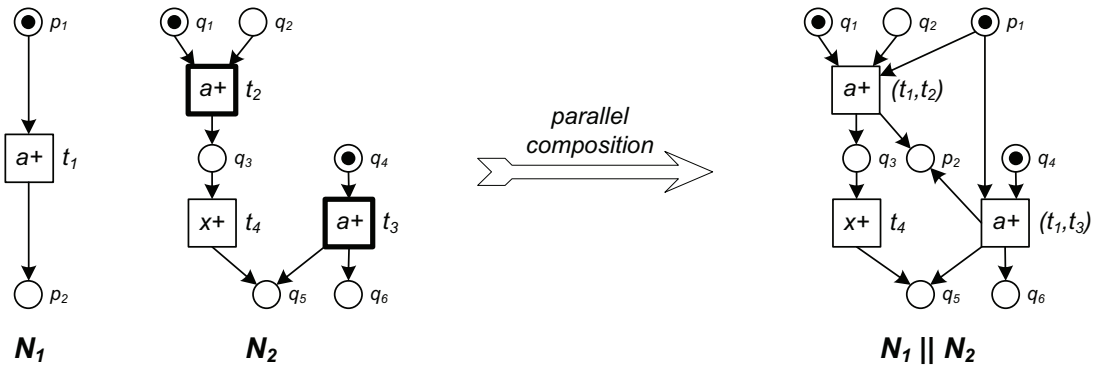

Figure 2: Parallel composition example. The two net fragments on the left share signal a, as an output in the left one and as input (twice) in the right one. Hence, in their parallel composition (right) a is an output.

The parallel composition of STGs N_1 and N_2 is defined if $Out_1 \cap Out_2 = Int_1 \cap Sig_2 = Int_2 \cap Sig_1 = \emptyset$. The place set of the composition is the disjoint union of the place sets of the components; thus, we can consider markings of the composition (regarded as multisets) as the disjoint union of markings of the components. To define the transitions, let $A = Sig_1 \cap Sig_2$ be the set of common signals. If e.g. s is an output of N_1 and an input of N_2, then an occurrence of an edge $s^{\pm}$ in N_1 is 'seen' by N_2, i.e. it must be accompanied by an occurrence of $s^{\pm}$ in N_2. Since we do not know a priori which $s^{\pm}$-labelled transition of N_2 will occur together with some $s^{\pm}$-labelled transition of N_1, we have to allow for each possible pairing. Thus, the *parallel composition* $N = N_1 \parallel N_2$ is obtained from the disjoint union of N_1 and N_2 by combining each $s^{\pm}$-labelled transition t_1 of N_1 with each $s^{\pm}$-labelled transition t_2 from N_2 if $s \in A$. Such transitions are pairs and the firing $(M_1 + M_2)[(t_1, t_2)\rangle(M_1' + M_2')$ of N corresponds to the firings $M_i[t_i\rangle M_i'$ in N_i, $i = 1, 2$; for an example of a parallel composition, see Fig. 2. More generally, we have $(M_1 + M_2)[w\rangle\rangle(M_1' + M_2')$ iff $M_i[w|_{N_i}\rangle\rangle M_i'$ for $i \in \{1, 2\}$, where $w|_{N_i}$ denotes the projection of the trace w onto the signals of the STG N_i.

It is easy to see that N is deterministic if N_1 and N_2 are. However, as illustrated in Fig. 2, N might have structural auto-conflicts even if none of the N_i has them. Obviously, we can define the parallel composition of a finite family (or collection) $(C_i)_{i \in I}$ of STGs as $\|_{i \in I} C_i$, provided that no signal is an output signal of more than one of the C_i.

Next, we introduce the concept of *lambdarising* a signal. It simply means to change the labelling function such that all transitions corresponding to this signal are labelled with λ and to remove this signal from the signature; *delambdarising* a signal means to restore the former labelling and signature. By contrast, *hiding* a signal set $H \subseteq Out$ from an STG N results in the STG $N/H = (P, T, W, M_N, In, Out \setminus H, Int \cup H, l)$, i.e. some output signals are now considered to be internal signals.

The most important operation for decomposition is transition contraction (see e.g. [And83] for an early reference); we essentially repeat from [VK06], where further discussions can be found.

this is a semantic problem, and we will not consider it here.

Definition 2.2 (Transition Contraction)
Let N be a Petri net and $t \in T$ with $l(t) = \lambda$, ${}^\bullet t \cap t^\bullet = \emptyset$ and $W(p,t), W(t,p) \leq 1$ for all $p \in P$. We define the *t-contraction* N' of N by

$$\begin{aligned} P' &= \{(p,\star) \mid p \in P \setminus ({}^\bullet t \cup t^\bullet)\} \\ &\quad \cup \{(p,p') \mid p \in {}^\bullet t, p' \in t^\bullet\} \\ T' &= T \setminus \{t\} \\ W'((p,p'),t_1) &= W(p,t_1) + W(p',t_1) \\ W'(t_1,(p,p')) &= W(t_1,p) + W(t_1,p') \\ l' &= l|_{T'} \\ M_{N'}((p,p')) &= M_N(p) + M_N(p') \\ In' = In \quad Out' &= Out \quad Int' = Int \end{aligned}$$

In this definition, $\star \notin P \cup T$ is a pseudo element; we assume $W(\star,t_1) = W(t_1,\star) = M_N(\star) = 0$.

We say that the markings M of N and M' of N' satisfy the *marking equality* if for all $(p,p') \in P'$

$$M'((p,p')) = M(p) + M(p').$$

For two different transitions t_1, t_2 with $t_1 \neq t \neq t_2$, we call the unordered pair $\{t_1,t_2\}$ a *new conflict pair* whenever ${}^\bullet t \cap {}^\bullet t_1 \neq \emptyset$ and $t^\bullet \cap {}^\bullet t_2 \neq \emptyset$ in N (or vice versa); if $l(t_1) = l(t_2) \neq \lambda$, we speak of a *new structural auto-conflict*.

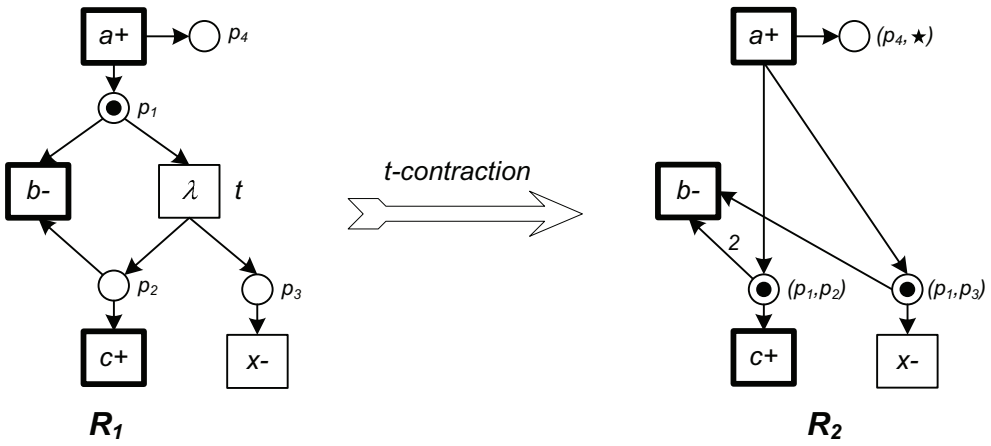

Figure 3: Example of a transition contraction in an STG.

A transition contraction is called *secure* if either $({}^\bullet t)^\bullet \subseteq \{t\}$ (*type-1 secure*) or ${}^\bullet(t^\bullet) = \{t\}$ and $M_N(p) = 0$ for some $p \in t^\bullet$ (*type-2 secure*). △

Note that, in general, N' might fail to be consistent, even if N is; but secure contractions preserve consistency (see [VK06]).

Fig. 3 shows a part of a net and the result of contracting the λ-transition, where the b^-- and the x^--labelled transition form a new conflict pair; note that this is also true for b^- and c^+, although they are already in structural conflict in N.

We now define redundant transitions and implicit places; the deletion of such a transition, place resp., (including the incident arcs) is another operation that can be used in

our decomposition algorithm. A transition t is *redundant* if either it is a λ-transition with $W(p,t) = W(t,p)$ for each place p (i.e. t is a *loop-only* transition), or there is another transition t' with the same label such that $W(p,t) = W(p,t')$ and $W(t,p) = W(t',p)$ for each place p (i.e. t is a *duplicate* transition).

A place p is *implicit* if it can be removed from the net without changing the set of firing sequences. However, detecting implicit places is PSpace-complete and during decomposition only redundant places are deleted.

Definition 2.3 (Redundant Places)
The place q is *(structurally) redundant* [Ber87] if there is a set of places Q – called *reference set* – with $q \notin Q$, a valuation $V : Q \cup \{q\} \to \mathbb{N}$ and some $d \in \mathbb{N}_0$ which satisfy the following properties for all transitions t:

(1) $V(q)M_N(q) - \sum_{p \in Q} V(p)M_N(p) = d$

(2) $V(q)(W(t,q) - W(q,t)) - \sum_{p \in Q} V(p)(W(t,p) - W(p,t)) \geq 0$

(3) $V(q)W(q,t) - \sum_{p \in Q} V(p)W(p,t) \leq d$

If V is not explicitly mentioned, it is implicitly given by $V(q) = 1$, $V(p) = 1$ if $p \in Q$ and $V(q) = 0$ otherwise. $\triangle$

Redundant places are implicit (but in general not vice versa). Since the techniques for the detection of implicit and redundant places resp. are still not efficient enough, only the subset of *shortcut places* is deleted. An MG-place p is a shortcut place if there is an MG-path w between $t \in {}^\bullet p$ and $t' \in p^\bullet$ with $M_N(p) \geq M_N(w)$. It is easy to see that shortcut places are indeed redundant.

We conclude this section by introducing some operations to deal with signal insertion. In particular, *gyroscope insertion* inserts essentially what is known as toggle-transition.

Definition 2.4 (Place-refinement, subnet-contraction, gyroscope insertion [WWSV09])
Let N be an STG.
(1) For a place $p \in P$, consider a net N' (cf. Figure 4) with:

- $P' = P \setminus \{p\} \cup \{p_{in}, p_{out}, p_1, p_2\}$
- $T' = T \cup \{g_1, g_2\}$
- $W'(x,y) = W(x,y)$ if $x, y \in P \setminus \{p\} \cup T$
  $W'(t, p_{in}) = W(t,p)$ for $t \in T$
  $W'(p_{out}, t) = W(p,t)$ for $t \in T$
  $W'(p_i, g_i) = W'(g_i, p_{3-i}) = W'(p_{in}, g_i) = W'(g_i, p_{out}) = 1$, $i = 1, 2$
- $M_{N'}(p') = M_N(p')$ for $p' \in P \setminus \{p\}$
  $M_{N'}(p_1) = 1$, $M_{N'}(p_2) = 0$
  $(M_{N'}(p_{in}), M_{N'}(p_{out})) = (in, out)$ with $in + out = M_N(p)$

In N', the labels of the new transitions and their signature can be chosen arbitrarily. Starting from N, N' is called a *place-refinement of p with initial marking* (in, out); starting from N', N is called a *subnet-contraction* if g_1 and g_2 are λ-transitions.

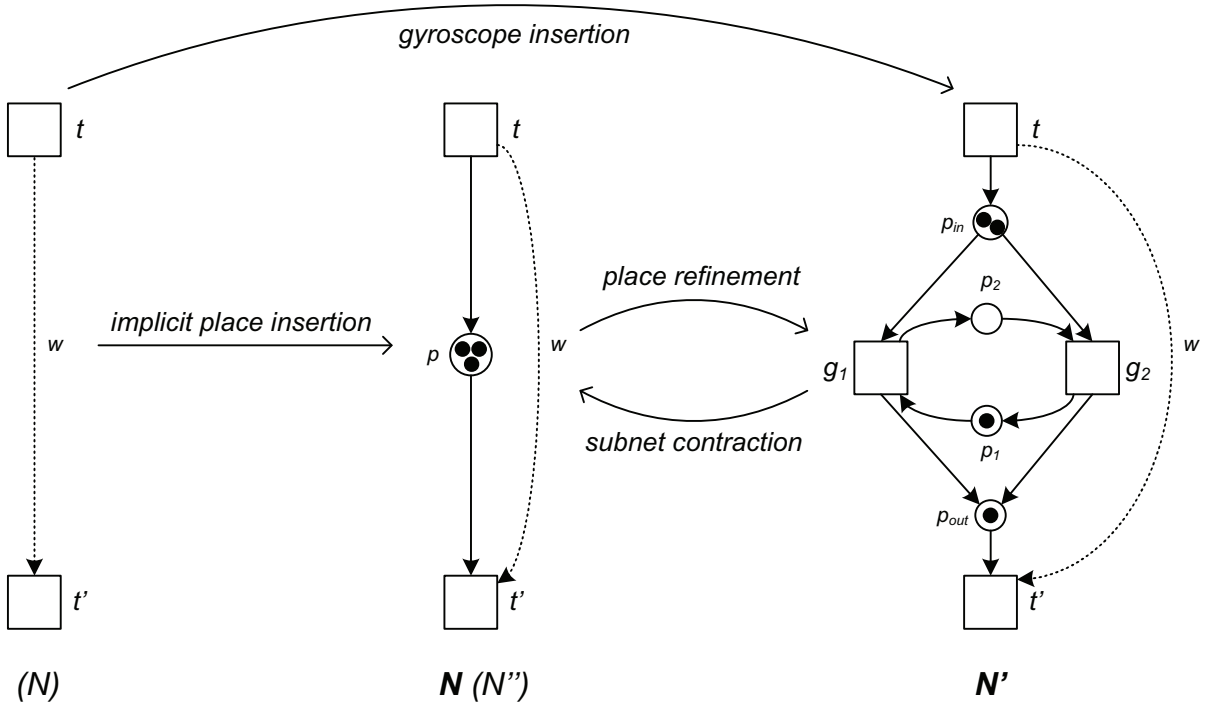

Figure 4: Gyroscope insertion with initial marking $(2, 1)$ via place insertion (here: a shortcut place due to w) and place refinement.

(2) A *gyroscope insertion* with initial marking (in, out) inserts a new implicit place $p \notin P$ with $in + out$ tokens into N (giving the intermediate N'') and applies place refinement with initial marking (in, out) to it (giving N').

A gyroscope insertion is called an *input/output/internal gyroscope insertion* if g_1 and g_2 are labelled in N' with s^+, s^- resp. and s is a fresh input, output or internal signal resp.; it is called a *dummy gyroscope insertion* if g_1 and g_2 are labelled with λ. △

Subnet-contraction is not really intended as reduction operation. But in principle, one could try to apply it if only backtracking is the alternative, though the odds for this to succeed seem to be low.

2.4 STG Decomposition

For the STG decomposition algorithm, from [VW02, VK06] a partition of the output signals of the given *specification* STG N is chosen, and the algorithm decomposes N into component STGs, one for each set in this partition. For synthesis, from each component equations for the corresponding outputs are derived from the respective state graph, instead of deriving the equations from the state graph of N.

Very often, the cumulated states of all component state graphs give a number much smaller than the state count of N, in which case the decomposition can be seen as successful. Actually, it might already be beneficial if each state graph is smaller than the one of N, in particular for reducing peak memory usage.

Of course, the behaviour of the specification should be preserved in some sense; this is captured by a variant of bisimulation, tailored to the specific needs of asynchronous

circuits:

Definition 2.5 (Correct Decomposition [SV07])
A collection of deterministic components $(C_i)_{i\in I}$ is a *correct decomposition* of (or simply *correct w.r.t.*) a deterministic STG N – also called *specification – when hiding* H, if $C = (||_{i\in I} C_i)/H$ is defined, $In_C \subseteq In_N$, $Out_C \subseteq Out_N$ and there is an *STG-bisimulation* $\mathcal{B}$ between the markings of N and those of C with the following properties:
$(M_N, M_C) \in \mathcal{B}$ and for all $(M, M') \in \mathcal{B}$, we have:
(N1) If $a \in In_N$ and $M[a^{\pm}\rangle\rangle M_1$, then either $a \in In_C$, $M'[a^{\pm}\rangle\rangle M_1'$ and $(M_1, M_1') \in \mathcal{B}$ for some M_1' or $a \notin In_C$ and $(M_1, M') \in \mathcal{B}$.
(N2) If $x \in Out_N$ and $M[x^{\pm}\rangle\rangle M_1$, then $M'[vx^{\pm}\rangle\rangle M_1'$ and $(M_1, M_1') \in \mathcal{B}$ for some M_1' with $v \in (Int_C^{\pm})^*$.
(N3) If $u \in Int_N$ and $M[u^{\pm}\rangle\rangle M_1$, then $M'[v\rangle\rangle M_1'$ and $(M_1, M_1') \in \mathcal{B}$ for some M_1' and $v \in (Int_C^{\pm})^*$.
(C1) If $x \in Out_C$ and $M'[x^{\pm}\rangle\rangle M_1'$, then $M[vx^{\pm}\rangle\rangle M_1$ and $(M_1, M_1') \in \mathcal{B}$ for some M_1 with $v \in (Int_N^{\pm})^*$.
(C2) If $x \in Out_i$ for some $i \in I$ and $M'|_{P_i}[x^{\pm}\rangle\rangle$, then $M'[x^{\pm}\rangle\rangle$. (no *computation interference*)
(C3) If $u \in Int_C$ and $M'[u^{\pm}\rangle\rangle M_1'$, then $M[v\rangle\rangle M_1$ and $(M_1, M_1') \in \mathcal{B}$ for some M_1 and $v \in (Int_N^{\pm})^*$.

Here, and whenever we have a collection $(C_i)_{i\in I}$, P_i stands for P_{C_i}, Out_i for Out_{C_i} etc. If $H = \emptyset$, we simply say that $(C_i)_{i\in I}$ is correct w.r.t. to N. △

In a simple case, $(C_i)_{i\in I}$ consists of just one component C_1 (immediately implying (C2)). For instance, this C_1 could be the result of solving a CSC conflict in N; if C_1 is correct w.r.t. N, then it has indeed the same behaviour as N.

(C2) ensures that no computation interference occurs, i.e. if a component produces an output (which is under the control of this component), then the other components expect this signal if it belongs to their inputs, and no malfunction of these other components must be feared. (C2) is actually also satisfied for $x \in Int_i$, since internal signals of one component are by definition unknown to the other components.

A speed-independent circuit operates in input/output mode with its related environment, i.e. output changes are generated by the circuit as a consequence of certain input changes which in turn are reactions on certain output changes again. Hence, an STG specifying an SI circuit has to be *input proper*, i.e. an input might not be activated by an internal edge, otherwise the environment might produce the input before the internal signal is produced by the circuit. Alternatively, one could make timing assumptions for the environment, such that an internal signal change is always faster than an input change. In this case, the enabling of an input by an internal signal should not be interpreted as a causal but a temporal relation.[4]

In [SV07] it was shown that the above correctness notion implies that the STG C in Definition 2.5 is in a sense input proper; in particular, if the solution of a CSC-conflict inserts an internal signal in front of an input, it is not correct in the sense of Definition 2.5. Also, correctness is essentially transitive.

We now discuss our decomposition algorithm in more detail. In the following, we

[4]This can be modelled by a so-called tcb-concurrency [WB00].

assume that we are given a deterministic, consistent specification N without internal signals[5], and for this paper, we assume all arcs of N to have weight 1.

First, one chooses a *feasible partition*, i.e. a family $(In_i, Out_i)_{i \in I}$ for some set I such that the sets Out_i are a partition of Out, $In_i \subseteq Sig \setminus Out_i$ for each i and furthermore:

- If two output signals x_1, x_2 are in structural conflict in N, 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 $l(t') \in Out_i$ implies $l(t) \in In_i \cup Out_i$.

Observe: if we have a feasible partition, we can build another feasible one by adding additional input signals to one of the members.

For each member (In_i, Out_i) of the partition, an *initial component* is generated from N: in a copy of the original STG N, every signal not in $In_i \cup Out_i$ is lambdarised and the signals in In_i are considered as inputs of this component – even if they are outputs of N. Then the following three *reduction operations* are applied to an initial component until no more λ-labelled transitions remain:

- secure contraction of a λ-labelled transition
- deletion of an implicit place
- deletion of a redundant transition

Unfortunately, it is not always possible to contract all λ-transitions. Besides the technical cases where the contraction is not defined or not secure (possibly leading to an incorrect decomposition), the contraction might also generate a new auto-conflict. The latter reveals non-determinism which is present in the respective initial component, but not in the specification. This indicates that the component has not enough information to properly produce its outputs. Such a contraction is *disallowed* and consequently a new signal is added as follows.

If λ-transitions remain, *backtracking* is applied, i.e. a new input is added to the component. Technically, this input is added to the initial partition and the new corresponding initial component is derived and reduced from the beginning. The new input signal is taken from the former label of a non-contractible λ-transition. As discussed above, the new partition is feasible again. This cycle of reduction and backtracking is repeated until all λ-transitions of the initial component can be contracted.

In principle, every so-called *totally admissible operation* [VK06] can be used for reduction. It is proven in [VK06] that the decomposition algorithm using arbitrary totally admissible operations always returns a *correct* decomposition. The precise definition of totally admissible is not important here; it is enough to know that the three operations from above as well as subnet-contraction [WWSV09] from Definition 2.4 are totally admissible. Furthermore, it will be relevant below that such operations have to preserve some invariants; in particular, they have to turn an STG satisfying (a) into one that satisfies (a) again, where

[5]For the decomposition algorithm, internal signals can be considered as outputs; see [SV07] for more details.

(a) There is no structural λ/output conflict, i.e. between a λ-transition and one labelled with an output.

Now we present a new generalisation of the decomposition algorithm; the point is that we can relax the definition of a feasible partition, i.e. that we can apply essentially the same algorithm with additional starting points. We still consider the same kind of specification N as above.

Definition 2.6 (Quasi-Feasible Partition)
A *quasi-feasible partition* is a family $(In_i, Out_i)_{i\in I}$ for some set I such that the sets Out_i are a partition of Out, $In_i \subseteq Sig \setminus Out_i$ for each i and furthermore:

- If two output signals x_1, x_2 are in *dynamic* conflict in N, 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 $l(t') \in Out_i$ implies $l(t) \in In_i \cup Out_i$. $\triangle$

In the definition of a totally admissible operation, condition (a) from above has to be changed to: there is no *dynamic* λ/output conflict. This modified (a) is also used as an invariant for the proof of our new correctness theorem for decomposition.

Theorem 2.7
Also when starting from a quasi-feasible partition, decomposition of a deterministic N (using totally admissible operations defined with the modified (a)) result in deterministic components that form a correct implementation of N. If only the operations listed in this paper are used and N is consistent (free of dynamic io-conflicts resp.), then so are the components.

Proof. The proof follows the lines of the proof of Theorem 4.1 in [VK06]. Carefully checking the all in all eight (sub-)proofs, one sees that no essential changes are needed when working with the modified (a) and also considering only the really essential (for SI-synthesis) dynamic conflicts between an input and an output transition instead of the structural ones. There are two exceptions though: for proving that secure transition contraction is totally admissible, one has to fill two proof gaps for Lemma 4.3 and 4.4.

We sketch how this can be done and start with Lemma 4.4. In the proof, all markings considered in angelic bisimulations are now assumed to be reachable – this is no problem. In case (c) for $i = j$, contraction is applied to a transition t of a component, let us call it C, and the result is C'; C satisfies (the now modified) (a). A reachable marking of C, call it M, is related by the marking equality to a reachable marking M' of C'; an output transition t_1 is enabled under M', and one has to show that it is also enabled under M.

For a contraction of type 1, the proof in [VK06] is still valid, and we still can assume that ${}^\bullet t_1 \cap t^\bullet$ is empty. Assume that t_1 is not enabled under M; this must be due to places in the common preset of t and t_1 (so we clearly have a type-2 contraction). Let k be the maximal number of tokens on some of these places missing for enabling t_1.

Since $M'[t_1\rangle$, there must be at least k tokens on each place in $t^\bullet$ under M (so that each new place containing the 'critical' place from ${}^\bullet t_1$ allows t_1 to fire in C'). These tokens have been put there by t (we consider a type-2 contraction, and in particular some place in $t^\bullet$ was empty initially); hence t has fired at least k times in the firing sequence w reaching M. The tokens were not needed by any transition for reaching M; thus, we can delete

the last k occurences of t in w, getting a firing sequence reaching some marking M''. M'' has k more tokens on each place in ${}^\bullet t$ compared to M, so it enables t_1. By choice of k, firing t under M'' disables t_1, while firing t_1 empties some place in ${}^\bullet t$, i.e. it disables t. This is a contradiction to (a), and we are done.

We continue with the proof gap for Lemma 4.3, which concerns Part (1). Here, we have to consider a component C satisfying the modified condition (a) – and for the claim about input/output conflicts in the theorem, we also have to consider the assumption that there are no dynamic input/output conflicts. Further, there is a transition t to be contracted in C, an output transition t_2 with ${}^\bullet t_2 \cap {}^\bullet t \neq \emptyset$ (i.e. the contraction is of type 2) and a dummy transition t_1 with ${}^\bullet t_1 \cap t^\bullet \neq \emptyset$; for the claim about input/output conflicts, we also have to consider the case that t_1 is an input transition. Assume that in C' (i.e. after the contraction) there is a dynamic conflict between t_2 and t_1 under reachable marking M'; we will derive a contradiction using (a) (freeness from input/output conflicts, resp.) and the fact that there is no dynamic conflict between t_2 and t in C (no dynamic conflict between t_2 and input transition t_1 resp.).

By Thm 3.3 (1) of [VK06], there is a reachable marking M of C related to M' by the marking equality (restrict the simulation in 3.3 (1) to reachable markings). By what we have shown above, we know that M enables t_2.

Let k be the maximal number of tokens that is missing on some $p \in {}^\bullet t \cap {}^\bullet t_1$ to enable t_1 under M. (In particular, if there is no place in the intersection, k is 0.) If k is positive, there must be k surplus tokens on all places in $t^\bullet$, and we can 'unfire' t k times as above; in this case, we now have a modified M, still satisfying the marking equality with M' and enabling t_2, such that for some $p \in {}^\bullet t \cap {}^\bullet t_1$, there are just enough tokens on p to enable t_1 under M.

If $k = 0$, it might be that all places in ${}^\bullet t$ have more tokens than needed to enable t_1 (this is true for some $p \notin {}^\bullet t_1$ if it has at least one token). Then we fire t until some $p \in {}^\bullet t$ has just enough tokens to enable t_1 under M (possibly, $p \notin {}^\bullet t_1$ is empty). Again, the modified M still satisfies the marking equality with M' and enables t_2.

Let $p_1 \in {}^\bullet t_1 \cap t^\bullet$; since (p, p_1) has enough tokens to enable t_1 under M', this means that p_1 has enough additional tokens that guarantee that t_1 is enabled under M as far as p_1 is concerned. Since this works for arbitrary $p_1 \in {}^\bullet t_1 \cap t^\bullet$, t_1 is enabled under M.

By (a) (freeness from input/output conflicts, resp.), any place not adjacent to t has enough tokens under M (or M') to enable t_1 and t_2 together. For any new place (p', p_1), p' also has enough tokens under M and the same holds for p_1 since M enables t_1 and $p_1 \notin {}^\bullet t_2$. Thus, (p', p_1) inherits enough tokens to satisfy independently the needs of t_2 and the combined needs of t_1, resulting from arcs from p' and p_1 to t_1. This is a contradiction to the assumption that there is a dynamic conflict between t_2 and t_1 under M'. □

3 Internal Communication for SI Synthesis

The pure application of STG decomposition can lead to irreducible CSC conflicts. Ignoring the cloud symbols in Figure 5, component C_r (including place p_{st}) has a self-trigger corresponding to t_{en} and t_{ex} (see the shaded states in (b)) although the initial specification N has none. Component STGs having such irreducible CSC conflicts are called *critical*, since they are not SI implementable.

However in the context of STG decomposition irreducible conflicts can be avoided

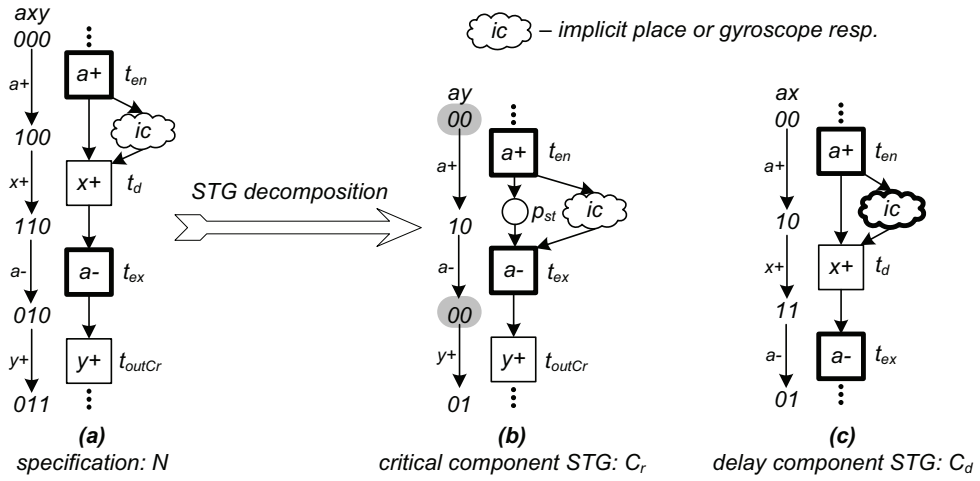

Figure 5: Self-trigger avoidance by internal communication.

1. by inserting an internal gyroscope labelled with a new signal ic 'between' the entry transition t_{en} and a so-called *delay transition* t_d (having a local signal) at the cloud's position in Figure 5a, and

2. by applying a second decomposition pass for C_r and the component STG C_d producing the signal of t_d (called the *delay component*), where ic is considered as an additional output of C_r and a necessary input of C_d; finally, the old versions of C_d and C_r have to be replaced with their recalculated versions.

This can avoid the conflict or make it at least reducible and so the former critical component is now SI implementable [WWSV09].

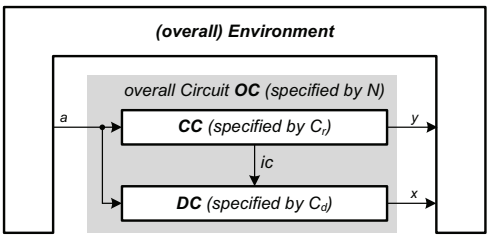

Figure 6: Circuit architecture of the resulting circuit.

From the circuit's perspective this introduces an internal communication via signal ic from the critical component to the delay component, as shown in Figure 6.

The key to apply this approach is to identify a delay transition in N. Consider a self-trigger of C_r between the markings M_1 and M_2, i.e. there is a transition sequence $M_1[t_{en}t_{ex}\rangle M_2$ and (t_{en}, t_{ex}) is the *entry/exit transition pair* of the self-trigger. Recall that the transitions are labelled with the same input signal of C_r, but complementary edges; we call the signal edge which is the label of t_{ex} *critical edge*, since its fast occurrence can lead to malfunction of the circuit; our aim is to delay the occurrence of the critical edge. A place p_{st} with $t_{en} \in {}^\bullet p_{st}$ and $t_{ex} \in p_{st}{}^\bullet$ is called *self-trigger place*, see also Figure 5b.

Definition 3.1 (Delay Transition)
A transition t_d of N is called *delay transition* w.r.t. a transition pair (t_{en}, t_{ex}) if t_d is labelled with a local (i.e. non-input) signal edge and t_d occurs in *all* transition sequences t_{en} ... t_{ex} enabled under a reachable marking of N. △

Intuitively, if t_{ex} fires after t_{en} then t_d must fire in between. Observe that delaying t_d also delays the firing of t_{ex}, i.e. the critical edge – and this avoids the self-trigger.

In the next sections, we only consider self-trigger avoidance, since self-triggers appear very often in our benchmark examples, cf. Section 5. Furthermore, a self-trigger is the most severe type of irreducible CSC conflict – w.r.t. to our approach – since only one transition pair (t_{en}, t_{ex}) can be identified for which a delay transition has to be found. In the remainder of this section we propose how to deal with general irreducible CSC conflicts by identifying several transition pairs that yield several opportunities to determine a delay transition.

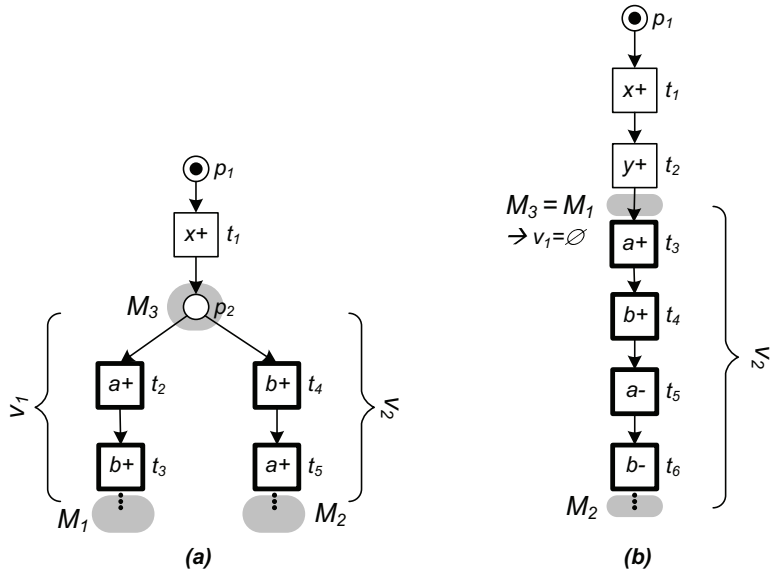

Figure 7: Components having irreducible CSC conflicts between M_1 and M_2: (a) a type I conflict in terms of [KMY06] and (b) a conflict of type II.

Consider the two critical components in Figure 7 having an irreducible CSC conflict between M_1 and M_2 each. Such conflicts can be characterised by a third marking M_3 and two traces $M_3[w_1\rangle\rangle M_1$ and $M_3[w_2\rangle\rangle M_2$, where w_1 and w_2 consist of input events only. In principle Mpsat and Petrify are able to report these traces, and they can be mapped to corresponding firing sequences v_1 and v_2; observe that v_1 could even be empty, as in Figure 7b. Following [KMY06] it is enough to consider conflict types as in Figure 7, since other CSC conflicts can be reduced to these types of conflicts; Figure 7a refers to a typical type II conflict (according to [KMY06]) and 7b to a type I conflict.

Figure 7 also demonstrates that, for general irreducible CSC conflicts, there are usually several pairs (t_a, t_b) – like an entry/exit transition pair – where one can look for a delay transition that fires between both transitions. For example in Figure 7a we can identify the

pairs $(t_2, t_3), (t_4, t_5)$ and in Figure 7b $(t_3, t_4), (t_5, t_6), (t_3, t_5)$ etc. Irreducible CSC conflicts can be avoided if our approach succeeds for just one of those transition pairs. Hence, the focus on self-triggers (which exhibit only one possibility for identifying an entry/exit transition pair) is not a real restriction.

4 Self-Trigger Avoidance for Unweighted STGs

In [WWSV09], it was shown how to avoid a self-trigger even *without* recalculation of the critical and the delay component, but only under strong restrictions: for a critical component with a self-trigger as in Figure 5b, the corresponding specification N must have a structure as in Figure 8a: there must be a restricted *MG-path* w from a transition t_1 labelled with a local signal edge of the delay component to the exit transition t_{ex} via the entry (t_{en}) and the delay transition (t_d); namely, all transitions of w between t_1 and t_{en} (excluding t_1) must have only one incoming arc and all transitions between t_d and t_{ex} (excluding t_{ex}) must have only one outgoing arc. The sub-STG between t_{en} and t_d must be a path where all transitions and all places have one incoming and one outgoing arc, only. In other words, the path from t_1 to t_d is non-joining, the one from t_{en} to t_{ex} is non-forking. These strong restrictions make it relatively easy to find a suitable path, but only in those cases where such a path exists.

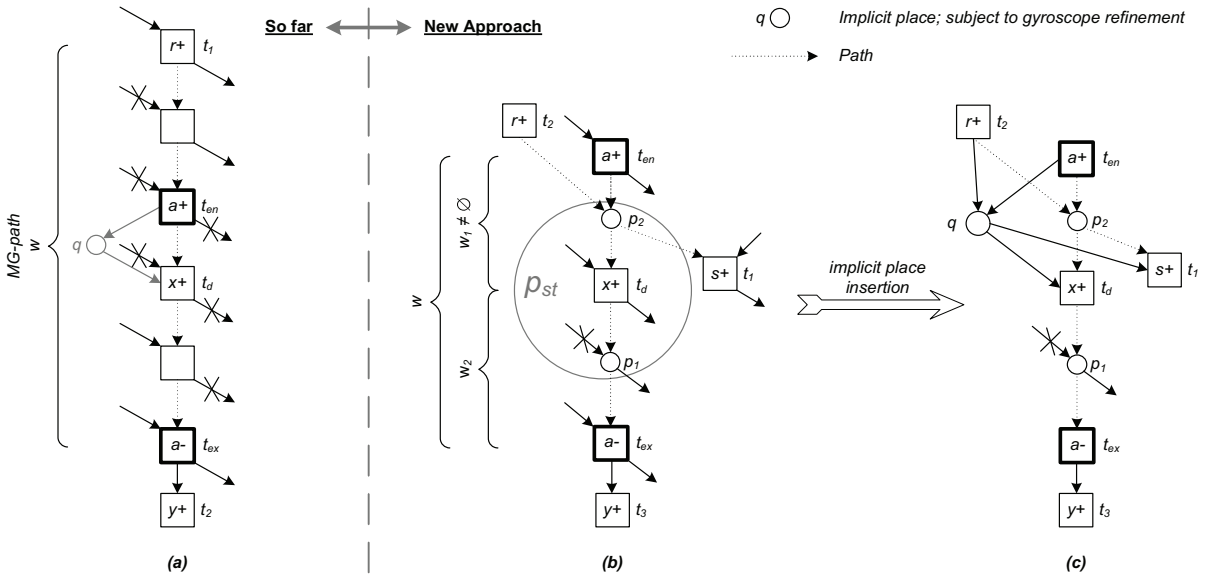

Figure 8: Specifications N yielding at least one delay candidate t_d for a potential self-trigger, given by t_{en} and t_{ex}; p_{st} sketches a self-trigger place of C_r as a result of reducing all elements in w, cf. also Figure 5b.

Here, we will show how to deal with much more general cases, e.g. like the one in Figure 8b, accepting a second reduction pass for the critical and the delay component. In Section 4.1, we propose a structural method to identify a delay transition for a given self-trigger, and we present a structural technique to insert an implicit place into a Petri net as a necessary step for the internal gyroscope insertion into N in Section 4.2. Finally, we propose in Section 4.3 how to avoid uncontrollable growth of the delay component

(i.e. preventing the delay component from generating unnecessary output signals) by introducing internal communication between the critical component and several auxiliary components.

4.1 How to Identify a Delay Transition

A delay transition is defined based on traces of the specification N (s. Definition 3.1), but due to the complexity of N we need a structural condition to identify such a transition. Since it is difficult to find a necessary condition that is not too restrictive or a really sufficient condition (that is *only* satisfied for delay transitions), we propose an indication for a delay transition – and just speak of a delay candidate:

Definition 4.1 (Delay Candidate)
For a specification N and an entry/exit transition pair (t_{en}, t_{ex}), a *delay candidate* t_d is a local transition on a path $w = w_1 w_2$ starting from t_{en} leading to t_{ex}, where w_1 is a path from t_{en} to t_d ($t_{en} \neq t_d$), while w_2 is a path from t_d to t_{ex} without merge places, i.e. $\forall p \in w_2 \, |{}^\bullet p| = 1$. No transition on $w \setminus \{t_{en}, t_{ex}\}$ has its signal in Sig_r. △

Observe that the path w_2 could even be empty; in this case there are no restrictions on the specification's structure. The non-merging path w_2 indicates that t_d has to fire before t_{ex}, except that the initial marking of w_2 might initially allow to fire t_{ex} without firing t_d. The last condition reflects the idea that w is contracted to p_{st} in C_r (cf. Figure 8b), i.e. none of its transitions except t_{en} and t_{ex} is *r-relevant*; the signals of C_r are called r-relevant, in particular when considered in the context of the full STG N.

Although a delay candidate is not necessarily a delay transition according to Definition 3.1, our criterion is sufficient for many benchmark examples – see our examples in Section 5.

4.1.1 Algorithmic Solution

Instead of simply searching for all paths w (from t_{en} to t_{ex}) containing local signal transitions (as possible delay candidates), we suggest to apply a more efficient technique:

First, we can ignore all transitions labelled with an *r-relevant* signal edge except t_{en} and t_{ex}; we delete them as well as all arcs to t_{en}[6] in N resulting in N'.

Second, we apply an ordinary depth-first search (DFS) [CLRS01] starting at t_{en} in order to determine all transitions in N' that are reachable from t_{en}; the others are removed as well. One could also perform a breadth-first search and store the distances from t_{en}; this would help to find a delay candidate close to t_{en}, see below.

Third, in N' a *backward* search starting at t_{ex} will be applied to find all paths w_2 from a delay candidate t_d to t_{ex}; the search can be restricted, because only non-merging places p should be visited (i.e. $|{}^\bullet p| = 1$). To find all such paths, we modify the backward-directed DFS [CLRS01]: when going backward, vertices are marked as visited as usual (this avoids repeated vertices on a path), but when returning from the recursion, the mark is erased (and the vertex can be used in other paths). One can also consider just one or a few paths, hoping that the respective candidates will help to avoid the self-trigger.

After a potential path w_2 is found, we apply a backward-directed breadth-first search (BFS) to find a shortest path w_1 from t_{en}. For this, all the vertices of w_2 must be marked

[6]The arcs from t_{ex} do not matter for Subsection 4.1.1.

as visited, initially. Only if t_{en} is reachable, t_d is a delay candidate. The idea is that the algorithm in Section 4.2 starts at t_{en} and tries to reach some t_d quickly; hence, t_d should be chosen such that w_1 is short.

Usually, there are several possible paths w_2 and so several possible delay candidates. It is subject to future work to investigate how the delay candidate selection influences the synthesis time and the resulting circuit area and performance.

4.2 Inserting Implicit Places into a Petri Net

For a given entry transition t_{en} and a delay candidate t_d (with a path w_1 from t_{en}) the algorithm in Figure 9 inserts a redundant place q (w.r.t. some set Q) into an unweighted STG N such that $t_{en} \in {}^\bullet q$ and $t_d \in q^\bullet$.

The insertion of q will be initiated by calling the function insertImplicitPlace. In line 8, a *forward traversal* in N from t_{en} towards t_d will be started by calling the place function with the unique place p as argument such that $p \in w_1$ and $p \in t_{en}{}^\bullet$. Every time the function place is processed as well as in lines 27 or 31, a redundant place q could be inserted via the operations specified in lines 9 and 10. The redundancy of q is assured, since for every call of place and every new place p of Q, ${}^\bullet p$ as well as $p^\bullet$ will be added to the potential preset (preq) and postset of q (postq) – except for loop transitions – see lines 25 and 26 (for a formal proof, see below). Consequently, for every incoming token to p a token is added to q, for every token removed from p a token is removed from q, too. If the redundant place q were unsafe, then we would get a CSC conflict due to firing two edges of the new internal signal ic; thus p is not added if it would lead to two tokens on q initially, see line 23.

To make q useful, the preset and postset of q must fulfil some application specific requirements which are tested by calling transitionF for each transition in $p^\bullet$ (line 28) and transitionB for each transition in ${}^\bullet p$ (line 29) – F and B resp. indicate a *forward or backward traversal*. The functions checkTransition{F/B} in line 14 are used to terminate the traversal at certain transitions, see Figure 10. They are tuned to make $q^\bullet$ consist of local transitions only – including t_d – such that the later gyroscope refinement of q with a new internal signal ic is input proper according to [SV07], which means that no input will be delayed by ic. Furthermore, ${}^\bullet q$ should consist of t_{en} and other transitions labelled with r-relevant edges only, such that the recalculated critical component C'_r gets no additional relevant signals, except for ic as an additional output; this avoids uncontrollable growth of C'_r.

If checkTransition{F/B} does not return to stop the search, then transition{F/B} calls the place function again to find a suitable place in the postset or preset resp. of t. With such a call, the post- and preset of q are extended until these fulfil the requirements specified in checkTransitionF and checkTransitionB resp. Note that the algorithm is correct for arbitrary functions checkTransition{F/B}, see Proposition 4.2 below.

To ensure that $t_d \in q^\bullet$, the algorithm applies a forward traversal straight to t_d as long as the delayFound flag is not set, see lines 16 – 18. Similarly, in the implementation of place, the next transition t' on w_1 is chosen first in line 28 (not shown). Eventually, the flag will be set by checkTransitionF, and then all the other possible paths starting from the places in w_1 will be traversed in forward and backward direction.

The algorithm works on a net N and constructs a copy N' extended with q in lines 9 and 10.

```
   // Global variables
1  multiset of transitions  preq, postq     // potential pre- and postset of q
2  set of places            Q
3  set of places            w_1             // shortest path from t_en to t_d
4  boolean                  delayFound

   // Main function
5  bool insertImplicitPlace()
6     preq ← ∅ ; postq ← ∅ ; Q ← ∅ ; delayFound ← false
7     p ← w_1 ∩ t_en•
8     if (place(p, t_en))
9        P' ← P ∪ {q} ; M_N'(q) ← M_N(Q)
10       for all t ∈ T do W'(t, q) ← preq(t) ; W'(q, t) ← postq(t)
11       return true
12    else return false

13 bool transitionF(transition t)              bool transitionB(transition t)
14    (stop, value) ← checkTransitionF(t)        (stop, value) ← checkTransitionB(t)
15    if (stop) return value                        ...
16    if (delayFound = false)
17       p ← w_1 ∩ t•
18       return place(p, t)
19    for all p ∈ t• \ Q do                      for all p ∈ •t \ Q do
20       if (place(p, t)) return true               ...
21    return false                                  ...

22 bool place(place p, transition t)   // p, t adjacent
23    if M_N(Q) + M_N(p) > 1 return false
24    Q ← Q ∪ {p}
25    preq ← preq + •p ; postq ← postq + p•
26    (preq, postq) ← (preq − postq, postq − preq)    // removes loops
27    res ← true
28    for all t' ∈ p• − {t} ∧ postq(t') > preq(t') do res ← res ∧ transitionF(t')
29    for all t' ∈ •p − {t} ∧ postq(t') < preq(t') do res ← res ∧ transitionB(t')
30    if (¬res) restore old values of in, out and Q  // before the present call of place
31    return res
```

Figure 9: Algorithm InsertImplicit for inserting an implicit place q. It is assumed that t_{en}, t_{ex} and t_d are known to the algorithm as global constants. '...' means line repetition from transitionF.

```
1   (bool,bool) checkTransitionF(transition t)
2     if (t = t_en) return (true,false)
3     if (delayFound = false)
4       if (t = t_d) delayFound ← true ; return (true, true)  // termination successful
5       if (t ∈ w_1) return (false,false)
6     if (l(t) ∈ Loc_N^±) return (true,true)                  // termination successful
7     if (t = t_ex) return (true,false)
8     return (false,false)
9   (bool,bool) checkTransitionB(transition t)
10    if (t = t_ex) return (true,false)
11    if (l(t) ∈ Sig_r^±) return (true,true)                  // termination successful
12    return (false,false)
```

Figure 10: Strategy CheckTransition. The functions know all global variables of InsertImplicit and check validity of transitions for the preset (checkTransitionB) and the postset (checkTransitionF) of q – with result $(stop, value)$. If $stop$ = false, the traversal should be continued and t is not valid ($value$ = false). Otherwise, $value$ says whether t is valid (successful termination) or not; in the latter case, place has to backtrack and to remove its current p from Q.

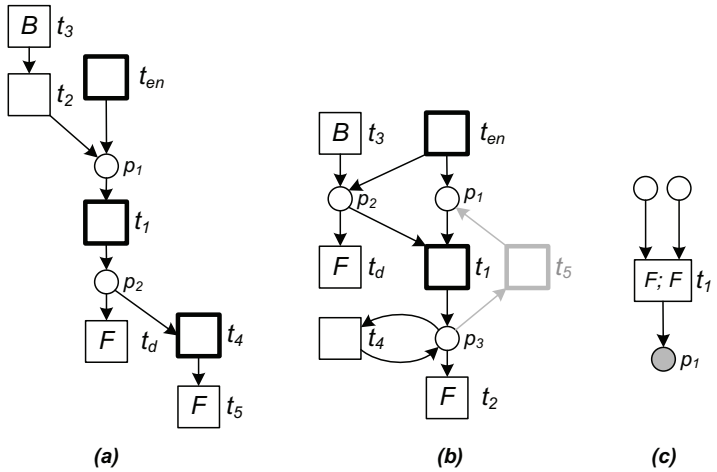

Figure 11: Examples for InsertImplicit: the loops in lines 28 and 29 process the resp. nodes from left to right; at transitions labelled with F the forward search terminates successfully, and analogously for B.

We try to clarify how InsertImplicit works with the help of some examples shown in Figure 11. For the example in (a), w_1 is $t_{en}p_1t_1p_2t_d$; thus, the algorithm calls place(p_1, t_{en}) (e.g. t_1 is added to postq), transitionF(t_1), place(p_2, t_1) (e.g. t_1 is removed from postq) and transitionF(t_d). Now flag delayFound is set and the last call returns successfully. Next, transitionF(t_4) does not directly terminate since t_4 is an input transition, but the next call of transitionF returns successfully as well. Going back to the call place(p_1, t_{en}), transitionB(t_2) and then transitionB(t_3) are performed. In the end, a redundant place q with ${}^\bullet q = \{t_{en}, t_3\}$ and $q^\bullet = \{t_d, t_5\}$ is inserted.

Next consider Figure 11b, first without transition t_5. The forward traversal reaches t_d, and then transitionF(t_1) and place(p_3, t_1) are called. Here, t_4 is added to preq and postq in line 25, and removed again in line 26. Thus, for t_4 neither transitionF nor transitionB are called in lines 28 or 29. Eventually, the algorithm inserts a redundant place q, where ${}^\bullet q = \{t_{en}, t_3\}$ and $q^\bullet = \{t_d, t_2\}$.

Now consider transition t_5; when reaching p_3, we get calls transitionF(t_5), place(p_1, t_5) and transitionF(t_1). Here, the check requires to continue (t_1 is an input transition), but the algorithm fails in line 19, since there is no place left in ${t_1}^\bullet \setminus Q$. Note that the repeated traversal of places in Q – like p_3 – would prevent the termination of the algorithm.

Figure 11c indicates a situation, where t_1 has already been visited by forward traversals – and declared as suitable for $q^\bullet$ via line 6 of checkTransitionF – for two times. Now place p_1 is reached via a call place(p_1, t), and t_1 is added to *preq* in line 25; this would make t_1 a (generalised) loop transition for q. But after line 26, $preq(t_1) = 0$ and $postq(t_1) = 1$, and due to the condition in line 29, transitionB will not be called for t_1; the search terminates by returning true.

Proposition 4.2
For any terminating checkTransition{F/B} operation without side-effects on N, Q, preq or postq, the algorithm InsertImplicit *terminates such that the place q is implicit in N'. If* false *is returned, no place q will be inserted.*

Proof. First observe the following properties: ($*$) for every call of place(p, t), we have $p \notin Q$; this follows immediately from lines 7 and 8, 19 and 20 resp., where these are the only places from which place(p, t) is called.

($**$) The number of calls of place on the call-stack is bounded by $|P|$; directly from ($*$).

Termination: Obviously, insertImplicitPlace() is only called at the beginning, and transition{F/B}() and place() are calling each other alternately in the sense that they alternate on the call-stack. Together with ($**$), this implies that the call-stack has bounded depth. Since only **for** loops are used, only finitely many functions are called in each function call f(), and eventually a function is called or f() returns. This implies the claim.

Correctness: Now, we prove by induction that the properties of Definition 2.3 are fulfilled every time a function is called or returned from. More precisely, we show that q would be redundant for the valuation $V \equiv 1$ if N would be modified as in lines 9 and 10. Hence, we only have to consider the values of *preq*, *postq* and Q.

Clearly, q is redundant for $V \equiv 1$ after the initialisation of the global variables in line 6: q is an unconnected place then, and $d = 0$. Furthermore, we only have to consider the lines 24–26, since only here *preq*, *postq* and Q will change significantly. If they are restored (to the values before line 24) in line 30, q is redundant by induction assumption.

We will denote the new values (i.e. after execution of line 26) with $preq'$, $postq'$, W' etc. Observe that $M_{N'}(p) = M_N(p)$ for every place $p \neq q$ and $W'(x,y) = W(x,y)$ if $x \neq q \neq y$.

(1)

$$\begin{aligned} & d' \\ = \; & M_{N'}(q) - \sum_{s \in Q'} M_{N'}(s) \\ \text{(definition of } M_{N'}(q)) \; = \; & M_{N'}(Q') - \sum_{s \in Q'} M_{N'}(s) = 0 = d \end{aligned}$$

(2)

$$\begin{aligned} & 0 \\ \leq \; & W'(t,q) - W'(q,t) - \sum_{s \in Q'} (W'(t,s) - W'(s,t)) \\ \text{(definition of } preq \text{ and } postq) \; = \; & preq'(t) - postq'(t) - \sum_{s \in Q'} (W(t,s) - W(s,t)) \\ \text{(lines 24–26 and } (*)) \; = \; & ((preq + {}^\bullet p) - (postq + p^\bullet))(t) \\ & -((postq + p^\bullet) - (preq + {}^\bullet p))(t) \\ & -(W(t,p) - W(p,t)) - \sum_{s \in Q} (W(t,s) - W(s,t)) \\ \text{(Definition 2.1)} \; = \; & \max(0, \underbrace{preq(t) + {}^\bullet p(t) - postq(t) - p^\bullet(t)}_{a}) \\ & - \max(0, \underbrace{postq(t) + p^\bullet(t) - preq(t) - {}^\bullet p(t)}_{-a}) \\ & -W(t,p) + W(p,t) - \sum_{s \in Q} (W(t,s) - W(s,t)) \\ (\ \max(0,a) - \max(0,-a) = a\ ) \; = \; & preq(t) + {}^\bullet p(t) - postq(t) - p^\bullet(t) \\ & -W(t,p) + W(p,t) - \sum_{s \in Q} (W(t,s) - W(s,t)) \\ \text{(definition of } preq \text{ and } postq, \; = \; & W(t,q) - W(q,t) + W(t,p) - W(p,t) \\ \text{e.g. } {}^\bullet p(t) = W(t,p) \leq 1) \; & -W(t,p) + W(p,t) - \sum_{s \in Q} (W(t,s) - W(s,t)) \\ = \; & W(t,q) - W(q,t) - \sum_{s \in Q} (W(t,s) - W(s,t)) \\ \text{(induction)} \; \geq \; & 0 \end{aligned}$$

(3)

$$
\begin{aligned}
& & & d' \\
& & \geq\ & W'(q,t) - \sum_{s \in Q'} W'(s,t) \\
& \text{(definition of } \mathit{postq}) & =\ & \mathit{postq}'(t) - \sum_{s \in Q'} W(s,t) \\
& \text{(lines 24–26 and } (*)) & =\ & ((\mathit{postq} + p^\bullet) - (\mathit{preq} + {}^\bullet p))(t) \\
& & & -W(p,t) - \sum_{s \in Q} W(s,t) \\
& \text{(Definition 2.1)} & =\ & \max(0, \underbrace{\mathit{postq}(t) + p^\bullet(t) - \mathit{preq}(t) - {}^\bullet p(t)}_{a}) \\
& & & -W(p,t) - \sum_{s \in Q} W(s,t)
\end{aligned}
$$

1. Case: $a \leq 0$

$$
\begin{aligned}
& \ldots & =\ & 0 - W(p,t) - \sum_{s \in Q} W(s,t) \\
& & \leq\ & W(q,t) - \sum_{s \in Q} W(s,t) \\
& \text{(induction and (1))} & \leq\ & d = d'
\end{aligned}
$$

2. Case: $a > 0$

$$
\begin{aligned}
& \ldots & =\ & \mathit{postq}(t) + p^\bullet(t) - \mathit{preq}(t) - {}^\bullet p(t) \\
& & & -W(p,t) - \sum_{s \in Q} W(s,t) \\
& \text{(definition of } \mathit{postq}, & =\ & W(q,t) + W(p,t) - \mathit{preq}(t) - {}^\bullet p(t) \\
& p^\bullet(t) = W(p,t) \leq 1) & & -W(p,t) - \sum_{s \in Q} W(s,t) \\
& & =\ & W(q,t) - \mathit{preq}(t) - {}^\bullet p(t) - \sum_{s \in Q} W(s,t) \\
& & \leq\ & W(q,t) - \sum_{s \in Q} W(s,t) \\
& \text{(induction and (1))} & \leq\ & d = d'
\end{aligned}
$$

□

4.3 Gyroscope Insertion

Let us now take a closer look at the internal gyroscope refinement via the implicit place q in N, which introduces internal communication to avoid the self-trigger in C_r.

If we apply the implicit place insertion of q in N as proposed in Section 4.2, then also after the gyroscope refinement of q, there are structural conflicts between all (local) transitions of $q^\bullet$. These structural conflicts imply that we have to change the feasible

partition for the second reduction pass. Since the delay component produces the signal of t_d, it now has to produce all the signals of the transitions in $q^\bullet$, also those that were produced by other so-called *auxiliary components* in the first pass. Thus, the modified delay component STG could be very complex such that a circuit can perhaps not be synthesised anymore. This phenomenon will be visualised with the help of Figures 8c and 12:

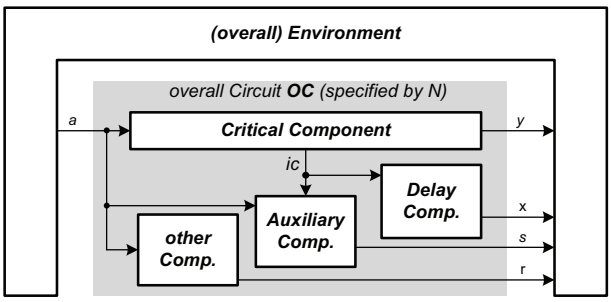

Figure 12: Circuit architecture of the resulting decomposed circuit specified by Figure 8b.

Assume the unmodified delay component only produces output signal x, another component produces signal s, and we have identified delay transition t_d for the self-trigger between t_{en} and t_{ex}. An internal gyroscope with a new signal ic is inserted via the implicit place q. Observe that $q^\bullet$ does not only contain t_d, but also t_1; thus t_d and t_1 are in structural conflict because of q – and after q's refinement the conflict is caused by the gyroscope place p_{out} (see also Figure 4). Using feasible partitions only, the component for x now has to produce s as well; we get a new component combining the delay component C_d and the component generating signal s. This can lead to large delay component STGs from which no circuit can be synthesised (because of their complexity).

Thus, for the second reduction pass, we make use of a quasi-feasible partition as studied in Subsection 2.4. After an output gyroscope insertion introducing ic via implict place q, we keep the original feasible partition, only adding ic as an output for C_r and as an input for all components producing a signal of a transition in the post-set of q; these components include C_d, the others are called auxiliary components; cf. algorithm Avoid in Figure 13 that generalises Avoid-0 from [WWSV09] since it deals with several auxiliary components, in addition to the delay component.

This algorithm does no necessarily produce a correct decomposition, but a failure can be recognised:

Proposition 4.3
The algorithm Avoid *is correct in the following sense: if $(C_i)_{i\in I}$ was obtained from N by decomposition (and hence correct w.r.t. N), then either the partition used by* Avoid *is quasi-feasible and $((C_i)_{i\in I'}, C'_r, C'_d, (C'_j)_{i\in J})$ with $I' = I \setminus (\{r, d\} \cup J)$ is correct w.r.t. N', which in turn is correct w.r.t. N when hiding $\{ic\}$, or there is some dynamic self-trigger with the two transitions of ic in C'_d.*

Proof. When decomposing N, a feasible partition was used (or at least a quasi-feasible one, if already some internal signal has been introduced). If the partition used in Avoid is not quasi-feasible, then there are two output transitions t_1 and t_2 in N' that are in conflict under some reachable marking and have signals produced by different components.

Input
specification N — critical component C_r with an irr. CSC conflict t_{en}, t_{ex} — delay component C_d with t_d 'between' t_{en} and t_{ex} in N, $l(t_d) \in Out_d^{\pm}$ — other components $(C_i)_{i \in I \setminus \{r,d\}}$

Output
modified specification N' — modified critical component C'_r — modified delay component C'_d — modified auxiliary components $(C'_j)_{j \in J},\ J \subset I$

---

1 in N, if insertImplicitPlace() returns true, then perform an output gyroscope insertion for a fresh output signal ic via the implicit place q and intermediate N''; in N'', $l({}^\bullet q) \subseteq Sig_r^{\pm}$ and $l(q^\bullet) \subseteq Out_d^{\pm} \cup \bigcup_{j \in J} Out_j^{\pm}$ where $J \subset I$ and C_j is called auxiliary component; this gives N'
2 C'_r is obtained from reduction of N' with partition member $(In_r, Out_r \cup \{ic\})$
3 C'_d is obtained from reduction of N' with partition member $(In_d \cup \{ic\}, Out_d)$
4 **for all** $j \in J$ **do** C'_j is obtained from reduction of N' with partition member $(In_j \cup \{ic\}, Out_j)$

Figure 13: Algorithm AVOID for inserting internal communication to avoid an irreducible CSC conflict – insertImplicitPlace() is specified in INSERTIMPLICIT in Figure 9.

Since this conflict is obviously new, t_1 and t_2 must have been in the postset of q in the intermediate N''. If they are in conflict under some reachable marking M there, this conflict must concern q, i.e. without q they would be concurrently enabled. With the implicit q, they can still fire one after the other in any order. Since q has no loops, this means that they are concurrently enabled under M, so in fact there is no conflict in N''.

If q is safe, there can be no conflict in N' as well: t_1 and t_2 are only enabled when p_{out} is marked, and then it is marked with one token as q was. Furthermore, the transitions for the new signal ic are not in conflict with any other transition.

If q is not safe, some reachable marking of N' puts at least two tokens on p_{in} resulting in a dynamic self-trigger for the latter transitions. This self-triger is still present in C'_d.

The correctness now follows from Theorem 2.7 as in the proof of Theorem 4.1 in [WWSV09]. In addition to one delay component C_d, we have several auxiliary components $C_{a_1}, ..., C_{a_n}$. The partition in AVOID is quasi-feasible, since

- the output transitions g_1 and g_2 are only in structural conflict with each other and they are labelled with the same signal anyway.
- g_1 and g_2 are only triggered by transitions with labels in $Sig_r = Sig'_r \setminus \{ic\}$ or by each other; they only trigger transitions with labels in $Out_d + \bigcup_{j \in J} Out_{a_j} = Out'_d + \bigcup_{j \in J} Out'_{a_j}$ and each other; $ic \in In'_d$, $ic \in In'_{a_j}$.

Consider a component C_k with $k \in I'$ and the corresponding initial component N'_k derived from N'. In N'_k, ic is lambdarised, since ic only triggers outputs of C_d and $C_{a_1}, ..., C_{a_n}$. As a first step, we can apply subnet contraction to the gyroscope yielding STG N''_k. The resulting place is implicit by Definition 2.4 and can be deleted yielding the original initial component N_k. Hence, N_k can be reduced to the final component C_k with the same sequence of reduction operations originally applied. □

In general, the components have many self-triggers or irreducible CSC conflicts resp., so AVOID has to be applied a number of times.

Often, k components ($k > 1$) yield the same self-trigger, i.e. characterised by the same entry/exit transition pair in N. For each component, a separate internal signal is needed, since it has to be produced by the respective component. Still, it is not necessary to apply the algorithm for finding an appropriate delay transition (cf. Section 4.1) and INSERTIMPLICIT k times. After the first run of these algorithms, an implicit place q with its sets ${}^\bullet q$ and $q^\bullet$ is known, and we can insert k concurrent gyroscopes via k copies $q_1, ..., q_k$ of q, i.e. with ${}^\bullet q_i = {}^\bullet q$ and $q_i{}^\bullet = q^\bullet, \forall i \in \{1...k\}$, see Figure 14a.

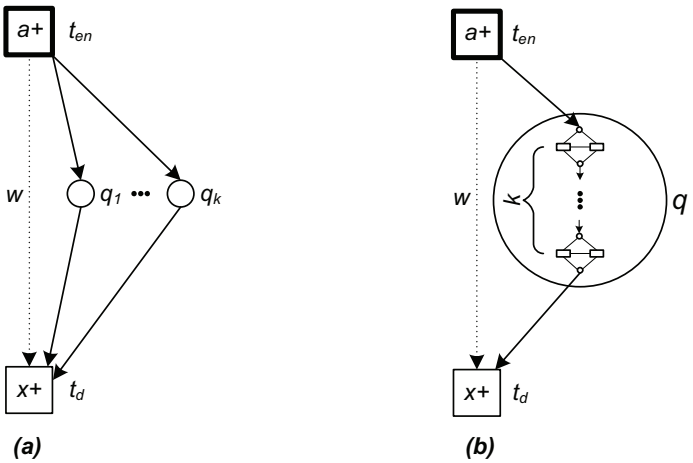

Figure 14: The same conflict occurs in k different components: (a) concurrent gyroscope insertion, (b) sequential gyroscope insertion.

Observe that this approach massively increases the concurrency degree of the new delay component (and also of the auxiliary components, not shown here), since after q_i's refinement – with a new gyroscope labelled with ic_i – each transition labelled with ic_i is a trigger of t_d; so each ic_i will be a relevant signal for the new delay component, i.e. $ic_i \in In_d, \forall i \in \{1...k\}$. This can be avoided by introducing only one place q and refine it via a sequence of k gyroscopes, as shown in Figure 14b. This is correct w.r.t. the concurrent insertion in Figure 14a, since each of the k internal gyroscopes must fire to activate t_d, but the order of firing does not matter, because each signal ic_i is internal (i.e its edge occurrence has no direct influence on an input occurrence from the environment). Hence, we can arbitrarily model a sequence of gyroscopes, as in Figure 14b; consequently, the new delay component gets only one new relevant signal instead of k signals. Of course, the chosen sequence of the k gyroscopes might have effects on the final circuit's performance, area and/or its synthesis time; this will be investigated in future work.

5 Results

We have integrated the algorithms INSERTIMPLICIT, CHECKTRANSITION and AVOID in our decomposition tool DESIJ [SWW09]. For identifying a delay candidate as well as the paths w_1 and w_2, so far a simple solution is implemented. We are currently working on integrating the ideas of Section 4.1.1. Right now, we can only avoid self-triggers, but

ideas to avoid more general conflicts are proposed in Section 3 (Figure 7). We applied decomposition-based logic synthesis via DesiJ and Petrify as synthesis backend to the benchmark examples in Table 1. Observe that the method presented in [WWSV09] cannot synthesise an SI circuit for any of the benchmark examples except for SPT-6 and SPT-7. We will now compare these results with the ones from Moebius, Mpsat and the pure application of Petrify. Note that Moebius implements an entire decomposition-based logic synthesis approach by

1. encoding of specifications that do not satisfy CSC by solving complex ILP problems,
2. decomposing them (i.e. computing the projection[7] of the encoded specification for each output signal, again by applying ILP), and
3. synthesising each component STG by applying state-based logic synthesis.

Initially, none of the benchmark examples satisfies CSC, so we cannot compare our results to a decomposition-based synthesis with Nutas.

Benchmark	Size $\lvert P\rvert/\lvert T\rvert$–$\lvert In\rvert/\lvert Out\rvert$	DesiJ *Dec+Int+Syn*	Moebius *Enc+Proj+Syn*	Mpsat	Petrify
2pp_wk.03	23/14–0/7	0	0	0	0
2pp_wk.06	47/26–0/13	0+0+1	1	0	2
2pp_wk.09	71/38–0/19	0+0+1	1+1+0	2	13
2pp_wk.12	95/50–0/25	0+0+1	1+2+1	7	149
tsend-csm	34/29–5/4	0+0+8	0+0+1	1	2
Shifter	67/51–12/13	0+0+9	4+1+0	dummy	1480
ArbTree	83/72–18/18	0+1+4	75+7+1	34	1117
SPT-6	403/260–65/65	7+14+788	2160+360+240	m.o.	m.o.
SPT-7	659/516–129/129	11+50+822	**9.5h**+2280+0	m.o.	m.o.

Table 1: Comparison of different logic synthesis approaches w.r.t. synthesis time (in seconds).

The benchmark computations, except for the Moebius results, were performed on a VMware® Linux guest system with 1 GB of RAM. The host machine was an Intel® Core™ 2 Duo E8400 CPU with 3 GHz and 4 GB of RAM running a Windows XP® 32-Bit version. The Moebius results were performed by Josep Carmona on an Intel® Xeon® CPU X3363 with 2.83 GHz and 24 GB of RAM running a Linux OS.

The first two columns of Table 1 report the benchmark names and their corresponding sizes in terms of place count ($|P|$), transition count ($|T|$), number of input signals ($|In|$) and output signals ($|Out|$). The remaining columns report the synthesis times for each benchmark in seconds. Note that in the DesiJ-column this time is split into decomposition time (*Dec*), the time for inserting internal communication signals as described in this paper (*Int*) and the cumulated time for Petrify synthesis of the component STGs (*Syn*). In the Moebius-column the times are also split into the encoding time to solve CSC for the specification N (*Enc*), the time to compute the projection of N for each output signal (*Proj*), and the Petrify-like logic synthesis for the projected components

[7] an alternative but restricted method of STG decomposition

(*Syn*). Observe that the encoding time for SPT-7 is 9.5 hours, all the other times are reported in seconds.

All benchmarks in the lower half of Table 1 arise when resynthesising handshake circuits consisting of control components [FC08], while the 2pp_wk.?? specifications are not related to resynthesis. All benchmarks up to 100 places or transitions can be handled within a few seconds by DesiJ. For instance, every other tool needs at least 34 seconds to synthesise the ArbTree circuit. Observe that Mpsat cannot deal with dummy transitions in the specification; in some cases, we have removed all such transitions via contraction. For the Shifter specification, this was not possible (though decomposition with DesiJ succeeded); after 37 seconds and five signal insertions, Mpsat cancelled the computation issuing the warning 'dummy cut-offs' (the 37 seconds could be taken as a lower bound for a potential circuit synthesis with Mpsat). Petrify and Mpsat cannot synthesise a circuit for SPT-6 and SPT-7 due to memory overflow (m.o.); for Moebius, the synthesis time is an order of magnitude higher than the DesiJ-based logic synthesis.

Note that DesiJ-based synthesis is not always the best approach to exploit the advantages of logic synthesis. In particular, for a small specification like tsend-csm an Mpsat or Petrify synthesis is better suited. DesiJ often introduces more internal signals than a pure logic synthesis would do. E.g. for the 2pp_wk.?? benchmarks, each of Petrify, Mpsat or Moebius only inserts one internal signal to solve CSC. With total decomposition (i.e. constructing one component for each output), DesiJ must avoid two self-triggers by introducing two internal signals, and during logic synthesis of the components Petrify introduces 5 CSC signals – hence, the DesiJ-based synthesis needs 7 internal signals to solve CSC for each 2pp_wk.?? benchmark.

As a consequence, we propose to apply DesiJ only for very large specifications where logic synthesis with other tools is impossible or takes unacceptably long time. Furthermore, total decomposition is not always the best approach, since it leads to many component STGs which potentially require many internal signal insertions to avoid irreducible CSC conflicts. In future work, we will try to find output partitions that result in few irreducible CSC conflicts, but still give component STGs that can be synthesised with Petrify, Mpsat or Moebius. This should maximally exploit the opportunities of logic synthesis, and lead to the most efficient circuit implementations that can be achieved.

6 Conclusion and Outlook

We improved the ideas presented in [WWSV09] such that we are now able to apply decomposition-based SI logic synthesis to many more specifications than before. This can enable SI logic synthesis for very complex STGs, in particular resulting from control resynthesis of handshake circuits.

Instead of solving the CSC encoding problem for a complex specification, we downscale the encoding problem to smaller component STGs. This makes it necessary to avoid irreducible CSC conflicts that can result from decomposition, and for this we introduce internal communication between the components.

In future work, we would like to optimise the presented approach by investigating how the choice of the delay candidate according to Section 4.1 can influence the synthesis time and the resulting circuit area and performance. Further research is needed to investigate how the order of the gyroscopes in a sequential gyroscope insertion as in Section 4.3

influences the resulting circuit performance and maybe its synthesis time.

As already mentioned, the application of total decomposition leads to many components, and many of them could have irreducible CSC conflicts; hence, many new internal signals have to be inserted in order to avoid these conflicts. In future work, we would also like to investigate how we can avoid the appearance of irreducible conflicts by controlling the decomposition process via suitable output partitions; i.e. we try to find output partitions that lead to a limited number of – not very complex – component STGs having few irreducible conflicts only. This could substantially reduce the number of internal signals to be inserted.

Acknowledgments

We would like to thank Josep Carmona for providing the MOEBIUS benchmarks and tool support for deriving benchmark STGs modelling the interface behaviour of control clusters during control resynthesis.

References

[And83] Charles André. Structural transformations giving b-equivalent pt-nets. In *Selected Papers from the 3rd European Workshop on Applications and Theory of Petri Nets*, pages 14–28, London, UK, 1983. Springer-Verlag.

[Ber87] G. Berthelot. Transformations and decompositions of nets. In *Advances in Petri Nets 1986-Part I: Central Models and Their Properties*, pages 359–376, London, UK, 1987. Springer.

[CCCGV06] Josep Carmona, José Manuel Colom, Jordi Cortadella, and F. García-Vallés. Synthesis of asynchronous controllers using integer linear programming. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 25(9):1637–1651, 2006.

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

[CKK+02] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. *Logic synthesis of asynchronous controllers and interfaces.* Springer-Verlag, ISBN: 3-540-43152-7, 2002.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2nd revised edition edition, 2001.

[EB02] Doug Edwards and Andrew Bardsley. Balsa: An asynchronous hardware synthesis language. *The Computer Journal*, 45(1):12–18, 2002.

[Ebe92] Jo C. Ebergen. Arbiters: an exercise in specifying and decomposing asynchronously communicating components. *Sci. Comput. Program.*, 18(3):223–245, 1992.

[FC08] Francisco Fernández and Josep Carmona. Logic synthesis of handshake components using structural clustering techniques. In *PATMOS 2008: Proceedings of the 18th International Workshop on Power and Timing Modeling, Optimization and Simulation, Lisbon, Portugal*, volume 5349 of *LNCS*, pages 188–198, 2008.

[KKY04] Victor Khomenko, Maciej Koutny, and Alex Yakovlev. Logic synthesis for asynchronous circuits based on Petri net unfoldings and incremental SAT. In *Proceedings of Fourth International Conference on Application of Concurrency to System Design (ACSD'04), June 16 - 18, 2004, Hamilton, Ontario, Canada*, pages 16–25. IEEE, June 2004.

[KMY06] Victor Khomenko, Agnes Madalinski, and Alex Yakovlev. Resolution of encoding conflicts by signal insertion and concurrency reduction based on STG unfoldings. In *ACSD '06: Proceedings of the Sixth International Conference on Application of Concurrency to System Design*, pages 57–68, Washington, DC, USA, 2006. IEEE.

[SV07] M. Schaefer and W. Vogler. Component refinement and CSC solving for STG decomposition. *Theoretical Computer Science*, 388(1–3):243–266, 2007.

[SVWK06] Mark Schaefer, Walter Vogler, Ralf Wollowski, and Victor Khomenko. Strategies for optimised STG decomposition. In *ACSD '06: Proceedings of the Sixth International Conference on Application of Concurrency to System Design*, pages 123–132, Washington (D.C.), USA, 2006. IEEE.

[SWW09] Mark Schaefer, Dominic Wist, and Ralf Wollowski. Desij – enabling decomposition-based synthesis of complex asynchronous controllers. *Application of Concurrency to System Design, IEEE International Conference on*, 0:186–190, 2009.

[vBJN99] C. H. Kees van Berkel, Mark B. Josephs, and Steven M. Nowick. Scanning the technology: Applications of asynchronous circuits. In *Proceedings of the IEEE*, volume 87, pages 223–233, 1999.

[VK06] W. Vogler and B. Kangsah. Improved decomposition of signal transition graphs. *Fundamenta Informaticae*, 76:161–197, 2006.

[VW02] W. Vogler and R. Wollowski. Decomposition in asynchronous circuit design. In *Concurrency and Hardware Design, LNCS 2549*, pages 152–190. Springer, 2002.

[WB00] R. Wollowski and J. Beister. Comprehensive causal specification of asynchronous controller and arbiter behaviour. In A. Yakovlev, L. Gomes, and L. Lavagno, editors, *Hardware Design and Petri Nets*, pages 3–32. Kluwer Academic Publishers, 2000.

[Wen77] S. Wendt. Using Petri nets in the design process for interacting asynchronous sequential circuits. In *Proc. IFAC-Symp. on Discrete Systems, Vol.2*, Dresden, pages 130–138, 1977.

[WWSV09] D. Wist, R. Wollowski, M. Schäfer, and W. Vogler. Avoiding irreducible CSC conflicts by internal communication. *Fundamenta Informaticae*, 95(1):1–29, 2009.

[YM07] Tomohiro Yoneda and Chris J. Myers. Synthesis of timed circuits based on decomposition. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 26(7):1177–1195, 2007.

Aktuelle Technische Berichte des Hasso-Plattner-Instituts

Band	ISBN	Titel	Autoren / Redaktion
30	978-3-86956-009-0	**Action Patterns in Business Process Models**	Sergey Smirnov, Matthias Weidlich, Jan Mendling, Mathias Weske
29	978-3-940793-91-1	**Correct Dynamic Service-Oriented Architectures: Modeling and Compositional Verification with Dynamic Collaborations**	Basil Becker, Holger Giese, Stefan Neumann
28	978-3-940793-84-3	**Efficient Model Synchronization of Large-Scale Models**	Holger Giese, Stephan Hildebrandt
27	978-3-940793-81-2	**Proceedings of the 3rd Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering**	Hrsg. von den Professoren des HPI
26	978-3-940793-65-2	**The Triconnected Abstraction of Process Models**	Artem Polyvyanyy, Sergey Smirnov, Mathias Weske
25	978-3-940793-46-1	**Space and Time Scalability of Duplicate Detection in Graph Data**	Melanie Herschel, Felix Naumann
24	978-3-940793-45-4	**Erster Deutscher IPv6 Gipfel**	Christoph Meinel, Harald Sack, Justus Bross
23	978-3-940793-42-3	**Proceedings of the 2nd. Ph.D. retreat of the HPI Research School on Service-oriented Systems Engineering**	Hrsg. von den Professoren des HPI
22	978-3-940793-29-4	**Reducing the Complexity of Large EPCs**	Artem Polyvyanyy, Sergy Smirnov, Mathias Weske
21	978-3-940793-17-1	**"Proceedings of the 2nd International Workshop on e-learning and Virtual and Remote Laboratories"**	Bernhard Rabe, Andreas Rasche
20	978-3-940793-02-7	**STG Decomposition: Avoiding Irreducible CSC Conflicts by Internal Communication**	Dominic Wist, Ralf Wollowski
19	978-3-939469-95-7	**A quantitative evaluation of the enhanced Topic-based Vector Space Model**	Artem Polyvyanyy, Dominik Kuropka
18	978-3-939469-58-2	**Proceedings of the Fall 2006 Workshop of the HPI Research School on Service-Oriented Systems Engineering**	Benjamin Hagedorn, Michael Schöbel, Matthias Uflacker, Flavius Copaciu, Nikola Milanovic
17	3-939469-52-1 / 978-3-939469-52-0	**Visualizing Movement Dynamics in Virtual Urban Environments**	Marc Nienhaus, Bruce Gooch, Jürgen Döllner
16	3-939469-35-1 / 978-3-939469-35-3	**Fundamentals of Service-Oriented Engineering**	Andreas Polze, Stefan Hüttenrauch, Uwe Kylau, Martin Grund, Tobias Queck, Anna Ploskonos, Torben Schreiter, Martin Breest, Sören Haubrock, Paul Bouché
15	3-939469-34-3 / 978-3-939469-34-6	**Concepts and Technology of SAP Web Application Server and Service Oriented Architecture Products** (noch nicht erschienen)	Bernhard Gröne, Peter Tabeling, Konrad Hübner