STG Decomposition: Avoiding Irreducible CSC Conflicts by Internal Communication

Dominic Wist, Ralf Wollowski

Technische Berichte Nr. 20

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

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

Nr. 20

STG Decomposition: Avoiding Irreducible CSC Conflicts by Internal Communication

Dominic Wist, Ralf Wollowski

Potsdam 2007

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

Die Reihe *Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam* erscheint aperiodisch.

Herausgeber: Professoren des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam

Redaktion: Dominic Wist, Ralf Wollowski

E-mail: {dominic.wist; ralf.wollowski}@.hpi.uni-potsdam.de

Vertrieb: Universitätsverlag Potsdam
Am Neuen Palais 10
14469 Potsdam
Fon +49 (0) 331 977 4517
Fax +49 (0) 331 977 4625
e-mail: ubpub@uni-potsdamde
http://info.ub.uni-potsdam.de/verlag.htm

Druck: allprintmedia gmbH
Blomberger Weg 6a
13437 Berlin
email: info@allprint-media.de

Heft Nr 20 (2007)
ISBN 978-3-940793-02-7
ISSN 1613-5652

STG Decomposition: Avoiding Irreducible CSC Conflicts by Internal Communication

Dominic Wist, Ralf Wollowski
dominic.wist@hpi.uni-potsdam.de, ralf.wollowski@hpi.uni-potsdam.de

For the logic synthesis of complex asynchronous controllers a decomposition based technique has been proposed. It is based on a structural STG decomposition method [21, 25, 26] and the speed independent logic synthesis [9]. But the decomposition may lead to component STGs with irreducible CSC conflicts (i.e. components that cannot be implemented as speed independent circuits). An idea is presented to avoid such irreducible CSC conflicts in component STGs by the introduction of internal communication among the corresponding component circuits. Furthermore, first concepts for a systematization are presented, based on the structural theory of Petri nets. The suitability of the approach is demonstrated by means of two examples (a FIFO controller and a sequencer/parallelizer tree).

1 Introduction

Nowadays, the semiconductor industry gives serious consideration to the adoption of asynchronous circuit technology. During the last decade, first entirely asynchronous ICs appeared on the market [11, 24]. In comparison to their synchronous counterparts the opportunities of asynchronous systems are high performance, low power consumption, better electromagnetic compatibility (EMC), modularity and the lack of clock skew. In spite of their proved benefits, the asynchronous design style is not fully accepted amongst designers of synchronous circuitry. The major drawback is the insufficient maturity level of existing CAD tools for their synthesis and verification.

For many years several research groups developed methods and tools for asynchronous system design. So far the most successful initiatives have been those guaranteeing a robust implementation of complex specifications at the expense of their efficiency. These methods are based on syntax-directed translation from some high level language such as TANGRAM [23] or BALSA [2].

The syntax-directed translation paradigm is characterized by the translation of each statement of the high-level design-entry language into a so-called *handshake component (HC)*. Thus the entire entry language program is translated into a netlist of communicating handshake components. Each HC is a handmade circuit, i.e. designed and optimized by hand without the aid of a CAD tool. As this paradigm supports the designers with a programming language design entry it fits well to the established design

flow for synchronous systems.[1] A further advantage of this paradigm is that it does not suffer from the state explosion problem, since the analysis of the whole state space of a highly concurrent system is not necessary. But syntax-directed translation can lead to inefficient implementations since logic synthesis is not applied (i.e. the power of boolean minimization techniques).

Thus a desirable aim is to overcome these drawbacks by the incorporation of logic synthesis into a design flow based on syntax-directed translation – and raise the competitiveness of the syntax-directed translation approach to achieve results comparable to those obtained by the current design flows in synchronous design. First promising results of this approach are presented in [5]. On this account we consider an EDA framework as shown Figure 1 (as proposed in a similar way e.g. in [6, 28]) that combines logic synthesis (for the entire control path) and syntax-directed translation (for the data path).

Due to the separate design of control and data path, we divide handshake components into plain data path, plain control and mixed components, as proposed in [1]. Then the control extraction can be realized by the transformation of each control component and parts of each mixed component (i.e. the control behavior part, see [1]) into signal transition graphs (STGs) – the so-called *HC-STGs*. The remaining part of each mixed component together with the plain data path components form the entire data path of the asynchronous system.

In order to exploit the power of a global logic synthesis for the entire control path, all handshake component STGs are composed into one complex controller STG N (by parallel composition, see Section 2.3). A significant reduction in the complexity of N can be reached by the contraction of transitions that only model internal HC communication among the handshake components. Anyway, this reduced N is usually still complex; on this account it is often impossible to generate its whole state space which is needed for the application of logic synthesis (state explosion problem). In order to overcome the state explosion problem a structural STG decomposition as introduced in [5, 26, 29] has been proposed. After this decomposition the component STGs are subject to one of the well-known logic synthesis approaches such as the SI synthesis method [9] or the synthesis based on (extended) burst mode machines [19, 30, 31].

The proposed design framework still has certain gaps (i.e. components that are not investigated so far, represented as rectangles with dashed borders in Figure 1). The design of these components is subject of future work.

As a contribution to complete the proposed EDA framework the aim of the presented work is to combine the decomposition approach of Vogler et al. [21, 25, 26] with the SI logic synthesis approach [9]. In fact, there are other STG decomposition approaches [5,29], but – in contrast to the decomposition method used here – these approaches are only applicable for STGs having the complete state coding (CSC) property (i.e. a given STG has to be transformed into an STG satisfying CSC as a precondition). However, the decomposition methods of [5, 29] guarantee that the resulting components also satisfy CSC and thus a speed independent logic synthesis is directly possible. But the

[1] Recently, C-like languages such as SPECC or SYSTEMC attract the attention of synchronous circuit designers, since they completely abstract from the (HW-)implementation. Hence, it is a promising future topic to investigate the translation of SPECC or SYSTEMC descriptions into asynchronous systems.

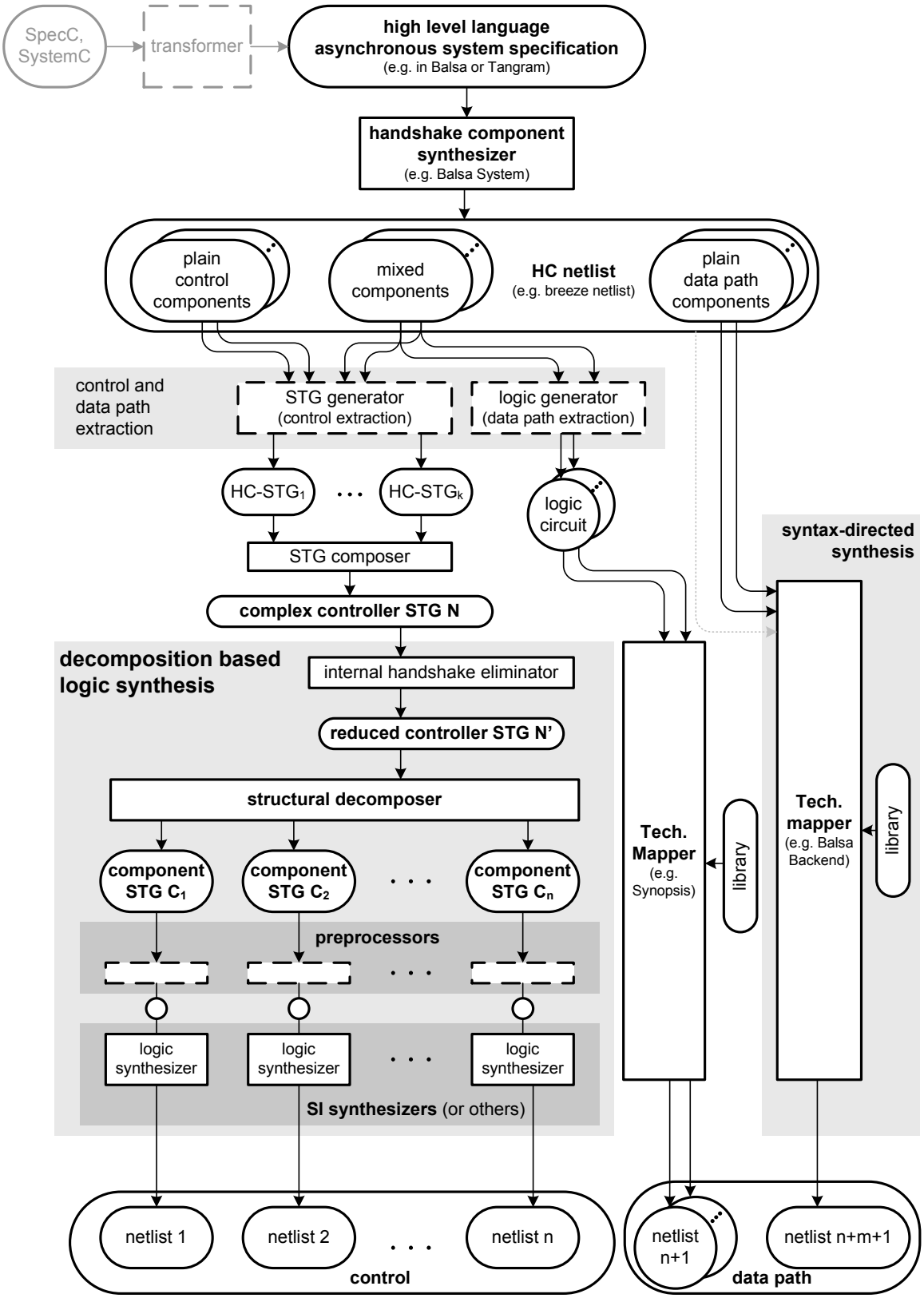


Figure 1: Proposed EDA framework for asynchronous systems (components with dashed borders are not investigated so far)

methods [5, 29] only deal with total decompositions, i.e. they generate a component for each single output signal, while the chosen decomposition method of Vogler et al. is able to handle components with more than one output signal – which can lead to better results (e.g. by means of the size of the component STGs given by the cumulated number of their reachable markings [21]). Further, for the decomposition approach in [29] a formal definition as well as a correctness proof is missing so far.

However, the decomposition according to Vogler et al. of an SI implementable STG N may lead to component STGs C_i with irreducible CSC conflicts even though the overall STG has none (i.e. C_i cannot be implemented as a speed independent circuit). This work tackles this problem and presents an idea and first concepts to avoid irreducible CSC conflicts in component STGs resulting from an STG decomposition according to Vogler et al. by the suitable introduction of internal communication among the component circuits. In this context, it provides a basis for the implementation of the preprocessors in Figure 1.

A further contribution of this work is to break down the problem of solving CSC conflicts for large and complex STGs (which is rather inefficient so far) to the solution of CSC conflicts in several smaller and less complex component STGs (which can be handled quite efficiently).

The organization of the paper is as follows: In the next Section some basic definitions are introduced. In Section 3, the motivation and the idea for avoiding irreducible CSC conflicts in component STGs by the introduction of internal communication is presented. First concepts for a systematization, i.e. considerations towards a structural method for the implementation of such an internal communication are given in Section 4. Section 5 demonstrates the suitability of the proposed method by means of two examples: a FIFO controller and a handshake circuit derived from a BALSA specification (a sequencer/parallelizer tree). Finally, we give our conclusions and – since this work introduces first concepts only – an outlook for future work in Section 6.

2 Basic Definitions

This section only presents the necessary definitions in order to support the presented work without detailed explanations, but we provide all needed references for background reading. We mainly follow the definitions section in [16].

2.1 Petri nets

A *Petri net* is a 4-tuple $N = (P, T, W, M_N)$ where P is a set of *places* and T a set of *transitions*. Both sets are finite and $P \cap T = \emptyset$. $W : P \times T \cup T \times P \rightarrow \mathbb{N}_0$ is the *weight function* and M_N the *initial marking*. A marking of a Petri net is an assignment of a non-negative integer k to each place. If k is assigned to a place $p \in P$ by a marking M, denoted $M(p) = k$, we will say that p is marked with k tokens. A Petri net can be considered as a bipartite graph with weighted and directed arcs between places and transitions.

Given a place or transition $x \in P \cup T$, its postset and preset are denoted by ${}^\bullet x$ and $x^\bullet$ resp. A transition $t \in T$ is *enabled* under a marking M, denoted by $M[t\rangle$, if each place p in ${}^\bullet t$ is marked with $M(p) \geq W(p,t)$. When a transition t is enabled, it can *fire*, yielding a new marking M' with $M'(p) = M(p) - W(p,t) + W(t,p)$ for all $p \in P$. We denote this by $M[t\rangle M'$.

A *transition sequence* $v = t_0 t_1 ... t_n$ is enabled under a marking M (yielding M') if $M[t_0\rangle M_0[t_1\rangle M_1 ... M_{n-1}[t_n\rangle M_n = M'$, denoted $M[v\rangle$ or $M[v\rangle M'$ resp. If $M_N[v\rangle$ holds then v is called a *firing sequence*. By the way, 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. We only consider Petri nets N such that the set $[M_N\rangle$ of reachable markings is finite i.e. N is *bounded*. A Petri net is *k-bounded* or simply *bounded* if for every reachable marking M and every place $p \in P$, $M(p) \leq k$. A Petri net is *safe* if it is 1-bounded.

A Petri net N is called *live* iff in any reachable marking M for all transitions $t \in T$ a transition sequence v exists with $M[v\rangle M'[t\rangle$.

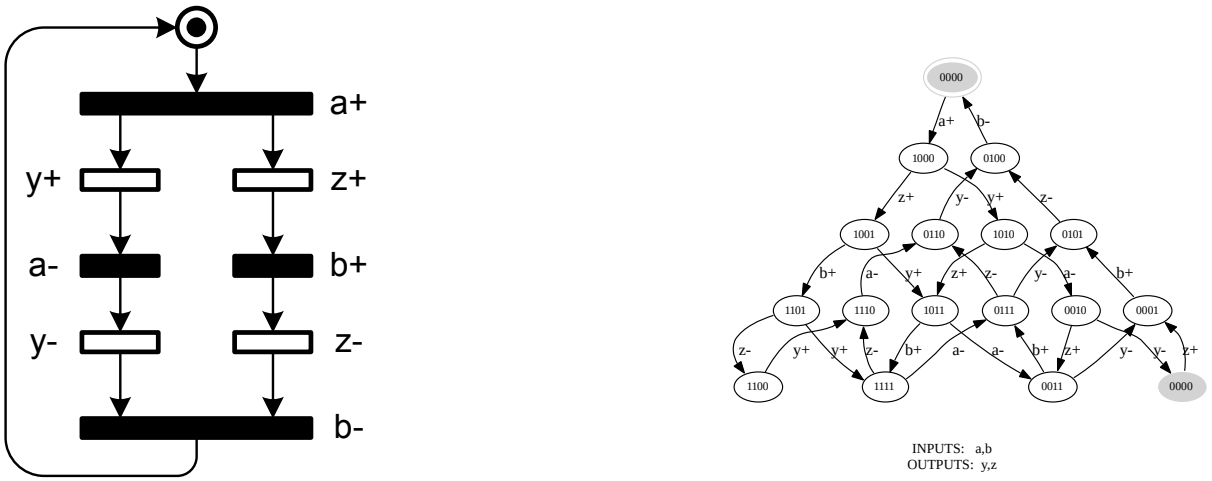
A Petri net N with a initial marking M_N is said to be *reversible* if for each reachable marking $M \in [M_N\rangle$, M_N is reachable from M; thus in a reversible net one can always get back to the initial marking.

2.2 Signal Transition Graphs and Complete State Coding (CSC)

A Signal Transition Graph (*STG*) is a tuple $N = (P, T, W, M_N, In, Out, Int, l)$ where (P, T, W, M_N) is a Petri net and In, Out and Int are disjoint sets of *input*, *output* and *internal* signals. We define the set of all signals $Sig := In \cup Out \cup Int$, the set of *local signals* $Loc := Out \cup Int$ and the set of all *external signals* $Ext := In \cup Out$. $l : T \to Sig \times \{+,-\} \cup \{\lambda\}$ is the *labeling* function. $Sig \times \{+,-\}$ or (for short) $Sig^\pm$ is the set of *signal edges* or *signal transitions*; its elements are denoted as s^+, s^- resp. (short for $(s,+)$, $(s,-)$ resp.). While the plus sign denotes a rising, the minus sign denotes a falling signal edge. We write $s^\pm$ if it is not important or unknown which signal edge takes place; if such a term appears more than once in the same context, it always denotes the same direction. For a short notation, input (output) signal edges are just called input (output) edges. An STG may initially contain transitions labeled with the empty word λ called *dummy*-transitions. They are only a design simplification and do not correspond to a signal transition (i.e. describe no physical reality).

An example of an STG is shown in Figure 2a. Places are drawn as circles containing the number of tokens according to their initial marking. Unmarked places with only one transition in their preset and postset resp. can be omitted if the corresponding arcs are weighted with 1 (short-hand notation), i.e. these places are implicitly given by an arc between these two transitions. Transitions are drawn as boxes together with their labeling and the weight function as directed arcs xy labeled with $W(x,y)$ if $W(x,y) > 1$.

We lift the notion of enabledness to transition labels: $M[l(t)\rangle\rangle M'$ if $M[t\rangle M'$. This is extended to sequences as usual – deleting λ labels automatically (e.g. $M[s^\pm\rangle\rangle M'$ means that a sequence of transitions fires, where one of them is labeled $s^\pm$ while the others are λ-labeled). A sequence v of elements of $Sig^\pm$ is called a *trace of a marking*



(a) STG N (b) State graph

Figure 2: STG example with its corresponding state graph

M if $M[v\rangle\rangle$, and a *trace* of STG N if $M_N[v\rangle\rangle$. The *language* $L(N)$ of N is the set of all traces of N.

An STG is called *consistent* if for each signal s the edges s^+ and s^- alternate in all traces, always beginning with the same signal edge. Only from consistent STGs a circuit can be synthesized.

An STG has *auto-concurrency* if there are distinct transitions t_1 and t_2 with $l(t_1) = l(t_2) \neq \lambda$ such that for some reachable marking M $\forall p \in P : M(p) \geq W(p,t_1) + W(p,t_2)$.

An STG has a *dynamic conflict* if there are distinct 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 an *auto-conflict* if $l(t_1) = l(t_2) \neq \lambda$.

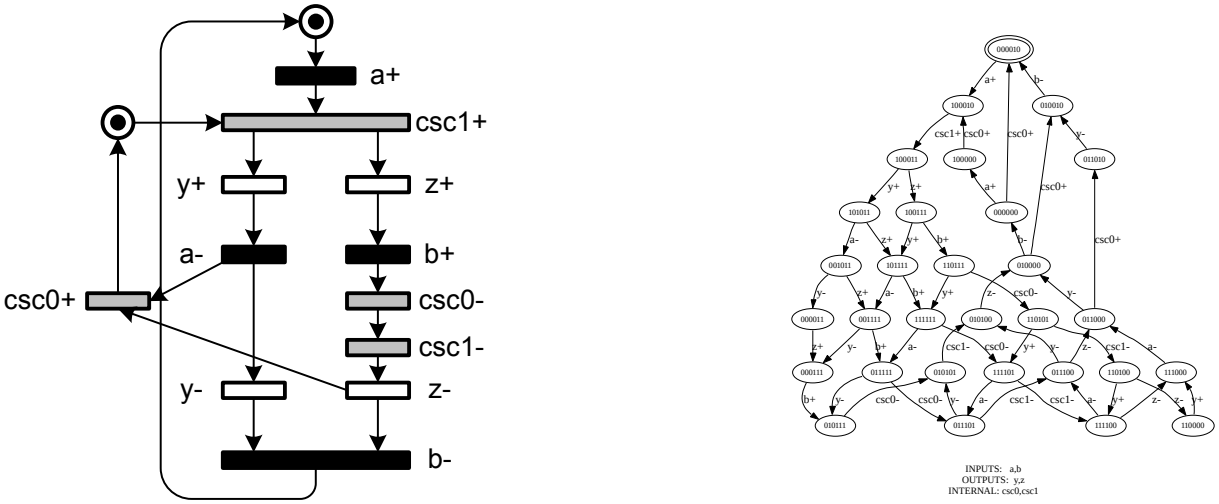
The *reachability graph* RG_N of an STG N is an edge-labeled directed graph on the reachable markings with M_N as root; there is an edge from M to M' labeled $l(t)$ whenever $M[t\rangle M'$. RG_N can be seen as a finite automaton where all states are accepting, and $L(N)$ is the language of this automaton. N is *deterministic* if its reachability graph is deterministic, i.e. if it contains no λ transitions and if for each reachable marking M and each signal edge $s^\pm$ there is at most one M' with $M[s^\pm\rangle\rangle M'$.

Let RG_N be the reachability graph of an STG N. A *state vector* is a function $sv : Sig \rightarrow \{0,1\}$ where '0' means logical low and '1' logical high. A *state assignment* assigns the corresponding state vector to each marking M of RG_N denoted by sv_M.

In Figure 2b the corresponding reachability graph of the STG in 2a is shown, annotated with its state assignment. This graph is usually called the *state graph* (and each of its nodes a *state*). The node highlighted by the double line border represents the root node (i.e. the initial marking).

A consistent STG N has *Complete State Coding (CSC)* if no two reachable markings M_1 and M_2 with the same state vector exist that enable the same output signals. Otherwise, N has a *CSC conflict* between M_1 and M_2. The STG in Figure 2a or its state graph in Figure 2b resp. has such a CSC conflict between the gray marked states

annotated with the same state vector $(a, b, y, z) = (0, 0, 0, 0)$ since either marking enables the output signal edge z^+ while the other marking (M_N) enables no output signal. If N violates CSC, no asynchronous SI circuit can be synthesized directly. However, it is possible to solve CSC automatically by the insertion of *internal signals* (i.e. outputs which are ignored by the environment, without changing the external behavior of the STG).



(a) STG N' having CSC

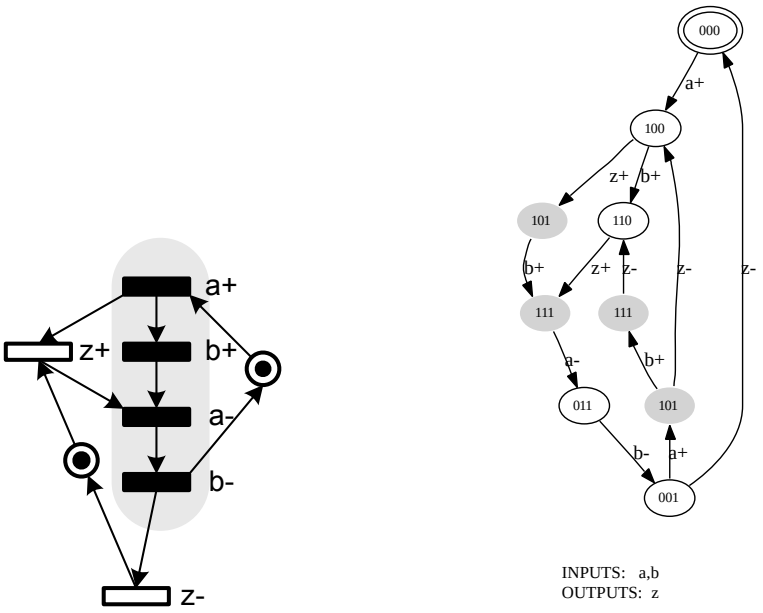
(b) State graph having CSC

Figure 3: CSC Solution for the STG and state graph resp. in Figure 2

There exist tools (such as PETRIFY [8]) which can modify an STG automatically in such a way that CSC is satisfied. For example PETRIFY extends the STG N in Figure 2a with internal transitions labeled with internal signal edges $csc0^{\pm}$ and $csc1^{\pm}$ leading to an STG N' satisfying the CSC property (see Figure 3a). Figure 3b shows the corresponding state graph whereas each state has a unified state vector now. However, if a speed independent (SI) implementation is required, it is not always guaranteed to get a CSC solution. In [20] such SI preserving internal event insertions in RG_N (in order to solve CSC) are called *input proper event insertions*. Thus an internal transition t_i may not be inserted in STG N as a *syntactical trigger* of an input transition; i.e. if t_i is connected with another transition t_x via a place p and two arcs (t_i, p) and (p, t_x) weighted with $W(t_i, p) = k \wedge W(p, t_x) = j \wedge j, k \in \mathbb{N}$ then t_x must not be an input transition.

If a CSC conflict in N cannot be solved by an input proper event insertion then it is called an *irreducible CSC conflict*. Figure 4 shows an STG N and its state graph with two irreducible CSC conflicts. The first one is between M_1 and M_2 (with $sv_{M_1} = sv_{M_2} = 101$) which are characterized by splitting up the trace $a^+z^+b^+a^-b^-a^+$ as follows: $M_N[a^+z^+\rangle\rangle M_1[b^+a^-b^-a^+\rangle\rangle M_2$. This conflict cannot be solved by an input proper event insertion in the trace $b^+a^-b^-a^+$, since a and b are input signals – so it specifies an irreducible CSC conflict. Note that the second CSC conflict between M_3 and M_4 with $sv_{M_3} = sv_{M_4} = 111$ is also irreducible.

An STG N is called *SI implementable* iff it is consistent, bounded, output-persistent and has no irreducible CSC conflicts. Intuitively, an STG is *output-persistent* if for each



(a) STG (b) State Graph

Figure 4: Example STG having irreducible CSC conflicts

output signal holds that its transitions are not disabled by transitions of another signal and input signals cannot be disabled by outputs. Output persistency therefore only allows input events to be in direct conflict (thus modeling non-deterministic choice in the environment). More detailed information about SI implementability is given e.g. in [9].

This report proposes a new idea to avoid irreducible CSC conflicts in component STGs resulting from STG decomposition which is briefly introduced in the next section.

2.3 STG Decomposition

To cope with the state explosion problem during logic synthesis of a complex STG its decomposition has been proposed. As outlined in Figure 5, the decomposition used in this work [25, 26] is based on the partition of the output signals in such a way that for each partition block one component STG will be extracted from the overall STG. This is realized by the application of three reduction operations: transition contraction, deletion of redundant transitions, and deletion of redundant places. Since the component STGs should be orders of magnitude smaller than the overall STG, logic synthesis of each STG component does not suffer from the state explosion problem anymore and leads to component circuits whose interconnection implements a modular circuit with the required overall behavior.

In the remainder of this section the reduction operations will be defined. Furthermore, a definition of parallel composition will be given, since on one hand it is the base for the correctness definition of our decomposition approach and on the other hand it specifies the behavior of the STG composer component in Figure 1. To get more information of this decomposition method, the reader is invited to take a deeper look in the

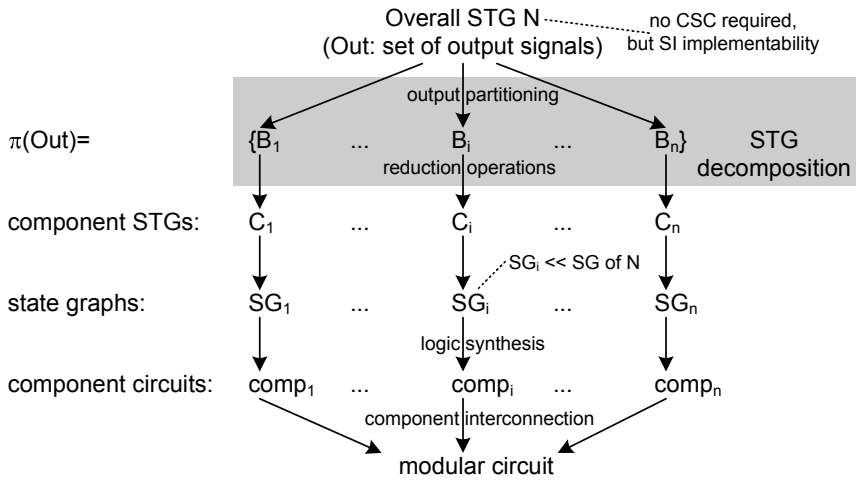
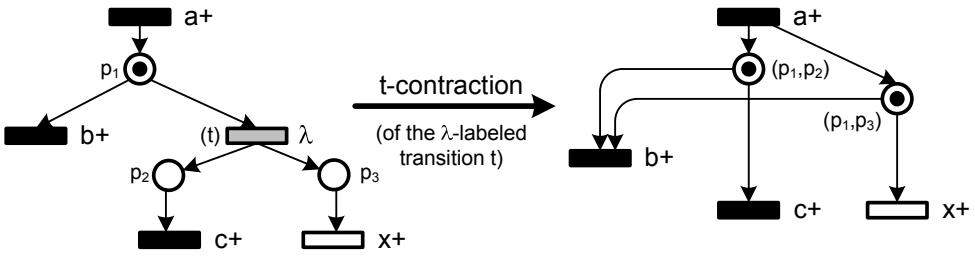


Figure 5: Design flow for an STG decomposition based synthesis (according to [21, 25, 26])

corresponding literature [21, 25, 26].

We now introduce *transition contraction* (t-contraction for short, see e.g. [4] for an early reference), which is the most important reduction operation in our decomposition procedure. In an STG N, a transition t can be contracted, yielding the STG $\bar{N}$, if ${}^\bullet t$ and $t^\bullet$ are disjoint and t is not adjacent to an arc with weight greater than 1. The contraction removes the transition t and replaces ${}^\bullet t$ and $t^\bullet$ by their Cartesian product (cf. Figure 6a, b for an example). Each new place (p, p') inherits the tokens and the connections to other transitions from p and p'.

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



(a) before contraction (b) after contraction

Figure 6: Example of a transition contraction

The remaining reduction operations in our decomposition algorithm are the deletion of redundant transitions and of redundant places. A *redundant transition* is either a λ-transition t, where each place $p \in {}^\bullet t \cup t^\bullet$ forms a loop with t with two arcs of the same weight (t is a *loop-only transition*) or some other transition has arcs to and from the same places with the same weight and has the same label as t (which is a *duplicate transition*). *(Structurally) redundant places* are defined e.g. in [4]. Such places can be deleted in a Petri net without changing the firing sequences.

These reduction operations and type-1 contractions preserve the behavior in a strong sense:

Proposition 2.1
If N' is obtained from an STG N by deleting a redundant transition or place or by a type-1 contraction then N and N' are bisimilar.

In the following definition of *parallel composition* $\|$, we will have to consider the distinction between input and output signals. The idea of parallel composition is that the composed systems run in parallel and synchronize on common signals – corresponding to circuits that are connected on signals with the same name. 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 one component can be an input of one or several others, and in any case it is an output of the composition. A composition can also be ill-defined due to what e.g. Ebergen [10] calls *computation interference*; this is a semantic problem, and we will not discuss it here.

The parallel composition of STGs N_1 and N_2 is defined if $Out_1 \cap Out_2 = \emptyset$. 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 $s^{\pm}$-labeled transition of N_2 will occur together with some $s^{\pm}$-labeled 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 $s^{\pm}$-labeled transition of N_1 with each $s^{\pm}$-labeled transition from N_2 if $s \in A$.

Since composition is associative and commutative up to isomorphism, 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. Since the place set of the composition is the disjoint union of the place sets of the components, we can consider markings of the composition (regarded as multisets) as the disjoint union of markings of the components and write a marking of the composition $(C_i)_{i \in I}$ as a tuple $(M_1, \ldots, M_n)$ if M_i is a marking of C_i for $i \in I = \{1, ..., n\}$. The formal definition and an example can be found e.g. in [26].

3 Achieving SI Implementability by Internal Communication

This section presents our idea to avoid irreducible CSC conflicts in component STGs – arising through STG decomposition – by the introduction of internal communication.

3.1 Motivation

In order to improve the efficiency (with respect to speed and area) of complex asynchronous circuits it has been proposed to generate their control paths by decomposition

based logic synthesis (cf. Section 1). To cope with the state explosion problem of complex STGs a structural STG decomposition was suggested, firstly by [7, 17], further enhanced by [5, 29] and [21, 25, 26].

However, the decomposition according to Vogler et al. [21, 25, 26] of an overall STG N which has no irreducible CSC conflicts can lead to component STGs having them; consequently, these component STGs are not SI implementable. The next figure illustrates this problem. Figure 7a shows a circuit in its overall environment and STG

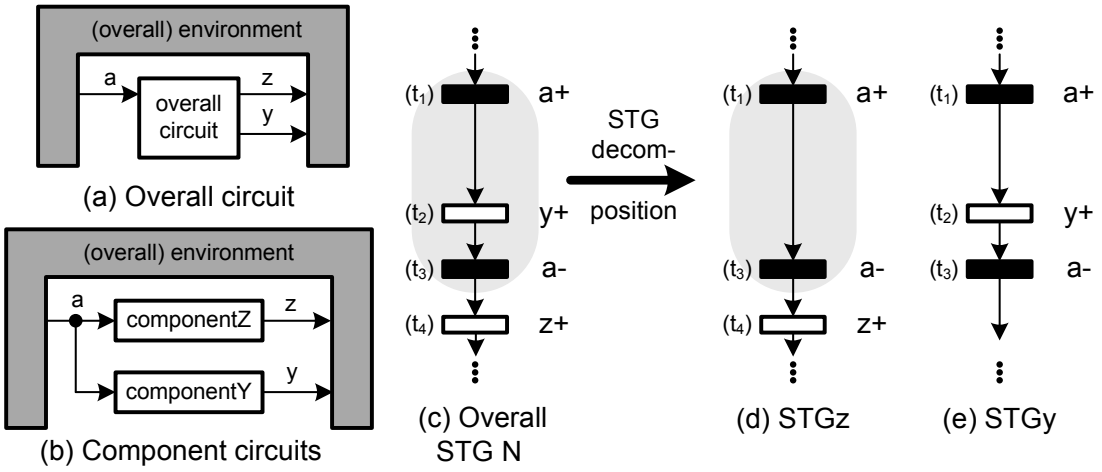


Figure 7: STG decomposition can lead to undesired irreducible CSC conflicts

N in Figure 7c models the circuit's behavior partially. Observe that (the shown part of) N is SI implementable, i.e. in particular it has no irreducible CSC conflicts. A total decomposition of N – i.e. each component is responsible to generate exactly one output signal (see Figure 7b) – leads to the component STGs in Figure 7d and e. In fact, STGy is SI implementable, but STGz has an CSC conflict (marked by the gray oval), since STGz enables a trace $M_1[a^+a^-\rangle\rangle M_2$ with $sv_{M_1} = sv_{M_2}$, but only M_2 enables an output transition (t_4, labeled with the output edge z^+). Moreover, this CSC conflict is irreducible since the only possibility to solve this conflict is to insert internal signal transitions 'between' a^+ and a^- – but this is not allowed because this would not be an input proper event insertion.

So the specified behavior of STGz is not SI implementable at all; thus the related circuit component(componentZ) is called *critical circuit component* and STGz itself *critical STG component*.

3.2 Idea

The goal now is to avoid such irreducible CSC conflicts arising through STG decomposition without changing the interface behavior (of the overall circuit) and without introducing any timing requirements for the environment (as e.g. in the fundamental operating mode [22]). Hence, it is an improper modification of STGz to insert t_5 (labeled with ic^+) as an *output* transition in the transition sequence t_1t_3, as indicated in Figure 8a. In fact, the state vectors of the considered markings which are in CSC conflict are different now (and of course this would solve the CSC conflict), but the insertion

of t_5 also demands that the environment must not generate a^- until the output edge ic^+ occurs which is an improper change of the interface behavior due to SI logic synthesis.

Alternatively – as already mentioned in the previous section – the insertion of ic as an *internal* signal is an input improper event insertion, i.e. is also an improper change of the interface behavior. This is because the environment cannot observe, and thus cannot register ic^+, i.e. the relation $ic^+ \rightarrow a^-$ is not a 'real' causal dependence (in terms of: the environment observes signal ic, waits for the generation of ic^+, and this enables the environment to generate a^-). Consequently, t_5 models a *desired temporal* relation, i.e. although a^- is not caused by ic^+, the edge a^- should always be generated *after* ic^+; if this temporal order is guaranteed, a circuit operation without anomalous behavior (such as metastability) is possible. In this way componentZ demands a *timing requirement* from the environment by means of this temporal relation, since (with respect to the local scope of componentZ due to STGz) the environment generates a^- independently from a reaction of the circuit (i.e. a^- can be produced after a^+ without regarding the circuit's reaction). Observe that an SI circuit can take an uncertain amount of time τ to generate ic^+ (unbounded gate delay model). Hence, a circuit designer who only knows the specification STGz must take into account that the environment can produce a^- before the end of the time interval τ (i.e. while the circuit is 'in motion' – just generating the signal transition ic^+ – it can be interrupted by the new input edge a^-) and this could lead to anomalous behavior. In order to reach correct behavior, the environment ought to delay the generation of a^- at least until the end of the time interval τ, but as mentioned above, such timing requirements for the environment are not allowed in SI operating mode.

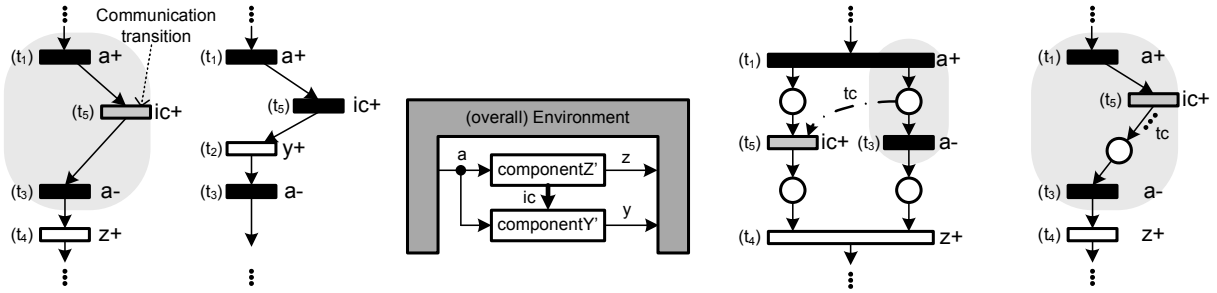


Figure 8: Solution of an irreducible CSC conflict by internal communication

So STGz_ic does not model the behavior precisely, in particular it does not show the problematic of such an internal signal insertion. In contrast, Figure 8d shows a precise model (due to the local scope of componentZ) according to [27]. The 'real' relation between ic^+ and a^- namely is a so-called *tcb-concurrency*: the environment (of componentZ) generates a^- concurrent to ic^+, but the occurrence of a^- prevents the occurrence of ic^+ (indeed, this can lead to anomalous behavior, but to model the possible malfunction explicitly makes no sense). Of course, the prevention of ic^+ by a^- is not desired, i.e. the circuit should generate ic^+ always before a^-, and this timing requirement is expressed by the label tc of the dot-dash line read arc. The tcb-concurrency is

usually approximated by a *pseudo-causality (of the tcb-type)* [27] as shown in Figure 8e. This pseudo-causality only models the desired, 'error-free' behavior. Observe that three dots next to the arc together with the label tc represent the required *temporal* sequence 'ic^+ before a^-' and not that a^- and ic^+ are *causally dependent* .

However, in the context of decomposition the introduction of t_5 in STGz as an internal transition (as shown in Figure 8a) *is allowed*, if there is another component – the so-called *delay component* – producing an output as a causal precondition for the generation of a^-. With respect to our example the delay component is componentY and the precondition for the generation of a^- is the occurrence of y^+. Since a circuit is responsible for the generation of its outputs, the designer is free to delay the generation of y^+ as long as componentZ' has completely registered a^+. This can be reached, if componentY is not allowed to produce y^+ until componentZ' generates ic^+. Hence, STGy must be extended by an ic^+ labeled input transition (t_5) which 'triggers' the output transition y^+. In Figure 8b this extended STGy is shown as STGy'. (Keep in mind that in STGy' the signal a becomes irrelevant now.)

In this regard, a one-sided internal communication from componentZ' to componentY' is introduced via signal ic in such a way that componentZ' indicates via ic^+ the occurrence and registration of a^+ to componentY'. The signal ic is called *(internal) communication signal* and t_5 (in STGz_ic) is called *internal communication transition (ICT)*.

Incidentally, an input edge (such as a^-) is called *critical edge*, if it may arrive too fast after another input edge (such as a^+) such that it forces malfunction of the circuit. The time distance between these both input edges – while anomalous behavior is possible – is called *critical distance*.

The arguments given above lead us to the conclusion that it is possible to insert an internal transition 'between' two transitions labeled with the same input signal in a component STG without violating the correct and speed-independent behavior of the resulting system of interacting components.

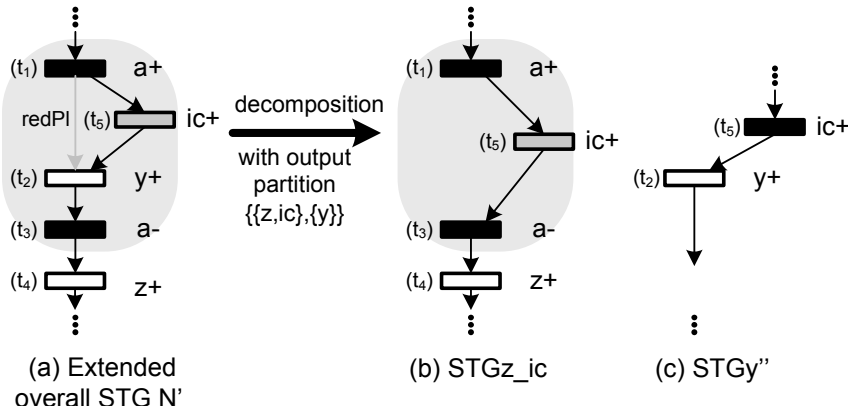


Figure 9: Solution of an irreducible CSC conflict by the insertion of ic^+ in the overall STG and its decomposition steered by a suitable output partition

Regarding a systematic method of the ICT insertion, it is possible to insert the ICT in the *overall* STG N (see N' in Figure 9a) and then to apply a decomposition of N' based

on a *suitable* partition of the output signals (which is not a total decomposition and ic is regarded as an output signal, see below) yielding the STG components STGz_ic and STGy" in Figure 9b and c. (Observe that STGy" could also be obtained by the contraction of the irrelevant a labeled transitions in STGy'.) The ICT insertion in N is possible, since only an output (y^+) is delayed in N' (i.e. it is an input proper event insertion and the feasibility of such a transition insertion has been discussed already in the context of the ICT insertion in STGy)[2]. Concerning the suitable decomposition, the output partition must contain blocks where the output(s) of a critical component C_c *and* the introduced internal signal(s) for the avoidance of the irreducible CSC conflict(s) in C_c are in one partition block. With respect to our example, a decomposition based on output partition $\{\{z, ic\}, \{y\}\}$ must be applied.

In fact, the latter suggestion requires two decomposition cycles, but the ICT insertion in the overall STG N ensures the correct insertion of the ICTs in the respective component STGs (critical component STG and delay component STG), due to the application of the decomposition method [20, 26] which is proved to be correct. In contrast, the ICT could also be inserted in the critical component STG as well as in the delay component STG directly. Indeed this avoids the second decomposition cycle, but it requires a more complex correctness proof for this kind of ICT insertion, since this insertion does not necessarily lead to the same component STGs which result from the decomposition of N' (cf. Figure 8b and 9c). Since the decomposition is not considered to be the bottleneck in logic synthesis so far (since it is a structural method) the introduction of the ICT in N is preferred.

Observe that in the context of the STG decomposition [26] the ICT formally has to be handled as an output transition. Consequently, this looks like a 'real' causal dependence $ic^+ \rightarrow a^-$ in STGz_ic which is actually a pseudo-causality of the tcb-type as discussed above (cf. Figure 8e). But due to the indirect delay of a^- (by the internal communication via ic) the desired temporal order (ic^+ before a^-) is always met and since the circuit synthesis is based on the reachability graph (which only specifies all the temporal orders of all signal edges), the decomposition approach can be applied without any changes.

Summing it up, irreducible CSC conflicts resulting from the STG decomposition [21, 25, 26] can be avoided by:

1. the introduction of internal communication transitions in N which are usually not needed for a 'conventional' CSC solution (but may serve as such signals, too),
2. and the subsequent decomposition of this extended STG N based on a suitable partition (which is not the finest possible partition according to a total decomposition).

The suggested solution is correct by means of its hardware implementation, since the generation of the input edge a^- is simply delayed by the internal communication signal (ic). This is an indirect delay, since the critical component notifies the delay component about the registration of a^+; then the delay component produces an output edge which allows the environment to generate the critical edge (a^-).

[2]It can be proved that such insertions are correct according to the correctness notion of [20].

In the next Section some considerations towards a systematic insertion of internal communication transitions in N are presented. Furthermore, the limitations of this method are investigated.

4 Towards a Structural Method

4.1 Consistent Introduction of Internal Communication Transitions

So far only parts of STGs were considered. However, if an internal communication transition (labeled with ic^+) has to be introduced in a consistent (overall) STG N the question arises how the complementary transition ic^- could be inserted without violating the consistency of N.

Figure 10a shows the whole STG N as a completion of the STG fragment in Figure 7c. Thus the same irreducible CSC conflict must be avoided in this example (see the gray highlighted part).

Let us assume, a transition t_7 labeled with ic^+ is inserted in N (as described in the previous section), then a second transition t_8 labeled with ic^- (the complementary transition) must be introduced in N in order to preserve its consistency. Furthermore, the introduction of t_8 must be an input proper event insertion, i.e. t_8 might not be inserted as a 'trigger' of an input transition in N (as discussed in Section 3.2).

As a first attempt we propose to insert the transition t_8 at the same (suitable) position in N where t_7 is introduced in such a way that the considered irreducible CSC conflict will be avoided by the alternating firing of t_7 and t_8. Initially, instead of a transition labeled with ic^+ a so-called *T-place* (p_T) is introduced at this position which then is refined into a *Toggle-net (T-net)* containing both transitions t_7 and t_8. This will be explained now by the help of Figure 10.

Since we only consider an extended version of the example from the previous section, the position for the insertion of the T-place is known, i.e. p_T will be introduced as an redundant place between t_1 and t_2 in N, see Figure 10b. (In Section 4.2 a systematic approach to find such a position in N will be introduced.) When p_T is introduced in N, the T-place will be refined by a T-net (see STG N^T in Figure 10c). In the meantime the place 'redPl' has turned to a redundant place and can be deleted. A detailed explanation of the refinement can be given by means of Figure 11: All ingoing arcs of p_T turn to ingoing arcs of p_o and all outgoing arcs of p_T turn to outgoing arcs of p_u. If the place 'redPl' was marked, then p_T as well as (after its refinement) p_u will also be a marked with the same amount of tokens, i.e. $M(redPl) = M(p_T) = M(p\_u)$. Furthermore, either place p_l or p_h must be marked (with one token); but, for the avoidance of the considered irreducible CSC conflict it is irrelevant which one is marked.

As mentioned above, the introduction of t_7 and t_8 in N^T by a T-net structure is an input proper event insertion, since the signal edges ic^+ and ic^- only delay an output edge (y^+). Furthermore, it is guaranteed (by the T-net structure) that ic^+ and ic^- always alternate in such a way that the consistency of N^T is preserved.

The decomposition of N^T according to the partition $\pi(Out) = \{\{z, ic\}, \{y\}\}$ leads

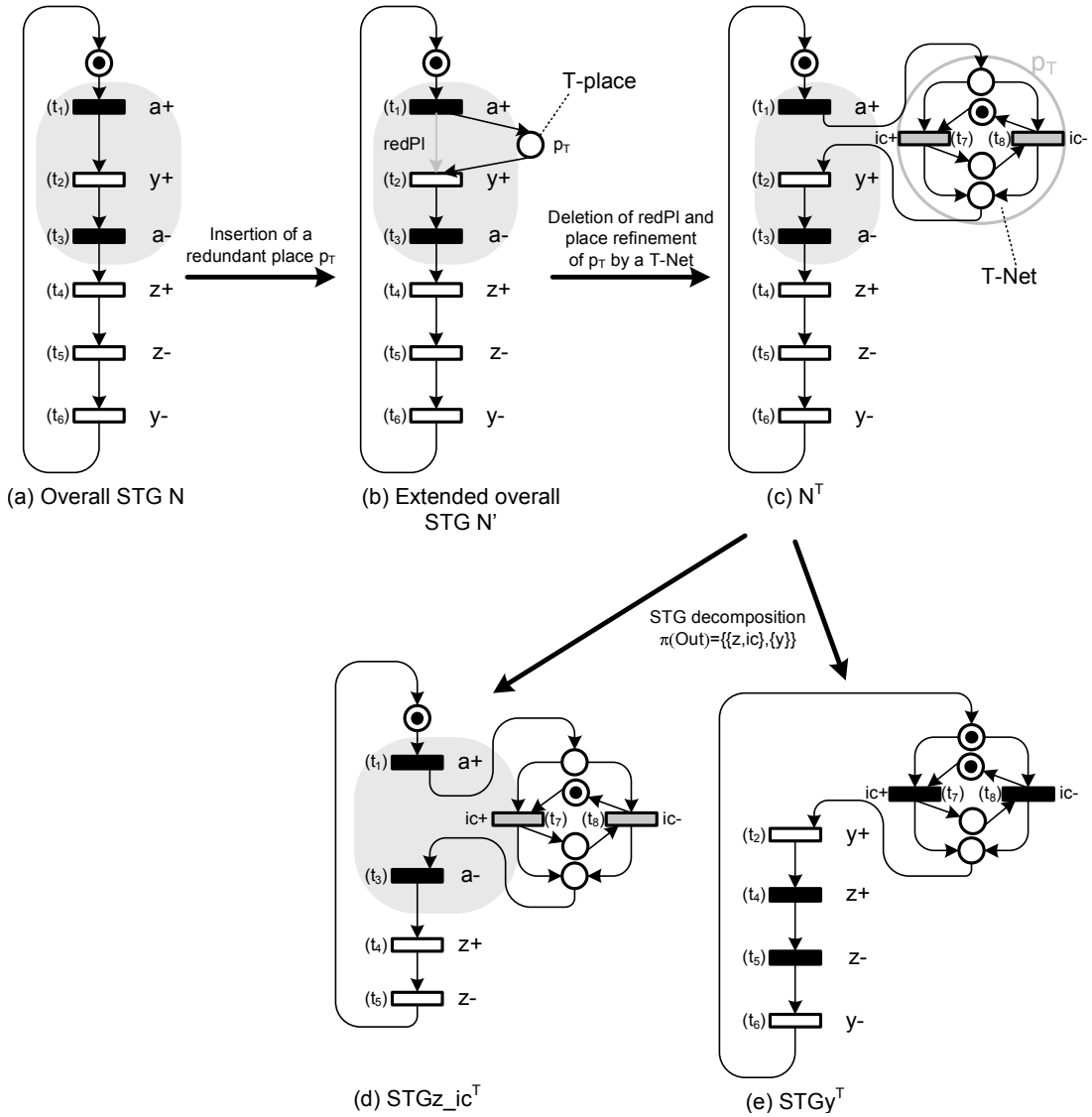


Figure 10: Consistent insertion of internal communication transitions ($ic^{\pm}$) using a T-net

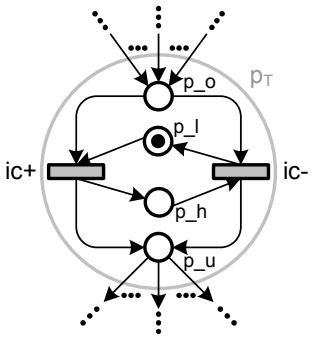


Figure 11: Place refinement by a T-net (modeling the internal signal transitions)

to the component STGs STGz_ic^T and STGy^T (see Figures 10d and e). Observe that STGz_ic^T has no irreducible CSC conflict (just like the component STG part in Figure 9b). The conflict is avoided by the introduction of the one-sided internal communication via signal ic. In addition to Figure 9b the complementary signal edge ic^- (represented by t_8) is introduced in a consistent and systematic manner by the T-net structure.

4.2 Suitable Introduction of a T-Place

In this section a structural method for the identification of a suitable position to insert the T-place p_T in the overall STG N is introduced, in such a way that after p_T's refinement and the application of a suitable decomposition of N the component STGs have no irreducible CSC conflicts anymore. Observe that N might not have any irreducible CSC conflict, but it is not needed that it satisfies CSC initially. Furthermore, N must be:

- live,
- reversible,
- and safe.

Then the introduction of p_T is based on three main steps following a standard decomposition cycle according to Vogler et al. [21, 25, 26]:

1. the identification of critical transition sequences in the critical STG components,
2. the extraction of critical transition pairs, and
3. the insertion of a T-place concerning a particular critical transition pair.

For each of these steps some ideas to solve the underlying problem are presented next. Note that our considerations so far focus only a special (but maybe the most important) type of an irreducible CSC conflict – so-called *input self-trigger* (which will be defined in 4.2.1) –, but some ideas for an extension regarding *general* irreducible CSC conflicts are proposed, too. Finally in Section 4.3 the limitations of the method are discussed and an approach in order to handle *all* component STGs resulting from a decomposition of N is outlined.

4.2.1 Identification of Critical Transition Sequences

At first, we define the term critical transition sequence:

Definition 4.1
A *critical transition sequence (CTS)* is an *input* transition sequence which belongs to a certain irreducible CSC conflict of a critical STG component C_c, i.e. the firing of the CTS starting from marking M_1 yields marking M_2, and between M_1 and M_2 is an irreducible CSC conflict. Considering only the label sequence of a CTS, the resulting trace is called a *critical trace*.

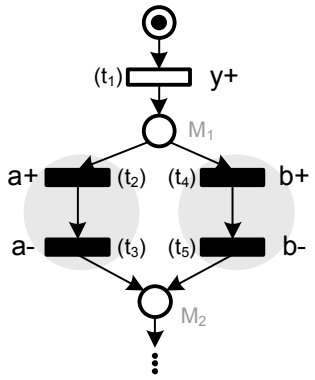


Figure 12: Two CTSs belonging to the same irreducible CSC conflict

Regarding the component STG in Figure 7d we find the critical transition sequence $M_1[t_1t_3\rangle M_2$. Even though there can be more than one CTS corresponding to a certain irreducible CSC conflict (e.g. Figure 12 shows an example with two CTSs t_2t_3 and t_4t_5 belonging to the same irreducible CSC conflict between M_1 and M_2) we are not aiming at a detailed investigation of this topic, but in further research we will do so. Here, we only consider irreducible CSC conflicts with one CTS between the conflicting markings (as it will be the case in our two examples in Section 5). Now we introduce a systematic way of finding this CTS for each CSC conflict in each component STG in order to identify all transitions of each CTS in the corresponding overall STG N. This knowledge is needed to insert T-places for each irreducible CSC conflict arising in some component STG in the corresponding overall STG N. In the remainder of Section 4.2 we will show how this may work.

First, let us recall the notion of the 'conventional' CSC solution (as implemented in standard tools such as PETRIFY or MPSAT [12, 13]). The 'conventional' CSC solution for an STG (or a component STG in this context) is based on the suitable introduction of internal signals in its reachability graph (RG). In Figures 13a and b the notion of this technique is outlined:

Consider the conflicting markings M_1 and M_2 (i.e. in particular $sv_{M_1} = sv_{M_2}$) of the RG which is assumed to belong to a certain STG component. This conflict can be solved by the insertion of an internal edge ic^+ in the trace w_1 from M_1 to M_2 which leads to the extended RG shown in Figure 13b. Since M_2 now has a different state vector than M_1, the considered CSC conflict is solved. Such an insertion of a signal edge ic^+ in w_1 is also called the *splitting of the trace* w_1 *by* ic^+. Note that the *position* of ic^+ in w_1 is arbitrary with respect to the solution of the considered CSC conflict. However, if the reachability graph should be the basis for a speed independent synthesis, then such an internal edge ic^+ must not be inserted as a trigger of an input edge (but an input proper event insertion is required) as discussed in Section 2.2.

Remember that an irreducible CSC conflict can be characterized by a critical trace w_1 with $M_1[w_1\rangle\rangle M_2$ consisting of inputs only. Consequently, due to the local view, an input proper event insertion in w_i is impossible. In this work, a solution of irreducible CSC conflicts in component STGs is proposed by a *suitable* splitting of critical traces with internal edges in such a way that the system of interacting component circuits

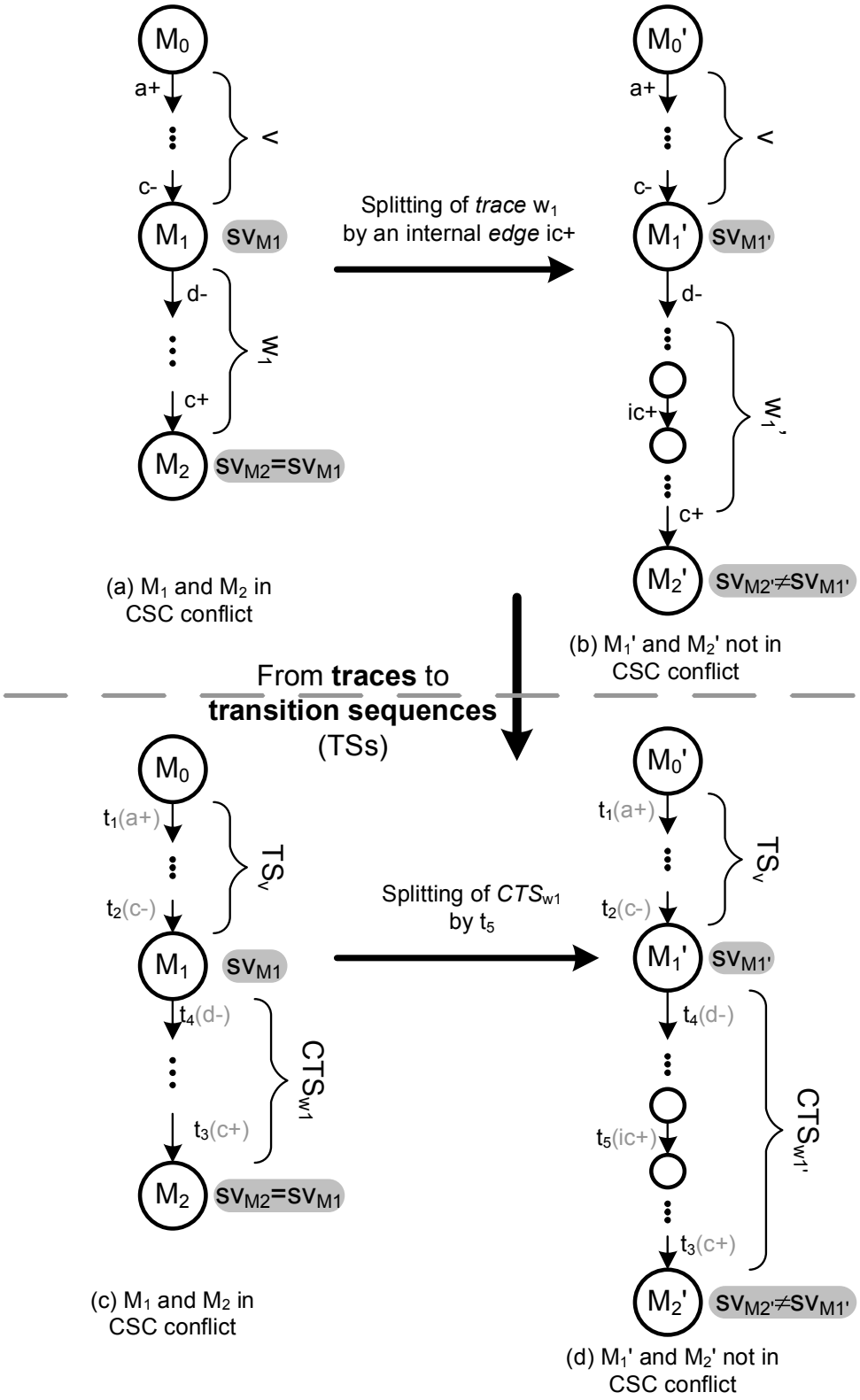


Figure 13: CSC solution based on the splitting of either traces or transition sequences

form the desired overall SI implementation. (In Section 3 this idea already has been introduced by means of an example.)

The splitting of the critical trace w_1 leading from M_1 to M_2 by the edge ic^+ can be reached by a suitable introduction of an internal communication transition in the *overall* STG N (see e.g. Figures 9a and 10c). Thus we lift the notion of the splitting of traces by some signal edge (ic^+) to the splitting of *transition sequences* by some transition (labeled with ic^+), see Figures 13c and d; i.e. the edge labels of the reachability graph are now replaced by the numbers of the corresponding firing transitions. In this example, the critical trace w_1 is mapped onto the critical transition sequence CTS_{w1}. The mapping 'critical traces $\rightarrow$ CTSs' is even unique, provided that we consider deterministic STGs.

The considered CSC conflict between M_1 and M_2 is now solved by splitting CTS_{w1} by the firing of t_5 which is labeled with ic^+ (see Figure 13d).

To avoid state explosion, t_5 might be inserted in the overall STG N directly, i.e. instead of constructing its reachability graph.

Provided that during decomposition the transition numbers are preserved, then all transitions of a critical transition sequence of a component STG C_i can be identified in N as well. Now the goal is to insert a T-place p_T in N 'between' two transitions of CTS_{w1} in such a way that after the refinement of p_T and a subsequent suitable decomposition of N the critical transition sequence CTS_{w1} in the component STG C_i is split up by the firing of the introduced internal communication transition. Such an introduction of a T-place is based on graph searching techniques (see Section 4.2.3), i.e. there must be a path in N between the two transitions which will be split up by the T-place. In consistent STGs such a path *always* exists between two transitions labeled with the same signal but complementary edges.

Definition 4.2
A *critical transition pair (CTP)* is an ordered pair of transitions (t_1, t_2) of a CTS (i.e. t_1 fires before t_2), and t_2 is the first successor of t_1 labeled with the complementary signal edge w.r.t. t_1. Transition t_1 is called *entry transition* and t_2 *exit transition*.

Often CTSs with only two elements occur; these will be called *input self-triggers*. Since we only consider consistent STGs, the two transitions of an input self-trigger are labeled with complementary edges (see e.g. Figure 12 with t_2t_3 and t_4t_5).

The identification of CTSs can be realized with the help of standard tools (such as PETRIFY or MPSAT). With these tools the critical trace belonging to an irreducible CSC conflict must be identified and then the critical trace can be mapped onto a critical transition sequence according to the scheme in Figure 13. For the special situation of input self-triggers even a necessary *structural* condition can be given which is more efficient than the PETRIFY approach for the identification of CTSs in component STGs because it does not require the construction of the reachability graph. Maybe in practical examples the structural test for input self-triggers in the components is often enough in order to obtain SI implementable component STGs, since more general CTSs do not occur in the examples we have checked so far (see also Section 5).

Proposition 4.3
Let us consider an input self-trigger t_1t_2 in STG N. Then $t_1 \in {}^\bullet({}^\bullet t_2)$ holds.

Consequently, if N is safe and consistent then t_1 and t_2 are *at least* connected by a place p with $W(t_1, p) = W(p, t_2) = 1$.

Here, the proposition will not be proved, but obviously the proposition cannot be a sufficient condition, since a structural 'connection' of t_1 and t_2 only does not necessarily force a transition sequence $...t_1t_2....$

Remember that an input self-trigger is characterized by two transitions t_1 and t_2 labeled with the same signal, but complementary edges. So Proposition 4.3 must be a necessary condition for input self-triggers, since t_1 must be a direct trigger of t_2, i.e. t_1 activates t_2 by its firing. Further, let us assume t_2 is already activated before t_1 has fired, then the STG N has auto-concurrency and this is not allowed.

4.2.2 Extraction of Critical Transition Pairs

As already explained, to avoid an irreducible CSC conflict in a component STG C_i a T-place p_T will be introduced in the overall STG N 'between' two transitions – a CTP – of a CTS of C_i. Then p_T will be refined into a T-net with two internal communication transitions whose firing will split the CTS, and thus avoiding the irreducible CSC conflict.

In order to find such critical transition pairs only ordered transition pairs (t_1, t_2) have to be extracted from a CTS where the firing of t_2 is causally dependent from the firing of t_1; in consistent STGs this holds in particular for *all* pairs (t_1, t_2) of a CTS where:

1. t_1 precedes t_2 regarding the order of the CTS, and

2. both transitions are labeled with complementary signal edges,

i.e. it holds for all critical transition pairs (see also Definition 4.2).

Regarding the example in Figure 14 the pairs (t_2, t_5), (t_3, t_6) and (t_4, t_7) are CTPs with respect to the critical transition sequence CTS_1 as well as CTS_2.

Observe that for each CTP there is a directed path starting from the entry transition to the exit transition in the overall STG N; this is important for the systematic introduction of a T-place in N based on graph traversals which is described in Section 4.2.3. By the way, the CTP condition does not hold for arbitrary extracted pairs from a CTS: Let us assume the overall STG N has the same structure with respect to the transitions t_3 and t_4 as the component STG in Figure 14a, then e.g. for the pair (t_3, t_4) (extracted from CTS_1) there is no path from t_3 to t_4 in N.

In general, from each CTS at least one CTP is extractable; for input self-triggers exactly one CTP can be extracted.

The structural splitting of a CTS (i.e. the splitting of a CTS of C_i by a transition insertion in the overall STG N) will be explained in the next section by the introduction of a T-net (or a T-place p_T resp.) with respect to a particular CTP.

4.2.3 Insertion of a T-Place concerning a Critical Transition Pair – a Case Study

In this section a systematic insertion of a T-place p_T in N will be considered in order to separate the entry transition from the exit transition by (the transitions of) p_T.

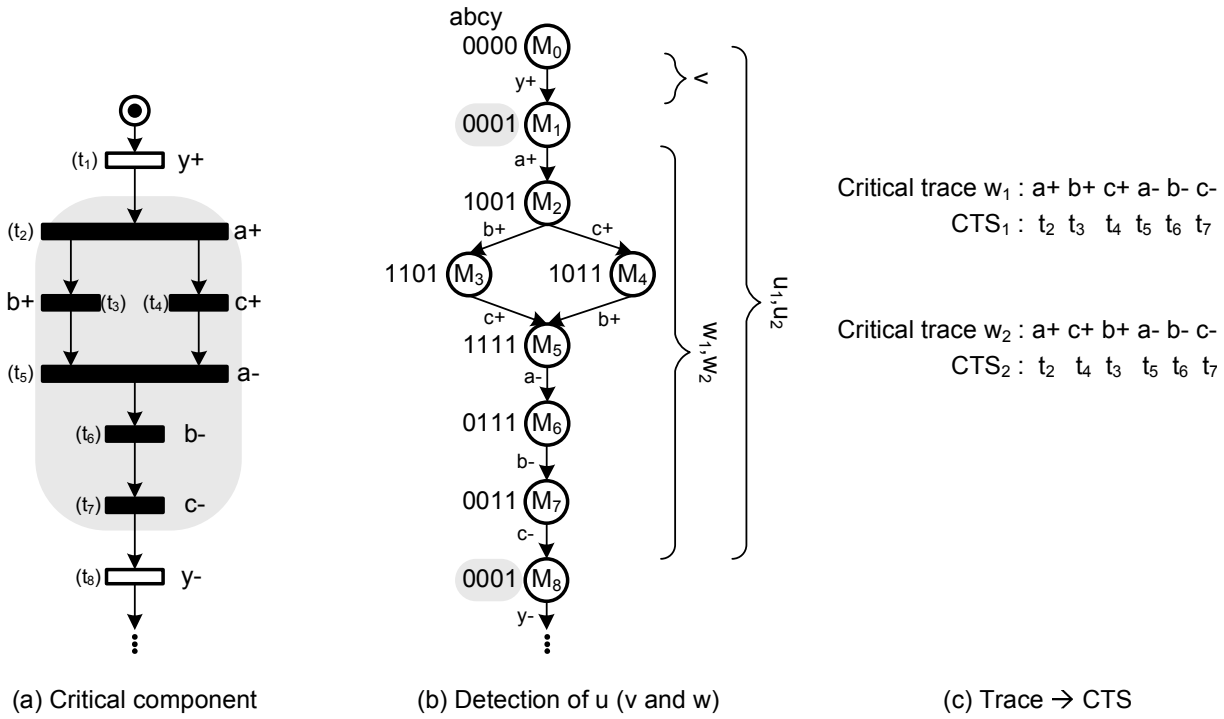


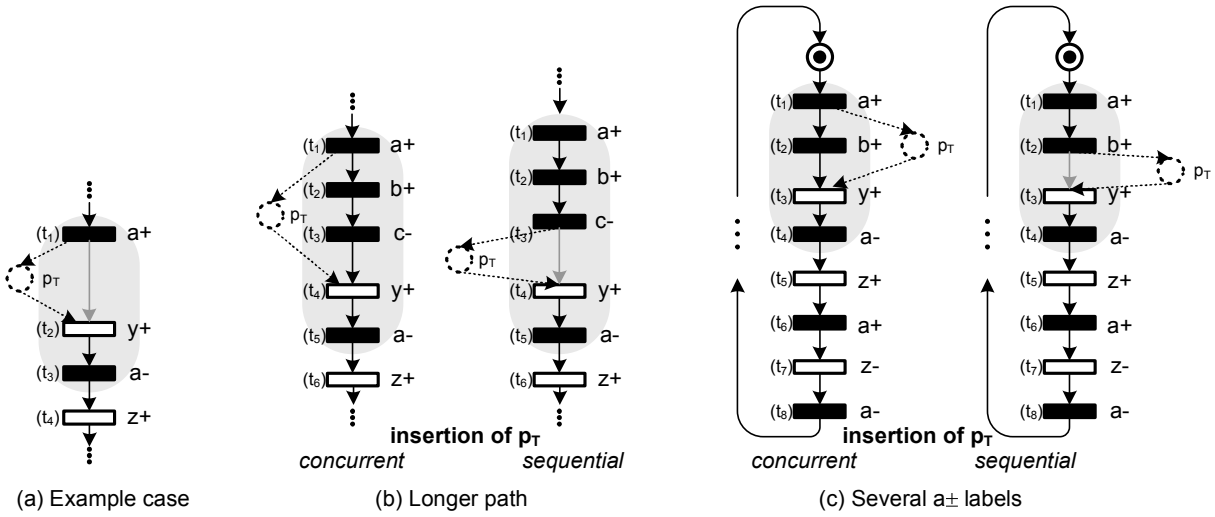
Figure 14: Critical transition sequences (without input self-triggers)

Here only CTSs with a single CTP will be considered, i.e. input self-triggers. The entry transition is always labeled with a^+ and the exit transition with a^-. Furthermore, all paths in N leading from the entry to the exit transition are highlighted by gray ovals. Observe that the restriction to input self-triggers is not just a hard restriction as it seems, because from a general CTS two or more CTPs can be extracted, but only for one of these CTPs the presented T-place insertion approach must work.

Now, the example of the previous section will be considered again (see Figure 15a). As one can see, the T-place p_T will be introduced between t_1 (entry transition) and t_2. Note that t_2 is an output transition, which is a trigger for the exit transition t_3.

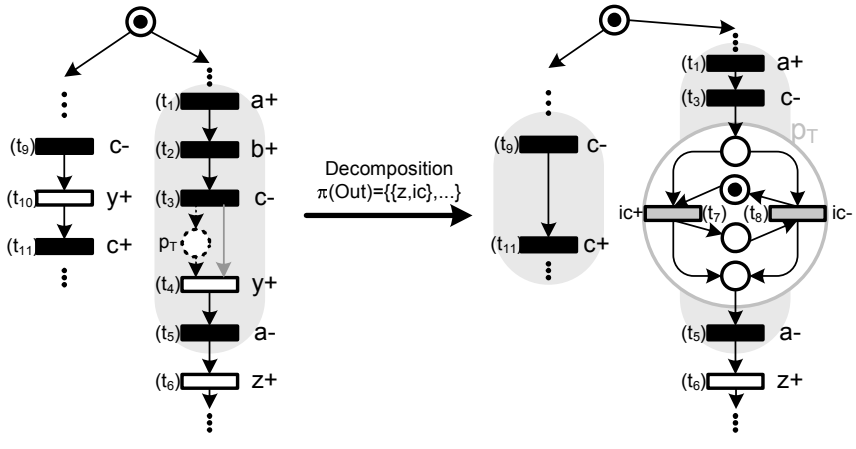
If there is a longer path from the entry to the exit transition, i.e. if it consists of more than the entry, exit and an output transition as e.g. shown in Figure 15b, then it is obvious to introduce the arc (p_T, t_4) weighted with 1 (due to input proper event insertion). However, the question is which transition of this path has to be connected to an ingoing arc of p_T; in Figure 15b two alternatives for the introduction of p_T are shown. The solution on the left (i.e. introducing the arc (t_1, p_T)) avoids after a suitable decomposition the considered irreducible CSC conflict, but it increases also the concurrency degree of N as well as the concurrency degree of the component STGs. However, the concurrent insertion leads to fast implementations since the critical component notifies the delay component earlier about the registration of the critical edge a^+ as in the sequential insertion of p_T shown on the right hand side of Figure 15b.

In fact, the sequential insertion avoids the increase of concurrency, but forces the so-called *uncontrollable growing* of the STG components; i.e. due to the T-place (or T-net) introduction in a sequential manner it is possible that further signals will get



(a) Example case (b) Longer path (c) Several a± labels

Figure 15: Dealing with a simple path from the entry transition leading to the exit transition



(a) Overall STG N^T (b) STGz_ic^T

Figure 16: Sequential insertion of p_T in N

relevant for a certain component STG and thus all transitions labeled with the new relevant signal cannot be contracted in the critical component – and this can lead to many side-effects, such as large component STGs or new irreducible CSC conflicts. Figure 16 shows this situation by an example. As one can see, the introduction of a T-net in STGz_ic (Figure 16b) forces the relevance of signal c in STGz_ic^T; however, this leads to a new irreducible CSC conflict in STGz_ic^T which is characterized by the input self-trigger $t_9 t_{11}$. Note that if the T-net would be concurrently inserted then such an uncontrollable growing of STGz_ic^T is not possible since due to the connection of p_T to the entry transition no further signals will get relevant in the critical component STG and thus no new irreducible CSC conflicts will arise.

Since both solutions have pros and cons, in the next examples we will show always both techniques, i.e. the *concurrent insertion* of p_T (as in Figure 15b on the left) as well as the *sequential insertion* (as in Figure 15b on the right).

First, let us state the basic idea for the systematization of the insertion of p_T in N (according to Figure 15b). A forward traversal must be started from the entry transition and must visit an output transition (t_4) before the exit transition is reached. Then p_T will be connected by an arc (p_T, t_4) to the output transition. An ingoing arc to p_T must be connected either to the entry transition (concurrent insertion of p_T by the arc (t_1, p_T)) or to the previously visited input transition (t_3) before the output transition (t_4) was reached (sequential insertion of p_T by the arc (t_3, p_T)). If no output transition is found before the exit transition is reached, then this procedure aborts and no solution can be given by the use of the presented approach (see also Section 4.3).

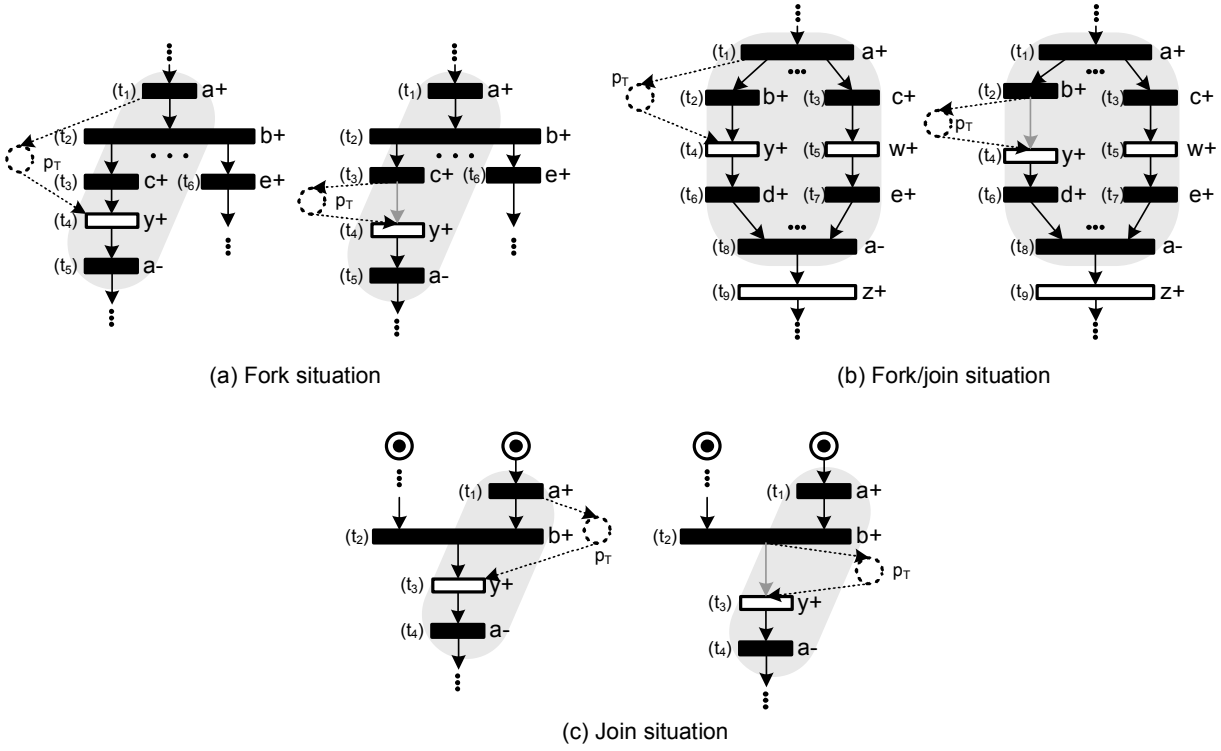
This graph searching procedure can also be applied if there are further transitions labeled with $a^{\pm}$ in N as one can see by means of an example in Figure 15c.

If there are any concurrent structures in N as shown in the examples of Figure 17, then the introduced graph searching procedure must be improved as follows:

First, we consider two different fork situations, i.e. the path starting from the entry transition forks at this entry transition (see Figure 17b) or at one of the succeeding transitions (see Figure 17a). Then it must be checked whether the exit transition can be reached on all forked paths before the entry transition is reached a second time (see Figure 17b) or the exit transition is only reachable on some of them (see Figure 17a), because the forked paths do not join until the exit transition is reached (then only one forked path part contains a transition labeled with a^- since auto-concurrency in N is not allowed).

Considering a structure according to Figure 17a, the forward traversal visiting the path part which contains the exit transition must traverse an output transition (t_4) prior to this exit transition and p_T must be connected to this output transition by an arc (p_T, t_4); otherwise the procedure aborts. The ingoing arc to p_T could be connected either to the entry transition (concurrent insertion of p_T by the arc (t_1, p_T), see Figure 17a left) or to the preceding transition of t_4 (sequential insertion of p_T by the arc (t_3, p_T) as shown in Figure 17a right).

Figure 17b describes the situation where the forked paths are joined at the exit transition t_8. In this case, it is enough that only one of the forked path parts contains an output transition which is connected to p_T by the arc (p_T, t_4). Note that in the considered safe, life and reversible STGs each transition of a forked path part always fires



(a) Fork situation

(b) Fork/join situation

(c) Join situation

Figure 17: Dealing with concurrency in N(left part: concurrent insertion; right part: sequential insertion)

before the exit transition fires and thus the introduction of (p_T, t_5) is also an opportunity. Again, a concurrent insertion of p_T by the arc (t_1, p_T) is possible as well as the sequential insertion (t_2, p_T) (or if the output transition t_5 is used, the arc (t_3, p_T) realizes the sequential introduction of p_T).

If there is no fork situation in N on the path of the transitions of a CTP, but a join situation as shown in Figure 17c, then nothing has to be changed for the suggested searching procedure. Consequently, p_T could be introduced in a concurrent or sequential manner as well (see Figure 17c).

For conflict situations in N the graph searching procedure must be further improved (see Figure 18). If the path starting from the entry transition (t_1) branches (at a place p_c), then (on account of the consistency of N) on all branches a transition labeled with a^- must be reached – e.g. the exit transition t_8 as in Figure 18b, or by means of Figure 18a at first the exit transition t_7 and second a different a^- labeled transition (t_6). In order to insert p_T concurrently, the forward traversal starting from the entry transition must visit on *each* branch an output transition which is connected to an outgoing arc of p_T. Such an output transition must be visited *before* either the exit transition is reached or the entry transition for a second time (or respectively a transition with the same label as the entry transition), otherwise the procedure aborts. For the examples on the left hand side in Figure 18a and b such necessary output transitions can be found, namely t_4 and t_5. These output transitions are connected to p_T by the arcs (p_T, t_4) and (p_T, t_5). Since the T-place should be introduced in a concurrent manner, the 1-weighted arc

(t_1, p_T) must also be added in N.

Let us assume the T-place would not be connected to an output transition on some branch z_i, then this would lead to an unbounded STG N, since if the introduced T-place is refined by a T-net (see Figure 11), the place p_o could get an unbounded amount of tokens if always branch z_i is chosen.

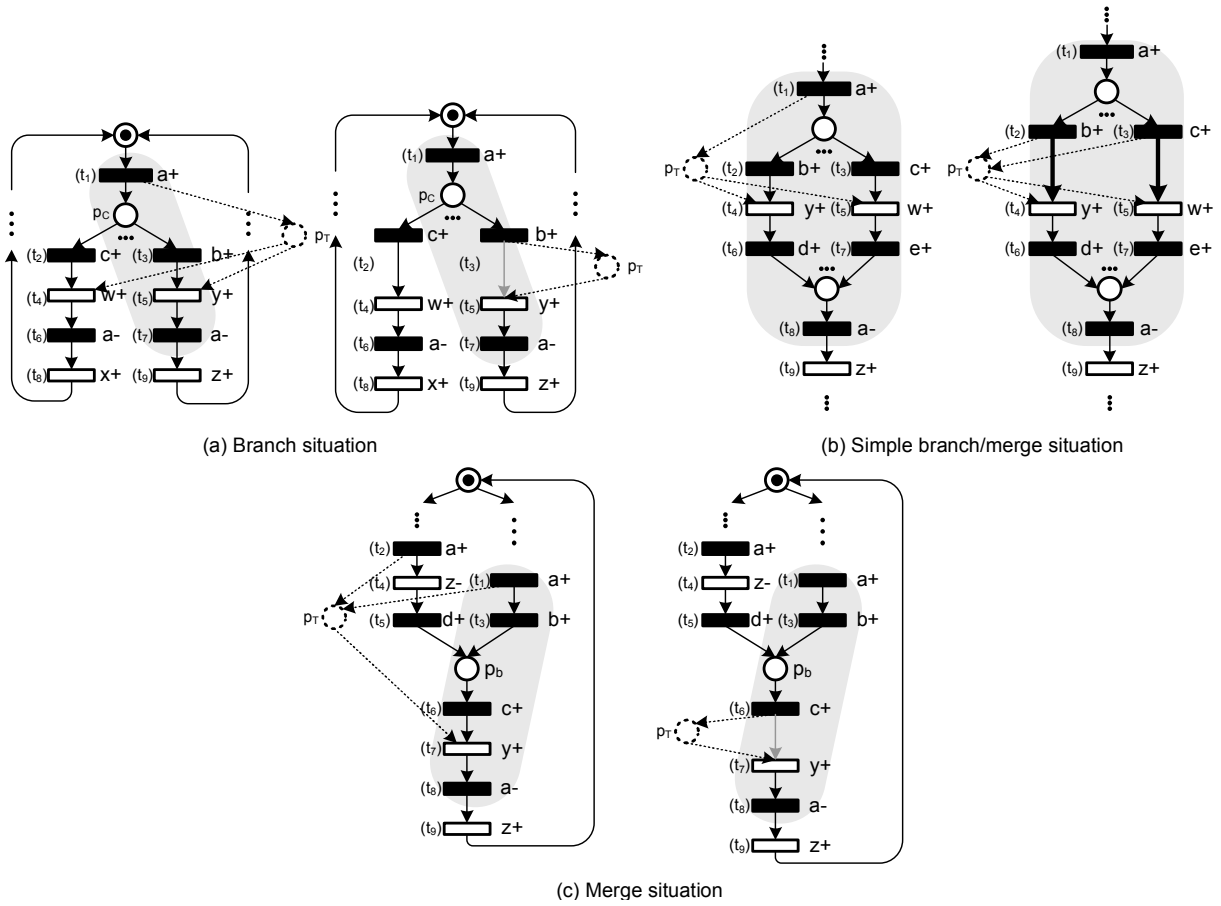
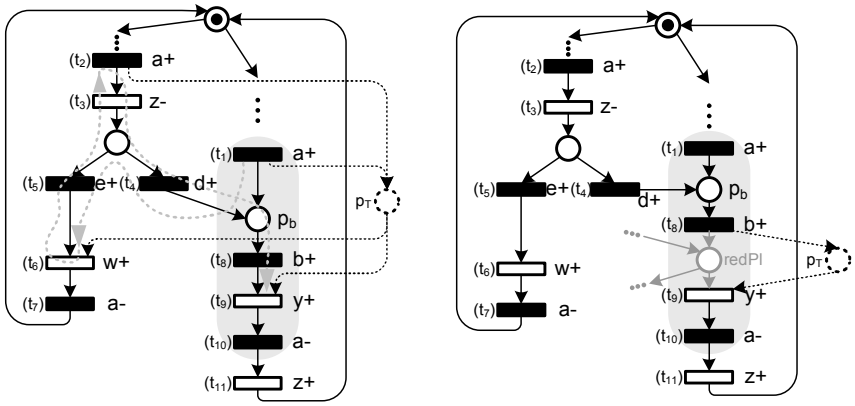


Figure 18: Dealing with conflicts in N(left part: concurrent insertion; right part: sequential insertion)

Now if p_T is inserted in a sequential manner (as shown on the right in Figures 18a and b), then it is enough to connect the outgoing arcs of p_T only to output transitions of branches leading to the exit transition (e.g. by the arcs (p_T, t_5) and, in Figure 18b, additionally (p_T, t_4)). Furthermore, there must be arcs to each preceding transition of t_5 or t_4 resp. as presented on the right hand sides in Figures 18a and b. Note that the bold highlighted arcs connecting t_2 and t_4 or t_3 and t_5 resp. in Figure 18b right must not be deleted, since this would introduce a dynamic conflict between t_4 and t_5, thus changing the specified behavior by unfair means.

Let us assume that p_T will be introduced concurrently in N as shown in Figure 18c on the left side: The forward traversal starting from the entry transition t_1 visits a place p_b (on which two or more paths in general merge); observe that the forward traversal has not visited an output transition or the exit transition so far. In this case, a *backward* traversal must be started for all paths starting from p_b besides the path that already has

been visited by the forward traversal. The backward traversal searches for transitions which are labeled with the same label as the entry transition (a^+); in Figure 18c this holds for transition t_2. T-place p_T has to be connected to this transition by an ingoing arc (t_2, p_T). Let us assume, the arc (t_2, p_T) has not been introduced in N, then p_b could be marked due to the firing of t_5 instead of t_3's firing. With the refinement of p_T by a T-net in mind, neither p_o nor p_u is marked in this situation. Thus the output transition t_7 could not fire anymore. Observe that a sequential introduction of p_T (see Figure 18c right) is much easier in this situation.



(a) Concurrent introduction of p_T (b) Sequential introduction of p_T

Figure 19: Merge situation with an included branch

In the more complex example of Figure 19 the sequential introduction of p_T (b) is also much easier in contrast to the concurrent introduction of the T-place (a). If a T-place p_T should be introduced in N concurrently 'between' the transitions of a CTP, then a recursive forward traversal must be started from the entry transition (following the gray dashed arrows in Figure 19a). If this forward traversal visits merge places (i.e. places with more than one ingoing arc, e.g. the place p_b) before reaching an output transition or the exit transition resp., then for all merge paths – besides the path which already has been visited by the forward traversal – a backward traversal must be started from the merge place. If such a backward traversal visits a new conflict place, then on such conflict place a new forward traversal has to be started on all forward directed paths starting from this conflict place (besides the path which has already been visited by the backward traversal). The aim of each forward traversal is the visiting of an output transition prior to the exit transition or a transition with the same label as the entry transition resp. If such an output transition cannot be found, then the procedure fails; otherwise, the found output transitions are connected to outgoing arcs of p_T. The aim of the backward traversal is to find transitions with the same label as the entry transition; these have to be connected to ingoing arcs of p_T.

Note that for the sequential introduction of p_T such recursive traversing procedures can often be simplified. With respect to the example in Figure 19b such complex traversing techniques only have to be applied if there is a branch situation or a merge situation at the place redPl between the found output transition (t_9) and their preceding transition (t_8). Such a branch or merge resp. on redPl is indicated by the gray arcs in

Figure 19b but since it does not really exist the insertion of p_T is much more simple in comparison to the T-place insertion shown in Figure 19a. Anyway, the disadvantage of the sequential insertion of p_T is the uncontrollable growing of the component STGs, i.e. in the critical component further signals can get relevant and so further transitions will not be contracted in the component STG and this could lead to new irreducible CSC conflicts (as already mentioned above). The concurrent introduction of p_T avoids the disadvantage of the uncontrollable growing of the component STGs, since the introduced T-place will only be connected to input transitions labeled with still relevant signal edges (in this context the concurrent insertion performs more locally than the sequential one) and provides faster implementations due to the concurrent generation of $ic^{\pm}$.

If the suggested graph searching procedure finishes without any failure and if during a traversal a marked place p_m was visited, then the introduced T-place p_T must be marked and after its refinement the place p_u, too, according to $M(p_m) = M(p_T) = M(p\_u)$. In all other cases $M(p_T) = M(p\_u) = 0$ holds.

4.3 Limitations and Suggestions Towards a Comprehensive Solution

As explained in the previous section the graph searching procedure could fail if there is no output transition on the path from the entry transition to the exit transition (i.e. there is no output that can be used to delay the critical edge in the critical component). Thus the considered irreducible CSC conflict cannot be avoided by the presented approach. Figure 20 shows an example of an overall STG n without irreducible CSC conflicts leading to a component C_c with an unsolvable irreducible CSC conflict since there is no output transition on the path in N from the entry (a^+) to the exit transition (a^-) which could be used to delay the transition labeled with the critical edge a^-.

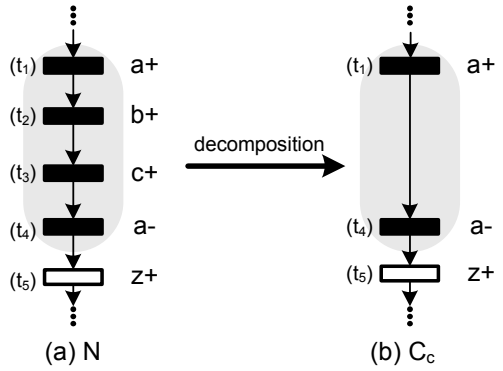


Figure 20: Unsolvable irreducible CSC conflict

Maybe delambdarization is an opportunity to deal with such examples. In this context, the term *delambdarization* means that an actual irrelevant signal for the generation of the outputs of a considered component is treated as relevant; i.e. the transitions labeled with the delambdarized signal are not subject to a transition contraction anymore

and are reintroduced as *input* transitions in the resulting component STG. In the example of Figure 20 signal b could be delambdarized in the critical STG component C_c leading to an STG component C_c' (resulting from N, by the contraction of transition t_3). In fact, this avoids the considered irreducible CSC conflict, but it may force further input self-triggers in C_c'; e.g. due to the existence of further transitions (modeling edges of signal b) new critical input traces b^+b^- are possible. Of course, this is an undesired side-effect which is called uncontrollable growing of C_c' (and has already been discussed in Section 4.2.3 in another context). Another example for delambdarization is shown in Figure 21c. In order to avoid the input self-trigger a^+a^- in STGz the signal y is delambdarized. Since the overall STG does not necessarily satisfy CSC (see Figure 21b), new (irreducible) CSC conflicts can arise; observe that STGz has a new CSC conflict characterized by the critical trace $a^+y^+a^-y^-$ (see Figure 21c) which is obviously irreducible.

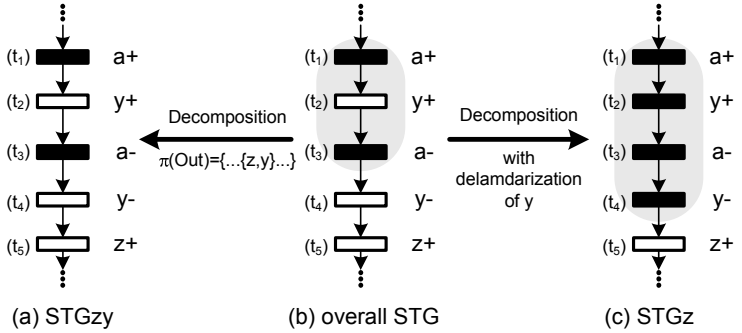


Figure 21: Delambdarization vs. coarse grained partitioning

Alternatively, signal y could be introduced as an output signal as well in STGz yielding STGzy (Figure 21a). The introduction of y as an output can be realized by a so-called *coarse-grained partitioning*, i.e. the output partition $\pi(Out)$ must look like $\{...\{z, y\}...\}$. This avoids the input self-trigger a^+a^- in STGzy, too, but due to the generation of the new output y by the critical component, maybe further triggers for all the y labeled transitions are needed and thus the set of relevant signals for STGzy may grow significantly. Moreover, it is possible that a certain output signal is needed to avoid irreducible CSC conflicts in more than one component STG, say in C_1 and C_2. Since an output can only be generated by one component, the STG components C_1 and C_2 must form one combined STG component $C_{1,2}$ which is subject to logic synthesis. As a consequence of this coarse-grained partitioning, perhaps only few transitions are contractable or, at the worst, no decomposition can be applied at all.

Thus the presented introduction of *new* internal communication signals or transitions resp. is more flexible and can be performed more locally with respect to the considered irreducible CSC conflict; i.e. a suitable introduction of ICTs avoids uncontrollable growing of the resulting component STGs, since it does not effect any transitions which are not related to some considered irreducible CSC conflict.

According to the previous discussion some conclusions will be given with respect to Figure 22 i.e. some considerations towards a comprehensive approach in order to deal with irreducible CSC conflicts.

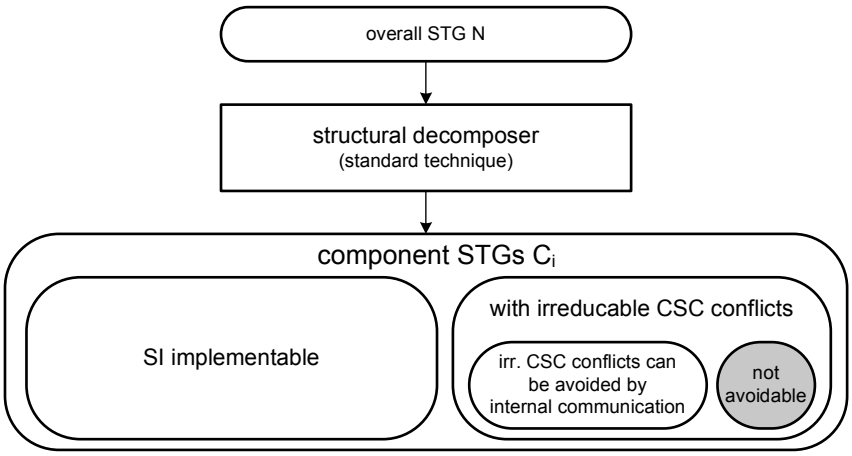


Figure 22: Classification of component STGs resulting from STG decomposition [26]

Let us assume, we would like to synthesize a complex circuit based on decomposition. Then the decomposition [21, 25, 26] could lead to component STGs C_i and on the one hand side some of them are SI implementable and on the other hand some of them have irreducible CSC conflicts. Hopefully, these irreducible CSC conflicts can often be solved by the presented approach (i.e. by the introduction of a suitable internal communication among the component STGs). However, if the conflicts of some of the critical STG components are not avoidable by the presented method (in Figure 22 these component STGs are represented by the gray filled circle), the application of delambdarization as described e.g. in [15] may be a promising alternative. Another opportunity to deal with such critical STG components is to build larger component STGs by using coarse-grained partitioning as discussed above. Further research will focus these possibilities.

5 Examples

In this section the suitability of the presented approach will be demonstrated on two examples. The first example – a FIFO controller – is a simple textbook example known from literature [3] and the second one is a handshake circuit controller design which is of more practical relevance since it describes the application of logic synthesis to a model resulting from an HDL specification (such as a BALSA-program).

5.1 FIFO Controller

This textbook example describes the design of a FIFO stage which can be used as a component of an entire FIFO memory realized by a cascade of such FIFO stages [3]. Incoming data of the entire FIFO memory will be distributed alternately throughout two D-register pipelines under the control of the signals RD. In the last FIFO stage a multiplexer controlled by RM generates the output of the entire FIFO memory (i.e. it restores the original order of the data items) and this last FIFO stage will be considered now (see

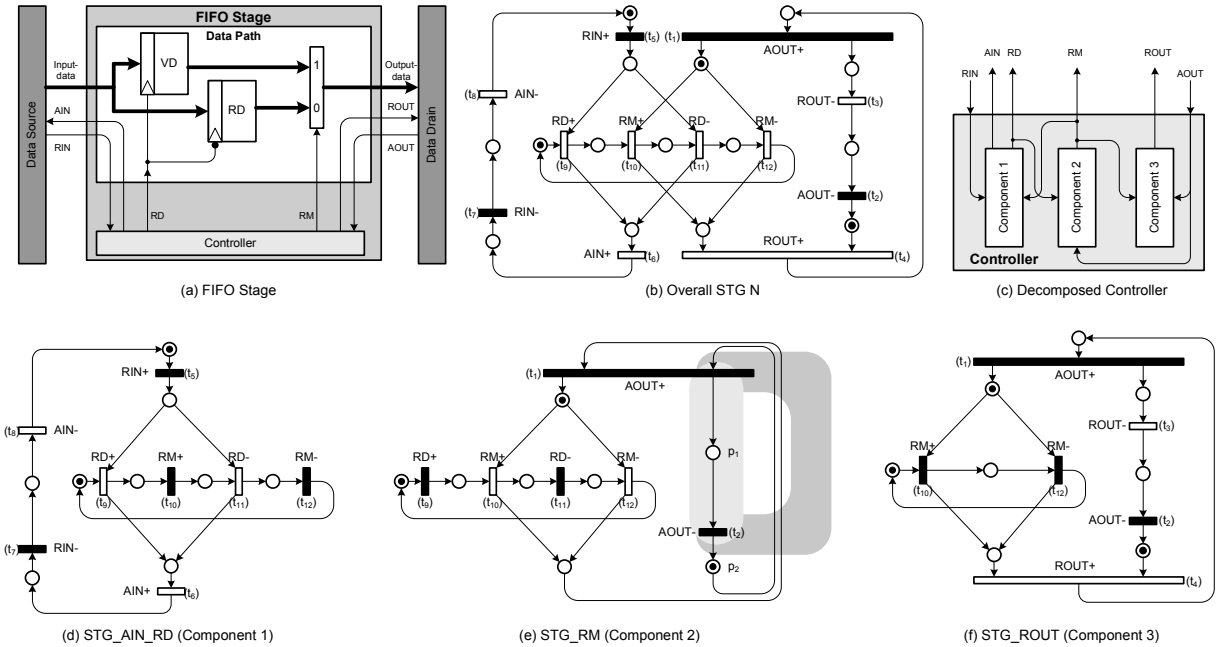


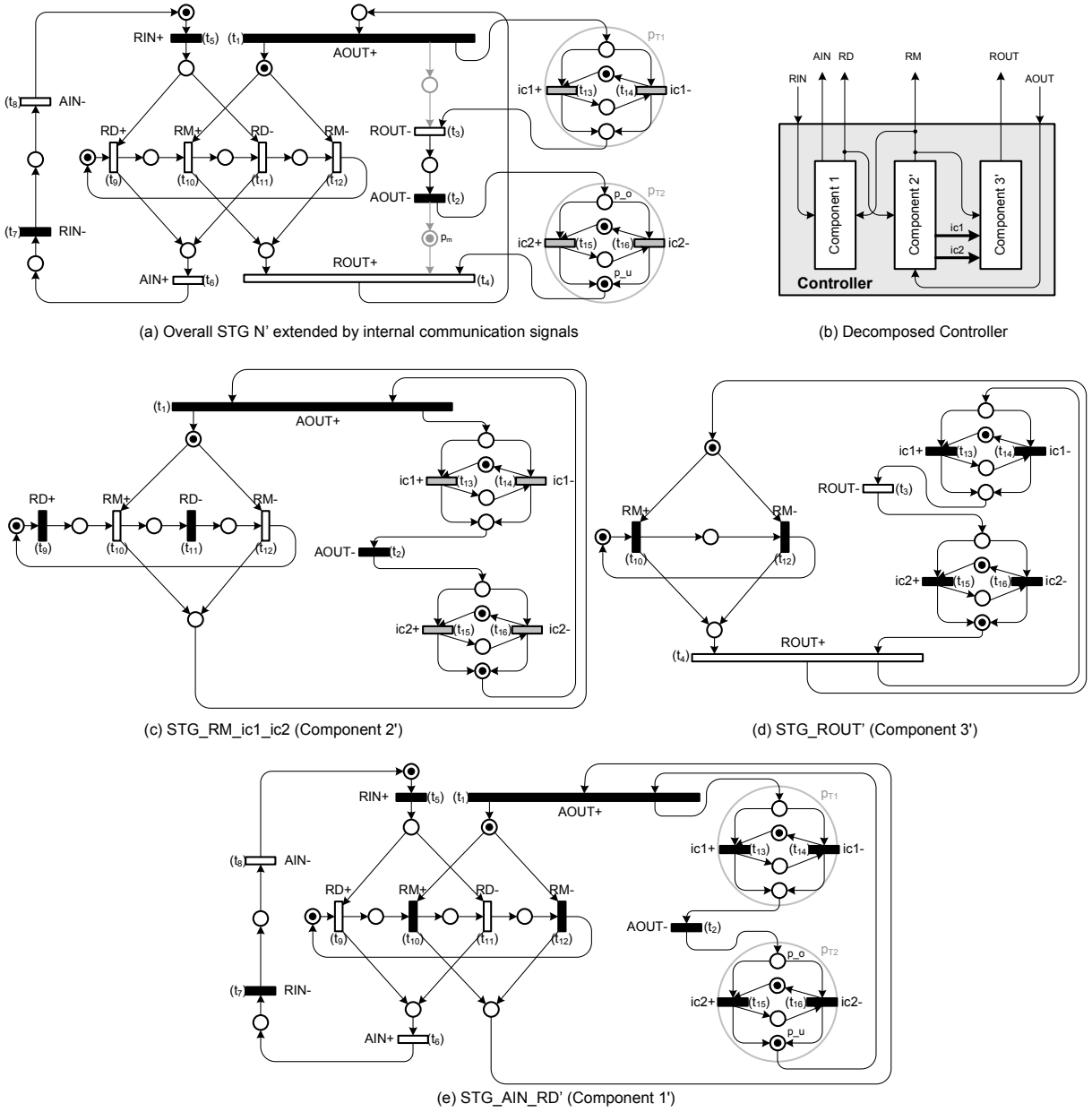
Figure 23: Design of a FIFO controller stage using STG decomposition (taken from [3])

Figure 23a). The data path of the FIFO stage should be controlled by an asynchronous controller whose interface behavior is described by the overall STG N in Figure 23b. It is intended to realize the controller by decomposition based synthesis according to the controller architecture in Figure 23c. Observe that component 1 generates two outputs while each of the other components generates only one output. So N has to be decomposed according to the output partition $\pi(Out) = \{\{AIN, RD\}, \{RM\}, \{ROUT\}\}$. Then the component STGs shown in Figures 23d, e and f result from decomposition.

Indeed, a speed independent logic synthesis of the component STGs STG_AIN_RD and STG_ROUT is possible, but the SI synthesis of STG_RM is impossible due to the two input self-triggers t_1t_2 and t_2t_1 which are highlighted gray in Figure 23e. These input self-triggers can be found in a structural way on the basis of Proposition 4.3. Since only input self-triggers are found, for each of them only one CTP can be identified, i.e. (t_1, t_2) and (t_2, t_1) resp. For each CTP a T-place p_{T1} and p_{T2} resp. has to be introduced according to the approach presented in Section 4.2 and then each T-place must be refined into a T-net consisting of internal communication transitions labeled with $ic1^{\pm}$ and $ic2^{\pm}$ resp. (see Figure 24a). This will be explained now in more detail.

In order to find a suitable position in N for the insertion of p_{T1} regarding the CTP (t_1, t_2) a forward traversal at the entry transition t_1 is started. On a simple path (without any branches, merges, joins or forks) the output transition t_3 is reached prior to the exit transition t_2. Thus p_{T1} must be connected to t_3 by an 1-weighted arc (p_{T1}, t_3). Since the entry transition is the preceding transition of t_3, the sequential as well as the concurrent introduction of p_{T1} lead to the same result: p_{T1} must be connected to the entry transition by the arc (t_1, p_{T1}) (as shown in Figure 24a).

The T-place p_{T2} is introduced with respect to the CTP (t_2, t_1) in N. A forward traversal is started at the entry transition t_2 and reaches also on a simple path the output



(a) Overall STG N' extended by internal communication signals

(b) Decomposed Controller

(c) STG_RM_ic1_ic2 (Component 2')

(d) STG_ROUT' (Component 3')

(e) STG_AIN_RD' (Component 1')

Figure 24: Extension of the FIFO controller in order to reach SI implementability

transition t_4 prior to the exit transition t_1. Consequently, the arcs (p_{T2}, t_4) and (t_2, p_{T2}) are introduced. Since during the traversal a marked place was visited, $M(p_m) = 1$, the T-place p_{T2} gets one token and thus the place p_u has to be marked, i.e. $M(p\_u) = 1$. This results in the extended overall STG N' (Figure 24a).

Now, the decomposition of N' must be applied according to a suitable output partition in such a way that the introduced internal communication transitions contribute to the avoidance of the considered irreducible CSC conflicts in STG_RM, i.e. we need the partition block $\{RM, ic1, ic2\}$ in $\pi(Out)$. Since STG_AIN_RD as well as STG_ROUT were already SI implementable, the partition blocks $\{AIN, RD\}$ and $\{ROUT\}$ remain unchanged. So we decompose N' according to the partition $\pi(Out) = \{\{RM, ic1, ic2\}, \{AIN, RD\}, \{ROUT\}\}$ and get the component STGs shown in Figures 24c, d and e. All of these STG components are SI implementable and thus the entire FIFO controller is.

However, the decomposition does not only lead to a modified STG of the critical component (STG_RM_ic1_ic2) and the delay component (STG_ROUT'), but it leads also to a modified version STG_AIN_RD' for component 1 (see Figure 24e), even though component 1 is not directly involved in the internal communication for the avoidance of the considered irreducible CSC conflicts in STG_RM. Anyway, this rather complex STG_AIN_RD' arises since the transitions of a T-net cannot be contracted in a secure manner according to [26]. To show this, let us assume the transition labeled with ic^- in the T-net in Figure 11 should be contracted, then a secure contraction of this transition t is impossible, since neither $({}^\bullet t)^\bullet \subseteq \{t\}$ nor ${}^\bullet(t^\bullet) = \{t\}$ holds, i.e. the t-contraction is neither type-1 secure nor type-2 secure. However, since component 1' does not contribute to the delay of the critical edge in component 2, it obviously must be possible to replace component 1' by component 1 in the resulting controller implementation (see Figure 24b).

As you can see in Figure 24b, the implementation has two additional signals $ic1$ and $ic2$ in comparison with the block diagram in Figure 23c. These additional two signals act as channels for the introduced one-sided internal communication from component 2' to component 3'.

Of course, the STG component STG_ROUT' (see Figure 24d) is also modified in comparison to STG_ROUT (Figure 23f), even though it was already SI implementable. However, such an extension of STG_ROUT by T-nets is necessary, since it specifies the behavior for the delay component which must realize one part of the introduced one-sided internal communication. Nevertheless, no further irreducible CSC conflicts are expected by such an extension of the delay component STG; of course, this must be proved in future work.

Finally, let us consider the complexity of the results shown in Table 1 (Note that C_{x_y} represents an STG component producing the outputs x and y.). The first row shows the sizes of the reachability graphs of the original overall STG (Figure 23b) and the corresponding component STGs (Figures 23d, e and f). Note that component STG_RM (see Figure 23e) is not SI implementable. Due to the introduction of the T-nets as presented in this work the component STGs grow at least twice as much in sizes of their reachability graphs in comparison to the component STGs without any extensions for internal communication. In particular STG_RM and STG_ROUT: they

	N	C_{AIN_RD}	C_{RM}	C_{ROUT}
standard technique (without int. comm.)	140	20	16	14
all T-nets	220	160	32	32
necessary T-nets	220	20	32	32
single transition insertion	220	20	32	16

Table 1: Marking counts of the component reachability graphs of the FIFO example

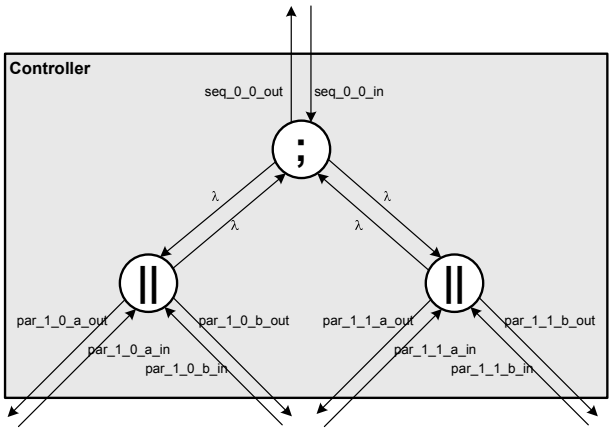
have 16 and 14 markings without any internal communication extensions and with both T-nets 32 markings. Since the decomposition approach of Vogler et al. so far cannot contract transitions of a T-net in a secure manner (as already explained) the component STG producing AIN and RD has 160 markings instead of 20 markings. Since the original FIFO controller STG in Figure 23b has only 140 markings the 160 markings are not acceptable and we use instead STG_AIN_RD from the decomposition of the original overall STG. This must be possible since component 1 does not contribute to the internal communication that avoids the considered input self-triggers.

In spite of the higher complexity of STG_RM_ic1_ic2 and STG_ROUT' in comparison to STG_RM and STG_ROUT they can be synthesized by PETRIFY and so the presented approach is an opportunity to cope with the state explosion problem during logic synthesis of complex overall STGs.

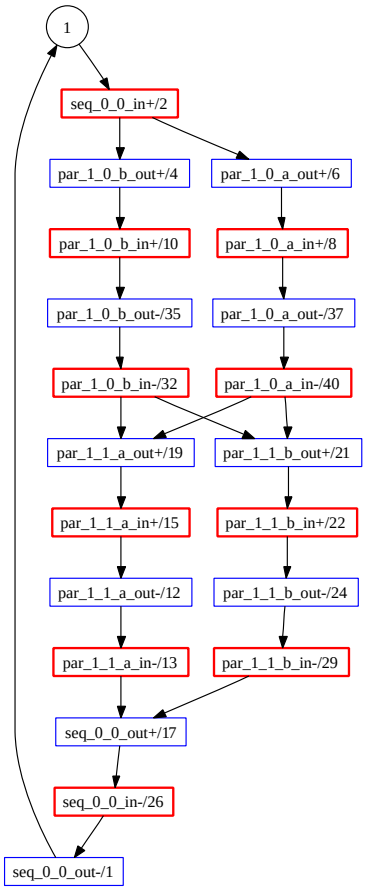
With regard to further optimizations, only single internal transitions labeled with ic^+ are introduced for the avoidance of a particular input self-trigger instead of introducing an entire T-net. In order to preserve the consistency of the overall STG, the complementary edge ic^- is inserted by hand (input proper event insertion) this will be called the *single transition insertion* approach. In fact, this does not lead to a smaller overall STG but at least to one smaller component STG (C_{ROUT}) with 16 instead of 32 markings (see the 4th row in Table 1).

5.2 Sequencer/Parallelizer Tree

The sequencer/parallelizer tree is of more practical relevance, since the design entry point is an HDL program (e.g. BALSA or TANGRAM). Then a netlist of so-called *handshake components (HCs)* could be derived by syntax directed translation, see Figure 25a. Note that the circles model these handshake components. We can get an (overall) STG modeling the behavior specified by this HC netlist due to parallel composition of the single HC STGs. Each HC STG describes the interface behavior of a particular handshake component (as already explained in the introduction). However, since only the *overall* I/O behavior of the entire control part is of interest for logic synthesis, the internal HC communication signals among the handshake components can be hidden in the parallel composed overall STG (i.e. their corresponding transitions can be contracted). In Figure 25a the signals to hide are labeled with λ. Hiding these internal HC communication signals hopefully leads to better hardware implementations compared to direct mapping techniques. The application of parallel composition to



(a) HC netlist (internal HC communication signals are labeled with λ)



(b) Overall controller STG (λ-transitions already contracted)

Figure 25: Sequencer/Parallelizer Tree with two levels

the HC STGs as well as the hiding of internal HC communication signals on the sequencer/parallelizer netlist in Figure 25a leads to the overall STG shown in Figure 25b. (Observe that input transitions are drawn in red and output transitions in blue. If you cannot distinguish colors on your print, the signal types in the HC netlist in Figure 25a give also the information whether a transition is an input or an output transition.)

Of course, this example could be expanded to a sequencer/parallelizer tree with more than two levels. But if it has four or more levels, then PETRIFY cannot synthesize the corresponding controller STG due to state explosion. Thus STG decomposition is needed, but with the decomposition according to [21, 25, 26] problems arise similar to those that arise in the two level sequencer/parallelizer tree and so – for simplicity – we consider only this smaller example now.

After a *total* decomposition of the overall STG (i.e. one output signal per STG component) at least one input self-trigger arises in each component STG (see Figure 26, each transition pair forming an input self-trigger is surrounded by a dashed polygon). The caption at the bottom of each component STG identifies its single output signal i.e. all transitions labeled with an edge of this output signal are output transitions and the all the other transitions are inputs.

Considering all component STGs in Figure 26, via Proposition 4.3 one can identify all transitions belonging to input self-triggers and, if during the decomposition the transition names are preserved, then the transitions forming the input self-triggers can be identified in the overall STG.

For all of the eight identified input self-triggers and their corresponding CTPs eight T-places p_{T1} ... p_{T8} have to be introduced in the overall STG leading to the extended overall STG N^T (shown in Figure 27). Then, each of these T-places must be refined by a T-net modeling the internal communication via the signals $ic1$, $ic2$, ... or $ic8$ resp.

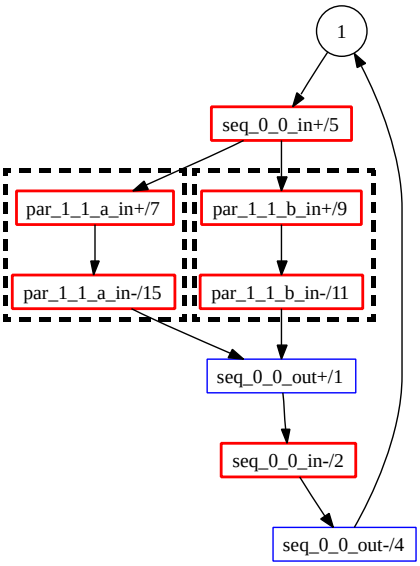
Finally, the extended STG N^T must be decomposed according to the following suitable output partition:

$$\pi(Out) = \{\{seq\_0\_0\_out, ic1, ic2\},\ \{par\_1\_0\_b\_out, ic8\},\ \{par\_1\_1\_a\_out, ic3, ic4\},\\ \{par\_1\_0\_a\_out, ic7\},\ \{par\_1\_1\_b\_out, ic5, ic6\}\}$$

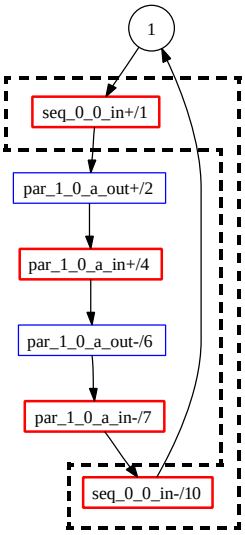
After this, all of the five resulting component STGs can be implemented as speed independent circuits e.g. by PETRIFY. Figure 28 shows one of these components, namely $C_{seq\_0\_0\_out}$. By the way, the index seq_0_0_out represents the output signal which is produced by this component.

However, the component STGs are rather complex due to the sizes of their reachability graphs, since the contraction of transitions of a T-net is not secure according to the decomposition of [25, 26]; thus each component STG still contains all eight T-nets. The second row of Table 2 shows the marking count of each STG component as the result of the decomposition [25, 26]. Indeed, these component STGs are SI implementable now, but their reachability graphs are up to a factor of 15 larger than the reachability graphs resulting from a standard decomposition of the STG shown in Figure 25b.

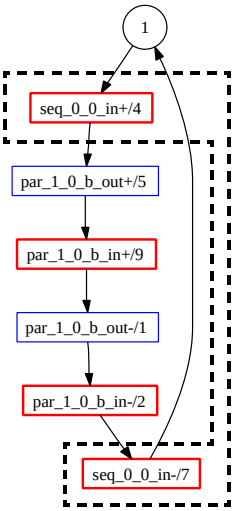
Regarding $C_{seq\_0\_0\_out}$ in Figure 28 the signals $ic3$, $ic4$, $ic5$ and $ic6$ are actually not relevant, but the provided decomposition approach cannot accomplish the contraction of the corresponding transitions. So in future work the contraction of entire STG parts, such as an entire T-Net structure, will be investigated. In this context, it is proposed



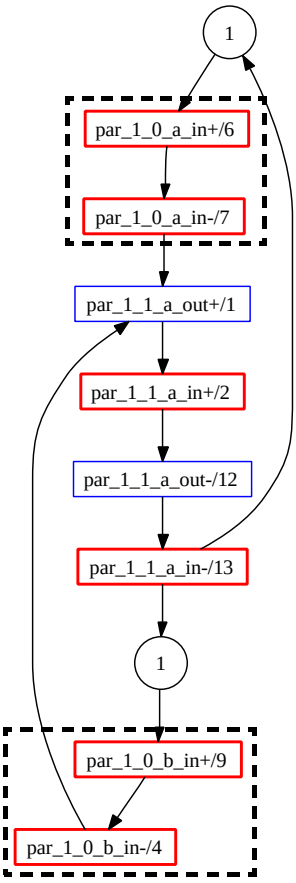
(a) seq_0_0_out



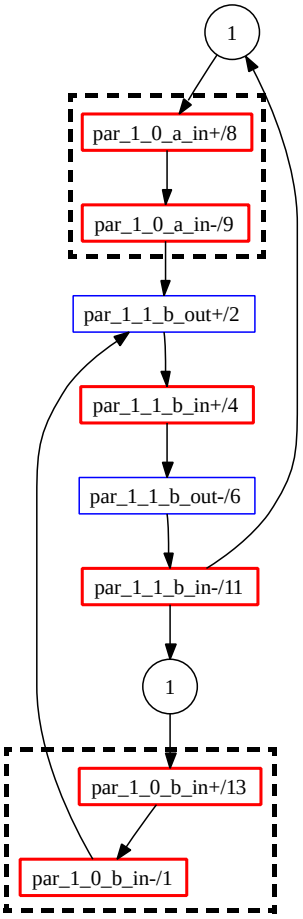
(b) par_1_0_a_out



(c) par_1_0_b_out



(d) par_1_1_a_out



(e) par_1_1_b_out

Figure 26: Decomposition results of the STG in Figure 25b (standard technique)

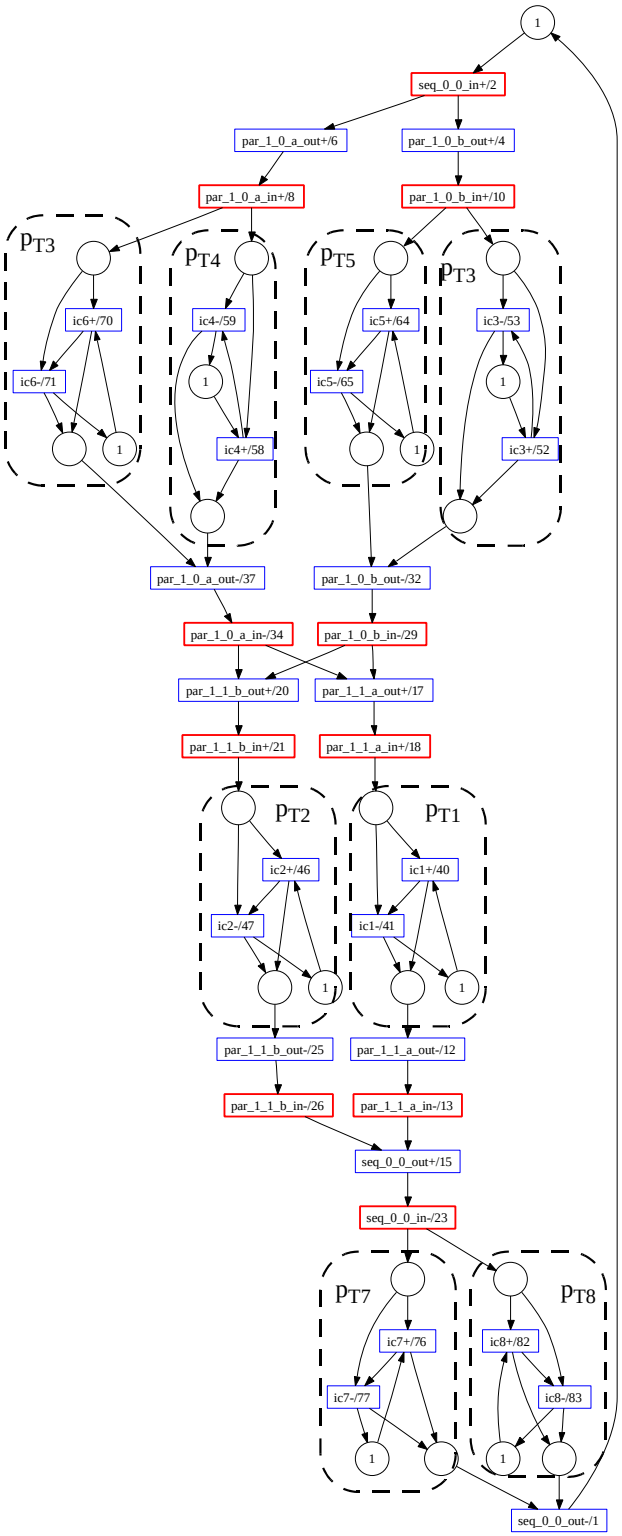


Figure 27: Overall STG N^T as an extension of N in Figure 25b modeling internal communication of the resulting component STGs by the outputs $ic1...ic8$ (to avoid irreducible CSC conflicts)

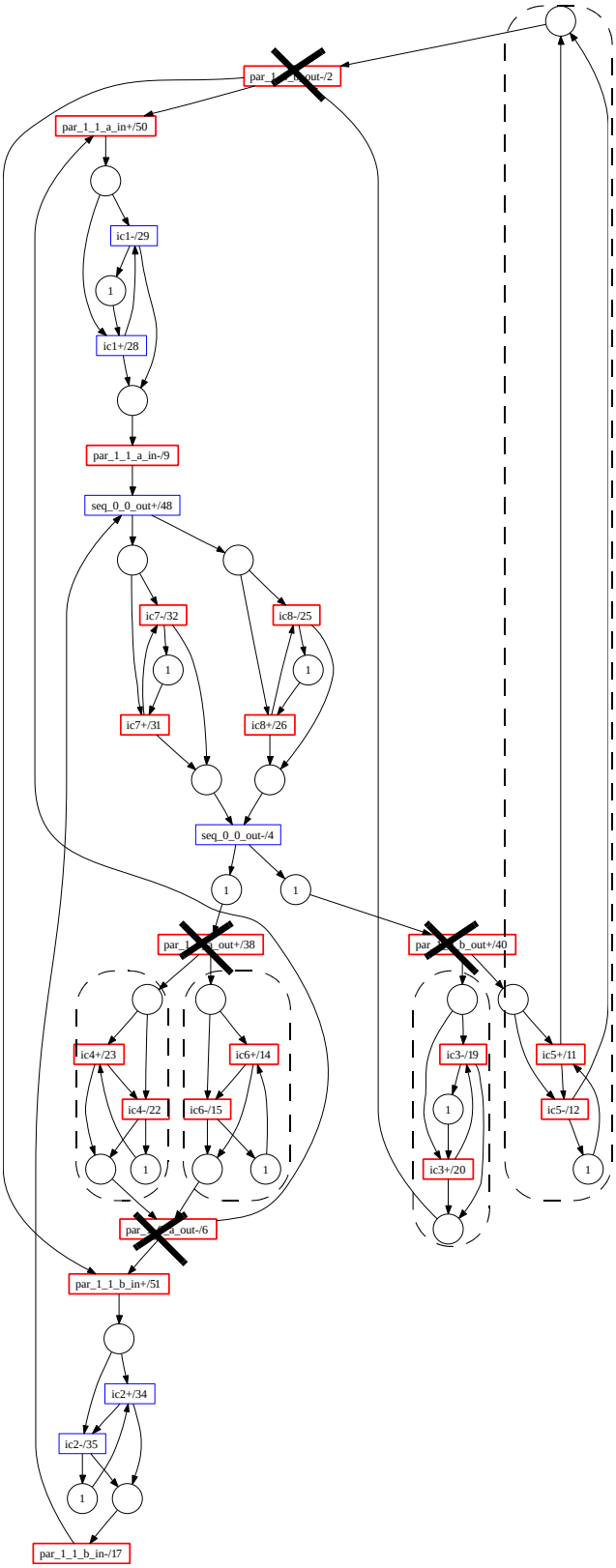


Figure 28: Component STG $C_{seq\_0\_0\_out}$ resulting from the decomposition of N^T (in Fig. 27)

	N	$C_{seq\_0\_0\_out}$	$C_{par\_1\_0\_a\_out}$	$C_{par\_1\_0\_b\_out}$	$C_{par\_1\_1\_a\_out}$	$C_{par\_1\_1\_b\_out}$
standard technique (w/o int. comm.)	52	12	6	6	12	12
all T-nets	210	110	90	90	94	94
necessary T-nets	210	40	16	16	36	36
single transition insertion	134	21	8	14	21	42

Table 2: Marking counts of the components for the two-level sequencer/parallelizer tree

to 're-refine' each T-net consisting of the transitions of an irrelevant signal by a T-place again, as outlined by the dashed circles in Figure 28. Then the standard decomposition technique even detects the signals par_1_0_a_out and par_1_0_b_out as irrelevant (and therefore the transitions labeled with these signals are crossed out in Figure 28). Thus the decomposition finally would lead to the reduced component STG $C'_{seq\_0\_0\_out}$ which is shown in Figure 29.

With this treatment of such T-net contraction in mind, for all component STGs resulting from the decomposition of N^T the component sizes and their marking counts are significantly smaller (cf. the second and the third row in Table 2). In fact, these marking counts are still larger than the amounts of markings of the STG components without any extensions for internal communication, but this enlargement averages to a factor of 3 instead a factor of 15. Note that the original overall STG (see Figure 25b) has 52 markings and no CSC while the component STGs containing only the necessary T-nets are *all* smaller and *do* satisfy already CSC, so they are more suitable to a PETRIFY synthesis in contrast to the original overall STG.

With regard to further optimizations, the single transition insertion approach leads to an overall STG which is much smaller (134 markings) as the overall STG containing the eight T-nets (210 marking). Moreover, its decomposition results are smaller than those containing the necessary T-nets except for the component generating output signal par_1_1_b_out (cf. 3rd and 4th row of Table 2).

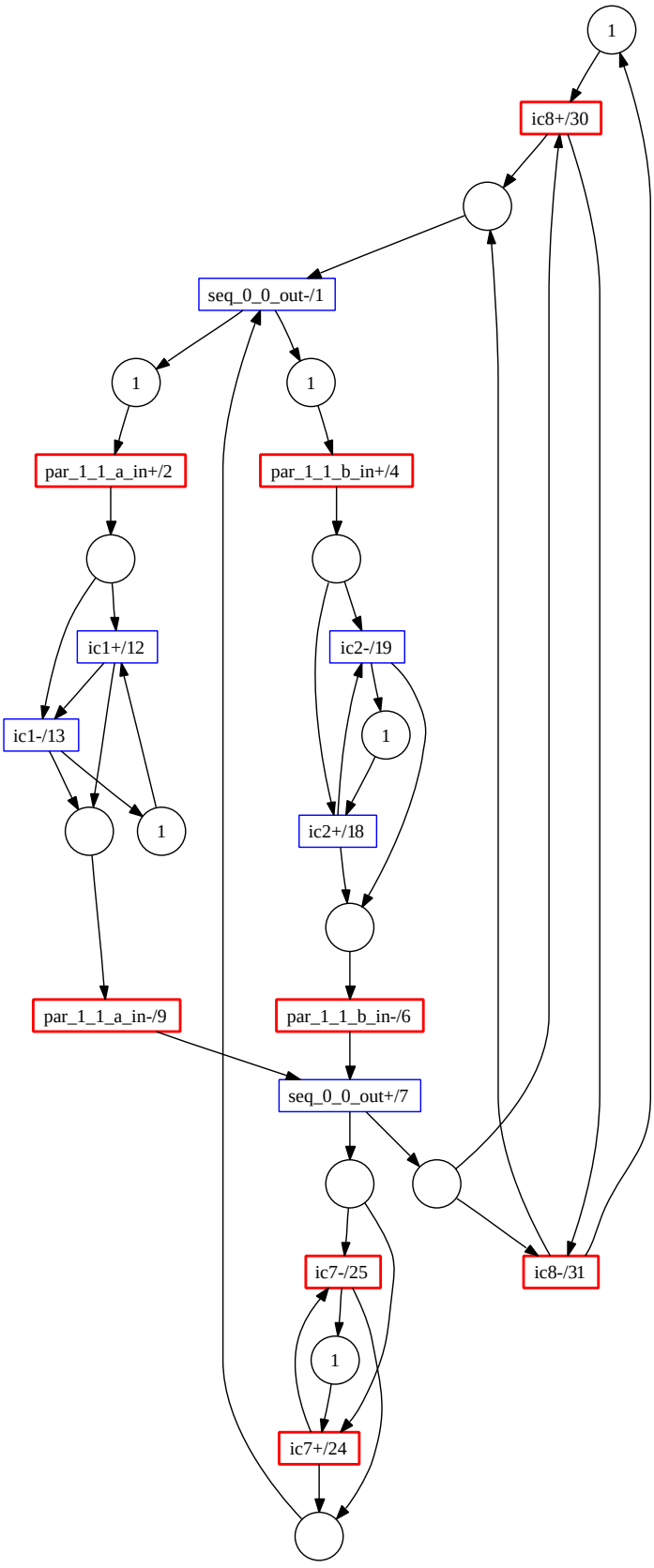


Figure 29: Component STG $C'_{seq\_0\_0\_out}$

6 Conclusions and Future Work

The presented idea may be an important step in asynchronous system design based on STG decomposition [21, 25, 26] and SI logic synthesis. Irreducible CSC conflicts in component STGs are avoidable by the introduction of internal communication among the STG components; this enables the SI logic synthesis in many cases and thus it contributes to the preprocessing problem (see also Figure 1).

Moreover, considerations towards an algorithmic structural solution for the introduction of internal communication are presented. Indeed, some restrictions on the net class must be accepted (see Section 4.3), but our approach seems to be useful in many practical cases. First promising results were presented in Section 5. However, a correctness proof for the presented method is still missing.

In future work, the approach explained in Section 4 has to be completed, yielding a 2-cycle algorithm that will be implemented and integrated in our decomposition tool DESIJ[3]. The two cycles are characterized as follows:

1. Find all irreducible CSC conflicts in the STG components and identify all transitions of the corresponding critical transition sequences in the overall STG.

2. Avoid the irreducible CSC conflicts by the introduction of internal communication transitions in the overall STG (according to the methods presented in Section 3 and 4) and a new decomposition with a suitable output partition.

So far especially input self-triggers are considered and in further research the handling of general CSC conflicts will be investigated in more detail. For that, we are aiming at the efficient extraction of a single CTP or at least a minimal set of CTPs from all CTSs belonging to a particular irreducible CSC conflict; then these extracted CTPs might be split by a single T-place and thus by a single T-net. Consequently, it is necessary to get all CTSs belonging to a certain irreducible CSC conflict at once and not only one instance of them as the tool PETRIFY does; maybe, the tool MPSAT provides an opportunity in order to deal with this problem.

Furthermore, the influence of the T-net introduction in the delay component by means of its SI implementability has to be investigated. Hopefully, it can be proved that it does not affect the SI implementability of the delay component, i.e. it should be proved that if a component STG producing some output x can be implemented as an SI circuit before the introduction of some internal communication transition, then it is still SI implementable after its introduction. In this context, also the side-effects of the T-net introduction with respect to STG components that are not involved in the internal communication (to avoid a particular irreducible CSC conflict) have to be investigated.

So far the decomposition of [21, 25, 26] cannot contract the transitions of a T-net in a secure manner. Thus the extension of the decomposition approach concerning the *secure* contraction of entire STG parts (such as entire T-nets) has to be investigated.

A different way to avoid this problem and to reduce the complexity of the resulting STGs is to replace the T-net insertion w.r.t. an internal signal ic by the insertion of a

[3] http://www.informatik.uni-augsburg.de/en/chairs/swt/ti/research/tools/desij

single internal transition labeled with one edge of ic only. (Of course we then have to insert the complementary transition at a different position in N in a consistent way.) This could lead to reachability graphs with significantly reduced sizes in comparison to the T-net introduction, as shown in the result tables in Section 5. Maybe the work presented in [14, 18] will give some suggestions for this improved transition insertion.

For STG components whose irreducible CSC conflicts are not avoidable by the presented method, the systematic reintroduction of *irrelevant* signals may help – or, in the case of overall STGs extracted from an HC netlist (resulting from a parallel composition of HC STGs with hidden internal HC communication signals) the reintroduction of these hidden signals could be successful. On the one hand these actual irrelevant signals can be reintroduced in the critical component STG as inputs ('delambdarization') or, on the other hand, as outputs (coarse-grained partitioning) – but both methods may lead to uncontrollable growing (see Section 4.3).

Acknowledgments
This research was supported by the DFG-project 'STG-Dekomposition' Wo814/1-3.
We would like to thank Mark Schäfer for the helpful cooperation.

References

[1] Andrew Bardsley. *Implementing Balsa Handshake Circuits*. PhD thesis, University of Manchester, Faculty of Science and Engineering, School of Computer Science, 2000.

[2] Andrew Bardsley and Doug Edwards. Compiling the language Balsa to delay insensitive hardware. In *CHDL'97: Proceedings of the IFIP TC10 WG10.5 international conference on Hardware description languages and their applications : specification, modelling, verification and synthesis of microelectronic systems*, pages 89–91, London, UK, 1997. Chapman & Hall, Ltd.

[3] Jochen Beister and Ralf Wollowski. Controller implementation by communicating asynchronous sequential circuits generated from a Petri net specification of required behavior. In *Synthesis for Control Dominated Circuits*, pages 103–115, 1992.

[4] Gerard Berthelot. Transformations and decompositions of nets. In *Proceedings of an Advanced Course on Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986-Part I*, pages 359–376, London, UK, 1987. Springer-Verlag.

[5] Josep Carmona and Jordi Cortadella. ILP models for the synthesis of asynchronous control circuits. In *ICCAD '03: Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, Washington, DC, USA, 2003.

[6] Tiberiu Chelcea and Steven M. Nowick. Resynthesis and peephole transformations for the optimization of large-scale asynchronous systems. In *DAC '02: Proceedings of the 39th conference on Design automation*, pages 405–410, New York, USA, 2002.

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

[8] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, 1997.

[9] 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.

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

[11] N. Karaki, T. Nanmoto, H. Ebihara, S. Utsunomiya, S. Inoue, and T. Shimoda. A flexible 8b asynchronous microprocessor based on low-temperature poly-silicon TFT technology. In *ISSCC '05: Proceedings of the Solid-State Circuits Conference, 2005*, pages 272–598. Seiko Epson Corp., Nagano, Japan, 2005.

[12] Victor Khomenko, Maciej Koutny, and Alex Yakovlev. Detecting state coding conflicts in STG unfoldings using SAT. In *Third International Conference on Application of Concurrency to System Design (ACSD'03), Guimares, Portugal*, pages 51–60, June 2003.

[13] 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, June 2004.

[14] 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.

[15] Victor Khomenko and Mark Schäfer. Combining decomposition and unfolding for STG synthesis. In *ICATPN '07: 28th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency*, pages 223–243, 2007.

[16] Victor Khomenko, Mark Schäfer, and Walter Vogler. Output-determinacy and asynchronous circuit synthesis. In *ACSD '07: Seventh International Conference on Application of Concurrency to System Design*, pages 147–156, 2007.

[17] A. Kondratyev, M. Kishinevsky, and A. Taubin. Synthesis method in self-timed design. Decompositional approach. In *Proc. ICVC'93*, pages 324–327, 1993.

[18] Agnes Madalinski, Alex Bystrov, Victor Khomenko, and Alex Yakovlev. Visualization and resolution of coding conflicts in asynchronous circuit design. In *DATE '03: Proceedings of the Conference on Design, Automation and Test in Europe*, Washington, DC, USA, 2003.

[19] Steven Mark Nowick. *Automatic Synthesis of Burst-mode Asynchronous Controllers*. PhD thesis, Computer Systems Laboratory Departments of Electrical Engineering and Computer Science at the Stanford University, 1993.

[20] Mark Schaefer and Walter Vogler. Component refinement and CSC solving for STG decomposition. In Vladimiro Sassone, editor, *Foundations of Software Science and Computational Structures: 8th International Conference, FOSSACS 2005*, volume 3441 of *LNCS*, pages 348–363. Springer-Verlag, 2005.

[21] 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.

[22] Stephen H. Unger. *Asynchronous Sequential Switching Circuits*. Wiley-Interscience, New York, USA, 1969.

[23] Kees van Berkel, Joep Kessels, Marly Roncken, Ronald Saeijs, and Frits Schalij. The VLSI-programming language Tangram and its translation into handshake circuits. In *EURO-DAC '91: Proceedings of the conference on European design automation*, pages 384–389, Los Alamitos, CA, USA, 1991.

[24] Hans van Gageldonk, Kees van Berkel, Ad Peeters, Daniel Baumann, Daniel Gloor, and Gerhard Stegmann. An asynchronous low-power 80c51 microcontroller. In *ASYNC '98: Proceedings of the 4th International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 96–107, Washington, DC, USA, 1998.

[25] W. Vogler and B. Kangsah. Improved decomposition of STGs. In *ACSD '05: Proceedings of the Fifth International Conference on Application of Concurrency to System Design*, pages 244–253, Washington, DC, USA, 2005.

[26] Walter Vogler and Ralf Wollowski. Decomposition in asynchronous circuit design. *Concurrency and Hardware Design — Advances in Petri Nets, Volume 2549 of Lecture Notes in Computer Science / J. Cortadella, A. Yakovlev, G. Rozenberg (Eds.)*, 2549:152–190, November 2002.

[27] 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.

[28] T. Yoneda, A. Matsumoto, M. Kato, and C. Myers. High level synthesis of timed asynchronous circuits. In *Proceedings of the 11th International Symposium on Asynchronous Circuits and Systems*, 2005.

[29] Tomohiro Yoneda, Hiroomi Onda, and Chris Myers. Synthesis of speed independent circuits based on decomposition. In *In ASYNC'04: 10th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 135–145, Los Alamitos, CA, USA, 2004.

[30] K.Y Yun and D.L. Dill. Automatic synthesis of extended burst-mode circuits. II.(automatic synthesis). In *Proceedings of the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 18, pages 118–132, 1999.

[31] K.Y Yun and D.L. Dill. Automatic synthesis of extended burst-mode circuits. I.(specification and hazard-free implementations). In *Proceedings of the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 18, pages 101–117, 1999.

Aktuelle Technische Berichte des Hasso-Plattner-Instituts

Band	ISBN	Titel	Autoren / Redaktion
19	978-3-939469-95-7	**A quantitative evaluation of the enhanced Topic-based Vector Space Model**	Dominic Wist, Ralf Wollowski
18	978-939-469-58-2	**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 / 978-939469-52-0	**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**	Bernhard Gröne, Peter Tabeling, Konrad Hübner
14	3-939469-23-8 / 978-3-939469-23-0	**Aspektorientierte Programmierung – Überblick über Techniken und Werkzeuge**	Janin Jeske, Bastian Brehmer, Falko Menge, Stefan Hüttenrauch, Christian Adam, Benjamin Schüler, Wolfgang Schult, Andreas Rasche, Andreas Polze
13	3-939469-13-0 / 978-3-939469-13-1	**A Virtual Machine Architecture for Creating IT-Security Labs**	Ji Hu, Dirk Cordel, Christoph Meinel
12	3-937786-89-9 / 978-3-937786-89-6	**An e-Librarian Service - Natural Language Interface for an Efficient Semantic Search within Multimedia Resources**	Serge Linckels, Christoph Meinel
11	3-937786-81-3	**Requirements for Service Composition**	Prof. Dr. M. Weske, Dominik Kuropka Harald Meyer
10	3-937786-78-3	**Survey on Service Composition**	Prof. Dr. M. Weske, Dominik Kuropka Harald Meyer
9	3-937786-73-2	**Sichere Ausführung nicht vertrauenswürdiger Programme**	Andreas Polze Johannes Nicolai, Andreas Rasche
8	3-937786-72-4	**Resourcenpartitionierung für Grid-Systeme**	Andreas Polze, Matthias Lendholt, Peter Tröger
7	3-937786-56-2	**Visualizing Design and Spatial Assembly of Interactive CSG**	Prof. Dr. Jürgen Döllner, Florian Kirsch, Marc Nienhaus

ISBN 978-3-940793-02-7
ISSN 1613-5652