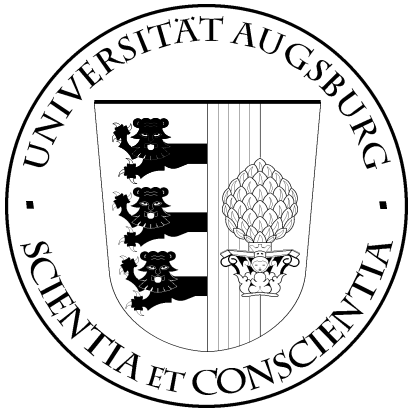
Decomposition in asynchronous circuit design

Walter Vogler, Ralf Wollowski

Angaben zur Veröffentlichung / Publication details:

Vogler, Walter, and Ralf Wollowski. 2002. “Decomposition in asynchronous circuit design.” Augsburg: Universität Augsburg.

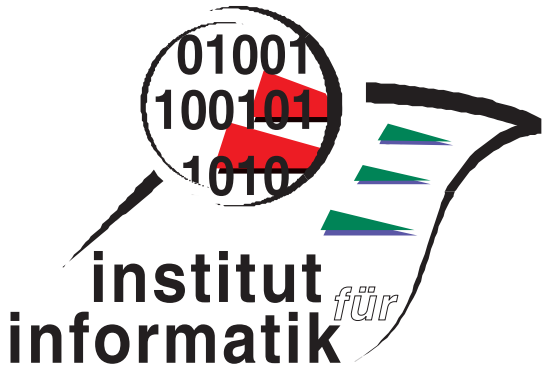
Universität Augsburg



Decomposition in Asynchronous Circuit Design

W. Vogler, R. Wollowski

Report 2002-5 März 2002



Institut für Informatik
D-86135 Augsburg

Decomposition in Asynchronous Circuit Design *

W. Vogler
Institut für Informatik
Universität Augsburg
vogler@informatik.uni-augsburg.de

R. Wollowski
FB Elektro- und Informationstechnik
Universität Kaiserlautern
wollo@rhrk.uni-kl.de

Abstract

Signal Transition Graphs (STGs) are a version of Petri nets for the specification of asynchronous circuit behaviour. It has been suggested to decompose such a specification as a first step; this leads to a modular implementation, which is often more efficient, and it can support circuit synthesis by possibly avoiding state explosion or allowing the use of library elements.

We present a decomposition algorithm and formally prove it correct, where an interesting aspect is the use of a bisimulation with angelic nondeterminism. In contrast to similar approaches in the literature, our algorithm is very generally applicable. We show that transition contraction – the main operation in the algorithm – can be applied with fewer restrictions than known so far. We also prove that deletion of redundant places can be used in the algorithm, which turns out to be very useful in examples.

keywords: concurrent systems, Petri nets, asynchronous circuits, Signal Transition Graphs, decomposition, modular implementation, state explosion

1 Introduction

Signal Transition Graphs (STGs), see e.g. [Wen77, RY85, Chu86], are a version of Petri nets for the specification of asynchronous circuit behaviour, also supported by the tools petrify (e.g. [CKK+97]) and CASCADE [BEW00]. The transitions are labelled with input or output signals;[1] the latter are thought to be controlled by the circuit, the former by its environment. In such a setting, I,O-automata [Lyn96] require that in each state each input can occur, and the same holds for the language theoretic framework of [Dil88]; in STGs though, the occurrence of an input signal in some state might not be specified, which formulates the assumption on the environment not to produce this signal.

Being Petri nets, STGs allow a causality-based specification style, and they give a compact representation of the desired behaviour since they represent concurrency

*This work was partially supported by the DFG-project 'STG-Dekomposition' Vo615/7-1 / Wo814/1-1.

[1] Usually, the labels in STGs are not signals, but rising and falling edges of signals, which for each signal are required to alternate; this is of minor importance in this paper, so we abstract from this distinction.

explicitly. As a first step in the synthesis of a circuit corresponding to a given STG N, one usually constructs the reachability graph, where one might encounter the state explosion problem; i.e. the number r of reachable states (markings) might be too large to be handled. To avoid this, one could try to decompose the STG into components C_i; their reachability graphs can be much smaller since r might be the product of their sizes. Also, a number of smaller circuits is often much easier to synthesize than one large one, and often their parallel composition gives a more efficient implementation than one large undivided circuit. Decomposition is also useful if it allows to split off a library element; in particular for arbiters, it is valuable to avoid the complicated synthesis of such a circuit and use a known one stored in a library.

[Chu87a, Chu87b, KKT93] suggest decomposition methods for STGs, but these approaches can only deal with very restricted net classes. [Chu87a] only decomposes live and safe free choice nets, which cannot model controlled choices or arbitration, and makes further restrictions; e.g. each transition label is allowed only once (which makes the STG deterministic in the sense of language theory), and conflicts can only occur between input signals. The conference version [Chu87b] restricts attention even further to marked graphs, which have no conflicts at all.

The method in [Chu87a, Chu87b] constructs for each output signal s a component C_i that generates this signal; C_i has as inputs all signals that – according to the net structure – may directly cause s. The component is obtained from the STG N by contracting all transitions belonging to the signals that are neither input nor output signals for this component. This contraction is required to be tr-preserving (as defined in [Chu87a]), and it might be necessary to add further signals to the inputs to ensure this.

In [Chu87a, Chu87b], it is stated that the parallel composition of the C_i – i.e. the (modular) *implementation* – has the same language as N; in the restricted setting of [Chu87a, Chu87b], this is the same as having isomorphic reachability graphs. Clearly, this isomorphism is very strict and not a necessary requirement for an implementation to be correct. On the other hand, language equivalence is too weak in general, since it ignores which choices are possible during a run, and in particular it ignores deadlocks; it seems that in general some form of bisimilarity would be more suitable. The formal proofs for the correctness statement in [Chu87a, Chu87b] are very involved.

A similar decomposition method is described in [KKT93]; only marked graphs with only output signals are considered and the treatment is quite informal. In contrast to [Chu87a], a component can generate several output signals and different components can generate the same signal; this gives more flexibility, but the latter feature necessitates additional components for collecting occurrences of the same signal generated by different components.

Further, rather informal, considerations of this decomposition method can be found in [BW93, Wol97]. We finally mention [KGJ96] where fork/join machines are decomposed, which are a restricted form of free choice STGs. In contrast to our setting, the decomposition is already uniquely traced out by the given fork/join structure; correctness only holds under fundamental mode conditions and is not formally proved.

In this paper, we have a fresh look at the decomposition problem. In particular, we will suggest a method where there are no restrictions on the graph-theoretic structure of the given STG N; to some degree we will even deal with arc weights greater 1 and

unsafe nets, which is a bit unusual for STGs, but can be useful as we will demonstrate in an example. There are restrictions on the labelling, e.g. conflicts between input and output signals are not allowed; an STG violating this restriction cannot be turned into a hazard-free asynchronous circuit. STGs are required to be deterministic; but very importantly, we allow several transitions to have the same label.

Our method is based on [Chu87a, Chu87b], but components may generate several output signals. We start from a partition of the output signals and construct for each class of the partition a component C_i; the component has as inputs all signals that – according to the net structure – may directly cause one of its outputs. C_i is obtained from N by contracting stepwise all transitions belonging to the signals that are neither input nor output signals for this component.

We call the contraction of a transition t *secure*, if no other transition takes a token from the preset of t or if no other transition puts a token onto the postset of t. This is a part of being tr-preserving in the sense of [Chu87a, Chu87b], but in contrast to tr-preservation in general, it is easy to check from the local net structure. We suggest to apply mainly secure contractions. Also in our approach, it might be necessary to add further input signals to a component during the stepwise contraction process; we give local graph-theoretic, i.e. easy, conditions when this is the case.

If we regard transition t as internal, i.e. as labelled with the empty word λ, then [Chu87a, Chu87b] shows that the tr-preserving contraction of t preserves the language. This is actually true for all secure contractions, as e.g. already indicated in [And83]. After presenting basic definitions of STGs in Section 2, we have a closer look at contractions in Section 3, also considering bisimilarity and non-secure contractions.

In Section 4, we describe our method in detail. We give a flexible description which allows not only secure contractions to be used but any operation that is admissible in some sense. A benefit of this approach is that e.g. also non-secure contractions turn out to be admissible under certain conditions, e.g. if each transition label occurs only once as in [Chu87a, Chu87b]. This of course depends on our correctness criterion, which has the following important features.

- We ensure that the composition of the C_i is free of what Ebergen calls computation interference [Ebe92], where one component produces an output that is an unspecified input for another; it seems that this problem is ignored in [Chu87a, Chu87b].

- We only consider behaviour where the environment behaves as specified by the original STG N, i.e. the composition of the components might specify additional inputs, but we ignore these and any subsequent behaviour since they cannot occur if the implementation runs in an appropriate environment. The same is done e.g. in [Dil88, Ebe92], so both these features are not new – but new in the context of STG decomposition.

- We achieve both these features with a bisimulation-like correctness definition. Since we restrict ourselves in this paper to the case that N and the C_i are deterministic, bisimilarity actually coincides with language equivalence; but there are several reasons for choosing a bisimulation style: First, for future extensions to nondeterministic STGs, the additional distinctive power of bisimulation will

be important. Second, although one could argue that the explicit treatment of markings in the definition of a bisimulation is not as elegant as the definition of the language, markings have to be treated anyway in order to deal with computation interference. In fact, the treatment of markings and the explicit requirements how input and output signals of one system are matched by the other system, should better clarify the notion of correctness – and this is the third reason. Fourth, the chosen style will be technically useful in our correctness proof.

Interestingly, our proof technique is based on a kind of angelic bisimulation, where internal transition occurrences only serve to find a matching behaviour, but are not required to be matched on their own.

The main contribution of this paper is that – transferring the first and second of these features to the area of STG decomposition – we obtain a more generally applicable decomposition method with a much easier correctness proof compared to [Chu87a, Chu87b]. At the end of Section 4, we additionally show that also deletion of redundant places and – under certain circumstances – non-secure contractions are admissible in our approach. We present two examples for our method in Section 5; further examples as well as a supporting tool are in preparation. Further research topics are discussed in the conclusion in Section 6.

2 Basic Notions of Signal Transition Graphs

In this section, we introduce the kind of Petri nets we study in this paper, some standard behaviour notions, and the operation of parallel composition. For general information on ordinary Petri nets, the reader is referred to e.g. [Pet81, Rei85]. A *Signal Transition Graph* or *STG* is a net that models the desired behaviour of an asynchronous circuit. Its transitions are labelled with signals from some alphabet Σ or with the empty word λ, and we distinguish between input and output signals. A transition labelled with λ represents an internal, unobservable signal, which can be an internal signal between components of a circuit. In this paper, we use λ-labelled transitions only in intermediate phases of our algorithm.

Thus, an *STG* $N = (P, T, W, l, M_N, In, Out)$ is a labelled net consisting of finite disjoint sets P of *places* and T of *transitions*, the *arc weight* $W : P \times T \cup T \times P \to \mathbb{N}_0$, the *labelling* $l : T \to In \cup Out \cup \{\lambda\}$, the *initial marking* $M_N : P \to \mathbb{N}_0$ and the disjoint sets $In \subseteq \Sigma$ and $Out \subseteq \Sigma$ of *input* and *output signals*; $\mathbb{N}_0$ denotes the natural numbers including 0. We usually use a, b, c for input and x, y, z for output signals; if $l(t) \in In$ ($l(t) \in Out$ resp.) then t is an input (an output resp.) transition, drawn as a black (a white resp.) box; if $l(t) = \lambda$, then t is an *internal* transition, drawn as a line or a box with two lines in it. When we introduce an STG N or N_1 etc., then we assume that implicitly this introduces its components P, T, W, ... or P_1, T_1, ... etc.

We say that there is an *arc* from $x \in P \cup T$ to $y \in P \cup T$ if $W(x, y) > 0$. For each $x \in P \cup T$, the *preset* of x is ${}^\bullet x = \{y \mid W(y, x) > 0\}$ and the *postset* of x is $x^\bullet = \{y \mid W(x, y) > 0\}$. If $x \in {}^\bullet y \cap y^\bullet$, then x and y form a *loop*. A *marking* is a function $P \to \mathbb{N}_0$ giving for each place a number of *tokens*. We now define the basic firing rule.

- A transition t is *enabled* under a marking M, denoted by $M[t\rangle$, if $W(.,t) \leq M$. If $M[t\rangle$ and $M' = M + W(t,.) - W(.,t)$, then we denote this by $M[t\rangle M'$ and say that t can *occur* or *fire* under M yielding the follower marking M'.

- This definition of enabling and occurrence can be extended to sequences as usual: a finite sequence $w \in T^*$ of transitions is *enabled* under a marking M, denoted by $M[w\rangle$, and yields the follower marking M' when *occurring*, denoted by $M[w\rangle M'$, if $w = \lambda$ and $M = M'$ or $w = w't$, $M[w'\rangle M''$ and $M''[t\rangle M'$ for some marking M'' and transition t. If w is enabled under the initial marking, then it is called a *firing sequence*.

- We can extend the labelling to sequences of transitions as usual, i.e. $l(t_1 \dots t_n) = l(t_1) \dots l(t_n)$; note that internal signals are automatically deleted in this *image* of a sequence. With this, we lift the enabledness and firing definitions to the level of signals: a sequence v of signals from Σ is *enabled* under a marking M, denoted by $M[v\rangle\rangle$, if there is some transition sequence w with $M[w\rangle$ and $l(w) = v$; $M[v\rangle\rangle M'$ is defined analogously. If $M = M_N$, then v is called a *trace*. The *language* $L(N)$ is the set of all traces. We call two STGs *language equivalent* if they have the same traces.

- A marking M is called *reachable* if $M_N[w\rangle M$ for some $w \in T^*$. The STG is *k-bounded* if $M(p) \leq k$ for all places p and reachable markings M; it is *safe* if it is 1-bounded and *bounded* if it is k-bounded for some k.

Often, STGs are assumed to be safe and to have only arcs with weight 1. In the first place, we are interested in such STGs; but we also deal with bounded STGs with larger arc weights, in particular since they can turn up in our decomposition algorithm. Note that there is no additional problem to synthesise a circuit from such an STG if the reachability graph is used as an intermediate construction [VYCLdM94, Wol97].

The idea of input and output signals is that only the latter are under the control of the circuit modelled by an STG. The STG requires that certain outputs are produced provided certain inputs have occurred, namely those outputs that are enabled under the marking reached by the signal occurrences so far. At the same time, the STG describes assumptions about the environment that controls the input signals: if some input signal is not enabled, the environment is supposed not to produce this input at this stage; if it does, the specified system may show arbitrary behaviour, and it might even malfunction. Inputs and outputs will become really important in Section 4.

In this paper, we deal with specifications that completely specify the desired behaviour in the sense of determinism (except for intermediate stages in our decomposition algorithm): an STG is *deterministic* if it does not have internal transitions and if for each of its reachable markings and each signal s, there is at most one s-labelled transition enabled under the marking. It is useful to distinguish two forms how determinism can be violated.

- Two different transitions t_1 and t_2 are *enabled concurrently* under a marking M if $W(.,t_1) + W(.,t_2) \leq M$, i.e. if there are enough tokens for both transitions together. If both transitions are labelled with the same signal $s \in \Sigma$, then s is

enabled auto-concurrently under M. An STG is *without auto-concurrency*, if no signal is enabled auto-concurrently under any reachable marking.

- Two different transitions t_1 and t_2 are *in conflict* under a marking M if they are not enabled concurrently under M, but $M[t_1\rangle$ and $M[t_2\rangle$. If both transitions are labelled with the same signal $s \in \Sigma$, then s is *in auto-conflict* under M and the STG has a *dynamic auto-conflict*.

- Two different transitions t_1 and t_2 – and also the signals labelling them – are *in structural conflict* if ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$. If both transitions are labelled with the same signal $s \in \Sigma$, then s is *in structural auto-conflict* and the STG *has* such a conflict. If t_1 is an input (or a λ-labelled) and t_2 an output transition, then they form a *structural input/output conflict* (or a *structural λ/output conflict*) and the STG *has* such a conflict.

Clearly, an STG without internal transitions is deterministic if and only if it is without auto-concurrency and without dynamic auto-conflict; the latter is ensured if there are no structural auto-conflicts. Note that internal transitions enabled concurrently or being in conflict do not introduce auto-concurrency or -conflict.

Simulations are a well-known important device for proving language inclusion or equivalence. A *simulation from N_1 to N_2* is a relation $\mathcal{S}$ between markings of N_1 and N_2 such that $(M_{N_1}, M_{N_2}) \in \mathcal{S}$ and for all $(M_1, M_2) \in \mathcal{S}$ and $M_1[t\rangle M_1'$ there is some M_2' with $M_2[l_1(t)\rangle\rangle M_2'$ and $(M_1', M_2') \in \mathcal{S}$. If such a simulation exists, then N_2 can go on simulating all signals of N_1 forever.

Theorem 2.1 *If there exists a simulation from N_1 to N_2, then $L(N_1) \subseteq L(N_2)$.*

Proof: easy induction. □

Often, nets are considered to have the same behaviour if they are language equivalent. But just as often this is not enough: consider the STGs in Figure 1, which all have the language $\{\lambda, \mathit{send}, \mathit{send\ receive}\}$; they model different channels for the communication of one message: the STG on the right can deadlock without the occurrence of any visible signal by firing the λ-transition, i.e. the channel can refuse to accept a message; the middle one can deadlock after *send*, i.e. it just forgets the message it has accepted in the *send*-action; the STG on the left will stop working only after the message was sent and received. This – clearly important – difference is taken into account by the more detailed behaviour equivalence bisimilarity.

A relation $\mathcal{B}$ is a *bisimulation* between N_1 and N_2 if it is a simulation from N_1 to N_2 and $\mathcal{B}^{-1}$ is a simulation from N_2 to N_1. If such a bisimulation exists, we call the STGs *bisimilar*; intuitively, the STGs can work side by side such that in each stage each STG can simulate the signals of the other. This is more than just requiring that there is a simulation from N_1 to N_2 and one from N_2 to N_1: the latter is true for the STGs in Figure 1 although they are not bisimilar. For deterministic STGs, language equivalence and bisimulation coincide.

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

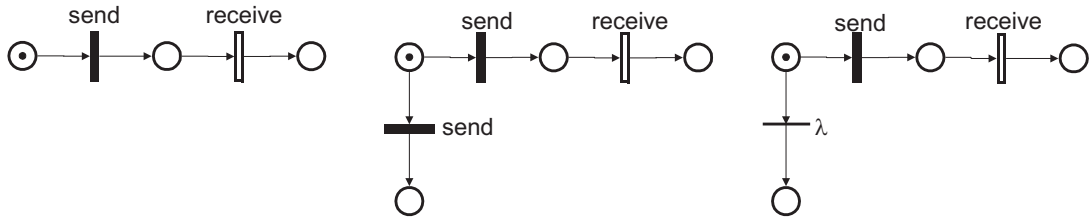


Figure 1

that the composed systems run in parallel synchronizing on common signals. Since a system controls its outputs, we cannot allow a signal to be an output of more than one component; input signals, on the other hand, can be shared. An output signal of 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 [Ebe92] calls computation interference; this is a semantic problem, and we will not consider it here, but later in the definition of correctness.

The parallel composition of STGs N_1 and N_2 is defined if $Out_1 \cap Out_2 = \emptyset$. Then, let $A = (In_1 \cup Out_1) \cap (In_2 \cup Out_2)$ be the set of common signals. In the *parallel composition* $N = N_1 \parallel N_2$, each s-labelled transition t_1 of N_1 is combined with each s-labelled transition t_2 from N_2 if $s \in A$. In the formal definition of parallel composition, $*$ is used as a dummy element, which is formally combined e.g. with those transitions that do not have their label in the synchronization set A. (We assume that $*$ is not a transition or a place of any net.) Thus, N is defined by

$$\begin{aligned}
P &= P_1 \times \{*\} \cup \{*\} \times P_2 \\
T &= \{(t_1, t_2) \mid t_1 \in T_1, t_2 \in T_2, l_1(t_1) = l_2(t_2) \in A\} \\
&\quad \cup \{(t_1, *) \mid t_1 \in T_1, l_1(t_1) \notin A\} \\
&\quad \cup \{(*, t_2) \mid t_2 \in T_2, l_2(t_2) \notin A\}
\end{aligned}$$

$$W((p_1,p_2),(t_1,t_2)) = \begin{cases} W_1(p_1,t_1) & \text{if } p_1 \in P_1,\ t_1 \in T_1 \\ \text{or} & \\ W_2(p_2,t_2) & \text{if } p_2 \in P_2,\ t_2 \in T_2 \end{cases}$$

$$W((t_1,t_2),(p_1,p_2)) = \begin{cases} W_1(t_1,p_1) & \text{if } p_1 \in P_1,\ t_1 \in T_1 \\ \text{or} & \\ W_2(t_2,p_2) & \text{if } p_2 \in P_2,\ t_2 \in T_2 \end{cases}$$

$$l((t_1,t_2)) = \begin{cases} l_1(t_1) & \text{if } t_1 \in T_1 \\ l_2(t_2) & \text{if } t_2 \in T_2 \end{cases}$$

$$M_N = M_{N_1} \dot{\cup} M_{N_2}, \text{ i.e. } M_N((p_1,p_2)) = \begin{cases} M_{N_1}(p_1) & \text{if } p_1 \in P_1 \\ M_{N_2}(p_2) & \text{if } p_2 \in P_2 \end{cases}$$

$$\begin{aligned}
In &= (In_1 \cup In_2) - (Out_1 \cup Out_2) \\
Out &= Out_1 \cup Out_2
\end{aligned}$$

Clearly, one can consider the place set of the composition as the disjoint union of the place sets of the components; therefore, we can consider markings of the composition (regarded as multisets) as the disjoint union of markings of the components;

the latter makes clear what we mean by the restriction $M|_{P_i}$ of a marking M of the composition.

We will denote a marking $M_1 \dot\cup M_2$ of the composition also by (M_1, M_2). By definition of $\|$, the firing $(M_1, M_2)[(t_1, t_2)\rangle(M_1', M_2')$ of N corresponds to the firings $M_i[t_i\rangle M_i'$ in N_i, $i = 1, 2$; here, the firing of $*$ means that the empty transition sequence fires. Therefore, all reachable markings of N have the form (M_1, M_2), where M_i is a reachable marking of N_i, $i = 1, 2$.

If the components do not have internal transitions, then also their composition has none. To see that N is deterministic if N_1 and N_2 are, consider different transitions (t_1, t_2) and (t_1', t_2') with the same label that are enabled under the reachable marking (M_1, M_2). The transitions differ in at least one component, say the first, and since it cannot be the case that t_1 is a transition while $t_1' = *$ (then we would have $l((t_1, t_2)) \in In_1 \cup Out_1$ but $l((t_1', t_2')) \notin In_1 \cup Out_1$), t_1 and t_1' are different transitions with the same label enabled under the reachable marking M_1, which contradicts that N_1 is deterministic. But note that N might have structural auto-conflicts even if none of the N_i has.

It should be clear that, up to isomorphism, composition is associative and commutative. Therefore, we can define the parallel composition of a 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. We will also denote the markings of such a composition by $(M_1, \ldots, M_n)$ if M_i is a marking of C_i for $i \in I = \{1, ..., n\}$.

3 Transition Contraction

We now introduce transition contraction, which will be most important in our decomposition procedure.

Definition 3.1 Let N be an STG and $t \in T$ with $W(., t), W(t, .) \in \{0, 1\}$, ${}^\bullet t \cap t^\bullet = \emptyset$ and $l(t) = \lambda$. We define the *t-contraction* $\overline{N}$ of N by

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

In this definition, we assume $W(*, t_1) = W(t_1, *) = M_N(*) = 0$. We say that the markings M of N and $\overline{M}$ of $\overline{N}$ satisfy the *marking equality* if for all $(p, p') \in \overline{P}$

$$\overline{M}((p, p')) = M(p) + M(p').$$

□

The first requirement about the arc weights of t is presumably not so essential for our results, compare [And83], hence it is more a convenience. It is motivated by the usual, though not necessary, assumption that STGs are safe and have no arc weights greater 1. Note that a contraction might destroy these properties such that a subsequent contraction of a λ-transition is not possible. For the moment, we only remark that comparable approaches would not even allow such 'destructive' contractions; we will consider generalizations in the future. The second requirement about the absence of loops seems difficult to avoid.

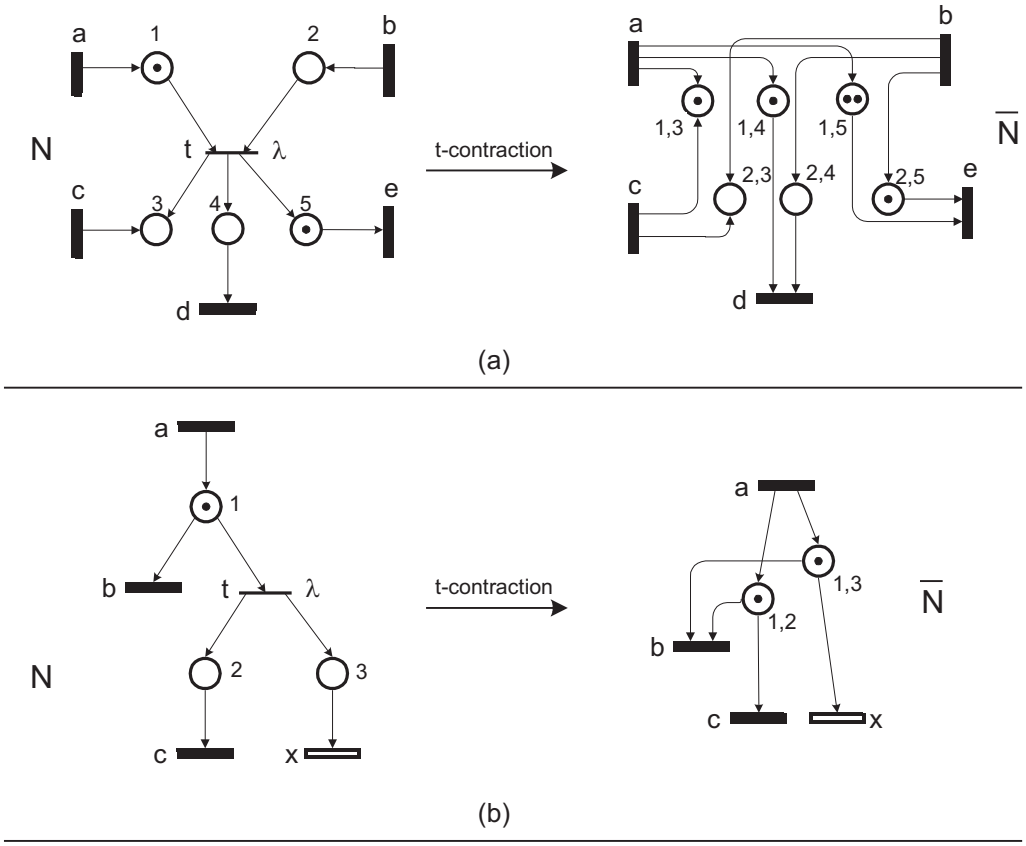


Figure 2

Figure 2 (a) shows a part of a net and the result when the internal transition is contracted. In many cases, the preset or the postset of the contracted transition has only one element, and then the result of the contraction looks easier as e.g. in Figure 2 (b).

We make the following easy observation:

Lemma 3.2 *If $t \in T_1$ satisfies the requirements of Definition 3.1 and $N_1 \parallel N_2$ is defined, then $(t, *)$ satisfies the requirements of 3.1 in $N_1 \parallel N_2$ and $\overline{N_1 \parallel N_2} = \overline{N_1} \parallel N_2$ (up to isomorphism), i.e. contraction and parallel composition commute.*

For the rest of this section, we fix an STG N with a transition t satisfying the requirements of Definition 3.1 and denote its t-contraction by $\overline{N}$. Furthermore, we define the relation $\mathcal{B}$ as $\{(M, \overline{M}) \mid M \text{ and } \overline{M} \text{ satisfy the marking equality}\}$.

The first theorem will show that contraction preserves behaviour in a weak sense; then, we will show that under additional assumptions, it preserves behaviour in a stronger sense. We begin with a lemma.

Lemma 3.3 *Let M and $\overline{M}$ be markings of N and $\overline{N}$ satisfying the marking equality.*

1. *$M[t\rangle M'$ implies $\overline{M}((p,p')) = M'(p) + M'(p')$.*
2. *If $M[t_1\rangle M'$ for $t_1 \neq t$, then $\overline{M}[t_1\rangle\overline{M'}$, and M' and $\overline{M'}$ satisfy the marking equality.*
3. *$M[v\rangle\rangle M_1$ implies $\overline{M}[v\rangle\rangle\overline{M}_1$ such that also M_1 and $\overline{M_1}$ satisfy the marking equality.*

Proof:

1. t removes a token from p and adds a token to p', hence $M(p)+M(p') = M'(p)+M'(p')$. This argument fails in general if ${}^\bullet t \cap t^\bullet \neq \emptyset$.
2. $W((p,p'),t_1) = W(p,t_1) + W(p',t_1) \leq M(p) + M(p') = \overline{M}((p,p'))$; hence, $\overline{M}[t_1\rangle\overline{M'}$ for some $\overline{M'}$.
   $\overline{M'}((p,p')) = \overline{M}((p,p'))-\overline{W}((p,p'),t_1)+\overline{W}(t_1,(p,p'))$; according to the marking equality and the definition of $\overline{W}$, the latter is $M(p)-W(p,t_1)+W(t_1,p)+M(p')-W(p',t_1)+W(t_1,p') = M'(p)+M'(p')$.
3. Apply the first two parts inductively to show that $M[w\rangle M_1$ implies $\overline{M}[w'\rangle\overline{M}_1$ such that also M_1 and $\overline{M_1}$ satisfy the marking equality, where w' is w after deleting all occurrences of t; then we are done, since $l(t) = \lambda$.

□

Theorem 3.4 *$\mathcal{B}$ is a simulation from N to $\overline{N}$; in particular, $L(N) \subseteq L(\overline{N})$.*

Proof: This follows from the last part of Lemma 3.3, since the initial markings satisfy the marking equality. □

The next two results show that under additional assumption the t-contraction is bisimilar or at least language-equivalent.

Theorem 3.5 *Assume that $({}^\bullet t)^\bullet \subseteq \{t\}$. Then:*

1. *$\mathcal{B}$ is a bisimulation from N to $\overline{N}$.*
2. *If t_1 and t_2 with $t_1 \neq t_2$ are concurrently enabled under a reachable marking $\overline{M}$ of $\overline{N}$, then there is a reachable marking M' of N that satisfies the marking equality with $\overline{M}$ and also enables t_1 and t_2 concurrently.*
3. *The contraction preserves boundedness and freedom from auto-concurrency.*

Proof: First observe that the third part follows from the other two, hence we concentrate on these. In particular, each reachable marking of $\overline{N}$ satisfies the marking equality with some reachable marking of N; hence, if N is k-bounded, then $\overline{N}$ is $2k$-bounded.

If ${}^\bullet t = \emptyset$, then $\overline{P} = P - t^\bullet$ and the marking $\overline{M}$ satisfying the marking equality with some marking M of N is given by $\overline{M} = M|_{\overline{P}}$. Since t can put arbitrarily many tokens onto $t^\bullet$, both claims are quite easy to see; hence, let ${}^\bullet t \neq \emptyset$.

$\mathcal{B}$ is a simulation by Lemma 3.3, hence we only have to show that $\mathcal{B}^{-1}$ is a simulation, too. Let $(M, \overline{M}) \in \mathcal{B}$ and $\overline{M}[t_1\rangle M_1$. Firing t under M as often as possible gives a marking M' that still satisfies the marking equality with $\overline{M}$ by Lemma 3.3 and $M'(p_0) = 0$ for some $p_0 \in {}^\bullet t$. We check the places $p \in P$ to see that M' enables t_1:

$p \notin {}^\bullet t \cup t^\bullet$: $W(p,t_1) = \overline{W}((p,*),t_1) \leq \overline{M}((p,*)) = M'(p)$
$p \in {}^\bullet t$: $W(p,t_1) = 0$ by assumption
$p \in t^\bullet$: $W(p,t_1) = W(p_0,t_1) + W(p,t_1) = \overline{W}((p_0,p),t_1) \leq \overline{M}((p_0,p))$
$= M'(p_0) + M'(p) = M'(p)$

Now we have $M'[t_1\rangle M_1'$ for some M_1' and $(M_1', M_1) \in \mathcal{B}$ by Lemma 3.3. Since a sequence of t's followed by t_1 has the same label as just t_1, we have shown the first part.

For the second part, one finds M' as above, and also the check that M' enables t_1 and t_2 concurrently is very similar; e.g. for $p \notin {}^\bullet t \cup t^\bullet$, we have:
$W(p,t_1) + W(p,t_2) = \overline{W}((p,*),t_1) + \overline{W}((p,*),t_2) \leq \overline{M}((p,*)) = M'(p)$ □

Theorem 3.6 *Assume that* ${}^\bullet(t^\bullet) = \{t\}$*; in particular,* $t^\bullet \neq \emptyset$*. Further, assume that* $\exists p_0 \in t^\bullet : M_N(p_0) = 0$*; then:*

1. $\{(\overline{M}, M) \in \mathcal{B}^{-1} \mid \exists q_0 \in t^\bullet : M(q_0) = 0\}$ *is a simulation from* $\overline{N}$ *to* N*;* N *and* $\overline{N}$ *are language equivalent.*

2. *If* t_1 *and* t_2 *with* $t_1 \neq t_2$ *are concurrently enabled under a reachable marking* $\overline{M}$ *of* $\overline{N}$*, then there is a reachable marking* M' *of* N *that satisfies the marking equality with* $\overline{M}$ *and also enables* t_1 *and* t_2 *concurrently.*

3. *The contraction preserves boundedness and freedom from auto-concurrency.*

Proof: First observe that the third part follows from the other two, hence we concentrate on these. In particular, each reachable marking of $\overline{N}$ satisfies the marking equality with some reachable marking of N; hence, if N is k-bounded, then $\overline{N}$ is $2k$-bounded.

To show the first part, observe that the initial markings are related by hypothesis. Now assume $(\overline{M}, M)$ is in the given relation, $q_0 \in t^\bullet$ with $M(q_0) = 0$, and $\overline{M}[t_1\rangle M_1$.

We choose $p_1 \in t^\bullet$ such that $m = W(p_1,t_1) - M(p_1)$ is maximal; due to q_0, m is not negative. We check that t can fire m times under M: for all $p \in {}^\bullet t$, we have $M(p) + M(p_1) = \overline{M}((p,p_1)) \geq \overline{W}((p,p_1),t_1) = W(p,t_1) + W(p_1,t_1)$, and thus

$M(p) \geq W(p_1,t_1) - M(p_1) + W(p,t_1) \geq m$. Firing t under M m times gives a marking M', which satisfies the marking equivality with $\overline{M}$ by Lemma 3.3. By choice of p_1, we have: $(*)$ $M'(p_1) = W(p_1,t_1)$ and $M'(p) \geq W(p,t_1)$ for all $p \in t^\bullet$.

We check that t_1 is enabled under M' by considering all places p:

$p \notin {}^\bullet t \cup t^\bullet$: $W(p,t_1) = \overline{W}((p,*),t_1) \leq \overline{M}((p,*)) = M'(p)$
$p \in t^\bullet$: see $(*)$
$p \in {}^\bullet t$: $W(p,t_1) + W(p_1,t_1) = \overline{W}((p,p_1),t_1) \leq \overline{M}((p,p_1)) = M'(p) + M'(p_1)$
$= M'(p) + W(p_1,t_1)$ by $(*)$, hence $W(p,t_1) \leq M'(p)$

Now we have $M'[t_1\rangle M'_1$ for some M'_1. Since $W(t_1,p_1) = 0$ by hypothesis, we have $M'_1(p_1) = 0$ by $(*)$. Therefore (M_1, M'_1) is in the given relation by Lemma 3.3. As above, since a sequence of t's followed by t_1 has the same label as just t_1, we have shown the first claim.

Language equivalence follows since, together with Theorem 3.4, we have simulations in both directions.

The second part can be shown in a similar way. If $\overline{M}$ is reachable, it is related to some reachable M by the simulation of the first part; let $q_0 \in t^\bullet$ with $M(q_0) = 0$.

We construct M' as above and check that M' enables t_1 and t_2 concurrently as above by adding in the above argument to every atomic term containing t_1 an analogous term containing t_2. E.g. we choose $p_1 \in t^\bullet$ such that $m = W(p_1,t_1) + W(p_1,t_2) - M(p_1)$ is maximal; due to q_0, m is not negative. □

If the preconditions of Definition 3.1 and Theorem 3.5 or 3.6 are satisfied, then we call the contraction of t *secure*.

4 Decomposing a Signal Transition Graph

For this section, we assume that we are given a fixed STG N as a specification of some desired behaviour. Our aim is to decompose it into a collection of components $(C_i)_{i\in I}$ that together implement the specified behaviour; in particular, this should help to avoid the state explosion problem, and therefore we have to avoid the complete state exploration for N.

In particular in the area of circuit design, it seems most often to be the case that, if an in- or output is specified, then its effects are specified without any choices; therefore, we assume that N is deterministic. The specifier is responsible for guaranteeing determinism, i.e. that N has no internal transitions and is without auto-concurrency and without dynamic auto-conflicts; in many cases, the latter two requirements can be proven by place invariants or N might even be free of structural auto-conflicts, such that determinism can be checked without state exploration. In this paper, we will concentrate on the construction of components that are also deterministic.

We further assume that N is free of input/output conflicts; these are very hard to implement, since the input, which is under the control of the environment, might occur at roughly the same time as the output, which is under the control of the system, and can therefore not prevent the output as specified; technically, this may even lead to malfunction.

In applications, N will be bounded and most often even safe; but our results also hold in the general case.

In the first subsection, we define when a collection of components is a correct implementation; in the second, we describe our decomposition algorithm, which uses what we call admissible operations. In the third subsection, we prove that it indeed produces correct components, and in the fourth, we show that certain contractions and place deletions are admissible.

4.1 Correctness definition

Components $(C_i)_{i\in I}$ are correct when their parallel composition 'somehow' matches the behaviour prescribed by the STG N; for this, their composition must of course be defined according to the definition in Section 2, but it must also be well-defined in the sense of being free of computation interference, see [Ebe92]. To compare the behaviour of N and $\|_{i\in I}\ C_i$, one could require language equivalence, as Chu does in [Chu87a, Chu87b], but this is actually too restrictive; one could also take a more liberal refinement notion that is still language based, like Dill does in [Dil88]. Our idea of correctness is very close to the notion of [Dil88, Ebe92], but we will define correctness in a bisimulation style.

We have discussed this choice in the introduction; recall that, in particular, the chosen style will be technically useful in our correctness proof. We now give the formal definition and comment on it afterwards.

Definition 4.1 A collection of deterministic components $(C_i)_{i\in I}$ is a *correct decomposition* or a *correct implementation* of a deterministic STG N, if the parallel composition C of the C_i is defined, $In_C \subseteq In_N$, $Out_C \subseteq Out_N$ and there is a relation $\mathcal{B}$ between the markings of N and those of C with the following properties.

1. $(M_N, M_C) \in \mathcal{B}$
2. For all $(M, M') \in \mathcal{B}$, we have:
   (a) If $a \in In_N$ and $M[a\rangle\rangle M_1$, then either $a \in In_C$ and $M'[a\rangle\rangle M_1'$ and $(M_1, M_1') \in \mathcal{B}$ for some M_1' or $a \notin In_C$ and $(M_1, M') \in \mathcal{B}$.
   (b) If $x \in Out_N$ and $M[x\rangle\rangle M_1$, then $M'[x\rangle\rangle M_1'$ and $(M_1, M_1') \in \mathcal{B}$ for some M_1'.
   (c) If $x \in Out_C$ and $M'[x\rangle\rangle M_1'$, then $M[x\rangle\rangle M_1$ and $(M_1, M_1') \in \mathcal{B}$ for some M_1.
   (d) If $x \in Out_i$ for some $i \in I$ and $M'|_{P_i}[x\rangle\rangle$, then $M'[x\rangle\rangle$. (no *computation interference*)

Here, and whenever we have a collection $(C_i)_{i\in I}$ in the following, P_i stands for P_{C_i}, Out_i for Out_{C_i} etc. □

In this definition, we allow C to have fewer input and output signals than N; the reasons are as follows: There might be some input signals that are not relevant for producing the right outputs; whereas N makes some assumptions on the environment

regarding these inputs, C does not – hence, the environment might produce these signals any time, but they are ignored. Analogously, there might be outputs that actually never have to be produced; if N has such outputs, this presumably indicates an error in the specification since most likely these outputs were intended to occur. In contrast, N might very well contain irrelevant inputs, in particular in the following case: N specifies a system that is itself a component of a larger system, and these inputs are important in other parts; but the designer has not realized that they are not important for this part. Thus, it is useful that our algorithm might detect such so-called *globally irrelevant* inputs.

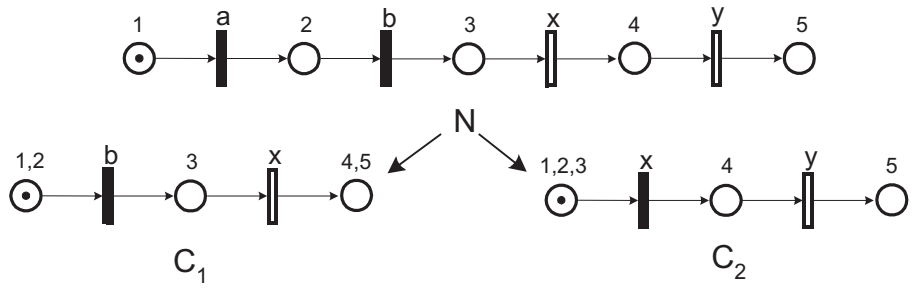


Figure 3

Figure 3 shows a very simple example of an STG N and a decomposition into two components C_1 and C_2 that can be constructed by our algorithm; a is only an input of N but not of any component, but still the latter is a correct implementation as demonstrated by $\mathcal{B} = \{(1,(12,123)),(2,(12,123)),(3,(3,123)),(4,(45,4)),(5,(45,5))\}$. (Here we have identified a marking of N or a component with its single marked place.)

The first three clauses of Part 2 are as usual; the first says that an input allowed by the specification is also allowed by C (or ignored), the second that specified outputs can be made by C, and the third that it does not make others. It should be remarked that in b) the then-part could simply require $M'[x\rangle\rangle$: due to determinism there is only one follower marking in C, which due to c) matches M_1, the unique follower marking in N. More interestingly, it would also be possible to require only $M'[y\rangle\rangle$ for some $y \in Out_C$: again, c) would ensure that this output is specified and the ensuing behaviour matches the specification; but this clause would only say that, in case of *several* specified outputs, at least *one* will be performed. Whether the others are possible or not cannot be observed, since outputs are under the control of the system and, once one output is performed, it cannot be checked whether others had been possible as well; this view is e.g. taken in [Seg93]. Our decomposition algorithm guarantees the stronger form of correctness given in clause b) above.

Remarkably, there is no clause requiring a match for inputs of C. If $M'[a\rangle\rangle M'_1$ for some input a, then either $M[a\rangle\rangle M_1$, in which case the uniquely defined M'_1 and M_1 match by a), or the input is not specified; in the latter case, the environment is not supposed to supply it, such that we can ignore this potential behaviour of C which will never occur in an appropriate environment, i.e. one that satisfies the assumption of the specification.

The usefulness of this feature is demonstrated by the simple example in Figure 4: C_1 and C_2 are an intuitively correct decomposition of N (again obtainable by our

algorithm), since together they answer an input a by x and a following b by y, just as specified. But in $C_1 \parallel C_2$, which is just the disjoint union of C_1 and C_2, b is enabled initially in contrast to N. Note that this implies that N and $C_1 \parallel C_2$ are not language equivalent, as e.g. required in [Chu87a, Chu87b].

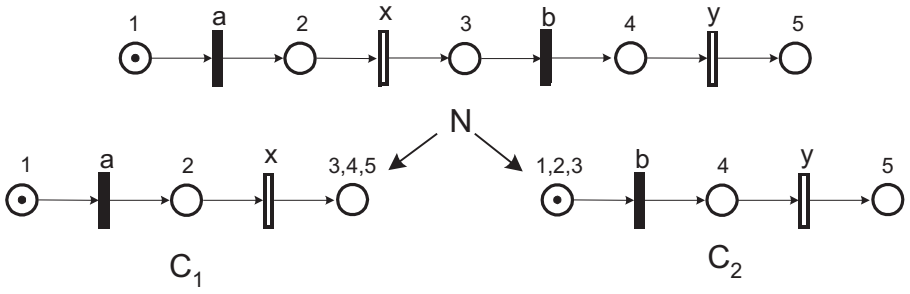


Figure 4

The fourth clause is a requirement that could easily be overlooked: if, in state M', C_i on its own could make an output x that is an input of some C_j, but not allowed there, then there simply exists no x-labelled transition enabled under M' due to the definition of parallel composition; but x is under the control of C_i, so it might certainly produce this output, and we must make sure that it is present in C, i.e. does not lead to a failure of C_j for instance.

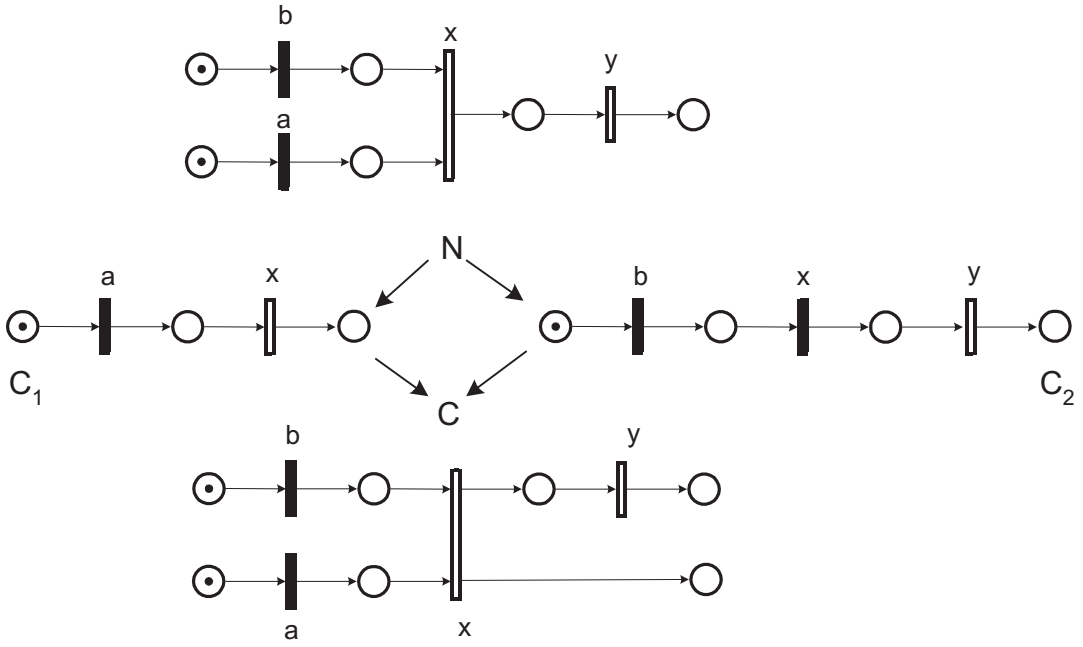


Figure 5

An example is shown in Figure 5. The parallel composition C of C_1 and C_2 looks very much the same as N, and they certainly have the same language - they are even bisimilar. But putting circuits for C_1 and C_2 together, C_1 will possibly produce output x after receiving input a, although x cannot occur in C after a alone; this occurrence

is not specified in N, and therefore C_1 and C_2 should not be a correct decomposition of N – and they are indeed not one due the fourth clause.

Remark: Since N and C are deterministic, there is a smallest $\mathcal{B}$ as above if there is one at all, and its elements correspond one-to-one to the reachable markings of the composition of the *mirror* of N with C; the mirror of N is N with input and output signals exchanged, it is the maximal environment compatible with the system specification N. Such a composition is sometimes considered in the definition of a correct implementation, see e.g. [Ebe92]. □

4.2 The decomposition algorithm

An essential notion of our algorithm is that of an *admissible operation* for the transformation of an STG; in particular, secure contractions will turn out to be admissible. We will define this notion in three steps; for understanding this section, it is enough to know that each admissible operation is a *tc/pd-operation*, i.e. a transition contraction or the deletion of a place (and its incident arcs). We will introduce the further properties as they are needed in the correctness proof; we hope this makes the definition easier to understand, which is tuned to the proof and rather technical.

To initialize the algorithm, one has to choose a *feasible partition* of the signals of N, i.e. a family $(In_i, Out_i)_{i\in I}$ for some set I such that the sets Out_i, $i \in I$, are a partition of Out_N and for each $i \in I$ we have $In_i \subseteq In_N \cup Out_N$, and furthermore:

(C1) If output signals x and y of N are in structural conflict, then $x \in Out_i$ implies $y \in Out_i$ for each $i \in I$.

(C2) If there are $t, t' \in T_N$ with $t^\bullet \cap {}^\bullet t' \neq \emptyset$ and $l_N(t') \in Out_i$ for some $i \in I$, then $l_N(t) \in In_i \cup Out_i$. ($l_N(t)$ *gives concession* to $l_N(t')$.)

For a feasible partition, the *initial decomposition* is $(C_i)_{i\in I}$, where $C_i = (P, T, W, l_i, M_N, In_i, Out_i)$ is a copy of N except for the labelling and the signals; $l_i(t) = l(t)$ if $l(t) \in In_i \cup Out_i$ and $l_i(t) = \lambda$ otherwise.

Now we transform the C_i stepwise by applying repeatedly an admissible operation to one C_i until either no λ-labelled transitions are left, in which case a decomposition has been found, or one of the following failures occurs:

- The C_i transformed last has a structural auto-conflict. Then the last operation was a contraction of some t, and one adds $l(t)$ to In_i and starts the algorithm for this i again from the new initial C_i; alternatively, it suffices to undo the last contractions on C_i up to and including the first contraction of a transition labelled $l(t)$, if this saves time.

- There are internal transitions in C_i, but no known admissible operation is applicable. In this case, one adds $l(t)$ to In_i for some internal transition t and restarts as above.

The reason for the backtracking and addition of an input in the first case is as follows. We take the structural auto-conflict as an indication of a dynamic auto-conflict. Such a dynamic auto-conflict shows that t was labelled in N with a signal C_i should better know about in order to decide which of the two equally labelled transitions to fire.

As an example, consider N in Figure 6 and the construction of a component generating x. Taking c as only input is feasible, and contracting the transitions that were y-, b- and a-labelled gives the component shown in the middle; here it is not clear whether x should be generated upon input of c or not. Now we undo the last contraction and add a as input signal; this gives the component on the left, which does what is required. Alternatively, we could have contracted the a-labelled before the b-labelled transition; then we would undo the contraction of the b-labelled transition and obtain the component on the right, which is alright as well.

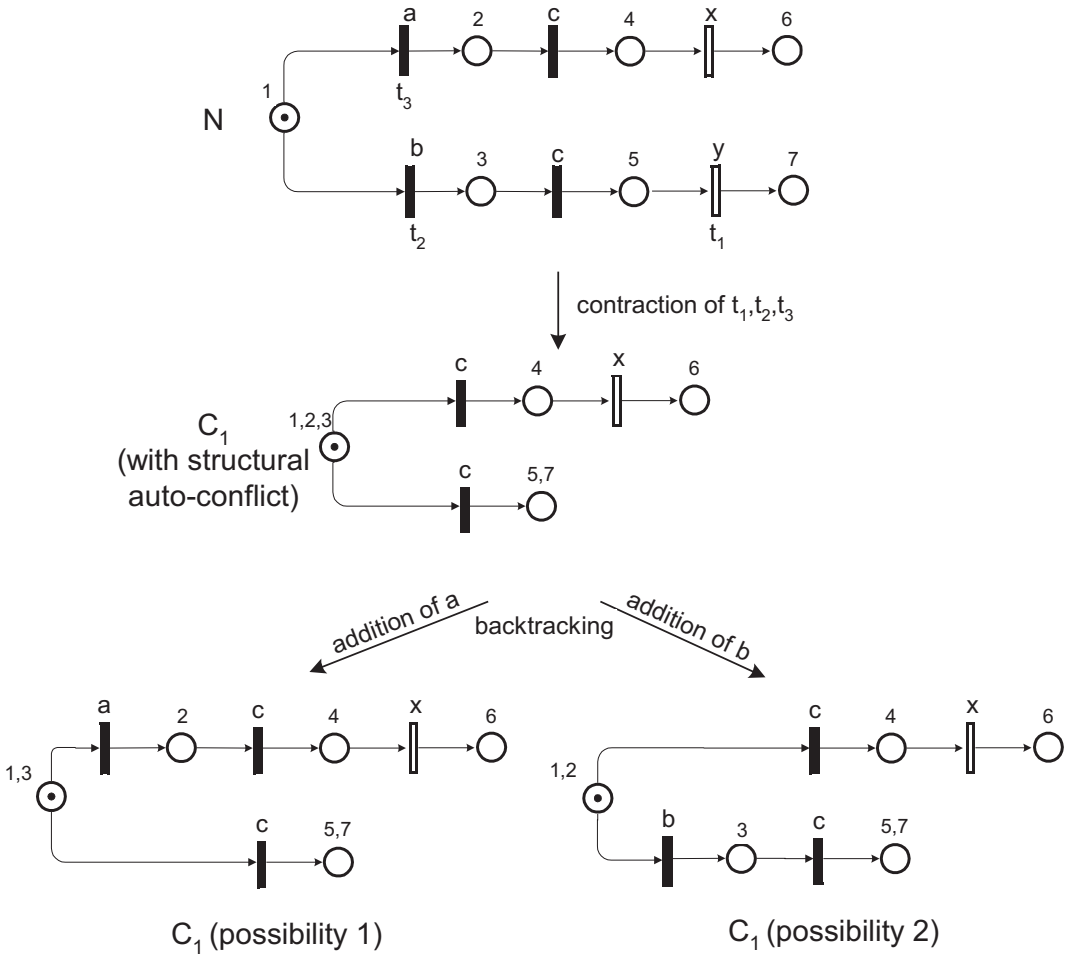


Figure 6

Note that this treatment of structural auto-conflicts might be over-cautious, because it could be that despite the structural auto-conflict there actually is no dynamic auto-conflict. To check this, one would – in general – perform a state exploration, which here we try to avoid. In a computer-aided procedure this is presumably the

place for human intervention which could possibly supply a proof that there actually is no dynamic auto-conflict requiring a restart.

There might even be some more room for improvement: even if C_i has a dynamic auto-conflict, this might occur under a marking that is in fact not reachable if C_i runs in the environment formed by the other components and an 'outer' environment as specified by N.

As indicated, if a failure occurs, the algorithm is restarted; therefore, in each step, for some C_i either the number of internal transitions goes down or it stays the same and the number of places decreases – since each admissible operation is a tc/pd-operation – or the number of signals (which is bounded by $|In_N \cup Out_N|$) increases in case of a restart. Thus, eventually a decomposition will be found in any case. Of course, this decomposition might not be useful since the C_i might be too large – in an extreme case, they could all be equal to N except for the labelling. But, as we have argued in the introduction, our algorithm is at least much more generally applicable than the approaches presented in the literature so far.

4.3 The correctness proof

Each admissible operation will be pre-admissible in the following sense:

Definition 4.2 An operation is *pre-admissible* if it is a tc/pd-operation that when applied to an STG without dynamic auto-conflicts and satisfying (a) and (b) below preserves freeness from auto-concurrency and gives an STG satisfying (a) and (b) again:

(a) There is neither a structural input/output nor a structural λ/output conflict.

(b) If t_2 is an output transition and ${t_1}^\bullet \cap {}^\bullet t_2 \neq \emptyset$, then t_1 is not an internal transition.

□

Two easy consequences of this definition are formulated in the following lemma.

Lemma 4.3 *1. At each stage of the algorithm, each C_i satisfies (a) and (b) of Definition 4.2 and is free of dynamic auto-conflicts.*

2. When the algorithm terminates, the resulting C_i and hence also C are deterministic.

Proof: The initial C_i satisfy (a), since by assumption on N an output transition of C_i can in N only be in structural conflict with another output transition, which will also be an output transition of C_i due to condition (C1) in the definition of a feasible partition. They satisfy (b) by condition (C2) of this definition. Furthermore, they are free of dynamic auto-conflicts since N is.

Hence, the first claim follows from Definition 4.2 and the fact that we restart the algorithm in case a structural auto-conflict turns up.

For the second claim observe that, by assumption and Part 1, there are no internal transitions and no dynamic auto-conflict. Since there is no auto-concurrency initially,

there is also none in the end due to Definition 4.2 and Part 1. The result follows, as already observed in Section 2. □

We now come to the central notion in our correctness proof; it is a variant of a bisimulation with an angelic treatment of internal transitions, and it is needed to describe in what sense the intermediate stages of our algorithm are correct (like a loop invariant). If there is an internal transition in an initial C_i, then this corresponds to a signal of the system that this component does not 'see'; if we assume that by some angelic intervention such a transition, which is internal to C_i and not connected in any way to the outside, fires if the signal occurs, then the C_i together work as intended, and this sort of behaviour is captured with an angelic bisimulation. For the final C_i, an angelic bisimulation guarantees correctness, since there are no internal transitions.

We regard it as an interesting contribution that this kind of bisimulation is useful even though we do not assume any angelic nondeterminism in our correctness definition. In the future, we will study decompositions where components communicate with each other by signals that are internal to the implementation; these internal signals will certainly have to be of a different kind compared to the λ-transitions we study here.

Definition 4.4 A collection of components $(C_i)_{i\in I}$ is an *angelically correct decomposition* or *implementation* of a deterministic STG N, if the parallel composition C of the C_i is defined, $In_C \subseteq In_N$, $Out_C \subseteq Out_N$ and there is an angelic bisimulation relation $\mathcal{B}$ between the markings of N and those of C, i.e. $\mathcal{B}$ satisfies the following properties.

1. $(M_N, M_C) \in \mathcal{B}$
2. For all $(M, M') \in \mathcal{B}$, we have:
   (a) If $a \in In_N$ and $M[a\rangle\rangle M_1$, then either $a \in In_C$ and $M'[a\rangle\rangle M_1'$ and $(M_1, M_1') \in \mathcal{B}$ for some M_1' or $a \notin In_C$ and $M'[\lambda\rangle\rangle M_1'$ and $(M_1, M_1') \in \mathcal{B}$ for some M_1'.
   (b) If $x \in Out_N$ and $M[x\rangle\rangle M_1$, then $M'[x\rangle\rangle M_1'$ and $(M_1, M_1') \in \mathcal{B}$ for some M_1'.
   (c) If $x \in Out_i$ for some $i \in I$ and $M'|_{P_i}[x\rangle\rangle$, then some M_1' and M_1 satisfy $M'[x\rangle\rangle M_1'$, $M[x\rangle\rangle M_1$ and $(M_1, M_1') \in \mathcal{B}$.

□

This definition looks very much like Definition 4.1; the differences are that here $[x\rangle\rangle$ in C might involve additional λ-transitions besides an x-labelled transition, that in 2(a) internal transitions are allowed to match an input of N that is not one of C, and that 2(c) is a combination of 4.1.2(c) and (d) and guarantees a matching only for some M_1' – this is an angelic part of the definition. It is also angelic that we do not require a match for the firing of only internal transitions in C.

We come to the final definition of an admissible operation, which leads to the correctness result.

Definition 4.5 We call a pre-admissible operation applied to some member of a family $(C_i)_{i\in I}$ that satisfies (a) and (b) of Definition 4.2 *admissible* if it preserves angelic correctness w.r.t. N. □

Theorem 4.6 *When the algorithm terminates, the resulting C_i are a correct decomposition of N.*

Proof: The initial decomposition is angelically correct due to $\mathcal{B} = \{(M, (M, \ldots, M) \mid M \text{ is reachable in } N\}$. The defining clauses for an angelic bisimulation are satisfied, since the firing of a transition t in N or in some C_i can be matched by firing all copies of this transition in N and all the C_i.

Admissible operations preserve angelic correctness by definition, since the C_i always satisfy conditions (a) and (b) of Definition 4.2 by Lemma 4.3. Hence, the resulting C_i are an angelically correct decomposition of N.

Furthermore, the C_i and C are deterministic by Lemma 4.3. Therefore, (a), (b) and (c) of Definition 4.4 immediately give (a), (b) and (d) of Definition 4.1. Further, M_1' in (c) of 4.1 is uniquely determined by M' and x by determinism of C, thus it is the M_1' in (c) of 4.4 and therefore also (c) of 4.1 follows. □

4.4 Admissible operations

It remains to present some admissible operations, and to show in particular that secure contractions are admissible.

Lemma 4.7 *1. A contraction applied to an STG satisfying conditions (a) and (b) of Definition 4.2 preserves these properties.*

2. Secure contractions are pre-admissible.

Proof: The second claim follows from the first and Theorems 3.5 and 3.6. Hence, we concentrate on the first claim.

So assume contraction of t is applied to some STG S satisfying conditions (a) and (b). Assume the result violates (a) due to some input or internal transition t_1 and some output transition t_2. Then there are places $pi \in {}^\bullet t_i$ such that $p1 \in {}^\bullet t$ and $p2 \in t^\bullet$ or vice versa; compare Figure 7(a). In the first case, t and t_2 violate (b) in S; in the second case, t and t_2 violate (a) in S.

Finally, assume the result violates (b) due to some internal transition t_1 and some output transition t_2. Then there are places $p1 \in t_1{}^\bullet$ and $p2 \in {}^\bullet t_2$ such that $p1 \in {}^\bullet t$ and $p2 \in t^\bullet$ or vice versa; compare Figure 7(b). In the first case, t and t_2 violate (b) in S; in the second case, t and t_2 violate (a) in S. (Note that in this case the contraction is not secure, but such contractions are considered in the first part, too.) □

The following is the essential lemma for the treatment of contractions; it shows what we need (a) and (b) of Definition 4.2 for in our approach.

Lemma 4.8 *Contraction applied to some member of a family $(C_i)_{i\in I}$ that satisfies (a) and (b) of Definition 4.2 preserves angelic correctness w.r.t. N.*

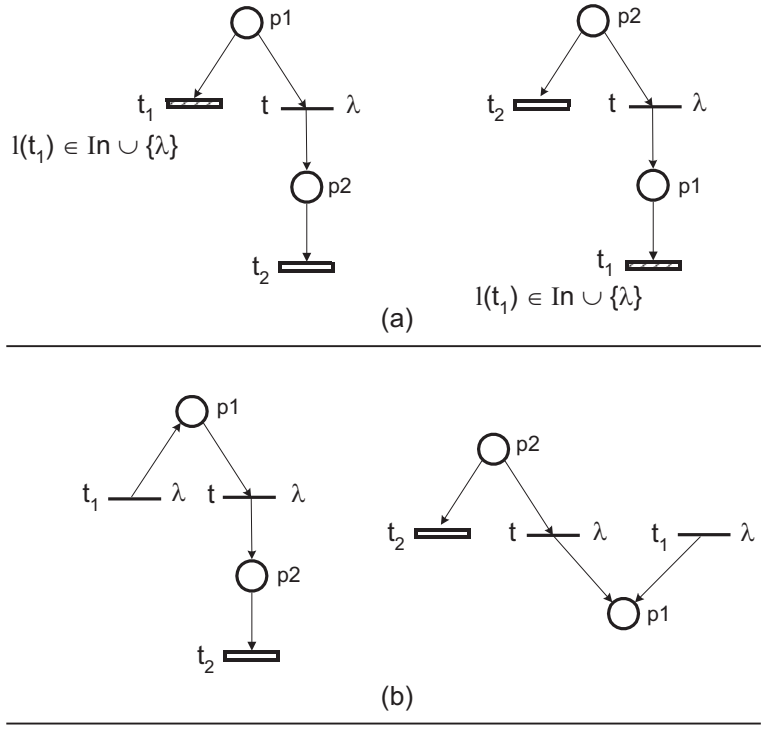


Figure 7

Proof: Assume contraction of t is applied to C_j and results in $\overline{C_j}$, $\overline{C}$ resp.; assume further that $\mathcal{B}$ is an angelic bisimulation for N and $(C_i)_{i\in I}$. We define $\overline{\mathcal{B}}$ as $\{(M, \overline{M}) \mid$ there is $(M, M') \in \mathcal{B}$ such that M' and $\overline{M}$ satisfy the marking equality $\}$. Similarly, we will denote by $\overline{M_1}$ the marking of $\overline{C}$ that satisfies the marking equality with the marking M_1 of C.

We check that $\overline{\mathcal{B}}$ is an angelic bisimulation for N and $(C'_i)_{i\in I}$, where C'_j is $\overline{C_j}$ and $C'_i = C_i$ otherwise. Clearly, the initial markings of N and $\overline{C}$ are related since the initial markings of N and C are.

So let $(M, \overline{M}) \in \overline{\mathcal{B}}$ due to $(M, M') \in \mathcal{B}$.

(a) Let $a \in In_N$ and $M[a\rangle\rangle M_1$. Either $a \in In_C$ and for some M'_1 $M'[a\rangle\rangle M'_1$ and $(M_1, M'_1) \in \mathcal{B}$; then we get $\overline{M}[a\rangle\rangle\overline{M_1}$ by Lemma 3.3.3 and $(M_1, \overline{M_1}) \in \overline{\mathcal{B}}$. Or $a \notin In_C$ and for some M'_1 $M'[\rangle\rangle M'_1$ and $(M_1, M'_1) \in \mathcal{B}$; again we get $\overline{M}[\overline{\rangle}\rangle M_1$ by Lemma 3.3.3 and $(M_1, \overline{M_1}) \in \overline{\mathcal{B}}$.

(b) analogously.

(c) Let $x \in Out_i$ for some $i \in I$ and $\overline{M}|_{P'_i}[x\rangle\rangle$. We have two subcases:

$i \neq j$: $\overline{M}$ and M' coincide on $P'_i = P_i$, hence $M'|_{P_i}[x\rangle\rangle$.

$i = j$: The image-firing of signal x involves an x-labelled transition t_1. Since contractions are pre-admissible, also C'_j satisfies (a) and (b) of Definition 4.2; by (b), only firing internal transitions cannot help to enable t_1, and thus t_1 must already be enabled under $\overline{M}|_{P'_i}$. By (a) and (b) of Definition 4.2 in C_j, no place in ${}^\bullet t_1$ can be in ${}^\bullet t \cup t^\bullet$; therefore, ${}^\bullet t_1$ is the same in C_j and C'_j, $\overline{M}$ and M' coincide on ${}^\bullet t_1$, and $M'|_{P_i}[x\rangle\rangle$.

In either case, some M_1' and M_1 satisfy $M'[x\rangle\rangle M_1'$, $M[x\rangle\rangle M_1$ and $(M_1, M_1') \in \mathcal{B}$, and we are done as in (a). □

The first result of this subsection immediately follows from these lemmata.

Theorem 4.9 *Contractions that preserve freeness of auto-concurrency, i.e. secure contractions in particular, are admissible.*

Thus, secure contractions can be used in the decomposition algorithm, but also others if they do not introduce auto-concurrency. This can possibly be checked without state exploration by using place invariants, or it holds automatically if each transition label occurs only once as assumed in [Chu87a, Chu87b]. We have no practical experience so far how useful such non-secure contractions can be; whenever we wanted to apply such a contraction, we were able to make it secure by first applying a place deletion as described in the following.

A place p of an STG S is (structurally) *redundant* (see e.g. [Ber87]) if there is a set of places Q, a valuation $V : Q \cup \{p\} \to \mathbb{N}_0$ and some $c \in \mathbb{N}_0$ with the following properties:

- $V(p)M_S(p) - \sum_{q\in Q} V(q)M_S(q) = c$
- for all transitions t, $V(p)(W(t,p) - W(p,t)) - \sum_{q\in Q} V(q)(W(t,q) - W(q,t)) \geq 0$
- for all transitions t, $V(p)W(p,t) - \sum_{q\in Q} V(q)W(q,t) \leq c$

If the third item holds (at least) for all output transitions t, we call p *output-redundant.*

The first two items ensure that the valuated token number of p is at least c larger than the valuated token sum on Q for all reachable markings, while the third item says that each transition or at least each output transition needs at most c 'valuated tokens' more from p than from the places in Q.

Clearly, the deletion of a redundant place in S turns each reachable marking of S into one of the transformed STG that enables the same transitions, hence the deletion gives a bisimilar STG. Still, it is not completely obvious that such deletions are admissible operations, since the latter are defined w.r.t. the structure of STGs, which certainly changes, and since such a deletion can increase the concurrency.

Theorem 4.10 *The deletion of an (output-)redundant place is an admissible operation.*

Proof: To check pre-admissibility, take an STG S that is without dynamic auto-conflicts and satisfies properties (a) and (b) of Definition 4.2.

First assume that the deletion introduces auto-concurrency, say the equally labelled transitions t and t' are concurrent under the reachable marking M'. Then, $M' = M|_{P_i - \{p\}}$ for some reachable marking of S, and t and t' are also enabled under M – this holds for any place deletion. Since they cannot be concurrent by assumption, they must be in conflict; this is a contradiction since S does not have dynamic auto-conflicts.

Since the deletion of a place does not add structural conflicts or arcs, it is clear that (a) and (b) of Definition 4.2 are preserved.

To conclude the proof, it follows from the above observation, that an angelic bisimulation for N and C before a deletion can be turned into one for N and C after the deletion by omitting p from each marking of C. This can clearly not disturb 1 and 2(a) and (b) of Definition 4.4, and output-redundancy ensures (c). □

A special case of a redundant place is a *loop-only place*, i.e. a marked place p such that p and t form a a loop with arcs of weight 1 for all $t \in {}^\bullet p \cup p^\bullet$. Another simple case is that of a duplicate: place p is an (extended) *duplicate* of place q, if for all transitions t $W(t,p) = W(t,q)$, $W(p,t) = W(q,t)$ and $M_N(p) \geq M_N(q)$.

5 Examples

We will now demonstrate our algorithm for two realistic examples. In these examples, we will use labels like $a+$ and $a-$, which really denote rising and falling edges of the signal a. To fit them into the approach presented here, one can either regard the $+$ and $-$ as comments and the a-labelled transitions as toggle-transitions that let rising and falling edges of a alternate. Alternatively, we can regard $a+$ and $a-$ as signals in the sense of this paper; the only adaptations needed are the following: we only consider feasible partitions, where $a+ \in In_i$ iff $a- \in In_i$ and the same for Out_i; when the algorithm has to be restarted and some $a+$ is added to In_i, we also add $a-$ and vice versa. This ensures that each STG has either both edges of a signal as outputs or none, and the same for inputs. (The latter view involves a little change of the algorithm, but the correctness proof still works without any changes.)

Our examples will be made available in greater detail on the web; for this, see http://www.eit.uni-kl.de/beister/eng/projects/deco_examples/main_examples.html

The first example demonstrates the possible savings w.r.t. the number of markings and deals with the specification of a FIFO-queue controller, see Figure 8. It has already been studied and decomposed somewhat informally in [BW93]. This STG is particularly simple, since it is a so-called *marked graph*, where the arc weights are 1 and each place has exactly one transition in its preset and one in its postset; thus, there are no conflicts.

Admissible operations preserve these properties; thus, every well-defined contraction is secure. The only possible obstacle for contracting an internal transition is a loop. If we assume that no transition is dead in N, i.e. each can be fired under some reachable marking, then each cycle must contain a token; since a loop generated by the algorithm arises from a cycle in the marked graph N, the loop place is marked and deleting it is an admissible operation. Therefore, in examples like this there will never be the need for a restart.

In our example, the deletion of loop-only places turns out to be necessary indeed; note that place deletions have not been discussed in the literature so far. In fact, it is also very convenient to delete redundant places early; in this example, several duplicates can be found. It should be remarked that – although N and all the constructed components are safe –, one does come across places with more than one token on them. Hence, it is important that we do not restrict ourselves to the treatment of safe nets.

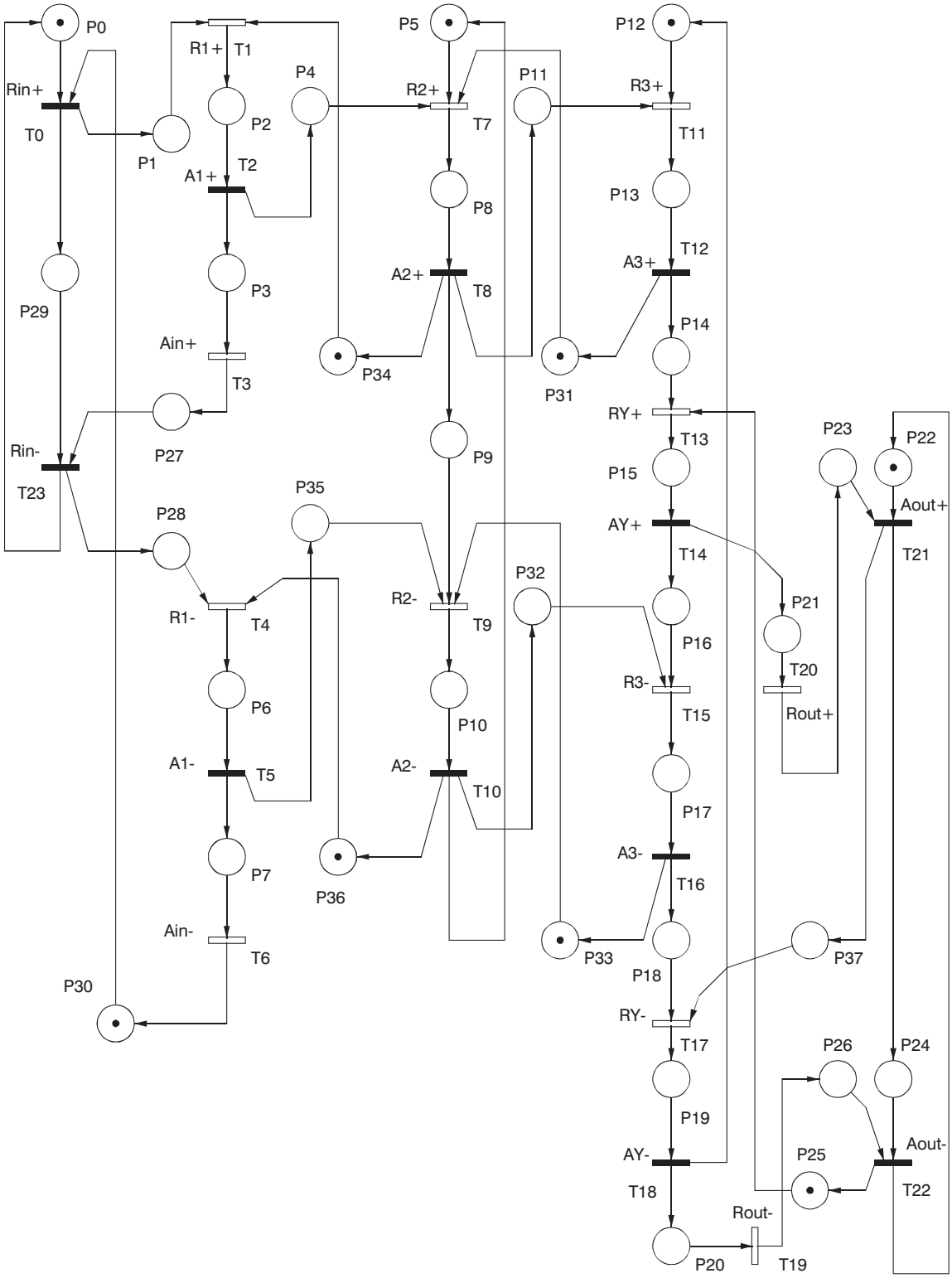

P0
R1+
T1
P5
P12
Rin+
P4
R2+
P11
R3+
T0
P2
T7
T11
P1
A1+
T2
P8
P13
A2+
T8
A3+
T12
P29
P3
P14
Ain+
P34
P31
T3
RY+
T13
P27
P9
P23
P22
Rin-
P15
T23
P35
AY+
Aout+
P28
T14
T21
P32
R2-
P21
R1-
T4
T9
P16
T20
P6
R3-
Rout+
P10
T15
A1-
T5
A2-
T10
P17
P7
A3-
P36
T16
Ain-
P33
P37
T6
P18
P30
RY-
T17
P26
P24
P19
AY-
P25
Aout-
T18
T22
Rout-
P20
T19

Figure 8

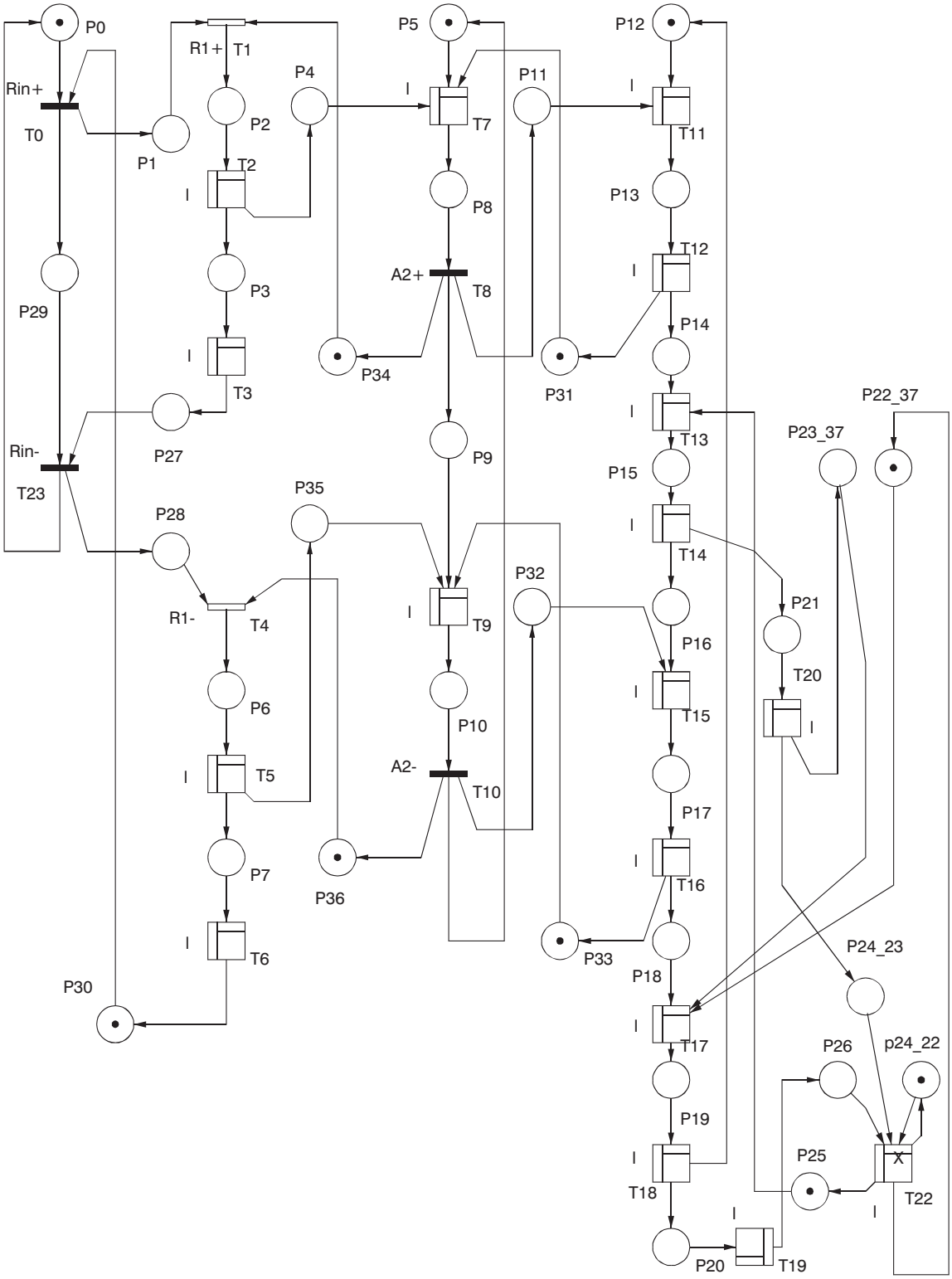

P0
Rin+
T0
P1
R1+
T1
P2
T2
P3
T3
P27
Rin-
T23
P29
P4
P5
T7
P8
A2+
T8
P34
P9
P11
P31
P12
T11
P13
T12
P14
T13
P15
T14
P16
T15
P17
T16
P18
T17
P19
T18
P20
T19
P22_37
P23_37
P21
T20
P24_23
P26
p24_22
P25
T22
P28
R1-
T4
P6
T5
P7
T6
P30
P35
P36
P32
T9
P10
A2-
T10
P33

Figure 9

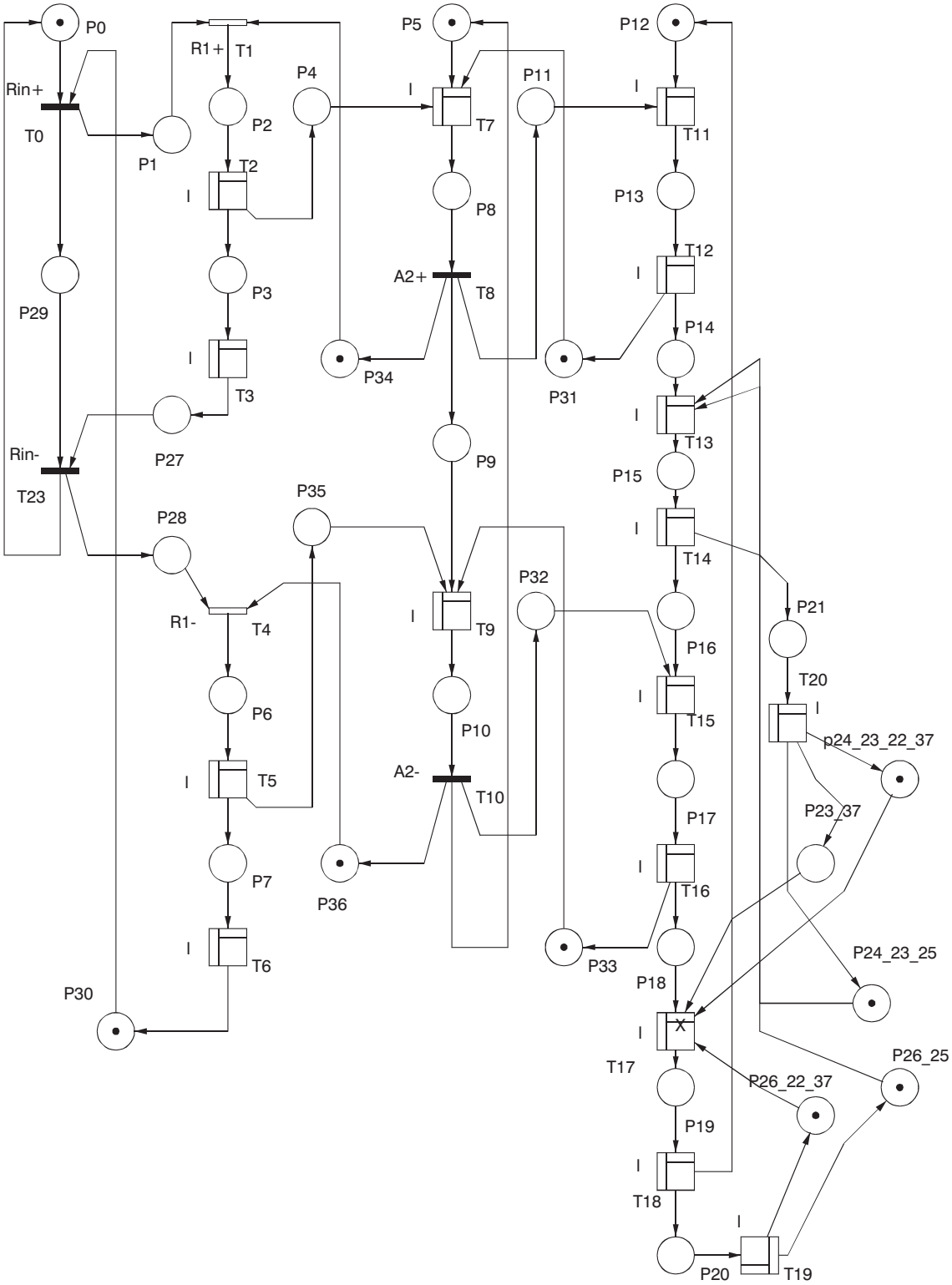

P0
R1+
T1
P5
P12
Rin+
T0
P1
P2
P4
P11
T7
T11
T2
P8
P13
P29
P3
A2+
T8
T12
P14
P34
P31
T3
P27
P9
T13
Rin-
T23
P15
P35
P28
T14
R1-
T4
T9
P32
P21
P16
T20
P6
T15
P10
p24_23_22_37
T5
A2-
T10
P17
P23_37
P7
P36
T16
P24_23_25
T6
P33
P18
P30
X
T17
P26_25
P26_22_37
P19
T18
P20
T19

Figure 10

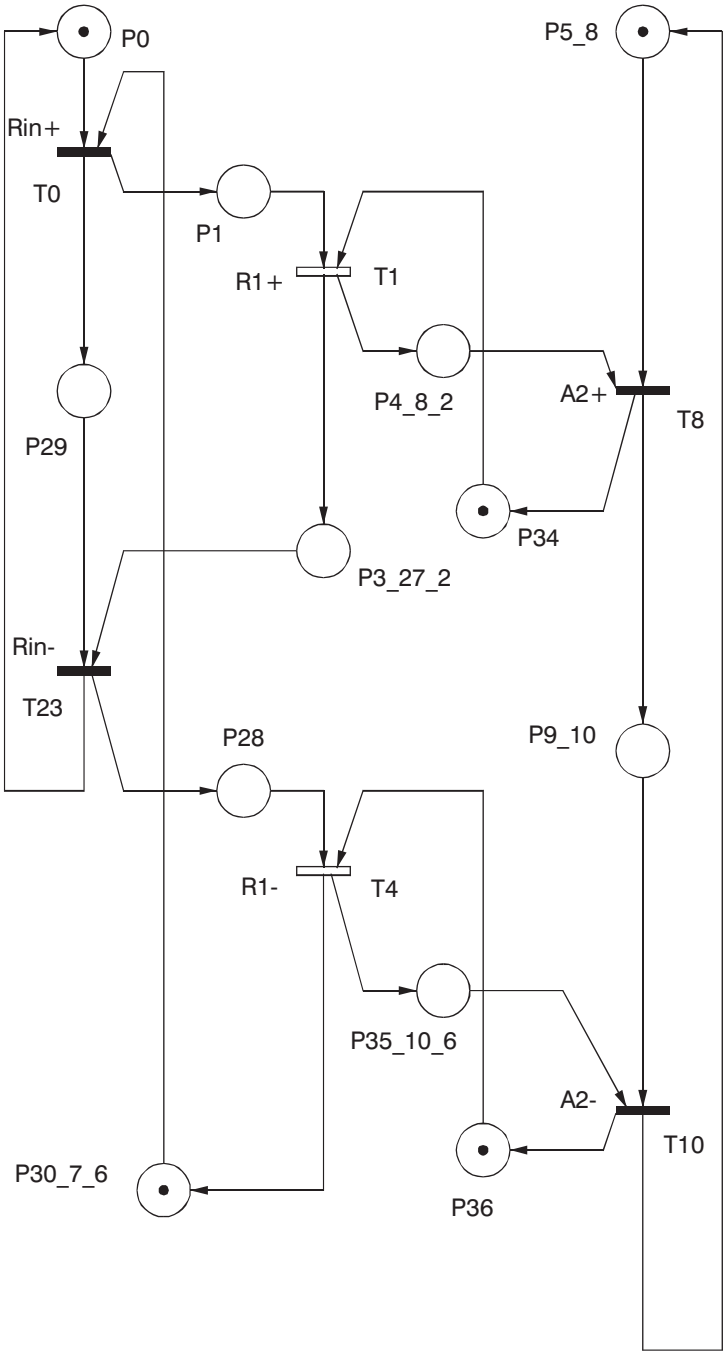

P0
P5_8
Rin+
T0
P1
R1+
T1
P4_8_2
A2+
T8
P29
P34
P3_27_2
Rin-
T23
P28
P9_10
R1-
T4
P35_10_6
A2-
T10
P30_7_6
P36

Figure 11

A component that generates $R1+$ and $R1-$ needs inputs $Rin+$, $Rin-$, $A2+$ and $A2-$. Making all other transitions internal and contracting $T21$ gives the component in Figure 9, which already has a loop-only place (P24_22). Deleting this place and contracting $T22$ gives the component in Figure 10; here, place $P24\_23\_22\_37$ is a duplicate of $P23\_37$. The final result for this component is shown in Figure 11; it has 12 reachable markings.

If we generate a component for each of the output signals (i.e. for each pair of edges of such a signal), we get a decomposition into six components, which have 12 (R1), 26 (R2), 12 (R3), 4 (Ain), 8 (RY), and 4 (Rout) reachable markings, while the original N has 832. Figure 12 shows the resulting modular controller structure.

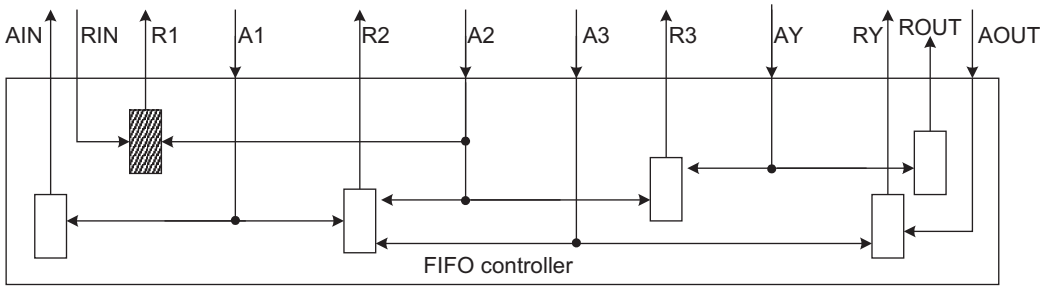


Figure 12

The second example demonstrates that our algorithm can deal with arbitration, and it gives a case where one finds a well known circuit specification as component in a correct decomposition. Figure 13 shows the STG specification of a low latency arbiter known from literature [YKKL94]. Upon a request $R1+$ or $R2+$, the circuit arbitrates with $A1+$ or $A2+$, i.e. there is a critical input race between $R1+$ and $R2+$ [WB00]. In parallel to the arbitration, the circuit requests a resource itself with $R+$. When it gets the resource ($G+$), it grants it depending on the arbitration with $G1+$ or $G2+$. When the resource is returned with $R1-$ or $R2-$, it returns the resource and all signals are returned to the original level. (Note that the place $P4$ is not safe.)

Race behaviour cannot be correctly implemented by a pure logic circuit; additional analogue circuitry is needed to avoid anomalous behaviour like metastability. Present day STG-based tools, therefore, cannot handle such behaviour. The problem is usually solved by using specific library elements, in particular a so-called two-way mutual exclusion (ME) element (e.g. [YKKL94]). This solution burdens the designer with the task to split off an ME-element; our decomposition method can give support here – with results that we have proven to be correct – and we will demonstrate this now. A component for the generation of the $Ai*$, $i = 1, 2$ and $* \in \{+, -\}$, must have the $Ri*$ as inputs according to the definition of a feasible partition. The corresponding initial C_1 is shown in Figure 14.

The secure contractions of $T12$, $T13$, $T6$ and $T7$ give the STG in Figure 15, and the further secure contractions of $T4$ and $T5$ give the STG in Figure 16.

The remaining internal transitions do not allow a secure contraction; but place $P18\_14$ is redundant due to $Q = \{P11, P12, P23\_15, P24\_16\}$ with $V \equiv 1$. After deletion of this place, we can perform the secure contractions of $T14$ and $T15$, which gives the STG in Figure 17. The places in 'the middle column' except for $P19$ can be

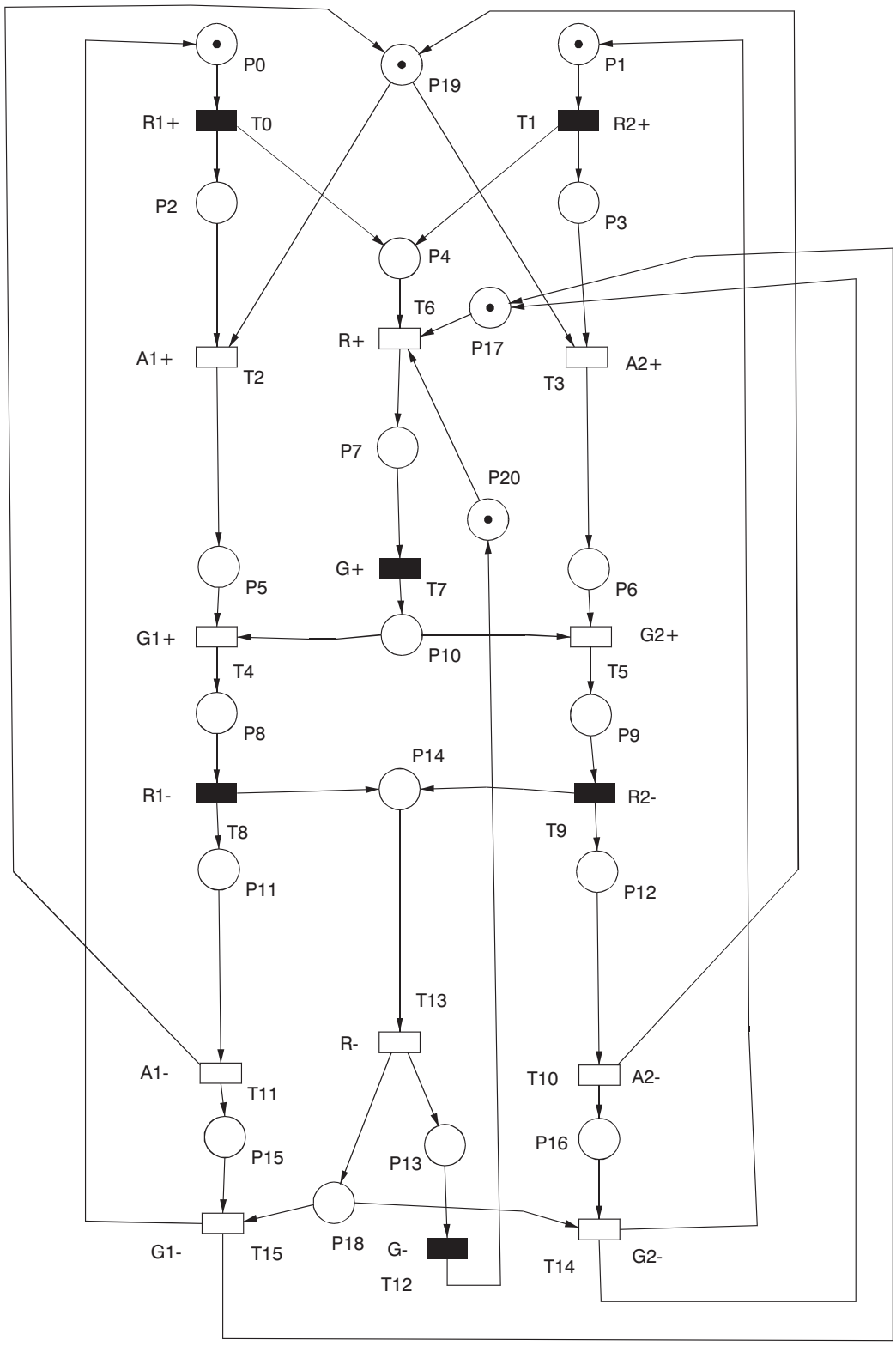

P0
P19
P1
R1+
T0
T1
R2+
P2
P3
P4
T6
R+
P17
A1+
T2
T3
A2+
P7
P20
G+
T7
P5
P6
G1+
P10
G2+
T4
T5
P8
P9
P14
R1-
R2-
T8
T9
P11
P12
T13
R-
A1-
T11
T10
A2-
P15
P13
P16
P18
G-
T12
G1-
T15
T14
G2-

Figure 13

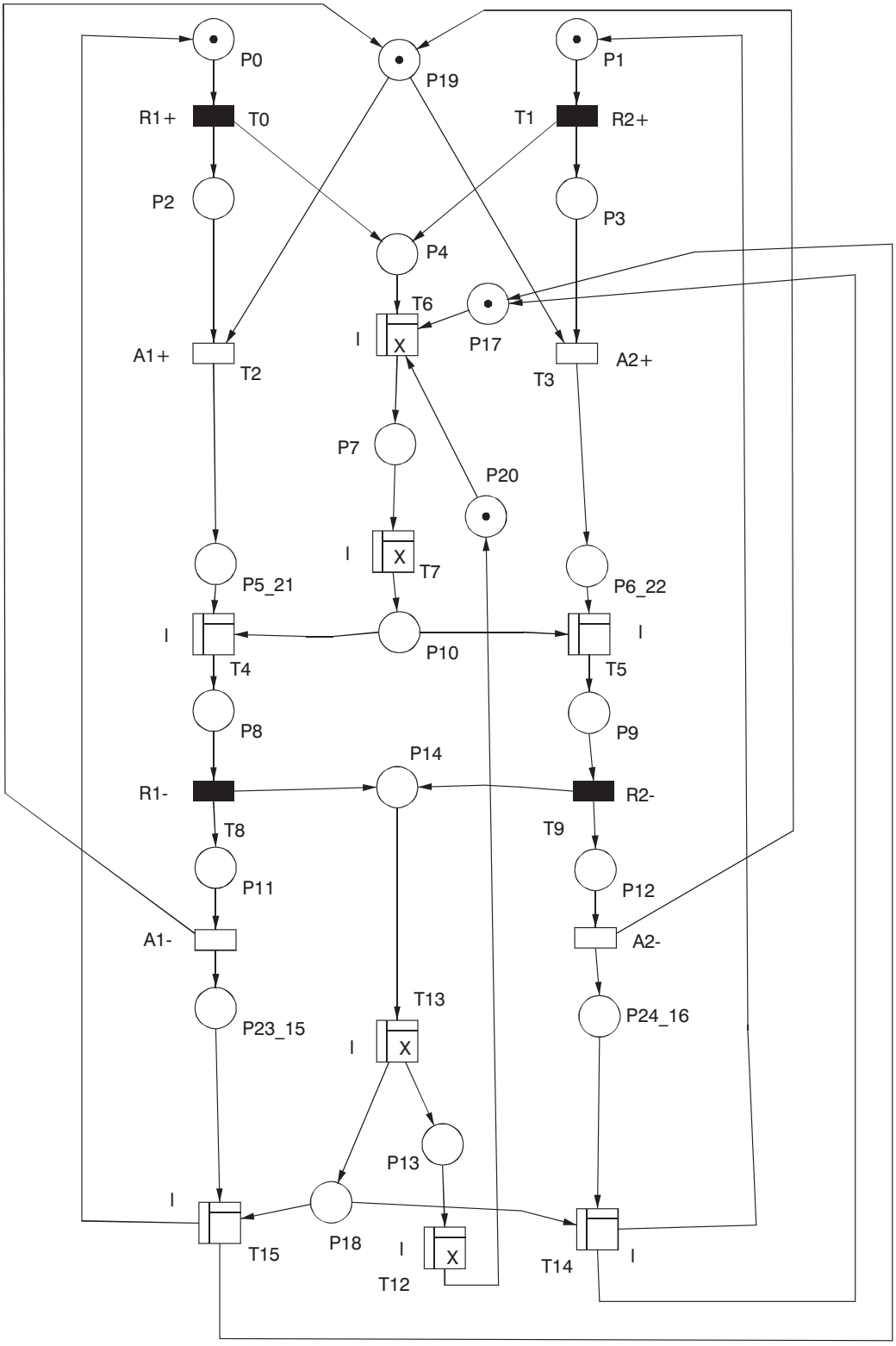

P0
P19
P1
R1+
T0
T1
R2+
P2
P3
P4
T6
P17
A1+
T2
T3
A2+
P7
P20
P5_21
P6_22
P10
T4
T5
P8
P9
P14
R1-
R2-
T8
T9
P11
P12
A1-
A2-
T13
P23_15
P24_16
P13
P18
T15
T12
T14

Figure 14

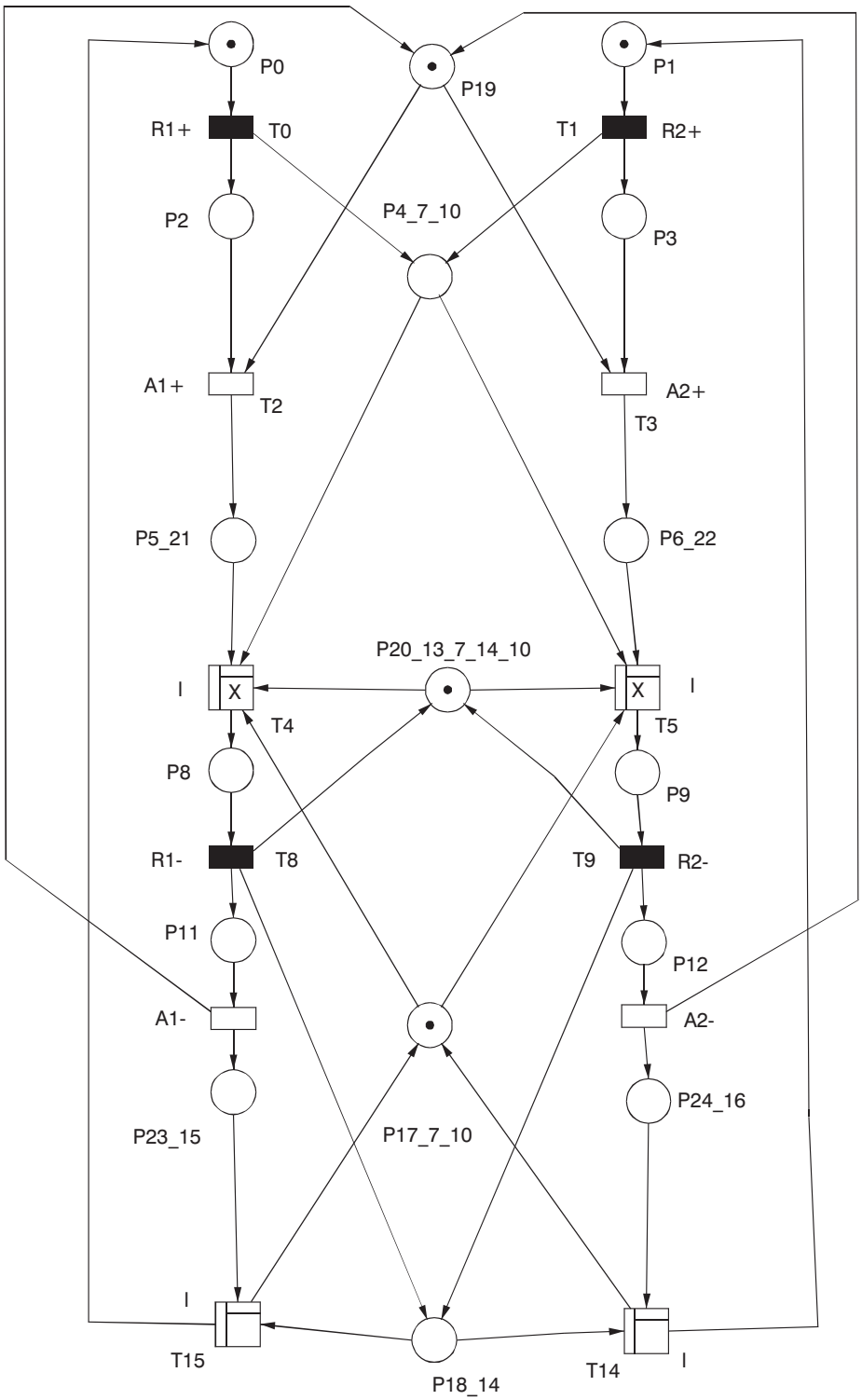

P0
P19
P1
R1+
T0
T1
R2+
P2
P4_7_10
P3
A1+
T2
A2+
T3
P5_21
P6_22
P20_13_7_14_10
I
X
X
I
T4
T5
P8
P9
R1-
T8
T9
R2-
P11
P12
A1-
A2-
P24_16
P23_15
P17_7_10
I
T15
T14
I
P18_14

Figure 15

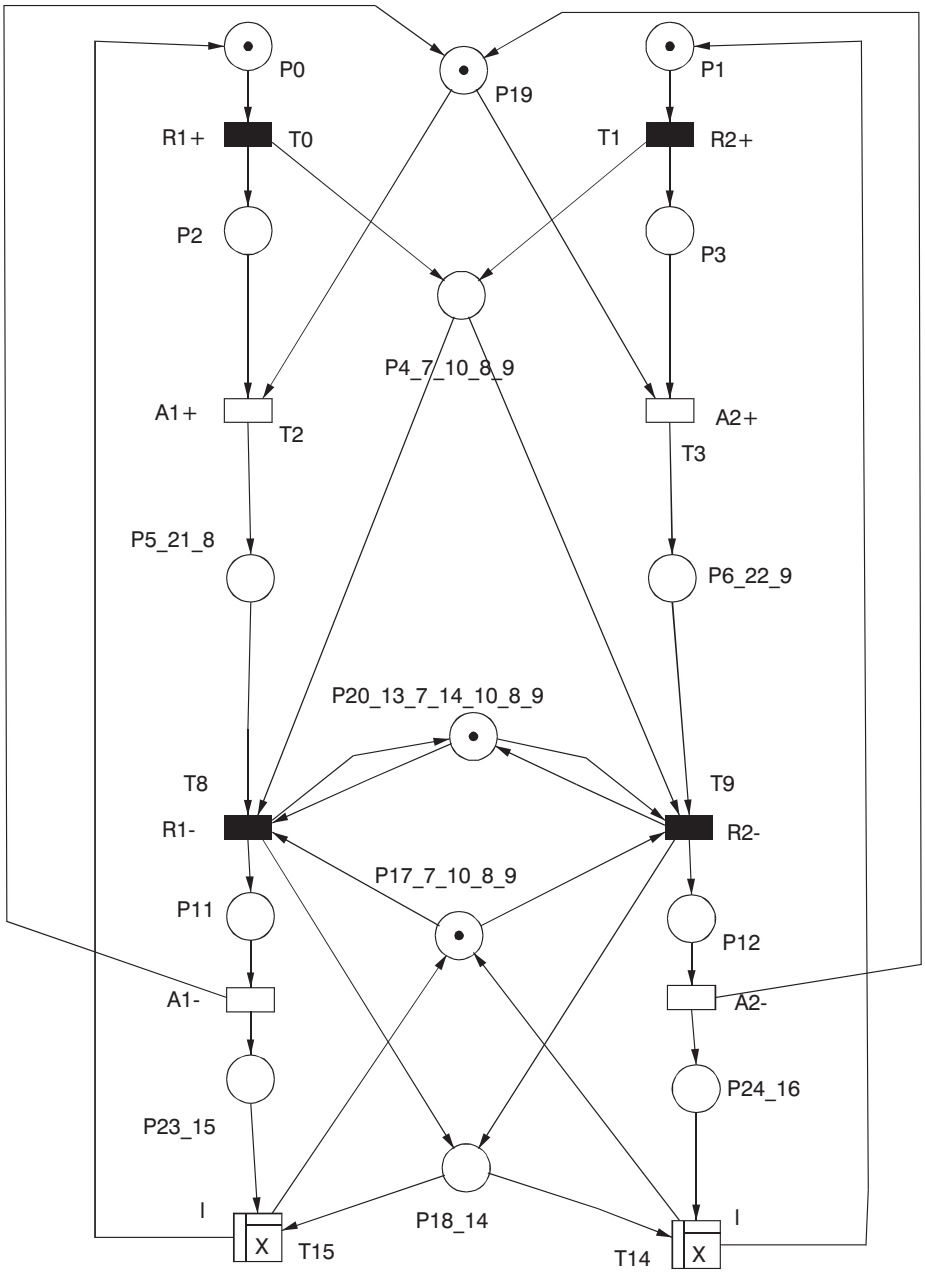

P0
P19
P1
R1+
T0
T1
R2+
P2
P3
P4_7_10_8_9
A1+
T2
A2+
T3
P5_21_8
P6_22_9
P20_13_7_14_10_8_9
T8
T9
R1-
R2-
P17_7_10_8_9
P11
P12
A1-
A2-
P24_16
P23_15
P18_14
I
X
T15
T14

Figure 16

shown to be redundant, and deleting them gives the standard STG representation of the ME-element (e.g. [YKKL94]). (Note that one of the deleted places is a loop-only place.)

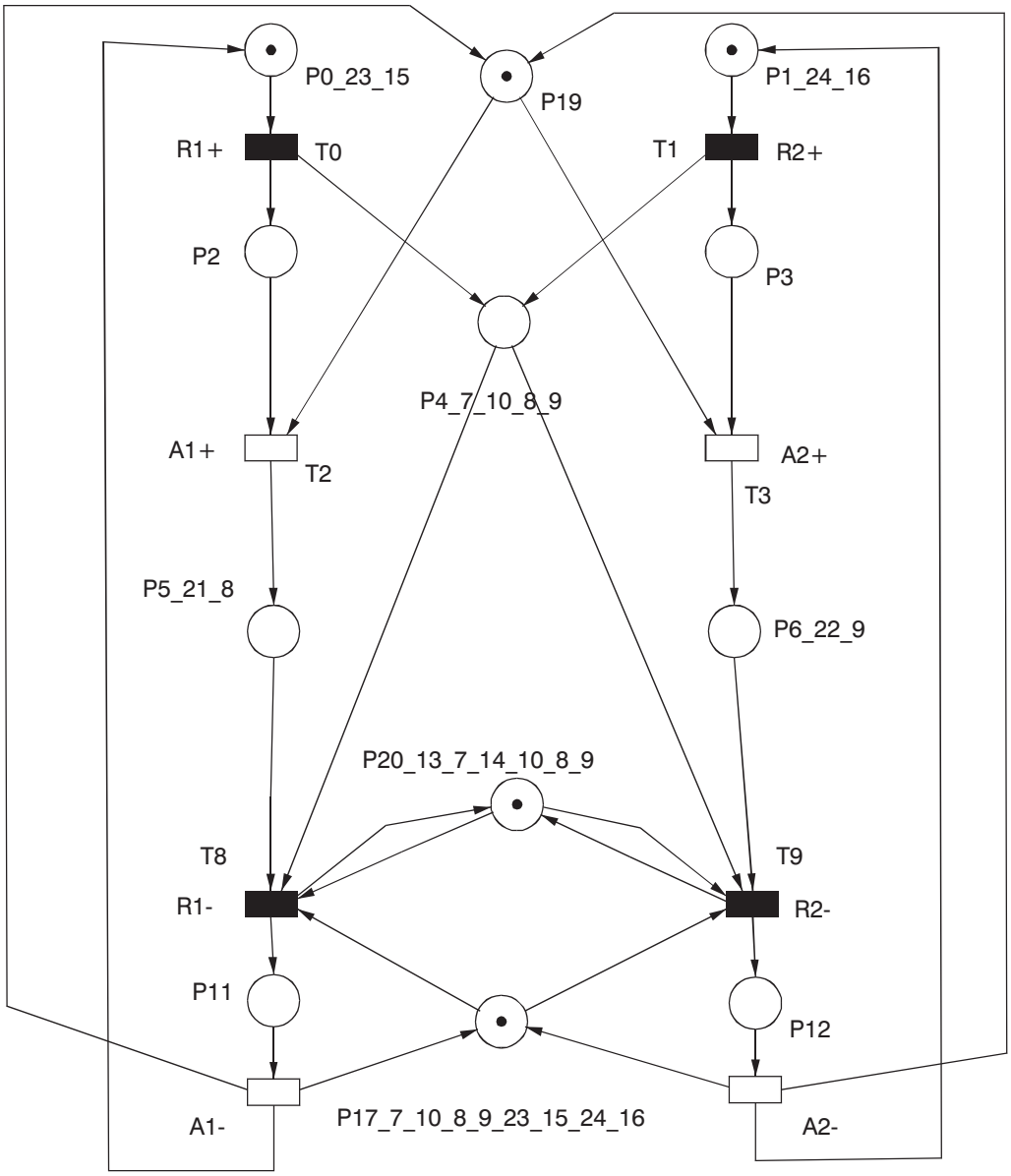


Figure 17

This example is concerned with splitting of a library element and not with savings in the state space. In fact, the second component that generates the other output signals $R*$ and $Gi*$, $i = 1, 2$ and $* \in \{+, -\}$, is not smaller than N. But it does not specify critical races (because the $Ai*$ are inputs for this component), and can therefore be implemented with the known STG-based methods.

6 Conclusion and Future Work

STG decomposition can help in the synthesis of circuits: it may avoid state explosion, it supports the use of library elements, and it leads to a modular implementation that can be more efficient. We have presented a decomposition algorithm that is much more generally applicable than those known from literature, and we have given a formal correctness proof. (In particular, we do not restrict ourselves to live and safe free-choice nets or to marked graphs, and we do not require that each label occurs only once.) The algorithm is based on what we call admissible operations, and certain transition contractions and place deletions have been shown to be admissible. We have demonstrated the usefulness of our algorithm with some practical examples.

There are a number of open problems we will tackle in the near future. The first one concerns determinism: we have assumed that the given STG-specification N, its components C_i we have constructed, and their parallel composition C are deterministic. One would get much more freedom in finding components, if one allowed internal signals for the communication between components; then the composition C would not be deterministic anymore, and to treat such STGs it becomes essential to use some form of bisimulation that is not angelic. Also for N, it can be useful to allow nondeterminism: e.g. to treat arbitration in general, it can be necessary to have several enabled transitions with the same label.

Of course, we have to study more examples; for this, integration of the decomposition algorithm into a tool is needed, and this is already under way. One question then is how useful non-secure contractions and output-redundant, but not redundant place deletions are. Another aspect is that we would like to see examples where it is useful to continue in the algorithm in cases where there is a structural auto-conflict but no dynamic auto-conflict; see the discussion in Section 4.2.

Depending on the choice of a feasible start partition and on the choices of signals for restarts (which for the first kind of failure in Section 4.2 depend on the order of the transition contractions), we can arrive at different decompositions – compare Figure 6. We want to study quality criteria (like the overall number of reachable markings) for decompositions and methods to find good decompositions.

Finally, as argued in [WB00], STGs are not always sufficient to specify the desired behaviour; as an improvement, gSTGs are suggested. Therefore, we want to generalize our approach to gSTGs, and we also want to generalize our correctness criterion to take concurrency into consideration, where it seems to be natural to require the modular implementation to exhibit at least the concurrency prescribed in the specification.

Acknowledgment The authors thank Ben Kangsah for helping with the figures and working out part of the examples.

References

[And83] C. André. Structural transformations giving B-equivalent PT-nets. In Pagnoni and Rozenberg, editors, *Applications and Theory of Petri Nets*, Informatik-Fachber. 66, 14–28. Springer, 1983.

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

[BEW00] J. Beister, G. Eckstein, and R. Wollowski. Cascade: a tool kernel supporting a comprehensive design method for asynchronous controllers. In M. Nielsen, editor, *Applications and Theory of Petri Nets 2000*, Lect. Notes Comp. Sci. 1825, 445–454. Springer, 2000.

[BW93] J. Beister and R. Wollowski. Controller implementation by communicating asynchronous sequential circuits generated from a Petri net specification of required behaviour. In G. Caucier and J. Trilhe, editors, *Synthesis for Control Dominated Circuits*, 103–115. Elsevier Sci. Pub. 1993.

[Chu86] T.-A. Chu. On the models for designing VLSI asynchronous digital systems. *Integration: the VLSI Journal*, 4:99–113, 1986.

[Chu87a] T.-A. Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications*. PhD thesis, MIT, 1987.

[Chu87b] T.-A. Chu. Synthesis of self-timed VLSI circuits from graph-theoretic specifications. In *IEEE Int. Conf. Computer Design ICCD '87*, pages 220–223, 1987.

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

[Dil88] D. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent circuits*. MIT Press, Cambridge, 1988.

[Ebe92] J. Ebergen. Arbiters: an exercise in specifying and decomposing asynchronously communicating components. *Sci. of Computer Programming*, 18:223–245, 1992.

[KGJ96] P. Kudva, G. Gopalakrishnan, and H. Jacobson. A technique for synthesizing distributed burst-mode circuits. In *33rd ACM/IEEE Design Automation Conf.*, pages 67–70, 1996.

[KKT93] A. Kondratyev, M. Kishinevsky, and A. Taubin. Synthesis method in self-timed design. Decompositional approach. In *IEEE Int. Conf. VLSI and CAD*, pages 324–327, 1993.

[Lyn96] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, San Francisco, 1996.

[Pet81] J.L. Peterson. *Petri Net Theory*. Prentice-Hall, 1981.

[Rei85] W. Reisig. *Petri Nets.* EATCS Monographs on Theoretical Computer Science 4. Springer, 1985.

[RY85] L. Rosenblum and A. Yakovlev. Signal graphs: from self-timed to timed ones. In *Proc. Int. Work. Timed Petri Nets*, Torino, Italy, 1985.

[Seg93] R. Segala. Quiescence, fairness, testing, and the notion of implementation. In E. Best, editor, *CONCUR 93*, Lect. Notes Comp. Sci. 715, 324–338. Springer, 1993.

[VYCLdM94] C. Vanbekbergen, C. Ykman-Couvreur, B. Lin, and H. de Man. A generalized signal transition graph model for specification of complex interfaces. In *European Design and Test Conf.*, pages 378–384. IEEE, 1994.

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

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

[Wol97] R. Wollowski. *Entwurfsorientierte Petrinetz-Modellierung des Schnittstellen-Sollverhaltens asynchroner Schaltwerksverbünde.* PhD thesis, Uni. Kaiserslautern, FB Elektrotechnik, 1997.

[YKKL94] A. Yakovlev, M. Kishinevsky, A. Kondratyev, and L. Lavagno. Or causality: Modelling and hardware implementation. In R. Valette, editor, *Applications and Theory of Petri Nets 1994*, Lect. Notes Comp. Sci. 815, 568–587. Springer, 1994.