A Polynomial-Time Algorithm for Checking Consistency of Free-Choice Signal Transition Graphs

Javier Esparza*
Institute for Formal Methods in Computer Science
University of Stuttgart
esparza@informatik.uni-stuttgart.de

Abstract

Signal Transition Graphs (STGs) are one of the most popular models for the specification of asynchronous circuits. A STG can be implemented if it admits a so-called consistent and complete binary encoding. Checking this is EXPSPACE-hard for arbitrary STGs, and so a lot of attention has been devoted to the subclass of free-choice STGs, which offers a good compromise between expressive power and analizability. In the last years, polynomial time synthesis techniques have been developed for free-choice STGs, but they assume that the STG has a consistent binary encoding. This paper presents the first polynomial algorithm for checking consistency.

1 Introduction

Asynchronous circuit design is attracting increasing interest due to some important potential advantages, like absence of clock skew problems and low power consumption [2, 7]. Signal Transition Graphs (STGs) [5, 6, 21] are one of the most popular specification models for asynchronous circuits. They are Petri nets in which the firing of a transition is interpreted as rising or falling of a signal in the circuit. Transitions corresponding to rising (falling) of signal a are labelled by a^+ (a^-). Signals are partitioned into input and output signals, which are supposed to be controlled by the environment and the circuit, respectively.

Given a STG, the synthesis problem consists in producing an asynchronous circuit exhibiting the same behaviour. Such a circuit is known to exist if the state graph of the STG admits a *consistent* and *complete binary encoding*. A binary encoding is a function that assigns to each reachable state a vector of booleans, one for each signal, indicating if the signal is up (1) or down (0). Loosely speaking, an encoding is consistent if it respects the semantics of transition labels: e.g., if the firing of a transition labelled by a^+ leads from state s to state s', then signal a must have value 0 in s and value 1 in s'. In particular, occurrences of a^+ and a^- must alternate along every firing sequence. Completeness requires the binary code of a state to contain enough information to determine the behaviour of the circuit; if two states have the same code, then they have to enable exactly the same output signals.

Early techniques for the synthesis of asynchronous circuits from STG specifications proceeded by first constructing the state graph of the STG, then computing a consistent and complete binary encoding (if it exists), and then synthesizing the circuit from the encoding. This approach is fairly well understood (see e.g. [16, 24, 12]), and there exists good tool support, e.g. like that provided by the Petrify tool [22]). If these techniques are used, checking consistency and completeness are minor problems, since the check can be performed in linear time in the size of the reachability graph.

However, these approaches suffer strongly from the state explosion problem: the number of states of the state graph can grow super-exponentially in the size of the STG, or even be infinite. As synthesis tools for asynchronous systems start to mature, the size of STGs increases and techniques based on the state graph become obsolete. Therefore, much effort is being devoted to synthesis techniques that avoid the construction of the state graph. In this new setting consistency and completeness become a major problem. Checking them is EXPSPACE-hard in the size of the STG, even if its state graph is known to be finite (a result that follows easily from the EXPSPACE-hardness of the reachability problem for Petri nets [18, 9]). For 1-bounded STGs, in which a place can contain at most 1 token (the case most common in practice), checking consistency and completeness is still PSPACE-complete. PSPACE-hardness follows easily from the PSPACE-hardness of the reachability problem for 1-bounded Petri nets [9], while membership in PSPACE is trivial.

In order to cope with this problem, syntactically defined

*This work was done while the author was at the University of Edinburgh.

subclasses of STGs have been studied. The class with the best compromise between expressive power and analizability are *free-choice* STGs [5], a class that allows to model both nondeterminism (necessary for modelling the environment) and concurrency (essential for asynchronous modelling), but restricts their interplay. As a matter of fact, STGs were originally defined in [5] as free-choice nets, and many papers still identify STGs with free-choice STGs. Loosely speaking, a STG is free-choice if for every place p, whenever some output transition of p is enabled, all output transitions of p are enabled, and so it is always possible "to freely choose" which of them fires. This is an adequate model of the behaviour of the environment, which should be able to freely produce any input signal to the circuit. Many asynchronous circuits can be naturally specified using free-choice nets, although they are not powerful enough to model arbiters.

Free-choice STGs and subclasses thereof have been studied in numerous papers (see e.g. [5, 3, 19, 23]). A number of techniques exist for the automatic implementation of STGs with consistent and complete encodings. In [3] it is also shown how to transform a free-choice STG having a consistent encoding into another one which admits a consistent and complete encoding. The problem of checking consistency has been studied in [19], where two polynomially checkable conditions for consistency, one sufficient and the other necessary, are presented. However, the exact computational complexity of checking consistency and completeness is still unknown.

In this paper we attack the consistency problem with results of the theory of free-choice Petri nets obtained in the early 90s [8]. We show that consistency can be checked in polynomial time for free-choice STGs known to be bounded, deadlock-free, and cyclic (meaning that the initial marking can be reached from any other reachable marking). These are standard conditions used by all papers on the subject since Chu's work [5], and in particular by [3, 19]. These conditions hold (or can be artificially made to hold, for instance by adding transitions that 'restore' the initial marking) for many real specifications, simplify the synthesis procedures, and can be checked in polynomial time [8].

The paper is organized as follows. Section 2 contains basic definitions. Section 3 and 3.3 introduce the checking procedure. The resulting algorithm is presented in Section 4, and Section 5 contains some brief conclusions. All proofs can be found in [11].

2 Basic definitions

A *net* is a triple (P, T, F), where P and T are disjoint sets of *places* and *transitions*, respectively, and F is a function $(P \times T) \cup (T \times P) \to \{0, 1\}$. Places and transitions are generically called *nodes*. Places are graphically represented as circles, and transitions as boxes. If $F(x, y) = 1$ then we say that there is an *arc* from x to y. The *preset* of a node x, denoted by ${}^\bullet x$, is the set of its input nodes, i.e., the set $\{y \in P \cup T \mid F(y, x) = 1\}$. The *postset* of x, denoted by $x^\bullet$, contains its output nodes, i.e., the set $\{y \in P \cup T \mid F(x, y) = 1\}$.

A *marking* M of a net (P, T, F) is a mapping $P \to \mathbb{N}$ (where $\mathbb{N}$ denotes the natural numbers including 0). Graphically, a marking is represented by drawing $M(p)$ tokens on the circle representing the place p. A marking M *enables* a transition t if it puts at least one token on each place $p \in {}^\bullet t$, i.e. if $M(p) > 0$ for each $p \in {}^\bullet t$. If t is enabled at M, then it can *fire* or *occur*, and its occurrence *leads to* a new marking L, obtained by removing a token from each place in the preset of t, and adding a token to each place in its postset; formally, $L(p) = M(p) + F(t, p) - F(p, t)$ for every place p. $M \xrightarrow{t} M'$ denotes that t is enabled at M and that its occurrence leads to M'.

A *Petri net* is a pair (N, M_0) where N is a net and M_0 is a marking of N, called the *initial marking*. A sequence of transitions $\sigma = t_1 t_2 \dots t_n$, $n \geq 0$, is an *occurrence sequence from M to M'* if there exist markings M_1, M_2, ..., M_{n-1} such that

$$M \xrightarrow{t_1} M_1 \xrightarrow{t_2} \dots M_{n-1} \xrightarrow{t_n} M'$$

We usually omit the intermediate markings and write $M \xrightarrow{\sigma} M'$ for an occurrence sequence; $M \xrightarrow{\sigma}$ denotes that σ can be fired from M. The *Parikh mapping* of $\sigma \in T^*$, denoted by $\vec{\sigma}$, is the mapping $T \to \mathbb{N}$ that assigns to each transition the number of times it occurs in σ. A marking M is *reachable* if there exists an occurrence sequence from M_0 to M. The *reachability graph* of a Petri net is a labelled graph having the set of reachable markings as nodes, and the restriction of the $\xrightarrow{t}$ relations to the set of reachable markings as edges.

Signal transition graphs. We fix a finite alphabet $A = \{a_1, \dots, a_n\}$ of *signals* partitioned into *input* and *output* signals. (All our results can be immediately extended to STGs with dummy transitions, we omit them for clarity.) Rising and falling of a signal a is denoted by a^+ and a^-, respectively. We call an element of $\mathcal{L} = A \times \{+, -\}$ a *label*. Loosely speaking, a signal transition graph is a Petri net whose transitions are labelled with elements of $\mathcal{L}$. Formally, a *signal transition graph* (STG) is a triple $S = (N, M_0, \ell)$, where (N, M_0) is a Petri net and ℓ is a surjective *labelling function* that assigns to each transition of N a label in $\mathcal{L}$.

The *state graph* of a STG has the reachable states of S as nodes. If $M \xrightarrow{t} L$ in the reachability graph, then the state graph contains en edge $M \xrightarrow{\ell(t)} L$. Figure 1 shows a signal transition graph and its state graph.

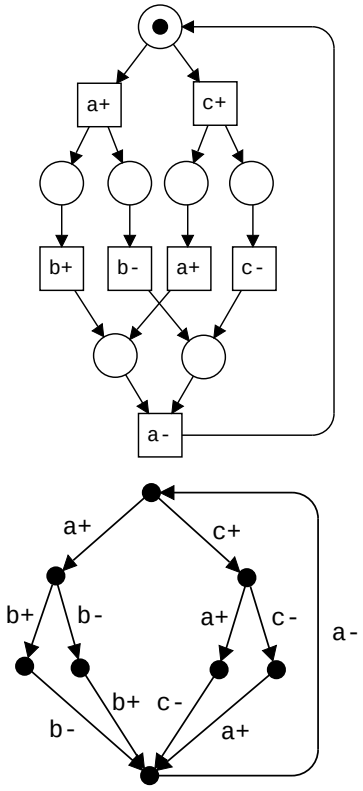

Figure 1. A free-choice STG and its state graph

A signal transition graph is a specification of the behaviour of the circuit under some assumptions on the environment. A signal transition graph S is implementable if there exists a *state coding mapping* λ that associates to each reachable marking M a vector of *signal values* $\lambda(M) \in \{0,1\}^n$ satisfying the following two properties:

- *Consistency*. If $M \xrightarrow{t} L$ and t is labelled by a_i^+, then the i-th components of $\lambda(M)$ and $\lambda(L)$ are 0 and 1, respectively, and all other components have the same value in $\lambda(M)$ and $\lambda(L)$. If t is labelled by a_i^-, then the i-th components of $\lambda(M)$ and $\lambda(L)$ are 1 and 0, respectively, and all other components have the same value in $\lambda(M)$ and $\lambda(L)$.
- *Completeness*: if two different reachable markings M, L satisfy $\lambda(M) = \lambda(L)$, then they enable exactly the same output labels.

Consistency is obviously necessary for implementability. Completeness is necessary because the state of an implementation is completely determined by the signal values of all signals. Therefore, if some output signal is enabled at M but not at L, M and L must correspond to different states of the implementation, and so they must differ in the value of at least one signal.

We say that a STG is consistent or complete if it has a consistent or complete binary encoding, respectively. The STG of Figure 1 is not consistent. To see why, let M be the marking reached by firing a^+. Since M enables both b^+ and b^-, a consistent encoding λ must satisfy both $\lambda(M)(b) = 0$ and $\lambda(M)(b) = 1$, and so it cannot exist. If the transition at the top labelled by a^+ and the transition labelled by b^- swap their labels, then the STG becomes consistent.

A marking M of a STG is *n-bounded* if $M(p) \leq n$ for every place p. A STG is n-bounded if all its reachable markings are n-bounded. Since the circuit implementation of a STG can be seen as a finite object with at most 2^n states, where n is the number of signals, STGs used in practice are bounded (even though in principle unbounded STGs could make sense), and in fact most of them are even 1-bounded. A STG is *deadlock-free* if every reachable marking enables some transition. It is *cyclic* if for every reachable marking there is a sequence σ such that $M \xrightarrow{\sigma} M_0$, where M_0 is the initial marking. STGs specifying circuits which continuously interact with their environment are deadlock-free. Many are also cyclic, since after each round of interaction the circuit usually returns, or at least may return, to a home state. A STG is *well formed* if it is deadlock-free, bounded, and cyclic. The STG of Figure 1 is well formed.

Checking well formedness of STGs or consistency of well formed STGs is EXPSPACE-hard. Therefore we introduce a syntactically defined subclass of STGs. A STG is *free-choice* if its underlying net (P, T, F) satisfies the following property: for each place p and every transition t, if $F(p,t) = 1$ then $F(p',t') = 1$ for every $p' \in {}^\bullet t$, $t' \in p^\bullet$. In a free-choice STG, if some output transition of a place is enabled at a marking, then all its output transitions are enabled, and it is possible to "freely" choose among them.

An important property of well formed free-choice STGs is that they are live (see for instance Theorem 4.31 of [8]). A STG is *live* if for every reachable marking M and each transition t $M \xrightarrow{\sigma t}$ holds for some sequence σ of transitions. Moreover well formedness of free-choice STGs can be checked in polynomial time (see e.g Corollary 6.18 and Theorem 8.12 of [8]).

We say that a STG is *alternating* if the signs of all signals alternate in all paths of its state graph. The STG of Figure 1 is not alternating: its state graph contains a path in which b^- occurs twice without any occurrence of b^+ in-between.

We conclude this section with a simple result showing that for live and cyclic systems consistency is equivalent to alternation.

Lemma 2.1 *(1) Consistent STGs are alternating.*

(2) Live, cyclic, and alternating STGs are consistent.

3 Checking consistency

The next three sections present a three-step procedure for checking consistency of well formed free-choice STGs. We will make implicit use of Lemma 2.1 along the way.

The first two steps check non-autoconcurrency and balancedness, two necessary conditions for consistency. The third step is a divide-and-conquer procedure which checks consistency of STGs that pass the first two tests.

3.1 Checking non-autoconcurrency

A marking M *concurrently enables* two transitions t and u of a net if it puts at least one token in all input places of t and u, and at least two tokens on every input place of both t and u. It follows easily from the firing rule that if M concurrently enables t and u then there is a marking L such that both $M \xrightarrow{tu} L$ and $M \xrightarrow{ut} L$ are occurrence sequences. A STG is *non-autoconcurrent* if no reachable marking concurrently enables two transitions labelled by the same signal (even if the labels carry opposite signs). We have the following (very easy to prove) result:

Lemma 3.1 *Consistent STGs are non-autoconcurrent.*

By Lemma 3.1, if a well formed free-choice STG does not pass the non-autoconcurrency test, we can already dismiss it as non-consistent. The STG of Figure 1 does not pass the test, because b^+ and b^- can be concurrently enabled. Checking non-autoconcurrency is EXPSPACE-complete for arbitrary STGs (this can be easily proved using the techniques of [9]). Fortunately, for well formed free-choice STGs we can do much better: It is proved in [14] that non-autoconcurrency can be checked in polynomial time. A summary of this result can be found in [11].

3.2 Checking balancedness

A STG is *balanced* if for every signal a, every cycle of the state graph contains the same number of occurrences of a^+ and a^-. Clearly, balancedness is a necessary condition for alternation and hence for consistency. We show how to check balancedness.

Let $S = (N, M_0, \lambda)$ be a STG, where $N = (P, T, F)$. A mapping $J: T \to \mathbf{Q}$, where $\mathbf{Q}$ denotes the rational numbers, is a *T-invariant* of N if $\sum_{t \in {}^\bullet p} J(t) = \sum_{t \in p^\bullet} J(t)$ for every place p. Notice that T-invariants form a vector space. A T-invariant J is *positive* (semi-positive) if $J(t) > 0$ ($J(t) \geq 0$) for all $t \in T$.

It is easy to show that if $M \xrightarrow{\sigma} M$ for some reachable marking M, then $\vec{\sigma}$ is a semi-positive T-invariant. However, given a semi-positive (integer) T-invariant J, there may be no cycle $M \xrightarrow{\sigma} M$ in the reachability graph of Σ such that $J = \vec{\sigma}$; when such a cycle exists we say that J can be *activated*.

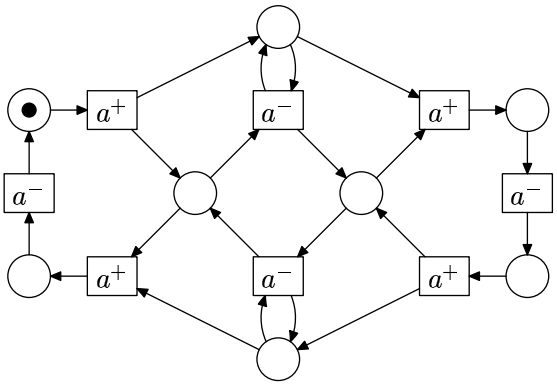

Figure 2. A well formed and balanced STG

Fortunately, for well formed free-choice STGs a limited version of the converse holds. Let the *support* of a semi-positive T-invariant be the set of transitions t such that $J(t) > 0$. A semi-positive T-invariant is *minimal* if there is no other semi-positive T-invariant with strictly smaller support.

Theorem 3.1 *([8], Theorems 5.17 and 5.20)*
Let S be a well formed free-choice STG. All minimal T-invariants of N can be activated.

We use this result to check balancedness. Given a mapping $J: T \to \mathbf{Q}$ and a signal a we write $\#(J, a^+)$ ($\#(J, a^-)$) to denote the sum of $J(t)$ over all transitions t labelled by a^+ (a^-), and define $bal(J): A \to \mathbb{N}$ by $bal(J) = \#(J, a^+) - \#(J, a^-)$ for every signal a.

Theorem 3.2 *Let S be a well formed free-choice STG. S is balanced iff $bal(J) = 0$ for every T-invariant J of S.*

The STG of Figure 2 (taken from [20] with some modifications) shows that Theorem 3.2 does not hold for arbitrary well formed STGs. The STG is balanced, but the mapping J that assigns 0 to the two transitions in the middle of the picture and 1 to the others is a T-invariant for which $bal(J) \neq 0$. Notice that this T-invariant cannot be activated.

T-invariants can be computed using linear algebra. The *incidence matrix* of a net N, denoted by $\mathbf{N}$, is a $|P| \times |T|$ matrix given by $\mathbf{N}(p, t) = F(t, p) - F(p, t)$. It is easy to see that $M \xrightarrow{\sigma} L$ implies $L = M + \mathbf{N} \cdot \vec{\sigma}$. (For $\sigma = t_1 \ldots t_n$, this is nothing but the equalities $L(p) = M(p) + F(t_1, p) - F(p, t_1) + \ldots + F(t_n, p) - F(p, t_n)$ written in matrix form.) A little calculation shows that J is a T-invariant iff $\mathbf{N} \cdot J = 0$. So we have the following corollary of Theorem 3.2:

Corollary 3.1 *Let S be a well formed free-choice STG. S is balanced iff every rational solution of the linear system of equations $\mathbf{N} \cdot X = 0$ satisfies $bal(X) = 0$.*

So balancedness can be solved by computing a basis of the solutions of $\mathbf{N} \cdot X = 0$ and checking that each element J of the basis satisfies $bal(J) = 0$.

3.3 A divide-and-conquer procedure

In the rest of the section we work with well formed free-choice STGs which are both balanced and non-autoconcurrent. We call these STGs *very well formed*.

The third step of our procedure to check alternation is a divide-and-conquer procedure for checking alternation of very well formed free-choice STGs (i.e., of those well formed free-choice STGs which have passed both the non-autoconcurrency and the balancedness tests): we decompose the STG into two parts and show that checking alternation of the whole reduces to checking alternation of each part. Recall that, by Lemma 2.1, very well formed STGs are consistent if and only if they are alternating.

The divide-and-conquer procedure is best explained by considering the special case of STGs in which every transition has exactly one input and one output place. In the sequel, these are called *state machine* STGs[1]. Figure 3 shows a very well formed state machine STG. This figure and the next ones follow two drawing conventions. First, only the label of a transition is drawn (its surrounding box is omitted). Second, places having only one input and one output transition are also omitted. Therefore, an edge of the form, say, $a^+ \to b^-$ indicates that the STG contains an arc from a transition labelled by a^+ to a place, and another one from this place to a transition labelled by b^-, and that the place has no other input or output transitions. Tokens on the intermediate place are just drawn on the arc.

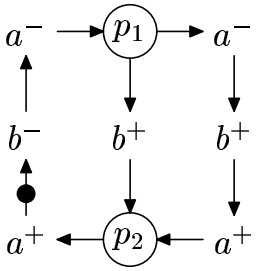

Figure 3. A very well formed state machine STG

Let S be a very well formed state machine STG. Now, let a *handle* be a simple path of (the underlying net of) S satisfying the following conditions: it starts and ends at a place, all its intermediate nodes have exactly one input and one output node, and after removing these intermediate nodes together with their adjacent arcs, the remaining STG is still strongly connected. In Figure 3, the path on the right that visits p_1, a^-, b^+, a^+, p_2 is a handle.

Without loss of generality, we assume that the intermediate places of S are empty of tokens at the initial marking (if this is not the case, we can fire transitions to remove those tokens; by cyclicity, alternation holds for the new STG if and only if it holds for the old one). Let H and R be STGs whose underlying nets are the handle and the remaining STG, respectively; the initial marking of H puts one token on its first place and no tokens elsewhere, while the initial marking of R is the projection of the initial marking of S onto R. H and R for the example of Figure 3 are shown in Figure 4.

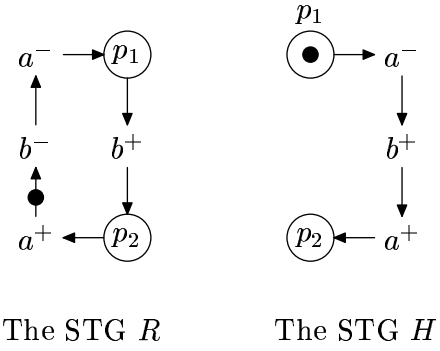

Figure 4. Dividing the STG of Figure 3

We would like to have that S is alternating if and only if H and R are alternating. However, Figure 3 shows that this is not true. In this example, S is not alternating because of the two consecutive a^- at the top of the figure, even though both H and R are alternating.

Our solution consists of modifying R to make sure that, if S is not alternating and H is alternating, then the new R is not alternating.

In order to see how to modify R, we look at the path of R visiting p_1, b^+, p_2. This is a 'mirror image' of H in R, i.e., a path of R with the same start and end nodes as H. Let α and β be the sequences of labels corresponding to H and its mirror image, i.e., $\alpha = a^- b^+ a^+$ and $\beta = b^+$. Intuitively, the modification of R should guarantee that if we can use α to produce a non-alternating sequence, then we can also use β, i.e., that α and β are 'equivalent' with respect to their potential for producing non-alternating sequences.

But how could a formal definition of 'equivalent' look like? It is not difficult to see that we could choose this one: for each signal a, if the projection of α on a is nonempty, then the projections of α and β on a begin with the same label and end with the same label. However, we have not been able to transform this definition into a polynomial divide-and-conquer procedure, and so we choose another

[1] Checking consistency of state machine STGs is a trivial problem. We use them in order to present some of the ideas of the free-choice case in a simple setting.

one which also works: (1) α and β have the same balance, (2) for each signal a, if a occurs in α then it also occurs in β, and (3) if a occurs in α and β then the first occurrence of a has the same sign in both sequences. In Figure 3, α and β are not equivalent because condition (2) is violated.

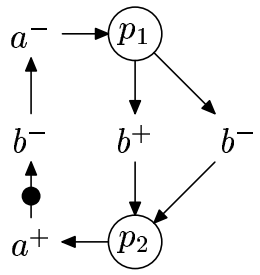

Figure 5. A state machine STG

So our modification of R should guarantee that α and β satisfy conditions (1), (2) and (3). Fortunately, it is easy to see that if (1) is violated then the STG is not very well formed. Figure 5 shows an example. We have $\alpha = b^-$ and $\beta = b^+$, and so condition (1) is violated; the STG contains a circuit with two occurrences of b^- and none of b^+, and so it is unbalanced. Since we assume that our initial STG is very well formed, we do not have to worry about ensuring condition (1): it holds automatically.

In order to guarantee (2) and (3), we insert a 'witness' in R, as shown in Figure 6 for the STG R of Figure 4. The witness 'records' that a occurs in α, and that the first occurrence of a has negative sign. Notice that, after introducing the witness, R is no longer alternating. The two consecutive a^- that were possible in S are now possible in the modified R as well.

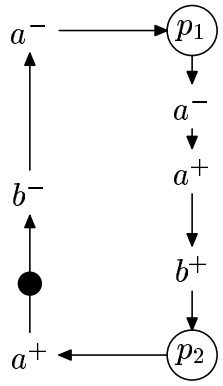

Figure 6. Inserting a witness in R

In the rest of the section we show how to extend these ideas to free-choice STGs. In particular, handles can no longer be just paths, due to the presence of concurrency. Section 3.3.1 introduces CP-subnets[2], which are a generalization of handles. Section 3.3.2 formalizes the insertion of witnesses. Section 3.3.3 shows the correctness of the divide-and-conquer steps: alternation holds for the original STG if and only if it holds for the parts. Finally, Section 3.3.4 shows how to directly check consistency of STGs that do not contain any CP-subnets.

3.3.1 Handles generalize to CP-subnets

A *transition-generated subnet* of a net $N = (P, T, F)$ is a net $N' = (P', T', F')$ such that $T' \subseteq T$, $P' = {}^\bullet T' \cup T'^\bullet$ (i.e., P' contains all input and output places of T' in the net N), and $F' = F \cap ((P' \times T') \cup (T' \times P'))$. Transition-generated subnets are just called subnets in the sequel. Given a subnet N', we define $N \setminus N'$ as the subnet having $T \setminus T'$ as set of transitions. A place p of N' is called *entry* (*exit*) place if some transition of ${}^\bullet p$ ($p^\bullet$) belongs to $N \setminus N'$. All other places of N' are called *internal*. The output transitions of the entry places are called *entry transitions*. All other transitions of N' are called *internal*.

N' is a *CP-subnet* of N if

(i) it is nonempty and weakly connected,

(ii) every internal place has exactly one input and one output transition, and

(iii) the net $N \setminus N'$ is strongly connected.

The handle of Figure 4 is an example of CP-subnet. A more interesting example is the subnet of Figure 1 generated by the three transitions of the left-half of the net labelled by a^+, b^+, and b^-. Finally, Figure 7 shows a free-choice STG, which in Figure 8 is divided into a CP-subnet, on the right, and the rest, on the left. (The fact that the initial markings of Figure 7 and Figure 8 are different will be explained later).

In the sequel, $\overline{N}$ denotes a CP-subnet of a net N. Given a STG $S = (N, M_0, \lambda)$, $S \setminus \overline{N}$ denotes the STG whose underlying net is $N \setminus \overline{N}$, and whose initial marking and labelling function are obtained by projecting M_0 and ℓ onto $N \setminus \overline{N}$.

We shall use the following result

Lemma 3.2 *([8]) CP-subnets of well formed free-choice STGs have a unique entry transition.*

A *flushing sequence* of $\overline{N}$ in S is an occurrence sequence $M \xrightarrow{\sigma} L$ of S such that:

- M and L are reachable markings of S that enable no internal transitions of $\overline{N}$, and
- $\sigma = \overline{t}_e \tau$, where $\overline{t}_e$ is the unique entry transition of $\overline{N}$, and τ contains no occurrences of $\overline{t}_e$.

[2]This strange name is due to historical reasons.

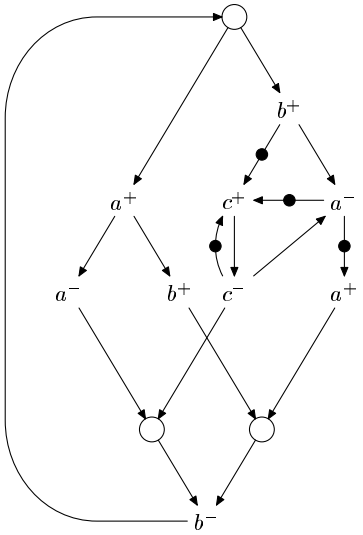

Figure 7. A free-choice STG

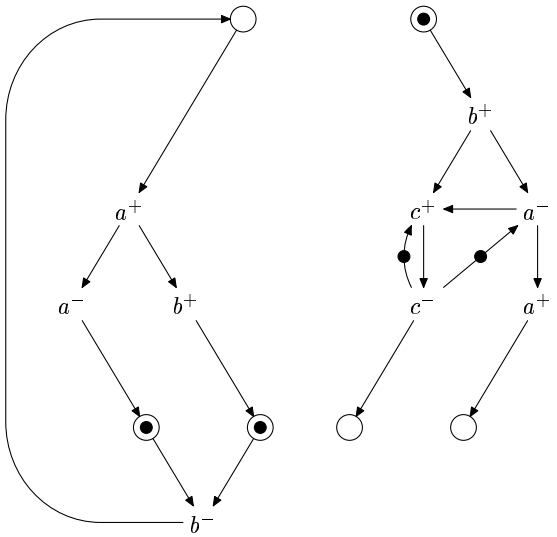

Figure 8. Dividing the STG of Figure 7

Intuitively, in a flushing sequence the entry transition of $\overline{N}$ fires once, and then the internal transitions of $\overline{N}$ fire for as long as possible. In the case of handles, a flushing sequence lets a token run along the handle, from its first to its last place, and so handles have one single flushing sequence. This is no longer the case for CP-subnets, because concurrency among the transitions of $\overline{N}$ is possible, and so we can get a new flushing sequence by changing the order in which concurrently enabled transitions fire. For instance, in the CP-subnet of Figure 8, there are three possible flushing sequences, corresponding to the sequences of labels $b^+a^-a^+c^+c^-$, $b^+a^-c^+a^+c^-$, and $b^+a^-c^+c^-a^+$. However, for very well formed STGs we can prove that CP-subnets enjoy many of the properties of handles:

Proposition 3.1 *Let S be a very well formed free-choice STG, let $\overline{N}$ be a CP-subnet of S, and let $M \xrightarrow{\sigma} M'$ and $L \xrightarrow{\tau} L'$ be arbitrary flushing sequences of $\overline{N}$ in S. Then*

(1) M, M', L, L' coincide on all internal places of $\overline{N}$,

(2) the projections of $\ell(\sigma)$ and $\ell(\tau)$ on each signal coincide, and

(3) every transition of $\overline{N}$ occurs exactly once in σ and τ.

So, even if in a very well formed free-choice STG a CP-subnet may have many different flushing sequences, they all start and end at the same marking of $\overline{N}$ (the markings may differ elsewhere), each transition occurs in them exactly once, and, for every signal a, their projections onto the labels a^+, a^- coincide.

Proposition 3.1 justifies the following definition. Given an arbitrary flushing sequence $M \xrightarrow{\sigma} L$ of $\overline{N}$ in S, we define the *characteristic marking* of $\overline{N}$, denoted by $\overline{M}$, as the marking that puts one token on the entry places of $\overline{N}$, and coincides with M (and L) elsewhere. The *characteristic sequence* of $\overline{N}$ with respect to signal a, denoted by $\overline{\sigma}_a$, is the projection of $\ell(\sigma)$ onto a^+ and a^-. The characteristic marking of the CP-subnet of Figure 8 is the one shown in the figure, and we have $\overline{\sigma}_a = a^-a^+$, $\overline{\sigma}_b = b^+$, and $\overline{\sigma}_c = c^+c^-$.

In the case of state machine STGs we divide the STG into a handle H and the rest R. In the free-choice case, the rôle of H is played by the STG S_{CP} having $(\overline{N}, \overline{M})$ as underlying Petri net. By Proposition 3.1(2), checking consistency of S_{CP} is very simple: if suffices to construct *any* flushing sequence (all of them have linear length), and check if they are alternating.

3.3.2 Inserting witnesses

In this section we define the STG playing the rôle of R. Let S be a very well formed free-choice STG with a CP-subnet $\overline{N}$. Without loss of generality, we assume that the initial marking M_0 of S does not enable any internal transition of $\overline{N}$. If this were not the case, we just let the internal transitions of $\overline{N}$ occur for as long as possible. For instance, in the STG of Figure 7 we let the transitions labelled by c^+, c^-, and a^+ occur. Since S is cyclic, the resulting STG is alternating if and only if S is alternating.

Let A' be the set of signals a such that $\overline{\sigma}_a$ is of the form $a^+\alpha a^-$ or $a^-\alpha a^+$ for some $\alpha \in \mathcal{L}^*$. The procedure to construct S_R is better described in an informal but hopefully precise way, since a formal definition would be difficult to read.

- Start with the STG $S \setminus \overline{N}$.
- Let P_e be the set of entry places of $\overline{N}$ (these are the

input places of the unique entry transition $\overline{t}_e$), which belong to $S \setminus \overline{N}$. Let $T' = P_e^\bullet \setminus \{\overline{t}_e\}$. Remove all the arcs (p,t) of $N \setminus \overline{N}$ such that $p \in P_e$ and $t \in T'$.

- Construct a net consisting of one single path, starting at a transition and ending at a place. The path, which we call the *witness* of a, contains two transitions t_a, u_a for each label $a \in A'$, with t_a preceding u_a. If $\overline{\sigma}_a$ is of the form $a^+\alpha a^-$, then t_a is labelled by a^+, and u_a is labelled by a^-. If it is of the form $a^-\alpha a^+$, then t_a is labelled by a^- and u_a is labelled by a^+. The order between transitions corresponding to different labels can be chosen arbitrarily. Let t_f be the first transition of the path, and let p_l be its last place.

- Add arcs $\{(p, t_f) \mid p \in P_e\}$ and $\{(p_l, t) \mid t \in T'\}$.

(At this point, the reader may ask why do we introduce witnesses only for the actions of the set A'. While we could safely introduce witnesses for all signals whose characteristic sequence is nonempty, it turns out to be superfluous.)

In the case of Figure 8 the set A' contains the signals a and c. The STG S_R is shown in Figure 9.

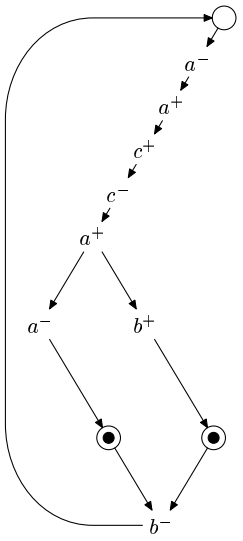

Figure 9. The STG S_R

Notice that if alternation holds for $S \setminus \overline{N}$, then it also holds for S_R. The converse, however, is not true, as shown by Figure 3.

3.3.3 Correctness of the divide-and-conquer strategy

We now state the two essential correctness properties of the divide-and-conquer procedure. First, we have to show that alternation holds for S (and for all signals) if and only if it holds for both S_{CP} and S_R, as defined in Sections 3.3.1 and 3.3.2.

Theorem 3.3 *Let S be a very well formed free-choice STG, and let $\overline{N}$ be a CP-subnet of S. S is alternating if and only if S_{CP} and S_R are alternating.*

Second, since we assume that S is a very well formed free-choice STG, we have to prove that S_R is also very well-formed, so that the procedure can be iterated with S_R.

Proposition 3.2 *S_R is a very well formed free-choice STG.*

3.3.4 The base case

Theorem 3.3 and Proposition 3.2 allow to iteratively remove CP-subnets from S until either non consistency is disproved, or no CP-subnets can be found. It is easy to show that one of the two will eventually be the case: every time a CP-subnet is removed, the sum over all places p of $|p^\bullet| - 1$ strictly decreases. If this sum becomes 0, then every place has exactly one output transition, and so no CP-subnets can exist.

It is shown in [8] (Proposition 7.11) that if no CP-subnets exist, then each place has exactly one output *and* exactly one input transition. Such nets are called T-nets or marked graphs, and we call the corresponding STGs *marked graph STGs*.

It is well known that well formed marked graphs have the following property: there exists an occurrence sequence $M_0 \xrightarrow{\sigma} M_0$ in which all transitions occur exactly once. This sequence can obviously be iterated an arbitrary number of times. We have the following result:

Theorem 3.4 *Let S be a very well formed marked graph STG with initial marking M_0. Let $M_0 \xrightarrow{\sigma} M_0$ be the sequence in which each transition occurs exactly once. S is consistent if and only if σ is alternating.*

4 The algorithm

In this section we present the complete algorithm for checking consistency.
Input: a well formed free-choice STG S, a signal a.
Output: 'consistent' or 'not consistent'.

(1) Check if S is non-autoconcurrent. If it is autoconcurrent, return 'not consistent', otherwise go to step 2.

(2) Check if S is balanced. If not, return 'non consistent', otherwise go to step 3.

(3) If S is a marked graph STG, construct a sequence $M_0 \xrightarrow{\sigma} M_0$ in which every transition occurs exactly once; if σ is alternating for every signal then return 'consistent', otherwise return 'not consistent'.
If S is not a marked graph, go to step 4.

(4) Select a CP-subnet $\overline{N}$ of S, and let its transitions occur as long as possible (each transition can occur at most once). Call this new STG (new because of the different initial marking) S'.

(5) Put one token on the entry place of $\overline{N}$, and execute a flushing sequence in which each transition occurs exactly once. Check if all signals alternate. If not, return 'not consistent', otherwise go to step 6.

(6) Remove from S' all internal places and transitions of $\overline{N}$. Let $S := S'_R$ as defined at the end of Section 3.3.1 (inserting witnesses where necessary), and go to step 3.

We show that the algorithm runs in $O((|P|+|T|)^3)$ time.
• The algorithm of [14] for Step (1) takes $O(|P| \cdot (|P| + |T|)^2)$ time.
• Step (2) requires to compute a basis of the solutions of the system $\mathbf{N} \cdot X = 0$, where $\mathbf{N}$ has dimension $|P| \times |T|$. This can be done in time $O(\mathcal{M}(\max\{|P|, |T|\}))$, where $\mathcal{M}(n)$ denotes the complexity of matrix multiplication, and so certainly in time $O((|P| + |T|)^3)$.
• The loop of the divide-and-conquer procedure can run for at most $|T|$ iterations, since each iteration reduces the number of output transitions of one place by 1. Moreover, at each step the STG S'_R has at most as many nodes as S (the inserted path of witnesses contains at most as many nodes as the removed CP-subnet).
• Steps (3) and (5) take $O((|P| + |T|)^2)$ time, even if we assume that it takes linear time to find an enabled transition.
• Step (4) takes $O((|P| + |T|)^2)$ time: there are at most $|T|$ candidates to be the entry transition of a CP-subnet, and we can check in $O(|P| + |T|)$ time for each of them if they indeed are (using Tarjan's algorithm for checking strong connectedness).
• Step (6) takes $O(|P| + |T|)$ time.

So the loop takes $O(|T| \cdot (|P| + |T|)^2)$, and the whole algorithm runs in $O((|P| + |T|)^3)$ time.

5 Conclusions

We have presented the first polynomial algorithm for checking consistency of well formed free-choice STGs. The correctness proof requires to use many results from Petri net theory. Almost all of the main theorems of the monograph [8] are directly or indirectly used. There could be a much simpler proof, although experience shows that results about free-choice nets often need long arguments.

Together with the techniques of [3] and [19], our algorithm can be used to produce free-choice STGs with consistent and complete encodings without having to construct their state graph, which could be exponentially larger than the STG itself.

We suspect that it is possible to get rid of the balancedness check at the price of a more complicated divide-and-conquer procedure. Exploring this is left for future research. The other obvious possibility for future work is to determine the computational complexity of completeness. We conjecture that it is also polynomial, but it seems to be a much harder problem.

The consistency problem is an instance of the more general problem of determining *synchronic distances* between sets of transitions of a Petri net (how often can transitions in one set occur without the others occurring?). These problems appear also in other applications of free-choice nets in the are of workflow processes [1]. We think that our results will also be useful there.

6 Acknowledgments

Many thanks to Jordi Cortadella and Alex Yakovlev for many helpful comments, pointers to the literature, and discussions. The author is very indebted to the anonymous referees for their careful reading and very helpful comments.

References

[1] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers, 8(1):21–66, 1998.

[2] C.H. van Berkel, M.B. Josephs and S.M. Nowick. Scanning the Technology: Applications of Asynchronous Circuits. Proceedings of the IEEE, vol. 87:2, 234–242, 1999.

[3] J. Carmona, J. Cortadella and Enric Pastor. A structural encoding technique for the synthesis of asynchronous circuits. Int. Conf. on Application of Concurrency Theory to System Design, 157–166, IEEE Computer Society, 2001.

[4] L.A. Cortés. Private communication, 2002.

[5] T.-A. Chu. Synthesis of Self-Timed VLSI Circuits from Graph-theoretic Specifications. P. D. Thesis, MIT, 1987.

[6] T.-A. Chu. On the models for designing VLSI asynchronous digital systems. Integration: the VLSI journal, 4:99-113, 1986.

[7] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev. Logic synthesis of asynchronous controllers and interfaces. Springer-Verlag, 2002.

[8] J. Desel and J. Esparza. Free Choice Petri Nets. Cambridge Tracts in Theoretical Computer Science 40, Cambridge University Press, 1995.

[9] J. Esparza. Decidability and complexity of Petri net problems - an introduction. Lectures on Petri Nets I: Basic Models. Advances in Petri Nets, LNCS 1491, 374-428, 1998.

[10] J. Esparza. Reduction and Synthesis of Live and Bounded Free Choice Petri Nets. Information and Computation 114(1), 50–87, 1994.

[11] J. Esparza. Full version of this paper, available from the authors's home page.

[12] A. Kondratyev, J. Cortadella, M. Kishinevsky, E. Pastor, O. Roig and A. Yakovlev. Checking Signal Transition Graph implementability by Symbolic BDD Traversal. Proc. of the European Design and Test Conference, 325–332, 1995.

[13] A. Kovalyov. Concurrency Relations and the Safety Problem for Petri Nets. Proc. of the 13th Application and Theory of Petri Nets 1992, LNCS 616, 299-309, 1992.

[14] A. Kovalyov and J. Esparza. A Polynomial Algorithm to Compute the Concurrency Relation of Free-choice Signal Transition Graphs. Proc. of the Int. Workshop on Discrete Event Systems, IEE, 1–6, 1996.

[15] L.H. Landweber and E.L. Robertson. Properties of Conflict-Free and Persistent Petri Nets. JACM 25(3), 352–364, 1978.

[16] L. Lavagno, C.W. Moon, R.K. Brayton and A. Sangiovanni-Vincentelli. Solving the state assignment problem for signal transition graphs. Proc. of the 29th IEEE/ACM Design Automation Conference, 568–572, IEEE Computer Society Press, 1992.

[17] K.-J. Lin, J.-W. Kuo, and C.-S. Lin. Direct synthesis of hazard-free asynchronous circuits from STGs based on lock relation and MG-decomposition approach. Proc. of the European Design and Test Conference, 178–183, 1994.

[18] R. Lipton. The Reachability Problem Requires Exponential Space. Technical Report 62, Yale University, 1976.

[19] E. Pastor, J. Cortadella, A. Kondratyev, and O. Roig. Structural Methods for the Synthesis of Speed Independent Circuits. IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems, 17:11, 1108–1129, 1998.

[20] W. Reisig. Petri Nets: An Introduction. EATCS Monographs in Computer Science 4, Springer, 1985.

[21] L.Y. Rosenblum and A.V. Yakovlev. Signal Graphs: from self-timed to timed ones. Proc. of the Int. Workshop on Timed Petri nets, 199-207, IEEE Computer Society, 1985.

[22] 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. on Information and Systems, E80-D(3):315–325, 1997.

[23] P. Vanbekbergen, G. Goosens, F. Catthoor and H. De Man. OPtimized Synthesis of Asynchronous Control Circuits from Graph-Theoretic Specifications. IEEE Trans. on Computer Aided Design, 11(11), 1426–1438, 1992.

[24] P. Vanbekbergen, B. Lin, G. Goosens, and H. De Man. A generalized state assignment theory for transformations on signal transition graphs. Proc. of the Int. Conf. on Computer Aided Design, 184-187, 1992.