Improved Parallel Composition of Labelled Petri Nets

Arseniy Alekseyev[1] Victor Khomenko[2] Andrey Mokhov[2] Dominic Wist[2] Alex Yakovlev[1]
[1]*School of Electrical, Electronic and Computer Engineering*
[2]*School of Computing Science*
Newcastle University, UK
{Arseniy.Alekseyev,Victor.Khomenko,Andrey.Mokhov,Dominic.Wist,Alex.Yakovlev}@ncl.ac.uk

Abstract

Parallel composition of labelled Petri nets is a fundamental operation in modular design. It is often used to combine models of subsystems into a model of the whole system. Unfortunately, the standard definition of parallel composition almost always yields a 'messy' Petri net, with many implicit places, causing performance deterioration in tools that are based on structural methods. In this paper we propose an optimised algorithm for computing the parallel composition. It often produces nets with fewer implicit places, which are thus better suited for subsequent application of structural methods.
Keywords: *parallel composition, re-synthesis, STG, asynchronous circuits.*

1. Introduction

Parallel composition (a.k.a. synchronous product) of labelled Petri nets is a fundamental operation in modular design. It is often used to combine models of subsystems into a model of the whole system. In particular, there is a nice correspondence between parallel composition of Signal Transition Graphs (STGs), a class of labelled Petri nets used for modelling asynchronous circuits, and connecting circuits by wires. Hence performing this operation efficiently is important in practice.

Unfortunately, the standard definition of parallel composition almost always yields a 'messy' Petri net, with many implicit places (even if the component Petri nets did not have them). Some of these places are easy to remove (e.g. duplicate places, which have the same pre- and postsets), but in general for removing others one needs full-blown model checking, which is infeasible if the resulting composition is large. Although implicit places do not have noticeable effect on tools based on state space exploration, such as PETRIFY [2], the performance of tools that are based on structural methods, such as DESIJ [12], often deteriorates.

Consider an example shown in Fig. 1, which shows the STG specifications of two components (a,b) and the specification of the environment (c). (The used shorthand drawing notation for STGs is explained in Sect. 2.) The model of the behaviour of the entire system can be obtained by constructing the parallel composition of these three STGs, which is shown in part (d) of this figure. One can see that it contains a few implicit places (which are not duplicate places); intuitively, they appear due to repeated causality specifications for every signal: the one coming from the component where this signal is an output, and others — from the components where it is an input. Removing these places yields a much 'cleaner' STG, coinciding with that shown in Fig. 2(d).

One operation where implicit places matter is *transition contraction,* [15] which is a crucial part of the re-synthesis approach [1, 8, 11]. The idea is to hide the internal communication between the components (by labelling the corresponding transitions as 'dummy' — they correspond to signals *a* and *b* in our example), contract as many of these dummy transitions as possible (whereby reducing the size of the STG), and re-synthesise the obtained STG as a circuit (which is often smaller than the original circuit due to removal of some signals). Transition contraction has to be performed on very large STGs (corresponding to the whole control path of the circuit), and so, for efficiency, it has to be a structural operation. Unfortunately, such structural contractions are not always possible (see Sect. 2), and implicit places in the preset and/or postset of a transition can prevent contracting it, even if a contraction is possible after removing these implicit places. In our example, DESIJ cannot contract any of the dummy transitions in the STG in Fig. 1(d), even though it performs some structural tests for place redundancy; however, it is able to contract all the dummy transitions if the implicit places are removed, i.e. when applied to the STG in Fig. 2(d).

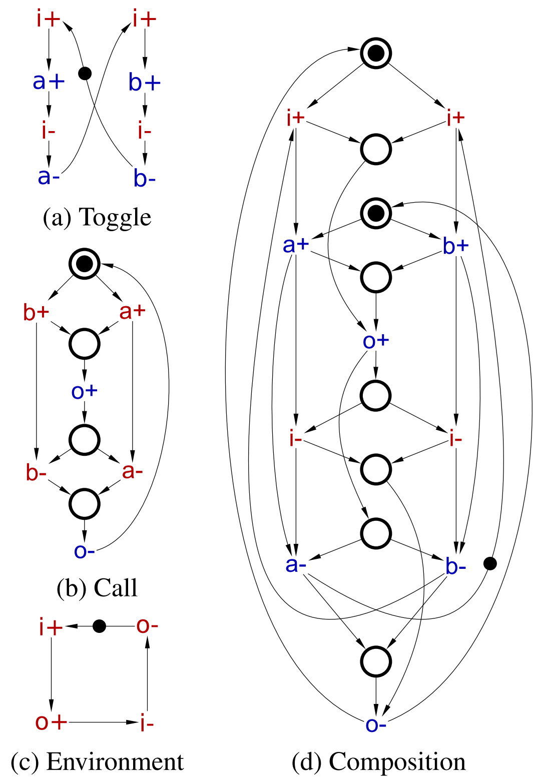


Figure 1. Example of standard STG composition.

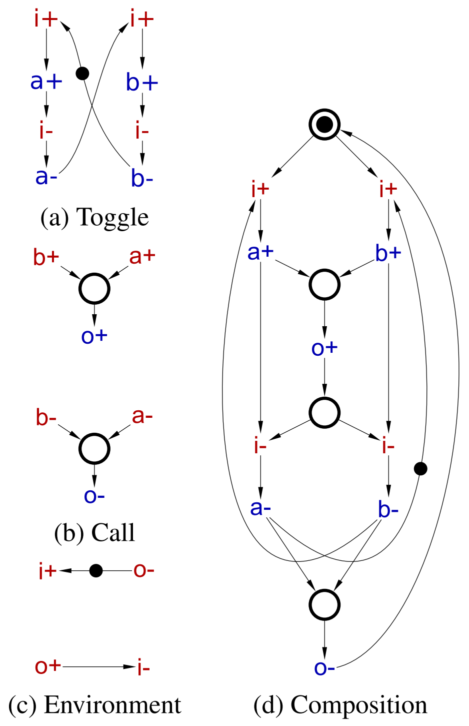


Figure 2. Example of improved STG composition: the components are obtained from the corresponding ones in Fig. 1 by removing some places, and then the standard parallel composition is applied to these modified components.

The main contribution of this paper is a new method for computing the parallel composition of labelled Petri nets, that generates fewer implicit places. It uses the *freeness from computation interference (FCI)* assumption, stating that the situation when one component wants to produce an output, but is prevented from doing so by another component which is not ready to receive it, is impossible. Violation of FCI means that the behaviour of the composition does not correspond to that of the physical system. For example, an output of a circuit component cannot be physically disabled by another component that is not ready to receive this signal, and so producing this output will lead to malfunction; however, the composition will be oblivious to it, and behave as if such an output could not be produced. Hence FCI is a basic correctness requirement — if it is violated, there is no point in computing parallel composition, as its behaviour will not describe that of the physical system. In practice, FCI is often guaranteed by construction, e.g. it is always guaranteed for the control path of a BALSA [4] or HASTE/TANGRAM [10, 14] specification of an asynchronous circuit. The idea of using the FCI condition is reminiscent of the method of input/output exposure in the synthesis by direct mapping described in [13], and of the correct by construction composition of Petri nets for circuit components and the environment used in the DI2PN tool [5].

The main idea of the method propose here is illustrated by the example in Fig. 2. Before doing the parallel composition, one can remove some of the places in the components as shown in parts (a–c) of the figure and then compose the modified STGs. The precise conditions that allow to remove a particular place will be stated in Sect. 3; at this point it is only important that they are structural and thus can be efficiently checked. This guarantees that the number of places in the resulting Petri net is smaller (as the number of places in the composition is the total number of places in all the components), and, under the FCI assumption, the resulting behaviour will be the same (in the sense of isomorphism of the reachability graphs). In particular, in our example, composing the modified components yields the STG in Fig. 2(d), which in this case contains no implicit places. Observe that the modified components on their own can have rather bad behaviour and in particular can be non-implementable; however, it does not matter, as they are never used on their own, but only in composition with other components, and the resulting behaviour of the composition is guaranteed to correspond to that of the standard composition.

Re-synthesis of asynchronous circuits is the intended application of the proposed method. However, we envisage that it has a much wider applicability, as composition of labelled Petri nets is a fundamental operation, and the FCI assumption often holds in practice.

2. Basic Definitions

A *Petri net* is a 4-tuple $N = (P, T, W, M_N)$ where P is a finite set of *places* and T is a finite set of *transitions* with $P \cap T = \emptyset$, $W : P \times T \cup T \times P \to \mathbb{N}_0$ is the *weight function,* and M_N is the *initial marking*, where a *marking* is a multiset of places, i.e. a function $P \to \mathbb{N}_0$ which assigns a number of *tokens* to each place. A Petri net can be considered as a bipartite graph with weighted arcs between places and transitions. If necessary, we write P_N etc. for the components of N or P' (P_i) etc. for the net N' (N_i) etc.

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

A transition t is *enabled under a marking M* if $\forall p \in {}^\bullet t : M(p) \geq W(p,t)$, which is denoted by $M[t\rangle$. An enabled transition t can *fire* yielding a new marking M', written as $M[t\rangle M'$, where $M'(p) = M(p) - W(p,t) + W(t,p)$, for all $p \in P$. A transition sequence $\sigma = t_1 \ldots t_n$ is *enabled under a marking M* (yielding M') if $M[t_1\rangle M_1 [t_2\rangle \ldots M_{n-1} [t_n\rangle M_n = M'$, and we write $M[\sigma\rangle$, $M[\sigma\rangle M'$ resp.; σ is called *execution of N* if $M_N[\sigma\rangle$. The empty transition sequence λ is enabled under every marking. M is called *reachable* if a transition sequence σ with $M_N[\sigma\rangle M$ exists.

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

A place p is *implicit* if it can be deleted from the net without changing the set of executions, and so an implicit place can be removed from the net without affecting its behaviour.[1] Unfortunately, detecting implicit places is expensive: the problem is PSPACE-complete for safe and EXPSPACE-complete for general Petri nets. A place p is *duplicate* if there is another place p' with the same pre- and postsets whose initial marking does not exceed that of p. Duplicate places are implicit, and are cheap to detect.

An *STG* is a tuple $N = (P, T, W, M_N, \mathit{In}, \mathit{Out}, \ell)$ where (P, T, W, M_N) is a Petri net and In and Out are disjoint sets of *input* and *output signals*. For $\mathit{Sig} = \mathit{In} \cup \mathit{Out}$ being the set of all signals, $\ell : T \to \mathit{Sig} \times \{+,-\} \cup \{\lambda\}$ is the *labelling* function. $\mathit{Sig} \times \{+,-\}$ or short $\mathit{Sig}^\pm$ is the set of *signal transitions*; its elements are denoted as s^+, s^- resp. instead of $(s,+)$, $(s,-)$ resp. A plus sign denotes that a signal value changes from *logical low* (written as 0) to *logical high* (written as 1), and a minus sign denotes the opposite direction. We write $s^\pm$ if it is not important or unknown which direction takes place.

An STG can contain transitions labelled with λ, called *dummy* transitions, which do not correspond to any signal change. *Hiding a signal s* means to change the label of all transitions labelled with $s^\pm$ to λ. (The idea of re-synthesis approach is to hide the signals used for communication between components, which results in an STG with fewer signals that often has a simpler implementation as a circuit.) The labelling of an STG is called *injective* if for each pair of distinct non-dummy transitions t and t', $\ell(t) \neq \ell(t')$.

Examples of STGs are shown in Figs. 1 and 2. Places are drawn as circles containing a number of tokens corresponding to the initial marking. Unmarked places which have only one transition in their presets and postsets are not drawn if the corresponding arcs have the weight 1; they are implicitly given by an arc between these two transitions (and if such a place contains tokens, they are drawn on the arc itself). Transitions are drawn simply as their labels, and the weight function is drawn as directed arcs (x,y) whenever $W(x,y) \neq 0$ (and labelled with $W(x,y)$ if $W(x,y) > 1$).

We lift the notion of enabledness to transition labels: we write $M[\ell(t)\rangle\rangle M'$ if $M[t\rangle M'$. This is extended to sequences as usual – deleting λ-labels automatically since λ is the empty word; i.e. $M[s^\pm\rangle\rangle M'$ means that a sequence of transitions fires, where one of them is labelled $s^\pm$ while the others (if any) are λ-labelled. A sequence $\nu \in (\mathit{Sig}^\pm)^*$ is called a *trace of a marking M* if $M[\nu\rangle\rangle$, and a *trace* of N if $M = M_N$. The *language $L(N)$ of N* is the set of all traces of N.

The *reachability graph* $\mathsf{RG}(N)$ of an STG N is an arc-labelled directed graph on the reachable markings of N with M_N as the root; there is an arc from M to M' labelled $\ell(t)$ whenever $M[t\rangle M'$. For bounded Petri nets and STGs, $\mathsf{RG}(N)$ can be seen as a finite automaton (where all states are accepting), and $L(N)$ is the language of this automaton. Observe that automata with

[1]Note that an implicit place can cease to be implicit if another implicit place is removed first.

accepting states only can be regarded as STGs (with the states as places, the initial state being the only marked place, etc.); hence, all definitions for STGs also apply to automata.

N is *deterministic* if $\mathsf{RG}(N)$ is a deterministic automaton: it contains no λ-labelled transitions and there are no *dynamic auto-conflicts,* i.e. for each reachable marking M and each signal transition $s^{\pm}$ there is at most one M' with $M[s^{\pm}\rangle\rangle M'$. (Note that a deterministic STG can have choices between different outputs, e.g. an STG modelling the standard arbiter is deterministic).

For deterministic STGs, language equivalence and bisimulation coincide, and the language can be taken as the semantics of such a specification. Unfortunately, the class of deterministic STGs is too restrictive in practice [6], e.g.:

- using dummy transitions is often convenient in manual design;
- modelling OR-causality [16] as a safe STG requires non-determinism;
- hiding internal communication (and thus introducing dummy transitions) is a crucial step in re-synthesis.

Hence, one has to deal with non-deterministic STGs as well.

One might think that if $\mathsf{RG}(N)$ is non-deterministic, it can be *determinised* (using well-known automata-theoretic methods), i.e. turned into a language-equivalent deterministic automaton with accepting states only; in particular, the resulting automaton will have no λ-arcs. Unfortunately, this is a bad idea, as shown in [6], where the semantics of non-deterministic STGs was developed. It is based on the concept of *output-determinacy,* which is a relaxation of determinism: An STG N is *output-determinate (OD)* if $M_N[v\rangle\rangle M_1$ and $M_N[v\rangle\rangle M_2$ implies for every $x \in Out_N$ that $M_1[x^{\pm}\rangle\rangle$ iff $M_2[x^{\pm}\rangle\rangle$. It turns out that OD STGs are exactly the STGs which have correct implementations according to the implementation relation introduced in [6]. Hence, non-OD STGs are ill-formed, and in particular cannot be correctly implemented as circuits. This shows that in general, *the language is not a satisfactory semantics of non-deterministic STGs;* in particular, *synthesising the determinised reachability graph of a non-OD STG will either fail or result in an incorrect circuit.* On the other hand, for the class of OD STGs [6] shows that their language is an adequate semantics, and implementation relation can be formulated purely in terms of the language. An important property of OD STGs is that in them the enabledness of an output signal is a function of the trace, i.e. given a trace v, the set of outputs by which v can be extended is uniquely determined, even though there could be multiple executions corresponding to v.

In the following definition of *parallel composition* $\|$, see e.g. [15], we will have to consider the distinction between input and output signals. The idea of parallel composition is that the composed systems run in parallel and synchronise on common actions – corresponding to circuits that are connected on the wires corresponding to the signals. Since a system controls its outputs, we cannot allow a signal to be an output of more than one component; input signals, on the other hand, can be shared. An output signal of a component may be an input of other components, and in any case it is an output of the composition.

The parallel composition of STGs N_1 and N_2 is defined if $Out_1 \cap Out_2 = \emptyset$. If we drop this requirement, the definition gives the *synchronous product* $N_1 \times N_2$, which is often useful. The place set of the composition is 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, and we will also write such a marking $M_1 \dot{\cup} M_2$ of the composition as (M_1, M_2). To define the transitions, let $A = (In_1 \cup Out_1) \cap (In_2 \cup Out_2)$ be the set of common signals. If e.g. s is an output of N_1 and an input of N_2, then firing of $s^{\pm}$ in N_1 is 'seen' by N_2, i.e. it must be accompanied by firing of $s^{\pm}$ in N_2. Since we do not know a priori which $s^{\pm}$-labelled transition of N_2 will fire together with some $s^{\pm}$-labelled transition of N_1, we have to allow for each possible pairing. Thus, the *parallel composition* $N = N_1 \parallel N_2$ is obtained from the disjoint union of N_1 and N_2 by fusing each $s^{\pm}$-labelled transition t_1 of N_1 with each $s^{\pm}$-labelled transition t_2 from N_2 if $s \in A$. Such transitions are pairs and the firing $(M_1, M_2)[(t_1, t_2)\rangle(M_1', M_2')$ of N corresponds to the firings $M_i[t_i\rangle M_i'$ in N_i, $i = 1, 2$; for an example of a parallel composition, see Fig. 3. More generally, we have $(M_1, M_2)[v\rangle\rangle(M_1', M_2')$ iff $M_i[v|_{N_i}\rangle\rangle M_i'$ for $i \in \{1, 2\}$, where $v|_{N_i}$ denotes the projection of the trace v onto the signals of the STG N_i. Hence, all reachable markings of N have the form (M_1, M_2), where M_i is a reachable marking of N_i, $i = 1, 2$.

Obviously, one can extend the notion of the parallel composition to a finite family (or collection) $(C_i)_{i \in I}$ of STGs as $\|_{i \in I} C_i$, provided that no signal is an output signal of more than one of the C_i. 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 $i \in I = \{1, ..., n\}$. As above, $(M_1, M_2, \ldots, M_n)[v\rangle\rangle(M_1', M_2', \ldots, M_n')$ iff $M_i[v|_{C_i}\rangle\rangle M_i'$ for all $i \in \{1, \ldots, n\}$. It is easy to see that C is deterministic if all C_i are. However, this is not true for a composition of OD STGs, as the result, in general, can be non-OD in such a case.

A composition can also be ill-defined due to *computation interference,* see e.g. [3]. Let $C \overset{\mathrm{df}}{=} \|_{i\in I} C_i$ be a composition of STGs. It is *free from computation interference (FCI)* if for every trace ν of C the following holds: if $\nu|_{C_j} x^{\pm}$ is a trace of C_j for some output x of C_j, then $\nu|_C x^{\pm}$ is a trace of C.

Transition contraction [15] is an important operation in circuit re-synthesis. It removes a dummy transition from an STG and combines each place of its preset with each place of its postset to 'simulate' the firing of the deleted transition, see Fig. 4. Unfortunately, transition contractions are sometimes undefined (e.g. in case the transition has a self-loop, i.e. some place occurs in both its preset and postset); moreover, even when a contraction is defined, it might change the semantics of the STG. Hence, [15] uses the notion of *secure* contractions, that preserve the semantics.

Transition contractions preserve boundedness, but in general, can turn a safe net into a non-safe one, as well as introduce weighted arcs. In practice, it is often convenient to work with safe nets, and for this [7] introduced *safeness-preserving* contractions, i.e. ones which guarantee that the transformed STG is safe if the initial one was. (Note that the transitions with weighted arcs must be dead in a safe Petri net, and so we can assume that the initial and all the intermediate STGs contain no such arcs.) Also, [7] developed a sufficient structural condition for a contraction to be safeness-preserving.

From the point of view of this paper, it is important to remark that implicit places can adversely affect the (secure) contractibility of a transition, i.e. it is possible to have a situation when a transition is not contractible (or not securely contractible), but becomes securely contractible after some implicit place is removed from the STG. As detecting implicit places is expensive, it is very desirable to reduce their number by some other means, in particular the approach proposed in this paper reduces the number of such places in STGs obtained by parallel composition. This has a direct effect on re-synthesis: if the composed STG has fewer implicit places, more dummy transitions in it can be contracted, and so it will be easier to synthesise the result.

3. Improved parallel composition

The improved parallel composition algorithm extends the conventional one by adding a pre-processing step, where some places are removed from the components, as they are guaranteed to be implicit in the result. To identify these places, one can note that a place is required in the final composition only if under some reachable marking it can be the only place that disables some transition in its postset.

For simplicity, consider the parallel composition $C = C_1 \parallel C_2$, whose components synchronise on a single signal s which is an output of C_1 and an input of C_2. Let (M_1, M_2) be a reachable marking of C, where M_1 and M_2 are some reachable markings of C_1 and C_2, respectively. Furthermore, suppose that M_1 enables, say, s^+ in C_1, where s is an output. Now, if M_2 does not enable s^+ in C_2, where s is an input, then there is computation interference. Therefore, if the FCI assumption holds, M_2 has to enable s^+ in C_2, i.e. whenever s^+ is enabled in C_1, it is also enabled in C_2. In other words, the firing of s^+ in C is fully controlled by C_1, and so *the constraints on firing of s that are present in C_2 can be ignored.* This means that the places in the preset of an s^+-labelled transition in C_2 will be implicit in the composition (subject to some technical conditions formulated below), and so can be removed before the composition is performed.

The above is true for the simple case of STGs with injective labelling and no dummies. However, the general picture is more complicated. In case of non-injective labelling, there can be multiple transitions corresponding to the same input signal transition, and the FCI assumption only guarantees the enabledness of one of them. Hence, some 'memory' (in the form of places) is required to trace which of these transitions has to be fired, which prohibits the removal of places from their presets. Furthermore, if the STG contains dummies, removing places from their postsets introduces some undesirable effects explained later. These considerations lead to the following conditions of applicability of the proposed optimisation.

Proposition 1. *Let $C \overset{\mathrm{df}}{=} \|_{i\in I} C_i$ be a composition of STGs that satisfies the FCI property and yields an output-determinate STG, and, for each $i \in I$, C'_i be the STG obtained from C_i by deleting all places p such that:*

1. *each transition $t \in p^\bullet$ is labelled with a signal, say s, and:*
   a) *s is an input;*
   b) *there is an STG C_j for which s is an output;*
   c) *there are at most one s^+- and at most one s^--labelled transition in C_i;*
2. *${}^\bullet p$ does not contain dummy transitions.*

Then $C' \overset{\mathrm{df}}{=} \|_{i\in I} C'_i$ and C have isomorphic reachability graphs.

Proof. First of all, observe that C' can be obtained from C by deleting some places. Hence, $\mathsf{RG}(C)$ is a subgraph

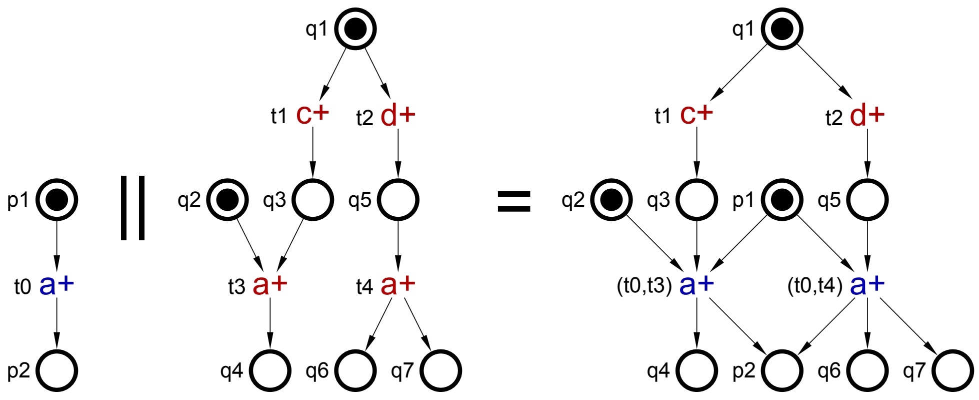


Figure 3. Parallel composition example. In the net fragment on the left hand side, signal a is an output, and in the fragment in the middle it is an input. Hence, in their parallel composition (right) it is an output. In this example, there is *computation interference*: the left component activates a^+ but the middle one is not ready to receive it.

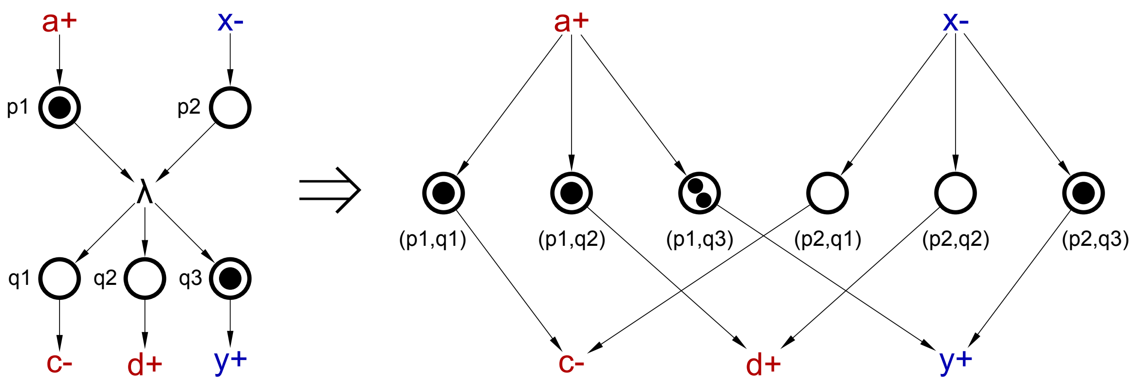


Figure 4. An example of a transition contraction.

of RG(C'), with the same initial state. Therefore it only remains to show that there are no additional states or arcs in the latter. For this, it is enough to show that for every state x of RG(C), the outgoing arcs of x are the same in both graphs. For the sake of contradiction, suppose there is a state x of RG(C) that has an outgoing arc a that is in RG(C') but not in RG(C).

The absence of a in RG(C) means that some of the deleted places p was in ${}^\bullet t$ for some transition t corresponding to the arc a in some of the component STGs C_i, and the number of tokens in this place at state x is smaller than the weight of the arc (p,t) in C_i (*). Since by condition 1a $p^\bullet$ can contain only input transitions, t must be labelled by $s^\pm$, where s is an input signal of C_i; wlog., we assume that the label is s^+. By condition 1b there is also a component STG C_j where s is an output signal.

Let σ be an execution of C terminating at state x, and ν be the trace corresponding to σ (note that such a σ always exists as the reachability graphs contain only reachable states). We proceed by showing that (i) $\nu|_{C_j}s^+$ is a trace of C_j and (ii) νs^+ is not a trace of C; these would mean that there is a violation of FCI in the original composition, leading to a contradiction.

(i) Since the arc a is present in RG(C'), the marking of C'_j corresponding to the global state x enables some s^+-labelled transition t_j in it. Since s is an output in C_j, no places were removed from ${}^\bullet t_j$ when building C'_j due to condition 1a, which means that t_j is also enabled by the marking of C_j corresponding to the global state x, and so $\nu|_{C_j}s^+$ is a trace of C_j.

(ii) For the sake of contradiction, suppose νs^+ is a trace of C. Due to the output-determinacy of C, the set of outputs by which ν can be extended is uniquely determined, and so s^+ must be enabled by x (perhaps, after firing several dummy transitions). By condition 1c there is only one s^+-labelled transition in C_i (viz. t), and so each s^+-labelled transition in C has p in its preset with the arc from p to this transition having the same weight as the arc (p,t) in C_i. Consequently, each s^+-transition in C is blocked at state x because by (*) the number of tokens in p is smaller than the weight of the corresponding arc. Moreover, firing only dummy transitions cannot increase the number of tokens in p and thus enable an s^+-labelled transition, as by condition 2 $^\bullet p$ contains no dummy transitions, a contradiction. Hence $v s^+$ is not a trace of C.

As explained above, (i) and (ii) imply a violation of FCI and so lead to a contradiction, which means that $\mathsf{RG}(C)$ and $\mathsf{RG}(C')$ must be isomorphic. □

We now discuss the conditions of Prop. 1 in more detail. The conditions 1a and 1b are intrinsic to the proposed method, and essentially state that due to the FCI assumption, firing of an input signal in a component can be controlled from the outside (viz. by the component controlling the corresponding output — whose existence is ensured by 1b), and so the component itself can get rid of the places controlling it.

The conditions 1c and 2 are technical restrictions on application of our method. If condition 1c is violated, there are several transitions that have the same label, say s^+ (where s is an input) in the component. When the corresponding output s^+ is produced by some other component, only one of these transitions should fire to match it — but to know which one, the component needs to control their firing, and so the places in their presets cannot be removed.

The necessity of condition 2 is illustrated by Fig. 5. Intuitively, the original STG on the left either receives a^+ followed by b^+ without outputting anything, or receives b^+ and produces x^+ in response. However, if the places in front of a^+ and b^+ are removed (which would be possible without condition 2), as shown on the right, then it might produce the unexpected x^+ after the trace $a^+ b^+$. Intuitively, in the initial STG firing of a^+ acts as an evidence that the dummy transition in the right branch has fired, while in the modified one the postset of this dummy transition has been removed, and so it is not possible anymore to guarantee that it has fired when a^+ fires.

In practice, when performing the parallel composition, one would like as few implicit places as possible in the result, and so it would be desirable to weaken the conditions in Prop. 1, so that as many places as possible are removed. As the conditions 1a and 1b are intrinsic, it is unlikely that they can be relaxed. However, the technical conditions 1c and 2 can be dealt with — by ensuring that the components always satisfy them. Indeed, as mentioned in Sect. 2, for output-determinate STGs the language is the semantics, and so one often can remove dummy transitions and enforce injective labelling without changing the language, e.g. using the PETRIFY tool [2]; this will ensure that conditions 1c and 2 hold.

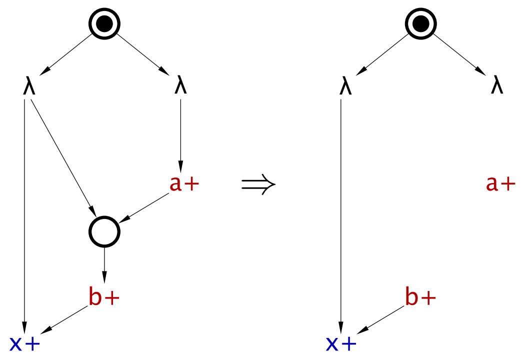


Figure 5. Example of an STG where removal of places in the postset of dummy transitions results in a wrong behaviour.

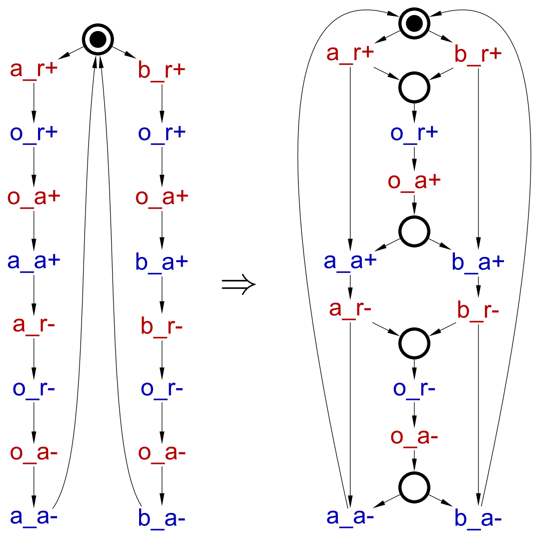


Figure 6. Example of enforcing injective labelling in an STG.

An example of such a transformation for the BALSA standard component Call is shown in Fig. 6. This operation is performed on (small) components rather than the (large) composition, and so is usually cheap. Moreover, in some applications, in particular circuit re-synthesis, the components are taken from a fixed library of component types, and so the transformation can be performed only once for each component type, and subsequently incur no runtime penalty at all.

4. Experiments

The proposed parallel composition algorithm has been evaluated on three series of scalable benchmarks (available from [9]), see Fig. 7. They are built of a subset of standard BALSA components [4]: Paral-

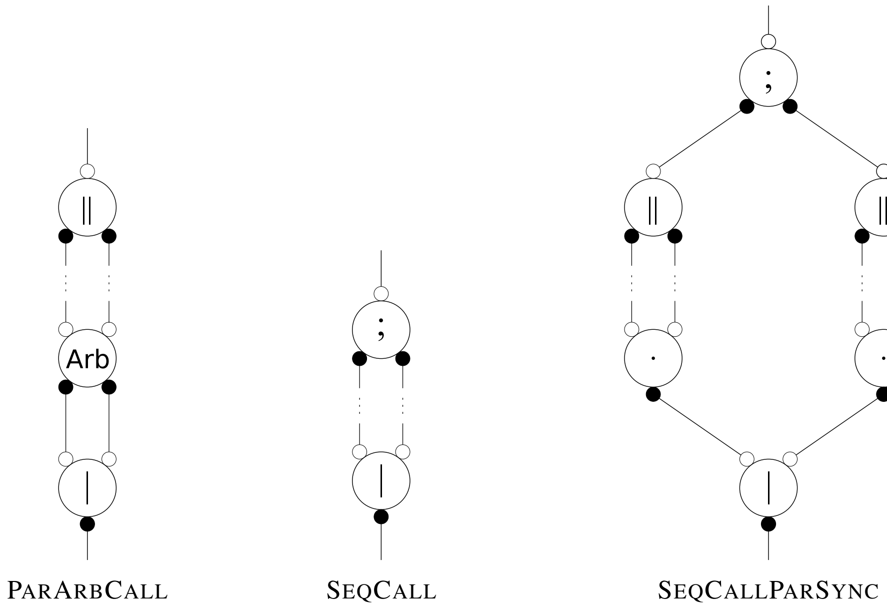


Figure 7. Scalable Balsa controllers used in experiments.

leliser (||), Sequencer (;), Call (|), Synchroniser (·) and Arbiter (Arb). These controllers are considered to be of size 1; a controller of size $k > 1$ is constructed by replacing the dotted lines with the controllers of size $k-1$. Each basic component is described by an individual STG; then these STGs are composed using four different techniques:

std the standard parallel composition;

opt the optimised parallel composition presented in this paper;

inj the standard parallel composition of the components with enforced injective labelling;

opt+inj the optimised parallel composition of the components with enforced injective labelling.

Note that all the used BALSA components except Call initially had injective labelling, so only the STG for Call was changed in the inj and opt+inj series. Both the standard and optimised parallel composition algorithms have been implemented in PCOMP tool [9]. The tool automatically deletes duplicate places in all compositions, so all the experimental results are subject to this simplification. The runtimes of PCOMP were negligible and so not reported.

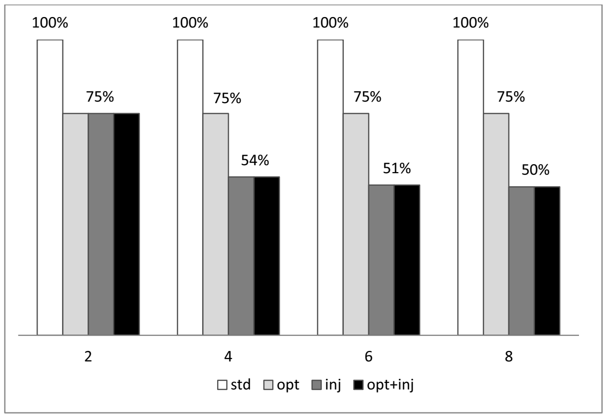
For each composed STG, the internal signals of the composition were hidden (i.e. turned into dummies), and the DESIJ tool [12] was used to structurally eliminate as many dummies as possible, using either secure or safeness-preserving secure contractions.

The results of our experiments are summarised by the charts in Fig. 8. There are six charts altogether, for each of the three benchmark series and each of the contraction modes (secure or safeness-preserving secure). Each chart reports, for each benchmark size within the corresponding series, the numbers of non-contractible dummy transitions (normalised w.r.t. the worst result) remaining in the STG for each of the four composition methods described above.

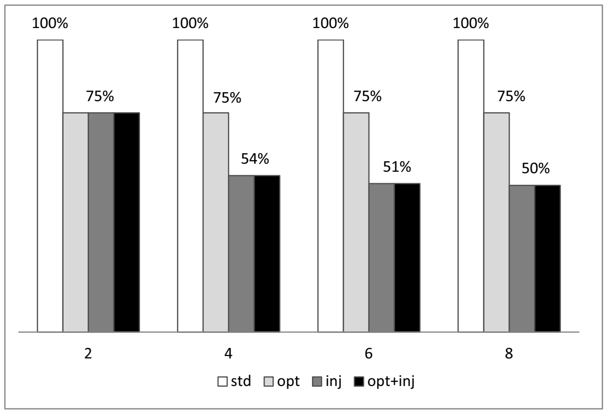
The experiments demonstrate that the optimised parallel composition is never worse than the standard one in terms of the STG structure that is used for removing dummies (the opt bars are never longer than the std bars, and the opt+inj bars are never longer than the inj bars), and is significantly better in some cases (e.g. for the SEQCALLPARSYNC (4) benchmark there is a factor of five improvement). Moreover, using the optimised technique in conjunction with injective labelling is usually advantageous (the fourth bar is the shortest in almost all cases).

5. Conclusion

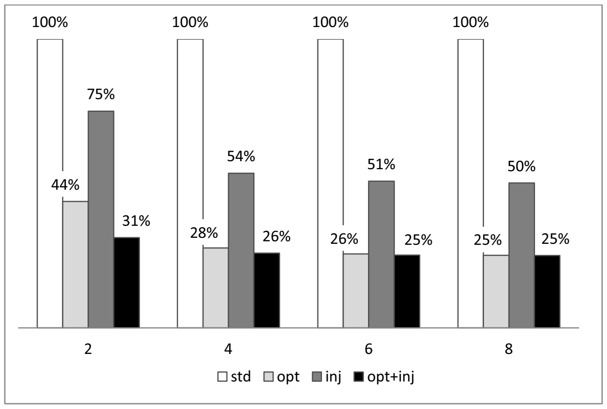
We have presented an improved algorithm for computing the parallel composition of STGs or labelled Petri nets. Under the FCI assumptions, it allows to produce nets with fewer implicit places, which aids the subsequent structural algorithms like dummy contraction. It uses only simple structural checks and thus is very efficient even for large compositions, so the improvement comes at negligible cost.



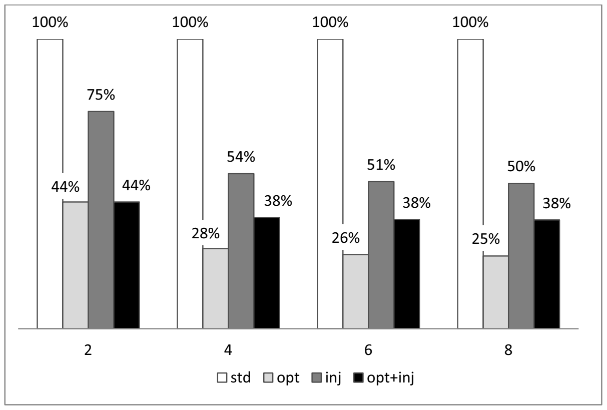
PARARBCALL: all contractions



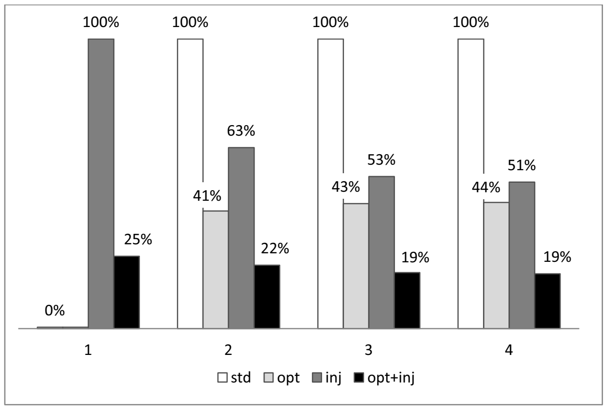
PARARBCALL: safeness-preserving contractions



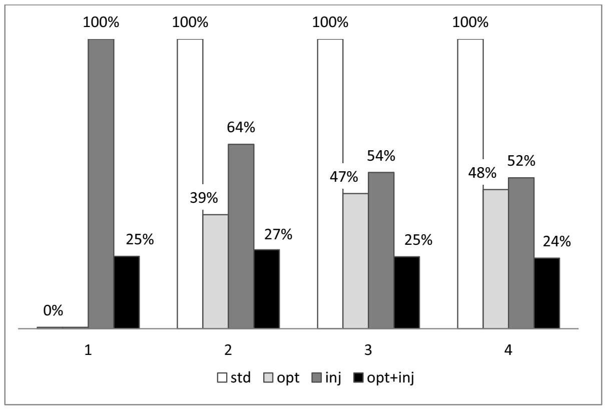
SEQCALL: all contractions



SEQCALL: safeness-preserving contractions



SEQCALLPARSYNC: all contractions



SEQCALLPARSYNC: safeness-preserving contractions

Figure 8. Experimental results.

The algorithm was implemented in the PComp tool and evaluated on a set of scalable benchmarks. The experiments proved its efficiency, which increases even more when the components are pre-processed to remove dummies and ensure injective labelling (this is usually cheap, as the components are small; moreover, if the components come from a standard library of component types, this step can be completely eliminated).

Another important advantage is that the improved algorithm places almost no additional effort on the user: the only requirement is to pass an additional command-line option to PComp so that it can assume the FCI property and apply the proposed optimisation.

Acknowledgements. This work was supported by EPSRC grant EP/G037809/1 (Verdad).

References

[1] T. Chelcea and S. Nowick. Resynthesis and peephole transformations for the optimization of large-scale asynchronous systems. In *Proc. DAC'02*, pages 405–410, 2002.

[2] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavag no, 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.

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

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

[5] M. Josephs and D. Furey. Delay-insensitive interface specification and synthesis. In *Proc. DATE'00*, pages 169–173, 2000.

[6] V. Khomenko, M. Schaefer, and W. Vogler. Output-determinacy and asynchronous circuit synthesis. *Fundamenta Informaticae*, 88(4):541–579, 2008. Special Issue on Best Papers from ACSD'07.

[7] V. Khomenko, M. Schaefer, W. Vogler, and R. Wollowski. STG decomposition strategies in combination with unfolding. *Acta Informatica*, 46(6):433–474, 2009.

[8] T. Kolks, S. Vercauteren, and B. Lin. Control resynthesis for control-dominated asynchronous designs. *Proc. ASYNC'96*, pages 233–243, 1996.

[9] PComp tool. URL: `http://homepages.cs.ncl.ac.uk/victor.khomenko/tools/pcomp`.

[10] A. Peeters and M. de Wit. Haste *Manual, v. 3.0*. Handshake Solutions, 2005. URL: http://handshakesolutions.com/Technology/Haste/Article-14902.html.

[11] M. Pena and J. Cortadella. Combining process algebras and Petri nets for the specification and synthesis of asynchronous circuits. In *Proc. ASYNC'96*, pages 222–232. IEEE Computer Society, 1996.

[12] M. Schaefer. DesiJ— a tool for decomposition. Technical Report 2007-11, University of Augsburg, 2007. URL: `http://www.informatik.uni-augsburg.de/de/forschung/reports/`.

[13] D. Sokolov, A. Bystrov, and A. Yakovlev. Direct mapping of low-latency asynchronous controllers from STGs. *IEEE Trans. CAD*, 26(6):993–1009, 2007.

[14] K. van Berkel, J. Kessels, M. Roncken, R. Saeijs, and F. Schalij. The VLSI-programming language Tangram and its translation into handshake circuits. In *Proc. European Conference on Design Automation (EDAC)*, 1991.

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

[16] A. Yakovlev, M. Kishinevsky, A. Kondratyev, L. Lavag no, and M. Pietkiewicz-Koutny. On the models for asynchronous circuit behaviour with OR causality. *Formal Methods in System Design*, 9:189–233, 1996.