# Synthesis of Reactive Systems: Application to Asynchronous Circuit Design*

Josep Carmona[1], Jordi Cortadella[2], and Enric Pastor[1]

[1] Universitat Politècnica de Catalunya, Computer Architecture Department, Barcelona, Spain
[2] Universitat Politècnica de Catalunya, Software Department, Barcelona, Spain

**Abstract.** Synthesis bridges the gap between specification and implementation by systematically transforming one model into another and approaching the primitives of the specification to those realizable by the implementation. This work faces the problem of the synthesis of reactive systems, in which there is an explicit distinction between input and output actions. The transformations used during synthesis must preserve the properties that characterize the correct interaction between the system and the environment. The concept of *I/O compatibilty* is proposed to define this correctness, and is used to provide a set of transformations for the synthesis of reactive systems.

The theoretical contributions of the work are applied to the synthesis of asynchronous circuits. Petri nets are used as specification model, and a set of structural methods are proposed for the transformations of the model and the synthesis of digital circuits.

## 1 Introduction

A reactive system is a concurrent system in which the behavior explicitly manifests an interaction with the environment. Unlike other models to express concurrency, reactive systems have a clear distinction between the actions performed by the environment (input events) and those performed by the system (output events). Additionally, a reactive system may also have internal actions not observable by the environment.

I/O automata [15] were proposed as a model for discrete event systems in which different components interact asynchronously. The distinction between input, output and internal events is fundamental to characterize the behavior of real-life systems. While a system can impose constraints on when to perform output or internal actions, it cannot impose any constraint on when an input action can occur. For this reason, input actions are always enabled in any state of an I/O automata.

In this work, we deal with the synthesis problem of reactive systems. Given a system and its environment, a synchronization protocol is committed

* This work has been partially funded by the Ministry of Science and Technology of Spain under contract TIC 2001-2476, ACiD-WG (IST-1999-29119), a grant by Intel Corporation and CIRIT 2001SGR-00254.

in such a way that, at any state, the environment is guaranteed to produce only those input actions acceptable by the system. For the specification of this type of systems, one might use I/O automata with a dummy state that is the sink of any input action not acceptable by the system.

We want to solve the following problem: given the specification of a reactive system, generate an implementation realizable with design primitives that commits the protocol established with the environment. In this particular work, we focus on the synthesis of asynchronous circuits, where the events of the system are rising and falling signal transitions and the design primitives are logic gates.

The specification formalism used for the synthesis of asynchronous circuits is Petri nets [17], an event based model that can express concurrency explicitly. For this reason, the state explosion problem may arise when state-level information is required for synthesis. This paper will tackle this problem by using structural techniques for synthesis.

### 1.1 Overview

A reactive system is assumed to interact with an enviroment (see Figure 1(a)). Both, the system and the environment have behaviors that can be described with some model for concurrent systems.

A reactive system has an alphabet of input, output and internal events $\Sigma = \Sigma_I \cup \Sigma_O \cup \Sigma_{INT}$. The environment also has another alphabet $\Sigma' = \Sigma'_I \cup \Sigma'_O \cup \Sigma'_{INT}$. A necessary condition for a system and an environment to talk to each other is that their alphabets can be "connected", i.e. $\Sigma_I = \Sigma'_O$ and $\Sigma_O = \Sigma'_I$, as shown in Figure 1(a). This paper is an attempt to formalize and answer questions such as

> *Which is the class of systems that can correctly dialogue with a given environment?*
> *How can a system be transformed so that the dialogue with its environment is not affected?*
> *What does "correct dialogue" mean?*

To understand the rest of this section, we will give an informal and intuitive idea of what "correct dialogue" means. We will say that two systems have a correct dialogue when

- every output event produced by one of them is expected by the other as an input event (safeness), and
- if one of them is only waiting for input events, the other one will eventually produce one of them (liveness).

Figures 1(b-d) depict an example to illustrate the main issues discussed in this paper. The models used to describe behavior are marked graphs, a subclass of Petri nets with no choice places, in which events represent

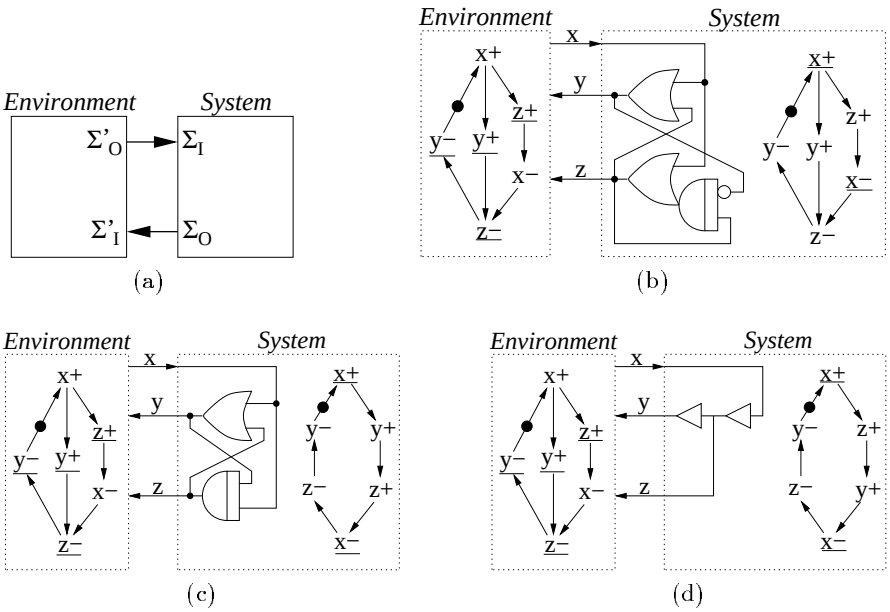



**Fig. 1.** (a) Connection between system and environment, (b) mirrored implementation of a concurrent system, (c) valid implementation with concurrency reduction, (d) invalid implementation.

rising (+) or falling (-) transitions of digital signals. The goal is to synthesize a circuit that can have a correct dialogue with the environment. We will assume that the components of the circuit have arbitrary delays. Likewise, the environment may take any arbitrary delay to produce any enabled output event.

Let us first have a look at Figure 1(b). The marked graph in the environment can be considered as a specification of a concurrent system. The underlined transitions denote input events. Thus, an input event of the environment must have a correspondence with an output event of the system, and vice versa. The behavior denoted by this specification can be informally described as follows:

> *In the initial state, the environment will produce the event $x+$. After that, the environment will be able to accept the events $y+$ and $z+$ concurrently from the system. After the arrival of $z+$, the environment will produce $x-$, that can occur concurrently with $y+$. Next, it will wait for the system to sequentially produce $z-$ and $y-$, thus leading the environment back to the initial state.*

The circuit shown in Figure 1(b) behaves as specified by the adjacent marked graph. In this case, the behavior of the system is merely a mirror of

the behavior of the enviroment. For this reason, the dialogue between both is correct.

Let us analyze now the system in Figure 1(c). In this case, the circuit implements a behavior in which $y+$ and $z+$ are produced sequentially. Still, the system can maintain a correct dialogue, since the environment is able to accept more behaviors than the ones produced by the system. We can observe that, even though the behavior is less concurrent, the implementation is simpler.

Let us finally look at Figure 1(d), in which the events $z+$, $y+$ and $x-$ are produced sequentially in this order. Due to this reduction in concurrency, two buffers are sufficient to implement such behavior. Even though the set of traces produced by the system is included in the set of traces produced by the environment, the dialogue between both is not correct. To illustrate that, let us assume the events $x+$ and $z+$ have been produced from the initial state. We are now in a state in which $x-$ is enabled in the environment (output event) but not enabled in the system. This violates one of the conditions for a correct dialogue: if an output event is enabled in one component, the corresponding event must also be enabled in the other component. In practice, if the environment would produce $x-$, the circuit could react with the event $z-$ before $y+$ is produced.

The previous examples suggest that a theory to characterize the fact that two systems can talk to each other is required. We will call *Input/Output-compatibility* this characterization. We will show that the well-known concepts of bisimulation and observational equivalence [16] are not appropriate to analyze properties related to the dialogue between systems, since there is no explicit distinction between input and output events.

Finally, the theory presented in this paper will be applied to a specific area of hardware design: asynchronous circuits. We will provide a kit of transformations that will assist in improving the quality of gate implementations by either reducing the complexity of the circuit or its performance.

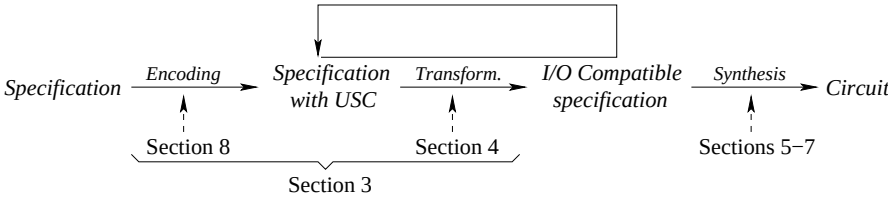



**Fig. 2.** Synthesis framework for asynchronous circuits.

Figure 2 depicts the synthesis framework for asynchronous circuits proposed in this work. It is also used to introduce the sections of this paper. Initially, a concurrent specification is given to express the protocol established between system and environment. The specification must be trans-

formed to properly encode the states of the system and guarantee a logic implementation. After the transformations required for encoding (Section 8), the specification is said to have the *Unique State Coding* property. Synthesis is performed by transforming the specification in such a way that the behavior is realizable by logic gates. In order to circumvent the state explosion problem, the transformations are performed on the structure of the specification, and not on its state-level representation (Section 4). In either case, the transformations performed for encoding and synthesis must preserve the correctness of the interaction with the environment. This correctness is called *I/O compatibility* and it is presented and discussed in Section 3. Some basic theory on the synthesis of asynchronous circuits is presented in Sections 5 and 6. The synthesis of speed-independent circuits by using structural methods is described in Section 7. Finally, Section 9 presents the complete design flow and some experimental results.

## 2 Models for reactive systems

Reactive systems [10] are systems that operate in a distributed environment. The events observed in a reactive system can be either input events, output events or internal events. An input event represents a change in the environment for which the system must react. In contrast, an output event can force other systems in the environment to react to. Finally, an internal event represents system's local progress, not observable in the environment. Typical examples of reactive system are a computer, a television set and a vending machine. The events executed in a reactive system are assumed to take arbitrary but finite time.

Two models for the specification of reactive systems are presented in this section: Transition systems and Petri nets.

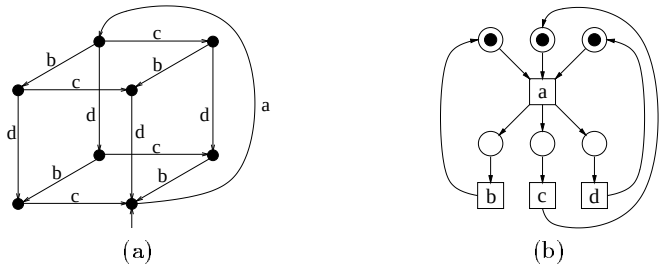
### 2.1 Transition Systems

**Definition 1 (Transition System).** A Transition System (TS) [1] is a *4-tuple* $A = (S, \Sigma, T, s_{in})$ where

- $S$ is the set of states
- $\Sigma$ is the alphabet of events
- $T \subseteq S \times \Sigma \times S$ is the set of transitions
- $s_{in} \in S$ is the initial state

Figure 3(a) depicts an example of TS. The initial state is denoted by an incident arc without source state.

**Reachability in a TS** The transitions are denoted by $(s, e, s')$ or $s \xrightarrow{e} s'$. An event is said to be *enabled* in the state $s$, denoted by the predicate $\mathsf{En}(s, e)$, if $(s, e, s') \in T$, for some $s'$. The *reachability relation* between




(a) (b)

**Fig. 3.** (a) Transition System. (b) Petri net.

states is the transitive closure of the transition relation $T$. The predicate $s \xrightarrow{\sigma} s'$ denotes a *trace* of events $\sigma$ that leads from $s$ to $s'$ by firing transitions in $T$. A state $s$ is *terminal* if no event is enabled in $s$. A TS is *finite* if $S$ and $T$ are finite sets.

**Language of a TS** A TS can be viewed as an automaton with alphabet $\Sigma$, where every state is an accepting state. For a TS $A$, let $L(A)$ be the corresponding language, i.e. its set of traces starting from the initial state.

The synchronous product of two transition systems is a new transition system which models the interaction between both systems [1].

**Definition 2 (Synchronous Product).** Let $A = (S^A, \Sigma^A, T^A, s^A_{in})$, $B = (S^B, \Sigma^B, T^B, s^B_{in})$ be two TSs. The synchronous product of $A$ and $B$, denoted by $A \times B$ is another TS $(S, \Sigma, T, s_{in})$ defined by

- $s_{in} = \langle s^A_{in}, s^B_{in} \rangle \in S$
- $\Sigma = \Sigma^A \cup \Sigma^B$
- $S \subseteq S^A \times S^B$ is the set of states reachable from $s_{in}$ according to the following definition of $T$.
- Let $\langle s1, s1' \rangle \in S$.
  - If $e \in \Sigma^A \cap \Sigma^B$, $s_1 \xrightarrow{e} s_2 \in T^A$ and $s'_1 \xrightarrow{e} s'_2 \in T^B$, then $\langle s_1, s'_1 \rangle \xrightarrow{e} \langle s_2, s'_2 \rangle \in T$
  - If $e \in \Sigma^A \setminus \Sigma^B$ and $s_1 \xrightarrow{e} s_2 \in T^A$, then $\langle s_1, s'_1 \rangle \xrightarrow{e} \langle s_2, s'_1 \rangle \in T$
  - If $e \in \Sigma^B \setminus \Sigma^A$ and $s'_1 \xrightarrow{e} s'_2 \in T^B$, then $\langle s_1, s'_1 \rangle \xrightarrow{e} \langle s_1, s'_2 \rangle \in T$
  - No other transitions belong to $T$

When the events of a TS are interpreted as events of a reactive system, the notion of *reactive transition system* arise:

**Definition 3 (Reactive Transition System).** A Reactive Transition System (RTS) is a TS $(S, \Sigma, T, s_{in})$ where $\Sigma$ is partitioned into three pairwise disjoint subsets of input $(\Sigma_I)$, output $(\Sigma_O)$ and internal $(\Sigma_{INT})$ events. $\Sigma_{OBS} = \Sigma_I \cup \Sigma_O$ is called the set of observable events.

**Properties of Reactive Systems** Some definitions depending on the interpretation of the events arise in a reactive transition system.

**Definition 4 (Livelock).** A livelock is an infinite trace of only internal events. A RTS is *livelock-free* if it has no livelocks.

**Definition 5 (Input-proper).** An RTS is *input-proper* when for every internal transition $s \xrightarrow{e} s'$, with $e \in \Sigma_{INT}$ and for every input event $i \in \Sigma_I$, $\mathsf{En}(s', i) \implies \mathsf{En}(s, i)$. In other words, whether or not an input event is enabled in a given state depends only on the observable trace leading to that state.

**Definition 6 (Mirror operator).** The *mirror* of $A$, denoted by $\overline{A}$, is another RTS identical to $A$, but in which the input and output alphabets of $A$ have been interchanged.

### 2.2 Petri Nets

Petri Nets [20,17] is a formal model for the specification, analysis and synthesis of concurrent systems. The basis of the model is the causality relation established between the set of events.

**Definition 7 (Petri Net).** A Petri Net (PN) is a *4-tuple* $N = (P, T, F, M_0)$ where

- $P$ is the set of places
- $T$ is the set of transitions
- $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ is the flow relation
- $M_0 : P \to \mathbb{N}$ is the initial marking

An example of PN is shown in Fig. 3(b).

**Paths** A *path* in a PN is a sequence $u_1 \ldots u_r$ of nodes such that $\forall i, 1 \leq i < r : (u_i, u_{i+1}) \in F$. A path is called *simple* if no node appears more than once on it.

**Reachability in a PN** Given a node $x$ of $N$, the set ${}^\bullet x = \{y \mid (y, x) \in F\}$ is the pre-set of $x$ and the set $x^\bullet = \{y \mid (x, y) \in F\}$ is the post-set of $x$. A transition $t$ is *enabled* in marking $M$ if each place $p \in {}^\bullet t$ is marked with at least $F(p, t)$ tokens. When a transition $t$ is enabled, it can *fire* by removing $F(p, t)$ tokens from place $p \in {}^\bullet t$ and adding $F(t, q)$ tokens to each place $q \in t^\bullet$. A marking $M'$ is *reachable* from $M$ if there is a sequence of firings $t_1 t_2 \ldots t_n$ that transforms $M$ into $M'$, denoted by $M[t_1 t_2 \ldots t_n\rangle M'$. A sequence of transitions $t_1 t_2 \ldots t_n$ is a *feasible sequence* if it is firable from $M_0$. The set of reachable markings from $M_0$ is denoted by $[M_0\rangle$. A marking is a *home marking* if it is reachable from every marking of $[M_0\rangle$. Let $R \subseteq [M_0\rangle$ be the set of markings where transition $t_i$ is enabled.

Transition $t_j$ *triggers* transition $t_i$ if there exists a reachable marking $M$ such that $M[t_j\rangle M'$, $M \notin R$ and $M' \in R$. Transition $t_j$ *disables* transition $t_i$ if there exists a reachable marking $M$ enabling both $t_i$ and $t_j$, but in the marking $M'$ such that $M[t_j\rangle M'$, $t_i$ is not enabled.

**Petri Net subclasses** A place in a PN is *redundant* if its elimination does not change the behavior of the net. A PN is *place-irredundant* if it does not have redundant places. A Petri net is called *ordinary* when for every pair of nodes $(x, y)$, $F(x, y) \leq 1$. All the Petri nets appearing in this chapter are ordinary[1]. If restrictions are imposed on the structure of the net, several subclasses can be defined [17]. Three subclasses are of interest in this paper:

- A *State Machine* (SM) is a PN such that each transition has exactly one input place and one output place.
- A *Marked Graph* (MG) is a PN such that each place has exactly one input transition and one output transition.
- A *Free-choice Petri net* (FC) is a PN such that if $(p, t) \in F$ then ${}^\bullet t \times p^\bullet \subseteq F$, for every place $p$.

**Liveness and safeness** A PN is *live* iff every transition can be infinitely enabled through some feasible sequence of firings from any marking in $[M_0\rangle$. A PN is *safe* if no marking in $[M_0\rangle$ assigns more than one token to any place. In the rest of the chapter, live and safe Petri nets will be assumed.

**Free-choice decomposition** A free-choice live and safe Petri net (FCLSPN) can be decomposed into a set of strongly-connected state-machines (marked graphs). An *SM-cover* (*MG-cover*) of a FCLSPN is a subset of state machines (marked graphs) such that every place (transition) is included at least in one state machine (marked graph). Moreover, a FCLSPN can be also decomposed into a set of strongly-connected *one-token* state-machines, i.e. state-machines that at most contain one token at any reachable marking [9].

A FCLSPN is shown in Figure 4(a). A one-token SM-cover of PN of Figure 4(a) is shown in Figure 4(b).

**Concurrency relations** The Concurrency Relation [5] between pairs of nodes $(P \cup T)$ of a PN is defined as a binary relation $\mathcal{CR}$, such that given places $p_i, p_j$ and transitions $t_i, t_j$:

$$(t_i, t_j) \in \mathcal{CR} \Leftrightarrow [\exists M \in [M_0\rangle : M[t_i t_j\rangle \wedge M[t_j t_i\rangle];$$
$$(p, t_i) \in \mathcal{CR} \Leftrightarrow [\exists M \in [M_0\rangle : M[t_i\rangle M' \wedge M(p) > 0 \wedge M'(p) > 0];$$
$$(p_i, p_j) \in \mathcal{CR} \Leftrightarrow [\exists M \in [M_0\rangle : M(p_i) > 0 \wedge M(p_j) > 0].$$

[1] Given that we deal with ordinary nets, we will abuse language and say $e \in F$ instead of $F(e) = 1$.

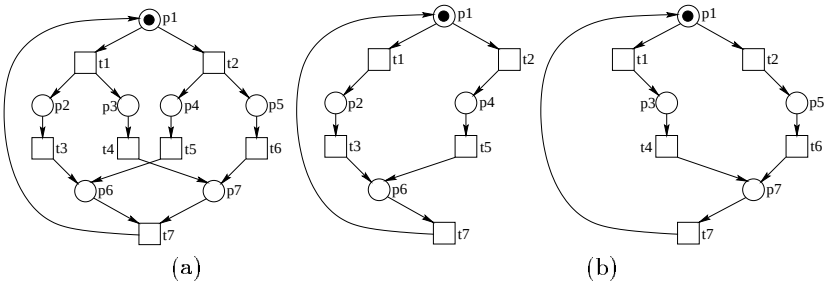



**Fig. 4.** (a) Free-choice petri net. (b) SM-cover of PN from (a).

Polynomial algorithms for the computation of the concurrency relations of a FCLSPN have been presented in [13].

As in the case of the TS model, a transition in a PN can represent an event occurring in a reactive system.

**Definition 8 (Reactive Petri Net).** A Reactive Petri Net (RPN) is a *3-tuple* $((P, T, F, M_0), \Sigma, \Lambda)$ where

- $(P, T, F, M_0)$ is a PN
- $\Sigma$ is a set of events defined as in the case of RTS
- $\Lambda : T \rightarrow \Sigma$

A RPN $A$ has an associated reactive transition system RTS($A$), in which each reachable marking corresponds to a state and each transition $t$ between a pair of markings to an arc labeled with $\Lambda(t)$. A RPN $A$ is *deterministic* if RTS($A$) is deterministic. Along this paper, only deterministic RPNs are considered.

When restrictions are imposed on the structure of a RPN, several subclasses can be defined. The IO-RPN class contains those free-choice deterministic RPNs fulfilling both that there is not an input event triggering another input event, and the transitions in the post-set of a *choice* place are all input events:

**Definition 9 (I/O Reactive Petri Net).** An IO-RPN is a deterministic free-choice RPN where the following conditions hold:

1. $\forall t_1, t_2 \in T : (t_2 \in (t_1^\bullet)^\bullet \wedge \Lambda(t_1) \in \Sigma_I \Rightarrow \Lambda(t_2) \notin \Sigma_I)$.
2. $\forall p \in P : (|p^\bullet| \geq 2 \Rightarrow \forall t \in p^\bullet : \Lambda(t) \in \Sigma_I)$.

Figure 5(a) presents an example of IO-RPN. The RPN of Figure 5(b) does not belongs to the IO-RPN class because input transition $a$ triggers input transition $b$. The IO-RPN class possesses some nice properties, which will be discussed in Sections 8 and 9.

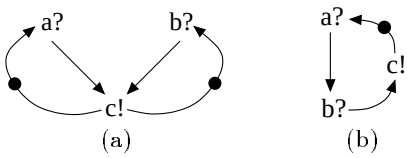



**Fig. 5.** (a) IO-RPN specification. (b) RPN which is not IO-RPN (the suffices ? and ! are used to denote input and output events, respectively).

## 3 Relations of reactive systems

In the first part of this section, the *observational equivalence* relation [16] is introduced. Afterwards, a set of conditions that ensure a correct dialogue between two reactive systems is presented: the *I/O compatibility*. Both notions are compared and the relations among them are outlined. Finally, the I/O compatibility is used for defining the set of conditions establishing when an object can be considered as a correct realization of a reactive system (*I/O preserving realization*).

### 3.1 Observational equivalence

The *observational equivalence* relation between two reactive systems was first introduced by Milner in [16]. The relation identifies those systems whose observable behavior is identical.

**Definition 10.** Let $A = (S^A, \Sigma^A, T^A, s^A_{in})$ and $B = (S^B, \Sigma^B, T^B, s^B_{in})$ be two RTSs. $A$ and $B$ are *observational equivalent* ($A \approx B$) if $\Sigma^A_{OBS} = \Sigma^B_{OBS}$ and there exists a relation $R \subseteq S^A \times S^B$ satisfying

1. $s^A_{in} R s^B_{in}$.
2. (a) $\forall s \in S^A$, $\exists s' \in S^B$ s.t. $sRs'$.
   (b) $\forall s' \in S^B$, $\exists s \in S^A$ s.t. $sRs'$.
3. (a) $\forall s_1 \in S^A$, $s'_1 \in S^B$: if $s_1 R s'_1$, $e \in (\Sigma^A_{OBS})$ and $s_1 \xrightarrow{e} s_2$ then $\exists \sigma_1, \sigma_2 \in (\Sigma^B_{INT})*$ such that $s'_1 \xrightarrow{\sigma_1 e \sigma_2} s'_2$, and $s_2 R s'_2$.
   (b) $\forall s_1 \in S^A$, $s'_1 \in S^B$: if $s_1 R s'_1$, $e \in (\Sigma^A_{OBS})$ and $s'_1 \xrightarrow{e} s'_2$ then $\exists \sigma_1, \sigma_2 \in (\Sigma^A_{INT})*$ such that $s_1 \xrightarrow{\sigma_1 e \sigma_2} s_2$, and $s_2 R s'_2$.

The two RTSs of Figure 6(a) are observational equivalent, because every observable sequence of one of them can be executed in the other. Figures 6(b)-(c) depict examples of non-observationally equivalent systems.

### 3.2 I/O compatibility of reactive systems

A formal description of the conditions needed for having a correct dialogue between two RTSs is given in this section. We call this set of conditions *I/O compatibility*. The properties of the I/O compatibility can be stated in natural language:

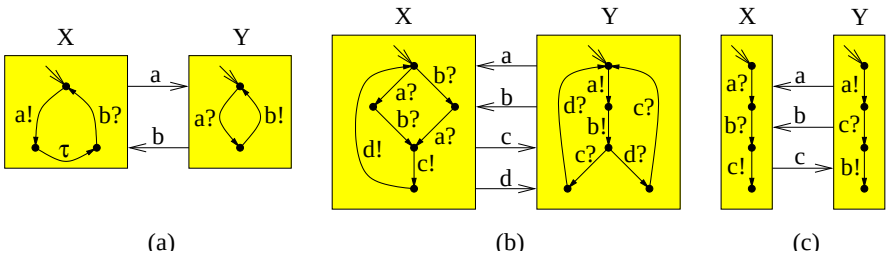



**Fig. 6.** Connection between different reactive systems (the suffixes ? and ! are used to denote input and output events, respectively).

*(a) Safeness: if system $A$ can produce an output event, then $B$ must be prepared to accept the event.*
*(b) Liveness: if system $A$ is blocked waiting for a synchronization with $B$, then $B$ must produce an output event in a finite period of time.*

Two RTSs are *structurally I/O-compatible* if they share the observational set of events, in a way that they can be connected.

**Definition 11 (Structural I/O compatibility).** Let $A = (S^A, \Sigma^A, T^A, s^A_{in})$ and $B = (S^B, \Sigma^B, T^B, s^B_{in})$ be two RTSs. $A$ and $B$ are *structurally I/O compatible* if $\Sigma^A_I = \Sigma^B_O$, $\Sigma^A_O = \Sigma^B_I$, $\Sigma^A \cap \Sigma^B_{INT} = \emptyset$ and $\Sigma^B \cap \Sigma^A_{INT} = \emptyset$.

The following definition gives a concise formalization of the conditions needed for characterizing the correct interaction of two RTSs:

**Definition 12 (I/O compatibility).** Let $A = (S^A, \Sigma^A, T^A, s^A_{in})$ and $B = (S^B, \Sigma^B, T^B, s^B_{in})$ be two structurally I/O compatible RTSs. $A$ and $B$ are *I/O compatible*, denoted by $A \rightleftharpoons B$, if $A$ and $B$ are livelock-free and there exists a relation $R \subseteq S^A \times S^B$ such that:

1. $s^A_{in} R s^B_{in}$.
2. *Receptiveness:*
   (a) If $s_1 R s'_1$, $e \in \Sigma^A_O$ and $s_1 \xrightarrow{e} s_2$ then $\mathsf{En}(s'_1, e)$ and $\forall s'_1 \xrightarrow{e} s'_2 : s_2 R s'_2$.
   (b) If $s_1 R s'_1$, $e \in \Sigma^B_O$ and $s'_1 \xrightarrow{e} s'_2$ then $\mathsf{En}(s_1, e)$ and $\forall s_1 \xrightarrow{e} s_2 : s_2 R s'_2$.
3. *Internal Progress:*
   (a) If $s_1 R s'_1$, $e \in \Sigma^A_{INT}$ and $s_1 \xrightarrow{e} s_2$ then $s_2 R s'_1$.
   (b) If $s_1 R s'_1$, $e \in \Sigma^B_{INT}$ and $s'_1 \xrightarrow{e} s'_2$ then $s_1 R s'_2$.
4. *Deadlock-freeness:*
   (a) If $s_1 R s'_1$ and $\{e \mid \mathsf{En}(s_1, e)\} \subseteq \Sigma^A_I$ then $\{e \mid \mathsf{En}(s'_1, e)\} \not\subseteq \Sigma^B_I$.
   (b) If $s_1 R s'_1$ and $\{e \mid \mathsf{En}(s'_1, e)\} \subseteq \Sigma^B_I$ then $\{e \mid \mathsf{En}(s_1, e)\} \not\subseteq \Sigma^A_I$.

Let us consider the examples of Figure 6. In Figure 6(a), the receptiveness condition fails and therefore $X$ and $Y$ are not I/O compatible. However, the RTSs of Figure 6(b) are I/O compatible. Finally, Figure 6(c) presents an example of violation of the deadlock-freeness condition.

Condition 4 has a strong impact on the behavior of the system. It guarantees that the communication between $A$ and $B$ has no deadlocks (see theorem 3).

**Lemma 1.** *Let $A$, $B$ be two RTSs such that $A \rightleftharpoons B$, let $R$ be an I/O compatible relation between $A$ and $B$ and let $A \times B = (S, \Sigma, T, s_{in})$ be the synchronous product of $A$ and $B$. Then, $\langle s, s' \rangle \in S \Rightarrow sRs'$*

*Proof.* See appendix.

**Theorem 1 (Safeness).** *Let $A$, $B$ be two RTSs such that $A \rightleftharpoons B$, and a trace $\sigma \in L(A \times B)$ of their synchronous product such that $s_{in} \stackrel{\sigma}{\rightarrow} \langle s, s' \rangle$. If $A$ can fire an output event in $s$, then the same event is enabled in state $s'$ of $B$.*

*Proof.* See appendix.

**Theorem 2.** *Let $A$, $B$ be two RTSs such that $A \rightleftharpoons B$, and let $A \times B$ be the synchronous product of $A$ and $B$. Then, $A \times B$ is livelock-free.*

*Proof.* See appendix.

The I/O compatibility relation represents implicitly a *liveness property*, stated in the following theorem:

**Theorem 3 (Liveness).** *Let $A$, $B$ be two RTSs such that $A \rightleftharpoons B$, and a trace $\sigma \in L(A \times B)$ of their synchronous product such that $s_{in} \stackrel{\sigma}{\rightarrow} \langle s, s' \rangle$. If only input events of $A$ are enabled in $s$, then there exists some trace $\langle s, s' \rangle \stackrel{\sigma'}{\rightarrow} \langle s, s'' \rangle$ such that some of the input events of $A$ enabled in $s$ are also enabled in $s''$ as output events of $B$.*

*Proof.* See appendix.

### 3.3 A sufficient condition for I/O compatibility.

A sufficient condition for having I/O compatibility between two reactive systems can be obtained when combining the notions of observational equivalence and input-properness:

**Theorem 4.** *Let $A = (S^A, \Sigma^A, T^A, s^A_{in})$, $B = (S^B, \Sigma^B, T^B, s^B_{in})$ be two livelock-free RTSs with $\Sigma^A_I = \Sigma^B_O$ and $\Sigma^A_O = \Sigma^B_I$. If $A$ and $B$ are input proper and $A \approx B$, then $A \rightleftharpoons B$.*

When considering a system $A$ and some I/O compatible system $B$, any transformation of $B$ preserving both input-properness and observational equivalence will lead to another I/O compatible system:

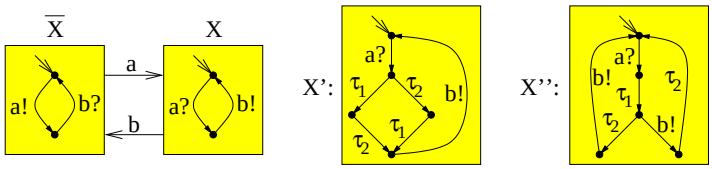



**Fig. 7.** Relation between I/O compatibility and observational equivalence.

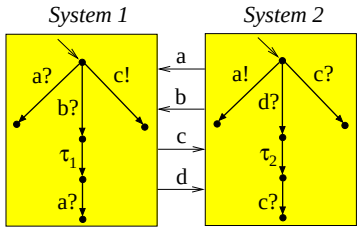



**Fig. 8.** Two I/O compatible systems that are not input-proper.

**Theorem 5.** *Let* $A = (S^A, \Sigma^A, T^A, s^A_{in})$, $B = (S^B, \Sigma^B, T^B, s^B_{in})$ *and* $C = (S^C, \Sigma^C, T^C, s^C_{in})$ *be three* RTSs. *If* $A \rightleftharpoons B$, $B \approx C$, $\Sigma^B_I = \Sigma^C_I$, $\Sigma^B_O = \Sigma^C_O$ *and* $C$ *is input-proper then* $A \rightleftharpoons C$.

*Proof.* See appendix.

Figure 7 shows an example of application of Theorem 5. The transformation of $X$ which leads to $X'$ preserves both observational equivalence and input-properness, and then, $\overline{X}$ and $X'$ can safely interact.

Finally, it must be noted that I/O compatibility does not require input-properness, as shown in Figure 8. This occurs when the non-input-proper situations are not reachable by the interaction of the two systems. For the sake of simplicity, only input-proper systems will be considered along this paper.

### 3.4 Realizations of a reactive system

In this section, it is of interest to characterize when the specification of a reactive system is correctly realized by a given implementation. For this purpose, two RTSs representing specification's and implementation's behavior are compared. The I/O compatibility can be used to determine when a pair ⟨specification, implementation⟩ represents a correct realization: for specification $A$, the system $\overline{A}$ represents a model of the environment where a possible implementation $B$ must correctly interact.

**Definition 13 (I/O preserving realization).** Let $A$ and $B$ be two RTSs, $A$ representing the specification of a reactive system. $B$ realizes $A$ (A $\models$ B) iff $\overline{A} \rightleftharpoons B$.

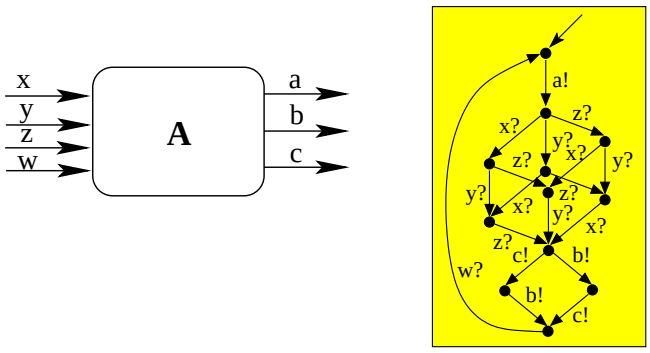



**Fig. 9.** (a) Interface of system A, (b) Transition System model.

Let us illustrate the concept with the example of system A, specified in Fig. 9. It observes events `x`, `y`, `z` and `w`, and generates events `a`, `b` and `c` (Fig. 9(a)). The behavior of such system is specified with the RTS of Fig. 9(b). Figure 10 depicts the RTSs of four possible realizations of system A.

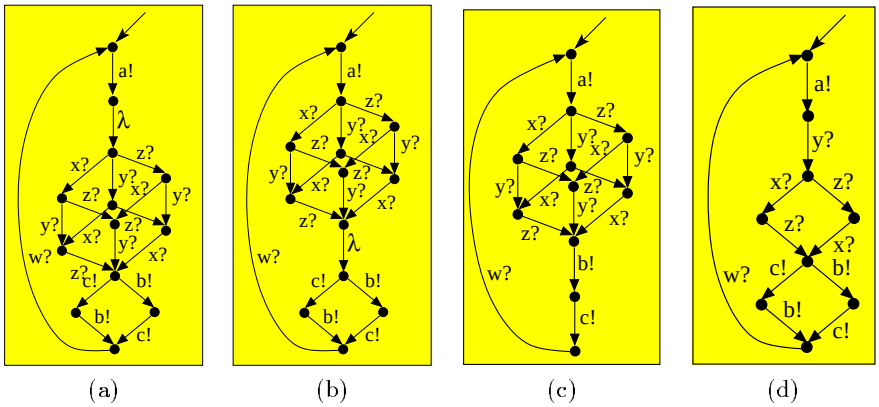



**Fig. 10.** RTS of different implementations of system A.

The reactive transition system 10(a) violates the input-proper property and can cause a malfunction in the entire system: if the realization of the internal event $\lambda$ takes a long delay, 10(a) is imposing that after event `a` has been generated, the environment must wait before generating the events `x`, `y` and `z`. Therefore, if implementation 10(a) was used, the system(s) in the environment responsible of generating events `x`, `y` and `z` must be re-synthesized. However, the insertion of $\lambda$ in 10(b) leads to an input-proper

RTS. Sometimes it is necessary to insert internal events in the specification, in order to fulfill some implementability conditions (see Section 8).

The reactive transition system 10(c) is a correct realization of system A, because it represents a modification of the specification 9(b), where the concurrency of the output events b and c is reduced: the environment will observe 10(c)'s events in a more restricted (but still expected) order. The transition system 10(d) represents an erroneous realization of system A: it is restricting the way input events must be received.

# 4 Synthesis of reactive systems

The action of synthesis can be defined as the process of combining transformations that preserve certain properties. In this section we are interested in transformations applied to a reactive system that preserve the I/O compatibility.

Highly concurrent systems suffer from the well-known *state explosion problem*: the set of states of a system can be exponentially large with respect to its set of events. Therefore, when dealing with large concurrent systems, state-based models like transition systems are not suitable for its representation. On the contrary, event-based models like Petri nets circumvent this problem by representing the system as a set of causality relations between the events. In this section we use the Petri net model as the formal model for specifying a system.

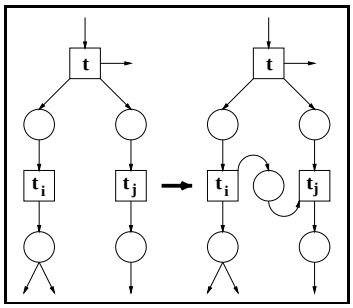
A kit of synthesis rules is presented in this section. It is based on the application of Petri net transformations over a RPN. The kit of transformations is defined in Section 4.1. Section 4.2 presents those transformations that preserve the I/O compatibility.

## 4.1 Kit of PN transformations

Three rules are presented for modifying the structure of a FCLSPN. The rule $\phi_r$ is used for sequencing two concurrent transitions. It was first defined in [2]. Here a reduced version is presented. Rule $\phi_i$ does the opposite: it increases the concurrency between two ordered transitions. $\phi_i$ can be obtained as a combination of the ones appearing in [17]. Finally, rule $\phi_e$ removes a given transition. It was first presented in [14]. All three rules preserve the liveness, safeness and free-choiceness.

**Rule $\phi_r$**

The purpose of the rule $\phi_r$ is to eliminate the concurrency between two transitions of the PN. This is done by inserting a place that connects the two transitions, ordering their firing. The following figure presents an example of concurrency reduction between transitions $t_i$ and $t_j$.




The formal definition of the rule is:

Let $N = (P, T, F, M_0)$, $N' = (P', T, F', M_0')$ be two FCLSPNs, and transitions $t_i, t_j \in T$. Then, $\phi_r(N, t_i, t_j) = N'$ if:

Conditions on $N$:

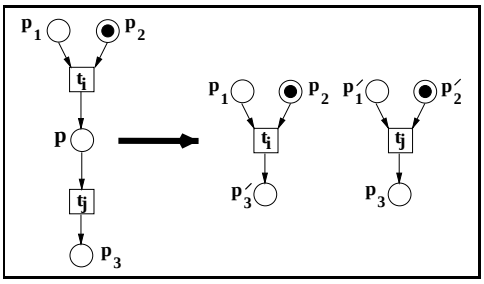
1. $\{t\} = {}^\bullet({}^\bullet t_i) = {}^\bullet({}^\bullet t_j)$
2. ${}^\bullet t_i = \{p_i\} \wedge |p_i^\bullet| = 1$
3. ${}^\bullet t_j = \{p_j\} \wedge |p_j^\bullet| = 1$
4. $M_0(p_i) = M_0(p_j)$

Conditions on $N'$:

1. $P' = P \cup \{p\}$
2. $F' = F \cup \{(t_i, p), (p, t_j)\}$
3. $M_0' = M_0 \cup \{p \leftarrow 0\}$

**Rule $\phi_i$**

Inversely to rule $\phi_r$, rule $\phi_i$ removes the causality relation between two ordered transitions, making them concurrent.The following figure presents an example of increase of concurrency between transitions $t_i$ and $t_j$.



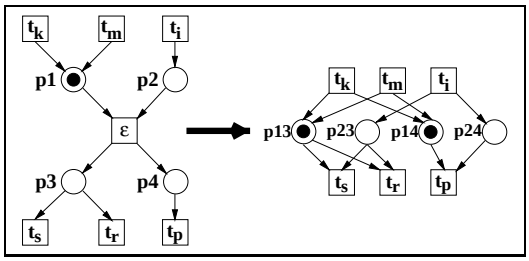

The formal definition of the rule is:

Let $N = (P, T, F, M_0)$, $N' = (P', T', F', M_0')$ be two FCLSPNs, and transitions $t_i, t_j \in T$. In the following definition, places $p_k'$ represent new places originated from places either in ${}^\bullet t_i$ $(k = i)$ or in $t_j^\bullet$ $(k = j)$. Then, $\phi_i(N, t_i, t_j) = N'$ if:

| Conditions on $N$: | Conditions on $N'$: |
|---|---|
| 1. $\{(t_i,p),(p,t_j)\} \subseteq F$ | 1. $P' = (P \setminus \{p\}) \cup \{p'_i \mid p_i \in {}^\bullet t_i\} \cup \{p'_j \mid p_j \in t_j^\bullet\}$ |
| 2. $\mid {}^\bullet p \mid = \mid p^\bullet \mid = 1$ | 2. $F' = (F \setminus \{(t_i,p),(p,t_j)\}) \cup$ |
| 3. $\forall q \in {}^\bullet t_i : \mid q^\bullet \mid = 1$ | $\{(y,p'_i) \mid (y,p_i) \in F\} \cup \{(p'_i,t_j) \mid (p_i,t_i) \in F\} \cup$ |
| 4. $t_i \notin (t_j^\bullet)^\bullet$ | $\{(p'_j,y) \mid (p_j,y) \in F\} \cup \{(t_i,p'_j) \mid (t_j,p_j) \in F\} \cup$ |
| | $\{(y,p'_j) \mid (y,p_j) \in F \wedge y \neq t_j\}$ |
| | 3. $M'_0 = M_0 \cup \{p'_k \leftarrow M_0(p_k) + M_0(p)\}$ |

**Rule $\phi_e$**

The rule $\phi_e$ eliminates a transition from the PN. The following figure presents an example of elimination of transition $\varepsilon$.




The formal definition of the rule is:

Let $N = (P,T,F,M_0)$, $N' = (P',T',F',M'_0)$ be two FCLSPNs, transition $\varepsilon \in T$ and let $P_\varepsilon = ({}^\bullet\varepsilon) \times (\varepsilon^\bullet)$. Then, $\phi_e(N,\varepsilon) = N'$ if:

| Conditions on $N$: | Conditions on $N'$: |
|---|---|
| 1. $\forall p : p \in {}^\bullet\varepsilon : p^\bullet = \{\varepsilon\}$ | 1. $P' = (P \setminus ({}^\bullet\varepsilon \cup \varepsilon^\bullet)) \cup P_\varepsilon$ |
| | 2. $T' = T \setminus \{\varepsilon\}$ |
| | 3. $F' = (F \setminus \{(a,b) \mid (a,b) \in F \wedge$ |
| | $(a = \varepsilon \vee b = \varepsilon)\}) \cup$ |
| | $\{(y,\langle p_1,p_2\rangle) \mid (y,p_1) \in F\} \cup$ |
| | $\{(\langle p_1,p_2\rangle,y) \mid (p_2,y) \in F\}$ |
| | 4. $M'_0 = M_0|_{P \setminus ({}^\bullet\varepsilon \cup \varepsilon^\bullet)} \cup$ |
| | $\{\langle p_1,p_2\rangle \leftarrow k \mid \langle p_1,p_2\rangle \in P_\varepsilon \wedge$ |
| | $k = M_0(p_1) + M_0(p_2)\}$ |

where $f|_C$ represents the restriction of function $f$ to set $C$.

### 4.2 I/O compatible transformations over RPN

The I/O compatible relation operator ($\rightleftharpoons$) can be lifted to RPNs:

**Definition 14 (I/O compatible relation over RPN).** Let $A$ and $B$ be two RPNs with corresponding RTSs $\mathsf{RTS}(A)$ and $\mathsf{RTS}(B)$. $A \rightleftharpoons B$ if $\mathsf{RTS}(A) \rightleftharpoons \mathsf{RTS}(B)$.

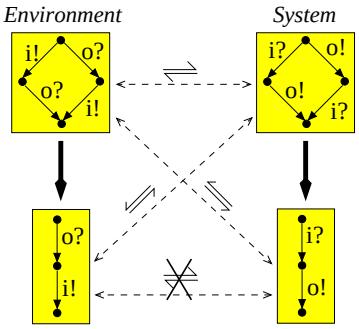



**Fig. 11.** Different possibilities for reducing concurrency.

For each transformation of the kit presented in Section 4.1, the following sections enumerate those situations where the transformation can be applied to the underlying FCLSPN of a deterministic RPN while preserving the I/O compatible relation.

### I/O compatible application of $\phi_r$

The application of $\phi_r(A, e_1, e_2)$ preserves $\rightleftharpoons$ when neither $e_1$ nor $e_2$ is an input transition. In fact, it is sufficient to require only $e_2$ to be non-input for the preservation of $\rightleftharpoons$, but then deadlock situations may arise. Figure 11 exemplifies this: initially, both environment and system can safely interact. Moreover, if either the environment or the system are transformed by reducing concurrency between an input and an output, the interaction can still be safe. However, the two transformed systems can not interact. The formalization of the transformation is:

**Theorem 6.** *Let the RPNs $A$, $B$ and $C$ with underlying FCLSPN and corresponding deterministic RTSs $(S^A, \Sigma^A, T^A, s^A_{in})$, $(S^B, \Sigma^B, T^B, s^B_{in})$ and $(S^C, \Sigma^C, T^C, s^C_{in})$, respectively. Assume $\Sigma^C = \Sigma^B$. If*

1. $A \rightleftharpoons B$
2. $\phi_r(B, e_1, e_2) = C$, *with* $e_1, e_2 \notin \Sigma^B_I$

*then* $A \rightleftharpoons C$.

*Proof.* See appendix.

### I/O compatible application of $\phi_i$

The application of $\phi_i$ preserves $\rightleftharpoons$ when:

1. at least one of the transitions involved is internal, and
2. no internal transition is inserted as trigger of an input transition

The purpose is to avoid the increase of concurrency between two observable transitions, in order to forbid the generation of unexpected traces either on the environment's or on the system's part. More formally:

**Theorem 7.** *Let the RPNs $A$, $B$ and $C$ with underlying FCLSPN and corresponding deterministic RTSs $(S^A, \Sigma^A, T^A, s^A_{in})$, $(S^B, \Sigma^B, T^B, s^B_{in})$ and $(S^C, \Sigma^C, T^C, s^C_{in})$. Assume $\Sigma^C = \Sigma^B$. If*

1. $A \rightleftharpoons B$
2. $\phi_i(B, e_1, e_2) = C$, *with either* $e_1 \in \Sigma^B_{INT}$ *or* $e_2 \in \Sigma^B_{INT}$
3. $B$ *is input-proper*
4. $\forall e \in (e_2^\bullet)^\bullet : e \notin \Sigma^B_I$

*then* $A \rightleftharpoons C$.

*Proof.* See appendix.

### I/O compatible application of $\phi_e$

Rule $\phi_e$ only preserves $\rightleftharpoons$ when applied to internal transitions.

**Theorem 8.** *Let the RPNs $A$, $B$ and $C$ with underlying FCLSPN and corresponding deterministic RTSs $(S^A, \Sigma^A, T^A, s^A_{in})$, $(S^B, \Sigma^B, T^B, s^B_{in})$ and $(S^C, \Sigma^C, T^C, s^C_{in})$. Assume $\Sigma^C_{OBS} = \Sigma^B_{OBS}$. If*

1. $A \rightleftharpoons B$
2. $\phi_i(B, e) = C$, *with* $e \in \Sigma^B_{INT}$
3. $B$ *is input-proper*

*then* $A \rightleftharpoons C$.

*Proof.* See appendix.

The transformations presented above can introduce redundant places in the target net. For dealing only with place-irredundant nets, the kit is augmented with a rule for eliminating redundant places. Linear programming techniques exist that decide the redundancy of a place efficiently [22]. Moreover, each time a transformation is performed, it can be locally determined the potential redundant places, and therefore the redundancy checking is only applied to a few places.

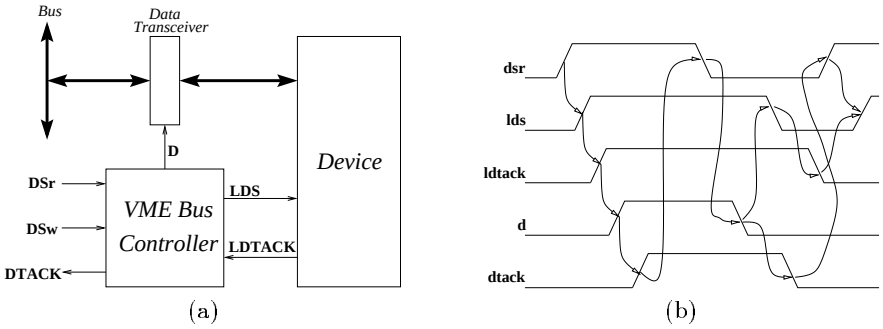



**Fig. 12.** (a) Interface, (b) Timing Diagram.

## 5 Asynchronous circuits

Asynchronous circuits are digital circuits that react to the changes of their input signals according to the functionality of the gates of the circuit [3]. Synchronous circuits can be considered as a particular case of asynchronous circuits in which some specific design rules and operation mode are imposed.

In general, any arbitrary interconnection of gates is considered an asynchronous circuit. The synthesis problem consists in generating a proper interconnection of gates that commits a correct interaction with the environment according to some specified protocol.

This section presents the models used in this work for the specification and synthesis of asynchronous circuits.

### 5.1 State graphs

Asynchronous circuits can be modeled with a RTS, where the events represent changes in the value of the system signals. The VME Bus Controller example in Fig. 12 will be used for illustrating the concepts. The interface is depicted in Fig. 12(a), where the circuit controls data transfers between the bus and the device. Figure 12(b) shows the timing diagram corresponding to the read cycle.

**Binary interpretation** A transition labeled as $x_i+$ ($x_i-$) denotes a rising (falling) of signal $x_i$: it switches from 0 to 1 (1 to 0). Figure 13 shows the RTS specifying the behavior of the bus controller for the read cycle. Each state of an asynchronous circuit can be encoded with a *binary vector*, representing the signal values on that state. The set of encoded states are *consistently encoded* if no state can have an enabled rising (falling) transition $a+$ ($a-$) when the value of the signal in that state is 1 (0) (see Section 6 for a formal definition of consistency). Correspondingly, for each signal of a RTS representing an asynchronous circuit, a partition of the

states of the RTS can be done by separating the states where the signal has value one, from those where the signal has value zero. This partition can only be done when the underlying asynchronous circuit is consistently encoded. Figure 14(a) shows the partition induced by considering signal `lds` in the RTS of Fig. 13. Each transition from `LDS=0` to `LDS=1` is labeled with `lds+` and each transition from `LDS=1` to `LDS=0` is labeled with `lds-`. A binary vector can be assigned to each state if such partition is done for each signal of the system. The encoded transition system is called *State Graph*.

**Definition 15 (State Graph).** A State Graph (SG) is a *3-tuple* $A = (A', \mathcal{X}, \lambda)$ where

- $A' = (S, \Sigma, T, s_{in})$ is a RTS
- $\mathcal{X}$ is the set of signals partitioned into inputs ($\mathcal{I}$), observable outputs ($\mathcal{O}bs$) and internal outputs ($\mathcal{I}nt$), and $\Sigma = \mathcal{X} \times \{+, -\} \cup \{\varepsilon\}$ , where all transitions not labeled with the silent event ($\varepsilon$) are interpreted as signal changes
- $\lambda : S \rightarrow \mathbb{B}^{|\mathcal{X}|}$ is the state encoding function

Figure 14(b) shows the SG of the bus controller.

We will denote by $\lambda_x(s)$ the value of signal $x$ is state $s$. The following definitions relate signal transitions with states. They will be used later to derive Boolean equations from an SG.

**Definition 16 (Excitation quiescent regions).** The positive and negative *excitation regions* (ER) of signal $x \in X$, denoted by ER($x+$) and ER($x-$), are the sets of states in which $x+$ and $x-$ are enabled, respectively, i.e.

$$\mathsf{ER}(x+) = \{s \in S \mid \exists s \xrightarrow{x+} s' \in T\}$$
$$\mathsf{ER}(x-) = \{s \in S \mid \exists s \xrightarrow{x-} s' \in T\}$$

The positive and negative *quiescent regions* (QR) of signal $x \in X$, denoted by QR($x+$) and QR($x-$) are the sets of states in which $x$ has the same value, 1 or 0, and is stable, i.e.

$$\mathsf{QR}(x+) = \{s \in S \mid \lambda_x(s) = 1 \ \wedge \ s \notin \mathsf{ER}(x-)\}$$
$$\mathsf{QR}(x-) = \{s \in S \mid \lambda_x(s) = 0 \ \wedge \ s \notin \mathsf{ER}(x+)\}$$

### 5.2 Signal transition graphs

As in the case of the RTS model, events of a RPN can represent signal changes of an asynchronous circuit. The model is called *Signal Transition Graph* [21].

**Definition 17.** A Signal Transition Graph (STG) is a 3-tuple $(N, \mathcal{X}, \Lambda)$, where

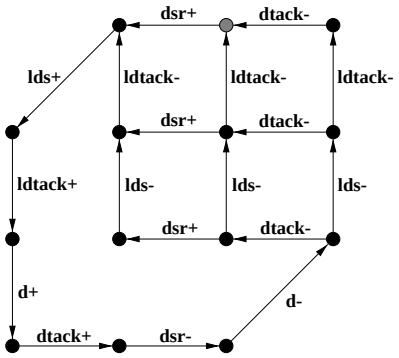



**Fig. 13.** Transition System specifying the bus controller.

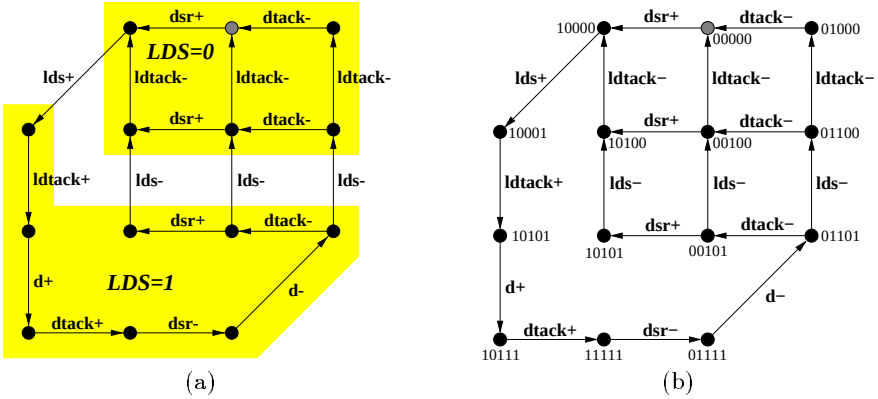



**Fig. 14.** (a) Partition induced by signal `lds`, (b) State graph of the read cycle. States are encoded with the vector (`dsr`,`dtack`,`ldtack`,`d`,`lds`).

- $N = (P, T, F, M_0)$ is a Petri net
- $\mathcal{X}$ and $\Sigma$ are defined as in the case of the SG.
- $\Lambda : T \rightarrow \Sigma$

**Adjacency** Transition $x_{i*}$ is said ti be a *predecessor* of $x_{j*}$ if there exists a feasible sequence $x_{i*}\sigma x_{j*}$ that does not include other transitions of signal $x$. Conversely, $x_{j*}$ is a *successor* of $x_{i*}$. We will also say that the pair $(x_{i*}, x_{j*})$ is *adjacent*. The set of predecessors (successors) of $x_{i*}$ is denoted by $prev(x_{i*})$ $(next(x_{i*}))$.

An example of STG specifying the bus controller is shown in Figure 15. Places of the STG with only one predecessor and one successor transition, are not shown graphically as convention. The RTS associated to an STG is an SG. The SG associated to the STG of Figure 15 is shown in 14(b).

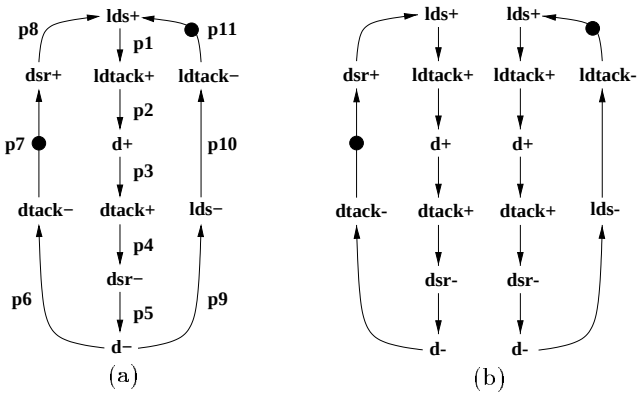



**Fig. 15.** (a) Signal Transition Graph specifying the bus controller, (b) State machine cover.

**Concurrency relations** Concurrency relations can be naturally extended to nodes and signals in a STG. The *Signal concurrency relation* between a node $u_j \in P \cup T$ and a signal $x \in \Lambda$ is defined as a binary relation $\mathcal{SCR}$, such that $(u_j, x) \in \mathcal{SCR} \Leftrightarrow \exists x_{i*} : (u_j, x_{i*}) \in \mathcal{CR}$ .

The class of STGs with underlying IO-RPN are defined. This class will be used in Sections 8 and 9.

**Definition 18.** An IO-STG is an STG with underlying IO-RPN.

Figure 15 shows an example of IO-STG.

## 6 Synthesis of Speed-Independent Circuits

Speed-independent (SI) circuits is the class of asynchronous circuits that work correctly regardless the delay of their components (gates). Currently, there is a robust theory, design flow and some tools [7] that support the automatic synthesis of SI circuits.

However, one of the major problems of the methods used for synthesis is that they require an explicit knowledge of the state graph. Highly concurrent systems often suffer the state explosion problem and, for this reason, the size of the state graph can be a limiting factor for the practical application of synthesis methods.

In this section, some basic concepts on the logic synthesis of SI circuits are presented. We refer the reader to [7] for a deeper theory on how to implement SI circuits. In this paper, we will focus on the main step in synthesis: the derivation of the Boolean equations that model the behavior of the digital circuit.

In Sections 7, 8 and 9, a synthesis framework that only uses structural methods on STGs to derive the Boolean equations will be provided. By only using structural methods, the computational complexity associated to the state explosion problem is avoided.

### 6.1 Implementability as a Logic Circuit

This section defines a set of properties that guarantee the existence of a SI circuit. They are defined at the level of SG, but can be easily extended to STGs. Instead of giving new definitions for STGs, we will simply consider that a property holds in an STG if it holds in its underlying SG.

The properties are the following: boundedness, consistency, complete state coding and output persistency.

**Boundedness.** A necessary condition for the implementability of a logic circuit is that the set of states is finite. Although this seems to be an obvious assumptions at the level of SG, it is not so obvious at the level of STG, since an STG with a finite structure may have a infinite number of reachable markings.

**Consistency.** As shown in Figure 14, each signal $x_i$ defines a partition of the set of states. The consistency of an SG refers to the fact that the events $x_i+$ and $x_i-$ are the only ones that cross these two parts according to their meaning: switching from 0 to 1 and from 1 to 0, respectively. This is captured by the definition of consistent SG.

**Definition 19 (Consistent SG).** An SG is *consistent* if for each transition $s \xrightarrow{e} s'$ the following conditions hold:

- if $e = x_i+$, then $\lambda_i(s) = 0$ and $\lambda_i(s') = 1$;
- if $e = x_i-$, then $\lambda_i(s) = 1$ and $\lambda_i(s') = 0$;
- in all other cases, $\lambda_i(s) = \lambda_i(s')$.

where $\lambda_i$ denotes the component of the encoding vector corresponding to signal $x_i$.

**Complete State Coding** This property can be illustrated with the example of Figure 14(b), in which there are two states with the same binary encoding: 10101. Moreover, the states with the same binary code are behaviorally different. This fact implies that the system does not have enough information to determine how to react by only looking at the value of its signals.

The distinguishability of behavior by state encoding is captured by the following two definitions.

**Definition 20 (Unique State Coding).** [5] An SG satisfies the *Unique State Coding* (USC) condition if every state in $S$ is assigned a unique binary code. Formally, USC means that the state encoding function, $\lambda$, is injective.

**Definition 21 (Complete State Coding).** [5] An SG satisfies the *Complete State Coding* (CSC) condition if for every pair of states $s, s' \in S$ having the same binary code the sets of enabled non-input signals are the same.

Both properties are sufficient to derive the Boolean equations for the synthesized circuit. However, given that only the behavior of the non-input signals must be implemented, encoding ambiguities for input signals are acceptable.

**Output persistency** This property is required to ensure that the discrete behavior modeled with SG has a robust correspondence with the real analog behavior of electronic circuits.

**Definition 22 (Disabling).** An event $x$ is said to *disable* another event $y$ if there is a transition $s \xrightarrow{x} s'$ such that $y$ is enabled $s$ but not in $s'$.

**Definition 23 (Output persistency).** An SG is said to be output persistent if for any pair of events $x$ and $y$ such that $x$ disables $y$, both $x$ and $y$ are input signals.

In logic circuits, disabling an event may result in non-deterministic behavior. Imagine, for example, that an AND gate has both inputs at 1 and the output at 0. In this situation, the gate starts the process to switch the signal towards 1 in a continuous way. If one of the inputs would fall to 0 during this process, the output would interrupt this process and start moving the signal to 0, thus producing an observable glitch. To avoid these situations, that may produce unexpected events, the property of output persistency is required.

### 6.2 Boolean equations

This sections describes the procedure to derive Boolean next-state functions for output signals from an SG. The procedure defines an incompletely specified function from which a gate implementation can be obtained after Boolean minimization.

An incompletely specified $n$-variable *logic function* is a mapping $F : \{0,1\}^n \rightarrow \{0,1,-\}$. Each element $\{0,1\}^n$ is called a *vertex* or binary code. A *literal* is either a variable $x_i$ or its complement $\overline{x_i}$. A *cube* $c$ is a set of literals, such that if $x_i \in c$ then $\overline{x_i} \notin c$ and vice versa. Cubes are also represented as an element $\{0,1,-\}^n$, in which value 0 denotes a complemented variable $\overline{x_i}$, value 1 denotes a variable $x_i$, and $-$ indicates the fact that the variable is not in the cube. A *cover* is a set of implicants which contains the on-set and does not intersect with the off-set.

Given a specification with $n$ signals, the derivation of an incompletely specified function $F^x$ for each output signal $x$ and for each $v \in \mathbb{B}^n$ can be formalized as follows:

$$F^x(v) = \begin{cases} 1 & \text{if } \exists\ s \in \mathsf{ER}(x+) \cup \mathsf{QR}(x+) : \lambda(s) = v \\ 0 & \text{if } \exists\ s \in \mathsf{ER}(x-) \cup \mathsf{QR}(x-) : \lambda(s) = v \\ - & \text{if } \nexists\ s \in S : \lambda(s) = v \end{cases}$$

The set of vertices in which $F^x(v) = 1$ is called the on-set of signal $x$ ($\mathsf{ON}(x)$), whereas the codes in which $F^x(v) = 0$ is called the off-set of $x$ ($\mathsf{OFF}(x)$).

The previous definition is ambiguous when there are two states, $s_1$ and $s_2$, for which $\lambda(s_1) = \lambda(s_2) = v$, $s_1 \in \mathsf{ER}(x+) \cup \mathsf{QR}(x+)$ and $s_2 \in \mathsf{ER}(x-) \cup \mathsf{QR}(x-)$. This ambiguity is precisely what the CSC property avoids, and this is why CSC is a necessary condition for implementability.

Figure 16 depicts an STG and the corresponding SG. Figure 17 shows the Karnaugh maps of the incompletely specified functions for signals $a$ and $d$.

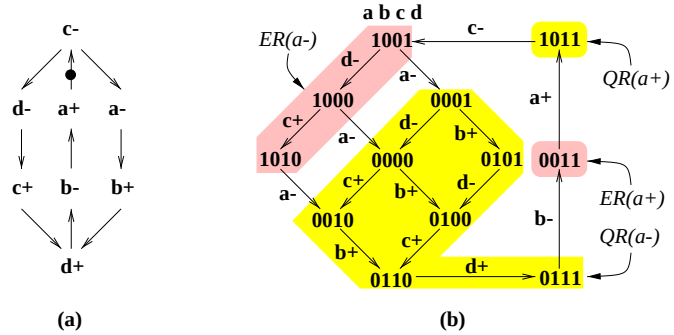



**Fig. 16.** Example *abcd*: (a) Signal Transition Graph, (b) State Graph

Form the incompletely specified functions, other circuit architectures can also be derived [7].

## 7 Structural approximations for logic synthesis

Most of the existing frameworks for the synthesis of SI circuits from STG specifications rely on the explicit computation of the underlying SG to check implementability conditions and generate Boolean equations [6] (see Section 6). Unfortunately, the underlying SG of a highly concurrent system can be exponential in the size of the STG specification, which leads to the well-known state explosion problem.

This section describes how to link structure and behavior with polynomial complexity if the STG specification has an underlying FCLSPN, thus avoiding exponentiality. The method is based on the analysis of the concurrency relations and state machine decompositions of the STG. States are

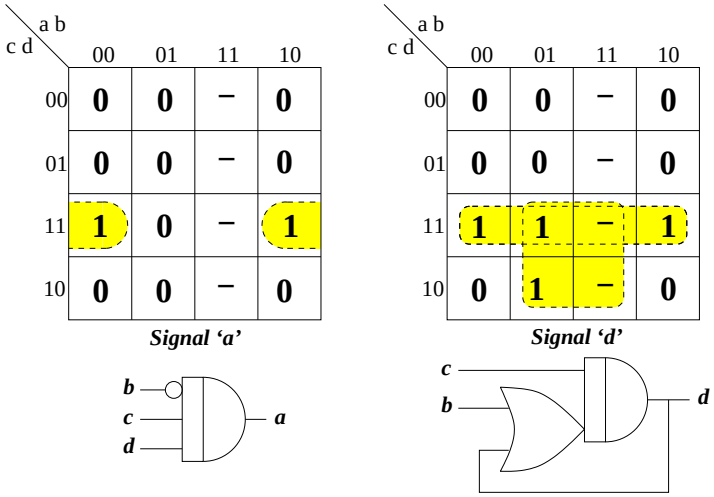



**Fig. 17.** Complex gate implementation for the *abcd* example

never generated, instead their encoding vectors are approximated by logic functions with various degrees of accuracy. State explosion is avoided by using a conservative approximation of the state graph, but providing computationally efficient algorithms that generate minimized implementations comparable to those generated by state-based tools. This section presents a general overview of results developed elsewhere. We refer to [18,19] for further details and proofs.

### 7.1 Structural approximations of the state graph

Initially we introduce a technique to approximate the underlying SG of an STG. This approximation will be used to check the USC conditions in Section 7.2, and to provide approximate logic functions for the logic synthesis process in Section 7.4. The structural approximation of the SG is based on a simple concept: characterize the states in which a given place is marked, this is the so called *marked region*. The goal is to derive a simple logic function to approximate each marked region.

**Structural consistency verification** Consistency is the necessary condition to assign binary codes to the states in the underlying SG of an STG (see Section 6). Before providing an approximation of the SG it is necessary to guarantee that it can be properly encoded.

Checking consistency in a STG with underlying FCLSPN can be reduced to checking (1) the absence of *auto-concurrent* transitions, and (2) that every sequence of signal transitions is *switchover correct* [11]. Auto-concurrent

transitions exists when a pair of transitions $x_{i*}$ and $x_{j*}$ of signal $x$ are concurrent. *Switchover correctness* requires the value of each signal $x$ to switch from 0 to 1 in response to a rising transition, and to switch from 1 to 0 due to a falling transition. Switchover correctness of a non auto-concurrent STG is verified by checking that all adjacent transitions of the same signal have alternating switching directions.

The adjacency of a pair of transitions of the same signal can be determined by finding a particular path in the STG connecting both transitions. The following property characterizes the relation between the formal definition of adjacency (see Section 5) and its structural computation on the structure of the STG, thus linking the feasible sequence concept to structural paths in the STG.

**Proposition 1 (Structural characterization of adjacency).** *[Necessary condition] In a STG with underlying FCLSPN, a transition $x_{j*} \in next(x_{i*})$ if there is a simple path $L$ between $x_{i*}$ and $x_{j*}$ such that:*

1. *no place $p \in L$ is concurrent to signal $x$; and*
2. *$L$ contains no other transitions of signal $x$ except $x_{i*}$ and $x_{j*}$.*

To obtain sufficient conditions to determine the adjacency between transitions of the same signal it is necessary to distinguish which concurrency relations are not relevant for adjacency. Additional details can be found in [19].

**Marked regions and cover cubes** A set of markings, named *marked region*, define the correspondence between the basic structural elements of an STG and its underlying SG. We will introduce this basic region, derive its fundamental properties, and show how to approximate its state encoding by using a single cover cube.

**Definition 24 (Marked region).** Given a place $p$, its marked region, denoted $\mathsf{MR}(p)$, is the set of markings in which $p$ has at least one token, i.e. $\mathsf{MR}(p) = \{M \in [M_0\rangle \ : \ M(p) > 0\}$ .

With an abuse of notation we introduce the operator $\widehat{\ }$ to define the characteristic function of the binary codes $\lambda(s)$ of all states in the underlying SG equivalent to a set of markings. Table 1 show the main objects used in the synthesis process. The first column refers to sets of markings in a STG, the second column refers to the binary vectors that encode those markings, and the last column refers to a logic function that includes all (and maybe more) binary vectors in the previous object.

All states in the underlying SG of a STG are contained in the union of the MR of places in any given SM. Additionally, if the SM satisfies the *one-token* condition, then the MR of its set of places define a total partition of the SG.

**Proposition 2 (Projection of the reachability set onto a SM).** *The following properties are satisfied for any State machine SM of a FCLSPN :*

1. *The union of the marked regions of every place in SM is equivalent to* $[M_0\rangle$, *i.e.* $[M_0\rangle = \bigcup_{p \in SM} \mathsf{MR}(p)$ .
2. *Additionally, if SM satisfies the* one-token *condition then the marked region of places in SM define a total partition of* $[M_0\rangle$, *i.e.* $\forall p, p' \in SM$ : $\mathsf{MR}(p) \cap \mathsf{MR}(p') = \emptyset$ .

As an example, Fig. 19 highlights the marked region for place $p_8$. Note that $\mathsf{MR}(p_8)$ is entered after firing transition `dsr+` and it is only left when transition `lds+` fires. The binary codes corresponding to those markings are characterized by $\widehat{\mathsf{MR}}(p_8) = \{\texttt{10101}, \texttt{10100}, \texttt{10000}\}$.

A cover cube for a MR must cover the binary encoding of all markings in the region. To make the approximation more accurate this cube should be the smallest among possible (with the largest number of literals) [12]. Any signal that does not change in the MR of a place (is not concurrent to the place) is represented by a corresponding literal in a cover cube. The value of this signal can be determined by an interleave relation. *Interleaving* characterizes the position of a node with respect to a pair of adjacent signal transitions.

**Definition 25 (Interleave relation).** The *Interleave Relation* is a binary relation $\mathcal{IR}$ between nodes in $P \cup T$ and pairs of adjacent transitions $x_{i*}$ and $x_{j*}$ of a signal $x$ such that, a node $u_j$ is *interleaved* with $(x_{i*}, x_{j*})$ $(u_j \in \mathcal{IR}(x_{i*}, x_{j*}))$, if there exists a simple path from $x_{i*}$ to $x_{j*}$ containing $u_j$.

**Proposition 3 (Consistent place interleaving).** *In a consistent STG if a place p is interleaved with a pair of adjacent transitions* $(x_i+, x_i-)$ *then p cannot be interleaved with any other adjacent pair* $(x_j-, x_j+)$ *and vice versa.*

Property 3 guarantees that if a place $p$ is non-concurrent to signal $x$ and interleaved between two adjacent transitions $(x_i+, x_i-)$ $((x_i-, x_i+))$, then all binary codes in $\widehat{\mathsf{MR}}(p)$ have value 1 (0) for signal $x$. This property is the basis to approximating markings by computing a single *cover cube* for each marked region. We can see in Fig. 19 that place $p_8$ is interleaved between `dtack-` and `dtack+` because the path `dtack-` $\rightarrow$ `dsr+` $\rightarrow$ `lds+` $\rightarrow$ `ldtack+` $\rightarrow$ `d+` $\rightarrow$ `dtack+` exists. Furthermore, all markings in $\mathsf{MR}(p_8)$ are encoded with `dtack` $= 0$.

**Proposition 4 (Boolean approximation for a MR [12]).** *The* cover cube $\mathcal{C}_p$ *for* $\mathsf{MR}(p)$ *is the smallest cube that covers* $\widehat{\mathsf{MR}}(p)$ *such that for every signal x:*

1. *if x non-concurrent to p* $((p, x) \notin \mathcal{SCR})$ *then*

$$\mathcal{C}_p^x = \begin{cases} 0 \text{ if } b = 0 \text{ in } \widehat{\mathsf{MR}}(p), \\ 1 \text{ if } b = 1 \text{ in } \widehat{\mathsf{MR}}(p). \end{cases}$$

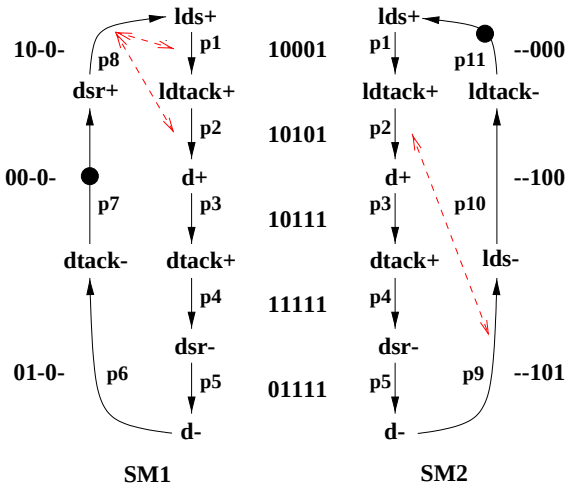



**Fig. 18.** SM-cover and cover cubes for the STGin Fig. 15. Cubes encoded with the vector (`dsr`,`dtack`,`ldtack`,`d`,`lds`).

*2. if $x$ concurrent to $p$ then* $\mathcal{C}_p^x = -$;

*where $\mathcal{C}^x$ indicates the $x$-th component bit of $\mathcal{C}$.*

Given a place $p$, a literal must appear in the cube for any non-concurrent signal $x$ to $p$ ($(x,p) \notin \mathcal{SCR}$). For any arbitrary place $p$ the value of the signal $x$ in a corresponding cover cube is determined by checking if $p$ is interleaved between pairs of adjacent rise-fall or fall-rise transitions. Property 3 guarantees that the value of signal $x$ is the same for all the adjacent pairs for which $p$ is in $\mathcal{IR}$. Therefore, the *Interleave Relation* gives a polynomial-time algorithm (for free-choice STG) to determine the value of literal $\mathcal{C}_p^x$:

$$\mathcal{C}_p^x = \begin{cases} 1 \text{ if } \exists \text{ adjacent } (x_i+, x_i-) : p \in \mathcal{IR}(x_i+, x_i-) \ , \\ 0 \text{ if } \exists \text{ adjacent } (x_i-, x_i+) : p \in \mathcal{IR}(x_i-, x_i+) \ , \\ - \text{ otherwise } . \end{cases}$$

Figure 18 shows the state machines for the STG depicted in Fig. 15. All places in this STG are implicit, thus only place names are shown. Every place in each state machine is annotated with its corresponding cover cube. Going back to Fig. 19 we can analyze cube $C_{p_8}$ = `10-0-`. MR($p_8$) is entered after firing `dsr+` thus the positive value for this signal. Neither `dtack` nor `d` fire inside MR($p_8$) thus their value remains constant. The correct value to assign is 0 since $p_8$ is interleaved between a negative and a positive transition for both signals. Finally, both `ldtack` and `lds` fire inside MR($p_8$) thus their value is undefined.

| Marking | Binary code | Cover |
|---|---|---|
| $\mathsf{MR}(p_i)$ | $\widehat{\mathsf{MR}}(p_i)$ | $\mathsf{cv}(p_i)$ |
| $\mathsf{ER}(x_{i*})$ | $\widehat{\mathsf{ER}}(x_{i*})$ | $\mathsf{cv}_{\mathsf{ER}}(x_{i*})$ |
| $\mathsf{QR}(x_{i*})$ | $\widehat{\mathsf{QR}}(x_{i*})$ | $\mathsf{cv}_{\mathsf{QR}}(x_{i*})$ |

**Table 1.** Main objects used during structural synthesis.

### 7.2 State coding verification

The USC property requires that no pair of different markings share the same binary encoding. Checking this definition requires the generation of the underlying SG of the STG and all pairwise intersection of states checked to be empty.

Chu introduced the first formal definition of the USC property at the STG-level in [5]. Each pair of markings violating the USC condition is related to the existence of a *feasible sequence* of transitions containing the same number of rising and falling transitions for all signals (called a *complementary sequence* of transitions).

**Detection of Coding Conflicts** The objective of this section is to provide an efficient algorithm to check USC. The methodology is based on the approximate analysis of the underlying SG and a set of SM of the STG. The combination of both elements allows the efficient detection of the complementary feasible sequences related to USC conflicts. Further details and proofs can be found in [19].

In addition to marked regions, the structure of the STG is applied by using a set of SM that completely covers the places in the STG. Such set is called an SM-cover, and any place in the STG should be included in at least one component of the cover.

Given the MR for all places and a SM-cover SMC, the following theorem provides *sufficient conditions* to detect USC conflicts.

**Theorem 9 (Sufficient conditions for USC).** *Given a SM-cover SMC, a STG with underlying FCLSPN satisfies the USC property if*

$$\forall\, SM \in SMC, \forall p_i, p_j \text{ covered by } SM,\ i \neq j \Rightarrow \widehat{\mathsf{MR}}(p_i) \cdot \widehat{\mathsf{MR}}(p_j) = \emptyset.$$

Theorem 9 cannot be considered a *necessary* conditions due to the existence of state machines that contain more than one token. Assume the existence of one SM with two places $p_1$ and $p_2$ that contain a token in the same marking $M$. Then $M \in \mathsf{MR}(p_1)$, $M \in \mathsf{MR}(p_2)$, and therefore $\widehat{\mathsf{MR}}(p_1) \cdot \widehat{\mathsf{MR}}(p_2) \neq \emptyset$. If it can be guaranteed that all SM in the SM-cover satisfy the *one-token-per-SM* restriction, then Theorem 9 is also a necessary condition.

Theorem 9 can be relaxed to obtain a conservative rule to check USC violations by using cover cubes. Intersections between marked regions $\widehat{\mathsf{MR}}(p_i) \cdot \widehat{\mathsf{MR}}(p_j) \neq \emptyset$ can be detected as intersections between cover cubes, *i.e.* $\mathcal{C}_{p_i} \cdot \mathcal{C}_{p_j} \neq \emptyset$. However, due to the conservative nature of the approximation, all non-empty intersections between cover cubes ($\mathcal{C}_{p_i} \cdot \mathcal{C}_{p_j} \neq \emptyset$) must be considered as a potential USC violation.

Figure 18 shows the state machines for the STG depicted in Fig. 15 annotated with its coding conflicts detected following Theorem 9. SM1 contains two coding conflicts, one between $p_8$ and $p_1$ and a second between $p_8$ and $p_2$. SM2 contains one coding conflict between $p_2$ and $p_9$. Conservatively we have to assume that the STG does not satisfy the USC property.

### 7.3 Refinement of the SG approximations

Many cube intersections detected by Theorem 9 could be *fake* due to the lossy approximation provided by cover cubes. Fake coding conflicts degrade the quality of the results obtained by the methodology. An STG satisfying the USC property may have to be encoded due to the existence of fake conflicts, and the number of state signals inserted in a STG not satisfying the USC property may be increased due to the additional conflicts to be solved.

Figure 19 shows a case of fake coding conflict. The states included in the marked regions $\mathsf{MR}(p_1)$, $\mathsf{MR}(p_2)$, and $\mathsf{MR}(p_8)$ are shadowed. The states covered by the cover cube $\mathcal{C}_{p_8}$ are grouped in the dotted region. $\mathcal{C}_{p_8}$ clearly overestimates $\mathsf{MR}(p_8)$ because it also includes $\mathsf{MR}(p_1)$, even though $\widehat{\mathsf{MR}}(p_1) \cdot \widehat{\mathsf{MR}}(p_8) = \emptyset$. On the other side, $\mathcal{C}_{p_8}$ also includes $\mathsf{MR}(p_2)$, but this is a real conflict because $\widehat{\mathsf{MR}}(p_2) \cdot \widehat{\mathsf{MR}}(p_8) \neq \emptyset$.

Fake conflicts can be detected and eliminated if the information provided by the SM-cover is correlated. Even though the process will remain conservative, experimental results demonstrate that most fake coding conflicts can be eliminated with little effort.

Fake conflict elimination relies on the observation that the information in a SM is complemented by the rest of elements in the SM-cover. A place may be covered by several SMs, containing coding conflicts in all or in none of them. In that case the information provided by the SM-cover is congruent. However, if a place has no coding conflicts in one state machine $\mathrm{SM}_i$ but has conflicts in a second state machine $\mathrm{SM}_j$, we can infer from $\mathrm{SM}_i$ that the cover cubes in $\mathrm{SM}_j$ are overestimated.

**Theorem 10 (Relaxed sufficient conditions for USC).** *Given a SM-cover SMC, a STG with underlying FCLSPN satisfies the USC property if*

$$\forall p_i \in P: \\ \exists SM \in SMC \wedge p_i \in SM : [\forall p_j \in SM,\ i \neq j \ \Rightarrow\ \widehat{\mathsf{MR}}(p_i) \cdot \widehat{\mathsf{MR}}(p_j) = \emptyset] \ .$$

This result can be exported to the particular case in which the binary codes in the marked regions are approximated by cover cubes. Note that the empty intersection between pairs of places $\mathcal{C}_{p_i} \cdot \mathcal{C}_{p_j} = \emptyset$ is only a sufficient condition to determine that no coding conflict exists.

From the previous theorem we can deduce that places without coding conflicts in one SM can be used to eliminate fake conflicts in some other SM. The fake conflict elimination can be applied in a single step by applying Theorem 10, thus eliminating the coding conflicts for a place $p_i$ in all state machines that cover that place if a SM covering $p_i$ exists such that the cover cube of no other place in SM interests with $\mathcal{C}_{p_i}$.

This restricted view of the conflict elimination technique can be extended by iteratively applying Theorem 10. It can be considered that for a given pair of places $\widehat{\mathsf{MR}}(p_i) \cdot \widehat{\mathsf{MR}}(p_j) = \emptyset$ (even though $\mathcal{C}_{p_i} \cdot \mathcal{C}_{p_j} \neq \emptyset$) if we have previously eliminated all the coding conflicts for place $p_i$ or $p_j$.

The analysis of the coding conflicts in Fig. 18 shows that the coding conflict between places $p_1$ and $p_8$ at $\mathrm{SM}_1$ is a *fake conflict*. Note that $p_1$ has a conflict at $\mathrm{SM}_1$ but has no conflicts at $\mathrm{SM}_2$, therefore the cover cube for $p_8$ is overestimating $\mathsf{MR}(p_8)$.

Given that a single cube approach is clearly insufficient we introduce a new approximation function for places, named *cover function* $\mathsf{cv}(p)$. Initially, cover functions will take the same value than cover cubes, i.e. $\mathsf{cv}(p) = \mathcal{C}_p$. However, the value of the cover function can be iteratively refined every time a fake conflict is eliminated.

Formally, the *refinement* of the cover function $\mathsf{cv}(p)$ by a SM is an algorithm that builds a new cover as a result of restricting $\mathsf{cv}(p)$ to the sum of the cover functions of any place $p_i \in \mathrm{SM}$ that is concurrent to $p$; that is:

$$\mathsf{cv}(p) \leftarrow \sum_{p_i \in \mathrm{SM} \,:\, (p,p_i) \in \mathcal{CR}} \mathsf{cv}(p_i) \cdot \mathsf{cv}(p) \ .$$

Place $p_8$ can be refined by using the information provided by $\mathrm{SM}_2$. Places $p_9$, $p_{10}$ and $p_{11}$ are concurrent to $p_8$, so $\mathsf{cv}(p_8)$ should be refined as

$$\mathsf{cv}(p_8) \leftarrow \mathsf{cv}(p_8) \cdot [\mathsf{cv}(p_9) + \mathsf{cv}(p_{10}) + \mathsf{cv}(p_{11})] = \mathtt{10\text{-}00} + \mathtt{1010\text{-}} \ .$$

Therefore $\mathsf{cv}(p_8)$ no longer includes $\mathsf{MR}(p_1)$ and the fake conflict is no longer present.

Property 2 guarantees that the refinement procedure is safe; that is, no marking in the marked region of a place can be left uncovered after applying a refinement because all reachable markings are covered by some $\mathsf{cv}(p)$ in the SM used for refinement.

### 7.4 Structural approximations for signal regions

The structural generation of Boolean equations for SI circuits that satisfy the synthesis conditions described in Section 6 requires conservative approximations of the signal regions $\mathsf{ER}(x_i*)$ and $\mathsf{QR}(x_i*)$.

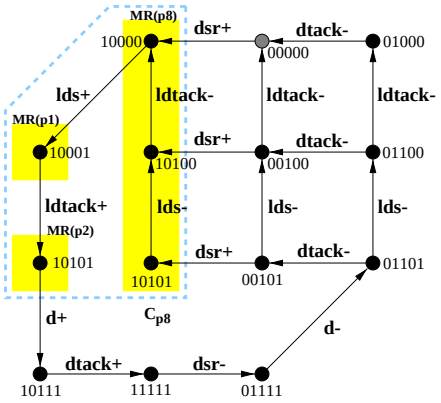



**Fig. 19.** Real and fake USC coding conflicts.

We define $\mathsf{cv}_{\mathsf{ER}}(x_{i*})$ to be a cover function for the binary codes in $\widehat{\mathsf{ER}}(x_{i*})$; and $\mathsf{cv}_{\mathsf{QR}}(x_{i*})$ a cover function for the binary codes in $\widehat{\mathsf{QR}}(x_{i*})$. This section will show how to build both $\mathsf{cv}_{\mathsf{ER}}(x_{i*})$ and $\mathsf{cv}_{\mathsf{QR}}(x_{i*})$ by using the cover functions of individual places previously introduced in Section 7.1.

An *excitation region* $\mathrm{ER}(x_{i*})$ corresponds to the set of markings in which transition $x_{i*}$ is enabled. $\mathrm{ER}(x_{i*})$ is easily expressed as the intersection of marked regions for input places of $x_{i*}$:

$$\mathrm{ER}(x_{i*}) = \bigcap_{p \in {}^\bullet(x_{i*})} \mathsf{MR}(p) \ .$$

Let $\mathrm{EPS}(x_{i*})$ be the set of predecessor places of $x_{i*}$, hence the binary codes in $\widehat{\mathrm{ER}}(x_{i*})$ are covered by a function $\mathsf{cv}_{\mathsf{ER}}(x_{i*})$ created as the intersection of the cover functions of places in $\mathrm{EPS}(x_{i*})$:

$$\mathsf{cv}_{\mathsf{ER}}(x_{i*}) = \bigcap_{p \in \mathrm{EPS}(x_{i*})} \mathsf{cv}(p) \ .$$

A marking is in the *quiescent region* $\mathrm{QR}(x_{i*})$ if it can be reached by firing a feasible sequence $\sigma_1 x_{i*}\, \sigma_2$ such that no successor transition $x_{j*} \in next(x_{i*})$ is enabled in any prefix of $\sigma_2$. In the previous sections we have shown that feasible sequences can be associated to simple paths in the PN. Hence we can informally state that the cover functions of all places interleaved between adjacent transitions $x_{i*}$ and $x_{j*} \in next(x_{i*})$ (see Property 1) can be used to build $\mathsf{cv}_{\mathsf{QR}}(x_{i*})$.

Hence, the domain of places required to approximate $\mathsf{QR}(x_{i*})$ should include all places interleaved between $x_{i*}$ and some transition $x_{j*} \in next(x_{i*})$. This domain is denoted $\mathrm{QPS}(x_{i*})$, i.e.

$$\mathrm{QPS}(x_{i*}) = \{p \mid \exists x_{j*} \in next(x_{i*}) : p \in \mathcal{IR}(x_{i*}, x_{j*}) \wedge p \notin {}^\bullet x_{j*}\} \ .$$

The binary codes in $\widehat{\mathsf{QR}}(x_{i*})$ are covered by a function $\mathsf{cv_{QR}}(x_{i*})$ created as the union of the cover functions of places in $\mathrm{QPS}(x_{i*})$:

$$\mathsf{cv_{QR}}(x_{i*}) = \bigcup_{p \in \mathrm{QPS}(x_{i*})} \mathsf{cv}(p) \ .$$

As an example we will build the $\mathsf{cv_{ER}}$ and $\mathsf{cv_{QR}}$ (see Fig. 20) approximations for signal dtack in Fig. 15. $\mathsf{ER}$(dtack+) and $\mathsf{cv_{ER}}$(dtack-) is build as $\mathsf{cv_{ER}}(\mathtt{dtack+}) = \mathsf{cv}(p_3) = \mathtt{10111}$ , and $\mathsf{cv_{ER}}(\mathtt{dtack-}) = \mathsf{cv}(p_6) = \mathtt{01-0-}$ . The covers $\mathsf{cv_{QR}}$(dtack+) and $\mathsf{cv_{QR}}$(dtack-) are approximated by the sets of places $\mathrm{QPS}(\mathtt{dtack+}) = \{p_4, p_5\}$ , and $\mathrm{QPS}(\mathtt{dtack-}) = \{p_7, p_8, p_1, p_2\}$ .

The resulting covers are: $\mathsf{cv_{QR}}(\mathtt{dtack+}) = \mathsf{cv}(p_4) + \mathsf{cv}(p_5) = \mathtt{-1111}$ , and $\mathsf{cv_{QR}}(\mathtt{dtack-}) = \mathsf{cv}(p_7) + \mathsf{cv}(p_8) + \mathsf{cv}(p_1) + \mathsf{cv}(p_2) = \mathtt{00-0-} + \mathtt{10-00} + \mathtt{1010-} + \mathtt{10-01}$ .

Starting from this region approximations logic minimization can be applied in order to derive the final logic equations that implement signal dtack (see Section 6). The on-set of the signal is build as:

$$\mathsf{ON}(\mathtt{dtack}) = \mathsf{cv_{ER}}(\mathtt{dtack+}) \cup \mathsf{cv_{QR}}(\mathtt{dtack+}) = \mathtt{1-111} + \mathtt{-1111}$$

and the off-set as:

$$\mathsf{OFF}(\mathtt{dtack}) = \mathsf{cv_{ER}}(\mathtt{dtack-}) \cup \mathsf{cv_{QR}}(\mathtt{dtack-}) = \mathtt{0--0-} + \mathtt{10-00} + \mathtt{1010-} + \mathtt{10-01}$$

In case we choose to implement the on-set of the function we will obtain a circuit equivalent to the following logic equation:

$$\mathtt{dtack}^{+} = (\mathtt{dsr} + \mathtt{dtack}) \cdot \mathtt{ldtack} \cdot \mathtt{d} \cdot \mathtt{lds} \ .$$

## 8 State encoding

One of the conditions for the SI implementation of an specification is to have a correct encoding (see Section 6). This section presents a technique for transforming an specification in order to force a correct encoding. The method presented guarantees a solution for the encoding problem and tackles the problem in linear complexity for the class of FCLSPNs. The technique is based on the insertion of a signal for each place of the STG, mimicking the token flow on that place.

### 8.1 A structural encoding transformation

The method presented has been inspired on previous work for the direct synthesis of circuits from Petri nets. One of the relevant techniques was proposed in [23], where a set of cells that mimic the token flow of the Petri net was

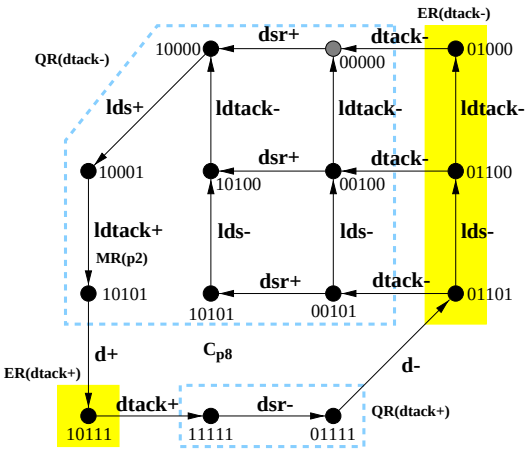



**Fig. 20.** Regions for output signal `dtack`.

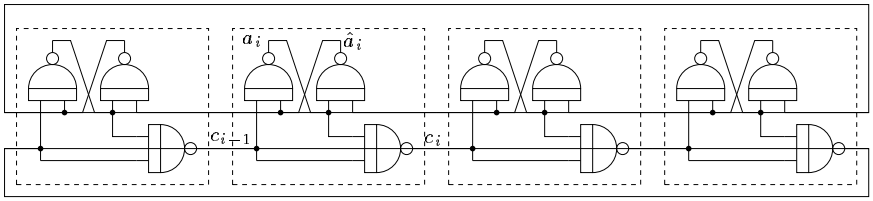



**Fig. 21.** Distributor built from David cells [11].

abutted for producing a circuit structure isomorphic to the original net. This type of cells, called David cells, were initially proposed in [8].

Figure 21 depicts a very simple example on how these cells can be abutted to build a distributor that controls the propagation of activities along a ring. The behavior of one of the cells in the distributor can be summarized by the following sequence of events:

$$\cdots \rightarrow \underbrace{c_{i-1}-}_{\substack{i\text{-th cell}\\ \text{excitation}}} \rightarrow \underbrace{a_i+ \rightarrow \hat{a}_i-}_{i\text{-th cell setting}} \rightarrow$$

$$\rightarrow \underbrace{\hat{a}_{i-1}+ \rightarrow a_{i-1}- \rightarrow c_{i-1}+}_{(i-1)\text{-th cell resetting}} \rightarrow \underbrace{c_i-}_{\substack{(i+1)\text{-th cell}\\ \text{excitation}}} \rightarrow \cdots$$

In [23], each cell was used to represent the behavior of one of the transitions of the Petri net. The approach presented in this paper is based on

1. Create the *silent* transitions $\varepsilon_1$ and $\varepsilon_2$.

2. For each place $p \in {}^\bullet t$, create a new transition with label $sp-$ and insert new arcs and places for creating a simple path from $\varepsilon_1$ to $\varepsilon_2$, passing through $sp-$.

3. For each place $p \in t^\bullet$, substitute the arc $(t, p)$ by the arc $(\varepsilon_2, p)$, create a new transition labeled as $sp+$ and insert new arcs and places for creating a simple path from $t$ to $\varepsilon_1$, passing through $sp+$.

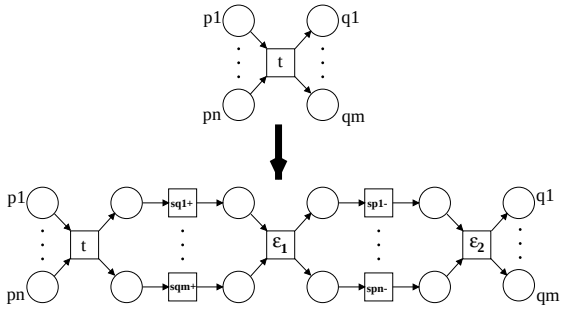



**Fig. 22.** Transformation rule for each transition $t \in T$.

encoding the system by inserting a new signal for each place with a behavior similar to a David cell.

Let $S = \langle \langle P, T, F, M_0 \rangle, \mathcal{X}, \Lambda \rangle$ be an STG with underlying FCLSPN. The Structural Encoding of $S$ derives the STG $Enc(S)$ in which a new internal signal $sp$ has been created for each place $p \in P$, and the transformation rule described in Figure 22 has been applied to each transition $t \in T$. The new transtions appearing in $Enc(S)$, labelled with $sp*$, will be called *E-transitions* along the paper.

**Proposition 5 ([4]).** *$Enc(S)$ is FCLSPN. It is consistent and observational equivalent to $S$, and has the USC property.*

Proposition 5 guarantees the fulfillment of the main properties needed in the synthesis framework presented in Section 7.

### 8.2 I/O-preserving structural encoding transformation

The previous section presents a method for transforming a specification by the insertion of internal signals. In order to reason about the correctness of the method, i.e. whether the transformed specification can safely interact in its assumed environment, we can check if the transformed specification and the environment (the mirror of the initial specification) are I/O compatible.

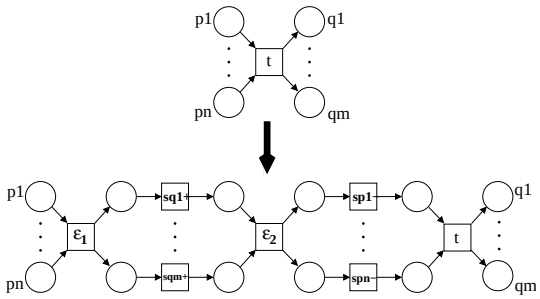



**Fig. 23.** Transformation rule for non-input signals to preserve the I/O interface.

This is analog to checking whether the transformed specification realizes the initial specification (see Definition 13).

However, the encoding technique presented in the previous section does not guarantee to preserve the I/O preserving realization with respect to the initial STG, because condition 2(a) of the I/O preserving realization can be violated in the interaction between $S$ and $Enc(S)$. In this section we present a refinement of the encoding technique presented in Section 8.1, deriving a new encoding method closed under the I/O preserving realization. The refinement presented below is only valid for the IO-STG class. In that class, the transformation rule shown in Figure 23 can be applied to any transition of a non-input signal. For input transitions, the previous transformation presented in Figure 22 is applied. This refined encoding technique is called *IO-Enc*$(S)$.

Note that the two transformations ($Enc(S)$ and *IO-Enc*$(S)$) only differ on the location of the E-transitions. For non-input signals, the E-transitions precede the transformed transition.

The proofs for preserving free-choiceness, liveness, safeness, consistency, observational equivalence and ensuring USC are similar to those presented in the previous section when applied to the class of IO-STGs [4]. It is important to note that this new technique is closed under the I/O preserving realization:

**Proposition 6.** *Let $S$ be an IO-STG with underlying FCLSPN and $SG(S)$ input-proper. Then $S \models$ IO-Enc$(S)$.*

*Proof.* The encoding preserves both the input-properness and the observational equivalence. Theorem 5 guarantees the I/O preserving realization between $S$ and *IO-Enc*$(S)$

Moreover, the *IO-Enc* technique ensures to fulfill the speed-independent conditions for implementability.

**Proposition 7 ([4]).** *Let $S$ be a consistent and output-persistent IO-STG with underlying FCLSPN. Then IO-Enc$(S)$ fulfills the speed-independent conditions.*

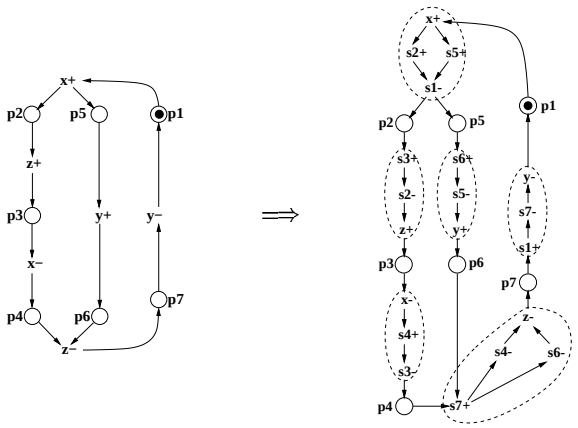



**Fig. 24.** Structural encoding ($x$ is input and $y$ and $z$ are outputs).

Figure 24 depicts an example of the I/O preserving structural encoding.

## 9 Framework for automatic synthesis. Example

Some of the structural techniques presented in previous sections can be combined to derive a structural framework for the synthesis of asynchronous circuits. The main properties of the framework presented below are:

1. It always guarantees to find a solution.
2. Only polynomial algorithms are involved.
3. The quality of the solutions obtained is comparable to the methods that require an explicit enumeration of the state space.

### 9.1 Design flow

This section presents an automatic methodology that starts from an IO-STG specifying the control part of a circuit and ends up with a set of Boolean equations both implementing the specification and fulfilling the SI conditions. The core of the framework is the idea of Petri net transformation: in the initial step, we apply the encoding transformation of Section 8.2 which ensures to have a correct encoding and preserves the I/O preserving realization with respect to the specification. Then, Petri net transformations (those from the kit presented in Section 4.1) are applied iteratively in order to improve the quality of the solution. Figure 25 presents the framework.

A natural strategy that can be mapped in the framework of Figure 25 is to try to eliminate the maximum number of signals inserted in the encoding step while preserving a correct encoding. Once no more signals can be deleted, start to apply iteratively any transformation from the kit until an admissible solution is obtained. In this way, an exploration of the solution space is performed iteratively, by modifying locally causal dependencies and concurrency relations between the events of the system.

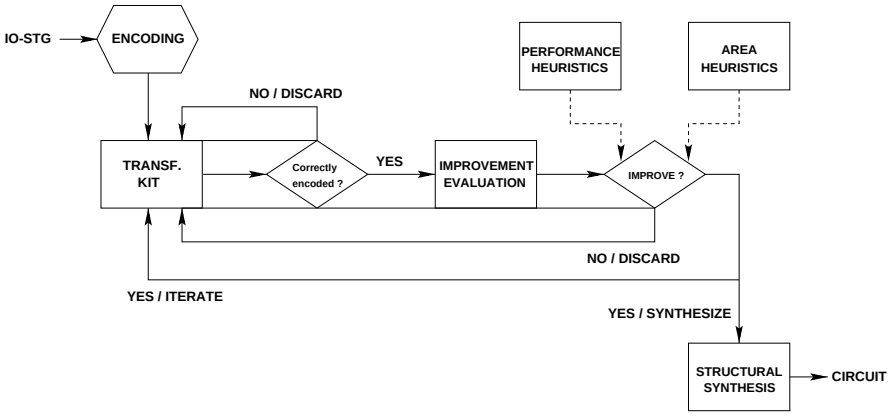



**Fig. 25.** Framework for the structural synthesis of asynchronous circuits.

### 9.2 An example

Let us present an example of application of the framework presented in the previous section. The example corresponds to a specification of an *analog-to-digital fast converter* with three input signals (Da, La and Za) and three output signals (Dr, Lr and Zr). The specification is shown in Figure 26(a). This IO-STG does not have a correct encoding. Figure 26(d) shows the IO-STG obtained after applying the structural encoding rules. The new internal signals $s_0 ... s_{14}$ correspond to the 15 places in the initial specification.

From the IO-STG in Figure 26(d), a sequence of eliminations of internal signals have been applied, always preserving a correct encoding. The resulting IO-STG and the corresponding circuit are shown in Figure 27(a), where only five extra signals remain. A typical way of measuring the quality of the circuit is to estimate the area needed for its implementation; a good heuristic is the number of literals in the Boolean equations. In the case of the IO-STG in Figure 27(a), this number is 35.

Figure 27(b) reports one of the intermediate solutions (31 literals) explored after obtaining the solution in Figure 27(a). The transformations of increase and reduction of concurrency have also been applied in the exploration: note in Figures 27(a)-(b) the position of transition $s_6-$ with respect to transitions $Lr+$ and $La+$. Figures 26(b) and 26(c) depict the final IO-STG, the Boolean equations and the circuit after applying the transformations and doing logic synthesis. This solution, which has been obtained mechanically, is identical to the one generated by the CAD tool petrify [6].

### 9.3 Experimental results

The synthesis strategy described above has been applied to a set of benchmarks. Initially, none of the specifications had the CSC property. The results are reported in Table 2.

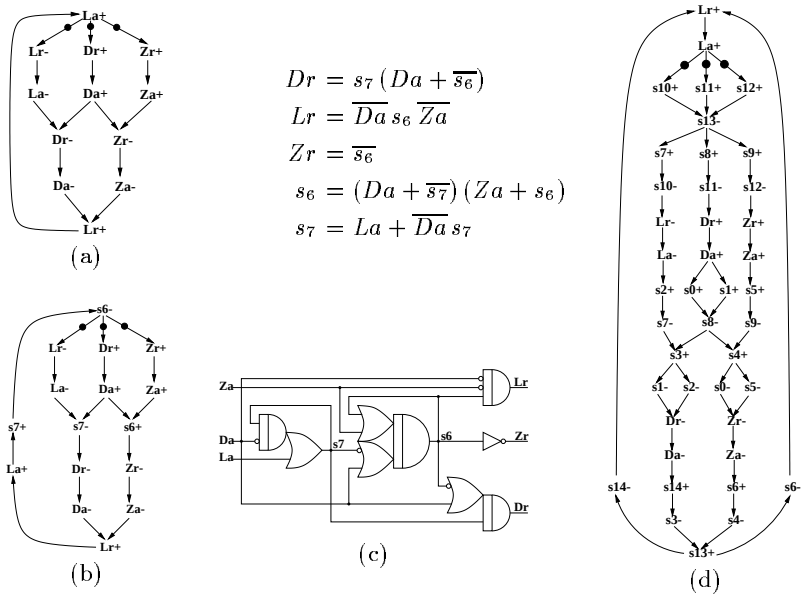



**Fig. 26.** Case study: adfast

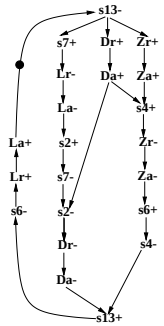
The columns labeled with *"petrify"* indicate the characteristics of the circuit obtained by the tool petrify. The number of inserted signals to solve CSC conflicts and the number of literals of the Boolean equations are reported.

The columns labeled with *"struct. encoding"* report the characteristics of the circuit after having applied first the encoding technique and then the elimination of internal signals. It is interesting to observe that the number of signals required to solve encoding conflicts when using the "local" encoding provided by the places is significantly larger than the number of signals required when "global" encoding methods are used.

The results of the final circuit, after having explored the design space with the set of transformations, are reported in the columns labeled *"str. enc. + optim."*. It can be observed that the quality of the solution can be highly improved by playing with the concurrency of the internal signals. In many cases, the obtained result is the same as the one generated by petrify. In other cases, the results are similar but with more internal signals than the ones inserted by petrify(e.g. master-read2, duplicator). This corroborates a known fact that states that the reduction of internal signals does not always implies an improvement on the quality of the circuit.

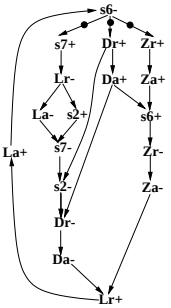
## 10 Conclusions

Asynchronous circuits are just a subclass of reactive systems in which the state is represented by a vector of Boolean variables. However, the underlying




$$
\begin{aligned}
Dr &= \overline{s_{13}}\, La + s_2 + s_7 \\
Lr &= \overline{s_7}\, La + s_{13}\, \overline{s_6} \\
Zr &= \overline{s_{13}}\, \overline{s_6}\, \overline{s_4} \\
s_2 &= \overline{Da}\, s_2 + s_7\, \overline{La} \\
s_4 &= Za\, (\overline{Lr}\, \overline{s_2}\, \overline{s_7} + Da) + s_4\, \overline{s_6} \\
s_6 &= \overline{s_{13}}\, s_6 + \overline{Za}\, s_4 \\
s_7 &= \overline{s_{13}}\, Lr + \overline{s_2}\, s_7 \\
s_{13} &= \overline{s_4}\, s_6\, \overline{Da} + s_{13}\, \overline{La}
\end{aligned}
$$

(a)




$$
\begin{aligned}
Dr &= La\, (\overline{s_6} + Dr) + \overline{Da}\, Dr + s_2 + s_7 \\
Lr &= \overline{Da}\, s_6\, \overline{Za} + Lr\, \overline{s_7} \\
Zr &= \overline{s_6} \\
s_2 &= \overline{Dr}\, s_2 + \overline{Lr}\, s_7 \\
s_6 &= Da\, (s_6 + Za) + \overline{La}\, (\overline{Dr}\, Za\, \overline{s_2}\, \overline{s_7} + s_6) \\
s_7 &= La\, (\overline{s_6} + Da) + \overline{s_2}\, s_7
\end{aligned}
$$

(b)

**Fig. 27.** Intermediate solutions in the design space.

| | | *petrify* | | *struct. encoding* | | *str. enc. + optim.* | |
|---|---|---|---|---|---|---|---|
| benchmark | states | #CSC | lit. | #CSC | lit. | #CSC | lit. |
| adfast | 44 | 2 | 14 | 5 | 35 | 2 | 14 |
| vme-fc-read | 14 | 1 | 8 | 2 | 14 | 1 | 8 |
| nak-pa | 56 | 1 | 18 | 3 | 35 | 1 | 18 |
| m-read1 | 1882 | 1 | 38 | 2 | 43 | 1 | 40 |
| m-read2 | 8932 | 8 | 68 | 13 | 95 | 10 | 70 |
| duplicator | 20 | 2 | 18 | 5 | 36 | 3 | 18 |
| mmu | 174 | 3 | 29 | 7 | 53 | 3 | 34 |
| seq8 | 36 | 4 | 47 | 22 | 147 | 4 | 47 |

**Table 2.** Experimental results.

theory to synthesize asynchronous circuits can be extended to other classes of systems.

This work has defined I/O compatibility as a key concept to characterize the correct interaction between system and environment. The rationale behind this concept is the following: one can always decide when an output or internal action must be performed as long as this does not break the correct interaction with the environment. This freedom to choose *"when"*, opens a

design space that can be explored to obtain the most appropriate realization for a behavior.

Typically, synthesis must consider a trade-off between complexity and performance. By sequentializing concurrent behaviors, one can derive systems with less complexity, at the expense of reducing the performance. By allowing more concurrent behaviors, the system's cost may increase, but possibly providing more performance. It is mainly the degree of concurrency what opens the space of solutions and drives the strategies to explore it.

## References


1. A. Arnold. *Finite Transition Systems.* Prentice Hall, 1994.
2. G. Berthelot. Checking Properties of Nets Using Transformations. In G. Rozenberg, editor, *Advances in Petri Nets 1985*, volume 222 of *Lecture Notes in Computer Science*, pages 19–40. Springer-Verlag, 1986.
3. Janusz A. Brzozowski and Carl-Johan H. Seger. *Asynchronous Circuits.* Springer-Verlag, 1995.
4. J. Carmona, J. Cortadella, and E. Pastor. A structural encoding technique for the synthesis of asynchronous circuits. In *Int. Conf. on Application of Concurrency to System Design*, June 2001.
5. Tam-Anh Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications.* PhD thesis, MIT Laboratory for Computer Science, June 1987.
6. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, March 1997.
7. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. *Logic synthesis of asynchronous controllers and interfaces.* Springer-Verlag, 2002. to appear.
8. René David. Modular design of asynchronous circuits defined by graphs. *IEEE Transactions on Computers*, 26(8):727–737, August 1977.
9. M. Hack. *Analysis of production schemata by Petri nets.* M.s. thesis, MIT, February 1972.
10. D. Harel and A. Pnueli. On the development of reactive systems. In Krzystof R. Apt, editor, *Logic and Model of Concurrent Systems*, volume 13 of *NATO ASI*, pages 477–498. Springer-Verlag, October 1984.
11. Michael Kishinevsky, Alex Kondratyev, Alexander Taubin, and Victor Varshavsky. *Concurrent Hardware: The Theory and Practice of Self-Timed Design.* Series in Parallel Computing. John Wiley & Sons, 1994.
12. Alex Kondratyev and Alexander Taubin. Verification of speed-independent circuits by stg unfoldings. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 64–75, November 1994.
13. A. Kovalyov and J. Esparza. A polynomial algorithm to compute the concurrency relation of free-choice signal transition graphs. In *Proceedings of the International Workshop on Discrete Event Systems, WODES'96*, pages 1–6, August 1996.

14. A. V. Kovalyov. On complete reducibility of some classes of Petri nets. In *Proceedings of the 11th International Conference on Applications and Theory of Petri Nets*, pages 352–366, Paris, June 1990.
15. Nancy A. Lynch and Mark R. Tuttle. An introduction to input/output automata. In *CWI-Quarterly*, volume 2, pages 219–246, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands, September 1989.
16. R. Milner. *A Calculus for Communicating Processes*, volume 92 of *Lecture Notes in Computer Science*. Springer Verlag, 1980.
17. Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–574, April 1989.
18. Enric Pastor. *Structural Methods for the Synthesis of Asynchronous Circuits from Signal Transition Graphs*. PhD thesis, Universitat Politècnia de Catalunya, February 1996.
19. Enric Pastor, Jordi Cortadella, Alex Kondratyev, and Oriol Roig. Structural methods for the synthesis of speed-independent circuits. *IEEE Transactions on Computer-Aided Design*, 17(11):1108–1129, November 1998.
20. C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Bonn, Institut für Instrumentelle Mathematik, 1962. (technical report Schriften des IIM Nr. 3).
21. L. Y. Rosenblum and A. V. Yakovlev. Signal graphs: from self-timed to timed ones. In *Proceedings of International Workshop on Timed Petri Nets*, pages 199–207, Torino, Italy, July 1985. IEEE Computer Society Press.
22. Manuel Silva, Enrique Teruel, and José Manuel Colom. Linear algebraic and linear programming techniques for the analysis of place/transition net systems. *Lecture Notes in Computer Science: Lectures on Petri Nets I: Basic Models*, 1491:309–373, 1998.
23. Victor I. Varshavsky, editor. *Self-Timed Control of Concurrent Processes: The Design of Aperiodic Logical Circuits in Computers and Discrete Systems*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1990.

# A Proofs of Sections 3 and 4

*Proof.* (of Lemma 1) If $\langle s, s' \rangle \in S$, then there is a trace $\sigma$ that leads from $s_{in}$ to $\langle s, s' \rangle$. We prove the lemma by induction on the length of $\sigma$.

- Case $|\sigma| = 0$. The initial states are related in Condition 1 of Definition 12.
- Case $|\sigma| > 0$. Let $\sigma = \sigma' e$, with $|\sigma'| = n$, and assume that it holds for any trace up to length $n$. Let $\langle s_1, s_1' \rangle$ be the state where the event $e$ is enabled. The induction hypothesis ensures that $s_1$ is I/O compatible to $s_1'$. Two situations can happen in $s_1$ depending on the last event $e$ of $\sigma$: either 1) $e \in \Sigma_O \cup \Sigma_{INT}$ is enabled in $s_1$, or 2) only input events are enabled in $s_1$. In situation 1), Conditions 2-3 of Definition 12 guarantee that $s$ is I/O compatible to $s'$. In situation 2), applying Condition 4 of Definition 12 ensure that some non-input event is enabled in state $s_1'$ of $B$. Definition 2 and Conditions 2-3 on $s_1'$ and the enabled non-input event $e$ guarantees $s$ to be I/O compatible to $s'$. □

*Proof.* (of Theorem 1) It immediately follows from Lemma 1 and the condition of receptiveness in the definition of I/O compatibility. □

*Proof.* (of Theorem 2) The definition of synchronous product implies that only livelocks appear in $A \times B$ if either $A$ or $B$ has a livelock. But $A$ and $B$ are livelock-free because $A \rightleftharpoons B$. □

*Proof.* (of Theorem 3) By Lemma 1 we have that $sRs'$. We also have that $\{e \mid \mathsf{En}(s, e)\} \subseteq \Sigma_I^A$. By Condition 4 of Definition 12 we know that $\{e \mid \mathsf{En}(s_1', e)\} \nsubseteq \Sigma_I^B$. Theorem 2 guarantees the livelock-freeness of $A \times B$, and therefore from $\langle s, s' \rangle$ there exists a trace of internal events reaching a state $\langle s, s'' \rangle$ where no internal event is enabled. We know by Lemma 1 that $sRs''$. Condition 4 of Definition 12, together with the fact that no internal event is enabled in $s''$ implies that there exists an output event enabled in $s''$, which is enabled as input in $s$. □

*Proof.* (of Theorem 4) Let $R$ be the relation induced by the observational equivalence between $A$ and $B$. We will prove that $R$ is also an I/O compatibility relation between $A$ and $B$. $R$ must fulfill the conditions of the I/O compatibility relation:

- **Condition 1:** $s_{in}^A R s_{in}^B$ by Definition 10.
- **Condition 2(a):** let $s_1 R s_1'$, and assume $s_1 \xrightarrow{e} s_2$, with $e \in \Sigma_O^A$. Figure 28(a) depicts the situation. The observational equivalence of $s_1$ and $s_1'$ implies that a trace $\sigma$ of internal events exists in $s_1'$ enabling $e$. The event $e$ is an input event in $B$, and therefore the input-properness of $B$ ensures that in every state $s'$ of $\sigma$, $\mathsf{En}(s', e)$ holds. In particular, it also holds in the first state and, thus, $\mathsf{En}(s_1', e)$. The definition of $R$ ensures that every $s_2'$ such that $s_1' \xrightarrow{e} s_2'$ is related with $s_2$ by $R$.

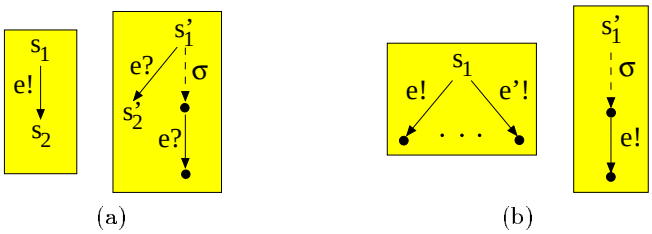



**Fig. 28.** Conditions 2(a) and 4(a) from the proof of Theorem 4.

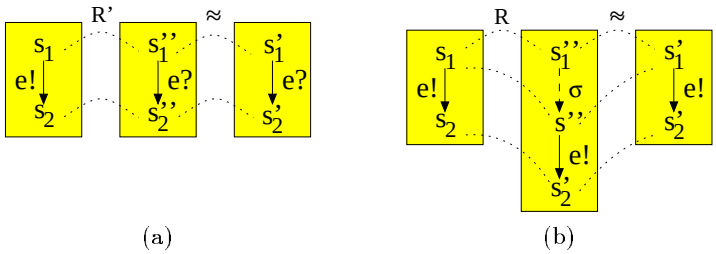



**Fig. 29.** Conditions 2(a) and 2(b) from the proof of Theorem 5.

- **Condition 3(a):** let $s_1 R s'_1$ and assume $s_1 \xrightarrow{e} s_2$, with $e \in \Sigma^A_{INT}$. The definition of $R$ implies that $s_2 R s'_1$.
- **Condition 4(a):** let $s_1 R s'_1$, and suppose $\{e \mid \mathsf{En}(s_1, e)\} \subseteq \Sigma^A_I$. Figure 28(b) depicts the situation. Let $e$ be one of the input events enabled in $s_1$. The observational equivalence between $s_1$ and $s'_1$ requires that a sequence $\sigma$ of internal events exists enabling $e$ starting in $s'_1$, and given that $e$ in not input in $B$ implies $\{e \mid \mathsf{En}(s'_1, e)\} \not\subseteq \Sigma^B_I$.

An identical reasoning can be applied in the symmetric cases (conditions 2(b), 3(b) and 4(b)). □

*Proof.* (of Theorem 5) Let $R'$ be the relation between $A$ and $B$, and $\approx$ the observational equivalent relation between states from $B$ and $C$. Define the relation $R$ as:

$$\forall s \in S^A, s'' \in S^B, s' \in S^C : sR's'' \wedge s'' \approx s' \Leftrightarrow (s, s') \in R$$

The conditions that $R$ must satisfy are the ones of Definition 12. Remember that $A \rightleftharpoons B$ implies that $\Sigma^B_O = \Sigma^A_I$ and $\Sigma^B_I = \Sigma^A_O$. Moreover, relation $B \approx C$ implies that $\Sigma^B_{OBS} = \Sigma^C_{OBS}$.

- **Condition 1:** the initial states are related in $R$ by definition.

- **Condition 2(a):** let $s_1 R s'_1$, and suppose $s_1 \xrightarrow{e} s_2$ with $e \in \Sigma_O^A$. Figure 29(a) depicts the situation. Given that $s_1 R' s''_1$, $e$ is enabled in $s''_1$ and for each $s''_2$ such that $s''_1 \xrightarrow{e} s''_2$, $s_2 R' s''_2$. The observational equivalence of $s''_1$ and $s'_1$, together with the fact that $C$ is input-proper implies that $e$ is also enabled in $s'_1$ (identical reasoning of condition 2(a) in Theorem 4), and the definition of $\approx$ implies that each $s'_2$ such that $s'_1 \xrightarrow{e} s'_2$ must be related in $\approx$ with $s''_2$. Then each $s'_2$ such that $s'_1 \xrightarrow{e} s'_2$ is related by $R$ with $s_2$.
- **Condition 2(b):** let $s_1 R s'_1$, and suppose $s'_1 \xrightarrow{e} s'_2$ with $e \in \Sigma_O^B$. Figure 29(b) depicts the situation. The observational equivalence of $s''_1$ and $s'_1$ implies that there is a sequence $\sigma$ of internal events starting in $s''_1$ and enabling $e$, and every state of $\sigma$ is observational equivalent to $s'_1$. Moreover, every state of $\sigma$ is also related to $s_1$ by the condition 3(b) of $R'$. In particular, $s_1$ is related by $R'$ with the state $s''$ of $\sigma$ s.t. $s'' \xrightarrow{e} s''_2$; applying Condition 2(b) of $R'$, $\mathsf{En}(s_1, e)$ holds and for each $e$ s.t. $s_1 \xrightarrow{e} s_2$, $s_2 R' s''_2$. The definition of $R$ and $\approx$ induces that each such $s_2$ is related with $s'_2$ by $R$.
- **Condition 3(a):** let $s_1 R s'_1$, and suppose $s_1 \xrightarrow{e} s_2$ with $e \in \Sigma_{INT}^A$. Then Condition 3(a) of $R'$ ensures $s_2 R' s''_1$ and then applying the definition of $R$ implies $s_2 R s'_1$.
- **Condition 3(b):** let $s_1 R s'_1$, and suppose $s'_1 \xrightarrow{e} s'_2$ with $e \in \Sigma_{INT}^C$. Then $s''_1 \approx s'_2$, and then $s_1 R s'_2$.
- **Condition 4(a):** let $s_1 R s'_1$, and suppose $\{e | \mathsf{En}(s_1, e)\} \subseteq \Sigma_I^A$. Condition 4(a) of $R'$ ensures that $\{e | \mathsf{En}(s''_1, e)\} \not\subseteq \Sigma_I^B$: let $a$ be an event such that $s''_1 \xrightarrow{a} s''_2$, with $a \notin \Sigma_I^B$. If $a \in \Sigma_O^B$, the related pair $s''_1 \approx s'_1$ ensures that in $s'_1$ there is a feasible sequence of internal events (which can be empty) enabling $a$, and therefore $\{e | \mathsf{En}(s'_1, e)\} \not\subseteq \Sigma_I^C$. If $a \in \Sigma_{INT}^B$, applying Condition 3(b) of $R'$ and the definition of $\approx$, $s_1 R' s''_2$ and $s''_1 \approx s'_1$ is obtained, respectively. The same reasoning applied to $s_1$, $s''_1$ and $s'_1$ can now be applied to $s_1$, $s''_2$ and $s'_1$. Given that $B$ is livelock-free, the sequence of internal events starting in $s''_1$ and passing through $s''_2$ must end in a state $s''$ where a observable event $a'$ is enabled. State $s''$ is also related by $R'$ with $s_1$, and by $\approx$ with $s'_1$ (applying inductively the same reasoning applied to $s''_2$). Event $a'$ belongs to $\Sigma_O^B$ because otherwise a violation of Condition 2(b) in $R'$ arise. The previous case ($a \in \Sigma_O^B$, enabled in $s''_1$) can be applied to $s''$.
- **Condition 4(b):** let $s_1 R s'_1$, and suppose $\{e | \mathsf{En}(s'_1, e)\} \subseteq \Sigma_I^C$. Let $a$ such that $s''_1 \xrightarrow{a} s''_2$. If $a \in \Sigma_O^B$, then a contradiction arise because $s''_1 \approx s'_1$ and $\{e | \mathsf{En}(s'_1, e)\} \subseteq \Sigma_I^C$. If $a \in \Sigma_I^B$, then identical conditions make $\mathsf{En}(s'_1, a)$ to hold. If $a \in \Sigma_{INT}^B$, then Conditions 3(a) of $R'$ and $\approx$ ensure that $s_1 R' s''_2$ and $s''_2 \approx s'_1$, and the same reasoning of $s_1$, $s'_1$ and $s''_1$ can be applied to $s_1$, $s'_1$ and $s''_2$ (but not infinite times, because $B$ is livelock-free). Therefore a feasible sequence of internal events (which can be empty) exist from

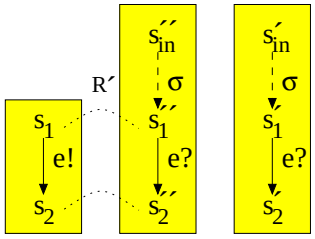



**Fig. 30.** Conditions 2(a) from the proof of Theorem 6.

$s_1''$ reaching a state $s''$ such that $\{e|\mathsf{En}(s'',e)\} \subseteq \Sigma_I^C$, with $s_1 R' s''$ and $s'' \approx s_1'$. Condition 4(b) of $R'$ ensures that $\{e|\mathsf{En}(s_1,e)\} \not\subseteq \Sigma_I^A$. □

*Proof.* (of Theorem 6) (*Case* $\{e_1,e_2\} = \{o_1,o_2\} \subseteq \Sigma_O^B$. *The other cases are similar*).
Let $R'$ be the relation between $A$ and $B$. Define $R$ as:

$$\forall s \in S^A, s'' \in S^B, s' \in S^C : sR's'' \wedge s_{in}^B \xrightarrow{\sigma} s'' \wedge s_{in}^C \xrightarrow{\sigma} s' \Leftrightarrow sRs'$$

- **Condition 1:** taking $\sigma = \lambda$ implies $s_{in}^A R s_{in}^C$.
- **Condition 2(a):** let $s_1 R s_1'$, and suppose $s_1 \xrightarrow{e} s_2$ with $e \in \Sigma_O^A$. Figure 30 depicts the situation. Condition 2(a) of $R'$ ensures that there exists $s_2'' \in S^B$ s.t. $s_1'' \xrightarrow{e} s_2''$ and $s_2 R' s_2''$. By definition of $R$, $\sigma$ is enabled both in $s_{in}^B$ and $s_{in}^C$. Then, each place marked by the sequence $\sigma$ in $B$ is also marked in $C$, because the flow relation of $B$ is included in the flow relation of $C$. Given that the initial marking of $B$ is preserved in $C$ and the set of predecessor places for each input event is also preserved, implies that $e$ is also enabled in $s_1'$. The definition of $R$ makes each $s_2'$ s.t. $s_1' \xrightarrow{e} s_2'$ to be related with $s_2$.
- **Condition 2(b):** let $s_1 R s_1'$, and suppose $s_1' \xrightarrow{e} s_2'$ with $e \in \Sigma_O^C$. The set of predecessor places of $e$ in $B$ is a subset or is equal to the one in $C$. Moreover, given that both the initial marking of $B$ is identical to the one in $C$, and each place marked by the sequence $\sigma$ in $B$ is also marked in $C$, implies that $e$ is also enabled in $s_1''$, i.e. $s_1'' \xrightarrow{e} s_2''$. Condition 2(b) of $R'$ ensures that $\mathsf{En}(s_1,e)$, and each $s_2$ such that $s_1 \xrightarrow{e} s_2$ is related by $R'$ with $s_2''$. The definition of $R$ induces that each such $s_2$ is related with $s_2'$.
- **Condition 3(a):** let $s_1 R s_1'$, and suppose $s_1 \xrightarrow{e} s_2$ with $e \in \Sigma_{INT}^A$, then a similar reasoning of Condition 2(a) can be applied.
- **Condition 3(b):** let $s_1 R s_1'$, and suppose $s_1' \xrightarrow{e} s_2'$ with $e \in \Sigma_{INT}^C$, then a similar reasoning of Condition 2(b) can be applied.

- **Condition 4(a):** let $s_1 R s_1'$, and suppose $\{e|\mathsf{En}(s_1, e)\} \subseteq \Sigma_I^A$. Condition 4(b) of $R'$ ensures that $\{e|\mathsf{En}(s_1'', e)\} \not\subseteq \Sigma_I^B$. If the non-input event enabled in $s_1''$ is different from $o_2$, then similar reasoning of previous cases guarantees that the event is also enabled in $s_1'$. If the event enabled in $s_1''$ is $o_2$ and no non-input event is enabled in $s_1'$, we will proof that $o_2$ is also enabled in $s_1'$. Assume the contrary: $o_2$ is enabled in $s_1''$ but no non-input event is enabled in $s_1'$. Applying the same reasoning of case 2(a) we can conclude that the place $p$ such that $\{p\} = {}^\bullet o_2$ in $B$ has a token in the marking $M$ corresponding to state $s_1'$. Moreover, the liveness of $C$ ensures that from $M$ there is a feasible sequence $\delta$ (let $\delta$ be minimal) reaching a marking $M'$ where $o_1$ is enabled. The minimality of $\delta$, together with the fact that the new place $p'$ added by $\phi_r$ between $o_1$ and $o_2$ is unmarked in $M$ (otherwise $o_2$ is enabled in $s_1'$, because $\{p, p'\} = {}^\bullet o_2$ in $C$) imply that $o_2 \notin \delta$, and therefore $M'(p) = 2$, which contradicts the safeness of $C$.
- **Condition 4(b):** let $s_1 R s_1'$, and suppose $\{e|\mathsf{En}(s_1', e)\} \subseteq \Sigma_I^C$, then similar reasons of the previous cases ensure that $\{e|\mathsf{En}(s_1'', e)\} \subseteq \Sigma_I^B$ and Condition 4(b) of $R'$ ensures that $\{e|\mathsf{En}(s_1, e)\} \not\subseteq \Sigma_O^A$.

Finally, it can be proven that the language of $\mathsf{RTS}(C)$ is a subset of the language of $\mathsf{RTS}(B)$. Therefore, no infinite trace of internal events can exist in $C$ implying that $C$ is livelock-free. □

*Proof.* (of Theorem 7) Conditions 1-4 of transformation $\phi_i$ ensure to preserve both the observational equivalence and the input-properness of $B$. Theorem 5 induces $A \rightleftharpoons C$. □

*Proof.* (of Theorem 8) If the observational languages of two deterministic systems coincide, then they are observational equivalent [16]. It can be proven that the observational language of $C$ is the same to the one of $B$. Moreover, $C$ is also input proper and therefore, applying the determinism of $B$ and Theorem 5 implies $A \rightleftharpoons C$. □