Analysing Signal-Net Systems

Peter H. Starke, Stephan Roch
Humboldt-Universität zu Berlin
Institut für Informatik
Unter den Linden 6, D-10099 Berlin
{starke,roch}@informatik.hu-berlin.de

September 2002

This report reflects the state of the art in the analysis of signal-net systems. It is intended to serve as a reference manual for all those interested in this new type of nets. Although the main problems are undecidable, analysis is possible.

Authors

Peter H. Starke, Stephan Roch
Humboldt-Universität zu Berlin
Institut für Informatik
Unter den Linden 6, D-10099 Berlin
{starke,roch}@informatik.hu-berlin.de

Karsten Schmidt, Adrianna Alexander, Paul Berthold
Humboldt-Universität zu Berlin
Institut für Informatik
Unter den Linden 6, D-10099 Berlin
{kschmidt,foremnia,berthold}@informatik.hu-berlin.de

This work was supported by the Deutsche Forschungsgemeinschaft under the Reference STA 450/4-2 and published as Informatik–Bericht 162, Humboldt-Universität zu Berlin, Institut für Informatik, September 2002.

Introduction

Modular modeling of systems is based on a concept of interaction for modules. If the modules are described by (classical) Petri nets, the only concepts for interaction available in net theory so far are token reading (by test arcs) and token passing (from module to module).

Hanisch and Rausch [Rau96, RH95] introduced a concept of one-sided synchronization of modules, where a signal-event from one module forces a simultaneous action in a second module, but only if that action is enabled. In this report, we resume the development of this approach on the Petri net level, i.e. recall the corresponding type of net, which we call signal-net system.

Extending Petri nets by incoming and outgoing signals is by no means new. Some classical concepts from Petri net application to discrete event controller design use such signal extensions, as, for example, the concept of König and Quäck [KQ88], and Graphcet [DA92]. These extensions, however, do not provide means for interconnecting several separate Petri nets with incoming and outgoing signals to a new model which has the same characteristics. The idea of condition/event systems provided by Sreenivas and Krogh [SK91] guided Hanisch and Rausch to a model based on Petri net representation of the dynamic behavior of basic modules of the system which has to be modeled and some extensions by incoming and outgoing signals which are used to connect the basic modules to a complete system model. Since the basic model form is derived from Petri nets and the signal concept is based on condition signals and event signals, they call their models net condition/event systems (NCES). In the case of autonomous systems (these are systems with no external inputs), analysis is possible. We call such autonomous systems signal-net systems. Different names for different variants have been used in the past [Sta97, SH97].

This report is structured as follows: In the first part we introduce the basic concepts of signal-net systems, show their syntax and dynamics and extend them with timing constraints on arcs and with colours.

In the second part we investigate dynamic properties. First, and unfortunately, the computational power of signal-net systems turns out to be that of a Turing-machine, so most important problems like reachability are undecidable. But, at least if the net is bounded, we can compute a reachability graph. We have investigated an on-the-fly test for boundedness, and different state space reduction techniques. We have adopted the stubborn set method and are able to use symmetries. In combination with the newly defined diamond reduction, efficient state space reduction of signal-net systems is possible.

The third part is about model checking. We recall the definition of CTL, a branching time temporal logic, and extend this logic with transition formulae and timing constraints. This part is merely a reference manual for the use of a model checker. The complete syntax can be found in the appendix.

The fourth part presents methods which are based on structural properties, namely deadlocks and traps and investigates free choice properties and compositions. The last part is about state, place, transition and step invariants.

We close this report with additional material in the appendix, e.g. bibliography, index, and SESA documentation. SESA is, like INA [RS98], a net analyzing tool without graphical interface [SR00]. We have implemented some of the algorithms presented in this report. The main focus lies on efficient (reduced) state space analysis and model checking. Please visit http://www.informatik.hu-berlin.de/lehrstuehle/automaten/sesa/ for more information.

If you are interested in modeling aspects and practical applications, then we suggest to have a closer look on publications written by our partners from the engineering department of Martin-Luther-Universität Halle-Merseburg [LH00,HPP+99,HKL99,Kar99, PPH98,Pan98,Sch97,VHSR00,VH99]. They use our tool SESA for the analysis of control devices and the verification of some aspects of the execution control of function blocks following the draft standard IEC 1499 which is currently under preparation.

Berlin, September 2002

Peter H. Starke and Stephan Roch

Contents

I. Preliminaries

1. Basic Definitions

Signal-net systems are generalizations of Petri nets in that they allow a one-sided synchronization of transitions by means of signals. In the sequel, we first give the mathematics of the basic model, avoiding the complications induced by colours and time constraints.

Let P be an arbitrary non-empty set. A mapping $m : P \to \mathbb{N}_0$ is called a *marking of* P (or a *multiset over* P, $BAG(P)$ is the set of all multisets over P). For $p \in P$, the number $m(p) \in \mathbb{N}_0$ often is referred to as the *number of tokens* on p or as the *multiplicity of* p *in* m.

For markings m and m' (of the same set) we define the *sum* $m + m'$, the *difference* $m - m'$ and the relation $m \leq m'$ pointwise. Moreover, reminding that markings are multisets, we define the *union* $m \cup m'$ and the *intersection* $m \cap m'$ by

$$\begin{aligned} m \cup m'(p) &:= \max(m(p), m'(p)), \\ m \cap m'(p) &:= \min(m(p), m'(p)). \end{aligned}$$

An (integer valued) mapping $v : A \to \mathbb{Z}$ is called an A-*vector*. The operations $+$, $-$ and the relation $\leq$ for A-vectors are defined pointwise.

The set of all *finite sequences* (or *words*) over an alphabet A is denoted by A^*. For a relation $R \subseteq X \times Y$ we define complement $\overline{R}$, inverse R^{-1}, and reflexive transitive closure R^* as usual.

1.1. Structure

$N = [P, T, F, V, B, W, S, M, m_0]$ is a *signal-net system* (*SNS* for short) iff:

1. P is a non-empty finite set (of places),
2. T is a non-empty finite set (of transitions), disjoint with P,
3. F is a subset of $(P \times T) \cup (T \times P)$ (the flow relation, the set of flow arcs),
4. V is a mapping which attaches a positive integer to every arc (the arc weight, $V : F \to \mathbb{N}$),
5. B is a subset of $P \times T$ (the set of condition arcs),
6. W is a mapping which attaches a positive integer to every condition arc (the condition arc weight, $W : B \to \mathbb{N}$),
7. S is a subset of $(T \times T) \setminus \mathrm{id}_T$, the irreflexive signal (flow) relation,
8. M is a mapping which attaches a (signal-processing) mode to every transition ($M : T \to \{\boxed{\wedge}, \boxed{\vee}\}$), and, finally,
9. m_0 is a marking of P called the initial marking or the initial state of N.

The sets P, T and F, and the mappings V and m_0 are interpreted in the usual way. Nevertheless, in general the tuplet $[P, T, F, V, m_0]$ is not a Petri net in the classical sense (i.e. $[P, T, F]$ is not a net) because we allow places and transitions to be isolated with respect to token-flow. With other words, we drop the condition

$$dom(F) \cup cod(F) = P \cup T,$$

which is assumed for Petri nets. This is necessary in our context since a place may serve as a condition and the firing of a transition can have an effect without changing the marking itself. Since some properties of signal-net systems can be concluded from properties of $[P, T, F, V, m_0]$ we shall call this tuplet the *underlying Petri net*, but, if we want to use a result from Petri net theory for the underlying Petri net we have to check whether it holds true if isolated places or transitions are present.

If $[p, t]$ is an element of B then we say that p is a (or serves as) *condition of* t, i.e., in order to fire t it is necessary that p is marked with at least $W(p, t)$ tokens. Figure 1.1 shows the graphical representation of the condition arc $[p, t]$. We consider condition

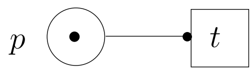


Figure 1.1: Graphical representation of the condition arc $[p, t]$

arcs $[p, t]$ as leading a piecewise constant signal which informs about the token load of the place p, i.e. the marking of p. From the firing rule (see below), one can see that a *condition arc with multiplicity* $W(p, t)$ between t and p has, in general, not the same effect as two flow arcs $[p, t]$ and $[t, p]$ of the same multiplicity.

If a pair $[t, t']$ of transitions is an element of the signal relation S, then we say that a *signal arc* leads from t to t', which means that firing the transition t sends a *signal-event* to the transition t'. We assume the signal relation to be irreflexive since it is not meaningful for a transition to send to itself a signal-event. Figure 1.2 shows the graphical representation of a signal arc $[t, t']$. Signal-events reflect the second type of

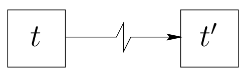


Figure 1.2: Graphical representation of the signal arc $[t, t']$

signals needed to connect the modules of a control device, the impulse type. They are described by time functions which have non-zero values only for isolated time points.

For any transition t the mode $M(t)$ determines the processing of the incoming signal-events. Consider a transition t which is the target of signal arcs coming from transitions $t_1, \ldots, t_n$, i.e. $[t_1, t], \ldots, [t_n, t] \in S$. If $M(t) = \boxed{\vee}$ then to fire t it is necessary that at least one signal arc $[t_i, t]$ leads a signal-event, i.e. t_i is just firing. If, otherwise, $M(t) = \boxed{\wedge}$ then to fire t it is necessary that all signal arcs leading to t lead a signal-event.

If a transition t has no incoming signal arcs, i.e., the set

$$St := \{t' \mid [t', t] \in S\}$$

is empty, then the transition t is called *spontaneous*, otherwise *forced*. By *Spont* we denote the set of all spontaneous transitions of N, by *Forc* the set of all forced transitions.

For any transition t we define the markings t^-, t^+, $\widehat{t}$ as follows:

$$t^-(p) \quad := \quad \begin{cases} V(p,t), & \text{if } [p,t] \in F \\ 0, & \text{else} \end{cases} \quad ,$$

$$t^+(p) \quad := \quad \begin{cases} V(t,p), & \text{if } [t,p] \in F \\ 0, & \text{else.} \end{cases}$$

and

$$\widehat{t}(p) \quad := \quad \begin{cases} W(p,t), & \text{if } [p,t] \in B \\ 0, & \text{else.} \end{cases}$$

For any subset $s \subseteq T$ the markings s^- resp. s^+ are the sum of the markings t^- resp. t^+ for $t \in s$, and, $\widehat{s}$ is the union of the markings $\widehat{t}$ for $t \in s$.

1.2. Dynamics

SNS are executed in steps, i.e. sets of transitions are fired simultaneously. The *firing rule* says, roughly speaking, that executable steps are formed by first picking up a nonempty set of enabled spontaneous transitions and then adding as many as possible of those transitions that are forced to fire by signal-events produced by transitions in the step. This implies that in every non-dead *SNS* there exists a spontaneous transition. To make this more precise we define the *signal-completeness* of transition sets inductively:

Basis: Every subset $s \subseteq \mathit{Spont}$ is *signal-complete*.

Step: If $s \subseteq T$ is *signal-complete*, $t \in \mathit{Forc}$ and

$$M(t) = \boxed{\vee} \text{ and } St \cap s \neq \emptyset$$

or

$$M(t) = \boxed{\wedge} \text{ and } St \subseteq s$$

then $s \cup \{t\}$ is *signal-complete*.

Obviously, the empty set is signal-complete and $\emptyset$ is the only signal-complete set containing no spontaneous transition. A signal-complete set of transitions may fire simultaneously as far as signal-events are concerned.

A transition $t \in \mathit{Forc}$ is said to be *forced by the set* s iff $t \notin s$ and $s \cup \{t\}$ is signal-complete.

A subset $s \subseteq T$ is said to be a *step of* N iff

1. $s \cap \mathit{Spont} \neq \emptyset$

   (s is signal-founded, i.e. there is at least one spontaneous transition in s), and

2. s is signal-complete

   (i.e. all necessary signal-events will occur).

A step s of N is called *enabled at the marking* m iff

3. $s^- \leq m$

   (s has token-concession, i.e. the transitions in s are concurrently enabled w.r.t. tokens), and

4. $\widehat{s} \leq m$

   (i.e. the conditions of all $t \in s$ are satisfied).

A step s of N is said to be *executable at the marking* m iff s is enabled at m and

5. there is no forced transition $t \in \mathit{Forc}$ such that $s \cup \{t\}$ also satisfies 1-4

   (s is signal-closed, i.e. maximal with respect to inclusion of forced transitions.)

A forced transition t with $M(t) = \boxed{\wedge}$ appears in an enabled step only if it receives signals from all its signal sources. Otherwise, a forced transition t with $M(t) = \boxed{\vee}$ appears in an enabled step if it receives a signal from at least one of its signal sources.

If s is an executable step at m, then s may fire, which leads to a new state of N, i.e. the marking $m' := m - s^- + s^+$. This is abbreviated as $m \xrightarrow{s} m'$. The reachability relation is defined as usual; let $R_N(m)$ denote the set of all markings m' such that a finite sequence of executable steps leads from m to m'. The *state* or *reachability graph* is a structure $[R_N(m_0), E]$ where E is the set of edges such that $(m, m') \in E$ iff $m, m' \in R_N(m_0)$ and there is a step s with $m \xrightarrow{s} m'$.

1.3. Step Computation and Options

The computation of the *list* of all executable steps at a given state is implemented in SESA as follows:

1. Compute the set En of all enabled spontaneous transitions.

2. Compute the list Sub of all nonempty subsets s of En which are concurrently enabled at the given state.

3. For every element s of the list Sub compute the list of all executable steps s' such that $s = \mathit{Spont} \cap s'$ (i.e. include into s in all possible ways and as much as possible enabled forced transitions).

4. Form the union of the lists computed in step 3.

In this way the list of executable steps (the *steplist*) is computed under the (default) options:
`- firing rule: N (arbitrary maximal steps)`
`- synchros : N (not to be used)`
`- greediness : N (not to be used)`
`- priorities : N (not to be used).`
Any different setting of the options leads to the exclusion of some steps from the default list. We are going to discuss the details now.
If the firing rule is set to
`- firing rule: S (maximal single spontaneous transition steps)`
then all steps will disappear from the steplist which contain more than one spontaneous transition. In this case, computation step 2 will compute the list of all singletons of elements of En.
If the firing should be guided by *synchronisation sets*, i.e. the option
`- synchros: Y (to be used)`
is set, then we will be asked for a nonempty *synchro*-list of pairwise disjoint sets of spontaneous transitions such that different sets have no preplaces in common (no static conflicts). The transitions in the same synchronisation set should fire simultaneously as much as possible. Therefore, a step s is deleted from the steplist, if there exist a synchronisation set Q in the synchro-list such that $s \cap Q$ is not empty and a step s' in the steplist such that $s \cap Q$ is a proper subset of $s' \cap Q$. In this case, the step s' contains more transitions to synchronise. If the synchro-list contains the set *Spont* as its only element then only those steps remain which contain a (w.r.t. set inclusion) maximal set of spontaneous transitions. The synchro option can be set only under the normal `firing rule: N (arbitrary maximal steps)`.
If the firing should favour *greedy* transitions, i.e. the option
`- greediness: Y (to be used)`
is set, then some spontaneous transitions (by means of the editor) have to be designated as *greedy*. If at the current state greedy transitions are enabled, then only steps containing at least one greedy (spontaneous) transition are executed, i.e. the other steps are deleted from the steplist. The greediness option can not be set under the `firing rule: S (maximal single spontaneous transition steps)`.
If the firing should follow *priorities*, i.e. the option
`- priorities: Y (to be used)`
is set, then to every (spontaneous) transition (by means of the editor) a natural number (its priority) must be attached (the default value is zero). Priorities of forced transitions will be ignored. Under the priority option from the set En of all enabled spontaneous transitions all transitions are removed which do not have the greatest occurring priority. Hence, during the computation of executable steps only the step 1 is changed: the set En contains only enabled spontaneous transitions of the highest occurring priority. If, e.g. the transition t_1 with priority 1, t_2 and t_3 with priority 2 are enabled at the given state then only t_2 and t_3 will be in En. Notice, that not all enabled spontaneous transitions of highest priority (if there are two or more) are forced to fire in the same (executable) step but that it is impossible to fire a spontaneous transition of lower priority.

2. Time Constraints on Arcs

In this section we consider *SNS* under time constraints applied to the input arcs of transitions [Han93]: to every pre-arc $[p, t] \in F$ we attach an interval $[eft(p, t), lft(p, t)]$ of natural numbers with $0 \leq eft(p, t) \leq lft(p, t) \leq \omega$.

The interpretation is as follows. Every place p bears a clock which is running iff the place is marked and switched off otherwise. All running clocks run at the same speed measuring the time the token status of its place has not been changed, i.e. the clock on a marked place p shows the age of the youngest token on p. If a firing transition t removes a token from the place p or adds a token to p then the clock of p is turned back to 0. A transition t is able to remove tokens from its pre-places (i.e. to fire) only if for any pre-place p of t the clock at place p shows a time $u(p)$ such that $eft(p, t) \leq u(p) \leq lft(p, t)$. Hence, the firing of transitions is restricted by the clock positions.

Definition 2.1
Let $N = [P, T, F, V, B, W, S, M]$ be an *SNS*, eft a mapping from $F \cap (P \times T)$ to $\mathbb{N}_0$ and, lft a mapping from $F \cap (P \times T)$ to $\mathbb{N}_0 \cup \{\omega\}$ such that always $eft(p, t) \leq lft(p, t)$ holds. Then $TN = [N, eft, lft]$ is an *arc-timed signal-net system.*

A *state* of TN is given by a pair $[m, u]$ where m is a marking of P, and u is the P-vector of the clock positions. We assume that a clock which is switched off shows the time 0, and, that the time-scale used is integer. Therefore u is a marking too, and for any (realizable) state it holds: If $u(p) > 0$ then $m(p) > 0$.

The initial state $[m_0, u_0]$ of TN in general (but not necessarily) consists of the initial marking of N and the zero time vector.

Arc-timed signal-net systems are executed in steps too. The execution of a step does not take time. Let $[m, u]$ be a state. A step s of N is said to be *enabled at the state* $[m, u]$ of TN (compare this to section 1.2) iff

3. $s^- \leq m$ and for every pre-place p of a transition $t \in s$ it holds $eft(p, t) \leq u(p) \leq lft(p, t)$

   (i.e. s has token-concession and the clocks are between eft and lft), and

4. $\widehat{s} \leq m$

Obviously, a step s may be enabled at the marking m in N, but not enabled at the state $[m, u]$ of TN because some clocks have not reached the *earliest firing time eft* or have passed already the *latest firing time lft*.

The state $[m, u]$ of an arc-timed signal-net system may change not only by execution of a step but also by elapsing of one time unit to $[m, u']$ where

$$u'(p) \quad := \quad \begin{cases} u(p) + 1, & \text{if } m(p) > 0, \\ 0, & \text{else.} \end{cases}$$

If a state $[m, u]$ of TN is such that no step is enabled or can become enabled by elapsing of time then this state is called *dead*. Otherwise, the minimal number of time units after which at least one step becomes enabled is called the *delay* $D(m,u)$ of the state $[m,u]$. Hence, the delay is defined only for non-dead states.

Since every executable step has to contain a spontaneous transition the delay of a non-dead state is the minimal number of time units after which at least one spontaneous transition becomes enabled. This number obviously may be zero.

Let $[m,u]$ be a non-dead state. Following the *weak earliest firing rule* we call a step s to be *executable at the state* $[m,u]$ iff s is enabled after elapsing of $D(m,u)$ time units.

Given a non-dead state $[m,u]$ we first compute the delay $D(m,u)$ and elapse $D(m,u)$ time units resulting in the state $[m,u']$. Next the set E of all spontaneous transitions enabled at $[m,u']$ is computed. Then we proceed with E like the normal firing rule does, resulting in a list of executable steps. These steps are considered as executable at the original state $[m,u]$ (they all have the delay $D(m,u)$).

The execution of an executable step s at the state $[m,u]$ then is done by first elapsing $D(m,u)$ time units and then firing s. The state $[m',u']$ reached by the execution of s is determined by

$m' = m - s^- + s^+$,

$$u'(p) \quad := \quad \begin{cases} u(p) + D(m,u), & \text{if } m(p) > 0 \wedge m'(p) > 0 \wedge p \notin (Fs \cup sF), \\ 0, & \text{else.} \end{cases}$$

During the computation of the list of executable steps synchronisation sets and/or greediness may be applied. If we put the set of all spontaneous transitions as the only synchronisation set we arrive at the (strict) *earliest firing rule* where a step s is *executable at the state* $[m,u]$ iff s is enabled after elapsing of $D(m,u)$ time units and

5. s is not contained properly in a step s' which is enabled after elapsing of $D(m,u)$ time units.

Remark. The earliest firing rule is often used in Petri nets under time constraints. It imposes *force to fire* to the system dynamics: an enabled transition which is not in conflict with another enabled transition must fire at once. In our setting steps s_1, s_2 which are simultaneously fireable in the sense of Section 7 may be executable. After execution of s_1 we arrive at a state where the delay is zero und s_2 is executable, i.e. in some sense this state is transient. If one is not interested in such states one should switch to the (strict) earliest firing rule by setting the synchro option as described above.

Definition 2.2

For any place p we define the *clock stop position of* p as

$$csp(p) \quad := \quad \begin{cases} 1 + max\{lft(p,t) \mid t \in pF \wedge lft(p,t) \neq \omega\}, & \text{if this set is not empty,} \\ max\{eft(p,t) \mid t \in pF\}, & \text{else.} \end{cases}$$

Consider two (reachable) states $[m,u], [m,u']$ which differ only in the clock positions u, u' in the following way: If $u(p) \neq u'(p)$ then $u(p)$, $u'(p) \geq csp(p)$. Then both states are indistinguishable in the sense that the same sequences of steps can be fired. Therefore, in our implementation, we stop every clock at their clock stop time, i.e. the clock position will not be increased by elapsing a time unit, although the clock is "running". In this way states of the above described kind will be identified.

3. Colours

As signal-net systems are Petri nets with additives, coloured signal-net systems will turn out to be coloured Petri nets [Jen92, Jen94] with additives. We therefore recall the definition of coloured Petri nets:

Definition 3.1
$CPN = [P, T, F, C, V, m_0]$ is a *coloured Petri net* iff

1. P is a non-empty finite set (of places),
2. T is a non-empty finite set (of transitions), disjoint with P,
3. F is a subset of $(P \times T) \cup (T \times P)$ (the flow relation, the set of arcs),
4. C is a mapping which attaches a non-empty finite set $C(x)$ of colours to every node $x \in P \cup T$,
5. V is a mapping defined on the set F of all arcs such that, for $f = [p, t] \in F$(resp. $f = [t, p] \in F$), the value $V(f)$ is a mapping from $C(t)$ into $BAG(C(p))$,
6. m_0 is the initial marking of CPN, i.e. a mapping which attaches a multiset $m_0(p)$ from $BAG(C(p))$ with every $p \in P$.

Let $p \in P$, $c \in C(p)$, $t \in T$, $d \in C(t)$. Then $m_0(p)[c]$ is the number of tokens of colour c that the place p holds initially and $V(p,t)[d][c]$ is the number of tokens of colour c that a firing of the transition t under colour d will take from the place p.
Let

$$\begin{array}{rcl} Pf & := & \{[p, c] \mid p \in P \wedge c \in C(p)\}, \\ Tf & := & \{[t, d] \mid t \in T \wedge d \in C(t)\}. \end{array}$$

Definition 3.2
$CN = [P, T, F, C, V, B, W, S, Z, M, m_0]$ is a *coloured signal-net system* ($CSNS$) iff

1. $[P, T, F, C, V, m_0]$ is a coloured Petri net,
2. B is a subset of $T \times P$,
3. W is a mapping defined on the set B of all condition arcs such that, for $b = [t, p] \in B$, the value $W(b)$ is a mapping from $C(t)$ into $BAG(C(p))$,
4. $S \subseteq T \times T$,
5. Z is a mapping defined on the set S of all signal arcs such that, for $s = [t, t'] \in S$, the value $Z(s)$ is a mapping from $C(t)$ into $2^{C(t')}$,
6. M is a mapping which attaches a mode to every colour of a transition ($M : Tf \to \{\boxed{\wedge}, \boxed{\vee}\}$).

Let $b = [t,p] \in B$ and $i = W(b)[d,c] > 0$. Then, to fire the transition t under colour d, it is necessary that the place p holds at least i tokens of colour c. If the pair $s = [t,t']$ is in S, then the firing of transition t under its colour c forces the transition t' to fire under colour $c' \in Z(s)[c]$.

Definition 3.3
Let CN be a $CSNS$. The semantics of CN is given by an SNS N called the *unfolding of* CN, denoted by $Unf(CN)$:

$$Unf(CN) \quad := \quad [Pf, Tf, Ff, Vf, B, W, S, M, mf_0],$$

where Pf, Tf, B, W, S and M are as above,

$$\begin{aligned} Ff &:= \{\, [[p,c],[t,d]] \mid V(p,t)(d)[c] > 0 \,\} \\ &\quad \cup \{\, [[t,d],[p,c]] \mid V(t,p)(d)[c] > 0 \,\}, \\ Vf([p,c],[t,d]) &:= V(p,t)(d)[c], \\ Vf([t,d],[p,c]) &:= V(t,p)(d)[c], \\ mf_0([p,c]) &:= m_0(p)[c]. \end{aligned}$$

The dynamic behaviour of CN is defined to be the dynamic behaviour of $Unf(CN)$. This means e.g. that a set $s \subseteq Tf$ of transition colours of CN is an executable step at the marking m iff s is an executable step of $Unf(CN)$ at the corresponding marking mf.

II. Dynamic Properties

4. Analysis

Analysis of a model is understood as the derivation of assertions on the behaviour of the model by means of algorithms. There has been considerable progress in this field for Petri nets of different types, and a variety of tools is available. Therefore, the first question should be whether and to what extent these tools can be used for our purposes. Unfortunately, the answer is negative:

Theorem 4.1
Any Turing-machine can be simulated by an SNS.

The *proof* uses the well-known fact that Petri nets — under the firing rule requiring that only maximal sets of concurrently enabled transitions be executed — can simulate counter machines (which are Turing-equivalent). Take any Petri net N, introduce a new transition t to N, and a signal arc from t to any old transition of N. The resulting *SNS* contains exactly one spontaneous transition, namely t. Any executable step consists of t and a maximal set of concurrently enabled old transitions. Hence, *SNS* can simulate Petri nets under the maximal firing rule, which, in their turn, can simulate counter machines.

As a consequence of Theorem 4.1, we have that all nontrivial problems for *SNS*, e.g. boundedness, liveness, are undecidable: these problems can be reduced to the halting problem for counter machines. This has the consequence that it is not possible to simulate *SNS* by Petri nets (under the single transition firing rule), which restricts the useability of Petri net tools.

Our proof shows that already *SNS* without condition arcs are Turing- equivalent. This raises the question whether *SNS* can be simulated by *SNS* without condition arcs by a local construction. The answer is affirmative:

Theorem 4.2
Any SNS can be simulated by an SNS without condition arcs.

Proof. Without loss of generality, we may confine ourselfs to *SNS* where every place p serves as a condition for at most one transition. If this is not the case, say, p has condition arcs to $n \geq 2$ transitions $t_1, \ldots, t_n$, then we replace p by n parallel places $p_1, \ldots, p_n$, which, initially, each hold the same number of tokens as p. Since they are parallel, this property is preserved during the execution of the net. Then we draw (for $i = 1, \ldots, n$) a condition arc from p_i to t_i with the multiplicity of the original condition arc from p to t_i.

Now, consider an *SNS* where every place p serves as a condition for at most one transition. The idea of the simulation is to replace the execution of a step in the original system by the execution of three steps in the new system which will be triggered by two additional spontaneous transitions *start* and *resume*.

In the first substep for any place p, which serves as condition for the transition t, a new transition $t_{p,1}$ checks whether p is marked with at least as many tokens as the multiplicity $W(p, t)$ of the condition arc from p to t is, and, in that case, puts one token to an auxiliary place p_h, which is initially clean. The place p_h loops around t (with multiplicity 1). Figure 4.1 shows the replacement of a single condition arc and the interconnection of newly introduced net elements.

The second substep simulates a step of the original system and, in the third substep, the new place p_h will be cleaned by the new transition $t_{p,2}$.

Let n be the number of spontaneous transitions in the original system. The sequence of the substeps is forced by three new places a, b, c and two new spontaneous transitions *start* and *resume*. Place a is initially marked with one token, b and c are clean. The transition *start* takes the token from a and sends n tokens to place b whilst sending signal-events to (i.e. forcing) all the new transitions $t_{p,1}$. Place b has a flow arc with multiplicity 1 to every spontaneous original transition and to $n-1$ *cleaning* transitions $c_1, \dots, c_{n-1}$. These transitions are connected by signal arcs to form a chain: $[c_1, c_2] \in S$, $\dots$, $[c_{n-2}, c_{n-1}] \in S$. Every original spontaneous transition sends a signal-event to c_1 which processes signal-events disjunctively. Figure 4.2 shows the interconnection of the additional net elements we have introduced.

Now, after the first substep, any executable step s of the original system containing $k \geq 1$ spontaneous transitions is simulated by the executable step $s' = s \cup \{c_1, \dots, c_{n-k}\}$.

Every original spontaneous transition and every cleaning transition c_i obtains a flow arc of multiplicity 1 to the place c, so that after the second substep we have n tokens on place c. Place c has a flow arc of multiplicity n to the transition *resume*. The transition *resume* then puts one run token back to place a, sending signal-events to all new transitions $t_{p,2}$. thus cleaning the new places p_h.

$\square$

The last theorem is only of theoretical interest showing that the signal arcs are the only essential new ingredients of *SNS*. Maybe this knowledge can be used in proving some properties for *SNS* by proving that these properties hold for *SNS* without condition arcs.

Finally we investigate the relation between the set of reachable markings of an *SNS* $N = [P, T, F, V, B, W, S, M, m_0]$ and its underlying Petri net $PN = [P, T, F, V, m_0]$. Since any transition t from a step s executable at m in N is enabled at m in PN, we have (under any setting of the options):

Proposition 4.3
For any marking m, $R_N(m) \subseteq R_{PN}(m)$.

Hence, every marking reachable in N is reachable in the underlying Petri net PN as well. Sometimes we will use this fact for the analysis of *SNS*.

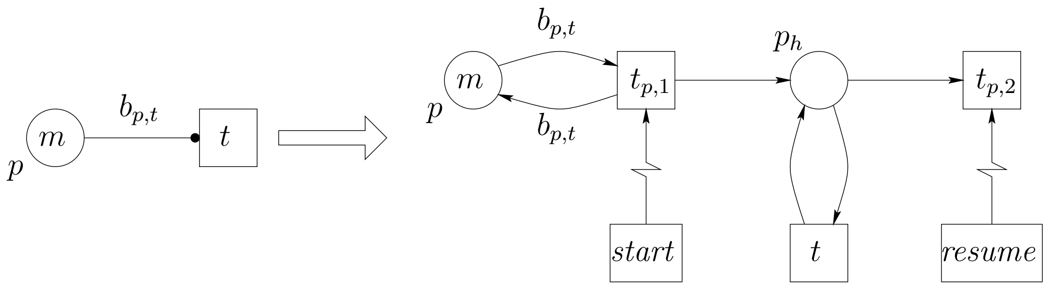


Figure 4.1: Local replacement of a condition arc

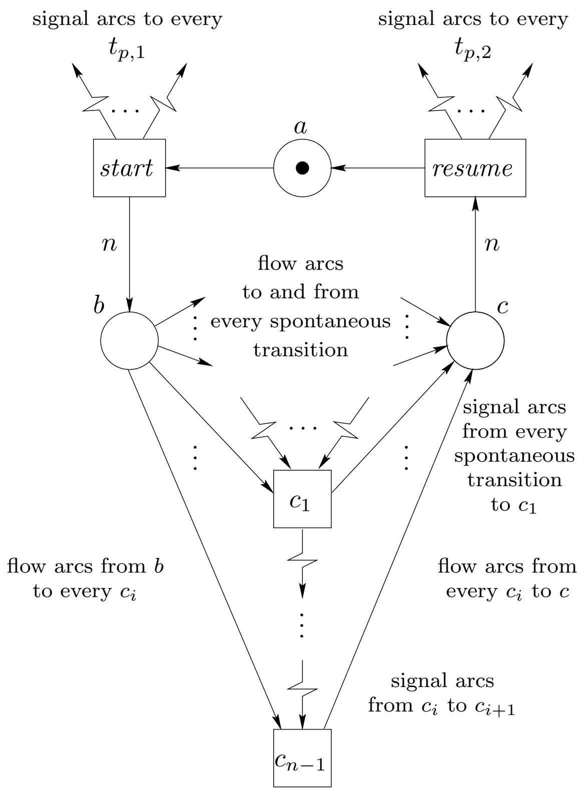


Figure 4.2: Additional places a, b and c and transitions *start*, *resume* and $n-1$ cleaning transitions c_i for the simulation

5. Reachability Graphs

Once we know that an *SNS* is bounded, we can (at least in principle) decide all further questions by construction of the reachability graph. But the state explosion problem urges us to look for methods which, depending on the question at hand, avoid unnecessary computations, i.e. which compute only a subgraph of the reachability graph:

- restrict the depth of the computed graph (applicable in the unbounded case too),
- use a "bad" predicate: only states (markings) not satisfying the predicate will be developed further while states satisfying the predicate will be included as dead states into the computed graph,
- use a CTL-formula: compute only that part of the reachability graph needed to check the formula,
- reduce the number of arcs by avoiding simultaneous firing of steps (see section 7),
- use the stubborn set method to compute a reduced reachability graph (see section 8),
- use symmetries of the net (see section 9).

6. Boundedness

A net is said to be *bounded* (at its initial marking) iff it has only finitely many states; it is called *structurally bounded*, iff it is bounded at any initial marking. Boundedness of *SNS* is an undecidable property because the boundedness problem of *SNS* can be reduced to the halting problem of counter machines.

Therefore, we have to look for decidable sufficient conditions for the boundedness or unboundedness of *SNS*. Clearly, by Proposition 4.3, the boundedness of an *SNS* N is implied by the boundedness of the underlying Petri net *PN* (and, consequently, by all conditions that imply the boundedness of *PN*, e.g. structural boundedness of *PN*, existence of a covering place invariant).

Theorem 6.1
Let $N = [P, T, F, V, B, W, S, M, m_0]$ be a SNS and $m_0 \xrightarrow{*} m_1 \xrightarrow{s_1} m_2 \ldots \xrightarrow{s_k} m_{k+1}$ a firing sequence such that $m_1 \leq m_{k+1}$ and $m_1 \neq m_{k+1}$. Moreover, let

$$Q \quad := \quad \{p \mid m_1(p) < m_{k+1}(p)\}.$$

Then if each transition $t \in T$ which is a post-transition of a place from Q or has a condition in Q is spontaneous, N is unbounded.

Proof. We shall show that the sequence $s_1 \ldots s_k$ of steps can be executed at the marking m_{k+1} again. This would lead to a marking $m_{2k+1} \neq m_{k+1}$ such that $m_{2k+1} \geq m_{k+1}$ and $\{p \mid m_{k+1}(p) < m_{2k+1}(p)\} = Q$. Hence, the places in Q are (simultaneously) unbounded.

Suppose s_1 is not executable at m_{k+1}. Since s_1 is executable at $m_1 \leq m_{k+1}$, there exists a step s', executable at m_{k+1}, such that $s_1 \subset s'$, $\emptyset \neq s' - s_1 \subseteq \{t \mid St \neq \emptyset\}$. The step s' contains the same spontaneous transitions as s_1 but more transitions forced by spontaneous transitions. Consider a (forced) transition $t \in s' - s_1$. Since $t \notin s_1$, a condition p_0 from Bt is not fulfilled at m_1 (i.e. $m_1(p_0) < W(p_0, t)$), or a preplace p_1 of t does not contain enough tokens (i.e. $m_1(p_1) < V(p_1, t)$). Since $t \in s'$, we obtain in the first case $m_{k+1}(p_0) \geq W(p_0, t) > m_1(p_0)$, i.e. $p_0 \in Q$, and in the second case we obtain $m_{k+1}(p_1) \geq V(p_1, t) > m_1(p_1)$, i.e. $p_1 \in Q$. Hence, t has a condition in Q or a pre-place in Q, which implies that t is spontaneous, contradicting $t \in \{t \mid St \neq \emptyset\}$.
Thus, s_1 is executable at m_{k+1}. Let $m_{k+1} \xrightarrow{s_1} m_{k+2}$. Then

$$m_{k+2} \geq m_2, \; m_{k+2} \neq m_2 \text{ and } m_{k+2} - m_2 = m_{k+1} - m_1.$$

Therefore, $\{p \mid m_2(p) < m_{k+2}(p)\} = Q$. By induction it follows that s_2 is an executable step at $m_{k+2}(p), \ldots$ □

The converse of Theorem 6.1 is not true; there exist unbounded *SNS* which do not fulfill the conditions (see Figure 6.1).

The assumptions of Theorem 6.1 can be used during the generation of the reachability graph of N to rule out some unbounded *SNS* (where the reachability graph is infinite). Safeness and k-boundedness are obviously trivial problems in the above sense: they are decidable.

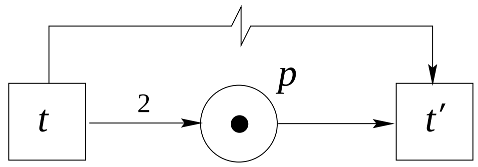


Figure 6.1: Counterexample to the converse of Theorem 6.1

If the underlying Petri net PN turns out to be unbounded there is still some hope that N may be bounded, but the only possibility to find out is to compute the reachability graph with on-the-fly checks accordings to Theorem 6.1. On the other hand, experience with unbounded SNS shows that unboundedness of N is in many cases shown much faster by this method than by deciding the unboundedness of the underlying Petri net PN. This is because PN has many more states than N.

7. Diamond Reduction

State space generation for signal-net systems is complicated, since sets of transitions have to be handled. This section describes techniques to reduce the number of steps needed for the reachability graph computation.

7.1. Simultaneous Execution

The normal (default) firing rule for *SNS* allows the simultaneous firing of steps: If two disjoint steps s_1, s_2 can be executed simultaneously in m. i.e. $s_1^- + s_2^- \leq m$, then the union $s_1 \cup s_2$ (or a proper superset) is also executable in m according to the normal firing rule.

In Petri nets we normally don't consider simultaneous firing of transitions in steps. We can see Petri nets in our setting as signal-net systems without condition- and signal-arcs, i.e. $B = S = \emptyset$; their firing rule is then equivalent to our firing rule `maximal single spontaneous transition steps`, since every transition of a Petri net is spontaneous in our sense. The generation of reachability graphs of Petri nets by firing only single transitions is justified by the fact that the set of all reachable markings is exactly the set, reached by pure interleaving.

If we consider the simultaneous firing of two or more steps in signal-net systems, we notice, that a signal-net system can reach markings, which are not reachable by pure interleaving of these steps.

In Fig. 7.1 and 7.2 we have for both signal-net systems that t_1 and t_2 are both executable in the initial marking (shown on the left). The same is true for the union $\{t_1, t_2\}$, but firing t_1 and t_2 simultaneously leads to markings, which are otherwise not reachable.

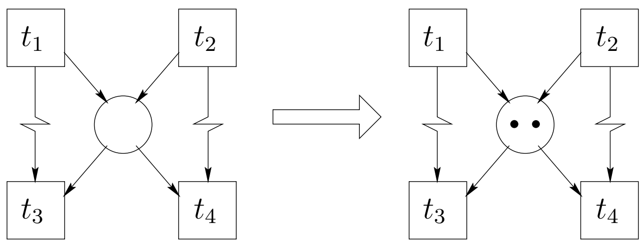


Figure 7.1: Simultaneous execution of t_1 and t_2 produces two tokens

This is a well known fact for so called contextual nets [MR95], e.g. Petri nets with read-arcs (we can see them as signal-net systems without signal-arcs; in our setting read-arcs can be noted as condition arcs), and for Petri nets with inhibitor-arcs [JK91b, JK95, MR95], if we allow the simultaneous firing of transitions [JK91a, JK93].

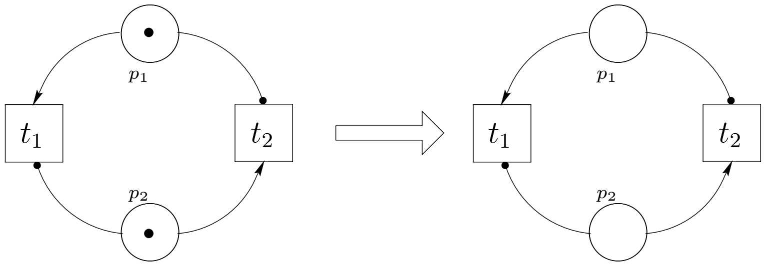


Figure 7.2: Simultaneous execution of t_1 and t_2 produces empty marking

7.2. Diamond Reduction

In many situations we have to execute a set of steps which have no influence on each other. According to the normal firing rule for signal-net systems we additionally have to take the union of all subsets of such a set into consideration. The reachability graph usually contains structures like in Fig. 7.3. This may lead to an exponential overhead, without generating any new marking, since normal interleaving of the steps would suffice. Our aim is to introduce a reduction technique which avoids such overhead.

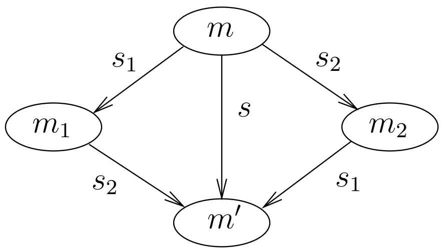


Figure 7.3: Diamond in the reachability graph for independent steps with $s = s_1 \cup s_2$

The next definition characterizes situations in which we can omit the firing of a step, which can be divided into two disjoint steps, without missing reachable markings.

Definition 7.1 (Diamond reduction)
A step s which is executable in m is *reducible* in m iff s is the disjoint union of two steps, i.e. $s = s_1 \cup s_2$ and $s_1 \cap s_2 = \emptyset$, and a sequence (interleaving) of s_1 and s_2 is executable in m, i.e. $m \xrightarrow{s_1\ s_2}$.

Proposition 7.2
If we construct a reduced reachability graph by considering only irreducible steps, then this diamond reduced graph has the same set of reachable markings as the full graph. Furthermore boundedness, liveness and resetability (reversability) and the truth value of CTL formulae (see section 11) build with the temporal operators AG and EF only are preserved by the reduction. If an A-CTL-formulae is true in the full graph, then it is

true in the reduced graph. If an E-CTL-formulae is true in the reduced graph, then it is true in the full graph.

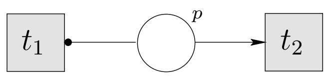
Proof. It is obvious that the diamond reduction preserves reachability of states (not minimal paths and distances) and boundedness. Preservation of liveness, resetability (reversability) and the truth value of formulae build with AG and EF only follows from the fact that the reduction preserves the strongly connected components (and the reachablity between them) of the full graph.

Since the reduced graph is simulated by the full graph, the truth of CTL-formulae build with A quantification only transfers from the full to the reduced and with E quantification the other way round only. □

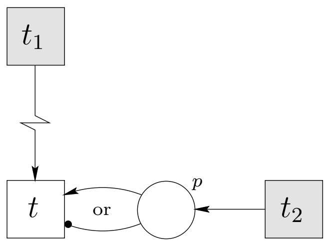
7.3. Diamond Reduction based on Strongly Connected Sets

The semantic definition given above does not give structural criteria for an efficient implementation. Our main idea is the following: We introduce a relation ◀ for spontaneous transitions and show that only strongly connected sets of spontaneous transitions with respect to ◀ need to be considered in reachability analysis. Strongly connected sets are like strongly connected components of a directed graph, i.e. every node is transitively connected with each other, but are not necessarily maximal with respect to set inclusion.

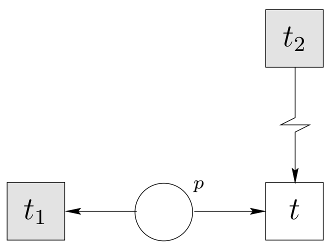
Definition 7.3 (Relation $\triangleleft$ for transitions)
Two transitions $t_1, t_2 \in T$ are in relation $\triangleleft$ iff one of the following conditions holds true:
(D1) $Bt_1 \cap Ft_2 \neq \emptyset$



(D2) t_1 forces t and $(Bt \cup Ft) \cap t_2F \neq \emptyset$



(D3) t_2 forces t and $Ft_1 \cap Ft \neq \emptyset$



(D4) a transition t is forced by t_1 and t_2

Note that the first two conditions in this definition are the two cases we considered in Definition 3.2 and Theorem 3.1 of [Roc00a].

Definition 7.4 (Relation $\blacktriangleleft$ for spontaneous transitions)
Two spontaneous transitions $t_1', t_2' \in \mathit{Spont}$ are in relation $\blacktriangleleft$ iff there are transitions $t_1, t_2 \in T$ such that $t_1 \lhd t_2$, t_1 is transitively forced by t_1', and t_2 is transitively forced by t_2'.
Summarized we have

$$\blacktriangleleft \; := \; S^* \circ \left(\underbrace{(B \circ F)}_{\text{D1}} \cup \underbrace{(S \circ (B \cup F) \circ F^{-1})}_{\text{D2}} \cup \underbrace{(F^{-1} \circ F \circ S^{-1})}_{\text{D3}} \cup \underbrace{(S \circ S^{-1})}_{\text{D4}} \right) \circ (S^{-1})^*$$

Theorem 7.5
If a step s is executable in a marking m leading to a marking m', i.e. $m \xrightarrow{s} m'$, and the set of spontaneous transitions in s, i.e. $s \cap \mathit{Spont}$, is not a strongly connected set of $(\mathit{Spont}, \blacktriangleleft)$ then s is reducible in m, i.e. m' is reachable by an interleaving of two disjoint steps s_1 and s_2.

Proof. Since $s \cap \mathit{Spont}$ is not a strongly connected set of $(\mathit{Spont}, \blacktriangleleft)$ there is at least one strongly connected component $s_1' \subseteq s \cap \mathit{Spont}$ such that $s_1' \not\blacktriangleleft s_2'$ for $s_2' := (s \cap \mathit{Spont}) \setminus s_1'$. It is clear that $s_1' \neq \emptyset \neq s_2'$ and we will show that s_1' and s_2' are sets of spontaneous transitions of two steps $s_1 := s_1' S^* \cap s$ and $s_2 := s_2' S^* \cap s$ such that

1. $s_1 \cup s_2 = s$
2. $s_1 \cap s_2 = \emptyset$
3. $m \xrightarrow{s_2}$
4. $m \xrightarrow{s_2\, s_1} m'$.

ad 1. Check the construction of s_1 and s_2.

$$\begin{aligned} s_1 \cup s_2 &= (s_1' S^* \cap s) \cup (s_2' S^* \cap s) \\ &= (s_1' S^* \cup s_2' S^*) \cap s \\ &= (s_1' \cup s_2') S^* \cap s \\ &= (s \cap \mathit{Spont}) S^* \cap s \\ &= s \end{aligned}$$

ad 2. Assume there is $t \in s_1 \cap s_2$. Since $s_1' \cap s_2' = \emptyset$ we have $t \notin \mathit{Spont}$ and t must be transitively forced by s_1' and s_2', i.e. we have $t \in s_1' S^*$ and $t \in s_2' S^*$, but this contradicts the assumption $s_1' \not\blacktriangleleft s_2'$ (see D4).

ad 3. We will show that s_2 is executable in m.

i. s_2 contains spontaneous transitions, since $s'_2 \neq \emptyset$

ii. $s_2^- \leq m$, since $m \xrightarrow{s}$ and $s_2 \subseteq s$

iii. $\widehat{s_2} \leq m$ holds for the same reason

iv. s_2 is signal-complete since s is signal-complete and every transition t in s_2 does not receive signals from transitions in s_1, otherwise we get the same contradiction as in 2

v. s_2 is signal-closed
Assume there is a forced transition $t \notin s_2$ such that $s_2 \cup \{t\}$ fulfills i–iv, i.e. $s_2^- + t^- \leq m$ and $\widehat{t} \leq m$ and t is forced by s_2. With $t \in s'_2 S^*$ we can conclude that $t \notin s$ (otherwise we have $t \in s_2$) and that $s^- + t^- \not\leq m$ (conditions can not prevent t from firing together with s since $\widehat{t} \leq m$). This implies that there must be a pre-place $p \in Fs \cap Ft$ such that $s^-(p) + t^-(p) > m(p)$, with $s_2^- + t^- \leq m$ and $s^- = s_1^- + s_2^-$ we can conclude that $p \in Fs_1$, but this contradicts the assumption $s'_1 \blacktriangleleft s'_2$ (see D3).

ad 4. Since $m' = m - s^- + s^+ = m - s_1^- + s_1^+ - s_2^- + s_2^+$ and $m \xrightarrow{s_2}$, we just have to show, that s_1 is executable after firing s_2 in m. Set $m'' := m - s_2^- + s_2^+$.

i. s_1 contains spontaneous transitions, since $s'_1 \neq \emptyset$

ii. $s_1^- \leq m''$, since $m \xrightarrow{s}$ implies $s_1^- + s_2^- \leq m$ and obviously $s_1^- + s_2^- \leq m + s_2^+$

iii. $\widehat{s_1} \leq m''$
Assume there is a condition $p \in Bs_1$ such that $\widehat{s_1}(p) > m''(p)$ Since $m \xrightarrow{s}$ implies $\widehat{s_1}(p) \leq m(p)$ we have $p \in Fs_2$, but this contradicts the assumption $s'_1 \blacktriangleleft s'_2$ (see D1)

iv. s_1 is signal-complete since s is signal-complete and every transition t in s_1 does not receive signals from transitions in s_2, otherwise we get the same contradiction as in 2

v. s_1 is signal-closed
Assume there is a forced transition $t \notin s_1$ such that $s_1 \cup \{t\}$ fulfills i–iv, i.e. $s_1^- + t^- \leq m''$ and $\widehat{t} \leq m''$ and t is forced by s_1. With $t \in s'_1 S^*$ we can conclude that $t \notin s$ (otherwise we have $t \in s_1$) and that $s^- + t^- \not\leq m$ or $\widehat{t} \not\leq m$, i.e. there is a place $p \in Bt \cup Ft$ such that $\widehat{t}(p) > m(p)$ or $s^-(p) + t^-(p) > m(p)$. With $s^- = s_1^- + s + 2^-$ we conclude $s_1^-(p) + t^-(p) > m(p) - s_2^-(p)$ and since $s_1 \cup \{t\}$ is enabled in $m''(p) = m - s_2^-(p) + s_2^+(p)$ we have $p \in s_2 F$, but this contradicts the assumption $s'_1 \blacktriangleleft s'_2$ (see D2).

Altogether we can conclude that s_1 is executable after firing s_2 in m, i.e. we have $m \xrightarrow{s_2\, s_1} m'$ and thus s is reducible in m. □

Corollary 7.6

We can construct a reduced reachability graph which contains the same reachable markings as the unreduced graph if we modify condition 1 of the step definition by

1' s contains a strongly connected set of $(Spont, \blacktriangleleft)$.

7.4. Final Remarks

The diamond reduction is implemented in SESA for the normal firing rule. Use `-diamond` to select it. The given criterion is sufficient but not necessary to detect situations, where simultaneous firing of steps can lead to markings that are not reachable by pure interleaving.

Notice that some properties change if the reachability graph computed under the normal firing rule is compared with the reachability graph with diamond reduction: distances, length of shortest paths and some CTL formulae. The model checker in SESA warns if a diamond reduced graph was generated and the truth value of a CTL formula in the complete graph is not deducible from the reduced graph. Due to reducible steps, you will only get an upper bound of the minimal length for paths to target states, although you will get exact results for minimal values. Diamond reduction is not available for nets with timing constraints, synchronisation sets, and priorities. You can also compute the diamond reduced firing list in the simulator of SESA.

This section was influenced by [Vog97, VSY98], where we first saw the need for considering something like our relation, but Vogler et. al. draw other consequences, because they have something other in mind. The notion of positive and negative contexts in [MR95] helps us, to understand and analyze the behavior of signal-net systems, although we do follow the view of concurrency as stated by Janicki and Koutny in [JK91a, JK91b, JK93, JK95]. The results were published in [Roc99b, Roc99a, Roc00a, Roc01]. Properties of the diamond reduced reachability graph were investigated in [Ber02].

8. Stubborn Sets

State space analysis is a powerful formal method for the verification of concurrent and distributed systems. Unfortunately, this method is limited by the state space explosion problem: the number of states tends to be very large even for small systems.

The stubborn set method is one of the techniques that try to alleviate the state space explosion problem: It takes advantage of concurrency by detecting situations where actions can occur in arbitrary order and tries to reduce the set of action sequences to be computed. This is done by executing only a subset of the set of all enabled actions in a state. The basic method preserves all terminal states (deadlocks) and the existence of nontermination [Val91, Var93, Val94]. More elaborated versions can handle even more properties, see e.g. [Sch99, KV00]. For an overview on state space reduction methods see [Val98]. Stubbornness for signal-net systems was first presented in [Roc98a, Roc98b].

8.1. Dynamic Stubbornness

The principles of stubborn sets can be defined for any modeling formalism in which actions depend on local states. A stubborn set is a subset of all possible actions. In signal-net systems local states are described by the marking of the places. The active parts are the steps. Consequently, a stubborn set $Stub(m)$ in a marking m of a signal-net system is a set of steps. Dynamic stubbornness in signal-net systems can then be defined on the base of the following two principles.

Definition 8.1 (Principle SD1)
A set $Stub(m)$ of steps fulfills the first principle of dynamic stubbornness (SD1 for short) iff no step disabled at m in $Stub(m)$ can become enabled as a result of firing steps outside $Stub(m)$:

$$\forall s \in Stub(m) \; : \neg m \xrightarrow{s} \quad \Rightarrow \quad \forall \sigma \in \left(\overline{Stub(m)}\right)^* \; : \; m \xrightarrow{\sigma}\Rightarrow \neg m \xrightarrow{\sigma s} .$$

$$\begin{tikzcd} m \arrow[r, "\sigma"] \arrow[d, "s"', "/" marking] & m_1 & \overset{\textbf{SD1}}{\Longrightarrow} & m \arrow[r, "\sigma"] \arrow[d, "s"', "/" marking] & m_1 \arrow[d, "s", "/" marking] \\ {} & & & {} & {} \end{tikzcd}$$

Definition 8.2 (Principle SD2)
A set $Stub(m)$ of steps fulfills the second principle of dynamic stubbornness (SD2 for short) iff all enabled steps at m in the stubborn set do accord with all steps outside

Stub(m):

$$\forall s \in \mathit{Stub}(m)\ \forall s' \notin \mathit{Stub}(m)\ \forall \sigma \in \left(\overline{\mathit{Stub}(m)}\right)^{*}\ \forall m'\ :$$
$$(m \xrightarrow{s} \wedge m \xrightarrow{\sigma} m' \wedge m' \xrightarrow{s} \wedge m' \xrightarrow{s'}) \ \Rightarrow\ (m' \xrightarrow{ss'} \wedge m' \xrightarrow{s's}),$$

$$\begin{tikzcd} m \arrow[r, "\sigma"] \arrow[d, "s"'] & m' \arrow[r, "s'"] \arrow[d, "s"'] & m_1 \\ {} & m_2 & \end{tikzcd} \quad \overset{\textbf{SD2}}{\Longrightarrow} \quad \begin{tikzcd} m' \arrow[r, "s'"] \arrow[d, "s"'] & m_1 \arrow[d, "s"] \\ m_2 \arrow[r, "s'"'] & m_3 \end{tikzcd}$$

Additionally, dynamic stubborn sets need to contain an enabled step:

Definition 8.3 (Dynamic stubbornness)
A set *Stub*(m) of steps is said to be a *strongly dynamic stubborn set* at m iff *Stub*(m) fulfills SD1 and SD2 and $\exists s \in \mathit{Stub}(m)\ :\ m \xrightarrow{s}$.

Applying the stubborn set method when generating a reachability graph means to fire, in every state, only the enabled actions that belong to a given (or computed) stubborn set at that state. The principles presented above are sufficient for weaker definitions of (dynamic) stubborn sets. Dynamic stubbornness does not seem to lead to a practical algorithm for computing stubborn sets.

8.2. Static Stubbornness for single spontaneous transition steps

To compute stubborn sets at a given marking, structural criteria are needed. These criteria for so-called static (or true) stubbornness should be based only on the current marking and on structural properties of the net.

This leads to no good reduction if the normal firing rule is assumed, since in this case the union of two concurrently enabled steps is also enabled and the unified step is always in a conflict with its two substeps. But unfortunately criteria for stubbornness are sensitive to such conflicts. Therefore we limit ourself only to the firing rule “single spontaneous transition”.
In the sequel, we will apply the term step to a subset $s \subseteq T$ only if

1. s contains exactly one spontaneous transition and

2. s is signal-complete.

We use the notion of markup- and markdown-sets.

Definition 8.4 (Markup- and markdown-sets)
Let N be a *SNS*, p a place and m a marking of N. Then

$$\mathit{Markup}(p,m) \;:=\; \left\{\, s \;\middle|\; \Delta s(p) > 0 \wedge s^-(p) \le m(p) \wedge \widehat{s}(p) \le m(p) \,\right\}$$

and

$$\mathit{Markdown}(p,m) \;:=\; \left\{\, s \;\middle|\; \Delta s(p) < 0 \,\right\}.$$

are the *markup-* and *markdown-set* of p at m, respectively.

Intuitively, $\mathit{Markup}(p,m)$ is the set of steps that could increase the number of tokens in p and are not disabled by p at m. Correspondingly, $\mathit{Markdown}(p,m)$ is the set of steps that could decrease the number of tokens in p. Note that in the implementation in SESA we have implemented $\mathit{Markup}(p,m)$ marking independent, i.e. we have skipped the condition $s^-(p) \le m(p) \wedge \widehat{s}(p) \le m(p)$. This has no consequences for the theory, but makes the implementation much easier.

It is obvious that a step s is executable at a marking m only if s both has token-concession and is condition enabled and is also maximal at m. Consequently, static criteria for stubbornness must take all of these into account. We start with two criteria for principle SD1.

Definition 8.5 (Criterion C1a)
If $s_1 \in \mathit{Stub}(m)$ and

$$\exists\, p \;\big(\, p \in P \wedge \big(\, s_1^-(p) > m(p) \vee \widehat{s_1}(p) > m(p) \,\big)\big).$$

then s_1 is disabled due to a so called *scapegoat place* p and $\mathit{Stub}(m)$ must contain the markup-set $\mathit{Markup}(p,m)$ of one of these p.

Lemma 8.6
If a set $\mathit{Stub}(m)$ fulfills criterion C1a, then $\mathit{Stub}(m)$ fulfills the principle SD1 with respect to token-concession and condition enabling of steps.

Proof. We will show that the following is true: For a disabled step $s_1 \in \mathit{Stub}(m)$ with a scapegoat place p is $m'(p) \le m(p)$ in all reachable markings m' with $m \xrightarrow{\sigma} m'$ and $\sigma \in \left(\overline{\mathit{Stub}(m)}\right)^*$.

We use induction on σ. The claim holds trivially when restricted to $\sigma = \varepsilon$. Our induction hypothesis is that the claim holds when restricted to any σ. Let $\sigma' = \sigma s_2$ with $s_2 \notin \mathit{Stub}(m)$ and $m \xrightarrow{\sigma} m_1 \xrightarrow{s_2} m_1'$. $s_2 \notin \mathit{Stub}(m)$ implies $s_2 \notin \mathit{Markup}(p,m)$. This implies either s_2 is not enabled with respect to the pre-place or condition p, i.e. $s_2^- > m(p)$ or $\widehat{s_2}(p) > m(p)$, or $\Delta s_2(p) \le 0$. The first case contradicts our assumption $m_1(p) \xrightarrow{s_2}$ because we have $m_1(p) \le m(p)$ by induction hypothesis. In the second case we conclude $m'(p) \le m_1(p)$ and using the induction hypothesis we get $m_1'(p) \le m(p)$.

Thus we conclude that s_1 can not become enabled by firing only steps from the complement of $\mathit{Stub}(m)$ because p remains to be a scapegoat. $\square$

Definition 8.5 is an extension of the criterion given in [Val91] for Petri nets.

Definition 8.7 (Criterion C1b)
If $s_1 \in \mathit{Stub}(m)$ and

$$\exists\, t\ \big(\ t \in T \wedge t \text{ forced by } s_1 \\ \Rightarrow \forall\, p\ \big(\ p \in Ft \cup Bt \wedge\ \big(\ t^-(p) \leq m(p) - s_1^-(p) \wedge \widehat{t}(p) \leq m(p)\ \big)\ \big)\ \big),$$

then s_1 is not maximal due to a so called *sufficiently marked transition* t with *sufficiently marked places* p and $\mathit{Stub}(m)$ must include the markdown-sets $\mathit{Markdown}(p, m)$ of all these p of one such t.

Lemma 8.8
If a set $\mathit{Stub}(m)$ fulfills criterion C1b, then $\mathit{Stub}(m)$ fulfills the principle SD1 with respect to maximality of steps.

Proof. We will show that the following is true: For a non-maximal step $s_1 \in \mathit{Stub}(m)$ with a sufficiently marked transition t having sufficiently marked places p is $m'(p) \geq m(p)$ in all reachable markings m' with $m \xrightarrow{\sigma} m'$ and $\sigma \in \left(\overline{\mathit{Stub}(m)}\right)^*$.

We use induction on σ. The claim holds trivially when restricted to $\sigma = \varepsilon$. Our induction hypothesis is that the claim holds when restricted to any σ. Let $\sigma' = \sigma s_2$ with $s_2 \notin \mathit{Stub}(m)$ and $m \xrightarrow{\sigma} m_1 \xrightarrow{s_2} m_1'$. $s_2 \notin \mathit{Stub}(m)$ implies $s_2 \notin \mathit{Markdown}(p, m)$. This implies $\Delta s_2(p) \geq 0$. We conclude $m'(p) \geq m_1(p)$ and using the induction hypothesis we get $m_1'(p) \geq m(p)$.

Thus we conclude that s_1 can not become enabled by firing only steps from the complement of $\mathit{Stub}(m)$ because p remains to be a sufficiently marked place of t and thus t remains to be a sufficiently marked transition. □

Now we define three static criteria for the second principle of dynamic stubbornness SD2.

Definition 8.9 (Criterion C2a)
If $s_1 \in \mathit{Stub}(m)$ and $m \xrightarrow{s_1}$ then

$$\big\{\ s_2 \mid \exists\, p\ \big(\ p \in P \wedge \min\ \big(\ s_1^+(p), s_2^+(p)\ \big) < \min\ \big(\ s_1^-(p), s_2^-(p)\ \big)\ \big\} \quad \subseteq \quad \mathit{Stub}(m).$$

Lemma 8.10
If a set $\mathit{Stub}(m)$ fulfills criterion C2a, then $\mathit{Stub}(m)$ fulfills the principle SD2 with respect to token-concession of steps.

Proof. [Val91] states the same criterion for Petri nets. A simple case analysis shows that steps inside and outside $\mathit{Stub}(m)$ commute in every marking m'. □

Definition 8.11 (Criterion C2b)
If $s_1 \in \mathit{Stub}(m)$ and $m \xrightarrow{s_1}$ then

$$\big\{\ s_2 \mid \exists\, p\ \big(\ (\ p \in Bs_1 \wedge s_2^+(p) < \widehat{s_1}(p) \wedge s_2^-(p) > 0\ ) \\ \bigvee\ \ (\ p \in Bs_2 \wedge s_1^+(p) < \widehat{s_2}(p) \wedge s_1^-(p) > 0\ )\ \big)\ \big\} \subseteq\ \mathit{Stub}(m).$$

Lemma 8.12
If a set $Stub(m)$ fulfills criterion C2b, then $Stub(m)$ fulfills the principle SD2 with respect to conditions of steps.

Proof. Similar to the previous proof [Val91] for pre-places. We show that for every m' we have $m' \xrightarrow{s_1 s_2}$ and $m' \xrightarrow{s_2 s_2}$ with respect to conditions under the assumption $m' \xrightarrow{s_2}$ and $m' \xrightarrow{s_2}$ for $s_1 \in Stub(m)$ and $s_2 \notin Stub(m)$.

We start with $m' \xrightarrow{s_2 s_1}$. Fix a place p. Since $s_2 \notin Stub(m)$, we have either $p \notin Bs_1$, i.e. p has no influence on condition enabling of s_1 at all, or $s_2^+(p) \geq \widehat{s_1}(p)$, i.e. p contains at least $\widehat{s_1}(p)$ tokens after execution of s_2, or $s_2^-(p) = 0$, i.e. s_1 remains to be condition enabled after s_1. The same holds true for $m' \xrightarrow{s_1 s_2}$ due to symmetrie in the criteria. □

Definition 8.13 (Criterion C2c)
If $s_1 \in Stub(m)$ and $m \xrightarrow{s_1}$ and

$$\forall t \; \big(\; t \in T \wedge t \text{ forced by } s_1 \\ \Rightarrow \exists p \; \big(\; p \in Ft \cup Bt \Rightarrow \big(\; t^-(p) \geq m(p) - s_1^-(p) \vee \widehat{t}(p) \geq m(p) \; \big) \; \big) \; \big) \; .$$

then every such t is a *blocked transition* having a *blocked place* p and $Stub(m)$ must include for every such t the markup-set $Markup(p, m)$ of one corresponding p.
Furthermore

$$\big\{ \; s_2 \; \big| \; \exists t \; \big(\; t \in T \wedge t \text{ forced by } s_2 \; \wedge \; \exists p \; \big(\; p \in Ft \cup Bt \\ \wedge \; \big(\; t^-(p) + s_2^-(p) > [\min](p) \vee \widehat{t}(p) > [\min](p) \; \big) \wedge \Delta s_1(p) > 0 \; \big) \; \big) \; \big\} \subseteq Stub(m),$$

with

$$[\min](p) \quad := \quad \max \; \big(\; s_1^-(p), s_2^-(p), \widehat{s_1}(p), \widehat{s_2}(p) \; \big)$$

where $[\min](p)$ is the minimal marking needed for the enabling of both s_1 and s_2 with respect to pre-places and conditions p.

Lemma 8.14
If a set $Stub(m)$ fulfills criterion C2c, then $Stub(m)$ fulfills the principle SD2 with respect to maximality of steps.

Proof. First we have to show that s_1 is maximal after firing of s_2 in every marking m' which is reachable from m only by steps from the complement of $Stub(m)$. This can be done by induction like in the proof for criterion C1b. Since $Markup(p, m)$ is included in $Stub(m)$, every blocked, i.e. disabled forced transition t remains disabled in m' with respect to a blocked place p and this is also also true after firing s_2 in m' since $s_2 \notin Markup(p, m)$.

Now we have to show that s_2 is maximal after firing of s_1 in every marking m' in which s_1 and s_2 are enabled. We have to show that every t forced by s_2 remains to be a blocked transition with respect to a blocked place p because s_1 does not increase the marking of p. We observe that $[\min](p)$ is the minimal marking of p in m' for the enabling of s_1 and s_2. If $t^-(p) + s_2^-(p) \leq m'(p)$ and $\widehat{t}(p) \leq m'(p)$ then s_2 would not be maximal in m', i.e. we have at least one $p \in Ft \cup Bt$ such that $t^-(p) + s_2^-(p) > m'(p) \geq [\min](p)$ or $\widehat{t}(p) > m'(p) \geq [\min](p)$. In this case we get $\Delta s_1(p) \leq 0$ since $s_2 \notin \mathit{Stub}(m)$, i.e. s_2 remains to be maximal after firing s_1 in m'. □

Putting all things together we have:

Definition 8.15 (Static stubbornness)
A set $\mathit{Stub}(m)$ of steps is said to be a *static stubborn set* at m iff for all $s \in \mathit{Stub}(m)$ either s is not executable in m

then criterion C1a or C1b must be fulfilled

or $m \xrightarrow{s}$

then criteria C2a, C2b and C2c must be fulfilled.

Theorem 8.16
If a set $\mathit{Stub}(m)$ of steps contains an enabled step and is static stubborn at m, then $\mathit{Stub}(m)$ is a strongly dynamic stubborn set at m.

8.3. Static Stubbornness for normal steps

In principle it is possible to combine the diamond reduction presented in Section 7 with the stubborn set method if we consider steps with spontaneous transitions forming a strongly connected set of $\blacktriangleleft$ instead of single spontaneous transition steps, i.e. if we apply the term step to a subset $s \subseteq T$ only if

1'. s contains a strongly connected set of $(\mathit{Spont}, \blacktriangleleft)$ and

2. s is signal-complete.

All dynamic principles and static criteria defined before still hold, but the practical computation is much more complex, because larger and more transition sets have to be handled during the calculation of stubborn sets.

8.3.1. Step approximation and simpler static criteria

We concentrate on the spontaneous transitions of a step to cut down the number of transition sets during the stubborn set calculation, i.e. we treat all steps with the same set of spontaneous transitions equal. Additionally we use stronger static criteria.

Let N be a *SNS*, *Spont* the set of spontaneous transitions of N and $\blacktriangleleft$ the relation defined in section 7. Then

$$\mathit{StrConS} \;:=\; \left\{\, s \;\middle|\; s \text{ is a strongly connected set of } (\mathit{Spont}, \blacktriangleleft) \,\right\}.$$

Throughout the rest of this section we will use unprimed variables s, s_1, s_2 for sets of spontaneous transitions from *StrConS*, and primed variables s', s_1', s_2' for steps build from such sets by inclusion of (transitively) forced transitions.

Definition 8.17 (Criterion A1)
If $s_1 \in ApproxStub(m)$ for $s_1 \in StrConS$ and

$$\exists\, p\ \big(\ p \in P \wedge \big(\ s_1^-(p) > m(p) \vee \widehat{s_1}(p) > m(p)\ \big)\big)\ .$$

then s_1 is disabled due to a so called *scapegoat place* p and $ApproxStub(m)$ must contain $s_2 \in StrConS$ for a fixed scapegoat place p iff $p \in s_2'F$ for $s_2' := s_2S^*$.

Definition 8.18 (Criterion A2)
If $s_1 \in ApproxStub(m)$ for $s_1 \in StrConS$, $s_1^- \leq m$, and $\widehat{s_1} \leq m$, i.e. s_1 is enabled, then $ApproxStub(m)$ must contain $s_2 \in StrConS$ iff one of the following conditions holds true for $s_1' := s_1S^*$ and $s_2' := s_2S^*$:
(A2a) $Fs_1' \cap Fs_2' \neq \emptyset$
or
(A2b) $Bs_1' \cap Fs_2' \neq \emptyset$
or
(A2c) $Fs_1' \cap Bs_2' \neq \emptyset$
or there exists a forced transition $t \notin Spont$ with
(A2d) $t \in s_1'$ and $(Bt \cup Ft) \cap s_2'F \neq \emptyset$
or
(A2e) $t \in s_2'$ and $s_1'F \cap (Bt \cup Ft) \neq \emptyset$.

Definition 8.19 (Approximative static stubbornness)
A set $ApproxStub(m)$ of strongly connected sets of $(Spont, \blacktriangleleft)$ is said to be an *approximative static stubborn set* at m iff $ApproxStub(m)$ fulfills criteria A1 and A2.

Theorem 8.20
If $ApproxStub(m)$ is an approximative static stubborn set at m, and

$$Stub(m) := \{\, s' \,\big|\, s' \subseteq sS^* \text{ such that } s' \text{ is a step with } s \subseteq s' \text{ for } s \in Stub(m)\,\}$$

contains an enabled step, then $Stub(m)$ is a strongly dynamic stubborn set at m.

Proof. Let $s_1' \in Stub(m)$ and $s_1 := s_1' \cap Spont$. First we show that SD1 holds true, i.e. we assume that s_1' is not fireable in m.

If s_1 is not enabled, i.e. there is a scapegoat place p for s_1 (see criterion A1), then p is a scapegoat place for s_1', too. Induction proofs that s_1' can not become enabled by firing only steps from the complement of $Stub(m)$ because p remains to be a scapegoat place (same argumentation as in the proof for criterion C1a).

If s_1 is enabled, then there may be a scapegoat place p for s_1'. Due to criterion A2d (there is a transition $t \in s_1'$ with $t \notin Spont$) every step containing a pre-transition of such a p is included in $Stub(m)$ and p remains to be a scapegoat place (same argumentation as before).

If s_1' has token-concession and is condition enabled, then s_1' may not be maximal due to a sufficiently marked transition t (see criterion C1b). Due to condition A2a and

A2b t remains to be enabled in every marking reachable by firing only steps from the complement of $Stub(m)$ (same argumentation as in the proof for criterion C1b).

Now we assume that s_1' is fireable in m and show that SD2 holds true. Due to A2a, A2b and A2c s_1' and $s_2' \notin Stub(m)$ commute in every marking with respect to token-concession and conditions (see C2a and C2b). It remains to show SD2 with respect to maximality.

Every blocked transition t forced by s_1' remains to be disabled due to criterion A2d (see proof of the first part of criterion C2c), i.e. s_1' remains to be maximal after firing $s_2' \notin Stub(m)$ in m. The same holds for $s_2' \notin Stub(m)$ after firing s_1' due to condition A2e. Together we conclude that SD2 holds true and $Stub(m)$ is a static stubborn set at m. □

Remark. For Petri nets, where *StrConS* contains only singleton sets, only the criteria A1 and A2a need to be considered. They match very simple static criteria which ignore arc weights and which can be found in literature and in implementations [Val94].

8.3.2. Reduction of disabled steps

Still the number of steps in *StrConS* is the limiting factor in the stubborn set computation. Without deeper analysis of ◀ the number of nontrivial strongly connected sets tends to be very large for practical nets. Such a set is nontrivial if it contains more than one spontaneous transition. In this section we will provide a theorem, which allows to reduce the number of disabled steps need to be considered during the stubborn set computation.

We start with an observation.

Proposition 8.21
If $s_1 \blacktriangleleft s_2$, $s_1 \in ApproxStub(m)$, and s_1 is enabled in m, then $s_2 \in ApproxStub(m)$ or a transition is transitively forced by s_1 and s_2, i.e. $s_1 S^* \cap s_2 S^* \neq \emptyset$.

Proof. Check the cases D1, D2, D3, and D4 in the definition of ◀. They match (in order) the conditions A2b, A2d, and A2a or the additional case mentioned in this proposition (transition transitively forced by both steps). □

Definition 8.22 (Criterion A2 revised)
We modify A2 defined in 8.18 and add the following case:
(A2f) $s_1' \cap s_2' \neq \emptyset$ i.e. a transition is transitively forced by s_1 and s_2.

Now we are ready to define and justify a new criterion.

Definition 8.23 (Criterion A3)
Do not add $s \in StrConS$ to $ApproxStub(m)$ by criterion A1 or A2 iff the following is true: s contains more than one spontaneous transition and at least one transition $t \in s$ is not enabled in m, i.e. there is at least one scapegoat place p such that $t^-(p) > m(p)$ or $\widehat{t}(p) > m(p)$.

Theorem 8.24
If we obey criterion A2f then it is not necessary to further investigate any nontrivial s which is prevented from inclusion in ApproxStub(m) according to criterion A3.

Proof. Due to the scapegoat place p of t, the nontrivial step s is not enabled in m. Thus, according to the principle SD1 we have to ensure that s is not enabled in every marking reachable only by steps build by spontaneous transition sets from the complement of $\mathit{ApproxStub}(m)$. This is done with criterion A1 by ensuring that $\mathit{ApproxStub}(m)$ contains a step $s_2 \in \mathit{StrConS}$ iff $p \in s_2'F$ for $s_2' := s_2S^*$ and a scapegoat place p.

We will show the following: If we obey A2f and A3 and skip s from further investigation during the stubborn set calculation, then s is automatically not enabled in every marking reachable by firing only steps build by spontaneous transition sets from the complement of $\mathit{ApproxStub}(m)$. Fix m, s and $t \in s$ which is not enabled in m.

First we concentrate on the moment were we (without A3) had to include s and show that a nonempty subset $s_i \subset s$ is in $\mathit{ApproxStub}(m)$. Check the conditions in A1 and A2 to see that at least one single spontaneous transition t_i is the reason for the inclusion of s and therefore s_i, i.e. we have shown that there is at least $t_i \in s_i$ and $s_i \in \mathit{ApproxStub}(m)$. It is clear that either $t \in s_i$ or $t \in s_d$ with $s_d := s \setminus s_i$. Since s contains more than one spontaneous transition, s_d is nonempty.

If $t \in s_i$ then s_i has not enough tokens in m and A1 prevents s_i from firing. Since $s_i \subset s$ this automatically also prevents s from becoming enabled, i.e. we do not have to consider s for the stubborn set computation.

In the second case with $t \in s_d$, it remains to show that s_d is in $\mathit{ApproxStub}(m)$. Since $s \in \mathit{StrConS}$ we have $s_i \blacktriangleleft s_d$. Obviously A2f is the additional case mentioned in Proposition 8.21 (transition transitively forced by both steps). Since $s_i \in \mathit{ApproxStub}(m)$ is enabled in m we can apply 8.21 and conclude that $s_d \in \mathit{ApproxStub}(m)$. Now we can use the same argumentation as before to show that again s is automatically prevented from firing due to A1 and $s_d \subset s$. □

8.4. Final Remarks

The basic stubborn set method preserving deadlocks and infinite paths is implemented in our tool SESA. The stubborn set method combined with the attractor set technique described in [Sch99] without the improvements of [KV00] is available for deciding reachability and the computation of paths to (partial) markings or to markings fulfilling a given state predicate. Note that the computed path has not necessarily minimal length if stubborn reduction is applied for the normal firing rule: you will only get an upper bound of the minimal length, due to reducible steps. If you use single spontaneous transition steps or are interested in minimal values only, you will get exact results. For single spontaneous transition steps you can choose between step approximation and the non-approximative computation of stubborn sets (use `-noapprox`). For the normal firing rule the approximative approach together with the reduction of disabled steps is implemented in SESA. In either case use `-stubborn` to select stubborn set reduced reachability. You can also check the stubborn set computation in the simulator of SESA.

Stubborn reduction is only available for nets without timing constraints, greedy transitions, synchronisation sets, and priorities. Deduction of truth values of CTL formulae from stubborn reduced graphs is not implemented in the SESA model checker.

9. Symmetries

Many systems are composed of many identical subsystem. Thus, the global system has a symmetric structure. Symmetric structure yields symmetric behavior. Knowing the symmetries, only parts of the system behavior need to be explored explicitly. In this section, we discuss the technology of symmetrically reduced reachability graphs. First, we define symmetries on the system structure. Then, the relation between symmetric structure and symmetric behavior is established. We define symmetrically reduced reachability graphs and study the derivation of properties from the reduced graph. Finally, we discuss briefly the availability of algorithms for the generation and interpretation of symmetrically reduced graphs.

A *symmetry* is a mapping of the system onto itself that preserves the system structure. For signal-net systems, a symmetry is a bijection of the nodes (i.e. places and transitions) that preserves the node type and the connecting arcs. Characteristics of inscriptions (for instance, the mode of transitions with respect to incoming signals) must be preserved, too. For arc-timed signal-net systems, symmetries respect additionally the time annotations of the pre-arcs.

Definition 9.1
A bijection σ of $P \cup T$ is a symmetry of the *SNS* $N = [P, T, F, V, B, W, S, M, m_0]$ iff

1. $\sigma(P) = P$, $\sigma(T) = T$ (σ respects the node type);
2. for all $x, y \in P \cup T$ and $i \in \mathbb{N}$, if $[x, y] \in F$ and $V([x, y]) = i$ then $[\sigma(x), \sigma(y)] \in F$ and $V([\sigma(x), \sigma(y)]) = i$ (σ respects the flow-arcs and their weight);
3. for all $p \in P,\ t \in T,\ i \in \mathbb{N}$, if $[p, t] \in B$ and $W([p, t]) = i$ then $[\sigma(p), \sigma(t)] \in B$ and $W([\sigma(p), \sigma(t)]) = i$ (σ respects the condition arcs and their weight);
4. for all $t, t' \in T$, $[t, t'] \in S$ iff $[\sigma(t), \sigma(t')] \in S$ (σ respects the signal arcs);
5. for all $t \in T$, $M(t) = M(\sigma(t))$ (σ respects the signal processing mode).

σ is a symmetry of an *arc-timed* signal-net system iff, additionally, for all all pre-arcs $[p, t] \in F$, $eft(p, t) = eft(\sigma(p), \sigma(t))$ and $lft(p, t) = lft(\sigma(p), \sigma(t))$ (i.e. σ respects the arc-time intervals).

If σ_1 and σ_2 are symmetries then so are $\sigma_1 \circ \sigma_2$ and σ_1^{-1}. Thereby $\sigma_1 \circ \sigma_2(x) = \sigma_2(\sigma_1(x))$. σ^{-1} is defined by the relation $\sigma^{-1}(x) = y$ iff $\sigma(y) = x$. Consequently, the set of all symmetries (denoted by Σ_N) with the concatenation $\circ$ and the inversion $^{-1}$ forms a subgroup of the group of all bijections of $P \cup T$. Every subgroup of Σ_N is called *symmetry group*.

Let Σ be a symmetry group. Σ induces a relation between nodes and a relation between markings. For nodes x, y, let $x \sim_\Sigma y$ iff there is a $\sigma \in \Sigma$ such that $\sigma(x) = y$. For a marking m, let $\sigma(m)$ be the mapping satisfying $\sigma(m)(\sigma(p)) = m(p)$ for all $p \in P$. This mapping reflects the "movement" of tokens according to the mapping of nodes. If an arc-timed *SNS* is considered, for clock positions u $\sigma(u)$ is defined in this way (since

u is a marking). Let $m \sim_\Sigma m'$ iff there is a $\sigma \in \Sigma$ such that $\sigma(m) = m'$. Both relations $\sim_\Sigma$ are equivalence relations. Denote the equivalence class of a node or marking z by $[z]_\Sigma$.

A marking m is called *symmetric* with respect to Σ iff $[m] = \{m\}$. A node $x \in P \cup T$ is called *fixed point* of Σ iff $[x]_\Sigma = \{x\}$.

The concepts for nodes can be generalized to steps. For a step s, let $\sigma(s) = \{\sigma(t) \mid t \in s\}$. Since any symmetry σ respects the signal arcs, one verifies easily that $\sigma(Spont) = Spont$ and $\sigma(Forc) = Forc$, and, that $\sigma(s)$ is signal–complete if s is signal-complete. Hence, σ maps steps to steps (preserving the number of spontaneous transitions). A step s is a fixed point of Σ iff, for all $\sigma \in \Sigma$, $\sigma(s) = s$.

Obviously, any symmetry σ of an *SNS* N is a symmetry of the underlying Petri net PN. The converse does not hold.

The relation between symmetric structure and symmetric behavior is reflected by the following theorem.

Theorem 9.2 (Symmetric behavior)
Let N be a signal-net system, m, m' markings, and s a step. Let σ be a symmetry (i.e. $\sigma \in \Sigma_N$). Then $m \xrightarrow{s} m'$ if and only if $\sigma(m) \xrightarrow{\sigma(s)} \sigma(m')$.

Proof. First, we show that $\sigma(s)$ is executable at $\sigma(m)$. For this purpose, we follow the definition of executable steps.

ad 1. We have to show that $\sigma(s) \cap Spont$ is not empty. This is clear because $s \cap Spont$ is not empty and σ maps spontaneous transitions to spontaneous transitions.

ad 2. We have to show that $\sigma(s)$ is signal-complete. Let $t \in \sigma(s)$ and $[t', t] \in S$. Thus, $\sigma^{-1}(t) \in s$. According to the fourth item of Definition 9.1, $[\sigma^{-1}(t'), \sigma^{-1}(t)] \in S$. Since s is signal–complete, we get $\sigma^{-1}(t') \in s$ (for at least one t' if $M(t) = \vee$, for all t' if $M(t) = \wedge$). Consequently, $t' \in \sigma(s)$. Finally, $M(\sigma(t)) = M(t)$ (by 5th item of Def. 9.1). Consequently, $\sigma(s)$ is signal–complete.

ad 3. Due to the second item of Definition 9.1, we have $\sigma(s)^-(p) = \sum_{t\in\sigma(s)} V(p,t) = \sum_{t\in s} V(p,\sigma(t)) = \sum_{t\in T} V(\sigma^{-1}(p), t)$. Since s is executable at m, we have $V(\sigma^{-1}(p), t) \leq m(\sigma^{-1}(p))$. Thus, $\sigma(s)^-(p) \leq m(\sigma^{-1}(p)) = \sigma(m)(p)$. Hence, $\sigma(s)$ has token-concession at $\sigma(m)$.

ad 4. Let $t \in \sigma(s)$, i.e. $\sigma^{-1}(t) \in s$. We show that $\widehat{t} \leq \sigma(m)$, i.e. for all $p \in Bt$ it holds $W(p,t) \leq \sigma(m)(p)$. From $[p,t] \in B$ we have $[\sigma^{-1}(p), \sigma^{-1}(t)] \in B$, since σ respects the condition arcs. Since s is enabled at m and $\sigma^{-1}(t) \in s$, we obtain $W(p,t) = W(\sigma^{-1}(p), \sigma^{-1}(t)) \leq m(\sigma^{-1}(p)) = \sigma(m)(p)$.

ad 5. Assume there is a s^* such that $s^* \supset \sigma(s)$ and s^* is enabled at $\sigma(m)$. By $1, \ldots, 4$, $\sigma^{-1}(s^*)$ is enabled at m. Furthermore, $\sigma^{-1}(s^*) \supset s$. This contradicts the executablity of s at m. Thus, $\sigma(s)$ is executable at $\sigma(m)$.

It remains to show that firing $\sigma(s)$ at $\sigma(m)$ yields $\sigma(m')$. Let $p \in P$. We have

$$\begin{aligned}
&\sigma(m)(p) - \sigma(s)^-(p) + \sigma(s)^+(p) = \\
&\qquad = m(\sigma^{-1}(p)) - \textstyle\sum_{t\in\sigma(s)} V(p,t) + \sum_{t\in\sigma(s)} V(t,p) = \\
&\qquad = m(\sigma^{-1}(p)) - \textstyle\sum_{t\in s} V(\sigma^{-1}(p),t) + \sum_{t\in s} V(\sigma^{-1}(p),t) =
\end{aligned}$$

$$= m'(\sigma^{-1}(p)) = \sigma(m')(p). \qquad \square$$

Corollary 9.3 (Reachability)
A marking m' is reachable from another marking m if and only if $\sigma(m')$ is reachable from $\sigma(m)$. If m is symmetric, then m' is reachable from m if and only if $\sigma(m')$ is reachable from m.

Symmetrically reduced reachability graphs store equivalence classes of states instead of single markings. An equivalence class is represented by one of its elements. Dependent on the algorithm to generate the reduced graph, the representative can either be the member of the class that has been reached first, or the least member of the class with respect to some (lexicographical) order. The general outline of the reduced graph generation is (Σ is the used symmetry group):

VAR Processed, Unprocessed: **SET OF** Marking;
VAR Edges: **SET OF** Marking $\times$ Step $\times$ Marking;
VAR m, m', m'': Marking;
VAR σ: Symmetry;
VAR s: Step;

BEGIN
  Processed := $\emptyset$;
  Unprocessed := $\{m_0\}$;
  Edges := $\emptyset$;
  WHILE Unprocessed $\neq \emptyset$ **DO**
    m := any element of Unprocessed;
    Unprocessed := Unprocessed $\setminus\{m\}$;
    Processed := Processed $\cup$ $\{m\}$;
    FOR ALL $s, m' : m \xrightarrow{s} m'$ **DO**
      IF $\exists \sigma \in \Sigma\ \exists m'' \in$ Unprocessed $\cup$ Processed: $\sigma(m') = m''$ **THEN**
        Edges := (Edges $\cup$ $\{[m, s, m'']\}$);
      ELSE
        Unprocessed = Unprocessed $\cup$ $\{m'\}$;
        Edges := Edges $\cup$ $\{[m, s, m']\}$;
      END;
    END;
  END;
  R_Σ := Processed; E_Σ := Edges;
END.

Obviously, the constructed graph $[R_\Sigma, E_\Sigma]$ is determined only up to symmetries from Σ; if Σ is the identity group then it coincides with the reachability graph.

Corollary 9.4 (Boundedness)
The symmetrically reduced graph $[R_\Sigma, E_\Sigma]$ is finite if and only if the signal-net system is bounded.

Corollary 9.5 (Equivalence)
1. $m_0 \in R_\Sigma \subset R_N(m_0)$.
2. For every $m \in R_N(m_0)$ *there exists exactly one* $m' \in R_\Sigma$ *such that* $m' \sim m$.

From the construction it is clear that there is at most one m' with $m \sim m' \in R_\Sigma$. To show that there is at least one m' we prove by induction on i that for all i and all sequences $m_0 \xrightarrow{s_0} m_1 \xrightarrow{s_1} \ldots \xrightarrow{s_{i-1}} m_i$ there exists a m' such that $m_i \sim m' \in R_\Sigma$. This is trivial for $i = 0$ by $m_0 \in R_\Sigma$. Let $m_i \sim m' \in R_\Sigma$, $m_i \xrightarrow{s_i} m_{i+1}$ and $\sigma \in \Sigma$ such that $\sigma(m_i) = m'$. Then by Theorem 9.2 $\sigma(s_i)$ is an executable step at $\sigma(m_i) = m'$, thus, there exists an edge $[m', \sigma(s_i), m''] \in E_\Sigma$. Obviously, $m_{i+1} \sim m'' \in R_\Sigma$.

In the sequel, let $eq(m)$ be the marking from R_Σ which is equivalent with $m \in R_N(m_0)$. Obviously, $eq(m_0) = m_0$.

With Theorem 9.2 and Corollary 9.3, we obtain

Corollary 9.6 (Reachability II)
1. If for some marking m*, an equivalent one is not contained in* R_Σ*, then* m *is not reachable from* m_0*.*
2. If m *is contained in* R_Σ*, then* m *is reachable.*
3. If m_0 *is symmetric with respect to the symmetry group* Σ *used, then* m *is reachable from* m_0 *if and only if a marking equivalent to* m *is contained in* R_Σ*.*

Consider the case where m is not contained in the reduced graph but a marking m' equivalent to m is. Then Corollary 9.3 states that m is reachable from $\sigma(m_0)$. $\sigma(m_0)$ is not necessarily a reachable marking. Therefore, we cannot derive the reachability of m. If, however, m_0 is symmetric, we have $\sigma(m_0) = m_0$ and we can assert the reachability of m. The stronger deduction rules for reachability justify the use of a non–maximal symmetry group when m_0 is symmetric with respect to the smaller group.

A *SNS* N is said to be *deadlock-free* iff no dead marking is reachable in N. If $m \in R_N(m_0)$ is dead then $eq(m)$ is a leaf in $[R_\Sigma, E_\Sigma]$ and vice versa:

Corollary 9.7 (Deadlock-freedom)
The signal-net system N *is deadlock-free iff* $[R_\Sigma, E_\Sigma]$ *contains no node where no edge emerges.*

For $m', m'' \in R_\Sigma$ we write $m' \xrightarrow{**} m''$ iff there is a path in $[R_\Sigma, E_\Sigma]$ leading from m' to m'', i.e. there is a finite sequence $([m_i, s_i, m_{i+1}])_{i=1,\ldots,k}$ of adjacent edges such that $m' = m_1$ and $m'' = m_k$.

Lemma 9.8
If $m \in R_N(m_0)$ *and* $m \xrightarrow{*} m'$ *then* $eq(m) \xrightarrow{**} eq(m')$*.*

Proof. Let $s_1 \ldots s_k$ be a sequence of steps such that $m \xrightarrow{s_1} m_1 \xrightarrow{s_2} m_2 \ldots m_{k-1} \xrightarrow{s_k} m'$, and, $\sigma \in \Sigma$ with $\sigma(m) = eq(m)$. Then $eq(m) = \sigma(m) \xrightarrow{\sigma(s_1)} \sigma(m_1)$, i.e. $\sigma(s_1)$ is an executable step at $eq(m)$, thus, there exists an edge $[eq(m), \sigma(s_1), m'_1] \in E_\Sigma$. We have $m'_1 \sim \sigma(m_1) \sim m_1$ and $eq(m_1) = m'_1$, i.e.

$eq(m) \xrightarrow{**} eq(m_1)$. By induction we obtain $eq(m) \xrightarrow{**} eq(m')$. □

A *SNS* N is called *reversible* or *resetable* iff its reachability graph is strongly connected, i.e. the initial marking m_0 is reachable from every reachable marking.

Theorem 9.9 (Resetability)
The signal-net system N is resetable iff $[R_\Sigma, E_\Sigma]$ is strongly connected.

Proof. Let N be resetable, $m' \in R_\Sigma$. We have to show that $m' \xrightarrow{**} m_0$, i.e. there is a path from m' to m_0 along edges from E_Σ. By $R_\Sigma \subseteq R_N(m_0)$ and the resetability of N we have $m \xrightarrow{*} m_0$, from which we obtain $eq(m') \xrightarrow{**} eq(m_0)$. Since $eq(m') = m'$ and $eq(m_0) = m_0$ we are ready.

To derive the resetability of N from the strong connectivity of the symmetrically reduced graph we need the following assertion:

If $m_0 \sim m \in R_N(m_0)$, then $m_0 \in R_N(m)$.

Let w be a sequence of steps with $m_0 \xrightarrow{w} m$ and $\sigma \in \Sigma$ such that $\sigma(m_0) = m$. Then $m = \sigma(m_0) \xrightarrow{\sigma(w)} \sigma(m) = \sigma^2(m_0)$. Hence, for all $i \geq 2$ there is a path in the reachability graph from m to $\sigma^i(m_0)$. Since Σ is a finite group there exists a number $j \geq 1$ such that σ^i is the identity. If $j = 1$ the σ itself is the identity, hence, $m = m_0 \in R_N(m)$, otherwise $m_0 = \sigma^j(m_0) \in R_N(m)$.

Now, let m be reachable from m_0 and $m' := eq(m) \in R_\Sigma$. If $m' = m_0$ then $m_0 \sim m$ and $m_0 \in R_N(m)$ follows from the assertion.

Otherwise, $m' \neq m_0$. Let $[m', s_1, m'_1], \ldots$ be a path in $[R_\Sigma, E_\Sigma]$ leading from m' to m_0 and σ_1 such that $\sigma_1(m') = m$. Since s_1 is executable at m', $\sigma_1(s_1)$ is executable at m. Let m_1 be the marking reached when $\sigma_1(s_1)$ is executed at m. Then $m_1 \sim m'_1$. Proceeding in this way we arrive at a path in the reachability graph which leads from m to a certain marking $m_k \sim m_0$. By our assertion we obtain $m_0 \in R_N(m)$. □

Let S_Σ denote the union of all steps s such that there is an edge $[m', s, m''] \in E_\Sigma$.

Theorem 9.10 (Dead transitions)
For all transitions $t \in T$ it holds:
1. If $[t] \cap S_\Sigma = \emptyset$, then t is dead in N.
2. If m_0 is symmetric and t is dead in N, then $[t] \cap S_\Sigma = \emptyset$.

Proof. Ad 1. If t is not dead in N, there exist a reachable marking m and a step s such that $m \xrightarrow{s}$ and $t \in s$. Hence, $eq(m) \in R_\Sigma$ and there exists a $\sigma \in \Sigma$ with $\sigma(m) = eq(m)$. Thus, $\sigma(s)$ is an executable step at $eq(m)$ and for some m' we obtain $[eq(m), \sigma(s), m'] \in E_\Sigma$. Hence, $\sigma(s) \subseteq S_\Sigma$ and for $t' := \sigma(t)$ it holds $t' \in [t] \cap S_\Sigma$.
Ad 2. Assume for contradiction that $[t] \cap S_\Sigma \neq \emptyset$, i.e. that there is a $t' \sim t$ and an edge $[m', s, m''] \in E_\Sigma$ with $t' \in s$. We have $m_0 \xrightarrow{*} m' \xrightarrow{s}$. Let $\sigma \in \Sigma$ such that $\sigma(t') = t$. Then, by the symmetry of m_0, we obtain $m_0 = \sigma(m_0 \xrightarrow{*} \sigma(m') \xrightarrow{\sigma(s)}$. Now, $t \in \sigma(s)$ contradicts that t is dead at m_0. □

A set U of transitions is said to be *collectively live at* m_0 iff from every reachable marking $m \in R_N(m_0)$ there is reachable a marking $m' \in R_N(m)$ such that a step s with $s \cap U \neq \emptyset$ is executable at m'.

Obviously, a transition t from a collectively live set U is not necessarily live, but it holds:

Corollary 9.11
1. A transition t is live at m iff $\{t\}$ is collectively live at m.
2. If U is collectively live at m, then $U \neq \emptyset$ and every superset $U' \supseteq U$ is collectively live at m.
3. N is deadlock-free iff T is collectively live at m_0.

For $m' \in R_\Sigma$ let $S_\Sigma(m')$ be the union of all steps s such that there is a vertex $m'' \in R_\Sigma$ with $m' \xrightarrow{**} m''$ and an edge $[m'', s, m'''] \in E_\Sigma$. Obviously, $S_\Sigma = S_\Sigma(m_0)$.

Theorem 9.12 (Collective liveness)
For every $t \in T$, the equivalence class $[t]$ is collectively live at m_0 iff for all $m' \in R_\Sigma$ it holds $[t] \cap S_\Sigma(m') \neq \emptyset$.

Proof. Let $[t]$ be collectively live at m_0 and $m' \in R_\Sigma$. We have to show that there is a path in $[R_\Sigma, E_\Sigma]$ leading from m' to a vertex m'' where an edge $[m'', s, m''']$ starts with $s \cap [t] \neq \emptyset$.
Since $m' \in R_N(m_0)$ and $[t]$ is collectively live at m_0 there exist a marking $m \in R_N(m')$ and a step s such that $m \xrightarrow{s}$ and $[t] \cap s \neq \emptyset$. We have $m_0 \xrightarrow{*} m' \xrightarrow{*} m \xrightarrow{s}$ and, therefore, $m_0 = eq(m_0) \xrightarrow{**} eq(m') = m' \xrightarrow{**} eq(m)$. Let $\sigma \in \Sigma$ such that $\sigma(m) = eq(m)$. Then, $eq(m) \xrightarrow{\sigma(s)}$. Thus, $m'' := eq(m)$ has properties to prove: $m' \xrightarrow{**} m'' \xrightarrow{\sigma(s)}$ and $\sigma(s) \cap [t] \neq \emptyset$.
Now assume, that for all $m' \in R_\Sigma$ it holds $[t] \cap S_\Sigma(m') \neq \emptyset$ and let $m_1 \in R_N(m_0)$. We have to show that there exist a marking m and a step s such that $m_1 \xrightarrow{*} m \xrightarrow{s}$ and $s \cap [t] \neq \emptyset$.
We consider $m'_1 := eq(m_1) \in R_\Sigma$. From $[t] \cap S_\Sigma(m'_1) \neq \emptyset$ we obtain that there exists edges $[m'_1, s_1, m'_2], [m'_2, s_2, m'_3], \ldots, [m'_k, s_k, m'_{k+1}] \in E_\Sigma$ such that $s_k \cap [t] \neq \emptyset$. Let $\sigma_0 \in \Sigma$ such that $\sigma_0(m'_1) = m_1$ and for $i = 1, \ldots, k$ let $\sigma_i \in \Sigma$ such that $\sigma_i(m'_{i+1}) = \sigma_{i-1}(m'_i + \triangle s_i)$. Then we have
$m_1 = \sigma_0(m'_1) \xrightarrow{\sigma_0(s_1)} \sigma_0(m'_1 + \triangle s_1) = \sigma_1(m'_2) \xrightarrow{\sigma_1(s_2)} \ldots \sigma_{k-1}(m'_k) \xrightarrow{\sigma_{k-1}(s_k)}$.
Hence, for $m := \sigma_{k-1}(m'_k)$ we obtain $m_1 \xrightarrow{*} m \xrightarrow{\sigma_{k-1}(s_k)}$ and $\sigma_{k-1}(s_k) \cap [t] \neq \emptyset$. □

For the implementation of reduced reachability graph algorithms, we need to solve two tasks. First, we need to calculate a representation of the symmetry group Σ used. It should be possible to define restrictions for the group to be calculated (such as keeping the initial state symmetric, defining fixed points and so on). The second task is the decision whether, for a given state m, an equivalent one already has been computed. The latter task is completely compatible to the Petri net case studied in [Sch00b].

For the calculation of the symmetry group, the ideas of [Sch00a] can be adapted.

There, a general framework is provided for many net classes. Nets are considered to consist of places, transitions, and arcs, all of which can be inscribed (using a mapping χ). For signal-net systems, this approach can be directly used if signals and conditions are considered to be special arcs. We can translate a signal-net system $N = [P, T, F, V, B, W, S, M, m_0]$ into a general net $[P, T, F', \chi, \mathcal{I}, m_0]$ according to [Sch00a] by setting:

- $F' = \{[x, y] \mid [x, y] \in F$ or $[x, y] \in S$ or $[x, y] \in B\}$;
- for $x \in T$, $\chi(x) = M(t)$; (if time constraints are involved, $\chi(t)$ assigns the time parameters to a transition);
- for $x \in P$, $\chi(x) = nil$;
- for $[x, y] \in F$, $\chi([x, y] := [''f'', V(x, y)]$;
- for $[x, y] \in S$, $\chi([x, y] := ''s''$;
- for $[x, y] \in B$, $\chi([x, y] := [''b'', W(x, y)]$;

We coded the type of the arc by a constant, and added multiplicities to arc inscriptions. The approach in [Sch00a] does not consider (signal) arcs between transitions, but non of the results depend on this limitation. Therefore, the algorithm presented there can be used to calculate the symmetries of signal-net systems. It returns a generating set of a symmetry group which is has at most $\frac{n(n-1)}{2}$ members (n is the number of nodes of the net). Restrictions such as fixed points and symmetric initial markings can be handled by the algorithm.

10. Conflicts

Two transitions t, t' of a Petri net are said to be in a ("true") *dynamic conflict* at the marking m iff they are both enabled (i.e. $t^- \leq m$ and $t'^- \leq m$) but not concurrently enabled (i.e. $t^- + t'^- \leq m$ holds not). In loop-free nets (where no place is pre-place and post-place of the same transition) this implies that t is not enabled after t' has fired and that t' is not enabled after t has fired. Therefore, this consequence often is used as a definition for conflict, but one has take in account that conflict in this sense is symmetric. In Petri nets with self-loops non-symmetric conflicts appear, e.g. t' is enabled after the firing of t but t is not after the firing of t'.

If t and t' are in a dynamic conflict at m, there exists a place p such that $t^-(p) + t'^-(p) > m(p)$; hence, t and t' have a common pre-place and we say that t and t' are in a *static conflict*. Note that the static conflict relation is symmetric but not transitive. Obviously, in a Petri net without static conflicts, no dynamic conflict can appear.

A non-empty set s of transitions of a Petri net is said to be *concurrent at m* iff $s^- \leq m$. Otherwise, s is called *conflicting at m*. Obviously, if s is concurrent at m then any sequence which contains every transition from s at most once can be executed at m (the converse, again, holds only for loop-free Petri nets). This implies that all states (reachable markings) of a Petri net can be found by firing sequences of single transitions only (instead of concurrent steps). We know from the previous section that this is not the case for *SNS*.

In *SNS*, we have to distinguish between step conflicts and transition conflicts.
Let s_1 and s_2 be executable steps at $m, m \xrightarrow{s_1} m_1$ and $m \xrightarrow{s_2} m_2$. The steps s_1, s_2 are said to be in *symmetric (dynamic) conflict* iff s_1 is not a executable step at m_2 and s_2 is not a executable step at m_1. The steps s_1, s_2 are said to be in *conflict* iff s_1 is not a executable step at m_2 or s_2 is not a executable step at m_1.

Two transitions t_1, t_2 are in *dynamic conflict at m* iff there exist executable steps s_1, s_2 at m with $t_1 \in s_1$, $t_2 \notin s_1$, $t_1 \notin s_2$, $t_2 \in s_2$, $m \xrightarrow{s_1} m_1$, and $m \xrightarrow{s_2} m_2$ such that there is no executable step at m_1 containing t_2 or such that there is no executable step at m_2 containing t_1.

It may be the case that, at a marking m, two executable steps are in a dynamic conflict but no two transitions are (see Figure 10.1). Obviously, the occurrence of a

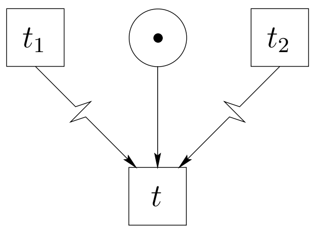


Figure 10.1: $s_1 = \{t_1, t\}$, $s_2 = \{t_2, t\}$ are in symmetric conflict and t_1, t_2 are not in dynamic conflict

dynamic transition conflict implies the occurrence of a step conflict.

Consider the *SNS* of Figure 10.2. At the given marking the steps $s_1 = \{t_1\}$, $s_2 = \{t_2\}$

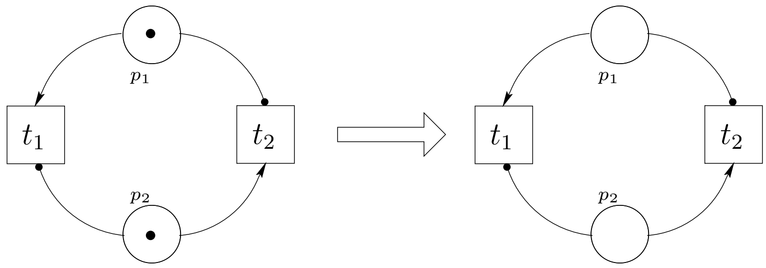


Figure 10.2: $s_1 = \{t_1\}$, $s_2 = \{t_2\}$ and $s = \{t_1, t_2\}$ are executable

and $s = \{t_1, t_2\}$ are executable. After the execution of any of these steps the net is dead, although $s = s_1 \cup s_2$, $s_1 \cap s_2 = \emptyset$, i.e. s_1, s_2 seem to be concurrently executable. The three reachable dead states are different. This shows that the *single spontaneous transition firing rule* where every executable step contains exactly one spontaneous transition, is not sufficient to construct all reachable states of a signal-net system. On the other hand, if we put three tokens to every place of our net we have a situation where the step s can be removed from the list of executable steps without changing the set of reachable states. This faces us with the (implementation) problem of reducing the list of executable steps as far as the diamond property holds.

Our tool SESA computes the static transition conflicts. The computation of (dynamic) step conflicts and dynamic transition conflicts can be done after a reachability graph has been (partly) generated; the conflicts reflected by the computed graph are displayed on demand.

III. Model Checking

11. Computation Tree Logic

Reachability analysis is a method for analyzing the dynamic behavior of a concurrent system described by one of numerous modeling techniques. Creating a reachability graph provides a way to characterize all possible behaviors of the system.

Temporal logics, such as the Computation Tree Logic CTL, offer facilities for the specification of properties that the behavior of the system must fulfill. The process of checking whether a temporal formula holds for a system is called model checking [CES86, CGP99].

In CTL all formulae specify behaviors of the system starting from an assigned state in which the formula is evaluated by taking paths, i.e. sequences of states, into account. A formula holds true for the system if the formula evaluates to true in the initial state of the system. Thereby, witnesses for existence-quantified sub-formulae, and counter examples for all-quantified sub-formulae can be determined and displayed. CTL allows us to use atomic propositions to express properties of certain states.

In this section, you will find a short introduction into the syntax and semantics of CTL (Computational Tree Logic), and a description of the state predicates and atomic state propositions used in SESA. See page 115ff in the appendix for a complete and detailed overview of the CTL input language and the SESA model checker.

11.1. Basic Definitions

The semantics of temporal formulae is defined with respect to a reachability graph. States and paths of this reachability graph are used for the evaluation of truth values.

11.1.1. Reachability graphs

The reachability graph of a system consists of all global states that the system can reach, starting in a given initial state. The basic structure can be seen as a directed graph.

Definition 11.1
Structure M is a *reachability graph*, which is a tuple $M = [Z, E]$, where

1. Z is a finite set of states
2. E is a finite set of transitions between states, i.e. a set of edges (z, z'), such that $z, z' \in Z$ and z' is reachable from z.

Note that E is a binary relation on Z. In the sequel we assume as usual that the reachability graph has no terminal states, i.e. states without a successor; otherwise, for every such state z we would add the edge (z, z) to E. This implies that the relation E is total.

11.1.2. Paths

Paths play the key role in the definition and evaluation of formulae expressed in a CTL like logic.

Definition 11.2
A *path* starting in the state z_0 is a sequence of states $(z_i) = z_0 z_1 \ldots$ such that for all $j \geq 0$ it holds that there is an edge $(z_j, z_{j+1}) \in E$. We use (z_i) to denote such a path.

Note that every path has infinite length due to the requirement that for every terminal state z there is an edge $(z, z) \in E$

11.1.3. State predicates and atomic state propositions

During the computation of minimal paths, the reachability analysis ("bad" predicate), and the model checking SESA works with state predicates.

A predicate is a disjunction of conjunctions of possibly negated atoms (disjunctive normal form). An atom consists of a statement of the form

$$place: \quad low \leq value \leq upp \qquad\qquad (\text{where } 0 \leq low \leq upp \leq \infty)$$

As an example for the state predicates used in SESA, $value$ =number of tokens on a place; an atom $p_1:\ low = 1,\ upp = 2$; is then satisfied by all states in which place p_1 contains one or two tokens.

In SESA you can even work with more general atomic state propositions, e.g. marking sums as in $m(p1) + m(p2) + m(p3) = 1$ or arbitrary boolean combinations of other relations as in $((m(p1) > 1) \wedge (m(p2) \geq 1)) \Rightarrow (m(p3) = 0)$.

In the sequel we use functions P mapping states $z \in Z$ to booleans, to express properties related to certain states.

11.2. Syntax and Semantics of CTL

11.2.1. Syntax

Definition 11.3
The set of *Computation Tree Logic formulae* is defined inductively.

Basis: Every predicate or atomic state proposition P and the constants true and false are CTL formulae.

Step: If φ and ψ are CTL formulae, so are the boolean combinations $\neg\varphi$, $(\varphi \wedge \psi)$, and $(\varphi \vee \psi)$, and the temporal operators $EX\,\varphi$, $AX\,\varphi$ $EF\,\varphi$, $AF\,\varphi$, $EG\,\varphi$, $AG\,\varphi$, $E[\,\varphi\, U\, \psi\,]$, $A[\,\varphi\, U\, \psi\,]$. $E[\,\varphi\, B\, \psi\,]$, and $A[\,\varphi\, B\, \psi\,]$.

11.2.2. Semantics of CTL

The truth value of CTL formulae is evaluated with respect to a certain state of the reachability graph. The semantics of predicates, atomic state propositions and boolean combinations is standard and does not need further explanation.

The relation $M, z_0 \models \varphi$ means that the CTL–formula φ is satisfied in the state z_0 within the given structure M. In the sequel we omit M in the definition of $\models$.

Definition 11.4
Let $z_0 \in Z$ be a state of the reachability graph and φ and ψ CTL formulae. Then the relation $\models$ for CTL formulae is defined inductively.

Basis:	$z_0 \models P$	iff P holds in z_0
	$z_0 \models \mathsf{true}$	always holds
	$z_0 \models \mathsf{false}$	never holds
Step:	$z_0 \models \neg\varphi$	iff not $z_0 \models \varphi$
	$z_0 \models (\varphi \wedge \psi)$	iff $z_0 \models \varphi$ and $z_0 \models \psi$
	$z_0 \models (\varphi \vee \psi)$	iff $z_0 \models \varphi$ or $z_0 \models \psi$
	$z_0 \models EX\,\varphi$	iff there exists a successor state z_1 such that there is an edge $(z_0, z_1) \in E$ and $z_1 \models \varphi$
	$z_0 \models AX\,\varphi$	iff $z_1 \models \varphi$ holds for all successor states z_1 with an edge $(z_0, z_1) \in E$
	$z_0 \models EF\,\varphi$	iff there is a path (z_i) and a $j \geq 0$ such that $z_j \models \varphi$
	$z_0 \models AF\,\varphi$	iff for all paths (z_i) there exists a $j \geq 0$ such that $z_j \models \varphi$
	$z_0 \models EG\,\varphi$	iff there is a path (z_i) and for all $j \geq 0$ it holds $z_j \models \varphi$
	$z_0 \models AG\,\varphi$	iff for all paths (z_i) and for all $j \geq 0$ it holds $z_j \models \varphi$
	$z_0 \models E[\varphi\,U\,\psi]$	iff there is a path (z_i) and a $j \geq 0$ such that $z_j \models \psi$ and for all $0 \leq k < j$ it holds $z_k \models \varphi$
	$z_0 \models A[\varphi\,U\,\psi]$	iff for all paths (z_i) there exists a $j \geq 0$ such that $z_j \models \psi$ and for all $0 \leq k < j$ it holds $z_k \models \varphi$
	$z_0 \models E[\varphi\,B\,\psi]$	iff there is a path (z_i) and a $j \geq 0$ such that $z_j \models \psi$ and there is a $0 \leq k < j$ such that $z_k \models \varphi$ holds
	$z_0 \models A[\varphi\,B\,\psi]$	iff for all paths (z_i) there exists a $j \geq 0$ such that $z_j \models \psi$ and there is a $0 \leq k < j$ such that $z_k \models \varphi$ holds

A formula φ holds true in $M = [Z, E]$ iff the formula is true in the initial state of the reachability graph.

11.2.3. Equivalences

Definition 11.5
Two formulae φ, ψ are said to be *equivalent* ($\varphi \equiv \psi$) if $z \models \varphi$ iff $z \models \psi$ for every reachability graph M and any state z of M.

For CTL we have the following equivalences:

- $AX(\varphi) \equiv \neg EX(\neg\varphi)$
- $AF(\varphi) \equiv A[\mathsf{true}\,U\,\varphi]$
- $EF(\varphi) \equiv E[\mathsf{true}\,U\,\varphi]$
- $AG(\varphi) \equiv \neg EF(\neg\varphi)$
- $EG(\varphi) \equiv \neg AF(\neg\varphi)$

- $A[\varphi\, U\, \psi] \equiv \neg(\, E[\,\neg\psi\, U\, \neg\psi \wedge \neg\varphi\,] \vee EG\,\neg\psi\,)$
- $E[\varphi\, U\, \psi] \equiv \psi \;\vee\; (\,\varphi \;\wedge\; EX\,(\, E[\varphi\, U\, \psi]\,)$
- $A[\varphi\, U\, \psi] \equiv \psi \;\vee\; (\,\varphi \;\wedge\; AX\,(\, A[\varphi\, U\, \psi]\,)$

The semi-formal meaning of the temporal operators is underlined by the symbols used: E stands for "exists", A for "always", X for "next", U for "until", B for "before", F for "future", and G for "globally".

12. Extended Computation Tree Logic

In CTL it is rather complicated to refer to information contained in certain state transitions between states. We try to give a solution for this problem by proposing an extension of CTL which we call eCTL *extend Computation Tree Logic* [Roc00b, Roc00c]. Most of the known equivalences between CTL operators also hold for the extended operators.

First experiences have shown the power and expressiveness of eCTL. With the extended next step operators you can express the need for certain state transitions along a path, e.g. $EF\,E\,t\,X\,EF\,a$ is true, if there is a path leading to a state fulfilling a and along this path t is contained in a step.

If either an existential quantified formula is true giving us a witness path or an universal quantified formula is false giving a counterexample, you can use a transition formula to limit the range of temporal quantifiers to exclude such a path and possible get another witness path or counterexample.

Our logic is only a subset of the action based logic ACTL [DV90], but we have choosen to extend an existing model checker rather than to translate action based formulae together with the underlying structure as it is proposed in [DV90].

12.1. Basic Definitions

We modify the standard definitions by using transition formulae and labeled reachability graphs.

12.1.1. Reachability graphs

Since we want to refer not only to state information but also to steps between states, multiple (labeled) edges between two nodes occur in our basic structure seen as a directed graph.

Definition 12.1
Structure M is a *reachability graph*, which is a tuple $M = [Z, E]$, where

1. Z is a finite set of states
2. E is a finite set of transitions between states, i.e. a set of labeled edges (z, s, z'), such that $z, z' \in Z$ and z' is reachable from z by executing the step s.

For every terminal state z we add the edge $(z, \emptyset, z)$ to E.

12.1.2. Transition formulae

We introduce transition formulae to refer to state transition information contained in the edges of the reachability graph. Since our reachability graph is labeled with steps, we use t to refer to a transition of a step, and τ to denote transition formulae (we do not have silent actions like [DV90]).

Definition 12.2
The set of *transition formulae* is defined inductively.

Basis: Every transition $t \in T$ and the constants true and false are transition formulae.

Step: If τ and ϱ are transition formulae, so are the boolean combinations $\neg\tau$, $(\tau \wedge \varrho)$, and $(\tau \vee \varrho)$.

The semantics of transition formulae is standard and does not need further explanation. The truth value is evaluated with respect to a certain edge of the reachability graph.

Definition 12.3
Let $(z, s, z') \in E$ be a state transition with a step s and τ and ϱ transition formulae. Then the relation $\models$ for transition formulae is defined inductively.

Basis:	$(z, s, z') \models t$	iff $t \in s$
	$(z, s, z') \models \mathsf{true}$	always holds
	$(z, s, z') \models \mathsf{false}$	never holds
Step:	$(z, s, z') \models \neg\tau$	iff not $(z, s, z') \models \tau$
	$(z, s, z') \models (\tau \wedge \varrho)$	iff $(z, s, z') \models \tau$ and $(z, s, z') \models \varrho$
	$(z, s, z') \models (\tau \vee \varrho)$	iff $(z, s, z') \models \tau$ or $(z, s, z') \models \varrho$

12.1.3. Paths and Sequences

We extend the usual path definition by taking transition formulae into account and defining τ-sequences. Note that a τ-sequence in general is not a path in the sense of Definition 11.2.

To obtain a τ-sequence from a path only τ fulfilling state transitions have to be taken into account, i.e. an edge $(z_k, s_k, z_{k+1}) \in E$ is ignored, if τ does not hold in (z_k, s_k, z_{k+1}). If the resulting sequence has finite length due to the ignoring of an edge, then infinitely many repetitions of the last state have to be added.

Definition 12.4
A *τ-sequence* for a transion formula τ is a sequence of states $(z_i) = z_0 z_1 \ldots$ starting in z_0 such that either

for all $j \geq 0$ it holds that there is an edge $(z_j, s_j, z_{j+1}) \in E$ with $(z_j, s_j, z_{j+1}) \models \tau$

(i.e. $z_0 z_1 \ldots$ is a path such that every state transition fulfills τ)

or there is a $k \geq 0$ such that there is for z_k no edge $(z_k, s, z) \in E$ with $(z_k, s, z) \models \tau$ for any state transition s and state z, but for all $0 \leq j < k$ there is an edge $(z_j, s_j, z_{j+1}) \in E$ with $(z_j, s_j, z_{j+1}) \models \tau$, and for all $i > k$ it holds that $z_i = z_k$

(i.e. z_k is the last state in a series of τ fulfilling state transitions and this state is repeated infintely).

We use $(z_i)^\tau$ to denote such a sequence.

Figure 12.1 shows on the left a reachability graph. The sequence $z_0, z_1, z_2, \dots$ is the only path, but $z_0, z_1, z_1, z_1, \dots$ is the only τ-sequence in this reachability graph.

$$z_0 \xrightarrow{\tau} z_1 \xrightarrow{\neg\tau} z_1 \dashrightarrow \qquad\qquad \begin{tikzcd} z_0 \arrow[r] & z_1 \arrow[loop right] \end{tikzcd}$$

Figure 12.1: Reachability graph with path $z_0 z_1 z_2 \dots$ and τ-sequence $z_0 z_1 z_1 z_1 \dots$

12.2. Syntax and Semantics of eCTL

Our logic eCTL is an extension of the Computation Tree Logic CTL. We use transition formulae to limit the range of quantifiers in temporal operators.

12.2.1. Syntax

Definition 12.5
The set of *extended Computation Tree Logic formulae* is defined inductively.

Basis: Every predicate or atomic state proposition P and the constants true and false are eCTL formulae.

Step: If φ and ψ are eCTL formulae, so are the boolean combinations $\neg\varphi$, $(\varphi \wedge \psi)$, and $(\varphi \vee \psi)$, and the temporal operators $E\,\tau\,X\,\varphi$, $A\,\tau\,X\,\varphi$ $E\,\tau\,F\,\varphi$, $A\,\tau\,F\,\varphi$, $E\,\tau\,G\,\varphi$, $A\,\tau\,G\,\varphi$, $E\,\tau[\,\varphi\,U\,\psi\,]$, $A\,\tau\,[\,\varphi\,U\,\psi\,]$, $E\,\tau[\,\varphi\,B\,\psi\,]$, and $A\,\tau\,[\,\varphi\,B\,\psi\,]$, for transition formulae τ.

12.2.2. Semantics

We postpone the interpretation of most temporal operators to Section 12.2.4 and start with the interpretation of the two next state operators $E\tau X\varphi$ and $A\tau X\varphi$.

Definition 12.6
Let $z_0 \in Z$ be a state of the reachability graph, τ a transition formula, and φ and ψ eCTL formulae. Then the relation $\models$ for eCTL formulae is defined inductively (see Definition 11.4 for the basis and boolean combinations).

Step:	$z_0 \models E\,\tau\,X\,\varphi$	iff there exists a successor state z_1 such that there is an edge $(z_0, s, z_1) \in E$ such that $(z_0, s, z_1) \models \tau$ and $z_1 \models \varphi$ holds
	$z_0 \models A\,\tau\,X\,\varphi$	iff $z_1 \models \varphi$ holds for all successor states z_1 with an edge $(z_0, s, z_1) \in E$ such that $(z_0, s, z_1) \models \tau$ holds

A formula φ holds true in $M = [Z, E]$ iff the formula is true in the initial state of the reachability graph.

Figure 12.2 shows three reachability graphs and gives examples for $A\,\tau\,X\,\varphi$ and $E\,\tau\,X\,\varphi$. The edges are labeled with executable steps. z_0 is the initial state. Note that the truth value of φ in state z_3 is not relevant, because this state is ignored due to the transition formula t_1.

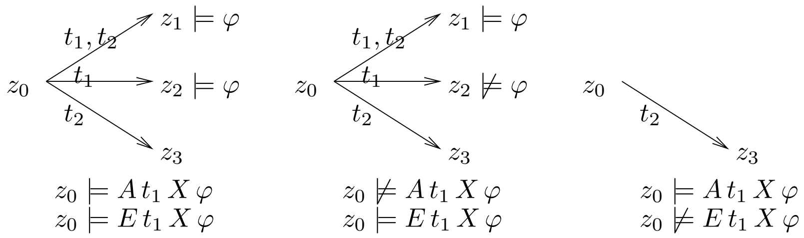


Figure 12.2: $A\,\tau\,X\,\varphi$ and $E\,\tau\,X\,\varphi$

First we want to investigate some properties of the next state operators.

Lemma 12.7

$$A\,\tau\;X\;\varphi \quad\equiv\quad \neg\,E\,\tau\,X\,\neg\varphi \tag{1}$$

As a consequence of the equivalence (1) we can omit the semantic definition of $A\tau\,X\,\varphi$ and derive this operator by setting (1) as an abbreviation.

The relation between the next state operators of CTL and eCTL is examined in the following proposition and lemma.

Proposition 12.8
The next step operators $EX\varphi$ and $AX\varphi$ of CTL can be derived by setting $\tau \equiv \mathsf{true}$.

Lemma 12.9
For every transition formula τ it holds

$$z \models E\tau\,X\,\varphi \quad\Rightarrow\quad z \models EX\varphi \tag{2}$$

$$z \models AX\varphi \quad\Rightarrow\quad z \models A\tau\,X\,\varphi \tag{3}$$

The reverse directions do not hold, e.g. if z_0 has a successor z_1 such that $z_1 \models \varphi$, i.e. $z_0 \models EX\varphi$, then there may be no such z_1 with $(z_0, s, z_1) \models \tau$.

12.2.3. Expressiveness

Figure 12.3 shows two reachability graphs that can be distinguished by the eCTL formula $A\,t\,X\,\varphi$. This is not possible in CTL, because both graphs are identical, if the state transition information is ignored.

Remark. Basically $E\,\tau\,X\varphi$ resp. $A\,\tau\,X\,\varphi$ are equivalent to $\langle\tau\rangle\varphi$ resp. $[\tau]\varphi$ of the modal μ-calculus [Koz83]. Nevertheless eCTL does not contain the fixpoint operators of the

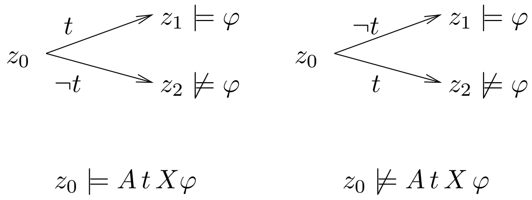


Figure 12.3: Two distinguishable reachability graphs

modal μ-calculus and has therefore not the same expressive power as the modal μ-calculus.

It is known that eCTL-like logics can be decided by transformation of formula and underlying structure and using standard CTL procedures [DV90].

12.2.4. Semantics of remaining eCTL temporal operators

The semantics is given like standard CTL but instead of a path we take τ-sequences into account.

Definition 12.10 (cont. of Definition 12.6)

$z_0 \models E\,\tau\,F\,\varphi$	iff there is a τ-sequence $(z_i)^\tau$ and a $j \geq 0$ such that $z_j \models \varphi$
$z_0 \models A\,\tau\,F\,\varphi$	iff for all τ-sequences $(z_i)^\tau$ there is a $j \geq 0$ such that $z_j \models \varphi$
$z_0 \models E\,\tau\,G\,\varphi$	iff there is a τ-sequence $(z_i)^\tau$ such that for all $j \geq 0$ it holds $z_j \models \varphi$
$z_0 \models A\,\tau\,G\,\varphi$	iff for all τ-sequences $(z_i)^\tau$ and for all $j \geq 0$ it holds $z_j \models \varphi$
$z_0 \models E\,\tau\,[\varphi\,U\,\psi]$	iff there is a τ-sequence $(z_i)^\tau$ and a $j \geq 0$ such that $z_j \models \psi$ and for all $0 \leq k < j$ it holds $z_k \models \varphi$
$z_0 \models A\,\tau\,[\varphi\,U\,\psi]$	iff for all τ-sequences $(z_i)^\tau$ there exists a $j \geq 0$ such that $z_j \models \psi$ and for all $0 \leq k < j$ it holds $z_k \models \varphi$

Lemma 12.11

$$
\begin{aligned}
A\,\tau\,G\,\varphi &\equiv \neg\,E\,\tau\,F\,\neg\varphi && (4)\\
A\,\tau\,F\,\varphi &\equiv \neg\,E\,\tau\,G\,\neg\varphi && (5)\\
&\equiv A\,\tau\,[\mathsf{true}\,U\,\varphi] && (6)\\
E\,\tau\,F\,\varphi &\equiv E\,\tau\,[\mathsf{true}\,U\,\varphi] && (7)\\
A\,\tau\,[\varphi\,U\,\psi] &\equiv \neg(\,E\,\tau\,[\,\neg\psi\,U\,\neg\psi \wedge \neg\varphi\,] \vee E\,\tau\,G\,\neg\psi\,) && (8)
\end{aligned}
$$

Equally valid formulations of eCTL are possible by taking only some operators as fundamental and deriving all other operators using the above equalities. One sufficient set is $E\,\tau\,X\,\varphi$, $E\,\tau\,G\,\varphi$, and $E\,\tau\,[\varphi\,U\,\psi]$, another set is $E\,\tau\,X\,\varphi$, $E\,\tau\,[\varphi\,U\,\psi]$, and $A\,\tau\,[\varphi\,U\,\psi]$. We can use this in proofs.

Lemma 12.12

$$E\,\tau\,[\varphi\,U\,\psi] \;\equiv\; \psi \,\vee\, (\,\varphi \,\wedge\, E\,\tau\,X\,(\,E\,\tau\,[\varphi\,U\,\psi]\,) \,\wedge\, E\,\tau\,X\,\mathsf{true}\,) \tag{9}$$

$$A\,\tau\,[\varphi\,U\,\psi] \;\equiv\; \psi \,\vee\, (\,\varphi \,\wedge\, A\,\tau\,X\,(\,A\,\tau\,[\varphi\,U\,\psi]\,) \,\wedge\, E\,\tau\,X\,\mathsf{true}\,) \tag{10}$$

Proof. (9) Case 1: $z_0 \models \psi$. Then $E\,\tau\,[\varphi\,U\,\psi]$ holds trivially in z_0. Case 2: $z_0 \not\models \psi$. Case 2a: $z_0 \not\models \varphi$. Then $E\,\tau\,[\varphi\,U\,\psi]$ does not hold in z_0. Case 2b: $z_0 \models \varphi$. If z_0 has no successor z_1 such that $(z_0, s, z_1) \models \tau$ then $E\,\tau\,[\varphi\,U\,\psi]$ does not hold in z_0. Otherwise $E\,\tau\,[\varphi\,U\,\psi]$ does only hold, iff $z_1 \models\; E\,\tau\,[\varphi\,U\,\psi]$.
(10) similar. □

Proposition 12.13
The temporal operators of CTL, i.e. $EF\varphi$, $AF\varphi$, $EG\varphi$, $AG\varphi$, $E[\varphi U\psi]$, and $A[\varphi U\psi]$, can be derived by setting $\tau \equiv \mathsf{true}$.

Corollary 12.14
CTL is a subset of eCTL.

Lemma 12.15

$$E\,\tau\,[\varphi\,U\,\psi] \;\Rightarrow\; E\,[\varphi\,U\,\psi] \tag{11}$$

$$E\,\tau\,F\,\varphi \;\Rightarrow\; EF\varphi \tag{12}$$

$$AG\varphi \;\Rightarrow\; A\,\tau\,G\,\varphi \tag{13}$$

The reverse directions and other implications between CTL and eCTL temporal operators do not hold, e.g. if $z_0 \models E\,\tau\,G\,\varphi$ then the τ-sequence may be shorter than a true-sequence.

12.3. Example eCTL formulae

$E\,\tau\,F\,\varphi$ can be used to express the reachability of a state fulfilling φ by a sequence containing only state transitions, in which τ holds true. In Figure 12.4 $z_0 \models E\,\tau\,F\,\varphi$ holds true because there exists a τ-sequence z_0, z_3, z_7 such that $z_7 \models \varphi$. The sequence to the state z_4 with $z_4 \models \varphi$ is not valid for this, because this sequence is not a τ-sequence. Note that $A\,\tau\,F\,\varphi$ holds true in z_0 as well, since z_0, z_3, z_7 is the only τ-sequence staring in z_0 in this reachability graph.
To express that whenever a state transition fulfilling t is possible this transition should lead to a successor state in which φ holds true, $AG\,A\,t\,X\,\varphi$ can be used. In Figure 12.5 only z_2, z_4, and z_6 have to fulfill the sub formula φ. $A\,t\,X\,\varphi$ holds trivially in all other states, because none of them has a τ successor.
The need for an acknowledgment of certain requests is specified by the formula

$$AG\;A\,t_{\mathsf{req}}\,X\;AF\;E\,t_{\mathsf{ackn}}\,X\,\mathsf{true}\,.$$

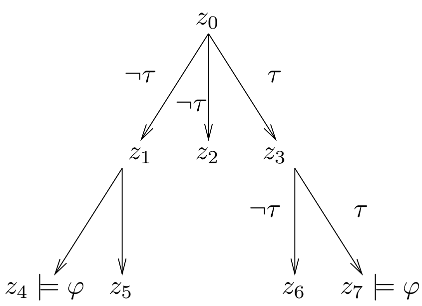


Figure 12.4: $z_0 \models E\,\tau\,F\,\varphi$

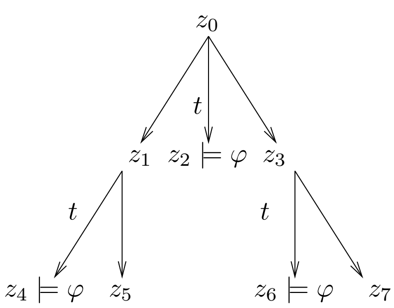


Figure 12.5: $z_0 \models AG\,A\,t\,X\,\varphi$

Figure 12.6 shows a reachability graph, in which this formula is true. Note that only z_3 and z_4 have to fulfill $AF\, E\, t_{\mathsf{ackn}}\, X\, \mathsf{true}$.

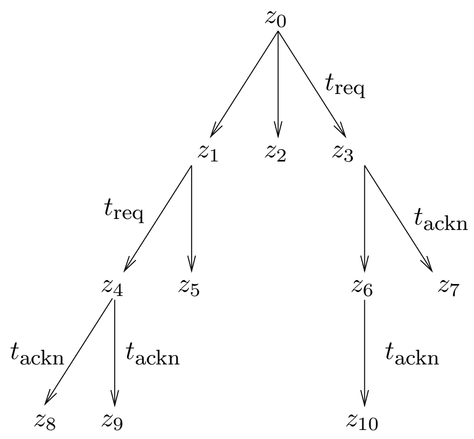


Figure 12.6: $z_0 \models AG\, A\, t_{\mathsf{req}}\, X\, AF\, E\, t_{\mathsf{ackn}}\, X\, \mathsf{true}$

eCTL is a conservative extension of CTL in the sense that every eCTL formula without any transition formula in it (τ always true) is interpreted in the same way as in plain CTL.

12.4. Implementation

This section sketches, how our extension can be embedded in existing model checking algorithms based on graph traversal.

These model checkers evaluate the truth value of a CTL formula by successively traversing the reachability graph starting in a certain state [CES86, Hel97]. Only the evaluation of atomic propositions, boolean combinations, and the temporal operators $A\,\tau\, X\,\varphi$, $E\,\tau\,[\varphi\, U\,\psi]$, and $A\,\tau\,[\varphi\, U\,\psi]$ have to be implemented, since all other eCTL operators can be derived. We use the local model checking algorithm described in [Hel97, Section 4].

A certain transition formula only limits the range of its temporal operator. Therefore nothing but a selection of successor states have to be implemented: We just modify the `successors(z)` routine [Hel97, Fig. 25-29 and 37-43] to return only successor states, for which there is an edge fulfilling the current transition formula. This can be done by passing τ_i to the `successors` routine.

Special care has to be taken to handle the semantic of the next step operator $A\,\tau\, X\,\varphi$. Two cases have to be distinguished, if the current state z has no τ successors: If z has at least one successor, i.e. $(z, s, z') \in E$, but $(z, s, z') \not\models \tau$, then $z \models A\,\tau\, X\,\varphi$ holds independently of φ, because there is no τ successor. If otherwise z is a terminal state, i.e. $(z, \emptyset, z)$ is the only state transition from z, then $z \models A\,\tau\, X\,\varphi$ holds if either $(z, \emptyset, z) \not\models \tau$, i.e. τ does not hold for the empty state transition, or $z \models \varphi$. This only affects the implementation of ax in [Hel97, Fig. 29]

Another model checking approach is based on fixpoint equations [BCM92, CGP99]. The equalities of Lemma 12.12 are a basis for the characterization of eCTL formulae in form of fixpoint equations. These fixpoints can be computed, giving as a result all states, for which a certain formula holds true. Since large sets have to be represented and manipulated, binary decision diagrams (BDDs) are mostly used for this [BCM92]. In a BDD based model checker $E\,\tau\,X\,\varphi$ plays a key role, because all other operators can be derived by negation and fixpoint computation. To evaluate $E\,\tau\,X\,\varphi$, only the representation of the state transition relation (coded as a BDD) must be changed to respect τ. We have implemented an eCTL model checker based on [Hel97] in our tool SESA.

13. Timed Computation Tree Logic

This section shows an extension of the *Computation Tree Logic* CTL that allows the melding of qualitative temporal assertions together with time constraints. The extension essentially consists in attaching a time bound to the modalities. A good survey can be found in [AH92], we were influenced esp. from [EMSS91].

We use intervals $[l, h]$ with $0 \le l \le h \le \omega$ as time constraints, but attach them only to the modalities X, F and U. Hence, a formula from *Timed Computation Tree Logic* TCTL is obtained from a CTL-formula by attaching intervals to some of these modalities. We evaluate formulae over discrete-time reachability graphs of arc-timed *SNS*. If $EX\varphi$ is a formula of CTL then $EX_{[l,h]}\varphi$ is a formula of TCTL which is satisfied by a state z if this state has a successor z' satisfying the formula φ and such that the state transition from z to z' takes at least l and at most h time units.

13.1. Basic Definitions

We define the semantics of TCTL based on the structure of a reachability graph of an arc-timed *SNS*.

Definition 13.1
For a reachability graph $[Z, E]$ we define the *state delay* D as a mapping $D : Z \to \mathbb{N}_0$.

For any state $z = [m, u]$ the number $D(z)$ is the number of time units which have to elaps at z before a step can be executed.

Definition 13.2
For any path (z_i) and any state $z \in Z$ we put

1. $D[(z_i), z] = 0\,,$ if $z_0 = z$
2. $D[(z_i), z] = D(z_0) + D(z_1) + \cdots + D(z_{k-1})\,,$ if $z_k = z$ and $z_0, \ldots, z_{k-1} \neq z$

With other words, $D[(z_i), z]$ is the number of time units after which the state z on the path (z_i) is reached the first time, i.e. the minimal time distance from z_0.

13.2. Syntax and Semantics of TCTL

The syntax of TCTL is like CTL, except the attachment of intervals $[l, h]$ with $0 \le l \le h \le \omega$ to the modalities X, F and U. The semantics for these temporal operators have to take delays into account.

13.2.1. Semantics

Propositions, boolean combinations, and the temporal operators AG, EG, AB, and EB have standard interpretation and are omitted in the following.

Definition 13.3
Let $z_0 \in Z$ be a state of the reachability graph and φ and ψ TCTL formulae. Then the relation $\models$ for TCTL formulae is defined as follows.

$z_0 \models EX_{[l,h]}\,\varphi$	iff there exists a successor state z_1 such that there is an edge $(z_0, z_1) \in E$ and $l \leq D(z_0) \leq h$ and $z_1 \models \varphi$ holds
$z_0 \models AX_{[l,h]}\,\varphi$	iff $z_1 \models \varphi$ holds for all successor states z_1 with an edge $(z_0, z_1) \in E$ and $l \leq D(z_0) \leq h$ holds
$z_0 \models EF_{[l,h]}\,\varphi$	iff there is a path (z_i) and a $j > 0$ such that $z_j \models \varphi$ and $l \leq D((z_i), z_j) \leq h$
$z_0 \models AF_{[l,h]}\,\varphi$	iff for all paths (z_i) there is a $j > 0$ such that $z_j \models \varphi$ and $l \leq D((z_i), z_j) \leq h$
$z_0 \models E[\varphi\, U_{[l,h]}\, \psi]$	iff there exists a path (z_i) and a $j > 0$ such that $z_j \models \psi$, $l \leq D((z_i), z_j) \leq h$ and for all $0 \leq k < j$ it holds $z_k \models \varphi$)
$z_0 \models A[\varphi\, U_{[l,h]}\, \psi]$	iff for all paths (z_i) there is a $j > 0$ such that $z_j \models \psi$, $l \leq D((z_i), z_j) \leq h$ and for all $0 \leq k < j$ it holds $z_k \models \varphi$)

13.2.2. Equivalences

- $z \models EX_{[l,h]}\,\mathsf{true}$ iff $l \leq D(z) \leq h$ iff $z \models AX_{[l,h]}\mathsf{true}$,
- $E[\mathsf{true}\, U_{[l,h]}\, \varphi] \equiv EF_{[l,h]}\, \varphi$,
- $A[\mathsf{true}\, U_{[l,h]}\, \varphi] \equiv AF_{[l,h]}\, \varphi$,
- $EX_{[0,\omega]}\, \varphi \equiv EX\varphi$,
- $AX_{[0,\omega]}\, \varphi \equiv AX\varphi$,
- $EF_{[0,\omega]}\, \varphi \equiv EF\varphi$,
- $AF_{[0,\omega]}\, \varphi \equiv AF\varphi$,
- $E[\varphi\, U_{[0,\omega]}\, \psi] \equiv E[\varphi\, U\, \psi]$,
- $A[\varphi\, U_{[0,\omega]}\, \psi] \equiv A[\varphi\, U\, \psi]$.

13.3. Example TCTL formulae

- A state z is called a time-deadlock iff all states z' which are reachable from z have the delay $D(z') = 0$. This is expressed by

$$z \models AGEX_{[0,0]}\mathsf{true} \text{ or } z \models \neg EF_{[1,\omega]}\mathsf{true}.$$

- Let φ be a formula which is satisfied exactly by the state z. From the state z_0 in any case the state z is reached after at most d units of time: $z_0 \models AF_{[0,d]}\varphi$.
- Let φ be a formula which is satified exactly for states $z = [m, u]$ with $m(p_1) = 3$ and let ψ be satisfied iff $m(p_2) = 4$. Then

  $$z \models \neg EF(\varphi \ \wedge \ \neg AF_{[0,d]}\psi)$$

  holds from any state z where p_1 has three tokens within at most d units of time a state is reached where p_2 has four tokens.

IV. Structural Properties

14. Static Deadlocks and Traps

One of the very few possibilities to prove liveness properties of an unbounded Petri net is provided by the Commoner-Theorem: An ordinary free-choice Petri net is live iff it has the Deadlock-Trap property. In this section we try to find definitions for a corresponding property of *SNS*. For simplicity we confine ourselves to *ordinary* signal-net systems (i.e. all multiplicities are equal to 1). Let us first consider traps.

The essential property of a *trap* is that a trap is not able to become clean after being marked.

Definition 14.1

1. A subset $Q \subseteq P$ is said to be a *dynamic trap at* m^* iff
$$\forall m \forall s \forall m'(m^* \xrightarrow{*} m \xrightarrow{s} m' \wedge m(Q) > 0 \Rightarrow m'(Q) > 0).$$
2. A subset $Q \subseteq P$ is said to be a *strongly dynamic trap at* m^* iff
$$\forall m \forall s(m^* \xrightarrow{*} m \xrightarrow{s} \wedge s^-(Q) > 0 \Rightarrow s^+(Q) > 0).$$

Lemma 14.2

1. *Any strongly dynamic trap at* m^* *is a dynamic trap at* m^*.
2. *If* Q *is a dynamic trap at* m^*, $m^*(Q) > 0$ *and* $m^* \xrightarrow{*} m'$ *then* $m'(Q) > 0$.
3. *If* Q *is a (strongly) dynamic trap at* m^* *and* $m^* \xrightarrow{*} m^{**}$, *then* Q *is a (strongly) dynamic trap at* m^{**}.

The converse of Lemma 14.2.2 is (even for Petri nets) not true. In Figure 14.1 the place set P is a dynamic trap at the marking $(1, 2)$ (since the zero marking is not reachable), but P is not a strongly dynamic trap at $(1, 2)$ because the step $s = \{t\}$ is executable at $(1, 2)$. Nevertheless, P is a strongly dynamic trap at $(0, 1)$, hence, the converses of 14.2.1 and 3 do not hold. In Figure 14.2 the set $Q = \{p_1, p_2\}$ is a strongly dynamic trap at $(1, 1, 0)$, and, $(1, 0, 1) \xrightarrow{\{t_3\}} (1, 1, 0)$, but Q is not a dynamic trap at the marking $(1, 0, 1)$.

Now, we try to find structural properties of place sets Q which imply that Q is a (strongly) dynamic trap.

Definition 14.3
Let $Poss(N) := \{s \mid \emptyset \neq s \subseteq T \wedge \exists m\ m \xrightarrow{s}\}$ be the set of all *possible* steps of N. Then $Q \subseteq P$ is said to be a *structural trap* of N iff
$$\forall s(s \in Poss(N) \wedge s \cap QF \neq \emptyset \Rightarrow s \cap FQ \neq \emptyset).$$

Lemma 14.4
Every structural trap is a strongly dynamic trap at m_0.

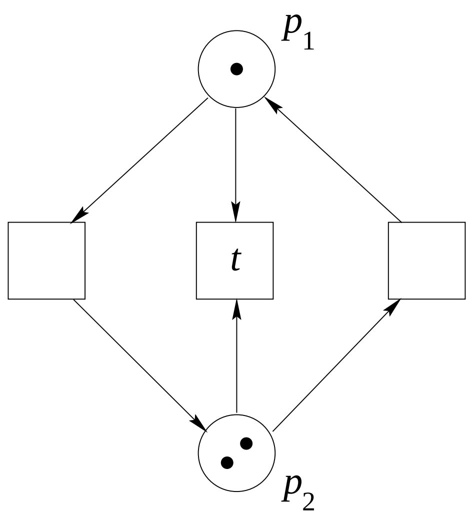


Figure 14.1: Counterexample to the converse of Lemma 14.2.1 and 14.2.3

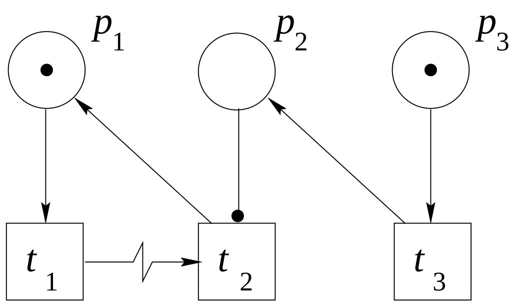


Figure 14.2: Counterexample to the converse of Lemma 14.2.1 and 14.4

Proof. Let $m_0 \xrightarrow{*} m \xrightarrow{s}$ and $s^-(Q) > 0$. Then $s \in Poss(N)$. By $s^-(Q) > 0$ we have $s \cap QF \neq \emptyset$. Since Q is a structural trap it holds $s \cap FQ \neq \emptyset$, i.e. $s^+(Q) > 0$. □

The converse does not hold: In Figure 14.2, the set $Q = \{p_1, p_2\}$ is a strongly dynamic trap at $(1, 1, 0)$ which is not a structural trap since $s = \{t_1\} \in Poss(N)$, but $s \cap FQ = \emptyset$.

Remark. If N is a Petri net then a set Q is a structural trap iff

$$QF \subseteq FQ.$$

Now, we are going to consider (static) *deadlocks*. The essential property of a deadlock is that a clean deadlock never can become marked.

Definition 14.5
Let $\emptyset \neq D \subseteq P$ and $m^* \in R_N(m_0)$.

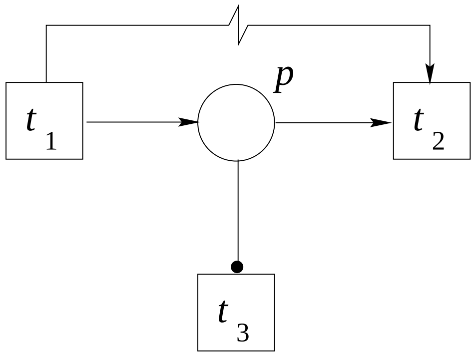


Figure 14.3: Counterexample to the converse of Lemma 14.4 and 14.6.3

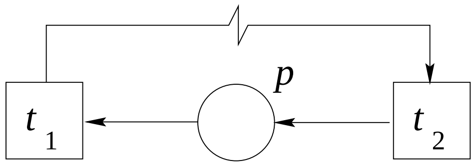


Figure 14.4: Counterexample to the converse of Lemma 14.4

1. D is a *dynamic deadlock at* m^* iff

$$\forall m \forall s \forall m' (m^* \xrightarrow{*} m \xrightarrow{s} m' \wedge m(D) = 0 \quad \Rightarrow \quad m'(D) = 0).$$

2. D is a *strongly dynamic deadlock at* m^* iff

$$\forall m \forall s (m^* \xrightarrow{*} m \xrightarrow{s} \wedge s^+(D) > 0 \quad \Rightarrow \quad s^-(D) > 0 \ \vee \ \widehat{s}(D) > 0).$$

Lemma 14.6

1. *Every strongly dynamic deadlock at* m^* *is a dynamic deadlock at* m^*.

2. *If* D *is a dynamic deadlock at* m^*, $m^*(D) = 0$ *and* $m^* \xrightarrow{*} m'$, *then* $m'(D) = 0$.

3. *If* D *is a (strongly) dynamic deadlock at* m^* *and* $m^* \xrightarrow{*} m^{**}$, *then* D *is a (strongly) dynamic deadlock at* m^{**}.

Proof. Let D be a strongly dynamic deadlock at m^* and $m^* \xrightarrow{*} m \xrightarrow{s} m'$, and $m(D) = 0$. Assume that $m'(D) > 0$. Then $s^+(D) > 0$, hence $s^-(D) > 0$ or $\widehat{s}(D) > 0$ which by $m \xrightarrow{s}$ is in contradiction with $m(D) = 0$. Hence, D is a dynamic deadlock at m^*. The remaining assertions are obvious. □

As we can see by Figure 14.5, the converse of 14.6.1 is not true: $D = \{p\}$ is a dynamic deadlock at m_0 because a marking m with $m(p) = 0$ is not reachable. Obviously, D is

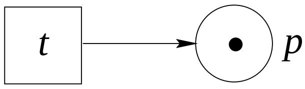


Figure 14.5: Counterexample to the converse of Lemma 14.6.1

not a strongly dynamic deadlock at m_0. D is not a dynamic deadlock at $m = (0)$, but D is a dynamic deadlock at $m' = (1)$, which is reachable from (0). In Figure 14.3, $\{p\}$ is a strongly dynamic deadlock at $m = (1)$, but not a strongly dynamic deadlock at (0): the converse of 14.6.3 does not hold as well.

Definition 14.7
A nonempty subset $D \subseteq P$ is said to be a *structural deadlock* iff

$$\forall s(s \in Poss(N) \ \wedge \ s \cap FD \neq \emptyset \ \Rightarrow \ s \ \cap (DF \cup DB) \neq \emptyset).$$

Lemma 14.8
Every structural deadlock is a strongly dynamic deadlock at m_0.

Proof. Let D be a structural deadlock, $m_0 \xrightarrow{*} m \xrightarrow{s}$ and $s^+(D) > 0$. From $m \xrightarrow{s}$ we obtain $s \in Poss(N)$; by $s^+(D) > 0$ we have $s \cap FD \neq \emptyset$. Therefore, $s \cap (DF \cup DB) \neq \emptyset$, i.e. $s^-(D) > 0$ or $\widehat{s}(D) > 0$. □

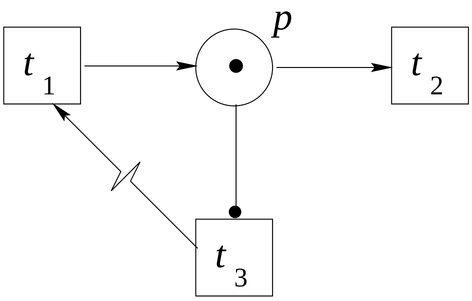


Figure 14.6: Counterexample to the converse of Lemma 14.8

The converse is not true. In Figure 14.3, $\{p\}$ is a strongly dynamic deadlock at $m = (1)$, but not a structural deadlock.

Lemma 14.9
If D is a dynamic deadlock at m, $m(D) = 0$, and $t \in DF$, then t is dead at m.

Proof. Assume, that t is not dead at m, then from m a marking m' is reachable such that t is an element of an executable step s at m'. By $t \in DF$ we obtain $m'(D) \geq s^-(D) \geq t^-(D) > 0$, contradicting the fact that D is a dynamic deadlock at m. □

Lemma 14.10
If a signal-net system N has no structural deadlock then it is deadlock-free, i.e. no dead marking is reachable in N.

Proof. We show, that a structural deadlock D exists, if a dead marking m^* is reachable in N. We consider the set

$$D \quad := \quad \{p \mid \; p \in P \;\wedge\; m^*(p) = 0\}.$$

The set D is not empty since otherwise $m^* \geq 1$, i.e. a spontaneous transition would be enabled (N is assumed to be ordinary). This contradicts that m^* is dead.

To show that D is a structural deadlock, let $s \in \mathit{Poss}(N)$ and $s \cap FD \neq \emptyset$. Since m^* is dead, s is not executable at m^*. By $s \in \mathit{Poss}(N)$ we obtain that s is not enabled at m^* or s is not maximal at m^*, but the latter would contradict that m^* is dead. Hence, we have $s^- \not\leq m^*$ or $\widehat{s} \not\leq m^*$. In the first case there exists a transition $t \in s$ and a place p such that

$$1 \geq t^-(p) > m^*(p) \geq 0,$$

hence, $m^*(p) = 0$ and therefore, $p \in D$, which implies $s \cap DF \neq \emptyset$. In the second case there exists a place p such that

$$1 \geq \widehat{s}(p) > m^*(p) \geq 0,$$

which implies $s \cap DB \neq \emptyset$. □

Definition 14.11
A *SNS* has the *deadlock-trap property* (*DTP* for short) iff every structural deadlock contains a structural trap which is marked at the initial marking.

Theorem 14.12 (cf. [SLH98])
Let N be an ordinary SNS (i.e. all multiplicities of arcs or conditions equal 1). If N has the DTP then N is deadlock-free.

Proof. We assume, that the dead marking m^* is reachable in N. From the proof of Lemma 14.10 we obtain that

$$D \quad := \quad \{p \mid \; p \in P \;\wedge\; m^*(p) = 0\}.$$

is a structural deadlock. By the deadlock-trap property D contains a structural trap $Q \subseteq D$ which is marked under m_0. By Lemma 14.4 Q is a strongly dynamic trap at m_0. From $m_0(Q) > 0$ by Lemma 14.2.2 we have $m^*(Q) > 0$, contradicting $m^*(Q) = 0$. □

15. Free Choice and Extended Simple Properties

In this section we define the EFC- and ES-properties and show their consequences for liveness properties. We consider here only *ordinary SNS* (where all multiplicities equal 1).

Definition 15.1
Let N be an *SNS*. For all $p \in P$ we put

$$Post(p) \quad := \quad pF \cup pB,$$

and for all $t \in T$

$$Pred(t) \quad := \quad St \cup Ft \cup Bt.$$

1. N is said to be *extended free choice* (EFC for short) iff for all transitions $t_1, t_2 \in T$ it holds:

$$Ft_1 \cap Pred(t_2) \neq \emptyset \quad \Rightarrow \quad Pred(t_1) = Pred(t_2) \ \wedge \ M(t_1) = M(t_2).$$

2. N is said to be *extended simple* (ES for short) iff for all $p, q \in P$ it holds:

$$Post(p) \cap Post(q) \neq \emptyset \quad \Rightarrow \quad Post(p) \subseteq Post(q) \ \vee \ Post(q) \subseteq Post(p).$$

Obviously, for *SNS* without condition and signal arcs these definitions coincide with the definitions known for Petri nets.

Corollary 15.2
For every transition t of an extended simple SNS there exists an enumeration of $Pred(t) \cap P = \{p_1, p_2, \dots, p_n\}$ such that

$$Post(p_1) \subseteq Post(p_2) \subseteq \ldots \subseteq Post(p_n).$$

Lemma 15.3
Every extended free choice SNS N such that for every place $p \in P$ the set pF is not empty is extended simple.

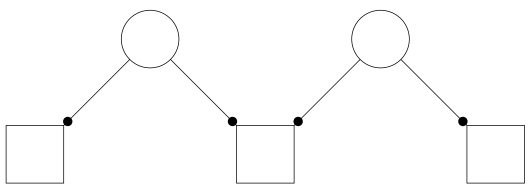


Figure 15.1: Extended free choice *SNS* which is not extended simple

Proof. We shall show that $Post(p) \cap Post(q) \neq \emptyset$ implies $Post(p) = Post(q)$. Let $t \in Post(p) \cap Post(q)$, $t^* \in pF$ and $t_1 \in Post(p)$. From $p \in Ft^*$ and $p \in Pred(t)$ by EFC we obtain $Pred(t^*) = Pred(t)$, hence $q \in Pred(t^*)$. From $p \in Ft^*$ and $p \in Pred(t_1)$ in the same way we obtain $Pred(t^*) = Pred(t_1)$, hence $q \in Pred(t_1)$, i.e. $t_1 \in Post(q)$. □

The converse of Lemma 15.3 does not hold even for Petri nets.

Definition 15.4
Let m be a marking.

1. A transition t is *dead at m* iff there is no marking m' reachable from m, such that t is an element of an executable step s at m'. We put

$$deadtr(m) \quad := \quad \{t \mid t \text{ is dead at } m\}.$$

2. A transition t is *live at m* iff there is no marking m' reachable from m, such that t is dead at m'.

3. The *SNS* N is said to be *live* iff all its transitions are live at the initial marking.

4. A place p is called *dead at m* iff $m'(p) = 0$ for all markings m' reachable from m. We put

$$deadpl(m) \quad := \quad \{p \mid p \text{ is dead at } m\}.$$

5. A place p is called *live at m* iff there is no marking m' reachable from m such that p is dead at m.

6. The *SNS* N is said to be *place-live* iff all its places are live at the initial marking.

Obviously, it holds

$$m' \in R_N(m) \quad \Rightarrow \quad deadtr(m) \subseteq deadtr(m') \wedge deadpl(m) \subseteq deadpl(m').$$

Definition 15.5
A marking m is said to be *maximal-dead* iff

$$m' \in R_N(m) \quad \Rightarrow \quad deadtr(m) = deadtr(m') \wedge deadpl(m) = deadpl(m').$$

Note, that any live marking m is maximal-dead with $deadtr(m) = \emptyset$. At a maximal-dead marking every transition (place) is either live or dead.

Lemma 15.6
Let N be an ordinary extended free choice SNS, m^* a maximal-dead marking and $t_0 \in deadtr(m^*)$ a dead transition. Then there exists a marking m^{**} which is reachable from m^* and such that every place p from $Pred(t_0)$ which is not dead at m^* is marked under m^{**}.

Proof. Let $Q_0 := \{p \mid p \in \mathit{Pred}(t_0) \ \wedge\ p \notin \mathit{deadpl}(m^*) \ \wedge\ m^*(p) = 0\}$. If Q_0 is empty we put $m^{**} := m^*$ and we are ready. Otherwise, select a place p_0 from Q_0. The place p_0 is live but clean at m^*. Therefore there exist steps $s_1, \dots, s_k$ and markings $m_1, \dots, m_k$ such that

$$m^* \xrightarrow{s_1} m_1 \xrightarrow{s_2} m_2 \dots m_{k-1} \xrightarrow{s_k} m_k$$

and $m_k(p_0) > 0$.

During this state transition no token has been removed from a place from Q_0: if a transition t from the step s_j takes a token from Q_0 then $Ft \cap \mathit{Pred}(t_0) \neq \emptyset$, hence by the EFC-property we obtain

$$\mathit{Pred}(t) = \mathit{Pred}(t_0) \quad \wedge \quad M(t) = M(t_0)$$

which implies that t_0 can be fired at m_{j-1} contradicting that t_0 is dead at m^*. This is seen easily if one reminds the construction of the step s_j at m_{j-1} – at the same stage when we include t we can include t_0 instead.

Hence we have $m_k(p) \geq m^*(p)$ for $p \in Q_0$. Let

$$Q_1 \quad := \quad \{p \mid p \in \mathit{Pred}(t_0) \ \wedge\ p \notin \mathit{deadpl}(m^*) \ \wedge\ m_k(p) = 0\}.$$

If Q_1 is empty we put $m^{**} := m_k$ and we are ready, otherwise Q_1 is a proper subset of Q_0 containing only places which are live at m^*, hence at m_k. We select a place p_1 from Q_1 and proceed in the same way, since Q_0 is finite, the construction terminates. □

Theorem 15.7

Let N be an ordinary extended free choice SNS. If N is place-live then any spontaneous transition of N is live at the initial marking.

Proof. For contradiction we assume that t_0 is a spontaneous transition which is not live at m_0. Then there exists a maximal-dead marking m^* such that $t_0 \in \mathit{deadtr}(m^*)$. Since N is place-live we have $\mathit{deadpl}(m^*) = \emptyset$. Since t_0 is spontaneous we have $St_0 = \emptyset$, $\mathit{Pred}(t_0) \subseteq P$. Because t_0 is dead at m^* the set $Q = \{p \mid p \in \mathit{Pred}(t_0) \ \wedge\ m^*(p) = 0\}$ is not empty. By Lemma 15.6 from m^* we can reach a marking m^{**} such that all places from $\mathit{Pred}(t_0)$ are marked, hence t_0 is enabled at m^{**} contradicting that t_0 is dead at m^*. □

Theorem 15.8

Let N be an ordinary extended simple SNS and let m^* be a maximal-dead marking. For every spontaneous transition $t^* \in \mathit{deadtr}(m^*)$, there exists a place $p \in \mathit{Pred}(t^*)$ which is dead at m^*.

Proof. Let $t^* \in \mathit{Spont} \cap \mathit{deadtr}(m^*)$. By $St^* = \emptyset$ we have $\mathit{Pred}(t^*) \cap P \neq \emptyset$ (otherwise t^* would be live); corresponding to Corollary 15.2 we have $\mathit{Pred}(t^*) = \{p_1, \dots, p_n\}$,

$$\mathit{Post}(p_1) \subseteq \mathit{Post}(p_2) \subseteq \ldots \subseteq \mathit{Post}(p_n).$$

For any $m \in R_N(m^*)$, t^* is dead at m, hence, there is a $p \in \mathit{Pred}(t^*)$ which is not marked, since N is ordinary. Let

$$i[m] \quad := \quad \min\{j \mid 1 \leq j \leq n \wedge m(p_j) = 0\}.$$

We show

$$m \in R_N(m^*) \quad \Rightarrow \quad \mathit{Post}(p_{i[m]}) \subseteq \mathit{deadtr}(m) = \mathit{deadtr}(m^*) \wedge m^*(p_{i[m]}) = 0.$$

Consider an arbitrary $t \in \mathit{Post}(p_{i[m]}) \subseteq \ldots \subseteq \mathit{Post}(p_n)$, hence,

$$\{p_{i[m]}, p_{i[m]+1}, \ldots, p_n\} \quad \subseteq \quad \mathit{Pred}(t).$$

If t is not dead at m, then there exists a firing sequence $s_1 \ldots s_k$ such that $m \xrightarrow{s_1 \ldots s_k} m'$ and t has token-concession at m', thus,

$$m'(p_j) > 0 \qquad \text{for} \quad j = i[m], \ldots, n.$$

The places $p_1, \ldots p_{i[m]-1}$ are marked under m. If there is a p_j such that $1 \leq j < i[m]$ and $m'(p_j) = 0$ (loosing its tokens during the transition from m to m'), then a transition

$$t' \in \mathit{Post}(p_j) \subseteq \mathit{Post}(p_{i[m]}) \subseteq \ldots \subseteq \mathit{Post}(p_n)$$

has fired, say within the step s_l $(1 \leq l \leq k)$. We choose l to be minimal and consider the marking m'', reached just before the execution of s_l. At m'' the places $p_1, \ldots, p_{j-1}$ are marked, since they are marked at m and l is minimal, moreover, $p_j, \ldots, p_n$ are marked, since t' may fire. Hence, at m'' the transition t^* is enabled, contradicting that t^* is dead at m^* and therefore is dead at m''. This proves $\mathit{Post}(p_{i[m]}) \subseteq \mathit{deadtr}(m^*)$.

Assume that $m^*(p_{i[m]}) > 0$. Then, during the transition from m^* to m a transition from $\mathit{Post}(p_{i[m]})$ has been fired, contradicting the fact that all these transitions are dead.

We now show that t^* has a place $p^* \in \mathit{Pred}(t^*)$ which is dead at m^*. If $p_{i[m*]}$ is dead at m^*, then we put $p^* := p_{i[m^*]}$ and we are ready. Otherwise this place starting from m^* can become marked; let $s_1 \ldots s_k$ be a firing sequence such that

$$m^* \xrightarrow{s_1 \ldots s_k} m_1, \; m_1(p_{i[m*]}) > 0.$$

At m^* the places $p_1, \ldots, p_{i[m^*]-1}$ are marked; these places are marked at m_1 as well, since all the transitions from $\mathit{Post}(p_{i[m^*]})$ are dead at m^* and all the transitions which remove tokens from $p_1, \ldots, p_{i[m^*]-1}$ are elements of $\mathit{Post}(p_{i[m^*]})$. Since t^* is dead at m_1 as well, there exists a place in $\mathit{Pred}(t)$ which is unmarked at m_1; consider the place $p_{i[m_1]}$. Obviously, $i[m^*] < i[m_1]$.

In case that $p_{i[m_1]}$ is dead at m_1, then $p^* := p_{i[m_1]}$ is dead at m^*, since m^* is maximal-dead, otherwise, from m_1 there is reachable a marking m_2, such that the places $p_1, \ldots, p_{i[m_2]-1}$ are marked at m_2 and $i[m_2] > i[m_1]$. Since $i[m_j]$ is increasing monotonically but is bounded by n, the second case cannot occur arbitrarily often, hence we shall find a dead preplace of t^*. This proves Theorem 15.8. □

Corollary 15.9
Let N be an ordinary extended simple SNS. If N is place-live then any spontaneous transition of N is live at the initial marking.

Theorem 15.10
Let N be an ordinary extended free choice SNS, which contains no signal arc circuit and such that

$$\forall t \left(t \in T \Rightarrow card(St) \leq 1 \;\vee\; M(t) = \boxed{\vee}\right).$$

If N is place-live then N is live.

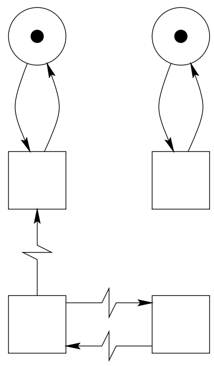


Figure 15.2: Ordinary extended free choice place-live *SNS* with signal arc circuit

Proof. For contradicition we assume that t_0 is not live at m_0. By Theorem 15.7, t_0 is not spontaneous, i.e. $St_0 \neq \emptyset$. Let m^* be a maximal-dead marking reachable from m_0 with $t_0 \in deadtr(m^*)$. Since N is place-live and EFC we may assume that all places $p \in Pred(t_0)$ are marked at m^*. We show now that all transitions $t_1 \in St_0 \neq \emptyset$ are dead at m^*. From this it follows that no spontaneous transition is in St_0 and for an arbitrary transition $t_1 \in St_0$ we can reach a marking where all places in $Pred(t_1)$ are marked, so that $Pred(t_1)$ contains no spontaneous transition, inductively leading to a contradiction since no circuit buildt from signal arcs only exists in the net.

Now, assume that $t_1 \in St_0$ is not dead at m^*. Then t_1 is live at m^* since m^* is maximal-dead. Hence, there exist executable steps s_i and reachable m_i such that

$$m^* \xrightarrow{s_1} m_1 \xrightarrow{s_2} m_2 \ldots m_{k-1} \xrightarrow{s_k} m_k$$

and $t_1 \in s_k$. During the state transition from m^* to m_{k-1} no tokens are removed from any place in $Pred(t_0)$ by a transition $t \in s_1 \cup \ldots \cup s_{k-1}$ because otherwise by $Ft \cap Pred(t_0) \neq \emptyset$ we have $Pred(t) = Pred(t_0)$ so that t_0 would not be dead at m^*. Hence, we obtained that at m_{k-1} all places in $Pred(t_0)$ are marked and $t_1 \in St_0$ is enabled. Because $\mathrm{card}(St_0) = 1$ or $M(t_0) = \boxed{\vee}$ the only possibility to prevent t_0 from firing at m_{k-1} is that t_1 takes a token from $Pred(t_0)$ which supplies us with $Pred(t_0) = Pred(t_1)$ which is impossible by $t_1 \in St_0 \subseteq Pred(t_0)$ leading to $t_1 \in St_1$ contradicting the irreflexivity of the signal relation. $\square$

The last theorem does not hold for extended simple *SNS*: Figure 15.3 shows an ordinary extended simple place-live *SNS* (which is not EFC) with a dead transition t_0.

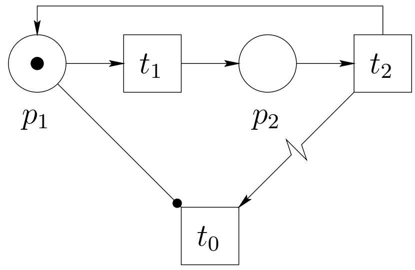


Figure 15.3: Ordinary extended simple place-live *SNS* with a dead transition

Theorem 15.11
Let N be an ordinary extended free choice SNS which has the deadlock-trap property and such that

$$\forall t \left(t \in T \Rightarrow \mathit{card}(St) \leq 1 \ \vee \ M(t) = \boxed{\vee} \right).$$

Then N is place-live.

Proof. We assume for contradiction that N is not place-live. Then there exists a maximal-dead marking m^* reachable from m_0 such that

$$D \ := \ \mathit{deadpl}(m^*) \neq \emptyset.$$

The nonempty set D is clean under m^*, hence it is a dynamic deadlock at m^*. We show, that D is a structural deadlock which contradicts the deadlock-trap property.

Let $s \in \mathit{Poss}(N)$ such that $s \cap FD \neq \emptyset$, i.e. any execution of s fires tokens to places in D. We have to show that $s \cap (DF \cup DB) \neq \emptyset$.

Clearly, any transition $t \in FD$ is dead at m^* since D contains only places which are dead at m^*, hence, $s \cap FD \subseteq \mathit{deadtr}(m^*)$.

Let $t_0 \in s \cap FD$. Since s is a possible step s contains a finite sequence $t_k, \ldots, t_1$ of pairwise different transitions such that t_k is spontaneous and

$$t_k S t_{k-1} S \ldots S t_1 S t_0.$$

The assumption that none of these transitions is an element of $DF \cup DB$ will lead to a contradiction as follows.

First note, that under our assumption, for $i = 0, \ldots, k$ the set $\mathit{Pred}(t_i)$ contains only places which are live under m^* and under every successor marking of m^* (since m^* is maximal dead).

Next we show, for $i = 0, \ldots, k-1$, that t_{i+1} is dead at m^* if t_i is dead at m^*. Assume for contradiction that t_{i+1} is not dead at m^*, i.e. t_{i+1} is live at m^*. Since all places in $\mathit{Pred}(t_i)$ are live at m^* by the EFC-property we can reach from m^* a marking

m_1 such that all places in $Pred(t_i)$ are marked under m_1. Obviously, t_i is dead at m_1 and t_{i+1} is live at m_1. Hence, a marking m^{**} is reachable from m_1 such that a step s^* containing t_{i+1} is executable at m^{**}. During the state transition from m_1 to m^{**} no token has been removed from a place in $Pred(t_i)$ because from $Ft \cap Pred(t_i) \neq \emptyset$ it follows $Pred(t) = Pred(t_i)$, i.e. we can fire t_i instead of t. Hence, all places in $Pred(t_i)$ are marked at m^{**} and $t_{i+1} \in St_i$ may fire at $Sm^{**}S$. By $M(t_i) = \boxed{\vee}$ or $St_i = \{t_{i+1}\}$, t_i may fire at m^{**} contradicting that t_i is dead at m^{**}.

Thus we have that the spontaneous transition t_k is dead at m^*, but all places in $Pred(t_k)$ are live at m^* from which it follows that from m^* a marking is reachable such that all places in $Pred(t_k)$ are marked contradicting that t_k is dead. □

Corollary 15.12
Let N be an ordinary extended free choice signal-net system without signal arc circuits which has the deadlock-trap property and such that

$$\forall t \left(t \in T \Rightarrow card(St) \leq 1 \ \vee \ M(t) = \boxed{\vee}\right).$$

Then N is live.

More general results can be found in the Diploma-Thesis of Adrianna Alexander [For99] and were published in [Sta99, FS00].

These results as well apply to non-ordinary *SNS*.

In our tool SESA we implemented (as an edit function) a check for the EFC-property. Since there are (so far) no practical relevant *SNS* which have that property we delayed the implementation of the check for the deadlock-trap-property.

16. Composition

Any signal-net system can be considered as a Petri net containing additional signal arcs and condition arcs. These arcs describe state and event signals flowing from one subsystem to the other. During a bottom-up design of a signal-net system often one starts with small Petri nets which then are interconnected by signal arcs and condition arcs to achieve the behaviour desired. In this section we investigate certain design rules which ensure that the resulting signal-net system (which we call the composition) has desired properties.

16.1. Basic Definitions

Let be $\mathcal{K} = \{K_i | i \in I\}$ a nonempty finite set of connected and pairwise disjoint ordinary Petri nets $K_i = [P_i, T_i, F_i, m_{0i}]$ which we call *components*. We put

$P := \bigcup_{i \in I} P_i, \qquad T := \bigcup_{i \in I} T_i, \qquad F := \bigcup_{i \in I} F_i, \qquad m_0 := \bigcup_{i \in I} m_{0i}.$

Moreover, let $B \subseteq P \times T$ be a set of condition arcs, let $S \subseteq T \times T$ a set of signal arcs (which considered as a relation in T is irreflexive and circuit-free) such that for all $i \in I$

(C1) $\quad p \in P_i \Rightarrow pB \cap T_i = \emptyset,$

(C2) $\quad t \in T_i \Rightarrow tS \cap T_i = \emptyset.$

Finally let M be a mapping from T into the set $\{\wedge, \vee\}$ ($M(t)$ is the signal-mode of $t \in T$).

The *composition* of $\mathcal{K}$ with $[B, S, M]$ is the signal-net system

$$Comp(\mathcal{K}, B, S, M) = [P, T, F, B, S, M, m_0].$$

The conditions (C1) and (C2) ensure that the components have no inner signals, i.e. if a place is connected to a transition by a condition arc, the place and the transition are in different components; and, if two transitions are connected by a signal arc they are in different components too. From this it is easy to see that there are signal-net systems which are not compositions.

The requirement of the connectedness of the components is obviously no restriction: we may include into $\mathcal{K}$ the connectivity components of the Petri net instead of the net itself.

Corollary 16.1

An ordinary signal-net system $N = [P, T, F, B, S, M, m_0]$ *is the composition of a set of components iff*

$$pBt \Rightarrow \neg p(F \cup F^{-1})^* t,$$
$$tSt' \Rightarrow \neg t(F \cup F^{-1})^* t'.$$

The state space of a composition obviously is a subset of the Cartesian product of the state spaces of its components. Therefore, if all the components of a composition are safe resp. bounded then the composition itself is safe resp. bounded. It is easy to see, that the converse it not true. In the sequel we will consider liveness properties of

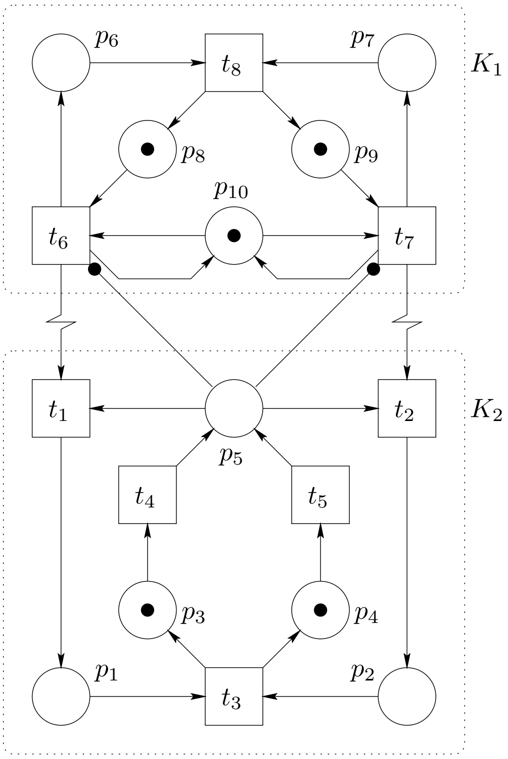


Figure 16.1: Composition

compositions.

For any nonempty subset $\mathcal{K}' = \{K_j | j \in I'\}$ of $\mathcal{K}$ we may consider the (sub-) composition $Comp(\mathcal{K}', B', S', M')$, where $B' := B \cap (P' \times T')$ and $S' := S \cap (T' \times T')$ contain exactly those arcs from B resp. S, which connect nodes from $\mathcal{K}'$. If M' is the restriction of M to $T' \subseteq T$,

$P' := \bigcup_{i \in I'} P_i, \quad T' := \bigcup_{i \in I'} T_i, \quad F' := \bigcup_{i \in I'} F_i, \quad m'_0 := \bigcup_{i \in I'} m'_{0i},$

then $Comp(\mathcal{K}', B', S', M') = [P', T', F', B', S', M', m'_0]$ is the subnet of $Comp(\mathcal{K}, B, S, M)$ adjoined with $\mathcal{K}'$.

Consider the composition represented in Figure 16.1. This composition is a *live* signal-net system, while the component K_2 is (as a Petri net) not live. Hence, a subcomposition of a live composition is not necessarily live.

For certain types of compositions we are able to show that reachability is monotonic with respect to subcompositions.

16.2. Conjunctive Compositions

A composition $Comp(\mathcal{K}, B, S, M)$ is called *conjunctive* iff for all transitions $t \in T$ it holds

$$card(St) > 1 \Rightarrow M(t) = \wedge.$$

Obviously, all the subcompositions of a conjunctive composition are conjunctive.

Theorem 16.2
Let $N = Comp(\mathcal{K}, B, S, M)$ be a conjunctive composition, m a marking of N and let s be an executable step at m in N. If $\mathcal{K}' \subseteq \mathcal{K}$, $N' = Comp(\mathcal{K}', B', S', M')$ is the corresponding subcomposition of N and $s' := s \cap T' \neq \emptyset$, then s' is an executable step at $m' := m|_{P'}$ in N'.

Proof. We have to show

(1) s' contains a spontaneous transition,
(2) s' is signal-complete,
(3) s' has token-concession,
(4) s' is enabled with respect to conditions,
(5) there is no forced transition $t' \in T' - s'$ such that $s' \cup \{t'\}$ satisfies (1) … (4).

Ad (1). The relation S is irreflexive and circuit-free, $S' \subseteq S$, hence, S' is irreflexive and circuit-free. Let be $t_1 \in s'$. If $S't_1 = \emptyset$ then t_1 is spontaneous in N' and we are ready, otherwise we choose a $t_2 \in S't_1$. Then $t_2 \in T'$ und $t_2 \in St_1 \subseteq s$, since N is conjunctive and s is signal-complete in N. Therefore, $t_2 \in s'$. If t_2 is not spontaneous in N' we proceed in the same way and choose a $t_3 \in S't_2$. If there is no spontaneous transition in s' we would arrive at a signal-circuit because s' is finite contradicting that S is circuit-free.

Ad (2). Since N' is conjunctive it suffices to show that for all $t \in s'$ always $S't \subseteq s'$ holds. Since s is signal-complete and N is conjunctive we have $St \subseteq s$ and $S't = St \cap T' \subseteq s \cap T' = s'$.

Ad (3). Since s has token-concession at m in N, for all $p \in P'$ it holds:

$$\textstyle\sum_{t \in s'} t^-(p) \leq \sum_{t \in s} t^-(p) \leq m(p),$$

hence, $[s']^- \leq m'$.

Ad (4). Since s is enabled with respect to conditions at m in N, all the places $p \in Bs$ are marked at m, consequently all places $p' \in Bs' \subseteq Bs$ are marked at m'.

Ad (5). Assume that s' is not maximal in N', i.e. there is a forced transition $t^* \in T' - s'$ such that $s' \cup \{t^*\}$ satisfies (1) … (4). Then $S't^* \subseteq s' \subseteq s$ and $s \cup \{t^*\}$ would fulfil (1) … (4) in N. $\square$

The example given in Figure 16.2 shows that conjunctivity is necessary for Theorem 16.2: $s = \{t_1, t_2\}$ is an executable step in N and $s' = \{t_1\}$ is not step in N'.

Theorem 16.3
Let $N = Comp(\mathcal{K}, B, S, M)$ be a conjunctive composition, $\mathcal{K}' \subseteq \mathcal{K}$ and, correspondingly, $N' = Comp(\mathcal{K}', B', S', M')$. If the marking m is reachable from m_0 in N then $m' := m|_{P'}$ is reachable from m'_0 in N'.

Theorem 16.3 is a consequence of Theorem 16.2 and the equation $\triangle s|_{P'} = \triangle(s \cap T')|_{P'}$.

The converse of Theorem 16.3 is not true as can be seen by Figure 16.1: in K_2 there is a marking holding two tokens on place p_1 reachable, no such marking is reachable in

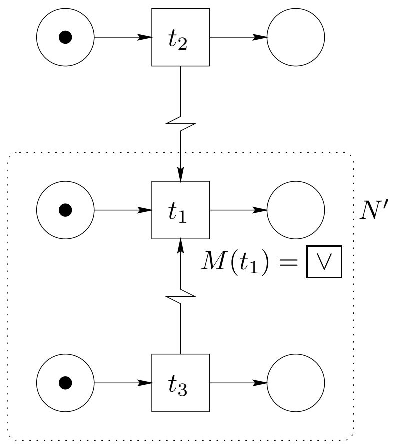


Figure 16.2: Disjunctive composition

N.

Theorem 16.4
Let $N = Comp(\mathcal{K}, B, S, M)$ be a conjunctive composition, $\mathcal{K}' \subseteq \mathcal{K}$ and correspondingly, $N' = Comp(\mathcal{K}', B', S', M')$. If no transition from N is dead at m_0 then no transition from N' is dead at m'_0.

Proof. Let $t \in T' \subseteq T$. Since t is not dead in N, in N there is reachable a marking m such that a step s is executable at m with $t \in s$. By Theorem 16.3 the marking $m' := m|_{P'}$ is reachable in N'. By Theorem 16.2 $s' := s \cap T'$ is an executable step at m' in N'. From $t \in s'$ we have that t is not dead in N'. □

On the other hand, there exist conjunctive compositions containing dead transitions and such that any proper subcomposition is live.

16.3. The Signal-Flow Relation

We define the signal-flow relation sig for components $K_i, K_j \in \mathcal{K}$ of an arbitrary composition $Comp(\mathcal{K}, B, S, M)$ by

$K_i sig K_j :\leftrightarrow (B \cup S)T_j \cap (P_i \cup T_i) \neq \emptyset.$

Hence, $K_i sig K_j$ holds iff there exists a condition arc $[p, t] \in B$ or a signal arc $[t', t] \in S$ leading from K_i to K_j, i.e. iff a transition $t \in T_j$ imports a condition $p \in P_i$ or a transition $t' \in T_i$ from K_i.

By (C1) und (C2) it holds

$K_i sig K_j \Rightarrow i \neq j.$

For the composition $Comp(\mathcal{K}, B, S, M)$, the relation sig describes the structure of the signal flow. By (C1), (C2) the relation sig is irreflexive. The reflexive-transitive closure of sig we denote by sig^*.

Mutual independence, i.e. *concurrency*, is given by the relation co which is defined as follows:

For $K, K' \in \mathcal{K}$ let

$$KcoK' :\leftrightarrow K \neq K' \wedge sig^*K \cap sig^*K' = \emptyset.$$

Different components K, K' are concurrent iff there is no component $K^* \in \mathcal{K}$ such that $K^* sig^* K$ and $K^* sig^* K'$.

Corollary 16.5
Two components K_i, K_j with $(B \cup S)T_i = \emptyset = (B \cup S)T_j$, i.e. without imports, are concurrent.

Using the signal-flow relation we are able to built natural subcompositions consisting of a component and all its (transitive) signal sources.

Theorem 16.6
Let $N = Comp(\mathcal{K}, B, S, M)$ be a composition, $K \in \mathcal{K}$ a component, $\mathcal{K}' := sig^*K$, and, $N' := Comp(\mathcal{K}', B', S', M')$. Then it holds:
(1) For all $t' \in T'$ it holds $B't = Bt$ and $S't = St$.
(2) If s is an executable step at m in N and $m' = m|_{P'}$, then $s' = s \cap T'$ is executable at m' in N'.
(3) If s' is an executable step at m' in N' and m is a marking of N with $m|_{P'} = m'$, then there is a step s executable at m in N such that $s' = s \cap T'$.
(4) A transition $t \in T'$ is dead in N iff t is dead in N'.

Proof. By definition we have $B't \subseteq Bt$ and $S't \subseteq St$. Let be $t \in T_j$ and $[p, t] \in B$. Then there is exactly one $i \in I$ with $p \in P_i$ and we have $K_i sig K_j$, hence, $i \in I'$. From $t \in T'$ we obtain $K_j sig^* K$. Consequently, $K_i sig^* K$, hence, $p\ inP'$ and $[p, t] \in B'$. In the same way $S't = St$ is proved.

From (1) it follows, that every transition $t \in T'$ in N has the same conditions, signal sources and preplaces as in N'. Therefore, t is spontaneous in N iff t is spontaneous in N', and, since s is signal-complete in N, s' is signal-complete in N', because transitions from different components are not conflicting. Assume that there is a forced transition $t' \in T' - s'$ such that $s' \cup \{t'\}$ satisfies the executability conditions (1) ... (4). Then $s \cup \{t'\}$ would satisfy (1) ... (4) which contradicts the executability of s. This proves assertion (2).

Since s' is executable at m' in N', the conditions (1) ... (4) are satisfied for s' in N at m. If condition (5) is violated by a transition t, then $t \in T - T'$. We include such transitions into s' to obtain finally a step s executable at m with $s' = s \cap T'$.

If t is not dead in N, then there is a firing sequence $m_0 \xrightarrow{s_1} m_1 \xrightarrow{s_2} m_2 \ldots m_k \xrightarrow{s_{k+1}}$ with $t \in s_{k+1}$.For $j = 0, 1, \ldots, k$ the step s_{j+1} is executable at m_j in N, therefore, by assertion (2), $s'_{j+1} := s_{j+1} \cap T'$ is executable at $m'_j := m_j|_{P'}$. We obtain a firing sequence $m'_0 \xrightarrow{s'_1} m'_1 \xrightarrow{s'_2} m'_2 \ldots m'_k \xrightarrow{s'_{k+1}}$ in N' with $t \in s'_{k+1}$, i.e. t is not dead in N'.

Conversely, if $t \in T'$ is not dead in N' at m'_0, then there exists a firing sequence $m'_0 \xrightarrow{s'_1} m'_1 \xrightarrow{s'_2} \ldots m'_k \xrightarrow{s'_{k+1}}$ with $t \in s'_{k+1}$. We have $m_0|_{P'} = m'_0$, hence, there is a step s_1 which is executable at m_0 in N with $s_1 \cap T' = s'_1$. For $m_1 := m_0 + \triangle s_1$, obviously, it holds $m_1|_{P'} = m'_1$, hence, there is a step s_2 with $s_2 \cap T' = s'_2$, and so on. By $t \in s'_{k+1} \subseteq s_{k+1}$, the transition t is not dead in N. $\square$

Obviously, Theorem 16.6 is interesting only in cases where $sig^* K \neq \mathcal{K}$ holds, e.g. if $sig^* K$ is a tree.

16.4. Tree-like Compositions

The composition of $\mathcal{K}$ with $[B, S, M]$ is said to be *tree-like*, iff the signal-flow relation sig of $Comp(\mathcal{K}, B, S, M)$ is

(C3) circuit-free, and,

(C4) mesh-free, i.e. for all $K, K', K'' \in \mathcal{K}$ it holds:
$K' \neq K'' \wedge K' sig K \wedge K'' sig K \Rightarrow K' co K''$.

Corollary 16.7

1. *Every tree-like composition contains a component K without imports, i.e. with $sigK = \emptyset$ and $sig^* K = \{K\}$.*

2. *For every component K of a tree-like composition the graph $[sig^* K, sig^{-1}]$ is a (directed) tree.*

The composition given in Figure 16.1 is not tree-like, the composition in Figure 16.2 is. Figure 16.3 shows a live, tree-like and conjunctive composition containing a component which is not live.

16.5. State-Machine Compositions

Most of the signal-net systems which are practically used as models for the verification of discrete event systems can be seen as compositions made of safe state-machines. State-machines are (ordinary) Petri nets where every transition has exactly one preplace and exactly one postplace. In a state-machine, therefore, the number of tokens is constant (invariant). Hence, a state-machine, where initially only one place is marked with one token, is safe. A state-machine with exactly one marked place is live and safe iff it is strongly connected.

The composition $Comp(\mathcal{K}, B, S, M)$ is said to be a *state-machine composition* (abbr. *SM-composition*), if every component $K \in \mathcal{K}$ is a state-machine containing exactly one token which is connected from the marked place. If, moreover, all the components are strongly connected, we call it *SCSM-composition.*

Corollary 16.8

1. *The components of a SM-composition are safe Petri nets without dead transitions.*

2. *The components of a SCSM-composition are live and safe.*

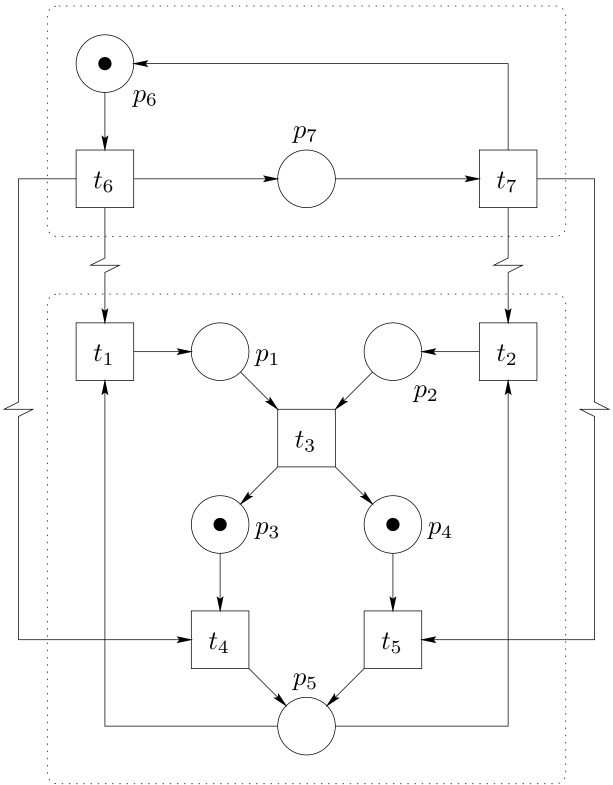


Figure 16.3: Tree-like composition

Let m be a reachable marking of the SM-composition $N = Comp(\mathcal{K}, B, S, M)$. Then, in every component of N, exactly one place is marked under m. Therefore, any transition, which imports two or more conditions from the same component, is dead.

With other words, to avoid dead transitions in a SM-composition, we have to obbey the following rule:

(R1) $p \neq p' \wedge p, p' \in Bt \wedge p \in P_i \wedge p' \in P_j \Rightarrow K_i \neq K_j$

If m is reachable in the SM-composition N and s is an executable step at m in N, then s contains from every component at most one transition (with marked preplace). Therefore, the following rule is necessary:

(R2) $t' \neq t'' \wedge t', t'' \in St \wedge t' \in T_i \wedge t'' \in T_j \wedge M(t) = \wedge \ \Rightarrow K_i \neq K_j$

Let us consider a transition t which imports a condition p as well as a transition t' from the same component K_i. The transition t may fire only if p is marked and t' fires. Hence, the preplace of t' must be marked, i.e. must equal p:

(R3) $pBt \wedge t'St \wedge p \in P_i \wedge t' \in T_i \Rightarrow pFt'$

A SM-composition $N = Comp(\mathcal{K}, B, S, M)$ is called *free-choice*, if conflicts are not decided by signals, i.e.

(FC) $Ft \cap Ft' \neq \emptyset \Rightarrow Bt = Bt' \wedge St = St'$.

Theorem 16.9
Let $N = Comp(\mathcal{K}, B, S, M)$ a free-choice tree-like SCSM-composition satifying (R1), (R2) and (R3). Then N is live.

Proof. We assume that N is not live. For any marking m of N let $dead_N(m)$ be the set of all transitions which are dead at m. Obviously, it holds $dead_N(m) \subseteq dead_N(m')$, if m' is reachable from m. A marking m is said to be *max-dead*, if $dead_N(m) = dead_N(m')$ for all markings m' reachable from m. At a max-dead marking every transition is either dead or live.

Since N is not live, we can reach in N a max-dead marking m_1 with $dead_N(m_1) \neq \emptyset$.

For every $t \in dead_N(m_1)$ let be $i(t) \in I$ such that $t \in T_{i(t)}$, i.e. $i(t)$ is the number of the component which contains t. Moreover, let $\mathcal{K}_t := sig^* K_{i(t)}$.

We now fix a transition $t \in dead_N(m_1)$ such that $\mathcal{K}_t$ is minimal. By Theorem 16.6 the marking $m_1' := m_1|_{P'}$ is reachable in the subcomposition $N' = Comp(\mathcal{K}_t, B', S', M')$ and $t \in dead_{N'}(m_1')$.

We have $dead_{N'}(m_1') = dead_N(m_1) \cap T' \subseteq T_{i(t)}$, otherwise there would exist a $t' \in T_j$, $j \neq i(t)$ which is dead at m_1'. The transition t' would be dead at m_1 in N and $K_j sig^* K_{i(t)}$, hence, $\mathcal{K}_{t'} \subset \mathcal{K}_t$, contradicting the minimality of $\mathcal{K}_t$.

Let p_1 be the place in $K_{i(t)}$ which is marked under m_1'. Since $K_{i(t)}$ is strongly connected, for any transition $t \in dead_{N'}(m_1')$ there is a shortest path $p_1 F t_1 F p_2 \ldots t_{n-1} F p_n F t$.

We choose a t, for which the number n is minimal. If $n = 1$ then p_1 is the preplace of t. From the minimality of n we obtain, that the transitions $t_1, \ldots, t_{n-1}$ on this path are live. Therefore, from m_1' we can reach in N' a marking m_2' such that the preplace p^* of t is marked.

Every condition $p \in Bt$ of t is imported, consequently postplace of a live transition. By (R1) different places $p, p' \in Bt$ are in different components K_i, K_j. Because the composition is tree-like these components are concurrent, i.e. the trees sig^*K_i and sig^*K_j are disjoint. Because of this independence from m_2' we can reach in N' a marking m_3', such that all the places from Bt are marked.

During this state transition the token on the preplace p^* of t is not removed, because for every transition $t^* \neq t$ with p^*Ft^* from the free-choice condition it follows that $Bt^* = Bt$ and $St^* = St$ hold. Hence, t^* is enabled iff t is, which is in contrdiction with t being dead. Under m_3' the place p^* therefore is marked.

Now, consider the case that $M(t) = \wedge$. By (R2) all $t' \in St$ are in different, thus, concurrent, components and are live. Firings in the upper components which are necessary to enable a step s mit $St \subseteq s$ do not disturb one another. Assume that during this enabling a condition $p \in Bt$ becomes unmarked. Then p is element of a component which is not concurrent with a component containing a transition $t' \in St$. Consequently, it is the same component. By (R3) it holds pFt'. Now, if a transition t^* tries to take the token from p, from the free-choice condition it follows that the firing of t^* is not necessary because t' has already concession (at any marking where t^* is enabled). Therefore, we can reach a marking m_4' where t may fire, contradiction.

In case that $M(t) = \vee$ the reasoning is analog. Since all $t' \in St$ are live, one can choose any signal source t'. $\square$

The result represented by Theorem 16.9 is unsatisfactory because the free-choice condition is in practice never valid. The problem remains to search for weaker conditions from which we can infer at least place-liveness or non-existence of dead transitions.

V. Invariants

17. State Invariants

A *State invariant* I (S-invariant for short) is a non-constant mapping defined on the set of all imaginable states which is constant on the set of all reachable states. Note that whether I is a state invariant of the system considered depends on its initial state.

We may use a known state invariant I to perform a quick non-reachability test: If $I(z) \neq I(z_0)$, where z_0 is the initial state, then the state z is not reachable from z_0.

For Petri nets or signal-net systems, the set of all imaginable states is the set of all markings; it can be considered as a subspace of the linear space of all integer valued place vectors, i.e. for all markings m one has:

$$m = \sum_{p \in P} m(p) e_p$$

where e_p is the marking such that $e_p(p) = 1$ and $e_p(p') = 0$ for $p' \neq p$.

We confine ourselves to integer valued state invariants; hence domain and range are (both) linear spaces. Then any *linear state invariant* I can be described by an integer valued place vector i:

$$\begin{aligned} i(p) &:= I(e_p), \\ I(m) &= \sum_{p \in P} m(p) I(e_p) = m \circ i := \sum_{p \in P} m(p) i(p). \end{aligned}$$

Here the P-vector i is not zero because I is not constant. On the other hand, every non-zero P-vector i such that $m \circ i = m_0 \circ i$ holds for all reachable states m defines a linear state invariant.
From $R_N(m_0) \subseteq R_{PN}(m_0)$ we obtain:

Theorem 17.1
Every (linear) state invariant of the underlying Petri net PN is a (linear) state invariant of N.

Let $Steps(m_0)$ denote the set of all steps s which are executable at a marking reachable from m_0:

$$Steps(m_0) = \{s \mid \exists m (m \in R_N(m_0) \wedge m \xrightarrow{s})\}.$$

Theorem 17.2
An integer valued P-vector $i \neq 0$ defines a (linear) S-invariant I iff for all steps s in $Steps(m_0)$ it holds $i \circ (s^+ - s^-) = 0$.

Proof. The mapping I with $I(m) = i \circ m$ is a state invariant iff for all $m, m' \in R_N(m_0)$ it holds $i \circ m = i \circ m_0$. This is the case iff for all $m \in R_N(m_0)$ and every step $s \in Steps(m_0)$ such that $m \xrightarrow{s}$ it holds

$$i \circ m = i \circ (m - s^- + s^+) = i \circ m + i \circ (s^+ - s^-).$$

For any step s let be $\triangle s$ be the P-vector with $\triangle s = s^+ - s^-$. We form a matrix C_{N,m_0} with $card(P)$ rows and $card(Steps(m_0))$ columns where the entry in the row corresponding to $p \in P$ and the column corresponding to $s \in Steps(m_0)$ is $\triangle s(p)$. Obviously, the P-vector $i \neq 0$ defines a linear state invariant of N iff $i \circ C_{N,m_0} = 0$. $\square$

If N is a Petri net then $Steps(m_0)$ corresponds to the set of non-dead transitions. In this case, C_{N,m_0} is the submatrix of the incidence matrix of N not containing the columns corresponding to dead transitions. In the general case, the columns of C_{N,m_0} are sums of columns of the incidence matrix C_{PN} of the underlying Petri net.

18. Place Invariants

Place invariants (P-invariants for short) are linear state invariants which hold for all initial states, i.e. being a place invariant is a structural property.

For Petri nets, the P-invariants are identified with the non-zero integer solutions of the homogeneous linear equation system $i \circ C = 0$, where C is the (full) incidence matrix of the net (with place vectors as columns and transition vectors as rows, where $C(p,t) := t^+(p) - t^-(p)$ is the entry in the row corresponding to p and the column corresponding to t). This is the case because there exists always an initial marking such that no transition is dead.

On the other hand one can show that, if a P-vector i defines a (linear) state invariant I at an initial state m_0 such that no transition t is dead at m_0, then i is a place invariant of PN. Hence, the linear state invariants of Petri nets without dead transitions can be easily computed.

Unfortunately, in *SNS*, there may exist steps (i.e. signal-complete sets of transitions containing at least one spontaneous transition) which are never executable because at any marking m where this step is enabled an additional forced transition will be enabled too.

Consider the set *eSteps* of all steps such that there exists a marking m with $m \xrightarrow{s}$ and let be i a place invariant of N. Then for any $s \in eSteps$ we have a marking m with $m \xrightarrow{s}$ and i is a state invariant at m, hence $i \circ m = i \circ (m + \triangle s)$, i.e. $i \circ \triangle s = 0$.

On the other hand, if $i \circ \triangle s = 0$ for all $s \in eSteps$, then i obviously is a place invariant of N. Let C_N denote the matrix with rows corresponding to the places and columns formed by the P-vectors $\triangle s$ for $s \in eSteps$. Then we have

Theorem 18.1
For any SNS $N = [P, T, F, V, B, W, S, M, m_0]$:

1. *A non-zero P-vector i is a place invariant of N iff $i \circ C_N = 0$.*
2. *Any P-invariant of PN is an P-invariant of N.*

The second assertion follows from the fact that the columns of C_N are sums of columns of the incidence matrix C_{PN} of the underlying Petri net.

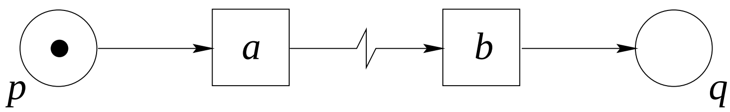


Figure 18.1: Counterexample to the converse of Theorem 18.1

The converse is not true; consider the *SNS* N depicted in Figure 18.1. Obviously, a is spontaneous, b is forced, $s = \{a, b\}$ is the only executable step, $eSteps = \{s\}$. Therefore the place vector i with $i(p) = i(q) = 1$ is a place invariant, but the underlying Petri net PN has no place invariants at all.

The problem is, for an *SNS* N, to compute one of the matrices C_N or C_{N,m_0} without knowledge of the set of all reachable markings, which in turn is needed to compute the set of all executable steps. Probably, we can do with an approximation of that set (e.g. ignoring all signal arcs gives the set of all singletons of transitions as an approximation providing the incidence matrix of PN).

19. Transition Invariants and Step Invariants

For Petri nets N, *transition invariants* (T-invariants for short) are defined as non-zero T-vectors j satisfying $C \circ j = 0$. If, in the reachability graph of N, there exists a circuit, i.e. a (reachable) marking m and a non-empty sequence w of transitions such that $m \xrightarrow{w} m$, then the Parikh-vector j of w (i.e., $j(t)$ is the number of occurrences of t in w) is a non-negative T-invariant of N. On the other hand, for any non-negative T-invariant j of N, one can find an initial marking m_0 such that a circuit with the firing count j exists in the corresponding reachability graph.

A Petri net N is said to be *covered by transition invariants* (*CTI* for short) iff there exists a solution j of $C \circ j = 0$ which is positive for any transition $t \in T$. It has been shown that every live and bounded Petri net is covered by transition invariants which provides us with a simple non-liveness test for bounded Petri nets.

For an *SNS* N, we have to distinguish between transition invariants and *step invariants*. For want of a better definition, we consider as *transition invariants of* N the transition invariants of the underlying Petri net PN. Hence, N is *CTI* iff PN is.

For an *SNS* N let $Steps(m_0)$ be the set of all steps executable at a marking m reachable from m_0. Then a mapping $j : Steps(m_0) \to \mathbb{Z}$ is called a *step invariant of* N *at the initial marking* m_0 iff $\sum_{s \in Steps(m_0)} j(s)(s^+ - s^-) = 0$, where 0 is the zero P-vector.

The *SNS* N is said to be *covered by step invariants* (*CSI* for short) iff N has a step invariant which is positive for any $s \in Steps(m_0)$.

An *SNS* N is said to be *step-live* (at its initial marking m_0) iff, for any marking $m \in R_N(m_0)$ and every step $s \in Steps(m_0)$, a marking m' exists which is reachable from m and such that s is executable at m'.

Theorem 19.1
If an SNS N is bounded and step-live, then N is covered by step invariants.

The *proof* runs along the same lines as the proof of the corresponding assertion (mentioned above) for Petri nets.

In order to use the theorem for testing non-step-liveness, we need a condition implying that the *SNS* N is not *CSI* since we do not know how to check that property without computation of $Steps(m_0)$.
We define

$$\delta(t,s) \quad := \quad \begin{cases} 1, & \text{if } t \in s \\ 0, & \text{else.} \end{cases}$$

Theorem 19.2

1. *If j is a step invariant of N then j^* is a transition invariant of N, where, for $t \in T$,*

$$j^*(t) \quad := \sum_{s \in Steps(m_0)} \delta(t,s) j(s).$$

2. *If an SNS N is CSI and has no dead transition at its initial marking, then N is covered by transition invariants.*

Proof. First we show now that j^* is a transition invariant:

$$\begin{aligned}\sum_{t\in T} j*(t)[t^+ - t^-] &= \sum_{t\in T}[\sum_{s\in Steps(m_0)} \delta(t,s)j(s)][t^+ - t^-] \\ &= \sum_{s\in Steps(m_0)} \sum_{t\in T} j(s)[t^+ - t^-] \\ &= \sum_{s\in Steps(m_0)} j(s)[s^+ - s^-] = 0.\end{aligned}$$

Now, let j be a covering step invariant, i.e. $j : Steps(m_0) \to \mathbb{Z}$ such that $j(s) > 0$ for any executable step s. Since N has no dead transition, every transition t is contained in at least one executable step s from $Steps(m_0)$, hence, $j^*(t) > 0$ for all $t \in T$, i.e. N is CTI. □

As a consequence, we have

Theorem 19.3
If a bounded SNS N has no dead transitions and is not covered by transition invariants then N is not step-live.

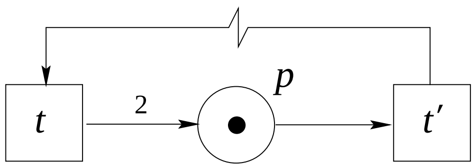


Figure 19.1: Counterexample to the converse of Theorem 19.3

On the other hand, there exist SNS which are CTI but not CSI. As an example, consider the net N depicted in Figure 19.1. The T-vector j with $j(t) = 1$ and $j(t') = 2$ is a covering transition invariant of N, but there is no step invariant, if p is marked in the initial marking.

We remarked above that in a Petri net PN, any non-negative transition invariant j occurs to be the Parikh-vector of a circuit in some reachability graph of PN. This does not hold for step invariants of SNS. Consider the SNS N depicted in Figure 19.2. Under the initial marking $m_0 = (1, 1, 0, 1)$, we have

$$Steps(m_0) = \{s_1,\ s_2,\ s_3,\ s_4\},$$

where

$$s_1 = \{t_1\},\ s_2 = \{t_2\},\ s_3 = \{t_3\},\ s_4 = \{t_2, t_3\}.$$

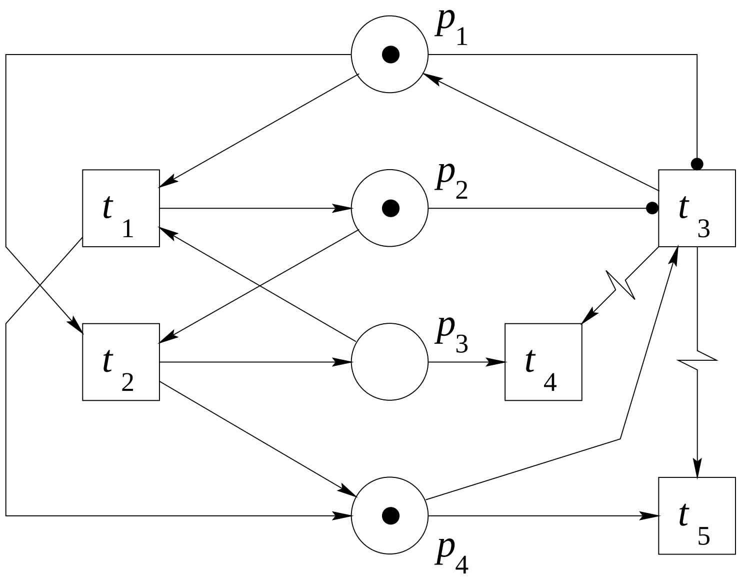


Figure 19.2: *SNS* with non negative step-invariant

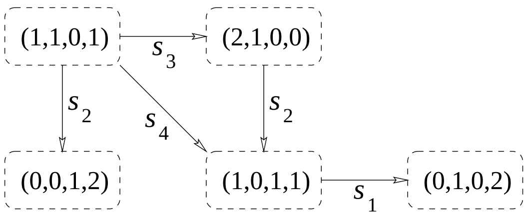


Figure 19.3: Reachability graph of the *SNS* depicted in Figure 19.2

Obviously, the mapping j with $j(s_1) = j(s_2) = 1$, $j(s_3) = 2$ is a non-negative step invariant of N at m_0. The reachability graph of N at m_0 (depicted in Figure 19.3) is circuit-free, and we shall show that no initial marking exists for N such that the corresponding reachability graph contains a circuit on which only the steps s_1, s_2, s_3 are fired.

Note that, if the step $s_3 = \{t_3\}$ is executable at a certain marking m then

- $m(p_1) \geq 1$ (since p_1 is a condition of t_3),
- $m(p_2) \geq 1$ (since p_2 is a condition of t_3),
- $m(p_3) = 0$ (since s_3 is not maximal otherwise), and
- $m(p_4) = 1$ (since s_3 is not enabled or not maximal otherwise).

Now, we assume that the reachability graph of N under a certain initial marking m_0

contains a circuit of that kind, i.e. there exist markings $m_k \in R_N(m_0)$ and steps $s^{(k)} \in \{s_1, s_2, s_3\}$ such that

$$m_1 \xrightarrow{s^{(1)}} m_2 \xrightarrow{s^{(2)}} \dots m_{l-1} \xrightarrow{s^{(l-1)}} m_l \xrightarrow{s^{(l)}} m_1.$$

For the Parikh-vector j of the sequence $s^{(1)}s^{(2)} \dots s^{(l-1)}s^{(l)}$ it holds

$$j(s_2) \quad = \quad j(s_1), \qquad j(s_3) = 2j(s_1).$$

Therefore, the step s_3 has to occur in the circuit. Without loss of generality, we can assume that $s^{(1)} = s_3$. Then s_3 is executable at m_1, hence,

$$m_1(p_1) \geq 1,\ m_1(p_2) \geq 1,\ m_1(p_3) = 0,\ m_1(p_4) = 1;$$

$$m_2(p_1) \geq 2,\ m_2(p_2) \geq 1,\ m_2(p_3) = 0,\ m_2(p_4) = 0.$$

Since s_2 is the only step executable at m_2 we have $s^{(2)} = s_2$ and

$$m_3(p_1) \geq 1,\ m_3(p_2) \geq 0,\ m_3(p_3) = 1,\ m_3(p_4) = 1.$$

Since t_4 is enabled at m_3 the step s_3 is not executable at m_3.
In the case that $s^{(3)} = s_1$ we obtain for m_4:

$$m_4(p_1) \geq 0,\ m_4(p_2) \geq 1,\ m_4(p_3) = 0,\ m_4(p_4) = 2.$$

Since t_5 is enabled at m_4 the step s_3 is not executable at m_4 and that property remains valid as long as only the steps s_1 and s_2 are fired because any step which removes tokens from p_4 contains t_3. Thus, the only possibility is that $s^{(3)} = s_2$.
In this case we obtain for m_4:

$$m_4(p_1) \geq 0,\ m_4(p_2) \geq 0,\ m_4(p_3) = 2,\ m_4(p_4) = 2.$$

The same argumentation as above shows that we arrived at a contradiction — the circuit does not exist.

The idea of the example was to prevent steps from being executable by enabling of a superset. Let us call a step s *saturated* iff it contains all its forced transitions, i.e. $sS^+ \subseteq s$. Obviously, the following monotonicity property holds.

Lemma 19.4
If a saturated step s is executable at m and $m' \geq m$, then s is executable at m'.

Consequently, a saturated step is executable iff it is enabled. Now, we can show

Theorem 19.5
Let j be a non-negative step invariant of the SNS N at m_0 such that for any step s, $j(s) > 0$ implies that s is saturated. Then there exists an initial marking for N such that j is the Parikh-vector of a circuit in the corresponding reachability graph.

Proof. Let c be the P-vector such that

$$c(p) \quad = \quad \sum_{[p,t]\in B} W(p,t).$$

Then the conditions of all transitions are fulfilled at any marking $m \geq c$. By *Step* we denote the set of all steps s such that $j(s) > 0$. We consider the marking

$$m \quad := \quad c + \sum_{s\in Step} j(s)s^{-}.$$

Obviously, any step $s \in Step$ is enabled ($j(s)$ times) at m, hence it is executable, i.e. we may execute the steps from *Step*, starting at m, the corresponding number of times in any order. Since j is a step invariant, this brings us back to the marking m. □

If, in our example, we remove the transitions t_4 and t_5 we obtain a net without signal arcs, hence every step is saturated. Then e.g. the sequence $s_2s_3s_1s_3$ can be executed at $m = (2,2,0,1)$.

Currently, our tool SESA computes (a base for all) place invariants of the underlying Petri net. These are then used for several different purposes:

- checking non-reachability of a given marking m:
  If there is a place invariant i such that $i \circ m \neq i \circ m_0$ then m is not reachable from m_0 in N.
- finding bounded places:
  If there is a non-negative place invariant $i \geq 0$ such that $i(p) > 0$, then the place p is bounded in N.
- saving memory:
  For any place invariant i, every place p with $i(p) \neq 0$ and any marking m reachable from m_0, the value $m(p)$ can be computed as

  $$m(p) \quad = \quad [i \circ m_0 - \sum_{p'\neq p} i(p')m(p')] : i(p),$$

  hence, we need not store it.

SESA also decides automatically whether the underlying Petri net is covered by place invariants. If this is the case, the *SNS* is structurally bounded (i.e. bounded under any initial marking).

Moreover, SESA computes a base of all transition invariants. The computation of step invariants has not been implemented so far.

Some results of the last part were first published in [Sta98].

Appendix

SESA Tool Description

SESA is our tool for the analysis of signal-net systems. Typical properties which can be verified are boundedness of places, liveness of transitions, and reachability of markings or states. General properties can be expressed in CTL and verified by the model checker of SESA. To reduce the size of the state space and the time for its construction, SESA offers several reduction methods. SESA also derives some analysis results from the underlying Petri net of a signal-net system. SESA inherited much of its code from the Petri net tool INA [RS98].

How to run SESA

To start the program, open a command shell, change to the directory which contains the net to be analysed, and into which the files to be constructed should be stored. Then enter `SESA.exe` at the shell prompt. Up to this version of SESA, there is no graphical user interface available. All output is displayed on the terminal of the command shell, where `SESA.exe` was started, and all input has to be typed in by the keyboard. In this section, we denote the typing of the key X by `<X>`. If SESA asks you for an output file name and you type the escape key `<esc>`, the output will be displayed on the terminal. On the other hand, if you type `<esc>` at an input file prompt, SESA expects your input from the keyboard.

If the file `OPTIONS.sna` is in the working directory, it will be read. On the other hand, the file `COMMAND.sna` will only be executed if you answer the question `same procedure as last time?` (which then appears at the beginning) with `<Y>`. In this case, the stored commands of the last session are repeated until `<H>` (for halt) is entered, or all commands have been executed. If there is no `COMMAND.sna`, or the question above has been answered with `<N>`, the main menu appears on the screen and the selected options are displayed:

```
>>>>>>>>>>>  Welcome to the Signal Net Analyzer  S E S A  <<<<<<<<<<<
Version 1.x                   Humboldt-University Berlin xx xyz 2002"

Current net options are:
    token  type: black
    time option: no times
    firing rule: arbitrary maximal steps
    synchros    : not to be used
    greediness : not to be used
    priorities : not to be used
    reductions : not to be used

Do You want to
    edit ? ......................................E
    fire ? ......................................F
    analyse ? ...................................A
    read the session report ? ...................S
    delete the session report ? .................D
    change options ? ............................O
    quit ? ......................................Q
```

```
choice >
```

By pressing the indicated letters the corresponding functions are selected. To stop a command, or quit a menu, `<Q>` can be entered in most cases. At many points this will also cancel running computations prematurely.

During a session, the program writes all analysis results and deductions into the file `SESSION.sna`. In the main menu, two menu items are offered, one to display the session report, and the other to erase it.

Options

After the start of SESA the current net options are displayed in the main menu. By entering `<O>`, these can be changed. There, the token type, the time option, the firing rule, the use of synchro sets, the use of greedy transitions, the use of priorities, the use of reductions, and line length are requested subsequently. The selected options are saved in the file `OPTIONS.sna`. The default net options of SESA can also be changed via the command line options at the start of the program.
In the following list, all possible options are individually presented:

token types With this option, you can determine whether the tokens are coloured.

```
<B>        black (indistinguishable) tokens
```

Hereby, you choose to work with (uncoloured) signal-net systems, whose markings consists of black indistinguishable tokens. The command line option therefore is `-black`.

```
<C>        coloured tokens
```

With this selection, you will work with coloured nets, which allows to work with coloured, i.e. distinguishable tokens. The command line option therefore is `-colour`. For more information about coloured nets, please, confront the section 3 on page 11.

If you have set the token type of a coloured net to black, SESA will ask: `Forget the colour structure?`; with `<Y>`, the colours are deleted. Warning: Information about the net can get lost this way!

If, on the other hand, you change the token type in the opposite way, the question appears: `Fold the Net?` If your answer is `<Y>`, you have to indicate how SESA should fold it. You can choose a maximal folding, a user-defined folding or no folding.

time option Here you can choose a clocked net type:

```
<N>        no time constraints
```

With this selection, no clocks will be used. The command line option therefore is `-notimes`.

```
<A>          arc timed
```

A time interval can be assigned to the input arcs of transitions. The command line option therefore is `-arctimed`. For more information about arc timed nets, refer to section 2 on page 8.

firing rule Here you can choose between different firing rules:

```
<N>          normal:  arbitrary maximal steps
```

That is the default firing rule. The executable steps under this rule are formed by first picking up a nonempty set of enabled spontaneous transitions and then adding as many as possible of those transitions that are forced to fire by signal-events produced by transitions in the step. The command line option for this rule is `-maximal`.

```
<S>          maximal single spontaneous transition steps
```

With this firing rule all steps will disappear from the step list which contain more than one spontaneous transition. The command line option for this rule is `-single`.

synchronisation sets With this option, you can determine the use of synchronisation sets. The transitions in the same synchronisation set should fire simultaneously as much as possible. The synchro option can be set only under the normal firing rule. The command line options for synchronisation sets are `-sync` and `-nosync`.

greedy transitions With this option enabled, only steps containing at least one greedy (spontaneous) transition are executed. If at the current state no greedy transitions are enabled, then the other steps are executed too. The greediness option can not be set under the single firing rule. The command line options for greedy transitions are `-greedy` and `-nogreedy`.

priorities Under the priority option only the spontaneous transitions with the greatest occurring priority are enabled. The command line options for priorities are `-priorities` and `-nopriorities`.

reductions The firing rule can also be influenced by the command line options `-diamond`, `-stubborn` and `-symmetric`. With the option `-diamond` the list of enabled steps under the normal firing rule is reduced with respect to diamonds. This means, that steps will be deleted from the step list which can be safely omitted without missing reachable markings (because the diamond property holds). For details about this reduction, please, confront the section 7 on page 21. The option `-stubborn` turns on the stubborn set reduction for the state space analysis. For a description of this reduction, see section 8 on page 27. For the firing rule maximal single spontaneous transition steps you can use `-noapprox` to select the non-approximative computation of stubborn sets. With the option `-symmetric`, SESA uses the symmetries of a signal-net system for the state space analysis. The symmetries are computed on demand. For details about symmetric reduction, see section 9 on page 37. More explanations of the firing rules, synchronisation sets, greedy transitions and priorities can be found in section 1.2 on page 5ff.

line length With this option, you can determine the length of the output lines to fit your terminal.

At the command line, it is also possible to change the file names of the session report, the command file and the options file. You can specify these names by the command line options `-session <sessfile>`, `-cmd <cmdfile>` and `-opt <optfile>`. With the commandline option `-prefix <prefix>` you can add a prefix for each of these file names. This is useful for running more than one instance of SESA in the same directory.

With the command line options `-noopt` and `-nocmd` you can prevent the processing of the files `OPTIONS.sna` and `COMMAND.sna` at the start of a session. The command line option `-reset` sets all options back to their default value. The order of command line options matters.

If you want the names of transitions and places to be displayed at the terminal and written in the session report, you can specify the option `-names` at the commandline, otherwise specify `-nonames`.

At the end of a command line, you can specify the file name of the net you want to analyse. With `-help` you get a short list of all possible command line options before the program starts:

```
SESA command line options summary

 -black   -colour
 -notimes -arctimed
 -maximal -single
 -[no]priorities
 -[no]greedy
 -[no]sync
 -stubborn -diamond -symmetric
 -[no]names
 -[no]opt <optfile>
 -[no]cmd <cmdfile>
 -session <sessfile>
 -prefix <prefix>
 -reset   -help
 <filename>
```

The net editor

By pressing `<E>` in the main menu, the menu of the editor is shown:

```
Do You want to
     Quit the editing process.....................Q
```

```
        execute Input operations.....................I
        execute Output operations....................O
        Change something in the current net..........C
        Delete something in the current net..........D
        check the EFC property.......................E
        Search for signal circuits...................S
        Test connectedness of the current net........T
        decompose or Merge...........................M
    edit>
```

If no net is loaded, then the edit menu is substituted by the file input menu, which appears also by pressing `<I>` in the editor.

In the file input menu, nets can be entered with the command `<T>`. Besides, if a net is requested as a file, you can switch to the terminal mode with `<esc>` and enter the net as described in the following.

First of all, the net number (default value: 0) and the net name (16 characters maximum) are requested. It is recommendable to fill out both, because the net number and name appear at many points in the protocol and in the saved files, and may therefore help to prevent confusion.

In the case of a uncoloured net SESA expects a list of places with their pre and post arcs next. First, SESA asks for the number of the place by the prompt `place nr.` and after typing in this number you can specify the `Token load (default = 0)` of that place. Then SESA asks for the numbers of the pre-transitions by the prompt `from transition nr.`, which can be stopped by pressing `<Q>` at the prompt. After that, the numbers of the post-transitions are expected by the prompt `to transition nr.`, which can also be stopped by pressing `<Q>`. Then SESA asks for the number of the next place, and so on... You can stop the input of the list of places by pressing `<Q>` at the `place nr.` prompt.

Because there can be isolated transitions in signal-net systems, the next question of SESA is: `Give the numbers of transitions without pre- and post-arcs`. The input of the numbers of these transitions can also be stopped by pressing `<Q>`.

After that, you have to type in the names for all the places and transitions you introduced before. After the name of each transition, SESA asks also for the numbers of places, which have a condition to that transition and the numbers of transitions, which have a signal to that transition. The input of these numbers can be stopped by pressing `<Q>`.

The input of a coloured net is a little bit more complicated. First, all places and their colours and token load have to be typed in. For each place and each colour you have to supply a name. Then SESA asks for the transitions and their colours. After the input of the names for the colours for a transition, you have to supply the pre- and post-arcs for this transition. Unfortunately conditions and signals cannot be typed in. But you can insert such arcs into a coloured net by using the change menu.

When you read a uncoloured net from a file by pressing `<F>` in the input file menu, the file must be accepted by the following grammar in EBNF:

```
<netfile> ::= "P   M   PRE,POST  NET " <nr> ":" <name> "<cr>"
```

```
                    <flowarcs>
                    "@<cr>"
                    "pl-nr. name             icp<cr>"
                    <places>
                    "@<cr>"
                    "<cr>"
                    "tr-nr. name             pri md conditions; signals;<cr>"
                    <transitions>
                    "@<cr>"
     <flowarcs> ::= { <nr> " " <tokens> " "
                      [ <prelist> ] [ "," <postlist> ] "<cr>" }
      <prelist> ::= { <nr> [ ":" <mult> ] " " }
     <postlist> ::= { <nr> [ ":" <mult> ] " " }
       <places> ::= { <nr> ": " <name> " " <icp> "<cr>" }
  <transitions> ::= { <nr> ": " <name> " " <priority> " " <modus> " "
                      [ <conditions> ] ";" [ <signals> ] ";<cr>" }
   <conditions> ::= { <nr> [ ":" <mult> ] " " }
      <signals> ::= { <nr> " " }
```

The following grammar describes the format of coloured net files:

```
   <colnetfile> ::= <netfile>
                    "AGGREGATION:<cr>"
                    "places:<cr>"
                    <plcolours>
                    "@<cr>"
                    "transitions:<cr>"
                    <trcolours>
                    "@<cr>"
    <plcolours> ::= { <nr> ":" <name> " " { <nr> " " } "<cr>" }
    <trcolours> ::= { <nr> ":" <name> " " { <nr> " " } "<cr>" }
```

After loading or typing in a net under the input file menu, you can perform output operations (by pressing `<O>`) or change the structure of the net (by pressing `<C>` or `<D>`) in the edit menu.

There are also test functions for some structural properties of the current net:

`<E>`	`check the EFC property`

With this function, you can check the EFC property of a signal-net system. For the definition of the EFC property see section 15 on page 72.

`<S>`	`Search for signal circuits`

This function searches for circuits in the signal relation of a net. In most cases, a signal circuit in a net is the result of an input error.

`<T>`	`Test connectedness of the current net`

With this function, you can test the connectedness of the nodes of a net with respect to the flow relation, the condition relation and/or the signal relation. If the net consists of more than one component, then you can write these components to separate files.

`<M>        decompose or Merge`

This function offers to double a node or to merge two nodes or two nets. Further, you can decompose a net into its elementary modules. For composition of modules see section 16 on page 79.

The simulator

By pressing `<F>` in the main menu, the program changes into the simulation mode. At the beginning and after each operation in this mode the current marking and a list of its executable steps are shown. Each step has its own number, so you can fire this step by typing in this number. If you want to cancel the execution of the last step, then press `<b>`. By pressing `<r>`, you reset the current marking to the initial marking.

If you want to see the stubborn set used by the stubborn reduced reachability graph, then press `<s>`. Press `<c>`, if you want to see the construction of this set. For more information about the stubborn set reduction, see section 8 on page 27. For the firing rule maximal single spontaneous transition steps you can toogle between the approximative and the non-approximative computation of stubborn sets by pressing `<a>`.

To see the step list reduced with respect to diamonds, press `<d>`. For details about the diamond reduction, see section 7 on page 21.

You can write the current marking into a `.mar` file by pressing `<w>`. To leave the simulation mode and return to the main menu enter `<q>`.

Analysing signal-net systems

By pressing `<A>` in the main menu, you enter the most important menu of SESA the analysis menu. In the analysis menu, different analysis procedures are offered depending on the net type and the status of the analysis. Only those procedures are offered which can lead to statements about the important dynamic properties. To return to the main menu, enter `<Q>`.

```
Analysis menu:

Non-reachability test of a partial marking using the state equation.........N
Compute a minimal path from the initial state to satisfy a predicate........P
Compute a minimal path from the initial state to a (sub-)marking............O
Check a CTL-formula.........................................................F
Compute a reachability graph................................................R

Compute the symmetries of the net...........................................Y
Define a concession predicate...............................................D
```

```
For the underlying Petri net (signal arcs ignored):
   Decide structural boundedness.......................................S
   Decide boundedness..................................................B
   Compute a base for all S/T-invariants [non-reachability test]...........I

choice >
```

Before the analysis menu is displayed, SESA executes a pre-analysis of the net, which investigates structural properties, and also checks which functions are available for the given net. At the beginning, you can set writing options. These include, for example, the output option for static conflicts: `Print the static conflicts?`

The progress of the analysis is indicated in a status line above the Analysis menu:

```
SCV SCF Ft0 tF0 Fp0 pF0 CPI CTI  B   SB  REV DSt BSt DTr DCF  L   LV L&B WL  CL
 Y   N   Y   N   N   N   ?   ?   Y   Y   ?   ?   ?   ?   ?   ?   ?   ?   ?   ?
```

The possible properties of the current net are listed. The symbol below each property indicates whether the property is fulfilled (`Y`), not fulfilled (`N`), or no decision could have been made yet (`?`). For an explanation of each property see the section on page 116.

In the following list, all possible functions of the analysis menu are presented:

`<N>`	`Non-reachability test of a partial marking using the state equation`

This function can decide the non-reachability of a marking using the state equation. A partial marking is sufficient here: the marking of the remaining places is considered to be not defined.

`<O>`	`Compute a minimal path from the initial state to a (sub-)marking`

A marking is tested for reachability, and, if possible, a path is displayed. The path is minimal with respect to the length (i.e. number of firing procedures) or the cost (the value of a path is the sum of the values of the fired transitions). Here, a partial marking is sufficient. The marking of the remaining places is considered as not defined.

The marking to be examined can be read from a file or, by pressing `<esc>`, entered directly.

For a net with time allocation, in addition to the computation of a minimal path with respect to length or cost, it is also possible to compute a path which is minimal with respect to time (i.e. a fastest one). Time allocation will be inquired anyway.

After entering (or reading) the target marking, you can enable some reduction methods for the construction of the state space. You can use the stubborn set reduction or the diamond reduction. In combination with the normal firing rule you will only get an upper bound of the minimal length, due to reducible steps.

In all other cases (single spontaneous transition steps or minimal values) you will get exact results. Reachability is preserved by both reductions. For details refer to section 7 on page 21 and section 8 on page 27.

It is also possible to cut of the construction of the state graph at states, which satisfy a defined "bad" predicate.

Then, SESA starts to construct the state graph of the net breadth first. The number of states so far computed is displayed: `States generated`.

If SESA encounters a marking that agrees with the target marking on the places defined, the examination is cancelled. The target marking is reported on the screen as reachable. The path can be written into a separate file with the extension `.tra` or in the session report.

If the target marking is not reachable, there are two cases: if the entire state graph can be constructed and saved, the marking will be regarded as unreachable (`The marking is not reachable`); otherwise (always in unbounded nets), SESA cancels with the message `Node overflow` and states that no decision can be made: `No decision possible.`

By pressing `<Q>`, the computation process can be cancelled at any time. Any dead states which may have been found are taken into account in the further analysis.

`<P>`	`Compute a minimal path from the initial state to satisfy a predicate`

This function computes a minimal path to a state which satisfies a previously specified state predicate. Otherwise, this function is similar to the one with `<O>`.

`<R>`	`Compute a reachability graph`

This command constructs the state graph of a signal-net system. For the construction of the state space, you can enable some reduction methods: You can use the symmetrical reduction, the stubborn set reduction or the diamond reduction. For these reductions refer to section 7 on page 21, section 8 on page 27 and section 9 on page 37. It is also possible restrict the depth of the state graph and to cut of the construction of the state graph at states, which satisfy a defined "bad" predicate.

Then, SESA starts to construct the state graph of the net depth first. The number of states so far computed is displayed: `States generated`.

Subsequently, different graph analyses can be executed, CTL-formulae and predicates can be created or checked, and certain results can be written either into separate files or in the session report. For more information about graph analysis in SESA see the section on page 113.

`<F>`	`Check a CTL-formula`

With this function, you can check CTL-formulae, i.e. determine their validity and generate proofs. For more information about checking CTL-formulae in SESA see the section on page 115.

<Y> Compute the symmetries of the net

When computing the symmetry group of the current net, SESA considers possible time allocations.

First, you have to state whether fixed points should be set for places and transitions, or whether the initial state should be considered as symmetric: Do you want to set fixpoints? or Initial state to be symmetric?

Sometimes, other symmetries or none at all are found in this way, and the computation is cancelled: Trivial transition partition!

By answering the question Write the symmetries to the session file? with <Y>, the generators of the symmetry group are written into the session-report. With <N>, only the decompositions of the place and transition sets into equivalence classes are written.

During the computation, a counter records the generators found; the running computation can be aborted with <Q>.

At the end of the computation, the number of generators and the number of symmetries, which can be obtained by combining them, are displayed.

The function <Y> can also be used to have a (already once computed) symmetry group be re-computed. In order to do this, you only have to answer the question Compute the symmetries once again? with <Y>. For example, you can set new fixed points, or consider the initial marking.

<D> Define a concession predicate

With this function, a predicate is defined based on a transition set to be specified; this predicate is satisfied exactly by those states in which at least one transition of the set is enabled.

<S> Decide structural boundedness

This function decides whether the net is covered by P-invariants. If this is the case, it is structurally bounded, i.e. bounded in every initial marking.

<B> Decide boundedness

This function decides the boundedness of a net by the computation of the coverability graph of the underlying Petri net. SESA computes the graph using the algorithm of Karp and Miller. In case the net is bounded, the coverability graph corresponds to the usual state graph of the underlying Petri net.

`<I>`	`Compute a base for all S/T-invariants [non-reachability test]`

This command computes a basis for the space of all invariants of the selected type. SESA states whether invariants were found, and possibly derives further deductions from it. The program decides whether the net is covered by invariants of the selected type, i.e. whether an invariant exists which is positive in all components. If such an invariant was actually computed, it will be displayed as well. If P-invariants were found, a fast non-reachability test is offered.

Further analysis of the reachability graph

After the (incomplete) construction of the reachability graph or after the model checking of a CTL-formula which led to a complete reachability graph, you enter the graph analysis menu. In this menu different graph analyses can be executed, CTL-formulae and predicates can be created or checked, and certain results can be written either into separate files or in the session report:

```
Graph analysis menu
Do You want to
     quit analysis of the computed graph ................ Q

     test the reachability/coverability of a marking ..... R
     convert a set of states to a predicate ............. C
     define a concession predicate ...................... E
     check a CTL-formula ................................ F
     compute distances .................................. A
     compute circuits ................................... K
     check liveness properties .......................... L
     compute strongly connected components .............. V
     check dynamic conflicts ............................ Y
     check for false diamonds ........................... U
     write the computed graph (states and arcs) .......... W
     write all arcs ..................................... X
     write all states ................................... M
     write all states satisfying a predicate ............ P
     write all states with a given successor ............ G
     write the dead states .............................. D
     write the bad states ............................... B
     write a trace to a state ........................... T
     write the list of executed steps ................... S
     inspect a result file .............................. I
choice>
```

Warning: If you leave this menu by pressing `<Q>` the memory of the reachability graph is freed. So, you have to construct the graph again, if you want to perform more graph analysis.
In the following list, all possible functions of the graph analysis menu are presented:

`<R>        test the reachability/coverability of a marking`

With this function, you can execute reachability or coverability tests, and find (not necessarily minimal) paths from the initial marking to a target marking.

`<C>        convert a set of states to a predicate`

With this command, a predicate for a set of states is defined which is satisfied exactly by these given states. You can construct the predicate with respect to all computed states, the dead states, states where a set of transitions is fireable, or a set to be specified by state numbers. The predicate defined can be saved in a file with the extension `.pdc`.

`<E>        define a concession predicate`

With this function, a predicate is defined based on a transition set to be specified; this predicate is satisfied exactly by those states in which at least one transition of the set is enabled.

`<F>        check a CTL-formula`

With this function, you can check CTL-formulae, i.e. determine their validity and generate proofs. For more information about checking CTL-formulae in SESA see the section on page 115.

`<A>        compute distances`

With this command, minimal and maximal distances between nodes of the state graph can be computed. The results are written into the session report.

`<K>        compute circuits`

With this command, circuits in the state graph can be computed and evaluated.

`<L>        check liveness properties`

With this command, a liveness analysis can be executed. This works only for completely computed state graphs without stubborn reduction.

If the net contains transitions which are dead in the initial state, then the liveness analysis is restricted to the transitions that are not dead. The net can be live if all dead transitions are considered as facts; the property `LV` (Liveness when ignoring dead transitions) is then set accordingly. Further weaker notions of liveness are explained on the screen, if necessary.

`<V>        compute strongly connected components`

With this function, the strongly connected components are computed, and, upon request, written into a file with the extension `.res`.

<Y> check dynamic conflicts

With this function, you can search the set of reachable states for dynamic conflicts (see in section 10 on page 44).

<U> check for false diamonds

This function looks for reachable states where the diamond property does not hold (see section 7 on page 21).

<...> write ...

With these commands, the computed graph, or parts of it, are written into the session report, or, upon request, into a file. By entering `<esc>` on the file name prompt, you can redirect the output to the screen. In most cases, a selection of states can be defined.

<I> inspect a result file

With this command, you can inspect different files created during the analysis.

CTL model checking

Testing the validity of Computation Tree Logic CTL-formulae in the initial state is called model checking. Thereby, witnesses for existence-quantified sub-formulae, and counter examples for all-quantified sub-formulae can be determined and displayed. See section 11ff for details about CTL and the timed and extended Computation Tree Logic.

At the `formula input file` prompt, you can then read a `.ctl` file, or, by pressing `<esc>`, enter a formula directly. A very short explanation of the CTL syntax is displayed when entering formulae by hand:

```
Syntax:

 Boolean combination: NOT f, f1 AND f2, f1 OR f2, f1 IMPL f2, f1 EQUIV f2
 Temporal operators : EX f, EF f, EG f, E[f1 U f2], E[f1 B f2]
                      AX f, AF f, AG f, A[f1 U f2], A[f1 B f2]
 Predicates         : disjunction of conjunctions of interval specifications
                      use P and a number or a file name (*.pdc with quotes)
                      to refer to a predicate e.g  P1  or  P"pred1.pdc"
 Atomic propositions: references to the marking of a place by number or name
                      and comparisons > < = <> >= <= of markings and marking
                      sums e.g.  p1   m(p1)=0  or   m(p1)+m(p2)=1
                      for reference by name use quotes e.g.  m(p"end")
                      a reference without m( ) is interpreted as  m( )>0
 Transition formulae: boolean combinations of references to transitions
                      attached to the quantifiers E and A to limit the
                      range of temporal operators e.g.  E t1 F m(p1)
                      or  A (t"one" OR t"two") G P"invar.pdc"

Please, type the formula to check:
```

A complete graphical diagram of the syntax of CTL used in SESA is shown in Fig. A.1 and A.2. Each nonterminal of the language is explained by a little diagram. The name of the nonterminal is shown in boldface on the top left corner of this diagram. Below this name, there is the entry point of the syntax diagram for the nonterminal. The boxes of the diagrams represent other nonterminals. Their names are written in the boxes. On the other hand, the circles represent the terminals. The symbols of these terminals are shown in the circles. To check the correctness of a sentence, start with the top left nonterminal "ctl" and then follow the arrows from the left hand side through the boxes and circles to the right end of the diagrams.

After typing in the formula and afterwards answering the question `ok?` with `<Y>`, you can write the formula to a file with the extension `.ctl`. Thereafter, SESA is parsing the formula and the predicates whose numbers were mentioned in the formula are requested. You can either read previously defined predicates from a file, or, by pressing `<esc>`, enter predicates directly. Before the computation is started, you can specify some output options for the proof of the formula. The progress of the computation is displayed by the number of generated states. The computation can be cancelled by pressing `<Q>`. After a successful computation, the value of the formula is displayed. SESA warns with `TRUE/FALSE in the computed subgraph` if a reduced or incomplete graph was generated and the truth value of a formula in the complete graph is not deducible (for reduction techniques see section 7 on page 21, section 8 on page 27 and section 9 on page 37). If the generated graph is complete, you enter the graph analysis menu, see the section on page 113. Otherwise, you can write the incomplete graph or check the next formula.

Properties

In the following list, all properties of the status line in the analysis menu and the graph analysis menu of SESA are explained:

`SCV` subconservative

A net is sub-conservative, if all transitions add at most as many tokens to their post-places as they subtract from their pre-places. The total number of tokens can therefore not increase.

`SCF` statically conflict-free

If two transitions have a common pre-place, they are in a static conflict about the tokens on this pre-place. Then, the net is not static conflict free.

Further information can be found in section 10 on page 44ff.

`Ft0` transition without pre-place

A net has $Ft0$-transitions, if there are spontaneous transitions without a pre-place and without a condition but with a post-place; $St = \emptyset$, $Ft = \emptyset$, $Cond(t) = \emptyset$, $tF \neq \emptyset$.

`tF0` **transition without post-place**

A net has $tF0$-transitions, if there are spontaneous transitions without a post-place and without any signal to another transition but with a pre-place; $St = \emptyset$, $tF = \emptyset$, $tS = \emptyset$, $Ft \neq \emptyset$.

`Fp0` **place without pre-transition**

A net has $Fp0$-places, if there are places without a pre-transition but with a post-transition; $Fp = \emptyset$, $pF \neq \emptyset$.

`pF0` **place without post-transition**

A net has $pF0$-places, if there are places without a post-transition but with a pre-transition; $pF = \emptyset$, $Fp \neq \emptyset$.

`CPI` **covered by place invariants**

A net is covered by place invariants, if there exists a P-invariant which assigns a positive value to each place. If this is the case, the net is structurally bounded, i.e. bounded under any initial marking.

`CTI` **covered by transition invariants**

A net is covered by transition invariants, if there exists a T-invariant which assigns a positive value to each transition.

Further information can be found in section 19 on page 95ff.

`B` **bounded**

A net is bounded, if there is a number k such that, in any reachable marking, there are never more than k tokens on a place.

`SB` **structurally bounded**

A net is structurally bounded, if it is bounded in every initial marking.

`REV` **reversible**

A net is reversible, if the initial state can be reached from every reachable state.

`DSt` **dead state reachable**

A dead state is reachable, if a state is reachable in which no transition can fire any more.

`BSt` **bad state reachable**

If a state satisfies a so-called "bad" predicate, it is not further developed when computing a state graph. In this case, the attribute `Bst` is set. However, after leaving the graph analysis, this attribute is reset to `?`.

`DTr` **dead transition exists (at the initial marking)**

This attribute indicates whether the net has dead transitions in the initial marking, i.e. facts.

`DCF` **dynamically conflictfree**

A net is said to be dynamically conflict free, if no state is reachable in which a step conflict or a transition conflict occurs.

Further information can be found in section 10 on page 44ff.

`L` **live**

A net is live, if all its transitions are live in the initial marking, i.e. no state is reachable in which a transition is dead.

`LV` **live if dead transitions ignored**

A net is live when ignoring dead transitions, if all its transitions, which are not already dead in the initial marking, are live. The transitions thereby ignored can be considered as unspecified facts.

`L&B` **live and bounded**

A net is live and bounded, if it is live and, if there is a number k such that, in any reachable marking, there are never more than k tokens on a place.

`WL` **weakly live**

A coloured net is weakly live, if all its transitions are weakly live, i.e. for each transition, there is a colour in which the transition is live in the initial marking.

`CL` **collectively live**

A coloured net is collectively live, if all its transitions are collectively live. A transition is collectively live, if for every reachable state a colour exists, such that in a state reachable from this marking, the transition can fire in this colour. In particular, every weakly live transition is also collectively live.

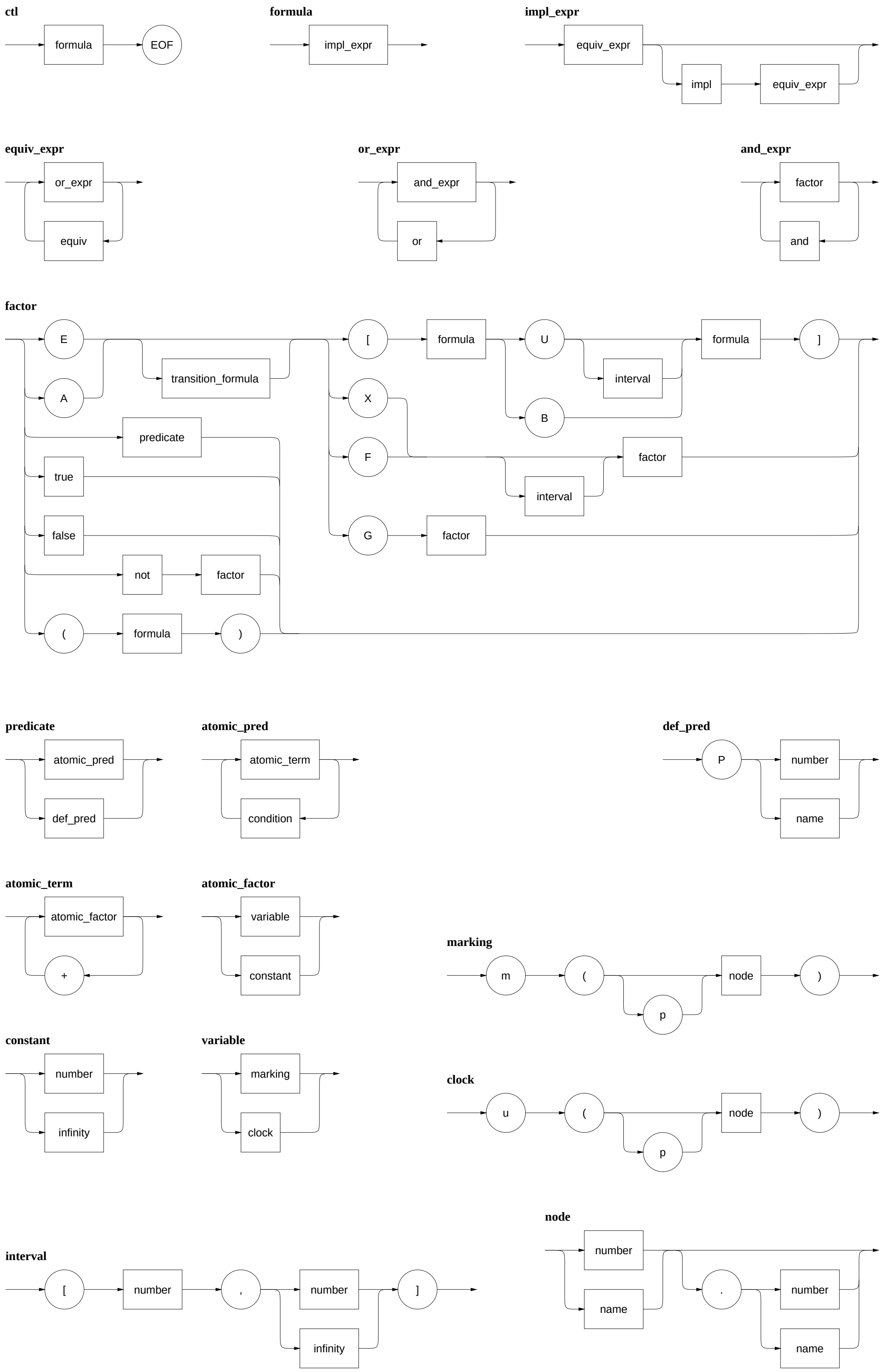


Figure A.1: CTL Syntax

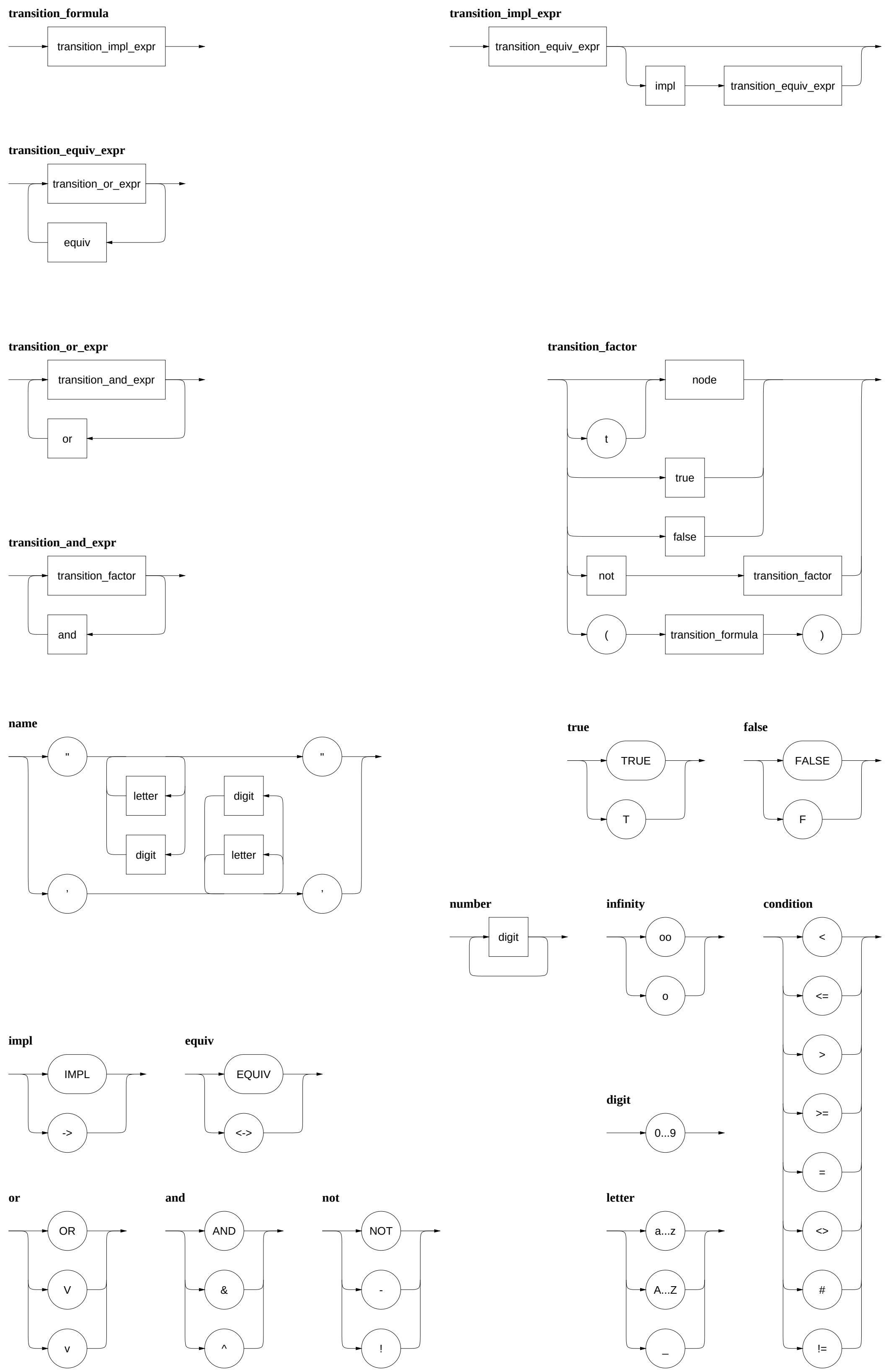


Figure A.2: CTL Syntax (cont'd)

References

[AH92] Rajeev ALUR and Thomas A. HENZINGER: *Logics and Models of Real-Time: A Survey.* In J. W. de BAKKER, C. HUIZING, W. P. de ROEVER, and G. ROZENBERG (editors): *Proceedings of Real-Time: Theory in Practice, REX Workshop, Mook, The Netherlands, June 3-7, 1991*, volume 600 of *Lecture Notes in Computer Science*, pages 74–106, Berlin, Germany, 1992. Springer-Verlag.

[BCM92] Jerry R. BURCH, Edmund M. CLARKE, and Kenneth L. MCMILLAN: *Symbolic Model Checking:* 10^{20} *States and Beyond. Information and Computation*, 98(2), June 1992.

[Ber02] Paul BERTHOLD: *Analyse von Signal-Netz-Systemen unter verschiedenen Schaltregeln.* Diplomarbeit, Humboldt-Universität zu Berlin, May 2002.

[CES86] Edmund M. CLARKE, E. Allen EMERSON, and A. Prasad SISTLA: *Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specification. ACM Transactions on Programming Languages and Systems*, 8(2): 233–263, 1986.

[CGP99] Edmund M. CLARKE, Orna GRUMBERG, and Doron PELED: *Model Checking.* MIT Press, December 1999.

[DA92] Rene DAVID and Hassane ALLA (editors): *Petri Nets and Grafcet: Tools for Modelling Discrete Event Systems.* Prentice Hall, New York, 1992.

[DV90] Rocco DE NICOLA and Frits W. VAANDRAGER: *Action versus state based logics for transition systems.* In I. GUESSARIAN (editor): *Semantics of Systems of Concurrent Processes, Proceedings LITP Spring School on Theoretical Computer Science, La Roche Posay, France*, volume 469 of *Lecture Notes in Computer Science*, pages 407–419. Springer-Verlag, 1990.

[EMSS91] E. Allen EMERSON, Aloysius K. MOK, A. Prasad SISTLA, and Jai SRINIVASAN: *Quantitative Temporal Reasoning.* In *Proc. 2nd International Computer Aided Verification Conference CAV*, volume 531 of *Lecture Notes in Computer Science*, pages 136–145. Spinger-Verlag, 1991.

[For99] Adrianna FOREMNIAK: *Struktureigenschaften von Signal-Ereignis-Netzen.* Diplomarbeit, Humboldt-Universität zu Berlin, August 1999.

[FS00] Adrianna FOREMNIAK and Peter H. STARKE: *Structural Analysis of Signal-Event Systems. Fundamenta Informaticae*, 41(3): 81–104, February 2000.

[Han93] Hans-Michael HANISCH: *Analysis of Place/Transition Nets with Timed Arcs and its Application to Batch Process Control.* In Ajmone MARSAN, M. (editor): *Lecture Notes in Computer Science; Application and Theory of*

Petri Nets 1993, Proceedings 14th International Conference, Chicago, Illinois, USA, volume 691, pages 282–299. Springer-Verlag, 1993.

[Hel97] Keijo Heljanko: *Model Checking the Branching Time Temporal Logic CTL.* Series A: Research Report 45, Helsinki University of Technology, Digital Systems Laboratory, Espoo, Finland, May 1997.

[HKL99] Hans-Michael Hanisch, Peter Kemper, and Arndt Lüder: *A Modular and Compositional Approach to Modeling and Controller Verification of Manufacturing Systems.* In *Proceedings of the 14th IFAC World Congress*, Beijing, China, July 1999. International Federation of Automatic Control.

[HPP+99] Hans-Michael Hanisch, Torsten Pannier, Dirk Peter, Stephan Roch, and Peter H. Starke: *Modeling and Verification of a Modular Level-Crossing Controller Design. at Automatisierungstechnik*, 47(8): 366–373, August 1999.

[Jen92] Kurt Jensen: *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use, Vol. 1: Basic Concepts.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag,

[Jen94] Kurt Jensen: *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use, Vol. 2: Analysis Methods.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1994.

[JK91a] Ryszard Janicki and Maciej Koutny: *Invariants and Paradigms of Concurrency Theory.* In E. H. L. Arts, J. van Leeuwen, and M. Rem (editors): *Parallel Architectures and Languages Europe (PARLE'91) Volume II: Paralell Languages, Eindhoven, The Netherlands, June 1991*, volume 506 of *Lecture Notes in Computer Science*, pages 59–74, Berlin, Heidelberg, 1991. Springer-Verlag.

[JK91b] Ryszard Janicki and Maciej Koutny: *Invariant Semantics of Nets with Inhibitor Arcs.* In J. C. M. Baeten and J. F. Groote (editors): *Proceedings of the 2nd International Conference on Concurrency Theory (CONCUR'91), Amsterdam, The Netherlands, August 1991*, volume 527 of *Lecture Notes in Computer Science*, pages 317–331, Berlin, Heidelberg, August 1991. Springer-Verlag.

[JK93] Ryszard Janicki and Maciej Koutny: *Structure of Concurrency. Theoretical Computer Science*, 112: 5–52, 1993.

[JK95] Ryszard Janicki and Maciej Koutny: *Semantics of Inhibitor Nets. Information and Computation*, 123: 1–16, 1995.

[Kar99] Sirko Karras: *Formale Verifikation und automatische Codegenerierung von SPS-Programmen.* Diplomarbeit, Otto-von-Guerike Universität Magdeburg, Lehrstuhl Steuerungstechnik, August 1999.

[Koz83] Dexter KOZEN: *Results on the propositional mu-calculus. Theoretical Computer Science*, 27: 333–354, 1983.

[KQ88] R. KÖNIG and L. QUÄCK (editors): *Petri Netze in der Steuerungstechnik.* Verlag Technik, Berlin, 1988.

[KV00] Lars Michael KRISTENSEN and Antti VALMARI: *Improved Question-Guided Stubborn Set Methods for State Properties.* In NIELSEN, M. and SIMPSON, D. (editors): *Lecture Notes in Computer Science: 21st International Conference on Application and Theory of Petri Nets (ICATPN 2000), Aarhus, Denmark, June 2000*, volume 1825, pages 282–302. Springer-Verlag, 2000.

[LH00] Arndt LÜDER and Hans-Michael HANISCH: *A Signal Extension for Petri nets and its Use in Controller Design. Fundamenta Informaticae*, 41(3): 415–431, February 2000. Previously published in Hans-Dieter Burkhard, Ludwik Czaja, and Peter H. Starke (editors): Proceedings of the CS&P'98 Workshop, number 110 in Informatik-Berichte, pages 98-105, Berlin, Germany, September 1998. Humboldt-Universität zu Berlin.

[MR95] Ugo MONTANARI and Francesca ROSSI: *Contextual Nets. Acta Informatica*, 32(6): 545–596, September 1995.

[Pan98] Torsten PANNIER: *Spezifikation, Verifikation und Implementation einer microrechnergestützten Bahnübergangssteuerung.* Diplomarbeit, Otto-von-Guerike Universität Magdeburg, Lehrstuhl Steuerungstechnik, October 1998.

[PPH98] Torsten PANNIER, Dirk PETER, and Hans-Michael HANISCH: *Safety-related Design and Verification of a Computer Aided Level-Crossing Control.* In Hans-Dieter BURKHARD, Ludwik CZAJA, and Peter H. STARKE (editors): *Proceedings of the CS&P'98 Workshop*, number 110 in Informatik-Berichte, pages 162–175, Berlin, Germany, September 1998. Humboldt-Universität zu Berlin.

[Rau96] Mathias RAUSCH: *Modulare Modellbildung, Syntese und Codegenerierung ereignisdiskreter Steuerungssysteme.* PhD thesis, Otto-von-Guerike Universität Magdeburg, Lehrstuhl Steuerungstechnik, 1996.

[RH95] Mathias RAUSCH and Hans-Michael HANISCH: *Net Condition/Event Systems with Multiple Condition Outputs.* In *Proc. EFTA'95, Paris*, volume 1, pages 592–600, October 1995.

[Roc98a] Stephan ROCH: *Definition und Berechnung sturer Schrittmengen in Signal-Ereignis-Netzen.* Diplomarbeit, Humboldt-Universität zu Berlin, August 1998. http://www.informatik.hu-berlin.de/˜roch/Diplom/.

[Roc98b] Stephan ROCH: *Stubborn Sets of Signal-Event Nets.* In Hans-Dieter BURKHARD, Ludwik CZAJA, and Peter H. STARKE (editors): *Proceedings of* *the CS&P'98 Workshop*, number 110 in Informatik-Berichte, pages 197–203, Berlin, Germany, September 1998. Humboldt-Universität zu Berlin.

[Roc99a] Stephan Roch: *Simultaneity in Signal-Event Systems.* In Josep M. Fuertes (editor): *Proceedings of the 7th IEEE International Conference on Emerging Techologies and Factory Automation ETFA'99*, pages 281–285, Barcelona, Spain, October 1999.

[Roc99b] Stephan Roch: *Simultaneity in Signal-Event Systems.* In Hans-Dieter Burkhard, Ludwik Czaja, Hung Son Nguyen, and Peter H. Starke (editors): *Proceedings of the CS&P'99 Workshop*, pages 196–203, Warsaw, Poland, September 1999. Final version appeared in *Fundamenta Informaticae.*

[Roc00a] Stephan Roch: *Analyzing and Reducing Simultaneous Firing in Signal-Event Nets. Fundamenta Informaticae*, 43: 321–330, August 2000.

[Roc00b] Stephan Roch: *extended Computation Tree Logic.* In Hans-Dieter Burkhard, Ludwik Czaja, Andrzej Skowron, and Peter H. Starke (editors): *Proceedings of the CS&P 2000 Workshop*, number 140 in Informatik-Berichte, pages 225–234, Berlin, Germany, October 2000. Humboldt-Universität zu Berlin.

[Roc00c] Stephan Roch: *extended Computation Tree Logic: Implementation and Application.* In *Proceedings of the AWPN 2000 Workshop*, Koblenz, Germany, October 2000. Universität Koblenz.

[Roc01] Stephan Roch: *Reducing Simultaenous Firing in Signal-Net Systems with Strongly Connected Sets.* In Hans-Dieter Burkhard, Ludwik Czaja, Hung Son Nguyen, and Peter H. Starke (editors): *Proceedings of the CS&P'2001 Workshop*, pages 230–241, Warsaw, Poland, October 2001.

[RS98] Stephan Roch and Peter H. Starke: *INA - Integrated Net Analyzer - Version 2.2 - Manual.* Technical report, first published (in German) as INA - Integrierter Netz Analysator - Version 2.1 - Handbuch, number 101 in Informatik-Berichte, Humboldt-Universität zu Berlin, Berlin, Germany, April 1998. Current version 2.2 (in English) available as http://www.informatik.hu-berlin.de/lehrstuehle/automaten/ina/ HTML, April 1999.

[Sch97] Dirk Schwanke: *Probleme des Zusammenwirkens von Bestandteilen komplexer Steuerungen und ihre netztheoretische Interpretation.* Diplomarbeit, Otto-von-Guerike Universität Magdeburg, Lehrstuhl Steuerungstechnik, 1997.

[Sch99] Karsten Schmidt: *Stubborn Sets for Standard Properties.* In Donatelli, Susanna and Kleijn, Jetty (editors): *Lecture Notes in Computer Science: Application and Theory of Petri Nets 1999, 20th International Confer-*

ence, ICATPN'99, Williamsburg, Virginia, USA, volume 1630, pages 46–65. Springer-Verlag, June 1999.

[Sch00a] Karsten SCHMIDT: *How to calculate symmetries of Petri nets. Acta Informatica*, 36(7): 545–590, 2000. Technical report MATH-AL-8-1997, Dresden, University of Technology, 1997.

[Sch00b] Karsten SCHMIDT: *Integrating Low Level Symmetries into Reachability Analysis.* In Susanne GRAF and Michael I. SCHWARTZBACH (editors): *Tools and Algorithms for Construction and Analysis of Systems, 6th International Conference, TACAS 2000, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS 2000, Berlin, Germany, March 25 - April 2, 2000, Proceedings*, volume 1785 of *Lecture Notes in Computer Science*, pages 315–330. Springer, 2000.

[SH97] Peter H. STARKE and Hans-Michael HANISCH: *Analysis of Signal/Event Nets.* In *Proc. of IEEE 6th International Conference on Emerging Technologies and Factory Automation*, pages 253–257, 1997.

[SK91] R. S. SREENIVAS and B. H. KROGH: *On condition/event systems with discrete state realizations. Discrete Event Dynamic Systems: Theory and Applications*, 2(1): 209–236, 1991.

[SLH98] Dirk SCHWANKE, Arndt LÜDER, and Hans-Michael HANISCH: *Dynamic Behaviour and the Deadlock-Trap Property of Signal/Event Nets.* In Hans-Dieter BURKHARD, Ludwik CZAJA, and Peter H. STARKE (editors): *Proceedings of the CS&P'98 Workshop*, number 110 in Informatik-Berichte, pages 215–220, Berlin, Germany, September 1998. Humboldt-Universität zu Berlin.

[SR00] Peter H. STARKE and Stephan ROCH: *INA et al.* In Kjeld H. MORTENSEN (editor): *Tool Demonstrations 21st International Conference on Application and Theory of Petri Nets*, pages 51–56, Aarhus, Denmark, June 2000. Department of Computer Science, University of Aarhus.

[Sta97] Peter H. STARKE: *Signal-Event Nets.* In Hans-Dieter BURKHARD, Ludwik CZAJA, and Peter H. STARKE (editors): *Proceedings of the CS&P'97 Workshop*, pages 124–130, Warsaw, Poland, October 1997.

[Sta98] Peter H. STARKE: *Invariants Of Signal-Event Nets.* In Hans-Dieter BURKHARD, Ludwik CZAJA, and Peter H. STARKE (editors): *Proceedings of the CS&P'98 Workshop*, number 110 in Informatik-Berichte, pages 245–256, Berlin, Germany, September 1998. Humboldt-Universität zu Berlin.

[Sta99] Peter H. STARKE: *Structural Analysis of Signal-Event Systems.* In Hans-Dieter BURKHARD, Ludwik CZAJA, Hung Son NGUYEN, and Peter H. STARKE (editors): *Proceedings of the CS&P'99 Workshop*, pages 236–250, Warsaw, Poland, September 1999.

[Val91] Antti VALMARI: *Stubborn Sets for Reduced State Space Generation.* In G. ROZENBERG (editor): *Proc. Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 491–515, Berlin, GermanyBerlin, Heidelberg, November 1991. Springer-Verlag.

[Val94] Antti VALMARI: *State of the Art Report: Stubborn Sets. Petri Net Newsletter*, 46: 6–14, April 1994. Gesellschaft für Informatik, Special Interest Group on Petri Nets and Related System Models.

[Val98] Antti VALMARI: *The State Explosion Problem. Lecture Notes in Computer Science: Lectures on Petri Nets I: Basic Models*, 1491: 429–528, 1998.

[Var93] Kimmo VARPAANIEMI: *Efficient Detection of Deadlocks in Petri Nets.* Series A: Research Report 26, Helsinki University of Technology, Digital Systems Laboratory, Espoo, Finland, October 1993.

[VH99] Valeriy VYATKIN and Hans-Michael HANISCH: *A Modeling Approach for Verification of IEC1499 Function Blocks using Net Condition/Event Systems.* In Josep M. FUERTES (editor): *Proceedings of the ETFA'99 Workshop*, pages 261–270, Barcelona, Spain, September 1999.

[VHSR00] Valeriy VYATKIN, Hans-Michael HANISCH, Peter H. STARKE, and Stephan ROCH: *Further Development of Formalisms for Modelling and Verification of IEC 1499 Distributed Control Applications.* In *Proceedings of Verteilte Automatisierung*, pages 72–79, Magdeburg, Germany, March 2000.

[Vog97] Walter VOGLER: *Partial Order Semantics and Read Arcs.* Technical report 1997-1, Universität Augsburg, Institut für Informatik,, 1997.

[VSY98] Walter VOGLER, Alex SEMENOV, and Alex YAKOVLEV: *Unfolding and Finite Prefix for Nets with Read Arcs.* In D. SANGIORGI and R. de SIMONE (editors): *Proceedings 9th International Conference on Concurrency Theory (CONCUR'98), Nice, France, September 1998*, volume 1466 of *Lecture Notes in Computer Science*, pages 501–516, Berlin, Heidelberg, 1998. Springer-Verlag.

Index