COMPUTING SCIENCE

ITL Semantics of Composite Petri Nets

Zhenhua Duan, Hanna Klaudel and Maciej Koutny

TECHNICAL REPORT SERIES

No. CS-TR-1296 November 2011

TECHNICAL REPORT SERIES

No. CS-TR-1296 November, 2011

ITL Semantics of Composite Petri Nets

Z. Duan, H. Klaudel and M. Koutny

Abstract

Interval Temporal Logic (ITL) and Petri nets are two well developed formalisms for the specification and analysis of concurrent systems. ITL allows one to specify both the system design and correctness requirements within the same logic based on intervals (sequences of states). As a result, verification of system properties can be carried out by checking that the formula describing a system implies the formula describing a requirement. Petri nets, on the other hand, have state based semantics and allow for a direct expression of causality aspects in system behaviour. As a result, verification of system properties can be carried out using partial order reductions or invariant based techniques. In this paper, we aim at providing a basic semantical link between ITL and Petri nets so that, in particular, one would be able to use both kinds of verification techniques of system properties.

Printed and published by Newcastle University,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

DUAN, Z., KLAUDEL, H., KOUTNY, M.

ITL Semantics of Composite Petri Nets
[By] Z. Duan, H. Klaudel, M. Koutny
Newcastle upon Tyne: Newcastle University: Computing Science, 2011.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1296)

Added entries

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series. CS-TR-1296

Abstract

Interval Temporal Logic (ITL) and Petri nets are two well developed formalisms for the specification and analysis of concurrent systems. ITL allows one to specify both the system design and correctness requirements within the same logic based on intervals (sequences of states). As a result, verification of system properties can be carried out by checking that the formula describing a system implies the formula describing a requirement. Petri nets, on the other hand, have state based semantics and allow for a direct expression of causality aspects in system behaviour. As a result, verification of system properties can be carried out using partial order reductions or invariant based techniques. In this paper, we aim at providing a basic semantical link between ITL and Petri nets so that, in particular, one would be able to use both kinds of verification techniques of system properties.

About the authors

Zhenhua Duan is a Professor at the Xidian University, Xian, China.

Hanna Klaudel is a Professor at the University of Evry, France.

Maciej Koutny obtained his MSc (1982) and PhD (1984) from the Warsaw University of Technology. In 1985 he joined the then Computing Laboratory of the University of Newcastle upon Tyne to work as a Research Associate. In 1986 he became a Lecturer in Computing Science at Newcastle, and in 1994 was promoted to an established Readership at Newcastle. In 2000 he became a Professor of Computing Science.

Suggested keywords

ITL
PETRI NET
BOX ALGEBRA
COMPOSITION
SEMANTICS

ITL Semantics of Composite Petri Nets

Zhenhua Duan[1], Hanna Klaudel[2], and Maciej Koutny[3]

[1] School of Computer Science and Engineering
Xidian University, Xi'an, P.R.China
zhenhua_d@yahoo.com
[2] IBISC, Université d'Évry, 91000 Évry, France
klaudel@ibisc.univ-evry.fr
[3] School of Computing Science, University of Newcastle
Newcastle upon Tyne, NE1 7RU, United Kingdom
maciej.koutny@newcastle.ac.uk

Abstract. Interval Temporal Logic (ITL) and Petri nets are two well developed formalisms for the specification and analysis of concurrent systems. ITL allows one to specify both the system design and correctness requirements within the same logic based on intervals (sequences of states). As a result, verification of system properties can be carried out by checking that the formula describing a system implies the formula describing a requirement. Petri nets, on the other hand, have state based semantics and allow for a direct expression of causality aspects in system behaviour. As a result, verification of system properties can be carried out using partial order reductions or invariant based techniques. In this paper, we aim at providing a basic semantical link between ITL and Petri nets so that, in particular, one would be able to use both kinds of verification techniques of system properties.

Keywords: ITL, Petri net, box algebra, composition, semantics.

1 Introduction

Temporal logics [8, 3] and Petri nets [13] are two different but, in many yet complementary formalisms for the specification and analysis of concurrent systems. A temporal logic, such as Interval Temporal Logic (ITL) [10, 12], allows one to specify both the system design and correctness requirements within the same logic framework based on sequences of global states. As a result, verification of a requirement captured by logic formula ϕ for a concurrent system expressed as logic formula ψ can be done by checking that the implication $\psi \supset \phi$ holds true. Petri nets, on the other hand, which are a graphical model with semantics based on local states allow, e.g., for a direct expression of causality aspects in system behavior. As a result, verification of system properties can be done using model checking techniques based on partial order reductions [18], or invariant techniques [17] based on graph structure of nets.

In this paper, we aim at providing a basic semantical link between ITL and Petri nets so that one would be able to use both kinds of verification techniques

of system properties. Finding such a link is bound to be difficult as temporal logics and Petri nets have strikingly different nature. The first step, therefore, would be an identification of a sufficiently expressive temporal logic and class of Petri nets which could be related in a clear and direct way. Intuitively, the difficulties encountered when matching temporal logics and Petri nets stem from the fact that the former are structured using composition operators, whereas the latter, in general, are not.

A notable exception is the Box Algebra (BA) [1] which supports Petri nets built using composition operators inspired by common programming constructs such as sequence, iteration, parallel composition and choice. Each Petri net $\mathsf{box}(E)$ is derived from a box expression E using a compositional mapping $\mathsf{box}(.)$. It is, therefore, natural to seek a temporal logic matching or supporting this particular set of composition operators. When looking at the existing temporal logics from this point of view, it was remarkable to realise that ITL (Interval Temporal Logic) is based on almost exactly the same set of programming constructs. As a result, we set out to explore the possibility of building a semantical bridge between temporal logics and Petri nets using two concrete formalisms, viz. ITL and BA. In this way, one should ultimately be able to take advantage of the individual strengths of these two formalisms, such as the analysis of systems with infinite data domains and fairness-related properties [16] for ITL, and unfolding based partial order model checking and invariant analysis for BA.

In concrete terms, our aim is to solve the following problem:

> *Given a* BA *expression E and a logic formula ψ expressed in* ITL *syntax, provide a translation $itl(E)$ into* ITL *such that* $\mathsf{box}(E)$ *satisfies ψ* iff *$itl(E) \supset \psi'$ holds true, where ψ' is a suitably adjusted ψ.*

In this paper, we provide a syntax-driven translation $itl(.)$ for the core BA [1] syntax comprising parallel composition, sequence, choice, synchronisation and iteration, but without considering data variables. Whereas translating the other control-flow operators is relatively straightforward, doing the same for synchronisation is much more involved and we rely here and adapt some ideas first formulated in [2] for the case of interprocess communication. We first consider a simpler case of CCS-like binary synchronisation [9], and then extend this to a general multi-way synchronisation scheme [1]. The main result is the soundness of the proposed translation.

Throughout the paper $\mathbb{N}$ denotes all positive integers, $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ and $\mathbb{N}_\omega = \mathbb{N}_0 \cup \{\omega\}$, where ω denotes the first transfinite ordinal. We extend to $\mathbb{N}_\omega$ the standard arithmetic comparison operators, assuming that $\omega = \omega$ and $n < \omega$, for all $n \in \mathbb{N}_0$. Moreover, we define $\preceq$ as $\leq \setminus \{(\omega, \omega)\}$. The concatenation operator for sequences will be denoted by $\circ$, and we will denote $\varnothing^\omega = \{\varnothing\varnothing \ldots\}$ and $\varnothing^* = \{\epsilon, \varnothing, \varnothing\varnothing, \ldots\}$ (i.e., $\varnothing^\omega$ comprises a single infinite sequence, and $\varnothing^*$ an infinite number of finite sequences).

2 Box algebra with one-to-one communication

We first consider SBA which is a simple sub-model of Box Algebra [1]. In particular, we will allow only one-to-one communication between concurrent sequential processes.

We assume a set of communication actions, each such action a having a unique conjugate action $\widehat{a}$ satisfying $\widehat{a} \neq a$ and $\widehat{\widehat{a}} = a$. Moreover, we allow synchronisation actions of the form $\tau_{\{a,\widehat{a}\}}$ representing simultaneous execution of two conjugate communication actions, a and $\widehat{a}$.

The syntax of SBA expressions E and sequential expressions S is as follows:

$$\begin{array}{lcl} S & ::= & \mathsf{stop} \mid a \mid S\,;S' \mid [S \circledast S' \circledast S''] \mid S \,\Box\, S' \\ E & ::= & (S_1 \parallel S_2 \parallel \ldots \parallel S_k)\,\mathsf{sco}\,A \end{array}$$

where a is a communication action, and A is a set of communication actions including conjugates. We assume that in an SBA expression

$$E = (S_1 \parallel S_2 \parallel \ldots \parallel S_k)\,\mathsf{sco}\,A \tag{1}$$

we have $A_i \cap A_j = \varnothing$, for $i \neq j$, where each A_i is the set of communication actions occurring in S_i.

In the above syntax, stop stands for a blocked process, $S \,\Box\, S'$ for choice composition, $S\,;S'$ for sequential composition, $[S \circledast S' \circledast S'']$ for a loop (with an initial part S, iterated part S', and terminal part S''), and finally $(S_1 \parallel S_2 \parallel \ldots \parallel S_k)\,\mathsf{sco}\,A$ for parallel composition of k sequential processes. The scoping part, $\mathsf{sco}\,A$, of the expression E in (1) synchronises conjugate communication actions in A belonging to different sequential processes, leaving the non-synchronised actions (i.e., those not belonging to A) intact.

Figure 1 shows as SBA expression $(S_1 \parallel S_2 \parallel S_3)\,\mathsf{sco}\,A$ modelling a system consisting of two one-place buffer processes, S_1 and S_2, and a merge process, S_3. Buffers S_1 and S_2 respectively use actions a and b to receive signals which are then send off to the merge process using the c and d actions. The merge process uses the conjugate actions, $\widehat{c}$ and $\widehat{d}$, to receive forwarded signals which are then passed on using the e signal. The scoping part $\mathsf{sco}\,A$ effects interprocess communication. (Note that the actions c, $\widehat{c}$, d and $\widehat{d}$ cannot be executed individually.) The three processes are started up using the actions $start_i$ which are executed once.

Remark 1. The syntax of SBA expression E in (1) incorporates two specific restrictions:

- *Only communication actions are used within sequential sub-expressions.* This is not a real problem as local (non-synchronised) actions are basically those communication actions which do not appear in A, and silent actions may be simulated by fresh communication actions.

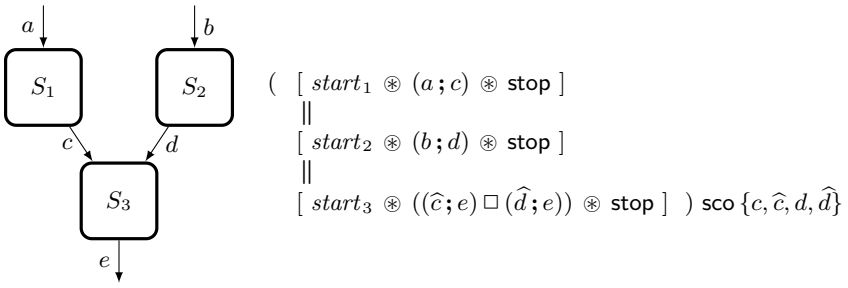

Fig. 1. Two one-place buffers and merge processes.

– *No action appears in more than one sequential expression S_i.*
If this does not hold, we can take any communication action $a \in A_i \cap A$ and then replace:
  - each occurrence of a by $\mathsf{stop} \,\Box\, a_{i,j_1} \,\Box\, \ldots \,\Box\, a_{i,j_l}$ within S_i, and
  - a by $a_{i,j_1}, \ldots, a_{i,j_l}$ within A,

  where $\{j_1, \ldots, j_l\} = \{j \mid j \neq i \wedge \widehat{a} \in A_j\}$.

Box algebra semantics

The semantics of SBA expressions is given through a mapping into Petri nets called boxes.

A box is a tuple $\Sigma = (P, T, F, \ell, M_0)$ where P and T are disjoint finite sets of respectively places and transitions; $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation; ℓ is a labelling function for places and transitions such that $\ell(p) \in \{\mathsf{e}, \mathsf{i}, \mathsf{x}\}$, for every place $p \in P$, and $\ell(t)$ is an action, for every transition $t \in T$; and $M_0 \subseteq P$ is an (initial) marking. In general, any subset of P is a marking.

We adopt the standard rules about representing nets as directed graphs. If the labelling of a place p is e, i or x, then p is an entry, internal or exit place, respectively.

For every place (transition) x, we use ${}^\bullet x$ to denote its pre-set, i.e., the set of all transitions (places) y such that there is an arc from y to x, that is, $(y, x) \in F$. The post-set $x^\bullet$ is defined in a similar way. The pre- and post-set notation extends in the usual way to sets R of places and transitions, e.g., ${}^\bullet R = \bigcup_{r \in R} {}^\bullet r$. By convention, ${}^\bullet\Sigma$ and $\Sigma^\bullet$ denote respectively the sets of entry and exit places of Σ.

We now introduce operators on boxes corresponding to operators used in SBA expressions. Let Σ_i (for $i = 1, 2, 3$) be boxes with disjoint sets of nodes satisfying $|{}^\bullet\Sigma_i| = |\Sigma_i{}^\bullet| = 1$ and $M_{0i} = \varnothing$. Then we have the following (below p and p' are fresh places):

– $\Sigma_1 \,\Box\, \Sigma_2 = (P, T, F, \ell, \varnothing)$ where:

$$\begin{aligned} P &= P_1 \cup P_2 \setminus ({}^\bullet\Sigma_1 \cup \Sigma_1{}^\bullet \cup {}^\bullet\Sigma_2 \cup \Sigma_2{}^\bullet) \cup \{p, p'\} \\ T &= T_1 \cup T_2 \\ F &= (F_1 \cup F_2)|_{(P\times T)\cup(T\times P)} \cup \{p\} \times ({}^\bullet\Sigma_1 \cup {}^\bullet\Sigma_2)^\bullet \cup \\ &\qquad\qquad {}^\bullet(\Sigma_1{}^\bullet \cup \Sigma_2{}^\bullet) \times \{p'\} \\ \ell &= (\ell_1 \cup \ell_2)|_{P\cup T} \cup \{p \mapsto \mathsf{e}, p' \mapsto \mathsf{x}\} \,. \end{aligned}$$

– $\Sigma_1 \,;\, \Sigma_2 = (P, T, F, \ell, \varnothing)$ where:

$$\begin{aligned} P &= P_1 \cup P_2 \setminus (\Sigma_1{}^\bullet \cup {}^\bullet\Sigma_2) \cup \{p\} \\ T &= T_1 \cup T_2 \\ F &= (F_1 \cup F_2)|_{(P\times T)\cup(T\times P)} \cup (\{p\} \times ({}^\bullet\Sigma_2)^\bullet) \cup ({}^\bullet(\Sigma_1{}^\bullet) \times \{p\}) \\ \ell &= (\ell_1 \cup \ell_2)|_{P\cup T} \cup \{p \mapsto \mathsf{i}\} \,. \end{aligned}$$

– $[\Sigma_1 \circledast \Sigma_2 \circledast S_3] = (P, T, F, \ell, \varnothing)$ where:

$$\begin{aligned} P &= P_1 \cup P_2 \cup P_3 \setminus (\Sigma_1{}^\bullet \cup {}^\bullet\Sigma_2 \cup \Sigma_2{}^\bullet \cup {}^\bullet\Sigma_3) \cup \{p\} \\ T &= T_1 \cup T_2 \cup T_3 \\ F &= (F_1 \cup F_2 \cup F_3)|_{(P\times T)\cup(T\times P)} \cup {}^\bullet(\Sigma_1{}^\bullet \cup \Sigma_2{}^\bullet) \times \{p\} \cup \\ &\qquad\qquad \{p\} \times ({}^\bullet\Sigma_2 \cup {}^\bullet\Sigma_3)^\bullet \\ \ell &= (\ell_1 \cup \ell_2 \cup \ell_3)|_{P\cup T} \cup \{p \mapsto \mathsf{i}\} \,. \end{aligned}$$

Moreover, for any box Σ, $\overline{\Sigma}$ is Σ with the initial marking set to ${}^\bullet\Sigma$.

The semantics of a box Σ is given through its step sequences. A set of transitions U, called a step, is enabled at a marking M if ${}^\bullet U \subseteq M$ and ${}^\bullet u \cap {}^\bullet t = \varnothing$, for all distinct $t, u \in U$. An enabled step U can be executed leading to a marking M' given by $M' = M \setminus {}^\bullet U \cup U^\bullet$. We denote this by $M[U\rangle M'$.

As far as a box Σ is concerned, only step sequences which start from its default initial marking ${}^\bullet\Sigma$ need to be considered. We will assume that each such step sequences is infinite which is a harmless requirement as any finite step sequence can be extended by an infinite sequence of empty steps (note that $M[\varnothing\rangle M$ for every marking M). In addition, we will single out a set of finite step sequences which lead from the default initial marking ${}^\bullet\Sigma$ to the default final marking $\Sigma^\bullet$. Intuitively, each such step sequence corresponds to a terminated execution of the box.

A step sequence of a box Σ is any infinite sequence of steps $\gamma = U_1 U_2 \ldots$ such that there are markings $M_1, M_2 \ldots$ satisfying

$${}^\bullet\Sigma[U_1\rangle M_1[U_2\rangle M_2 \ldots \,.$$

We denote this by $\gamma \in \mathit{step}(\Sigma)$. Moreover, a terminated step sequence of Σ is a finite sequence of steps $\gamma = U_1 \ldots U_m$ such that there are markings $M_1, \ldots, M_{m-1}$ satisfying

$${}^\bullet\Sigma[U_1\rangle M_1[U_2\rangle \ldots M_{m-1}[U_m\rangle \Sigma^\bullet \,.$$

We denote this by $\gamma \in tstep(\Sigma)$.

Step sequences built of sets of transitions are low-level descriptions of executed behaviours. A more abstract (and practically relevant) view is provided by step sequences built of steps of labels of executed transitions. Hence, for any finite or infinite step sequence γ as defined above, we will use $\ell(\gamma)$ to denote a sequence of multisets of labels obtained from γ by replacing each step U_i by the multiset Γ_i of labels of the transitions belonging to U. We then define:

$$lstep(\Sigma) = \ell(step(\Sigma)) \quad \text{and} \quad ltstep(\Sigma) = \ell(tstep(\Sigma)) \ .$$

From box expressions to boxes

We define a mapping box from SBA expressions to boxes compositionally. First, for the blocked expression stop and any communication action a:

$$\begin{aligned} \mathsf{box}(a) &= (\{p, p'\}, \{t_a\}, \{(p, t_a), (t_a, p)\}, \{p \mapsto \mathsf{e}, t_a \mapsto a, p' \mapsto \mathsf{x}\}, \varnothing) \\ \mathsf{box}(\mathsf{stop}) &= (\{p, p'\}, \varnothing, \varnothing, \{p \mapsto \mathsf{e}, p' \mapsto \mathsf{x}\}, \varnothing) \end{aligned}$$

and, for any sequential expressions S_1, S_2 and S_3:

$$\begin{aligned} \mathsf{box}(S_1 \,;\, S_2) &= \mathsf{box}(S_1) \,;\, \mathsf{box}(S_2) \\ \mathsf{box}(S_1 \,\Box\, S_2) &= \mathsf{box}(S_1) \,\Box\, \mathsf{box}(S_2) \\ \mathsf{box}([S_1 \circledast S_2 \circledast S_3]) &= [\mathsf{box}(S_1) \circledast \mathsf{box}(S_2) \circledast \mathsf{box}(S_3)] \end{aligned}$$

Then, for an SBA expression E as in (1), the net $\mathsf{box}(E)$ is obtained by:

- creating Σ which is a disjoint union of $\mathsf{box}(S_i)$, for $i = 1, \ldots, k$.
- creating Σ' from Σ by adding fresh transitions $t = t_{\{u,v\}}$, where u is a transition in $\mathsf{box}(S_i)$ and v is a transition in $\mathsf{box}(S_j)$ $(i \neq j)$, $\ell_i(u) = a \in A$ and $\ell_i(u) = \widehat{a}$. The label of t is $\tau_{\{a,\widehat{a}\}}$, and it inherits the connectivity of u and v, i.e., ${}^\bullet t = {}^\bullet u \cup {}^\bullet v$ and $t^\bullet = u^\bullet \cup v^\bullet$.
- deleting all transition labelled by the communication actions in A.

We also define, for any SBA expression F:

$$lstep(F) = lstep(\overline{\mathsf{box}(F)}) \quad \text{and} \quad ltstep(F) = ltstep(\overline{\mathsf{box}(F)}) \ .$$

Behaviours of boxes obtained through the above translation from SBA expressions exhibit clear compositional properties [1]. Figure 2 provides a full characterisation of the behaviours of sequential SBA expressions whereas for concurrent SBA expressions we have the following.

Proposition 1. *Let E be an SBA expression as in (1). Then:*

- *a finite sequence $\gamma = \Gamma_1 \ldots \Gamma_m$ belongs to $ltstep(E)$ iff for $j = 1, \ldots, k$:*

$$\Gamma_1^j \ldots \Gamma_m^j \in ltstep(S_j)$$

where $\Gamma_i^j = \Gamma_i|_{A_j} + \{a \in A_j \mid \tau_{\{a,\widehat{a}\}} \in \Gamma_i\}$, for $i = 1, \ldots, m$.

– *an infinite sequence* $\gamma = \Gamma_1\Gamma_2\ldots$ *belongs to* $lstep(E)$ *iff for* $j = 1, \ldots, k$*:*

$$\Gamma_1^j\Gamma_2^j\ldots \in lstep(S_j)$$

where $\Gamma_i^j = \Gamma_i|_{A_j} + \{a \in A_j \mid \tau_{\{a,\widehat{a}\}} \in \Gamma_i\}$*, for* $i = 1, 2, \ldots$.

Note: $+$ *and* $|_{A_j}$ *respectively denote multiset sum and restriction.* □

Figure 3 illustrates the construction of a box representing an SBA expression of Figure 1.

3 Interval Temporal Logic

We now provide the syntax and semantics of a small fragment of ITL. The chosen fragment includes only those constructs (basic and derived) which are used in the subsequent translation of SBA expressions.

The formulas of the fragment of the ITL logic we need are defined below, where V is a set of boolean variables, $v \in V$ and $V' \subseteq V$:

$$\phi ::= \mathsf{flip}(v) \mid \mathsf{keep}(V') \mid \phi \wedge \phi' \mid \phi \vee \phi' \mid \phi\,\widehat{;}\,\phi' \mid \phi^* \mid \mathsf{inf}$$

Note that $\mathsf{flip}(v)$ inverts the value of boolean variable v over a unit interval, $\mathsf{keep}(V')$ keeps the value of variables in V' over a unit interval, $\widehat{;}$ is a weak version of the standard sequential composition operator (denoted by ; and called chop), and inf indicates an infinite interval.

A state is a mapping which assigns values to the (boolean) variables V, and an interval σ is a possibly infinite non-empty sequence of states. Its length, $|\sigma|$, is ω if σ is infinite, and otherwise its number of states minus 1. To simplify

$$\begin{aligned}
ltstep(\mathsf{stop}) &= \varnothing \\
ltstep(a) &= \varnothing^* \circ \{\{a\}\} \circ \varnothing^* \\
ltstep(S_1\,;\,S_2) &= ltstep(S_1) \circ ltstep(S_2) \\
ltstep(S_1 \,\Box\, S_2) &= ltstep(S_1) \cup ltstep(S_2) \\
ltstep([S_1 \circledast S_2 \circledast S_3]) &= ltstep(S_1) \circ ltstep(S_2)^* \circ ltstep(S_3) \\
lstep(\mathsf{stop}) &= \varnothing^\omega \\
lstep(a) &= \varnothing^\omega \cup \varnothing^* \circ \{\{a\}\} \circ \varnothing^\omega \\
lstep(S_1\,;\,S_2) &= lstep(S_1) \cup ltstep(S_1) \circ lstep(S_2) \\
lstep(S_1 \,\Box\, S_2) &= lstep(S_1) \cup lstep(S_2) \\
lstep([S_1 \circledast S_2 \circledast S_3]) &= lstep(S_1) \cup \\
&\quad ltstep(S_1) \circ ltstep(S_2)^* \circ (lstep(S_2) \cup lstep(S_3))
\end{aligned}$$

Fig. 2. Properties of behaviours of sequential SBA expressions.

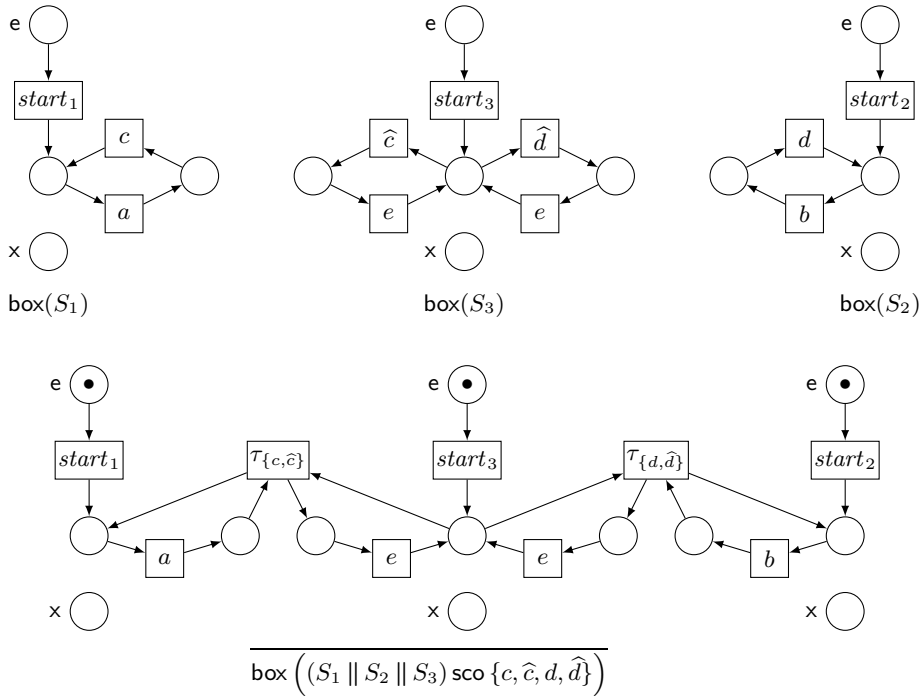

Fig. 3. Top: unsynchronised unmarked boxes for S_1, S_2 and S_3; bottom: complete box semantics of the three-process system with the default initial marking. All the i labels of internal places are omitted.

definitions, we will denote σ as $\langle \sigma_0, \sigma_1, \ldots, \sigma_{|\sigma|} \rangle$, where $\sigma_{|\sigma|}$ is undefined if σ is infinite. With such a notation, for $0 \leq i \preceq j \leq |\sigma|$:

$$\sigma_{i..j} = \langle \sigma_i, \ldots, \sigma_j \rangle \quad \text{and} \quad \sigma^i = \langle \sigma_0, \ldots, \sigma_i \rangle \quad \text{and} \quad \sigma^{(i)} = \langle \sigma_i, \ldots, \sigma_{|\sigma|} \rangle$$

The meaning of formulas is given by the satisfaction relation defined as follows:

- $\sigma \models \mathsf{flip}(v)$ iff $|\sigma| = 1$ and $\sigma_1(v) = \neg\sigma_0(v)$.
- $\sigma \models \mathsf{keep}(\{v_1, \ldots, v_m\})$ iff $|\sigma| = 1$ and $\sigma_1(v_i) = \sigma_0(v_i)$, for $i = 1, \ldots, m$.
- $\sigma \models \phi \vee \phi'$ iff $\sigma \models \phi$ or $\sigma \models \phi'$.
- $\sigma \models \phi \wedge \phi'$ iff $\sigma \models \phi$ and $\sigma \models \phi'$.
- $\sigma \models \phi \mathbin{\widehat{;}} \phi'$ iff one of the following holds:
  - $|\sigma| = \omega$ and $\sigma \models \phi$.
  - there is $r \preceq |\sigma|$ and $\sigma^r \models \phi$ and $\sigma^{(r)} \models \phi'$.
- $\sigma \models \phi^*$ iff one of the following holds:
  - $|\sigma| = 0$.
  - there are $0 = r_0 \leq r_1 \leq \ldots \leq r_{n-1} \preceq r_n = |\sigma|$ such that, for all $1 \leq l \leq n$, $\sigma_{r_{l-1}..r_l} \models \phi$.

- $|\sigma| = \omega$ and there are infinitely many integers $0 = r_0 \leq r_1 \leq \ldots$ such that $\lim_{i\to\infty} r_i = \omega$ and for all $l \geq 1$, $\sigma_{r_{l-1}..r_l} \models \phi$.

– $\sigma \models \mathsf{inf}$ iff $|\sigma| = \omega$.

Note that $\phi \,\widehat{;}\, \phi'$ is equivalent to the formula $\phi \wedge \mathsf{inf} \vee \phi \,;\, \phi'$.

4 From simple box algebra to ITL

We now present a translation from an SBA expression E defined as in (1) into semantically equivalent ITL formula.

A key idea inspired by [2] is to represent each of the actions a appearing in E by a separate boolean variable v_a, and then to model an execution of a by the change of the value of v_a. In the case of synchronisation between a and $\widehat{a}$, the two corresponding variables, v_a and $v_{\widehat{a}}$, have to change their values simultaneously.

In what follows, the set of variables corresponding to each A_i (actions occurring within S_i) are denoted by V_i, and no other variables are used. The translation is then given by:

$$itl(E) \;=\; itl_1(S_1) \wedge \ldots \wedge itl_k(S_k)$$

where we have the following (below $i = 1, \ldots, k$, $a \in A_i \cap A$ and $b \in A_i \setminus A$):

$$\begin{aligned}
itl_i(\mathsf{stop}) &= \mathsf{keep}(V_i)^* \wedge \mathsf{inf} \\
itl_i(a) &= \mathsf{keep}(V_i)^* \,\widehat{;}\, \big(\mathsf{keep}(V_i \setminus \{v_a\}) \wedge \mathsf{flip}(v_a) \wedge \mathsf{flip}(v_{\widehat{a}})\big) \,\widehat{;}\, \mathsf{keep}(V_i)^* \\
itl_i(b) &= \mathsf{keep}(V_i)^* \,\widehat{;}\, \big(\mathsf{keep}(V_i \setminus \{v_b\}) \wedge \mathsf{flip}(v_b)\big) \,\widehat{;}\, \mathsf{keep}(V_i)^* \\
itl_i(S \,;\, S') &= itl_i(S) \,\widehat{;}\, itl_i(S') \\
itl_i(S \,\Box\, S') &= itl_i(S) \vee itl_i(S') \\
itl_i([S \circledast S' \circledast S'']) &= itl_i(S) \,\widehat{;}\, itl_i(S')^* \,\widehat{;}\, itl_i(S'')
\end{aligned}$$

Intuitively, the value of a variable v_a is kept unchanged, unless we simulate an execution of action a which results in flipping the value of v_a. Moreover, synchronisation involving a and $\widehat{a}$ flips the values of both v_a and $v_{\widehat{a}}$. In this way, executions carried out in concurrent components are synchronised.

With each interval σ satisfying $\sigma \models itl(E)$, we associate a sequence of multisets $\gamma_\sigma = \Gamma_1 \ldots \Gamma_{|\sigma|}$, were each Γ_j is defined as follows:

$$\Gamma_j = \{a \mid a \notin A \wedge \sigma_{j-1}(v_a) \neq \sigma_j(v_a)\} + \{\tau_{\{a,\widehat{a}\}} \mid a \in A \wedge \sigma_{j-1}(v_a) \neq \sigma_j(v_a)\} \,.$$

Similarly, with each interval σ satisfying $\sigma \models itl(S_i)$, we associate a sequence of multisets $\gamma_\sigma = \Gamma_1 \ldots \Gamma_{|\sigma|}$, were each Γ_j is defined as follows:

$$\Gamma_j = \{a \in A_i \mid \sigma_{j-1}(v_a) \neq \sigma_j(v_a)\} \,.$$

We then define, for any ITL formula ϕ appearing on the r.h.s. of the above translation:

$$ltstep(\phi) = \{\gamma_\sigma \mid \sigma \models \phi \wedge |\sigma| < \omega\} \quad \text{and} \quad lstep(\phi) = \{\gamma_\sigma \mid \sigma \models \phi \wedge |\sigma| = \omega\} \; .$$

Note that $ltstep(\phi) \circ \varnothing^\omega \subseteq lstep(\phi)$.

Proposition 2. *Let S be a sequential* SBA *expression. Then:*

$$lstep(itl(S)) = lstep(S) \quad \textit{and} \quad ltstep(itl(S)) = ltstep(S) \; .$$

Proof. The proof proceeds by induction on the structure of S, using Figure 2 and the properties of the logic operators.

Case 1: $S = \mathsf{stop}$. Then:

$$\begin{aligned} ltstep(itl(\mathsf{stop})) &= \varnothing = ltstep(\mathsf{stop}) \\ lstep(itl(\mathsf{stop})) &= \varnothing^\omega = lstep(\mathsf{stop}) \; . \end{aligned}$$

Case 2: $S = a$. Then:

$$\begin{aligned} ltstep(itl(a)) &= \varnothing^* \circ \{\{a\}\} \circ \varnothing^* = ltstep(a) \\ lstep(itl(a)) &= \varnothing^\omega \cup \varnothing^* \circ \{\{a\}\} \circ \varnothing^\omega = lstep(a) \; . \end{aligned}$$

Case 3: $S = S_1 \,;\, S_2$. Then:

$$\begin{aligned} ltstep(itl(S_1 \,;\, S_2)) &= ltstep(itl(S_1)) \circ ltstep(itl(S_2)) \\ &= ltstep(S_1) \circ ltstep(S_2) \\ &= ltstep(S_1 \,;\, S_2) \\ lstep(itl(S_1 \,;\, S_2)) &= lstep(itl(S_1)) \cup ltstep(itl(S_1)) \circ lstep(itl(S_2)) \\ &= lstep(S_1) \cup ltstep(S_1) \circ lstep(S_2) \\ &= lstep(S_1 \,;\, S_2) \; . \end{aligned}$$

Case 4: $S = S_1 \,\square\, S_2$. Then:

$$\begin{aligned} ltstep(itl(S_1 \,\square\, S_2)) &= ltstep(itl(S_1)) \cup ltstep(itl(S_2)) \\ &= ltstep(S_1) \cup ltstep(S_2) \\ &= ltstep(S_1 \,\square\, S_2) \\ lstep(itl(S_1 \,\square\, S_2)) &= lstep(itl(S_1)) \cup lstep(itl(S_2)) \\ &= lstep(S_1) \cup lstep(S_2) \\ &= lstep(S_1 \,\square\, S_2) \; . \end{aligned}$$

Case 5: $S = [S_1 \circledast S_2 \circledast S_3]$. Then:

$$\begin{aligned}
ltstep(itl([S_1 \circledast S_2 \circledast S_3])) &= ltstep(itl(S_1)) \circ ltstep(itl(S_2))^* \circ \\
&\quad\ ltstep(itl(S_3)) \\
&= ltstep(S_1) \circ ltstep(S_2)^* \circ ltstep(S_3) \\
&= ltstep([S_1 \circledast S_2 \circledast S_3]) \\
lstep(itl([S_1 \circledast S_2 \circledast S_3])) &= lstep(itl(S_1)) \cup \\
&\quad\ ltstep(itl(S_1)) \circ ltstep(itl(S_2))^* \circ \\
&\quad\ (lstep(itl(S_2)) \cup lstep(itl(S_3))) \\
&= lstep(S_1) \cup \\
&\quad\ ltstep(S_1) \circ ltstep(S_2)^* \circ \\
&\quad\ (lstep(S_2) \cup lstep(S_3)) \\
&= ltstep([S_1 \circledast S_2 \circledast S_3])\ .
\end{aligned}$$

This completes the proof. □

Theorem 1. *Let E be an* SBA *expression as in (1). Then:*

$$lstep(itl(E)) = lstep(E) \quad \textit{and} \quad ltstep(itl(E)) = ltstep(E)\ .$$

Proof. Follows from Propositions 1 and 2, and the basic properties of the conjunction operator in ITL. Note that the assumption that no communication action occurs in more than one sequential expression S_i is crucial to demonstrate the result. □

What we have just presented is one of possible ways of translating SBA expressions into equivalent ITL formulas. Another possibility would be to enforce synchronisation between variables corresponding to conjugate communications globally using the $\Box$ operator of ITL:

$$itl(E) \ = \ itl_1(S_1) \wedge \ldots \wedge itl_k(S_k) \wedge \bigwedge_{a \in A} \Box v_a = v_{\widehat{a}}$$

and then to simplify the rest of the translation (below $b \in A_i$):

$$\begin{aligned}
itl_i(\mathsf{stop}) &= \mathsf{keep}(V_i)^* \wedge \mathsf{inf} \\
itl_i(b) &= \mathsf{keep}(V_i)^* \,\widehat{;}\, \big(\mathsf{keep}(V_i \setminus \{v_b\}) \wedge \mathsf{flip}(v_b)\big) \,\widehat{;}\, \mathsf{keep}(V_i)^* \\
itl_i(S\,;S') &= itl_i(S) \,\widehat{;}\, itl_i(S') \\
itl_i(S \,\Box\, S') &= itl_i(S) \vee itl_i(S') \\
itl_i([S \circledast S' \circledast S'']) &= itl_i(S) \,\widehat{;}\, itl_i(S')^* \,\widehat{;}\, itl_i(S'')
\end{aligned}$$

Yet another possibility would be to use a single variable $v_{\{a,\widehat{a}\}}$ to represent both a and $\widehat{a}$, leading to the following translation (below $a \in A_i \cap A$ and $b \in A_i \setminus A$):

$$\begin{array}{rcl}
itl(E) & = & itl_1(S_1) \wedge \ldots \wedge itl_k(S_k) \\
itl_i(\mathsf{stop}) & = & \mathsf{keep}(V_i)^* \wedge \mathsf{inf} \\
itl_i(a) & = & \mathsf{keep}(V_i)^* \mathbin{\widehat{;}} \\
 & & \big(\mathsf{keep}(V_i \setminus \{v_{\{a,\widehat{a}\}}\}) \wedge \mathsf{flip}(v_{\{a,\widehat{a}\}})\big) \mathbin{\widehat{;}} \mathsf{keep}(V_i)^* \\
itl_i(b) & = & \mathsf{keep}(V_i)^* \mathbin{\widehat{;}} \big(\mathsf{keep}(V_i \setminus \{v_b\}) \wedge \mathsf{flip}(v_b)\big) \mathbin{\widehat{;}} \mathsf{keep}(V_i)^* \\
itl_i(S\,;S') & = & itl_i(S) \mathbin{\widehat{;}} itl_i(S') \\
itl_i(S \,\Box\, S') & = & itl_i(S) \vee itl_i(S') \\
itl_i([S \circledast S' \circledast S'']) & = & itl_i(S) \mathbin{\widehat{;}} itl_i(S')^* \mathbin{\widehat{;}} itl_i(S'')
\end{array}$$

The latter alternative will be adopted in the translation of more complicated BA expressions described in the next section.

5 Box algebra with general synchronisation

We now consider BA equipped with a powerful operator of general synchronisation which subsumes a majority of those usually employed by process algebras.

Let L be a set of actions. A synchronisation relation is $\rho \subseteq L^+ \times L$ such that if $(a_1, \ldots, a_n, a) \in \rho$ then $a_1, \ldots, a_n$ represents actions which can be synchronised to yield a new action a. The general syntax of BA expressions E and sequential BA expressions S is as follows:

$$\begin{array}{rcl}
S & ::= & \mathsf{stop} \mid a \mid S\,;S' \mid [S \circledast S' \circledast S''] \mid S \,\Box\, S' \\
E & ::= & (S_1 \parallel S_2 \parallel \ldots \parallel S_k)[\rho]
\end{array}$$

where $a \in L$ is an action, and ρ is a synchronisation relation. There is no restriction on the presence of actions in different sequential subexpressions of E as in the case of SBA.

The translation for a sequential expression S is the same as in the simpler case. Moreover, given a BA expression

$$E = (S_1 \parallel S_2 \parallel \ldots \parallel S_k)[\rho] \tag{2}$$

we proceed as follows:

- For every $\xi = (a_1, \ldots, a_n, a) \in \rho$ we denote:

$$\Delta_\xi = \{k_1 \ldots k_n \mid \forall i \leq n : a_i \in A_{k_i} \wedge \forall i \neq j : k_i \neq k_j\}\,.$$

  Intuitively, Δ_ξ denotes all combinations of sequential processes which can potentially generate synchronisations captured by ξ.
- $\mathsf{box}(E)$ is obtained by:
  - creating Σ which is a disjoint union of $\mathsf{box}(S_i)$, for $i = 1, \ldots, k$.

$$\begin{array}{l}(\ [\ \mathit{start} \circledast (a\,;c) \circledast \mathsf{stop}\ ] \parallel \\ \ \ [\ \mathit{start} \circledast (f\,;e) \circledast \mathsf{stop}\ ] \parallel \\ \ \ [\ \mathit{start} \circledast (b\,;c') \circledast \mathsf{stop}\ ]\ )\,[\rho]\end{array} \qquad \begin{array}{l}\text{with } \rho = \{\ (\{\mathit{start}, \mathit{start}, \mathit{start}\}, \mathit{start}), \\ \qquad\qquad (\{c, f\}, \{c', f\}, g), \\ \qquad\qquad (a, a), (b, b), (e, e) \quad \}\end{array}$$

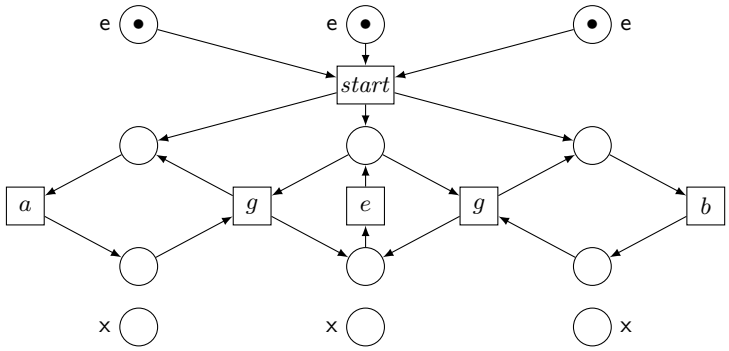

Fig. 4. A system with many-to-one communication.

- creating Σ' from Σ by adding a transition $t = t^{\xi}_{t_1 \ldots t_n}$ whenever:

$$\xi = (a_1, \ldots, a_n, a) \in \rho \ \text{ and } \ k_1 \ldots k_n \in \Delta_\xi$$

  and, for all $i \leq n$, t_i belongs to $\mathsf{box}(S_{k_i})$ and has the label a_i. The label of t is a, ${}^\bullet t = {}^\bullet t_1 \cup \ldots \cup {}^\bullet t_n$ and $t^\bullet = t_1{}^\bullet \cup \ldots \cup t_n{}^\bullet$.
- deleting all transitions coming from Σ.

The translation from the BA expressions to boxes follows the same pattern as in the case of SBA expressions, and its result is illustrated in Figure 4.

From box algebra to ITL

In the translation from BA expressions as in (2) to ITL, for every synchronisation pattern $\xi = (a_1, \ldots, a_n, a) \in \rho$, we will use distinct variables

$$\mathcal{V}_\xi = \{v^{\xi}_{k_1,\ldots,k_n} \mid k_1, \ldots, k_n \in \Delta_\xi\}$$

corresponding to different combinations of sequential sub-expressions which may realise synchronisations captured by ξ. Moreover, for every $i \leq k$ and $b \in A_i$,

$$V^i_b = \{v^{(a_1,\ldots,a_n,a)}_{k_1 \ldots k_n} \mid \exists j \leq n : k_j = i \wedge a_j = b\}$$

are all the variables corresponding to potential synchronisations in which b occurring in S_i can participate. The translation of BA expressions is then defined

in the following way (below $i = 1, \ldots, k$, $b \in A_i$ and $V_i = \bigcup_{b \in A_i} V_b^i$):

$$\begin{array}{rcl} itl(E) & = & itl_1(S_1) \wedge \ldots \wedge itl_k(S_k) \\ itl_i(\mathsf{stop}) & = & \mathsf{keep}(V_i)^* \wedge \mathsf{inf} \\ itl_i(b) & = & \mathsf{keep}(V_i)^* \,\widehat{;}\, \\ & & \big(\bigvee_{v \in V_b^i} \mathsf{keep}(V_i \setminus \{v\}) \wedge \mathsf{flip}(v)\big) \,\widehat{;}\, \mathsf{keep}(V_i)^* \\ itl_i(S\,;S') & = & itl_i(S) \,\widehat{;}\, itl_i(S') \\ itl_i(S \,\Box\, S') & = & itl_i(S) \vee itl_i(S') \\ itl_i([S \circledast S' \circledast S'']) & = & itl_i(S) \,\widehat{;}\, itl_i(S')^* \,\widehat{;}\, itl_i(S'') \end{array}$$

Note that a synchronisation can now involve more than two sequential subcomponents and is realised by a single variable which may appear in several different sub-formulas. The interpretation of satisfying intervals in terms of step sequences of multisets of actions requires only a small modification of the one used previously.

Properties of the newly defined translation are very much the same as in the simpler case. Crucially, we have the following.

Theorem 2. *Let E be a* BA *expression as in (2). Then:*

$$lstep(itl(E)) = lstep(E) \quad \textit{and} \quad ltstep(itl(E)) = ltstep(E)\ .$$

Proof. Similar as in the case of SBA expression. □

6 Conclusions

In the past, various kinds of logics have been used as formalism for expressing correctness properties of systems specified using Petri nets. When it comes to the relationship between logics and Petri net, we feel that the work on the connections between liner logic [5] and Place Transition nets was the closest one. However, the main concern there was the handling of multiple token occurrences in net places whereas boxes are safe nets. Another way in which logics and Petri nets were discussed was reported in [14] which provided a characterisation of Petri net languages in terms of second-order logical formulas.

The results presented in this paper demonstrate that one can develop a very close structural connection between BA and ITL. It is therefore important to further investigate the extent to which such a connection could be generalised and exploited. In particular, we plan to investigate what is the subset of ITL which can be modelled by BA. A longer time goal is the development of a hybrid verification methodology combining ITL and BA techniques. For example, sequential algorithms and data structures could be treated by ITL techniques [11], while intensive parallel or communicating aspects of systems could be treated by net unfoldings [4, 6] or other Petri net techniques [15].

Acknowledgement

This research is supported by the 973 Program Grant 2010CB328102, NSFC Grants 60910004 and 60873018, and EPSRC VERDAD project.

References

1. E.Best, R.Devillers and M. Koutny: Petri Net Algebra. Monographs in Theoretical Computer Science, Springer (2001)
2. A.Cau and H.Zedan: Refining Interval Temporal Logic Specifications. Lecture Notes in Computer Science 1231 (1997) 79–94
3. E.A.Emerson: Temporal and Modal Logic. In: Handbook of Theoretical Computer Science, Elsevier Science (1990) 995–1072
4. J.Esparza: Model Checking Using Net Unfoldings. Science of Computer Programming 23 (1994) 151–195
5. J.-Y.Girard: Linear Logic. Theoretical Computer Science 50 (1987) 1–102
6. V.Khomenko and M.Koutny: Towards An Efficient Algorithm for Unfolding Petri Nets. Lecture Notes in Computer Science 2154 (2001) 366–380
7. S.Kripke: Semantical Analysis of Modal Logic I: Normal Propositional Calculi. Z. Math. Logik Grund. Math. 9 (1963) 67–96
8. Z.Manna and A.Pnueli: Verification of Concurrent Programs: The Temporal Framework. In: The Correctness Problem in Computer Science, Academic Press (1981) 215–273
9. R.Milner: A Calculus of Communicating Systems. Springer Verlag (1980)
10. B.Moszkowski: Compositional Reasoning About Projected and Infinite Time. Proceedings of ICECCS (1995) 238–245
11. B.Moszkowski: Executing Temporal Logic Programs. Cambridge University Press (1986)
12. B.Moszkowski and Z.Manna: Reasoning in Interval Temporal Logic. Lecture Notes in Computer Science 164 (1984) 371–382
13. T.Murata: Petri Nets: properties, Analysis and Applications. Proceedings of the IEEE 77 (1989) 541–80
14. M.Parigot and E.Pelz: A Logical Approach of Petri Net Languages. Theoretical Computer Science 39 (1985) 155–169
15. Petri net tools homepage: http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html
16. F.Siewe: A Compositional Framework for the Development of Secure Access Control Systems. PhD thesis, De Montfort University (2005)
17. M.Silva, E.Teruel and J.-M.Colom: Linear Algebraic and Linear Programming Techniques for the Analysis of Place/Transition Net Systems. Lecture Notes in Computer Science 1491 (1998) 309–373
18. A.Valmari: Stubborn Sets for Reduced State Space Generation. Lecture Notes in Computer Science 483 (1989) 491–515