From Petri Nets with Shared Variables to ITL

Hanna Klaudel
IBISC Laboratory
University of Evry
Evry, France
hanna.klaudel@ibisc.univ-evry.fr

Maciej Koutny and Ben Moszkowski
School of Computing Science
Newcastle University
NE1 7RU, United Kingdom
{maciej.koutny,ben.moszkowski}@ncl.ac.uk

***Abstract*—Petri nets and Interval Temporal Logic (ITL) are two formalisms for the specification and analysis of concurrent computing systems. Petri nets allow for a direct expression of causality aspects in system behaviour and in particular support system verification based on partial order reductions or invariant-based techniques. ITL, on the other hand, supports system verification by proving that the formula describing a system implies the formula describing a correctness requirement.**

It would therefore be desirable to establish a strong semantical link between these two models, thus allowing one to apply diverse analytical methods and techniques to a given system design. We have recently proposed such a semantical link between the propositional version of ITL (PITL) and Box Algebra (BA), which is a compositional model of basic (low-level) Petri nets supporting handshake action synchronisation between concurrent processes. In this paper, we extend this result by considering a compositional model of (high-level) Petri nets where concurrent processes communicate through shared variables. The main result is a method for translating a design expressed using a high-level Petri net into a semantically equivalent ITL formula.

***Keywords*-concurrency; ITL; Petri nets; relations between models; compositional translation; behavioural consistency.**

I. Introduction

Petri nets [23] and temporal logics [6], [18] are two formalisms for the specification and analysis of concurrent computing systems. Petri nets allow for a direct expression of causality aspects in system behaviour and in particular support system verification based on net unfoldings or invariant-based techniques. Temporal logics such as Interval Temporal Logic (ITL) [20], [22], on the other hand, support system verification by proving that the formula describing a system implies the formula describing a correctness requirement.

Establishing a strong semantical connection between these two models appears quite desirable as it would allow applying diverse analytical methods and techniques to a given system design. We have recently proposed in [5], [14] such a semantical link between the propositional version of ITL (PITL) and Box Algebra (BA) [1] which is a compositional model of basic (low-level) Petri nets supporting handshake action synchronisation between concurrent processes. Focusing on the Box Algebra facilitated the development of such a link as this model supports Petri nets built using composition operators inspired by common programming constructs, including sequence, iteration, parallel composition and choice, which are also very similar to the basic composition operators of ITL. Such a similarity facilitated the development of a very efficient (linear) representation of composite nets in terms of the derived logic formulas.

To formalise a semantical link between BA and ITL, for every logic formula we introduced a step sequence semantics which records variables changing their values at each computational step. This allowed us to compare its behaviour with the step sequences of the corresponding Petri net and conclude their full equivalence (similar to the key result of this paper, Theorem 1). In essence, the latter stated that any property which can be captured within the step sequence model can be analysed using either of the two equivalent representations, i.e., a Petri net or a logic formula.

In this paper, we extend the approach of [5], [14] by considering a compositional model of (high-level) Petri nets [2], [13] where concurrent processes communicate through shared variables of arbitrary types. (In addition, the variables can be used to carry out local computations.) The main result is a method for translating a design expressed using a high-level Petri net into a semantically equivalent ITL formula. The key idea behind the presented solution is to use in ITL formulas typed variables corresponding to the shared variables from the Box Algebra, and then to implement a mechanism of mutual exclusion for accessing these variables based on special control variables (such a mutually exclusive access is inherent in Petri nets but absent in ITL). This leads to a compositional, elegant and conservative solution in which concurrent processes evolve in parallel unless they compete for the same resource(s).

A. Some related work

Different kinds of logics have been previously used as formalisms for expressing correctness properties of systems specified using Petri nets. When it comes to the relationship between logics and Petri nets, we feel that the work on the connections between linear logic [9] and Place Transition nets has been the closest one. However, the main concern there is the handling of multiple token occurrences in net places. This, however, is not a feature of boxes considered in this paper; they can never contain multiple tokens in individual places. Another way to combine logics and Petri nets is reported in [24] and characterises Petri net languages in terms of second-order logical formulas. More recently, [25] has developed an algorithm for model checking monadic second-order formulas on 1-safe Petri nets.

B. Paper organisation

The paper is organised as follows: The following two sections introduce BASV, or the Box Algebra with shared variables, and the fragment of ITL used in this paper. Both are illustrated using the same small running example (the technical report [15] provides more extensive example and the proof of Theorem 4.1). Section IV defines formally the method in which we associate equivalent ITL formulae with BASV, and establishes the correctness of the proposed method. The paper ends with concluding remarks.

C. Notations

$\mathbb{N}$ denotes the set of 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; moreover, $\preceq$ is $\leq$ without the pair (ω, ω).

II. Box Algebra

In this paper we consider *Box Algebra with shared variables* (BASV), which is a simple sub-model of BA [1] supporting interprocess communication using shared variables.

The syntax of BASV-expressions E, sequential expressions S, and parallel expressions P is as follows:

$$S ::= \mathsf{stop} \ | \ d \ | \ S \,\Box\, S' \ | \ S\,;S' \ | \ [\![S \circledast S' \circledast S'']\!]$$
$$P ::= S_1 \,\|\, S_2 \,\|\, \dots \,\|\, S_k$$
$$E ::= P \ \mathsf{using}\ \Delta$$

where d is an action, and $\Delta = \delta_1, \dots, \delta_m$ $(m \geq 0)$ is a list of declarations of shared variables. Each declaration has the form $\tau_x{:}x = init_x$, where x is a variable, τ_x is a data type for x, and $init_x \in \tau_x$ is the default initial value of x.[1]

We will denote by X the set of all shared variables declared in Δ. Each action d is a well-formed Boolean expression (predicate) which can involve constants, operators and terms of the form $'x$ and x', where x is a variable in X. The set of all such variables will be denoted by var_d.

In the above syntax, stop stands for a blocked process, $S \,\Box\, S'$ for non-deterministic 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

$$E_{basv} \ = \ S_1 \,\|\, S_2 \,\|\, \dots \,\|\, S_k \ \mathsf{using}\ \Delta \qquad (1)$$

[1]We deliberately avoid here a full introduction of various (application-oriented) data types, and only assume that each data type τ is a set of values with associated operators which yield values of this type or some other known type.

for a parallel composition of k sequential processes which can communicate and synchronise using the shared variables X declared in Δ. *To simplify formal notations, we will assume that each action occurring in E_{basv} is unique. This is a harmless assumption since one can always add a conjunct $(k = k)$ to the k-th action appearing in E_{basv} (i.e., $(1 = 1)$ to the first action, $(2 = 2)$ to the second action, etc).*

A. Box nets with shared variables

The semantics of BASV-expressions is given through a mapping $box()$ into Petri nets called BASV-boxes. We will approach the definition of such a mapping in two stages corresponding intuitively to the two-stage definition of a BASV-expression E_{basv} in (1).

The first part of the definition of E_{basv} yields the sequential sub-expressions S_i which capture the control flow aspects in the execution involving the actions occurring in each individual sequential sub-system, and then puts them in parallel, producing the net representing a parallel expression

$$E_{flow} = S_1 \,\|\, S_2 \,\|\, \dots \,\|\, S_k \ . \qquad (2)$$

The second part, leading to E_{basv} declares the shared variables needed in particular for interprocess communication.

B. Deriving the net representing E_{flow}

A Petri net representing E_{flow}, called a box, is a very simple net (basically it is a collection of finite state machines).

Each box net is a tuple $\Sigma = (P, T, F, \ell)$ where P and T are disjoint finite sets of respectively places (representing local control states) and transitions; $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation; and $\ell : P \to \{\mathsf{e}, \mathsf{i}, \mathsf{x}\}$ is a labelling for places, i.e., e for 'entry', i for 'inner', and x for 'exit'.

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 p, we use ${}^\bullet p$ to denote its pre-set comprising all transitions t such that there is an arc from t to p, i.e., $(t, p) \in F$. The pre-set ${}^\bullet t$ of a transition t is defined analogously. The post-sets $p^\bullet$ and $t^\bullet$ are 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 which can be thought of as interfaces allowing one to compose boxes.

For each operator in E_{flow}, there is a matching operator on boxes (see [5] for formal definitions). Below, the Σ_i's are boxes with disjoint sets of nodes, each with a single entry place and a single exit place:

- $\Sigma_1 \,\Box\, \Sigma_2$ glues together the entry places of the two boxes creating a new entry place, as well as their exit places creating a new exit place.
- $\Sigma_1\,;\Sigma_2$ glues together the exit place of Σ_1 with the entry place of Σ_2. The entry place of the composite box is the entry place of Σ_1, and the exit place is the exit place of Σ_2.

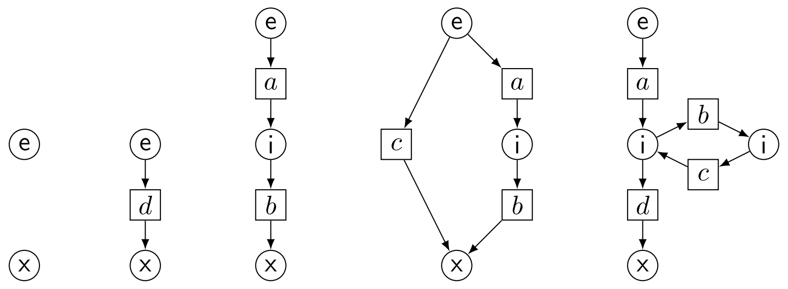

Figure 1. The diagrams of $box(\mathsf{stop})$, $box(d)$, $box(a\,;b)$, $box(c \,\square\, (a\,;b))$, and $box([\![a \circledast (b\,;c) \circledast d]\!])$.

- $[\![\Sigma_1 \circledast \Sigma_2 \circledast \Sigma_3]\!]$ glues the exit places of Σ_1 and Σ_2 with the entry places of Σ_2 and Σ_3. The entry place is that of Σ_1 and the exit place is that of Σ_3.
- $\Sigma_1 \,\|\, \dots \,\|\, \Sigma_k$ puts next to each other the boxes $\Sigma_1, \dots, \Sigma_k$. The exit and entry status of all places is retained.

We define a mapping $box(.)$ from the sequential expressions to boxes compositionally. For the blocked expression stop and an action d we have:

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

In other words, $box(\mathsf{stop})$ consists just of two places, and $box(d)$ of two places with transition d joining them (see Figure 1). Moreover, for any sequential expressions, we have (see Figure 1 for examples):

$$\begin{aligned} box(S \,\square\, S') &= box(S) \,\square\, box(S') \\ box(S\,;S') &= box(S)\,;box(S') \\ box([\![S \circledast S' \circledast S'']\!] &= [\![box(S) \circledast box(S') \circledast box(S'')]\!]. \end{aligned}$$

And the box corresponding to E_{flow} as in (2) is

$$box(E_{\mathit{flow}}) = box(S_1) \,\|\, \dots \,\|\, box(S_k)\ .$$

C. Deriving the net representing E_{basv}

To construct the BASV-box of E_{basv} as in (1), we add to $box(E_{\mathit{flow}})$ high-level places representing the data variables X declared in Δ. More precisely, we add a set of places X and assume that each place $x \in X$ is a high-level place of type τ_x. Intuitively, such a place will store the current value of variable $x \in X$. We do not add any new transitions, but rather connect the existing ones with the newly added places. The BASV-box corresponding to E_{basv}, denoted by $box(E_{basv})$, is obtained from $box(E_{\mathit{flow}})$ by adding the high-level places X and then, for every transition d (with guard d), adding the arcs (x,d) labelled $'x$ and (d,x) labelled x', for every variable $x \in var_d$.

D. Adding the initial marking

A marking (in other words, a state) of the BASV-box $box(E_{basv})$ consists of two parts reflecting the different nature of places derived from: (i) the control-flow part E_{flow} of E_{basv}; and (ii) the variable declarations Δ. Each such marking is a pair $\mathcal{M} = (M, \theta)$, where M is a set of places (carrying a token $\bullet$) of $box(E_{\mathit{flow}})$, and θ is a mapping assigning a value in τ_x, for every $x \in X$. The initial marking $\mathcal{M}_0 = (M_0, \theta_0)$ of $box(E_{basv})$ is defined as follows (see for example Figure 2(a)): (i) M_0 is the set ${}^\bullet box(E_{\mathit{flow}})$ of all entry places of $box(E_{\mathit{flow}})$; and (ii) $\theta_0(x) = init_x$, for every $x \in X$. In what follows, we will use $\mathcal{B} = (box(E_{basv}), \mathcal{M}_0)$ to denote BASV-box $box(E_{basv})$ together with the initial marking $\mathcal{M}_0$.

E. Executing BASV-box $\mathcal{B}$

The execution semantics of the BASV-box $\mathcal{B}$ is given through the set of its step sequences. We start by specifying what it means to execute instances of single transitions.

As $\mathcal{B}$ contains high-level places, we cannot just execute a transition since it is in general a shorthand for possibly infinitely many different basic activities. We therefore have to introduce substitutions which turn high-level transitions into concrete (executable) instances called *transition occurrences*. In what follows, a (satisfying) substitution for a transition d of $\mathcal{B}$ is any mapping ϱ with the domain $\{'x \mid x \in var_d\} \cup \{x' \mid x \in var_d\}$ such that d (considered here as a predicate) evaluates to *true* if each $'x$ is replaced by $\varrho('x)$, and each x' is replaced by $\varrho(x')$.

Let $\mathcal{M} = (M, \theta)$ be a marking of $\mathcal{B}$, and ϱ be a substitution for a transition d of $\mathcal{B}$. Then $d : \varrho$ is a transition occurrence enabled at $\mathcal{M}$ if ${}^\bullet d \subseteq M$ and $\theta(x) = \varrho('x)$, for all $x \in var_d$.

Suppose now that $\mathcal{M} = (M, \theta)$ is a marking of the Petri net $\mathcal{B}$ and $U = \{d_1 : \varrho_1, \dots, d_m : \varrho_m\}$ for $m \geq 0$, is a set of transition occurrences, each such transition occurrence being enabled at $\mathcal{M}$, such that ${}^\bullet d_i \cap {}^\bullet d_j = var_{d_i} \cap var_{d_j} = \varnothing$, for all $i \neq j$ (i.e., a single place can be accessed by at most one executed transition). Then U is a *step enabled* at the marking $\mathcal{M}$ which can be *executed*, leading to a marking $\widetilde{\mathcal{M}} = (\widetilde{M}, \widetilde{\theta})$ such that $\widetilde{M} = M \setminus {}^\bullet U \cup U^\bullet$ and, for all

$x \in X$ (we denote this by $\mathcal{M}[U\rangle\widetilde{\mathcal{M}}$ or $\mathcal{M}[U\rangle$):

$$\widetilde{\theta}(x) = \begin{cases} \varrho_i(x') & \text{if } x \in var_{d_i} \ (i \leq m) \\ \theta(x) & \text{otherwise .} \end{cases}$$

As far as the semantics $\mathcal{B}$ is concerned, only sequences of executed steps which start from its initial marking $\mathcal{M}_0$ need to be considered. We will assume that each such step sequence is infinite, which is a harmless requirement as any finite step sequence can be extended by an infinite sequence of empty steps (note that $\mathcal{M}[\varnothing\rangle\mathcal{M}$ for every marking $\mathcal{M}$). A step sequence of $\mathcal{B}$ is any infinite sequence $\gamma = \mathcal{M}_0 U_1 \mathcal{M}_1 U_2 \mathcal{M}_2 \ldots$ such that we have:

$$\mathcal{M}_0[U_1\rangle\mathcal{M}_1[U_2\rangle\mathcal{M}_2 \ldots$$

We denote this by $\gamma \in stepseq(\mathcal{B})$. Note that the step sequences of $stepseq(\mathcal{B})$ provide a complete account of the behaviour of the BASV-box $\mathcal{B}$.

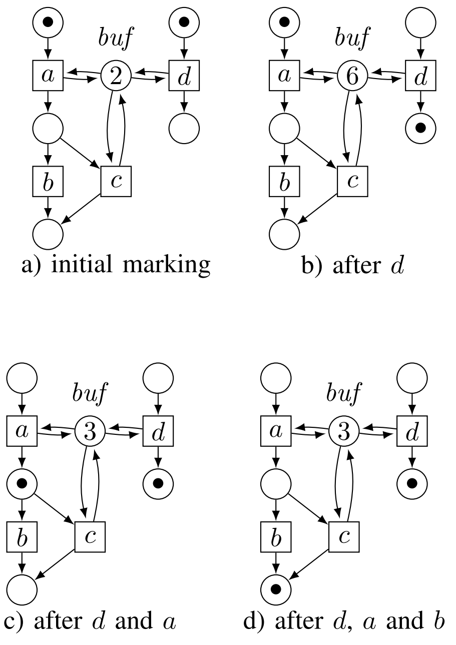

Figure 2. An execution of a Petri net corresponding to the BASV-expression $ProdCons$. Note that the actions used in $ProdCons$ play the role of Petri net transitions and the corresponding predicates play the role of the transition guards. For brevity we omit in the diagrams all the transition guards as well as the labels $'buf$ on arcs out-going from place buf, and the labels buf' on arcs incoming to buf.

F. Running example and its Petri net representation

Let us consider a very simple concurrent system consisting of two sequential processes, a producer and a consumer, synchronising through a shared (buffer) variable buf which initially holds 2 items. The two processes follow a simple scenario: (i) the producer inserts 4 items into the buffer and terminates; (ii) the consumer waits for the new items to be inserted as it needs to consume 3 items in order to proceed; and (iii) the consumer terminates its operation (note that if the buffer contained at least 6 items at this point, the consumer could also consume 6 items before terminating). This example can easily be written using the following BASV-expression, where $\mathsf{INT}{:}buf{=}2$ is a declaration of a shared variable buf of type INT initialised to 2:

$$ProdCons = a\,;(b \,\square\, c) \parallel d \text{ using } \mathsf{INT}{:}buf{=}2 \qquad (3)$$

In the above, the sequential sub-expressions, $Cons = a\,;(b \,\square\, c)$ and $Prod = d$, use the following atomic actions (basically, guarded commands) describing enabling conditions w.r.t. the current value of buf, and the effect (if any) of their executions on the variable buf declared in the BASV-expression $ProdCons$:

- $a = \langle 'buf \geq 3 \wedge buf' = 'buf - 3\rangle$ can only be executed if the current value of buf (represented by $'buf$) is at least 3, and if a is executed then the next value of buf (represented by buf') is the current value of buf minus 3;
- $b = \langle true\rangle$ can be enabled without accessing buf; it does not affect buf;
- $c = \langle 'buf \geq 6 \wedge buf' = 'buf - 6\rangle$ can only be executed if the current value of buf is at least 6, and if c is executed then buf is decremented by 6; and
- $d = \langle buf' = 'buf + 4\rangle$ can be enabled without accessing buf, and if d is executed then buf is incremented by 4.

A Petri net corresponding to the BASV-expression $ProdCons$ is obtained by first associating simple nets with the sequential sub-systems, $Prod$ and $Cons$, and adding a high-level data place buf representing the buffer. The buffer place is then connected to those transitions (actions of $ProdCons$) which access the buffer variable buf. The resulting BASV-box for $ProdCons$ is depicted in Figure 2(a) together with its initial marking (a token • in each entry place and the initial value of buf in the corresponding high-level place). (Note that we omitted the obvious net variables on arcs adjacent to place buf as well as all the transition guards.) In such a marking only transition d can be executed, leading to the net depicted in Figure 2(b). Then, after executing transitions a and b we obtain respectively the nets shown in Figure 2(c) and Figure 2(d).

III. Interval Temporal Logic

We now provide the syntax and semantics of a fragment of ITL, including only those constructs (basic and derived) which are used in the subsequent translation of the BASV-expression E_{basv}.

Let V be a countable set of Boolean control variables, and $\widehat{V}$ be a countable set of data variables (together with associated value domains and operations). The formulas of the fragment of the ITL logic we need are defined by:

$$\begin{array}{rcl} \phi & ::= & \mathsf{flip}(v) \mid \mathsf{skipst}(W) \mid \phi \wedge \phi' \mid \\ & & \phi \vee \phi' \mid \phi\,;\phi' \mid \phi^* \mid \mathsf{inf} \mid e \end{array}$$

where $v \in V$ and W is a finite subset of $V \cup \widehat{V}$. The intuition behind the above constructs is as follows: (i) $\mathsf{flip}(v)$ inverts the value of Boolean control variable v over a unit interval; (ii) $\mathsf{skipst}(W)$ keeps the values of the variables in W over a unit interval (in the examples, we will use $\mathsf{skipst}(w_1, \ldots, w_m)$ to denote $\mathsf{skipst}(\{w_1, \ldots, w_m\})$); (iii) "$;$" is a sequential composition operator (called chop); (iv) "*" is an iterative version of chop; (v) inf indicates an infinite interval; and (vi) e is a well-formed Boolean expression involving data variables x and terms $\bigcirc x$ as well as suitable operations on x and $\bigcirc x$ (note that $\bigcirc x$ refers to the value of x in the next state).

A state is a mapping which assigns values to the variables $V \cup \widehat{V}$, 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 definitions, we denote σ as $\langle \sigma_0, \sigma_1, \ldots, \sigma_{|\sigma|} \rangle$, where $\sigma_{|\sigma|}$ is undefined if σ is infinite. Then, for $0 \leq i \preceq j \leq |\sigma|$: $\sigma_{i..j} = \langle \sigma_i, \ldots, \sigma_j \rangle$, $\sigma^i = \langle \sigma_0, \ldots, \sigma_i \rangle$ and $\sigma^{(i)} = \langle \sigma_i, \ldots, \sigma_{|\sigma|} \rangle$.

The meaning of formulas is given by the satisfaction relation $\models$:

- $\sigma \models \mathsf{flip}(v)$ iff $|\sigma| = 1$ and $\sigma_1(v) = \neg\sigma_0(v)$.
- $\sigma \models \mathsf{skipst}(\{w_1, \ldots, w_m\})$ iff $|\sigma| = 1$ and $\sigma_1(w_i) = \sigma_0(w_i)$, for $i \leq m$.
- $\sigma \models \phi \vee \phi'$ iff $\sigma \models \phi$ or $\sigma \models \phi'$.
- $\sigma \models \phi \wedge \phi'$ iff $\sigma \models \phi$ and $\sigma \models \phi'$.
- $\sigma \models \phi\,;\phi'$ iff one of the following holds:
  - $|\sigma| = \omega$ and $\sigma \models \phi$.
  - there is $r \preceq |\sigma|$ with $\sigma^r \models \phi$ and $\sigma^{(r)} \models \phi'$.
- $\sigma \models \phi^*$ iff one of the following holds:
  - $|\sigma| = 0$.
  - there are $0 = r_0 \leq \ldots \leq r_{n-1} \preceq r_n = |\sigma|$ 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 $|\sigma| = \omega$.
- $\sigma \models e$ iff e evaluates to true if each x is replaced by $\sigma_0(x)$, and each $\bigcirc x$ is replaced by $\sigma_1(x)$.

IV. From BASV to ITL

We will now translate a BASV-expression E_{basv} defined as in (1) into a semantically equivalent ITL formula. Recall that we assumed that no action d occurs in E_{basv} more than once. Below, for each $x \in X$ declared in (1), we denote $\mathit{cvar}_x = \{d \mid x \in \mathit{var}_d\}$. That is, cvar_x are the control variables which correspond to the transitions accessing the data variable x.

A key idea inspired by [4] is to represent transition d by a separate Boolean variable d, and then to model an execution of d by the change of the value of d together with a suitable update of the data variables in var_d. In what follows, the set of actions occurring within S_i is denoted by V_i. Moreover, for each action d occurring in E_{basv}, $\mathit{itl}(d)$ is a formula obtained from d by replacing each $'x$ by x, and each x' by $\bigcirc x$. The translation is then given by:

$$\begin{array}{rcl} \mathit{itl}(E_{basv}) & = & \mathit{itl}_1(S_1) \wedge \ldots \wedge \mathit{itl}_k(S_k) \\ & & \wedge \\ & & \bigwedge_{x\in X}((x = \mathit{init}_x) \wedge \mathit{Ctrl}_x^*)\ , \end{array}$$

where we have (below $i \leq k$, and d is an action which occurs in S_i):

$$\begin{array}{ll} \mathit{itl}_i(\mathsf{stop}) & \\ \quad = & \mathsf{skipst}(V_i)^* \wedge \mathsf{inf} \\ \mathit{itl}_i(d) & \\ \quad = & \mathsf{skipst}(V_i)^*\,; \\ & \mathsf{skipst}(V_i \setminus \{\mathit{var}_d\}) \wedge \mathsf{flip}(d) \wedge \mathit{itl}(d)\,; \\ & \mathsf{skipst}(V_i)^* \\ \mathit{itl}_i(S \,\square\, S') & \\ \quad = & \mathit{itl}_i(S) \vee \mathit{itl}_i(S') \\ \mathit{itl}_i(S\,; S') & \\ \quad = & \mathit{itl}_i(S)\,; \mathit{itl}_i(S') \\ \mathit{itl}_i([S \circledast S' \circledast S'']) & \\ \quad = & \mathit{itl}_i(S)\,; \mathit{itl}_i(S')^*\,; \mathit{itl}_i(S'') \\ \mathit{Ctrl}_x & \\ \quad = & \mathsf{skipst}(\mathit{cvar}_x \cup \{x\}) \ \vee \\ & \bigvee_{d\in \mathit{cvar}_x}(\mathsf{flip}(d) \wedge \mathsf{skipst}(\mathit{cvar}_x \setminus \{d\})) \end{array}$$

Intuitively, the value of a variable d is kept unchanged unless we simulate an execution of action d by flipping the value of d. Note that we define $\mathit{itl}_i(S_i)$ rather than $\mathit{itl}(S_i)$ since the translation relies on 'knowing' the variables present in S_i in order to correctly map any stop occurring within S_i into a suitable formula. The semantics of the formula $\mathcal{F} = \mathit{itl}(E_{basv})$ is then captured by the set $\mathit{intervals}(\mathcal{F}) = \mathit{intervals}(\mathit{itl}(E_{basv}))$ of all the infinite intervals over which $\mathcal{F}$ is satisfied.

A. ITL *representation of the running example*

The relevant states of a BASV-box are those reachable from the initial one, and its possible executions can be represented by sequences of executed transitions (or sets of transitions, called steps). In the ITL context, however, a state is a mapping which assigns values to a set of variables, and possible executions are represented by sequences of such assignments, called intervals. Our solution relates these two rather different ways of representing the states and executions of concurrent systems. It is realised by associating with each transition t of the BASV-box representing $\mathit{ProdCons}$ a distinct Boolean (control) variable, also denoted by t, and then by representing each execution of transition t by flipping the value of variable t; otherwise the value of variable t is kept unchanged. The integer (data) variable buf is also used in the ITL formula corresponding to $\mathit{ProdCons}$ and is kept unchanged when not accessed by any currently

$$
\begin{aligned}
\mathsf{Prod} &= \mathsf{skipst}(d)^* \,; \\
&\quad \mathsf{flip}(d) \wedge (\bigcirc buf = buf + 4) \,; \\
&\quad \mathsf{skipst}(d)^*
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{Cons} &= \mathsf{skipst}(a,b,c)^* \,; \\
&\quad \mathsf{skipst}(b,c) \wedge \mathsf{flip}(a) \wedge \\
&\qquad (buf \geq 3 \wedge \bigcirc buf = buf - 3) \,; \\
&\quad \mathsf{skipst}(a,b,c)^* \\
&\quad ; \\
&\quad \left(\begin{array}{l}
\mathsf{skipst}(a,b,c)^* \,; \\
\mathsf{skipst}(a,c) \wedge \mathsf{flip}(b) \,; \\
\mathsf{skipst}(a,b,c)^* \\
\vee \\
\mathsf{skipst}(a,b,c)^* \,; \\
\mathsf{skipst}(a,b) \wedge \mathsf{flip}(c) \wedge \\
\quad (buf \geq 6 \wedge \bigcirc buf = buf - 6) \,; \\
\mathsf{skipst}(a,b,c)^*
\end{array} \right)
\end{aligned}
$$

$$
\begin{aligned}
Ctrl_{buf} &= \mathsf{skipst}(a,c,d,buf) \vee \\
&\quad (\mathsf{flip}(a) \wedge \mathsf{skipst}(c,d)) \vee \\
&\quad (\mathsf{flip}(c) \wedge \mathsf{skipst}(a,d)) \vee \\
&\quad (\mathsf{flip}(d) \wedge \mathsf{skipst}(a,c))
\end{aligned}
$$

Figure 3. Formulas used in the definition of ProdCons in (4).

executed transition t (or, equivalently, if variable t is not flipped).

The intended behaviour of the BASV-expression $ProdCons$, and consequently the actual behaviour of the corresponding Petri net, can be captured by the following ITL formula:

$$\mathsf{ProdCons} = \mathsf{Prod} \wedge \mathsf{Cons} \wedge (buf = 2) \wedge Ctrl^*_{buf} \quad (4)$$

where the various formulas used are given in Figure 3 where, e.g., $\mathsf{flip}(d)$ means that the Boolean variable d is flipped, and $\mathsf{skipst}(a,b,c)$ means that the variables a,b,c are kept unchanged over a unit interval.

A key idea behind the above formula is to link the control flow of $ProdCons$ with the data-related aspects involving the buffer variable. Intuitively, this is achieved by joining together the flipping of control variables with the corresponding manipulation of buf, e.g., as in

$$\mathsf{flip}(d) \wedge (\bigcirc buf = buf + 4) \,.$$

Moreover, when constructing the formula in (4) we have to ensure that the following hold throughout the entire execution: (i) at most one action/transition coming from a single sequential process is executed at any given point (this is achieved by using sub-formulas like $\mathsf{skipst}(a,b,c)$ and $\mathsf{skipst}(b,c)$ in the first two lines of Cons); and (ii) at most one action/transition accessing the buffer variable is executed at any given point; moreover, if none is executed then buf does not change its value (this is achieved by adding $Ctrl^*_{buf}$ which effectively implements a mutually exclusive access to buf).

B. Main result

Recall that step sequences of $stepseq(\mathcal{B})$ provide a complete account of the behaviour of the BASV-box $\mathcal{B}$. Similarly, $intervals(\mathcal{F})$ provide a complete account of the behaviour of the ITL formula $\mathcal{F}$. Our aim now is to show that these two semantical captures, i.e., $stepseq(\mathcal{B})$ and $intervals(\mathcal{F})$ are very closely related; in fact, they are in essence identical.

Before we can formulate the desired equivalence result, we need to discuss the very nature of such a relationship. The reason is that the two sets of behaviours in $stepseq(\mathcal{B})$ and $intervals(\mathcal{F})$ are strictly speaking different and so cannot be directly compared. To be able to make a meaningful comparison, we will now show how the intervals belonging to $intervals(\mathcal{F})$ can be interpreted as step sequences. Let σ be an interval over which the formula $\mathcal{F}$ is satisfied, i.e.,

$$\sigma = \langle \sigma_0, \sigma_1, \sigma_2, \ldots \rangle \in intervals(\mathcal{F}) \,. \quad (5)$$

We then denote by σ_{sseq} an infinite sequence

$$\sigma_{sseq} = \mathcal{N}_0 R_1 \mathcal{N}_1 R_2 \mathcal{N}_2 \ldots \quad (6)$$

defined as follows:

- Each R_i is a set of transitions of $\mathcal{B}$ (i.e., the control variables of $\mathcal{F}$) such that, for each transition d, $d \in R_i$ iff $\sigma_{i-1}(d) \neq \sigma_i(d)$. Moreover, we denote by $\#_i(d)$ the number of times d occurs in the sequence $R_1 \ldots R_i$.
- Each $\mathcal{N}_i$ is a pair (N_i, ψ_i) such that N_i is a set of places of $box(E_{flow})$ and ψ_i is a valuation for the variables in X; moreover, $\mathcal{N}_0 = \mathcal{M}_0$.
- For all places p of $box(E_{flow})$ and $i \geq 1$,
$$N_i(p) = M_0(p) - \sum_{d \in p^\bullet} \#_i(d) + \sum_{d \in {}^\bullet p} \#_i(d) \,,$$
where we identify the set N_i with its characteristic function.
- For all $i \geq 1$, $\psi_i = \sigma_i|_X$.

Then

$$stepseq(\mathcal{F}) = \{\sigma_{sseq} \mid \sigma \in intervals(\mathcal{F})\}$$

is the set of step sequences induced by the ITL formula $\mathcal{F}$. In other words, we interpret each flipping of the value of a control variable d as an execution of the transition d in $\mathcal{B}$.

Moreover, the valuations of data variables carry over without change from the states of σ to the corresponding markings of σ_{sseq}. The only more involved aspect of the definition of σ_{sseq} relates to a control place p. In such a case, we simply take into account the initial marking of a place p in $\mathcal{B}$ and then calculate the marking resulting from the execution of the steps $R_1 \ldots R_i$. We then obtain a key result which states that $\mathcal{F}$ and $\mathcal{B}$ basically capture the same sets of behaviours.

Theorem 4.1: $stepseq(\mathcal{F}) = stepseq(\mathcal{B})$.

Sketch of the proof: To show the inclusion $stepseq(\mathcal{F}) \subseteq stepseq(\mathcal{B})$, suppose that σ and $\sigma_{sseq} \in stepseq(\mathcal{F})$ are respectively as in (5) and (6). We will demonstrate that $(\mathcal{M}_0 =)\mathcal{N}_0[R_1\rangle\mathcal{N}_1$ in $\mathcal{B}$ which, together with

$$\sigma^{(i)}_{sseq} = \mathcal{N}_i R_{i+1} \mathcal{N}_{i+1} R_{i+2} \mathcal{N}_{i+3} \ldots$$

for all $i \geq 1$, suffices to prove by induction that $\sigma_{sseq} \in stepseq(\mathcal{B})$.

To show $\mathcal{N}_0[R_1\rangle\mathcal{N}_1$ we proceed as follows. First of all, the results of [5] and the fact that the construction of $\mathcal{F}$ is a conservative extension of that presented in [5], imply that if we disregard all the aspects related to the data variables (effectively, if we assume that $X = \varnothing$ and each action in the BASV-expression E_{basv} as in (1) is equivalent to $true$) then $\mathcal{N}_0[R_1\rangle\mathcal{N}_1$ does hold. This means that all the requirements related to the enabledness and execution of the step R_1 w.r.t. the non-data places are satisfied.

Take now a data place $x \in X$, and have two cases.
Case 1: $R_1 \cap cvar_x = \varnothing$. Then, from the definition of $Ctrl_x$ it follows that $\sigma_0(x) = \sigma_1(x)$ and so $\psi_0(x) = \psi_1(x)$, as required by the net semantics.
Case 2: $R_1 \cap cvar_x \neq \varnothing$. Then, again due to $Ctrl_x$, there is d such that $R_1 \cap cvar_x = \{d\}$. Then it must be the case that $itl(d)$ holds over the unit interval $\langle \sigma_0, \sigma_1 \rangle$. Therefore, $\psi_0(x) = \sigma_0(x)$ does not block the execution of transition d and if d (and R_1) is executed at $\mathcal{N}_0$, the value of x will be as specified by $\psi_1(x) = \sigma_1(x)$.
From the above it follows that R_1 is not blocked by the data places, and its execution updates the data variables as given by $\psi_1(x)$. Therefore $\mathcal{N}_0[R_1\rangle\mathcal{N}_1$.

To show $stepseq(\mathcal{F}) \supseteq stepseq(\mathcal{B})$, suppose that

$$\gamma = \mathcal{M}_0 U_1 \mathcal{M}_1 U_2 \mathcal{M}_2 \ldots \in stepseq(\mathcal{B})$$

is a step sequence such that $\mathcal{M}_i = (M_i, \theta_i)$, for all $i \geq 0$. We then define an interval

$$\gamma_{invl} = \langle \sigma_0 \sigma_1 \sigma_2 \ldots \rangle$$

such that $\sigma_i|_X = \theta_i$, for every $i \geq 0$, and, for all transitions d: $\sigma_i(d) = 0$ if d occurred an even number of times in $U_1 \ldots U_i$, and $\sigma_i(d) = 1$ otherwise. For example, if we take the step sequence γ illustrated in Figure 2, then we obtain γ_{invl}:

variables	$\gamma_{invl\,0}$	$\gamma_{invl\,1}$	$\gamma_{invl\,2}$	$\gamma_{invl\,3}$	$\ldots$
a	0	0	1	1	$\ldots$
b	0	0	0	1	$\ldots$
c	0	0	0	0	$\ldots$
d	0	1	1	1	$\ldots$
buf	2	6	3	3	$\ldots$

Note that in γ_{invl} all the control variables start with the value 0. However, it would be possible to start with any other combination of Boolean values obtaining an interval behaving in the desired way.

It is easy to see that $\gamma = (\gamma_{invl})_{sseq}$. Hence it suffices to demonstrate that $\gamma_{invl} \in intervals(\mathcal{F})$. To show this, we again take advantage of the results established in [5], in order to conclude that γ_{invl} would belong to $intervals(\mathcal{F})$ if we replaced all the $itl(d)$'s and $Ctrl_x$'s within $\mathcal{F}$ by $true$.

We then observe that $\gamma_{invl} \models Ctrl_x$, for each $x \in X$, which follows from the definition of step enabledness and execution in $\mathcal{B}$. More precisely, we have: (i) $var_d \cap var_{d'} = \varnothing$, for all distinct $d, d' \in U_i$; and (ii) executing U_i has no effect on x if $x \notin var_d$, for all $d \in U_i$.

Finally, a formula $itl(d)$ is satisfied over any finite interval where the variable d is flipped. This follows from the fact that $itl(d)$ always occurs in conjunction with $\mathsf{flip}(d)$, and the flipping of d corresponds to the execution of transition d, as we already noted. Hence $\gamma_{invl} \in stepseq(\mathcal{F})$ which completes the proof.

V. Conclusion

In this paper we considered a compositional model of (high-level) Petri nets where concurrent processes communicate through shared variables. The main result is a method for translating a design expressed using a high-level Petri net into semantically equivalent formula of ITL.

The results presented in this paper demonstrate that one can develop a very close structural connection between BASV 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 BASV. A long-term goal is the development of a hybrid verification methodology combining logic and Petri net techniques. For example, sequential algorithms and infinite data structures, as well as various forms of fairness, could be treated by ITL techniques [3], [4], [11], [21], while intensive parallel or communicating aspects of systems could be treated by net unfoldings [8], [12] or other Petri net techniques.

Acknowledgement

We would like to thank the referees for their comments and useful suggestions. This research was supported by the Royal Society IE141148 and ANR SYNBIOTIC projects.

References

[1] E. Best, R. Devillers and M. Koutny, "Petri Net Algebra", Monographs in Theoretical Computer Science, Springer, 2001.

[2] E. Best, W. Frączak, R.P. Hopkins, H. Klaudel, and E. Pelz, "M-nets: an Algebra of High Level Petri Nets, with an Application to the Semantics of Concurrent Programming Languages", Acta Informatica 35, Springer, 1998.

[3] A. Cau, H. Janicke and B. Moszkowski, "Verification and Enforcement of Access Control Policies", Formal Methods in System Design 43, Springer, 2013.

[4] A. Cau and H. Zedan, "Refining Interval Temporal Logic Specifications", LNCS 1231, Springer, 1997.

[5] Z. Duan, H. Klaudel and M. Koutny, "ITL Semantics of Composite Petri Nets", The Journal of Logic and Algebraic Programming, Elsevier, 2012.

[6] E.A. Emerson, "Temporal and Modal Logic", In: Handbook of Theoretical Computer Science, Elsevier Science, 1990.

[7] J. Esparza, "Model Checking Using Net Unfoldings", Sci. of Comp. Programming 23, Elsevier, 1994.

[8] J. Esparza, S. Römer and W. Vogler, "An Improvement of McMillan's Unfolding Algorithm", LNCS 1055, Springer, 1996.

[9] J.-Y. Girard, "Linear Logic", Theoretical Computer Science 50, Elsevier, 1987.

[10] C.A.R. Hoare, "An axiomatic basis for computer programming", CACM, 12, ACM, 1969.

[11] H. Janicke, A. Cau, F. Siewe and H. Zedan, "Dynamic Access Control Policies: Specification and Verification", The Computer Journal 56, Oxford University Press, 2012.

[12] V. Khomenko and M. Koutny, "Towards An Efficient Algorithm for Unfolding Petri Nets", LNCS 2154, Springer, 2001.

[13] H. Klaudel and F. Pommereau, "M-nets: a survey", Acta Informatica 236, Springer, 2008.

[14] H. Klaudel, M. Koutny and Z. Duan, "Interval Temporal Logic Semantics of Box Algebra", LNCS 8370, Springer, 2014.

[15] H. Klaudel, M. Koutny and B. Moszkowski, "From Petri Nets with Shared Variables to ITL", Report CS-TR-1476, School of Computing Science, Newcastle University, 2015.

[16] S. Kripke, "Semantical Analysis of Modal Logic I: Normal Propositional Calculi", Z. Math. Logik Grund. Math. 9, 1963.

[17] L. Lamport, "The temporal logic of actions", ACM ToPLAS, 16, 1994.

[18] Z. Manna and A. Pnueli, "Verification of Concurrent Programs: The Temporal Framework", In: The Correctness Problem in Computer Science, Academic Press, 1981.

[19] R. Milner, "A Calculus of Communicating Systems", Springer, 1980.

[20] B. Moszkowski, "Compositional Reasoning About Projected and Infinite Time", Proceedings of ICECCS, IEEE,1995.

[21] B. Moszkowski, "Executing Temporal Logic Programs", Camb. University Press (1986)

[22] B. Moszkowski and Z. Manna, "Reasoning in Interval Temporal Logic", LNCS 164, Springer, 1984.

[23] T. Murata, "Petri Nets: properties, Analysis and Applications", Proceedings of the IEEE 77, 1989.

[24] M. Parigot and E. Pelz, "A Logical Approach of Petri Net Languages", TCS 39, Elsevier, 1985.

[25] M. Praveen and K. Lodaya, "Parameterized Complexity Results for 1-safe Petri Nets", LNCS 6901, Springer, 2011.

[26] F. Siewe, "A Compositional Framework for the Development of Secure Access Control Systems", PhD thesis, De Montfort University (2005)

[27] M. Silva, E. Teruel and J.-M. Colom, "Linear Algebraic and Linear Programming Techniques for the Analysis of Place/Transition Net Systems", LNCS 1491, Springer, 1998.

[28] A. Valmari, "Stubborn Sets for Reduced State Space Generation", LNCS 483, Springer, 1989.