The Journal of Logic and Algebraic Programming 82 (2013) 95–110

Contents lists available at SciVerse ScienceDirect

The Journal of Logic and Algebraic Programming

journal homepage: www.elsevier.com/locate/jlap

ITL semantics of composite Petri nets

Zhenhua Duan [a], Hanna Klaudel [b], Maciej Koutny [c,*]

[a] School of Computer Science and Engineering, Xidian University, Xi'an, PR China
[b] IBISC, Université d'Évry-Val-d'Essonne, 91000 Évry, France
[c] School of Computing Science, Newcastle University, Newcastle upon Tyne, NE1 7RU, United Kingdom

ARTICLE INFO

Article history:
Received 14 September 2011
Received 20 September 2012
Accepted 4 December 2012
Available online 26 December 2012

Keywords:
ITL
Petri net
Box algebra
Composition
Semantics

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 action and local state based semantics which allows 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 investigate a basic semantical link between temporal logics and compositionally defined Petri nets. In particular, we aim at providing a support for the verification of behavioural properties of Petri nets using methods and techniques developed for ITL.

1. Introduction

Temporal logics [3,10] and Petri nets [15] are two different but, in many ways complementary formalisms for the specification and analysis of concurrent systems. A temporal logic, such as interval temporal logic (ITL) [12,14], 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, are a graphical model with semantics based on actions and local states which allows, e.g., for a direct expression of causality aspects in system behaviour. As a result, verification of system properties can be done using model checking techniques based on partial order reductions [19], or invariant techniques [18] based on the graph structure of nets.

In this paper, we aim at capturing a basic semantical link between temporal logics and Petri nets so that one would be able to use both kinds of verification techniques of system properties. Developing such a link is not straightforward as temporal logics and Petri nets have strikingly different nature. The first step, therefore, is the identification of a sufficiently expressive temporal logic and a 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 in the Petri net domain 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 box(E) is derived from a box expression E using a compositionally defined mapping 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

[*] Corresponding author.
E-mail addresses: zhenhua_d@yahoo.com (Z. Duan), klaudel@ibisc.univ-evry.fr (H. Klaudel), maciej.koutny@newcastle.ac.uk, maciej.koutny@ncl.ac.uk (M. Koutny).

1567-8326/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jlap.2012.12.001

remarkable to realise that interval temporal logic (ITL) is based on almost exactly the same set of fundamental 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 [7] for ITL, and unfolding based partial order model checking and invariant analysis for BA. Basically, our aim is to address the following problem:

Given a box expression E, define a translation itl(E) into ITL *so that* box(E) *and* $itl(E)$ *satisfy the same instances of behavioural properties coming from some rich and interesting set.*

In this paper, we provide a syntax-driven translation $itl(E)$ 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 a binary synchronisation similar to that in CCS [11], and then discuss how it could be extended to a general multi-way synchronisation scheme of [1]. The main result is the soundness of the proposed translation $itl(E)$ of E.

Temporal logic is commonly agreed to be the right formalism for capturing behavioural properties of concurrent systems. One would then normally expect that parallel composition should correspond to conjunction in the logic, i.e., if $S_1, \ldots, S_n$ respectively satisfy formulas $f_1, \ldots, f_n$, then $S_1 \parallel \ldots \parallel S_n$ should satisfy $f_1 \wedge \ldots \wedge f_n$. It is well known that Hoare logic [6] does not enjoy this property, i.e., $\{P_i\}\ S_i\ \{Q_i\}$, for all i, does not imply

$$\{P_1 \wedge \ldots \wedge P_n\}\, A_1 \parallel \ldots \parallel A_n\, \{Q_1 \wedge \ldots \wedge Q_n\}\ .$$

One of the strengths of TLA [9], where concurrency is modelled by sharing variables, is precisely this. The results presented this paper imply that if one uses ITL as specification language, then one can also match parallel composition and conjunction for the synchronisation mechanisms of a process algebra, even for the very general synchronisation operator of BA.

2. Preliminaries

Throughout the paper $\mathbb{N}$ 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, assuming that $\omega = \omega$ and $n < \omega$, for all $n \in \mathbb{N}_0$. Moreover, we define $\preceq$ as $\leq$ without the pair (ω, ω). 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.

3. Box algebra with one-to-one communication

We first consider SBA which is a simple sub-model of box algebra [1]. In particular, it only allows a 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 will use (silent) 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:

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

where a is a communication action, and A is a set of communication actions including all the conjugates (i.e., $\widehat{A} = \{\widehat{a} \mid a \in A\} \subseteq A$). 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

$$A_i \cap \widehat{A_i} = \varnothing \text{ and } A_i \cap A_j = \varnothing \text{ for } i \neq j\ ,$$

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 $(S_1 \parallel S_2 \parallel \ldots \parallel S_k)\ \mathsf{sco}\, A$