Science of Computer Programming 113 (2015) 119–148

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

An algorithm for compositional nonblocking verification using special events

Colin Pilbrow, Robi Malik [*]

Department of Computer Science, University of Waikato, Hamilton, New Zealand

A R T I C L E   I N F O

Article history:
Received 30 April 2014
Received in revised form 12 May 2015
Accepted 14 May 2015
Available online 4 June 2015

Keywords:
Discrete event systems
Finite-state machines
Model checking
Compositional verification
Nonblocking

A B S T R A C T

This paper proposes to improve compositional nonblocking verification of discrete event systems through the use of special events. *Compositional verification* involves abstraction to simplify parts of a system during verification. Normally, this abstraction is based on the set of events not used in the remainder of the system, i.e., in the part of the system not being simplified. Here, it is proposed to exploit more knowledge about the remainder of the system and check how events are being used. *Always enabled* events, *selfloop-only* events, *failing* events, and *blocked* events are easy to detect and often help with simplification even though they are used in the remainder of the system. Abstraction rules from previous work are generalised, and experimental results demonstrate the applicability of the resulting algorithm to verify several industrial-scale discrete event system models, while achieving better state-space reduction than before.

1. Introduction

The *nonblocking property* is a weak liveness property commonly used in *supervisory control theory* of discrete event systems to express the absence of livelocks and deadlocks [1,2]. This is a crucial property of safety-critical control systems, and with the increasing size and complexity of these systems, there is an increasing need to verify the nonblocking property automatically. The standard method to check whether a system is nonblocking involves the explicit composition of all the automata involved, and is limited by the well-known *state-space explosion* problem. *Symbolic model checking* has been used successfully to reduce the amount of memory required by representing the state space symbolically rather than enumerating it explicitly [3].

Compositional verification [4–6] is an effective alternative that can be used independently of or in combination with symbolic methods. Compositional verification exploits the fact that large systems are typically modelled by several components interacting in synchronous composition. Then *compositional minimisation* or *abstraction* [6] can be used to simplify individual components before computing their synchronous composition, gradually reducing the state space of the system and allowing much larger systems to be verified in the end. The ways how components can be simplified to ensure correct verification results depend on the property being verified [7].

The nonblocking property considered in this paper is logically different from most properties commonly studied for compositional verification, and requires very specific abstraction methods [8]. A suitable theory is laid out in previous work [9], where it is argued that abstractions used in nonblocking verification should preserve a process-algebraic equivalence called *conflict equivalence*. Various abstraction rules preserving conflict equivalence have been proposed and used for compositional

[*] Corresponding author.
E-mail addresses: cgp5@students.waikato.ac.nz (C. Pilbrow), robi@waikato.ac.nz (R. Malik).

http://dx.doi.org/10.1016/j.scico.2015.05.010
0167-6423/© 2015 Elsevier B.V. All rights reserved.

nonblocking verification. First, *observer projection* [10] and *weak observation equivalence* [11] have been used to simplify automata. The journal paper [8] introduces conflict equivalence to compositional nonblocking verification and proposes a set of conflict-preserving abstraction rules. The same technique has also been applied to compositional verification of the *generalised nonblocking* property [12], giving rise to an improved set of abstraction rules. It has also been proposed to replace abstraction rules by more general simplification processes using *annotated automata* [13] or *canonical automata* [14].

All the above methods are based on conflict equivalence, and make no assumptions about the automata *not* being simplified. If a part of a system is replaced by a conflict-equivalent abstraction, the nonblocking property is guaranteed to be preserved independently of the other system components [9]. While this is easy to understand and implement, more simplification is possible by considering the other system components. To improve the degree of simplification, it has been proposed to take more information about the remainder of the system into account and to consider that certain events are *always enabled* or *selfloop-only* in the remainder of the system [15].

This paper is an extended version of the workshop paper [15]. It contains more detailed results about always enabled and selfloop-only events including formal proofs of correctness, plus the additional special event types of *failing* and *blocked* events. It also includes a description of the algorithm for compositional nonblocking verification with special events, at a level of detail not published before, and more elaborate experimental results.

In the following, Section 2 introduces the background of nondeterministic automata, the nonblocking property, conflict equivalence, and compositional nonblocking verification. Section 3 introduces four types of special events and their key properties for use in compositional nonblocking verification, and Section 4 presents simplification rules that exploit these special events. Then Section 5 describes the compositional nonblocking verification algorithm, and Section 6 presents the experimental results. Finally, Section 7 adds some concluding remarks.

2. Preliminaries

2.1. Events and languages

Event sequences and languages are a simple means to describe discrete system behaviours [1,2]. Their basic building blocks are *events*, which are taken from a finite *alphabet* $\mathbf{A}$. In addition, two special events are used, the *silent event* τ and the *termination event* ω. These are never included in an alphabet $\mathbf{A}$ unless mentioned explicitly using notation such as $\mathbf{A}_\tau = \mathbf{A} \cup \{\tau\}$, $\mathbf{A}_\omega = \mathbf{A} \cup \{\omega\}$, and $\mathbf{A}_{\tau,\omega} = \mathbf{A} \cup \{\tau, \omega\}$.

$\mathbf{A}^*$ denotes the set of all finite *traces* of the form $\sigma_1\sigma_2 \cdots \sigma_n$ of events from $\mathbf{A}$, including the *empty trace* ε, while $\mathbf{A}^+ = \mathbf{A}^* \setminus \{\varepsilon\}$ does not include the empty trace. The *concatenation* of two traces $s, t \in \mathbf{A}^*$ is written as st. A subset $L \subseteq \mathbf{A}^*$ is called a *language*. The *natural projection* $P \colon \mathbf{A}^*_{\tau,\omega} \to \mathbf{A}^*_\omega$ is the operation that deletes all silent (τ) events from traces.

2.2. Nondeterministic automata

System behaviours are modelled using finite automata. Typically, system models are deterministic, but abstraction may result in nondeterminism.

Definition 1. A (nondeterministic) *finite automaton* is a tuple $G = \langle \mathbf{A}, Q, \to, Q^\circ \rangle$ where $\mathbf{A}$ is a finite set of *events*, Q is a finite set of *states*, $\to \subseteq Q \times \mathbf{A}_{\tau,\omega} \times Q$ is the *state transition relation*, and $Q^\circ \subseteq Q$ is the set of *initial states*.

The transition relation is written in infix notation $x \xrightarrow{\sigma} y$, and is extended to traces $s \in \mathbf{A}^*_{\tau,\omega}$ in the standard way. For state sets $X, Y \subseteq Q$, the notation $X \xrightarrow{s} Y$ means $x \xrightarrow{s} y$ for some $x \in X$ and $y \in Y$, and $X \xrightarrow{s} y$ means $x \xrightarrow{s} y$ for some $x \in X$. Also, $X \xrightarrow{s}$ for a state or state set X denotes the existence of a state $y \in Q$ such that $X \xrightarrow{s} y$, and $G \xrightarrow{s} x$ means $Q^\circ \xrightarrow{s} x$.

The termination event $\omega \notin \mathbf{A}$ denotes completion of a task and does not appear anywhere else but to mark such completions. It is required that states reached by ω do not have any outgoing transitions, i.e., if $x \xrightarrow{\omega} y$ then there does not exist $\sigma \in \mathbf{A}_{\tau,\omega}$ such that $y \xrightarrow{\sigma}$. This ensures that the termination event, if it occurs, is always the final event of any trace. The traditional set of *accepting* states is $Q^\omega = \{x \in Q \mid x \xrightarrow{\omega}\}$ in this notation. For graphical simplicity, states in Q^ω are shown shaded in the figures of this paper instead of explicitly showing ω-transitions.

To support silent events, another transition relation $\Rightarrow \subseteq Q \times \mathbf{A}^*_\omega \times Q$ is introduced, where $x \overset{s}{\Rightarrow} y$ denotes the existence of a trace $t \in \mathbf{A}^*_{\tau,\omega}$ such that $P(t) = s$ and $x \xrightarrow{t} y$. That is, $x \xrightarrow{s} y$ denotes a path with *exactly* the events in s, while $x \overset{s}{\Rightarrow} y$ denotes a path with an arbitrary number of τ events shuffled with the events of s. Notations such as $X \overset{s}{\Rightarrow} Y$ and $x \overset{s}{\Rightarrow}$ are defined analogously to $\to$.

Definition 2. Let $G = \langle \mathbf{A}_G, Q_G, \to_G, Q^\circ_G \rangle$ and $H = \langle \mathbf{A}_H, Q_H, \to_H, Q^\circ_H \rangle$ be two automata. The *synchronous composition* of G and H is

$$G \parallel H = \langle \mathbf{A}_G \cup \mathbf{A}_H, Q_G \times Q_H, \to, Q^\circ_G \times Q^\circ_H \rangle , \tag{1}$$

where