Information and Computation 242 (2015) 108–127

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Synthesizing structured reactive programs via deterministic tree automata

Benedikt Brütsch

RWTH Aachen University, Lehrstuhl für Informatik 7, Germany

ARTICLE INFO

Article history:
Received 21 October 2013
Available online 24 March 2015

ABSTRACT

Existing approaches to the synthesis of controllers in reactive systems typically involve the construction of transition systems such as Mealy automata. In 2011, Madhusudan proposed structured programs over a finite set of Boolean variables as a more succinct formalism to represent the controller. He provided an algorithm to construct such a program from a given ω-regular specification without synthesizing a transition system first. His procedure is based on two-way alternating ω-automata on finite trees that recognize the set of "correct" programs.

We present a more elementary and direct approach using only deterministic bottom-up tree automata and extend Madhusudan's results to the wider class of programs with bounded delay, which may read several input symbols before producing an output symbol (or vice versa). In addition, we show a lower bound for the size of these tree automata. Finally, we prove a lower bound for the number of Boolean variables that are required for a structured program to satisfy a given LTL specification, almost matching the known upper bound.

© 2015 Elsevier Inc. All rights reserved.

1. Introduction

Algorithmic synthesis is a rapidly developing field with many application areas such as reactive systems, planning and economics. Most approaches to the synthesis of controllers in reactive systems, for instance [1–4], involve synthesizing transition systems such as Mealy or Moore automata. Unfortunately, the resulting transition systems can be very large, which has motivated the development of techniques for the reduction of their state space (for example, [5]). Furthermore, the method of bounded synthesis [6,7] can be used to synthesize minimal transition systems by iteratively increasing the bound on the size of the resulting system until a solution is found. However, it is not always possible to obtain small transition systems. For example, for certain specifications in linear temporal logic (LTL), the size of the smallest transition system satisfying the specification is doubly exponential in the length of the formula [8].

The subject of the present paper is the synthesis of controllers represented in a more succinct way: We consider *structured reactive programs* over a finite set of Boolean variables, building on work of Madhusudan [9], refining his results and showing limitations of this approach. Structured reactive programs are non-terminating while-programs with input and output statements. They can be significantly smaller (regarding the length of the program code) than equivalent transition systems, and in contrast to transition systems, they are structured in the sense that they can be decomposed into subprograms.

E-mail address: bruetsch@automata.rwth-aachen.de.

http://dx.doi.org/10.1016/j.ic.2015.03.013
0890-5401/© 2015 Elsevier Inc. All rights reserved.

Madhusudan [9] proposes a procedure to synthesize such programs from ω-regular specifications without computing a transition system first, using the fact that structured programs can be represented by their syntax trees. Given a finite set of Boolean variables and a nondeterministic Büchi automaton (which we call the specification automaton) recognizing the complement of the specification, he constructs a two-way alternating ω-automaton on finite trees that recognizes the set of *all* programs over these variables that satisfy the specification. This automaton can be transformed into a nondeterministic tree automaton (NTA) to check for emptiness and extract a minimal program (regarding the height of the corresponding tree) from that set. In contrast to the transition systems constructed by classical synthesis algorithms, the synthesized program does not depend on the specific syntactic formulation of the specification, but only on its meaning.

In this paper, we present a direct construction of a deterministic bottom-up tree automaton (DTA) recognizing the set of correct programs, without a detour via more intricate types of automata. The DTA inductively computes a representation of the behavior of a given program in the form of so-called *co-execution signatures*. A co-execution is a pair consisting of a computation of the program and a corresponding run of the specification automaton, and a co-execution signature for a given program and a given specification automaton captures the essential information about their possible co-executions. A similar representation is used by Lustig and Vardi in their work on the synthesis of reactive systems from component libraries [10] to characterize the behavior of the components.

Our approach is not limited to programs that read input and write output in strict alternation, but extends Madhusudan's results to the more general class of programs with *bounded delay*: In general, a program may read multiple input symbols before writing the next output symbol, or vice versa, causing a delay between the input sequence and the output sequence. In a game-theoretic context, such a program corresponds to a strategy for a controller in a game against the environment where in each move the controller is allowed to either choose one or more output symbols or skip and wait for the next input (see [11]). This can be a suitable model for systems where the interaction between the controller and the environment is based on buffers for the input and output symbols. We consider programs that never cause a delay greater than a given bound $k \in \mathbb{N}$.

For a fixed k, the complexity of our construction matches that of Madhusudan's algorithm. In particular, the size of the resulting DTA is exponential in the size of the given specification automaton and doubly exponential in the number of program variables. In fact, we establish a lower bound, showing that the set of all programs over m Boolean variables that satisfy a given specification cannot even be recognized by an NTA with less than $2^{2^{m-1}}$ states, if any such program exists. However, note that a DTA (or NTA) accepting precisely these programs enables us to extract a minimal program for the given specification and the given set of program variables. Hence, while the tree automaton may be large, the synthesized program itself might be rather small.

To lay a foundation for our study of the synthesis of structured reactive programs, we define a formal semantics for such programs, which is only informally indicated by Madhusudan. To that end, we introduce the concept of *Input/Output/Internal machines (IOI machines)*, which are composable in the same way as structured programs. This allows for an inductive definition of the semantics.

The second half of this paper is dedicated to the question of how many (Boolean) variables are needed to satisfy a given specification. More specifically, we show that for certain specifications in LTL, at least $\Omega(2^{\sqrt{n}})$ Boolean variables are required, where n is the size of the respective LTL formula. This lower bound almost matches the exponential upper bound that can be derived from the doubly exponential upper bound for the size of transition systems for a given LTL specification [8]. In order to prove this lower bound, we draw on concepts from graph theory and exploit the fact that the so-called transition graphs of structured programs over a small number of variables have small tree-width. We show that for certain specifications, the tree-width of the transition graphs of the programs satisfying these specifications must be large, which allows us to deduce a lower bound for the number of variables used by these programs.

Related work Besides the concept of structured reactive programs, there are several other approaches to controller synthesis that may yield succinct implementations of the controller instead of classical transition systems. For instance, Aminof, Mogavero and Murano [12] provide a round-based algorithm to synthesize *hierarchical transition systems*, which can be exponentially more succinct than corresponding "flat" transition systems. The desired system is constructed in a bottom-up manner: In each round, a specification is provided and the algorithm constructs a corresponding hierarchical transition system from a given library of available components and the hierarchical transition systems created in previous rounds. Thus, in order to obtain a small system in the last round, the specifications in the previous rounds have to be chosen in an appropriate way.

Current techniques for the synthesis of (potentially) compact implementations in the form of circuits or programs typically proceed in an indirect way, by converting a transition system into such an implementation. For example, Bloem et al. [13] first construct a symbolic representation (a binary decision diagram) of an appropriate transition system and then extract a corresponding circuit. However, this indirect approach does not necessarily yield a succinct result.

Another succinct model for the representation of strategies in infinite games was introduced by Gelderie [14] in the form of *strategy machines*. They are equipped with a set of control states and a memory tape, providing a similar separation between control flow and memory as structured programs.

Download English Version:

https://daneshyari.com/en/article/426402

Download Persian Version:

https://daneshyari.com/article/426402

Daneshyari.com