ABSTRACT

The paper presents the LTL preserving stubborn set method for reducing the amount of work needed in the automatic verification of concurrent systems with respect to linear time temporal logic specifications. The method facilitates the generation of reduced state spaces such that the truth values of a collection of linear temporal logic formulas are the same in the ordinary and reduced state spaces. The only restrictions posed by the method are that the collection of formulas must be known before the reduced state space generation is commenced, the use of the temporal operator "next" is prohibited, and the (reduced) state space of the system must be finite. The method cuts down the number of states by utilising the fact that in concurrent systems the nett result of the occurrence of two events is often independent of the order of occurrence.

1. INTRODUCTION

The automatic verification of temporal properties of finite-state systems has been a topic of intensive research during the recent decade. A typical approach is to generate the state space of the system and then apply a model checking algorithm on it to decide whether the system satisfies given temporal logic formulas [Clarke & 86] [Lichtenstein & 85]. A well known problem of the approach is that the state spaces of systems tend to be very large, rendering the verification of medium-size and large non-trivial systems impossible with a realistic computer. This problem is known as the state explosion problem.

Concurrency is a major contributor to state explosion. It introduces a large number of execution sequences which lead from a common start state to a common end state by the same transitions, but the transitions occur in different order causing the sequences to go through different states. This phenomenon has been recognized long ago and the choice of a coarser level of atomicity has been suggested as a partial solution (see [Pnueli 86]). Unfortunately, the power of coarsening the level of atomicity is limited. Consider a system consisting of n processes which execute k steps without interacting with each other and then stop. The system has (k+1)^n states. Each of the processes of the system can be coarsened to a single atomic action. Coarsening reduces the number of states to 2^n which is still exponential...
in the number of the processes [Valmari 88c]. However, it seems intuitively that to check various properties of the system it would be sufficient to simulate the processes in one arbitrarily chosen order, thus generating only $nk+1$ states.

To our knowledge the first person to suggest a concurrency-based state space reduction method potentially capable of changing a state space from exponential to polynomial in the number of processes was W. Overman [Overman 81]. Overman’s work is little known, perhaps because he considered a very restricted case (the terminal states of systems consisting of processes which do not branch or loop), and the algorithm he gave as part of his method for finding certain sets was not efficient enough from the practical point of view. He suggested also a modified method with a faster algorithm, but the modification destroyed the ability of changing exponential state spaces to polynomial.

The problems in Overman’s approach were effectively solved by Valmari when he presented the so-called stubborn set method [Valmari 88a, 88b]. The stubborn set method has been developed in a series of papers [Valmari 88a, 88b, 88c, 89a, 89b]. Originally the method could be used only to investigate deadlocks but more advanced versions of the method have been gradually developed in order to verify more properties. The method was initially applied to ordinary Petri nets but now it is applicable to a rather general model of concurrency, the variable/transition systems. Two profoundly different versions of the method have been distinguished: weak and strong. The weak theory is more complicated and more difficult to implement, but it leads to better reduction results.

The present paper extends the stubborn set method to almost full linear temporal logic — almost, because the operator “next state” is forbidden. For simplicity, we have chosen to use the strong stubborn set framework, although with minor refinements the results are valid in the weak theory as well. The theory is developed in Chapter 2. Chapter 3 (not in this abridged version) discusses how the theory may be implemented and Chapter 4 contains an example.

This paper is a revised and abridged version of a paper with the same name which appeared in DIMACS Technical Report 90-31, “Workshop on Computer-Aided Verification”, Rutgers University, NJ, USA, June 1990, Volume I.

2. DEFINITIONS AND BASIC THEOREMS

2.1 Variable/Transition Systems

To develop the stubborn set method we look at concurrent systems as systems consisting of a finite set $V$ of variables and a finite set $T$ of transitions. Each variable $v$ has an associated set called type and denoted by type$(v)$, and at every instant of time $v$ has a unique value belonging to its type. Assuming an ordering of $V$, the Cartesian product of the types of the variables is the set of syntactic states and is denoted by $S$. The value of variable $v$ at syntactic state $s$ is denoted by $s(v)$. There is a partial next state function next from $S \times T$ to $S$ which defines when a transition is enabled and what is the result of the occurrence of an enabled transition. Transition $t$ is enabled in state $s$, denoted by $en(s,t)$, iff $next(s,t)$ is defined. If $next(s,t) = s'$ we say that $t$ may occur at $s$ producing $s'$ and often write $s - t \rightarrow s'$. We often merge the states between successive transition occurrences and write $s_0 - t_1 \rightarrow s_1 - t_2 \rightarrow \ldots - t_n \rightarrow s_n$ instead of $s_0 - t_1 \rightarrow s_1 \land s_1 - t_2 \rightarrow s_2 \land \ldots \land s_{n-1} - t_n \rightarrow s_n$. A sequence like this is called a finite execution sequence and its length is $n$. The concatenation of the execution sequences $\sigma = s_0 - t_1 \rightarrow \ldots - t_n \rightarrow s_n$ and $\rho = r_0 - d_1 \rightarrow \ldots - d_m \rightarrow r_m$ where $s_n = r_0$ is defined by $\sigma \rho = s_0 - t_1 \rightarrow \ldots - t_n \rightarrow s_n - d_1 \rightarrow \ldots - d_m \rightarrow r_m$. "$\rightarrow^*$" is defined by $s \rightarrow^* s' \iff \exists t \in T: s - t \rightarrow s'$. "$\rightarrow^*$" is the reflexive transitive closure of "$\rightarrow". There is a distinguished state $s_0$ called the initial state of the system. A variable/transition system or v/t-system is the 5-tuple $(V,T,type,next,s_0)$, where the components of the 5-tuple are as just explained.