The Stubborn Set Method in Practice

Kimmo Varpaaaniemi and Marko Rauhamaa

Helsinki University of Technology, Digital Systems Laboratory,
Otakaari 1, SF-02150, Finland
Kimmo.Varpaaniemi@hut.fi, Marko.Rauhamaa@hut.fi

Abstract. A project for studying the stubborn set method in practice is described. The stubborn set method has been implemented in PROD, a reachability analysis tool for PrT-nets, and will be studied in practical cases using PROD.

1 Introduction

In theory, reachability analysis is one of the best methods to analyze concurrent and distributed systems. In practice, the state space explosion problem strongly restricts the usefulness of reachability analysis.

Often it is the case that only the terminal states of a system are of interest. Intuitively it seems that the terminal states of a system should be found without having to generate the whole state space. Some methods [1] have been developed in which a reduced state space is generated so that no terminal state is lost. The stubborn set method [1, 3, 4, 5, 6] is a very promising such method. The aim of the project is to study the practical value of the stubborn set method. Experimental results are useful for both practice and theory.

In Section 2 we describe the stubborn set method and mention some results concerning its efficiency. In Section 3 we describe the analysis tool that we are going to use. The actual project is described in Section 4.

2 The stubborn set method

The stubborn set method [1, 3, 4, 5, 6] takes advantage of the fact that there is often a lack of interaction between transitions in a concurrent or distributed system. Stubborn sets are sets of transitions that are not affected by the transitions not in the stubborn set. The method reduces the state space of a system but preserves all terminal states. There are variations of the method which preserve not only the terminal states but also some other important properties of a system, such as livelocks and testing equivalence. There are several algorithms for implementing the basic method and its variants.

For the classical system of $n$ dining philosophers, the size of the whole reachability graph is exponential in $n$. The stubborn set method generates a graph the size of which is quadratic in $n$ [3]. Similar reduction occurs in the number of computation steps [1]. Some other systems have been studied experimentally [6]. In some cases the stubborn set method remarkably reduces the state space but in other cases there is no significant reduction.
No satisfactory theory has been presented about the efficiency of the stubborn set method. The experiments made so far are not by any means sufficient to show how efficient the method is in practice. For both practical and theoretical reasons, the method should be studied in a variety of practical cases.

3 PROD

PROD is a reachability analysis tool for PrT-nets. PROD is a product of Digital Systems Laboratory of Helsinki University of Technology. PROD is primarily a research tool but suits well for industrial and educational purposes, too.

The net description language of PROD is the C preprocessor language extended with some net description directives. Example: A PrT-net model of the classical problem of dining philosophers is presented in Figure 1. In Figure 2 the net is presented in PROD’s description language.

![Diagram](image)

**Fig. 1.** A PrT-net modelling the problem of four dining philosophers.
#define PHILCNT 4
#define LEFT(x) (x)
#define RIGHT(x) (1 + ((x) % PHILCNT))

#place thinking lo(<.1.>) hi(<.PHILCNT.>) mk(<.1..PHILCNT.>)
#place forks mk(<.1..PHILCNT.>)
#place withLeft1 lo(<.1.>) hi(<.PHILCNT.>)
#place eating lo(<.1.>) hi(<.PHILCNT.>)
#place withLeft2 lo(<.1.>) hi(<.PHILCNT.>)

#trans takeLeft
  in { thinking: <.ph.>; forks: <.LEFT(ph.).>; }
  out { withLeft1: <.ph.>; }
@endr

#trans takeRight
  in { forks: <.RIGHT(ph.).>; withLeft1: <.ph.>; }
  out { eating: <.ph.>; }
@endr

#trans putRight
  in { eating: <.ph.>; }
  out { withLeft2: <.ph.>; forks: <.RIGHT(ph.).>; }
@endr

#trans putLeft
  in { withLeft2: <.ph.>; }
  out { thinking: <.ph.>; forks: <.LEFT(ph.).>; }
@endr

Fig. 2. The net of Figure 1 presented in PROD's description language.

Care has been taken that PROD would generate reachability graphs as fast as possible. Contrary to other approaches where the reachability graph is generated by interpreting the net description, PROD compiles the net description into an executable graph generator program.

Graph generation can take place in either high or low level. In the high level generation, the instances of a transition are computed by substituting the input arc expressions of the transition according to the current marking, and substituting the output arc expressions according to the current input variable substitution. (User defines how the output variable substitutions depend on the input variable substitution. For each input variable substitution there can be one or more output variable substitutions.) The high level generation has the advantage that transition variables can have virtually infinite ranges. The low level generation means generating the reachability graph of a P/T-version of the original net. (When the user asks questions about the graph, the information is presented in high level form.)

In addition to the ordinary non-reductive reachability graph generation method, two reductive methods have been implemented in PROD so far. The stubborn set method and the priority method [1] are available. The stubborn set method has been implemented using a modified version of the algorithm that uses strongly connected components [1]. Instead of choosing the first stubborn set, our algorithm chooses one having the least number of enabled transitions. (The collection of sets from which the winner is chosen may, of course, be a minor subset of the collection of
all stubborn sets.) Our algorithm preserves the terminal states of a system but does not, in general, preserve other system properties. The stubborn set method is always applied in low level. (We have not seen any variant of the method that would be practical when transition variables are unbounded.)

PROD has a query program for asking questions about reachability graphs. A CTL-like [2] query language has been implemented, which can express various important system properties. All information about terminal nodes, false facts and strongly connected components of a reachability graph is available without delay because corresponding computations are made either during or immediately after graph generation. The query language is extendible by user through aliasing. Example: Some questions about the reduced reachability graph of the net of Figure 1 are asked in Figure 3. The graph has been generated by using the stubborn set method.

Fig. 3. Queries about the reduced reachability graph of the net of Figure 1.

Queries can be made before graph generation has been completed. Graph generation can be interrupted and continued arbitrary many times. For example, user can ask information about the terminal nodes found so far, even if the total reachability graph is infinite.

PROD is very independent of machine and operating system. Up till now it has
been installed on a number of UNIX machines and an MS-DOS PC.

4 Studying the stubborn set method in practice

We will study the stubborn set method in a number of practical cases using PROD. A ‘real world’ protocol is a typical practical case. TCAP is one of the protocols to be modelled and analyzed.

It might be useful to study different implementations of the stubborn set method. For example, the deletion algorithm [1] could be implemented and studied. Extra computation per each generation step might sometimes remarkably reduce graph size and thus the total number of computation steps.

We hope that the project would make us conclude some theoretical results concerning the efficiency of the stubborn set method. At least, the project should give a lot of information about the practical value of the method.

Acknowledgements

In addition to the authors, following persons have been developing or documenting PROD: Jaakko Halme, Johannes Helander, Kari Hiekkakoski, Jari Janhunen, Ismo Kangas, Kenneth Oksanen, Olavi Pesonen, James Reilly, Heikki Suonsivu, and Kimmo Valkealahti.

References