ProbDiVinE: A Parallel Qualitative
LTL Model Checker

Jiří Barnat  Luboš Brim  Ivana Černá  Milan Češka  Jana Tůmová
Faculty of Informatics, Masaryk University, Botanická 68a, 60200 Brno, Czech Republic

Abstract—We introduce a parallel model checker for checking Markov decision processes against linear time properties. The model checker extends the parallel model checker DiVinE and supports verification of qualitative properties.

I. INTRODUCTION

Probabilistic systems like Markov chains and Markov decision processes provide a reasonable semantics for systems that exhibit uncertainty. A number of qualitative and quantitative model checking algorithms for finite state probabilistic systems have been proposed [12], [18], [8], [1], [9], [14], [10]. Model checking of probabilistic systems branches into qualitative and quantitative approach. In the qualitative setting it is checked whether a property holds with probability 0 or 1; in a quantitative setting it is verified whether the probability for a certain property meets a given lower or upper bound.

For probabilistic systems the state explosion problem is more critical than in the non-probabilistic case. Several methods that have been developed for non-probabilistic systems to avoid the state explosion, were adapted to probabilistic systems. For branching time logics these are the symbolic approach [3] implemented in the model checker PRISM [15], [13] and the MDP model checker RAPTURE [6] which uses an iterative abstraction refinement. For linear time logic the most prominent partial order approach has been recently adapted as well [4], [2] and implemented in the verification tool LiQuor [7].

Over the past decade, many techniques using distributed and/or parallel processing have been proposed to combat the computational complexity of non-probabilistic verification, model checking in particular. However, not much has been done in applying these techniques to the verification and analysis of probabilistic systems. A notable exception is the work on parallelizing the symbolic model checker PRISM [19], [20], [16].

In this short tool paper we introduce a parallel model checker ProbDiVinE for qualitative model checking of finite state Markov decision processes (MDPs) against LTL properties. We use the automata-theoretic approach [18], [9], [11] where qualitative LTL model checking of MDPs is reduced to the question whether the product automaton for a given MDP with a Büchi acceptance condition contains an accepting end component (AEC).

II. HOW THE TOOL WORKS

The tool implements a parallel adaptations of the algorithm of de Alfaro (dA) [5]. It computes the set of states that contains all accepting end components. In particular, the algorithm maintains approximation set of states that may belong to an AEC. The algorithm repeatedly refines the approximation set by locating and removing states that cannot belong to an AEC, we call this a pruning step. The core of the algorithm are conditions determining the states to prune.

The algorithm by de Alfaro was the only one from the existing sequential approaches that allowed for a reasonable parallelization. The other algorithms, like the one of Courcoubetis and Yannakakis (CY) and CY with recursive elimination (CY+RE) [9] are based on repeated decomposition of the underlying graph of product MDP into strongly connected components and elimination of states violating their ergodic consistency or other techniques that are inherently sequential. For convenience ProbDiVinE also provides these serial algorithms and can use them on a single machine.

byte fork[3]={0,0,0};
byte hungry[3]={1,1,1};
process Philosopher_0 {
  state thinks, eats,
  want_L,want_R, has_L, has_R;
  init thinks;
  trans
    thinks => { want_L:1, want_R:1 },
    want_L -> has_L { guard fork[0] == 0; effect fork[0] = 1; },
    has_L -> eats { guard fork[1] == 0; effect fork[1] = 1; },
    has_L -> thinks{ guard fork[1] == 1; effect fork[1] = 0; },
    want_R -> has_R { guard fork[1] == 0; effect fork[1] = 1; },
    has_R -> eats { guard fork[0] == 0; effect fork[0] = 1; },
    has_R -> thinks{ guard fork[0] == 1; effect fork[0] = 0; },
    eats -> thinks{ effect hungry[0] == 0; },
}
process Philosopher_1 {
  ;
}
process Philosopher_2 {
  ;
}
}

system async;

Fig. 1. Example of PROBDVE source-code.

The input language PROBDVE of ProbDiVinE is
a modification of the DiVinE’s native language DVE. PROBDVE models systems as a composition of processes, which can change their states via probabilistic (\(\rightarrow\)) or non-probabilistic (\(\rightarrow\)) transitions and can synchronize using channels. An example of a PROBDVE source-code for randomized solution to the dining philosophers problem [17] is given in Figure 1.

A non-probabilistic transition may have a guard (a condition which has to be satisfied for the transition to be enabled), an effect (assignment to a variable), and a sync expression (for synchronization with another transition via channel). A probabilistic transition just determines probability of resulting state, which is given by weights assigned to states. For example \(s \rightarrow \{ s : 3, t : 2, u : 2 \}\) means that from the state \(s\) system results in the state \(s\) with probability \(\frac{1}{2}\), in the state \(t\) with the probability \(\frac{2}{3}\), as well as in the state \(u\).

PROBDVE is build on the top of the DiVinE library that offers common functions needed to develop a parallel or distributed enumerative model checker. The only extension to the library that was necessary was the extension of the state generator to a probabilistic state generator, hence, it can handle probabilistic transitions of PROBDVE as introduced above. For the structure of PROBDVE implementation and connection to DiVinE see Figure 2.

![Diagram of PROBDVE](image)

**Fig. 2.** ProbDiVinE structure.

**PROBDVE** is currently operated in command-line mode. The input is the model given as a *.probde* file and the LTL formula given as an *.ltl* file. Further parameters include the algorithm to be used, number of workstations involved in the computation etc. Details are given in the documentation. The tool can be downloaded together with the DiVinE tool from the page [http://anna.fi.muni.cz/divine/probdive](http://anna.fi.muni.cz/divine/probdive).

Our initial experiments confirmed good scalability on many examples. In several cases the sequential algorithms were much faster than the parallel algorithm, however, once the internal memory has been exhausted only the parallel algorithm was able to finish the computation.

### III. Future plans

Our intention is to extend PROBDVE to quantitative LTL model checking by using a distributed linear solver, to add the possibility of checking reward properties, as well as to build a suitable graphical user interface.

### References


