ProbVerus: Probabilistic Symbolic Model Checking

Vicky Hartonas-Garmhausen[1], Sergio Campos[2], Ed Clarke[3]

[1]Carnegie Mellon University, Department of Engineering and Public Policy,
5000 Forbes Avenue, Pittsburgh, PA 15213, USA
hartonas@cs.cmu.edu
[2]Federal University of Minas, Gerais, Brasil
scampos@dcc.ufmg.br
[3]Carnegie Mellon University, Department of Computer Science,
5000 Forbes Avenue, Pittsburgh, PA 15213, USA
emc@cs.cmu.edu

Abstract. Model checking can tell us whether a system is correct; probabilistic model checking can also tell us whether a system is timely and reliable. Moreover, probabilistic model checking allows one to verify properties that may not be true with probability one, but may still hold with an acceptable probability. The challenge in developing a probabilistic model checker able to handle realistic systems is the construction of the state space and the necessity to solve huge systems of linear equations. To address this problem, we have developed ProbVerus, a tool for the formal verification of probabilistic real-time systems. ProbVerus is an implementation of probabilistic computation tree logic (PCTL) model checking using symbolic techniques. We present ProbVerus, demonstrate its use with a simple manufacturing example, and report the current status of the tool. With ProbVerus, we have been able to analyze, within minutes, the safety logic of a railway interlocking controller with 10^{27} states.

1 Introduction

The large size and high complexity of real-world mission-critical systems makes the verification of these systems an extremely difficult problem. To study the complete system, one must also include the system's interface with the environment. Physical systems are stochastic in nature and randomization makes the verification of probabilistic systems even more difficult due to its nonintuitive effects. At the same time, industries such as the transportation, pharmaceutical, chemical, and nuclear, are required to meet this challenge being constantly under the scrutiny of process and product specification. There is a great need for industrial-strength formal methods as well as real-world case studies that can demonstrate their feasibility.

Probabilistic model checking is a method for the formal verification of stochastic systems. The state of the art in probabilistic verification research includes numerous theoretical studies that have lead to efficient algorithms; for example, there is an LTL model checking algorithm which is exponential in the size of the formula and polynomial in the size of the Markov chain [9]. The bottleneck in developing a probabilistic model checker able to handle realistic systems is the construction of the state space and the necessity to solve huge systems of linear equations. This paper proposes a novel approach; it presents an implementation of probabilistic model checking using multi terminal binary decision diagrams (MTBDDs) to perform the probability calculations. MTBDDs, introduced in [8], differ from binary decision diagrams (BDDs) in that the leaves may have values other than 0 and 1; in this case the leaves contain transition probabilities. Hachtel et al. have used algebraic decision diagrams ADDs (same as MTBDDs) in the Markovian steady state analysis of large finite state machines (with up to 10^{27} states) [10]. MTBDDs can be integrated with a symbolic model checker and have the potential to outperform other matrix representations because they are very compact, by eliminating redundancy and allowing computations on sets of states rather than on individual states. While it is difficult to provide precise time complexity estimates for probabilistic model checking using MTBDDs, the success of BDDs in practice made the MTBDD representation worthwhile to explore.

We have developed ProbVerus, a probabilistic symbolic model checker, which combines Probabilistic Computation Tree Logic (PCTL) model checking [11] and symbolic techniques. PCTL, which allows the expression of time and probability, has been selected for its expressive power and the simplicity of the verification algorithms it involves. Sections 2 and 3 introduce the building blocks of ProbVerus, PCTL and MTBDDs. Section 4 describes ProbVerus with a short run down of the syntax and the semantics of ProbVerus programs.

In section 5 we demonstrate the use of ProbVerus in the verification of engineering systems by modeling and analyzing the reliability of a simple manufacturing system. By extending model checking to the analysis of probabilistic systems, we have been able to model the stochastic behavior of manufacturing systems: arrival time of successive raw workpieces, processing time on a machine, machine setup time, material handling time, message transmission time, lifetime of a tool, time to failure of a machine or a robot, repair time, and so on. Probabilistic model checking allows us to verify properties of these systems, such as “the probability of the system reaching a deadlock within the first 12 hours of operation is 2%”. Such information is extremely useful in the design of such capital-intensive systems since deadlocks and unscheduled downtime of equipment due to failures are major factors on system performance. Section 6 reports the current status of ProbVerus and concludes with a discussion of the feasibility of the approach in realistic applications.

2 Probabilistic Computation Tree Logic

We use Probabilistic real time Computation Tree Logic (PCTL) introduced in [11]. PCTL augments Clarke, Emerson, and Sistla's CTL [7] with time and probability. The temporal operators (**F**, **G**, **U**) are extended with time bounds and probabilities ($\mathbf{F}^{\leq t}_{\geq p}$, $\mathbf{G}^{\leq t}_{\geq p}$, $\mathbf{U}^{\leq t}_{\geq p}$). The expressive power of the resulting logic is illustrated by the following examples:

- $req\,\mathbf{U}^{\leq t}_{\geq p}\,ack$: there is at least a probability p that there is an acknowledgment to the request within t units and *req* will be true until *ack* becomes true.

- $\mathbf{G}_{\geq p}^{\leq t}$ *Sysfail*: there is no system failure *Sysfail* for *t* time units with a probability of at least *p*.
- $\mathbf{F}_{\geq p}^{\leq t}$ *alarm*: there is a probability of at least *p* that an alarm will be generated within *t* time units.

PCTL formulas are interpreted over finite state discrete-time Markov chains. This model has been used in the analysis of complex probabilistic systems, such as dependability analysis of fault-tolerant real-time control systems and performance analysis of commercial computer systems and networks. The Markov model has been the standard model used for probabilistic model checking (Hart and Sharir [13], Lehman and Shelah [16], Vardi [17], Hansson and Jonsson [11], Alur, Courcoubetis, and Dill [1], Courcoubetis and Yannakakis [9], Aziz el al [2]).

Markov models are constructed from states and state transitions; a state describes the system at one time instant; a state transition describes the change in state at one tick. In discrete-time models, all state transitions occur at fixed intervals and are assigned probabilities. The basic underlying assumption is that the probability of a next state transition depends only on the current state. For reliability analyses, the Markov model fits with the standard assumption of constant failure rates, exponentially distributed interarrival times for failures, and Poisson arrivals of failures. Reliability models usually assume that repair of a failed system restores it so that the failure rate of the repaired system is the same as if no failure had occurred. This assumption is valid during the useful life cycle of a component, but not accurate for components that improve with time (burn-in period) or components subject to wear and aging (wear-out period). This assumption is made nevertheless to allow for analytic solutions.

Model of Computation. PCTL formulas are interpreted over labelled discrete time Markov chains. Formally, a *labelled Markov chain* is a tuple (S, s^i, T, L), where

- S is a finite set of *states*,
- s^i is an *initial state,*
- T is the *transition relation* $T = S \times S \rightarrow [0, 1]$ such that for all s in S we have
$$\sum_{s' \in S} T(s, s') = 1,$$
- L is a *labeling function* assigning atomic propositions to states, $L\colon S \rightarrow 2^{AP}$

A path π from a state s_0 is an infinite sequence $\pi = s_0\ s_1\ s_2 \dots s_n \dots$ of states with s_0 as the first state and nonzero transition probabilities $P(s_{i-1}, s_i) > 0$, i=1, 2,.... The first state is denoted by *first*(π), the k+1th state of path π is denoted by $\pi(k)$. A prefix of π of length k is defined by $s_0 \rightarrow s_1 \rightarrow \dots \rightarrow s_k$. We define the probability measure *Prob* on the probability space $\langle Path_\omega(s), \Sigma(s)\rangle$, where $Path_\omega(s)$ is the set of infinite paths π with *first*(π)=s, and $\Sigma(s)$ is the smallest σ-algebra on $Path_\omega(s)$ that contains the paths $\{\pi \in Path_\omega(s) : \tau \text{ is a prefix of } \pi\}$ where τ ranges over all finite execution sequences starting in s. The probability measure *Prob* on $Path_\omega(s)$ is the unique measure with

$$Prob\{\pi \in Path_{\omega}(s) : \tau \text{ is a prefix of } \pi\} =$$

$$P(\tau) = P(s_0 s_1 \ldots s_k) = T(s_0, s_1) T(s_1, s_2) \ldots T(s_{k-1}, s_k)$$

Syntax and Semantics of PCTL. PCTL formulas are state formulas, i.e they represent properties of states. Given a probabilistic process P described by a labelled Markov chain $M = (S, s^i, T, L)$, PCTL formulas are defined inductively as follows:

- Each atomic proposition is a state formula.
- If f_1 and f_2 are state formulas, then so are $\neg f_1$ and $(f_1 \wedge f_2)$.
- If f_1 and f_2 are state formulas and t is a nonnegative integer, then $f_1 \mathbf{U}^{\leq t} f_2$ (strong until) and $f_1 \mathbf{U}^{\leq t} f_2$ (weak until) are path formulas.
- If f is a path formula and p is a real number with $0 \leq p \leq 1$, then $[f]_{\geq p}$ and $[f]_{> p}$ are state formulas.

For a given state s, formulas $[f]_{\geq p}$ and $[f]_{> p}$ express that the probability of paths starting in s fulfilling the path formula f is at least p and greater than p, respectively. We discuss only the bounded operators.

The operator **U** is the *strong until* operator and **U** is the *weak until* operator. Intuitively, $[f_1 \mathbf{U}^{\leq t} f_2]_{\geq p}$ means that with a probability of at least p both f_2 will become true within t units and f_1 will hold until f_2 becomes true. $[f_1 \mathbf{U}^{\leq t} f_2]_{\geq p}$ means that there is at least a probability p that either f_1 will remain true for t time units, or that both f_2 will become true within t time units and that f_1 will hold until f_2 becomes true.

The truth of PCTL formulas for a labelled Markov chain $M = (S, s^i, T, L)$ is defined by the satisfaction relation $s \models_M f$ which intuitively means that the state formula is true at state s in M. To define the satisfaction relation for states we also use the relation $\sigma \models_M f$ which intuitively means that the path σ satisfies a path formula f in M. The relations are defined inductively as follows:

$s \models_M a$ if and only if the atomic proposition $a \in L(s)$

$s \models_M \neg f$ if and only if not $s \models_M f$

$s \models_M f_1 \wedge f_2$ if and only if $s \models_M f_1$ and $s \models_M f_2$

$s \models_M f_1 \vee f_2$ if and only if $s \models_M f_1$ or $s \models_M f_2$

$s \models_M f_1 \rightarrow f_2$ if and only if $s \models_M \neg f_1$ or $s \models_M f_2$

$\sigma \models_M f_1 \mathbf{U}^{\leq t} f_2$ if and only if there exists $i \leq t$ such that $\sigma(i) \models_M f_2$ and $\forall j : 0 \leq j < i :$ $\sigma(j) \models_M f_1$

$\sigma \models_M f_1 \mathbf{U}^{\leq t} f_2$ if and only if $\sigma \models_M f_1 \mathbf{U}^{\leq t} f_2$ or $\forall j : 0 \leq j \leq t : \sigma(j) \models_M f_1$

$s \models_M [f_1 \mathbf{U}^{\le t} f_2]_{\ge p}$ if and only if the probability measure of the set of paths σ from s for which $\sigma \models_M [f_1 \mathbf{U}^{\le t} f_2]$ is at least p.

$s \models_M [f_1 \mathbf{U}^{\le t}\ f_2]_{\ge p}$ if and only if the probability measure of the set of paths σ from s for which $\sigma \models_M [f_1 \mathbf{U}^{\le t}\ f_2]$ is at least p.

$s \models_M [f_1 \mathbf{U}^{\le t} f_2]_{> p}$ if and only if the probability measure of the set of paths σ from s for which $\sigma \models_M [f_1 \mathbf{U}^{\le t} f_2]$ is greater than p.

$s \models_M [f_1 \mathbf{U}^{\le t}\ f_2]_{> p}$ if and only if the probability measure of the set of paths σ from s for which $\sigma \models_M [f_1 \mathbf{U}^{\le t}\ f_2]$ is greater than p.

By definition,

$$M \models \Phi \text{ if and only if } s^i \models \Phi$$

i.e., process P satisfies a PCTL formula Φ if and only if its initial state s^i satisfies Φ.

We use the shorthand notation:

$$f_1 \mathbf{U}^{\le t}_{\ge p} f_2 \text{ for } [f_1 \mathbf{U}^{\le t} f_2]_{\ge p}$$

$$f_1\ \mathbf{U}^{\le t}_{\ge p} f_2 \text{ for } [f_1 \mathbf{U}^{\le t}\ f_2]_{\ge p}$$

Formulas $f_1 \mathbf{U}^{\le t}_{> p} f_2$ and $f_1\ \mathbf{U}^{\le t}_{> p} f_2$ have analogous meanings.

Model Checking in PCTL . Next we describe the model checking algorithm, which labels the states of a labelled Markov chain $M = (S, s^i, T, L)$ with the PCTL formula $f_1 \mathbf{U}^{\le t}_{\ge p}\ f_2$. We introduce the function $p(s, t)$ for $s \in S$ and t a non-negative integer, defined as the measure of the set of paths π starting in s which satisfy $f_1\ \mathbf{U}^{\le t}\ f_2$:

$$p(s, t) = Prob\{\pi \in Path_{\omega}(s) : \sigma \text{ is a prefix of } \pi \text{ and } \sigma \models_M f_1 \mathbf{U}^{\le t} f_2\}$$

The probability function $p(s, t)$ satisfies the following recurrence equation. For $t \ge 1$:

$$p(s, t) = \text{if } s \models_M f_2 \text{ then } 1$$

$$\text{else if } s \not\models_M f_1 \text{ then } 0$$

$$\text{else } \sum_{s' \in S} T(s, s') \times p(s', t-1)$$

For $t = 0$, $p(s, t) = \text{if } s \models_M f_2$ then 1 else 0.

The proof of the above recurrence equation can be found in [11].

This recurrence equation leads to the following algorithm, which computes $\bar{p}(s,t)$ and labels states with $f_1 \mathbf{U}^{\le t}_{\ge p} f_2$ if $\bar{p}(s, t)$ is greater or equal to the given probability p.

1. Build the transition probability matrix P

$$P[s_k, s_l] = \begin{cases} T[s_k, s_l] \text{ if } s_k \models f_1 \text{ and } s_k \not\models f_2 \\ 1 \text{ if } k = l \text{ and } s_k \models f_2 \text{ or } (s_k \not\models f_1 \text{ and } s_k \not\models f_2) \\ 0 \text{ otherwise} \end{cases}$$

2. Create a vector $\bar{v}$ indexed by the states such that the i-th element of $\bar{v}$ is set to 1 if $s_i \models f_2$ and 0 otherwise.

3. Compute P^t, the t-th power of the transition probability matrix.

4. Multiply matrix P^t by the vector $\bar{v}$: $\bar{p}(s, t) = P^t\, \bar{v}$

5. $s \models f_1 \mathbf{U}^{\le t}_{\ge p} f_2$ if $\bar{p}(s, t) \ge p$.

6. $M \models f_1 \mathbf{U}^{\le t}_{\ge p} f_2$ if and only if the initial state $s^i \models f_1 \mathbf{U}^{\le t}_{\ge p} f_2$.

3 Multi-terminal Binary Decision Diagrams

BDDs are a canonical representation of boolean functions $f{:}\{0, 1\}^n \to \{0, 1\}$ proposed by Bryant [3]. They are often more compact than traditional normal forms, such as conjunctive normal form and disjunctive normal form, and can be manipulated efficiently. For these reasons, BDDs have found wide use in CAD applications, including symbolic simulation, verification of combinational logic, and verification of sequential circuits. In 1993 Clarke et. al [8] showed that BDDs can be generalized to represent functions $f{:}\{0, 1\}^n \to D$ where D is any set of values. Such diagrams are called multi terminal binary decision diagrams (MTBDDs). MTBDDs can be used to represent D-valued matrices efficiently.

MTBDD Representation of Labelled Markov Chains. Let $M = (S, s^i, T, L)$ be a labelled Markov chain. We fix an ordering of atomic propositions and identify each state with the boolean n-tuple $e(s)$ of atomic propositions that are true in that state. The transition probabilities are arguments of the function $F{:}\{0,1\}^{2n} \to [0,1]$, $F(x_1, y_1,..., x_n, y_n) = P((x_1,..., x_n)(y_1,..., y_n))$ which allows us to represent the transition probability matrix by an MTBDD over $(x_1, y_1,..., x_n, y_n)$.

The following example illustrates the MTBDD representation of the transition probability matrix: consider a single machine which may be in one of the following states: *setup, processing, down*. Figure 1 shows the state transition diagram, which captures the transitions between the possible states. After the setup operation, the machine starts processing, it makes a transition to the processing state with probability 1. In the processing state, there are two possibilities, the machine finishes processing

$$\begin{tikzcd}
& \mathit{Setup}\ 00 \arrow[d, "1"] \\
& \mathit{Processing}\ 01 \arrow[u, bend right=60, "p"'] \arrow[dl, "f"] \\
\mathit{down}\ 10 \arrow[uur, bend left, "1"] &
\end{tikzcd}$$

$$\begin{array}{c|cccc|}
 & 00 & 01 & 10 & 11 \\
00 & 0 & 1 & 0 & 0 \\
01 & p & 0 & f & 0 \\
10 & 1 & 0 & 0 & 0 \\
11 & 0 & 0 & 0 & 1
\end{array}$$

Figure 1. State Transition Graph and corresponding Transition Probability Matrix

and goes to the setup state, or the machine fails. We label the transition *setup* → *processing* with *p* and the transition *processing* → *down* with *f*, such that $p+f=1$. After the machine is repaired, it returns to the *setup* state with probability 1.

For simplicity, we use two atomic propositions, a_1 and a_2, to label the states: $L(\mathit{setup}) = \emptyset$, $L(\mathit{processing}) = \{a_1\}$, $L(\mathit{down}) = \{a_2\}$. We fix the order of a_1 and a_2 and encode the states as follows, $e(\mathit{setup}) = 00$, $e(\mathit{processing}) = 01$, $e(\mathit{down}) = 10$. Figures 2 and 3 show the function *F*, and the corresponding MTBDD respectively.

$$F(x_1, y_1, x_2, y_2) = \begin{cases} 1 \text{ if } x_1y_1x_2y_2 \in \{0001, 1000, 1111\} \\ p \text{ if } x_1y_1x_2y_2 = 0010 \\ f \text{ if } x_1y_1x_2y_2 = 0110 \\ 0 \text{ otherwise} \end{cases}$$

Figure 2. Function $F(x_1,y_1,x_2,y_2)$

4 ProbVerus

ProbVerus is an implementation of PCTL model checking using symbolic techniques. It is an extension of Verus [6], a verification tool which combines powerful symbolic model checking techniques and efficient quantitative algorithms for computing minimum and maximum time delays between two events. Verus has been already used to verify several real-time systems: an aircraft controller, the PCI local bus, and a robotics controller [4, 5]. By extending Verus with PCTL model checking and MTBDDs we have created a single verification environment in which we have access to correctness checking, performance analysis, and reliability analysis of a single model of the system.

ProbVerus features a language which is designed especially to simplify writing probabilistic real-time programs. It is based on the core Verus language, an imperative

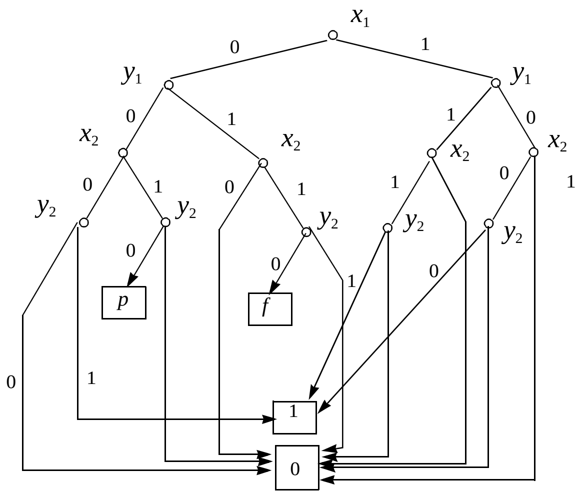

Figure 3. MTBDD representation of the Transition Probability Matrix

language with a syntax similar to C, that is enriched by special constructs that allow the straightforward expression of timing and stochastic behavior. Designers can use ProbVerus to verify models that are very close to the actual implementation expressing time and probabilities in a natural and accurate way.

Syntax of the ProbVerus Language. The ProbVerus language includes primary expressions, boolean expressions, assignments, sequential execution, conditionals, probabilistic choice, and loops:

stmt ::= `wait`(1) | $v = expr$ | stmt$_1$; stmt$_2$ |

`while` *cond* stmt | `if` *cond* stmt$_1$ `else` stmt$_2$

`pselect`(p_1 : stmt$_1$; ... p_m : stmt$_m$)

where $p_1 \ldots p_m \in [0, 1]$ with $p_1 + \ldots + p_m = 1$.

where

- *Var* is a set of boolean variables
- $v \in Var$
- *expr* are Verus expressions
- *expr*::= `true` | `false` | v | $expr_1$ || $expr_2$ | $expr_1$ && $expr_2$ | $!expr$
- *cond* is a boolean expression

ProbVerus only has boolean variables. The compiler introduces an auxiliary variable, the *wait counter wc*, which indicates the `wait` statement reached by the program in the current state. The integer *wc* is encoded as the respective sequence of booleans to suit the format. The length of the bitstring needed to represent the *wait counter* is fixed and the values of *wc* in a specific program are determined at compile time of that program.

Semantics of ProbVerus Programs. ProbVerus programs describe Markov chains, the behavior of which is specified by the initial state and the transition probability matrix.

A *state* in the model corresponds to an assignment of values to the state variables $v_1, v_2, \ldots, v_n$ and is represented by the vector $\langle v_1, \ldots, v_n \rangle$. There are two copies of the state variables: one for the current state ($v \in V$) and one for the next state ($v' \in V'$). A *transition* is a relation between states. For two states $s = \langle v_1, \ldots, v_n \rangle$ $s' = \langle v'_1, \ldots, v'_n \rangle$, the transition relation $N(s, s')$ evaluates to true when there is a transition in the model from state s to state s'. There is a transition between two states s and s' when the code between two successive `wait` statements changes state s to state s'. An important feature of the tool is the use of the `wait` statement to synchronize concurrent processes, control time (time elapses by one unit at each `wait`), and update the values of the global program states. By executing a block of code between two consecutive `wait` statements atomically, thus collapsing contiguous statements to one transition, leads to models with fewer states, which allows the verification of much larger systems. The *transition relation* of the program is obtained by taking the disjunction of all relations between `wait` statements. A *path* in the transition graph is defined as a sequence of states $s_0, s_1, s_2, \ldots$ such that $N(s_i, s_{i+1})$ is true for every $i \geq 0$. All computations are performed on states reachable from the initial state set.

Our stochastic model does not consider nondeterminism; we define one initial state by assigning fixed values to all the state variables before the first `wait` statement. Verus restricts the set of accepted programs to those for which a `wait` statement is traversed at each loop iteration. The last statement of every accepted program ends with the statement

```
while (true) wait(1);
```

which guarantees that the final state is observed.

The semantics of ProbVerus programs is computed statement by statement using the following function R:

$$R\colon \text{stmt} \times M \times M \times \mathrm{N} \rightarrow M \times M \times \mathrm{N} \times B$$

where M is the set of matrices m: $ST \times ST \rightarrow [0,1]$, ST is the state space, N is the set of naturals, B is the set of booleans.

R has the following arguments:

- statement `stmt`,
- matrix $r{:}ST \times ST \rightarrow [0,1]$ containing the probabilities of the transitions in the program since the last `wait` statement, and
- transition probability matrix $m{:}ST \times ST \rightarrow [0, 1]$ accumulated between the previously executed `wait` statements
- $w \in N$ represents the number of syntactic occurrences of the `wait` statement encountered so far

Given the matrix *r*, the matrix *m*, and the counter *w*, which describe the program until the execution of statement `stmt`, function $R[\![$`stmt`$]\!] \langle r, m, w\rangle$ produces the matrix r', the matrix m', the counter w', which describe the program after executing `stmt`, and determines the value of $b \in B$, which is *true* if a `wait` statement is traversed in each control path through `stmt`.

[15] defines $R[\![$`stmt`$]\!] \langle r, m, w\rangle$ for the individual statements and includes a proof that the global transition probability matrix of a ProbVerus program is a stochastic matrix, i.e. the entries of the matrix are real numbers in the [0, 1.0] range and the sum of the elements of each row is equal to 1.

Implementation of PCTL Model Checking. We have implemented the *strong until* operator (algorithm on page 101), and the $\mathbf{F}^{\le t}_{\ge p}$, and $\mathbf{G}^{\le t}_{\ge p}$ operators and from the dual formulas [11]:

$$\mathbf{F}^{\le t}_{\ge p} f \equiv \mathit{true}\ \mathbf{U}^{\le t}_{\ge p} f$$

$$\mathbf{G}^{\le t}_{\ge p} f \equiv \neg\, \mathbf{F}^{\le t}_{>(1-p)} (\neg f)$$

All probability calculations are performed with MTBDDs and make use of the following operators:

APPLY Operator: The *APPLY* operator allows elementwise application of the binary operator *op* to two MTBDDs. If Q_1 and Q_2 are two MTBDDs, and *op* is an operation on the real numbers, then *APPLY*(Q_1, Q_2, *op*) yields an MTBDD which represents the function $f(Q_1)$ *op* $f(Q_2)$, where $f(Q_1)$ is the formula associated with Q_1, and $f(Q_2)$ the formula associated with Q_2.

Matrix and Vector Operators: The standard operations on matrices and vectors have corresponding operations on the MTBDDs that represent them, such as multiplication of a matrix by a vector or multiplication of two matrices. These operations benefit from the recursive structure of the MTBDD-representation of the respective matrices.

5 A Manufacturing Example

In this section we illustrate the use of ProbVerus with a simple example. We model the stochastic behavior and check stochastic properties of a small automated manufacturing system comprised of two numerically controlled machines *M1* and *M2* that are identical in all respects. Each machine is modeled as a Discrete Time Markov Chain (DTMC) that has three states: (0) the machine is being setup, (1) the machine is processing, and (2) the machine is undergoing repair following a breakdown. Figure 4 shows the state transition diagram, which captures the transitions among these three states. Each arc is labeled with the corresponding one-step transition probability; each transition corresponds to advancing the discrete time one time unit.

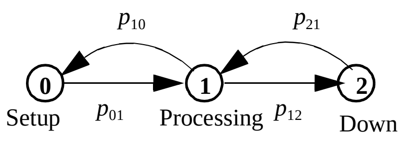

Figure 4. DTMC model of fail-prone machine under resume policy

After the setup operation the machine starts processing, hence after state 0 the next state is with certainty state 1 (p_{01}=1). If the current state is state 1, the next state is state 0 or state 2 with probabilities

$$p_{10} = \text{Prob}\{\text{processing finishes before failure}\}$$

$$p_{12} = \text{Prob}\{\text{failure occurs before finishing of processing}\}$$

After undergoing repair the machine resumes the processing operation, hence if the current state is state 2, then the next state is always state 1 (p_{21}=1).

We can describe the behavior of the above machine in ProbVerus as follows.

Figure 6 displays the DTMC model of the AMS comprised of the two machines M1 and M2. In state **s0** the two machines are being setup. In state **s1** one of the two machines is processing and in state **s2** both machines are processing. While a machine is processing, there are two possibilities: either the machine finishes and returns to the setup state or the machine fails. In **s3** one of the machines fails which then moves for repair in state **s4**. The system is down when both machines are under repair (state **s5**).

Using ProbVerus, we have modeled the AMS of Figure 6 and checked the PCTL formula $\mathbf{F}^{\leq t}_{\geq p}$ `AMS_is_down`, which states that AMS will fail, i.e. that it will reach a state labeled with `AMS_is_down` (machine M1 is down and machine M2 is down), within *t*

```
Machine = Setup;
wait(1.0);

while (Machine != Down) {
      if ( Machine == Setup) {
            Machine = Processing;
            wait(1.0);
      }
      else if ( Machine == Processing) {
            pselect {p10: Machine = Setup; p12: Machine = Down;};
            wait(1.0);
      }
}
Machine = Processing;

while (true) {wait(1.0);}
```

Figure 5. ProbVerus model of a fail-prone machine

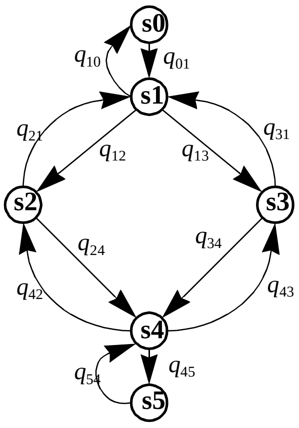

Figure 6. State Transition Graph of a Two-Machine Manufacturing System

units after the setup of the system with non-zero probability p. We have used the following one-step transition probabilities:

q_{01}=1,

q_{10}=0.1663, q_{12}=0.8316, q_{13}=0.0021,

q_{21}=0.987654, q_{24}=0.0123456,

q_{31}=0.04762, q_{34}=0.95238,

q_{42}=0.19802, q_{43}=0.79208, q_{45}=0.0099,

q_{54}=1,

to compute the probabilities of reaching the state where both machines are down for the first 5 time units of operation of the AMS, as seen in Table 1. For example, after 4 time units all states are labeled with the $\mathbf{F}^{\leq 4}_{\geq 0,00012}$ `AMS_is_down` since $P(s_i,4) \geq 0{,}000121$ for all the states s_i. Such properties are expressible in PCTL since PCTL formulas are interpreted over DTMCs, where each transition of the Markov chain corresponds to one time unit.

Table 1. Successive values of p of $\mathrm{F}^{\leq t}_{\geq p}$ `AMS_is_down`

state/ time	**0**	**1**	**2**	**3**	**4**	**5**
s_0	0	0.000000	0.000000	0.000000	0.000121	0.000121
s_1	0	0.000000	0.000000	0.000000	0.000121	0.000333
s_2	0	0.000000	0.000122	0.000122	0.000334	0.000335
s_3	0	0.000000	0.009430	0.009430	0.016570	0.016570
s_4	0	0.009901	0.009900	0.017394	0.017394	0.023100
s_5	1	1	1	1	1	1

6 Current Status

We have been testing the modeling range of ProbVerus with success in a series of application domains: manufacturing, transportation, and fault-tolerant industrial process control systems. The example, that we have described, demonstrates that ProbVerus provides the language to describe the stochastic behavior of manufacturing systems in a straightforward manner. PCTL allows the expression of the stochastic properties, which is critical information when designing automated manufacturing systems and analyzing their performance, availability, and reliability. Moreover, probabilistic model checking allows us to combine the performance (quantitative timing analysis) and reliability (quantitative probability analysis) early in the design phase of manufacturing systems. This is significant because real-world automated manufacturing systems must meet competitive levels of productivity and pay-back ratio where high costs are involved.

An important question is how our techniques scale up to large systems. Our largest case study is based on ACC ("Apparato Centrale a Calcolatore") [14], a complex industrial safety-critical system developed by Ansaldo Transporti for the control of medium and large-size railway stations. A set of qualitative (e.g. safety, liveness) and quantitative (e.g. response times and probabilities) properties have been automatically analyzed. Despite the complexity of the system (the model has about 10^{27} states) specifications were checked in 329 seconds using a Sparc 10 with dual processors and 256Mb of memory. During the verification of the ACC design, we discovered a subtle and anomalous behavior leading to a deadlock of the system. The anomalous behavior was pinpointed by an automatically generated counterexample trace, showing precisely the behavior leading to the violating state. The same behavior blocked the entire operation during a field test of an earlier version of the system. When we modeled possible failures in the communication between the controller and the physical devices, i.e. when the control variables of the *safety logic* are not in agreement with the values of the corresponding physical level crossing variables, then safety specifications were violated. Therefore, a failure in the sensing operation may lead to a `CLEAR_SIGNAL` although the gate may be still moving, or the actual level crossing is not closed. We have computed the probability of reaching unsafe states within seconds. Our numerical analysis has been performed assuming a serial link between the controller and the physical devices for which reliability information is published. Vital railway controllers, such as the ACC system, implement appropriate error recovery mechanisms for the occurrence of vital/non-vital communication failures, which have not been modeled. Our probabilistic analysis has intended to illustrate the valuable information probabilistic model checking can provide to the engineers who design and validate safety-critical systems.

7 Conclusions and Future Work

This paper presents a new tool for the formal verification of stochastic systems. We have implemented PCTL model checking using multi terminal binary decision diagrams (MTBDDs) within the Verus verification tool. The PCTL logic allows the expression of time and probabilities which are needed for the verification of stochastic systems, such as fault-tolerant real-time control systems, networks, and manufacturing systems. BDDs and MTBDDs provide a compact representation of the model by representing sets of states rather than individual states. Moreover, symbolic techniques are amenable to efficient verification algorithms. By extending Verus, we have created an environment which allows the verification and quantitative analysis of a single model using CTL, RTCTL, and PCTL model checking. While model checking can tell us whether a system is correct, probabilistic model checking can also tell us whether the system is timely and reliable. Moreover, an advantage of probabilistic model checkers over non-probabilistic model checkers is that it allows one to verify properties that may not be true with probability equal to one, but may still hold with some acceptable probability. ProbVerus allows the safety analysis and verification to take place concurrently with the system design so that design errors are detected and corrected prior to the traditional test phase. The use of symbolic techniques (BDDs and MTBDDs) in contrast to an explicit-state implementation of PCTL model checking makes it possible to analyze larger systems making probabilistic symbolic model checking a feasible approach worth further development.

References

1. R. Alur, C. Courcoubetis, D. Dill. Model Checking for Probabilistic Real-time Systems. Automata, Languages, and Programming 18th International Colloquium Proceedings; Madrid, Spain; 8-12 July 1991.
2. A. Aziz, V. Singhal, F. Balarin, R. K. Brayton, A.L. Sangiovanni-Vincentelli. It usually works: the temporal logic of stochastic systems. In Proceedings of CAV'95.
3. R. E. Bryant. Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers, C-35(8), 1986.
4. S. Campos, E. Clarke, W. Marrero, M. Minea and H. Hiraishi, Computing quantitative characteristics of finite-state real-time systems. In IEEE Real-Time Systems Symposium, 1994.
5. S. Campos and E. Clarke, Real-time symbolic model checking for discrete time models. In AMAST Series in Computing: Theories and Experiences for Real-Time System Development. T. Rus, C. Rattray, editors. World Scientific Publishing Company, 1995.
6. S. V. Campos. A Quantitative Approach to the Formal Verification of Real-Time Systems. Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, 1996.
7. E. M. Clarke, E. A. Emerson. Synthesis of Synchronization Skeletons for Branching Time Temporal Logic. Logic of Programs: Workshop, Yorktown Heights, NY, May 1981, volume 131 of Lecture Notes in Computer Science. Springer Verlag, 1981.
8. E. M. Clarke, M. Fujita, P.C. McGeer, K. McMillan, J. C.-Y. Yang, X. Zhao. Multi-Terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation. IWLS '93: International Workshop on Logic Synthesis. Tahoe City, May 1993.
9. C. Courcoubetis, M. Yannakakis. The Complexity of Probabilistic Verification. Journal of the Association for Computing Machinery, Vol. 42, No. 4, July 1995.
10. G. Hachtel, E. Macii, A. Pardo, F.Somenzi. Markovian Analysis of Large Finite State Machines. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 15. No. 12, December 1996.
11. H. Hansson and B. Jonsson. A Framework for Reasoning about Time and Reliability. In Proceedings of 10th IEEE Real Time System Symposium, pp. 102-111, 1989.
12. H. Hansson. Time and Probability in Formal Design of Distributed Systems. Elsevier, 1994.
13. S. Hart and M. Sharir. Probabilistic Temporal Logics for Finite and Bounded Models. In Proceedings of the ACM Symposium on the Theory of Computing, pp. 1-13, 1984.
14. V. Hartonas-Garmhausen, S. Campos, A. Cimatti, E. Clarke, F. Giunchiglia. Verification of a Safety-Critical Railway Interlocking System with Real-Time Constraints. In Proceedings of FTCS-28 The Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing, pages 458-463, June, 1998.
15. V. Hartonas-Garmhausen. Probabilistic Symbolic Model Checking with Engineering Models and Applications. Ph.D. Thesis. Carnegie Institute of Technology, Carnegie Mellon University, 1998.
16. D. Lehmann and S. Shelah. Reasoning with time and chance. Information and Control 53, pp. 165-198, 1982.
17. M. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In Proceedings of the 26th IEEE Symposium on Foundations of Computer Science. IEEE, New York, pp. 327-338, 1985.