A Modest Approach to Checking Probabilistic Timed Automata

Arnd Hartmanns, Holger Hermanns
Universität des Saarlandes
Saarbrücken, Germany
Email: {arnd, hermanns}@cs.uni-sb.de

Abstract—Probabilistic timed automata (PTA) combine discrete probabilistic choice, real time and nondeterminism. This paper presents a fully automatic tool for model checking PTA with respect to probabilistic and expected reachability properties. PTA are specified in Modest, a high-level compositional modelling language that includes features such as exception handling, dynamic parallelism and recursion, and thus enables model specification in a convenient fashion. For model checking, we use an integral semantics of time, representing clocks with bounded integer variables. This makes it possible to use the probabilistic model checker PRISM as analysis backend. We describe details of the approach and its implementation, and report results obtained for three different case studies.

Keywords—Probabilistic timed automata; digital clocks; model checking; compositional modelling

I. INTRODUCTION

Model checkers for probabilistic systems, especially PRISM [1], have gained considerable popularity, with a total number of downloads in the order of tens of thousands. The same holds, even more so, for real-time model checkers, with UPPAAL [2] being the most popular representative. One of the core models on which the PRISM engine operates is the model of probabilistic automata (PA) [3], while UPPAAL revolves around timed automata (TA) [4].

Probabilistic timed automata (PTA) combine discrete probabilistic choice, real time and nondeterminism. They arise as the natural orthogonal combination of probabilistic and timed automata. Since PTA overarch PA and TA, the potential audience for a fully automatic PTA model checker is immense.

This paper presents mcpta, a tool that enables the fully automatic analysis of PTA via model checking. mcpta supports probabilistic and expected reachability properties. It uses PRISM as its backend solution engine. PTA are specified in Modest [5], a high-level compositional modelling language that includes features such as exception handling, dynamic parallelism and recursion, and thus enables model specification in a convenient fashion. The idea of model checking PTA is not new, and there has been a wealth of theoretical and practical results. The existing approaches can be classified into two broad categories: Symbolic techniques, based on either forwards [6], [7] or backwards reachability [8], and the digital clocks approach [9]. Table I summarises their properties: While the backwards reachability approach allows checking full logical formulas, the digital clocks approach, although limited to closed and diagonal-free PTA, covers both probabilistic and expected reachability properties. Notably, the earliest abstraction-refinement based probabilistic model checker, RAPTURE [10], was designed with abstraction techniques in mind particularly tailored to a digital clocks encoding of PTA [11].

These approaches all document the feasibility of model checking PTA, nevertheless there is as yet no tool available that accepts the model of PTA (in some form) and is fully automatic. The unavailability of tool support has recently led to increased implementation activities, in particular in the European FP7 STREP project Quasimodo1. We present our Modest approach here. For model checking, we use the digital clocks semantics [9] which we adapt in order to translate Modest specifications to properly synchronised collections of PA. These PA are encoded as PRISM modules, and we thus exploit the static parallelism available inside PRISM. So, mcpta can also be viewed as providing another syntactic frontend for PRISM, with enriched expressiveness.

Since Modest in full generality allows the use of continuous probability distributions, arbitrary recursion, exception handling, and nested and dynamic parallelism, we can obviously not support the entire language. Indeed, our translation disallows the use of continuous distributions. However, we do support exception handling as well as limited forms

---

1 http://www.quasimodo.aau.dk/

---

Table I

<table>
<thead>
<tr>
<th>Results</th>
<th>Properties</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>forwards reachability</td>
<td>upper bounds</td>
<td>feasibility study</td>
</tr>
<tr>
<td>[6], [7]</td>
<td>max. probabilistic reachability</td>
<td></td>
</tr>
<tr>
<td>backwards reachability</td>
<td>exact</td>
<td>prototype, unavailable</td>
</tr>
<tr>
<td>[8]</td>
<td>full PTCTL</td>
<td>manual</td>
</tr>
<tr>
<td>digital clocks</td>
<td>exact</td>
<td>transformations + PRISM</td>
</tr>
<tr>
<td>[9]</td>
<td>full probabilistic and expected reachability</td>
<td></td>
</tr>
</tbody>
</table>
of recursion and dynamic parallelism. The details of the translation are explained in this paper, together with a description of the tool, and a selection of case studies.

II. BACKGROUND

This section provides the definition of probabilistic timed automata, gives an introduction to the Modest language, and investigates the relation between Modest and PTA. Let us begin with some preliminaries along the lines of [8]:

Probability distributions: A discrete probability distribution over a countable set $Q$ is a function $\mu : Q \to [0,1]$ such that $\sum_{q \in Q} \mu(q) = 1$. Let $\text{Dist}(Q)$ denote the set of all probability distributions over $Q$. The support of a distribution $\mu$, $\text{support}(\mu)$, is the largest set $Q' \subseteq Q$ such that $\mu(q) > 0$ for all $q \in Q'$. We call $\mu \in \text{Dist}(Q)$ finite if $\text{support}(\mu)$ is finite. If its support is a singleton set $\{q\}$, $\mu$ is the point distribution for $q$, denoted $D(q)$.

Clocks: Clock variables, or clocks, keep track of time, which advances at a constant rate that is the same for all clocks. They take values from a time domain, in our case $\mathbb{R}_+$. Clock constraints are expressions of the form

$$ CC ::= \text{true} \mid \text{false} \mid \neg CC \mid CC \land CC $$

where $x, x_1, x_2$ are clock variables and constants $t \in \mathbb{N}$ are used. For a set of clock variables $\mathcal{X}$, $CC(\mathcal{X})$ is the set of clock constraints over $\mathcal{X}$ according to this grammar. As usual for timed automata, the only value that can be assigned to a clock variable is zero, thus resetting the clock.

A. Probabilistic Timed Automata

Syntax: We follow the definition of PTA given in [9], which includes action labelled edges and nondeterminism: A PTA is a 6-tuple

$$(L, l_0, \mathcal{X}, \Sigma, \text{inv}, \text{prob})$$

where $L$ is the set of locations, $l_0$ is the initial location, $\mathcal{X}$ is a finite set of clock variables, $\Sigma$ is a finite set of actions, $\text{inv} : L \to CC(\mathcal{X})$ is a total function labelling locations with invariants—clock constraints that prevent time from passing when not satisfied—and

$$ \text{prob} \subseteq L \times CC(\mathcal{X}) \times \Sigma \times \text{Dist}(2^\mathcal{X} \times L) $$

is the probabilistic edge relation. An edge $(l, g, a, \mu) \in \text{prob}$ consists of an originating location $l$, a guard $g$ that determines whether the edge is enabled, an action label $a$ and a target distribution over pairs of locations and clock variables that are to be reset to zero.

A PTA is finite if its set of locations is finite, and it is well-formed [12], [8] if every edge that is enabled can be taken without violating the invariant of any destination location.

Example 1 Figure 1 shows the graphical representation of a small example PTA, adapted from a similar example in [9]. It models a communication scenario with a channel that has a transmission delay between 1 and 3 time units and loses messages with probability 1/100. In order to save space when drawing PTA, we usually omit the edge label $\tau$, guards that are constant true, and empty assignments in the sequel.

Semantics: The semantics of probabilistic timed automata is defined in terms of probabilistic timed transition systems (PTS), which are 4-tuples $(S, s_0, T, \rightarrow)$ where $S$ is the set of states, $s_0$ is the initial state, $T = \Sigma \uplus \mathbb{R}_+$ comprises the transition labels, partitioned into actions and delays, and $\rightarrow \subseteq (L, T, \text{Dist}(L))$ is the probabilistic transition relation. Transitions labelled with delays have to be deterministic, i.e. the third component has to be a point distribution, and they have to satisfy two additional conditions, time additivity and time determinism. The details of the semantic mapping from PTA to PTS follow the mapping from TA to timed transition systems [4]. Since these details do not influence the topic of the paper, we refer to [8], [9] for them.

Parallel Composition: It is often useful to describe complex systems as the parallel composition of several independently specified but possibly interacting components. We use CSP-style mandatory synchronisation on the actions of the shared alphabet. Given PTA

$$ A_i = (L_i, l_{i0}, \mathcal{X}_i, \Sigma_i \cup \{\tau\}, \text{inv}_i, \text{prob}_i) $$

with silent action $\tau$, the parallel composition $A_1 \parallel A_2$ is [9]:

$$ A_1 \parallel A_2 = (L_1 \times L_2, l_0, \mathcal{X}_1 \cup \mathcal{X}_2, \Sigma_1 \cup \Sigma_2 \cup \{\tau\}, \text{inv}, \text{prob}) $$

where $l_0 = (l_{01}, l_{02})$, for all $(l_1, l_2) \in L_1 \times L_2$ we have

$$ \text{inv}((l_1, l_2)) = \text{inv}_1(l_1) \land \text{inv}_2(l_2) $$

and $((l_1, l_2), g, a, \mu) \in \text{prob}$ if and only if

$$ a \in \Sigma_1 \cup \{\tau\} \land \Sigma_2 \land (l_1, g_1, a, \mu_1) \in \text{prob}_1. $$

$$ \mu = \mu_1 \cdot \text{D}((\emptyset, l_2)) $$

or $a \in \Sigma_2 \cup \{\tau\} \land \Sigma_1 \land (l_2, g_2, a, \mu_2) \in \text{prob}_2. $$

$$ \mu = \text{D}((\emptyset, l_1)) \cdot \mu_2 $$

or $a \in \Sigma_1 \land \Sigma_2 \land (l_1, g, a, \mu_i) \in \text{prob}_i$ for $i = 1, 2$. 

$$ g = (g_1 \land g_2) \land \mu = \mu_1 \cdot \mu_2 $$

where . is the product operation on probability distributions:

$$ (\mu_1 \cdot \mu_2)(X_1 \cup X_2, (l_1, l_2)) = \mu_1(X_1, l_1) \cdot \mu_2(X_2, l_2) $$

![Figure 1. A probabilistic timed automaton](image-url)
Variables: Transition systems can be extended with discrete-valued variables in a natural way (see e.g. [13, Chapter 2] for the general recipe). The same is often done for TA, for PA, and is also possible for PTA; we call this class of variable decorated automata VPTA. The variables can occur in guards, invariants and assignments.

A VPTA $A_V$ can be transformed into a standard PTA $A$ by encoding the values of these variables in the locations: Every location $l \in L$ of $A_V$ is replaced by a location $(l, v) \in L \times Val$ where $Val$ is the set of valuations for the discrete variables of the VPTA, the subformulas of guards and invariants containing discrete variables are precomputed according to the valuation of the corresponding location, and assignments to discrete variables redirect edges to locations with matching valuations. Note that $A$ is finite iff $A_V$ is finite and the ranges of all discrete variables used are finite. In this case, we say that the VPTA is finitary.

Random assignments, like $x := \text{DiscreteUniform}(0, 5)$ to select an integer value between 0 and 5 according to the uniform distribution, can be allowed in VPTA by representing them as a probabilistic choice between locations with the corresponding valuations in the resulting PTA.

Example 2 We have extended the communication scenario of Figure 1 with a limit on the number of retries by adding an integer variable that could have bounded range $\{0, \ldots, 4\}$ in Figure 2.

Deadlines: Our PTA definition is based on the standard model of TA with location invariants. An alternative model is that of TA with deadlines [14], where time progress is restricted using deadlines or urgency constraints associated with edges. Based on this, the probabilistic edge relation of PTA with deadlines, or DPTA, can be defined as

$$\text{prob} \subseteq L \times \text{CC}(\mathcal{X}) \times \text{CC}(\mathcal{X}) \times \Sigma \times \text{Dist}(2^X \times L)$$

where the second clock constraint is the deadline of the edge. Semantically, where in the PTA setting time can pass in a location just as long as the invariant holds, this is only possible in a location of a DPTA as long as no deadline of any outgoing edge evaluates to $true$.

When considering the parallel composition of DPTA, deadlines are strictly more expressive than location invariants. Since this is due to synchronisation, as long as the edges with deadlines cannot synchronize, deadlines can be encoded in location invariants and vice-versa. We explore this in more detail in Section III-B.

B. PRISM

Since we map to PRISM code, we briefly review the PRISM modelling language.

Guarded commands and modules: The language provides syntax to specify collections of PA with variables, VPA, that run concurrently according to the parallel composition semantics described above. The number of parallel components does not change over the lifetime of the model, so we call this type of parallelism “static” parallelism. Each VPA is finitary and is specified as a PRISM module. Such a module contains a number of bounded integer (or Boolean) variables and a set of guarded commands of the form

$$[\text{label}] \text{ guard } \rightarrow p_1 : A_1 + \cdots + p_n : A_n$$

that represent edges labelled $\text{label}$ and enabled iff $\text{guard}$, a predicate over the variables, evaluates to $true$, that lead to a distribution over assignments to variables $A_i$ according to the probabilities given by the real-valued expressions $p_i$.

Composition expressions: PRISM also supports process-algebraic expressions for relabelling and hiding of actions as well as for finer control of the synchronisation alphabets. An example is given in Figure 3.

C. Modest

Modest is a “modelling and description language for stochastic timed systems” [5]. It combines a compositional modelling approach with concepts known from modern programming languages (e.g. exceptions and exception handling) and extends these with probabilistic branching, continuous probability distributions and time.

Example 3 To get a first impression of the language, Figure 4 shows a Modest formulation of the communication scenario with discrete variables introduced in Figure 2. It contains the most frequently appearing constructs and concepts like probabilistic and nondeterministic choice (palt and alt), parallel composition of processes (par), and loops (do). The Modest fragments in Figures 5, 8 and 9 exhibit further constructs like the exception-related keywords throw, try and catch.

Semantics: Modest has a two-step semantics: First, an operational semantics associates a stochastic timed automaton (STA) with each Modest process. STA are VPTA with deadlines where both variables and probability distributions need not be discrete. The STA, in turn, has a semantics in
This section discusses how we translate a Modest specification into a collection of PRISM modules. In order to illustrate the main steps, we proceed by example, using a modular and rather elegant Modest version of the simple communication protocol introduced in the previous section. In particular, since upper-layer protocols usually rely on an independent lower layer for the actual transmission, we use a dedicated process `LinkLayerSend` (Figure 5) modelling a link layer that loses messages with 1% probability or reports an acknowledgment after at most 3 time units. Should we want to model a different and possibly more involved link layer, we would simply modify or replace this one process.

### A Translational Checking Approach

We first present the most basic transformation, namely the translation of a single Modest process like `LinkLayerSend` into a module in PRISM’s guarded command language. The first step is to derive the VPTA corresponding to the process according to Modest’s operational semantics (Figure 6). We then add a new integer `state variable s` to enumerate the locations of the PTA and transform the edges into guarded commands as shown in lines 6 to 8 in Figure 7. We follow the ideas of [9] here, including the translation of the timed behaviour:

Clocks, in accordance with the integral semantics, are represented by bounded integer variables where the upper bound is the maximum constant the variable is ever compared to plus one. Time advances in increments of one on transitions labelled `tick`, where the guards are used to represent the constraints on time introduced by the location

---

**Figure 5.** Link-layer sending part in Modest

**Figure 6.** PTA generated for the link-layer sending part

**Figure 7.** PRISM module generated for the link-layer sending part
invariants. In Figure 7, this can be seen in lines 9 and 10. An overall similar transformation was given for timed safety automata [15], although no encoding of time was necessary due to the target language already having a notion of time.

B. Parallel composition of processes

PRISM’s parallel composition of modules can be exploited to compose several Modest processes, each represented as a guarded command module. This is possible because Modest and PRISM share the same synchronisation mechanism. PRISM’s process-algebraic expressions can be used to represent the respective Modest constructs such as relabel or extend as well.

Consider, for instance, another protocol component of our Modest example, shown in Figure 8. This component waits for an acknowledgment to arrive and throws an exception on timeout. We can directly represent the parallel composition of this process and the link-layer send process considered before in PRISM by simply including the two modules generated for the two Modest models in one file. Notably, giving the time increment transitions the label tick and thereby enforcing synchronisation leads to the desired behaviour in parallel composition: Time advances synchronously and at the same rate in all modules, and since invariants are represented by guards, the global invariant is the conjunction of the current location invariants of all modules.

If we were to allow arbitrary deadlines on edges instead of location invariants, we would encounter problems during parallel composition because an edge labelled a out of location \( l_A \) in automaton A may no longer be possible in location \( (l_A,l_B) \) in \( A \parallel B \) if, for example, a is in the shared alphabet, but there is no edge labelled a out of \( l_B \) in B. Whether a deadline affects a location after parallel composition thus depends on its possible synchronisation partners, so correct composition of deadlines is a feature of the Modest parallel composition semantics, which the one in PRISM lacks (since PRISM has no concept of deadlines).

We can therefore only support deadlines on non-synchronising edges. Using just Modest’s invariant construct or restricting urgency constraints to \( \tau \)-labelled edges are two possible ways to ensure that the use of deadlines is provably safe. mctpa issues a warning if urgency constraints appear on potentially synchronising edges.

C. Dynamic parallelism: Modest vs. PRISM

To complete the elegant Modest implementation of the communication protocol, we want to encapsulate the parallel composition of LinkLayerSend and WaitDone in a coordinating process as shown in Figure 9. This process represents the unbounded variant of the communication protocol with an additional initialisation phase, as illustrated by the corresponding PTA in Figure 10. However, it contains dynamic parallelism, i.e. the parent process initially does some work, and at a later point spawns child processes that can communicate with the parent by successfully terminating, by issuing a break, or, as in the example, by throwing an exception. The parent can react on these events, for example by catching some of the exceptions. Using recursion, a parent process could also spawn children and call itself again as another part of the parallel composition, which can lead to an unbounded number of possible processes running in parallel. PRISM’s static parallel composition of modules clearly is not sufficient to represent this.

However, if we restrict the use of recursion by only allowing tail-recursive process calls\(^2\), the set of parallel processes is bounded. In this case, we are able to create one module for every possible process such that the child modules stay inactive until they are activated by their parent module. In addition to that, the possible interactions between parent and children have to be suitably encoded.

We can thus identify the following requirements for a correct representation of parent and child processes: Each child

- must be able to be activated and deactivated,

\(^2\)More precisely: We only allow a cycle on the call graph as long as the calls on the cycle are from within the last element of any sequential composition and not below any static operators such as \texttt{do} or \texttt{relabel}.\n
\begin{figure}[h]
\centering
\begin{verbatim}
1 process WaitDone() {
2   invariant (c <= 6) alt {
3     :: when (c >= 4) throw(timeout)
4     :: done
5   }
6 }
\end{verbatim}
\caption{Timeout in Modest}
\end{figure}

\begin{figure}[h]
\centering
\begin{verbatim}
1 process CommProtocol() {
2   try {
3     init; // initialise
4     par {
5       :: LinkLayerSend()
6       :: WaitDone()
7     }
8   }
9     catch timeout {
10    CommProtocol() // retry
11  }
12 }
\end{verbatim}
\caption{A very modular communication protocol model}
\end{figure}

\begin{figure}[h]
\centering
\begin{verbatim}
\end{verbatim}
\caption{TA representation of the modular communication model}
\end{figure}
1 module WaitDone
2 s2 : [0..2] init 0;
3 e2 : bool init false;
4 f2 : bool init false;
5
6 [timeout] e2 & s2=0 & c>=4 & c<=6 \\
7  \Rightarrow 1:(s2'=0);
8 [done] e2 & s2=0 & c<=6 \\
9  \Rightarrow 1:(s2'=2) & (f2' = true);
10 [tick] e2 & s2=0 & c < 6 \Rightarrow 1: true;
11 [tick] e2 & (s2=1 | s2=2) \Rightarrow 1: true;
12 [reset] e2 & s2=0 \Rightarrow 1: true;
13 [reset] true \Rightarrow 1: (s2'=0) & (f2' = false);
14 endmodule

Figure 11. Child module with timeout in PRISM

- must not change the system’s behaviour when inactive,
- must signal successful termination, break and exceptions to the parent, and
- must offer the possibility to be reset to its initial state.

The last requirement is crucial to allow for multiple activations, e.g. when inside a do. The parent, in turn
- must activate the correct children at the correct moment,
- must continue execution when all children terminate,
- must react appropriately whenever one child throws an exception or issues a break.

Based on these requirements, we have determined a set of transformations on the level of the Modest code that, if applied to a process containing dynamic parallelism, results in one that contains only static parallelism with parent and child processes satisfying their respective requirements. Due to the expressiveness of the Modest language, the transformations are technically quite involved but can be modularised. The correctness of the entire transformation is assured by two observations: Each transformation can be decomposed into a finite sequence of basic steps, that are either (i) inserting silent steps, or (ii) duplicating locations and splitting their incoming edges. (The first is the PTA equivalent of the process algebraic rewrite rule \( a.P \Rightarrow a.\tau.P \), while the second is akin to \( a.P + b.P \Rightarrow a.P + b.Q \) with \( P \) and \( Q \) isomorphic.) Both basic steps are easily shown to preserve weak probabilistic bisimilarity on the underlying PTS, which is thus inherited by the entire transformation. We refer to [16] for the technical details, and instead present two technical remarks are in order: First, note that a process can, of course, be parent and child at the same time, which may occur due to multiply-parallel composition. Because we rename reset to reset_children in the parent so that it could be reset by a parent with an edge labelled reset as well, the modifications are orthogonal and this is unproblematic. This renaming (and the fact that timeout is caught), however, makes it necessary to use PRISM’s process-algebraic expressions to specify the exact composition of the modules, precisely as displayed in Figure 3. Second, in our examples, process WaitDone contained its enabling flag \( e_2 \). Since a module can only assign to its local variables, \( e_2 \) should actually be a local variable in CommProtocol.

\[ D. \text{Summary} \]

We have discussed how to translate a large subset of the Modest language into guarded commands for use with PRISM. The requirements for the supplied Modest code are that all random assignments only sample from finite distributions, urgency constraints do not occur on synchronising edges, \texttt{par} constructs are guarded, and process calls are tail-recursive. In particular, the latter restricts the possible point. The exception \texttt{timeout} is represented by a normal action that the parent synchronises on in order to handle it.

\textbf{Parent processes:} The PRISM code for \texttt{CommProtocol} is given in Figure 12. After performing the initial work, the two child processes are enabled in line 6. Note that these assignments are placed on the edge that leads to the parallel composition. To make sure that such an edge always exists, we actually require a parallel composition that is to be preserved in terms of separate modules to be guarded, so lines 3 and 4 in Figure 9 should actually read \texttt{init par ...}

In location 1, \texttt{CommProtocol} then waits for the children to terminate (line 7) or to throw exception \texttt{timeout} (line 9), which was converted into an action, so we can synchronise on it. As soon as one of these interactions occurs, the children are reset and the parent proceeds. In case the exception was caught, the process starts over again, which is why the children had to be reset at all.

Two technical remarks are in order: First, note that a process can, of course, be parent and child at the same time, and this is unproblematic. This renaming (and the fact that timeout is caught), however, makes it necessary to use PRISM’s process-algebraic expressions to specify the exact composition of the modules, precisely as displayed in Figure 3. Second, in our examples, process \texttt{WaitDone} contained its enabling flag \( e_2 \). Since a module can only assign to its local variables, \( e_2 \) should actually be a local variable in \texttt{CommProtocol}.

\[ D. \text{Summary} \]

We have discussed how to translate a large subset of the Modest language into guarded commands for use with PRISM. The requirements for the supplied Modest code are that all random assignments only sample from finite distributions, urgency constraints do not occur on synchronising edges, \texttt{par} constructs are guarded, and process calls are tail-recursive. In particular, the latter restricts the possible point. The exception \texttt{timeout} is represented by a normal action that the parent synchronises on in order to handle it.

\textbf{Parent processes:} The PRISM code for \texttt{CommProtocol} is given in Figure 12. After performing the initial work, the two child processes are enabled in line 6. Note that these assignments are placed on the edge that leads to the parallel composition. To make sure that such an edge always exists, we actually require a parallel composition that is to be preserved in terms of separate modules to be guarded, so lines 3 and 4 in Figure 9 should actually read \texttt{init par ...}

In location 1, \texttt{CommProtocol} then waits for the children to terminate (line 7) or to throw exception \texttt{timeout} (line 9), which was converted into an action, so we can synchronise on it. As soon as one of these interactions occurs, the children are reset and the parent proceeds. In case the exception was caught, the process starts over again, which is why the children had to be reset at all.

Two technical remarks are in order: First, note that a process can, of course, be parent and child at the same time, which may occur due to multiply-nested parallel composition. Because we rename reset to \texttt{reset_children} in the parent so that it could be reset by a parent with an edge labelled \texttt{reset} as well, the modifications are orthogonal and this is unproblematic. This renaming (and the fact that \texttt{timeout} is caught), however, makes it necessary to use PRISM’s process-algebraic expressions to specify the exact composition of the modules, precisely as displayed in Figure 3. Second, in our examples, process \texttt{WaitDone} contained its enabling flag \( e_2 \). Since a module can only assign to its local variables, \( e_2 \) should actually be a local variable in \texttt{CommProtocol}.

\[ D. \text{Summary} \]

We have discussed how to translate a large subset of the Modest language into guarded commands for use with PRISM. The requirements for the supplied Modest code are that all random assignments only sample from finite distributions, urgency constraints do not occur on synchronising edges, \texttt{par} constructs are guarded, and process calls are tail-recursive. In particular, the latter restricts the possible point. The exception \texttt{timeout} is represented by a normal action that the parent synchronises on in order to handle it.

\textbf{Parent processes:} The PRISM code for \texttt{CommProtocol} is given in Figure 12. After performing the initial work, the two child processes are enabled in line 6. Note that these assignments are placed on the edge that leads to the parallel composition. To make sure that such an edge always exists, we actually require a parallel composition that is to be preserved in terms of separate modules to be guarded, so lines 3 and 4 in Figure 9 should actually read \texttt{init par ...}

In location 1, \texttt{CommProtocol} then waits for the children to terminate (line 7) or to throw exception \texttt{timeout} (line 9), which was converted into an action, so we can synchronise on it. As soon as one of these interactions occurs, the children are reset and the parent proceeds. In case the exception was caught, the process starts over again, which is why the children had to be reset at all.

Two technical remarks are in order: First, note that a process can, of course, be parent and child at the same time, which may occur due to multiply-nested parallel composition. Because we rename \texttt{reset} to \texttt{reset_children} in the parent so that it could be reset by a parent with an edge labelled \texttt{reset} as well, the modifications are orthogonal and this is unproblematic. This renaming (and the fact that \texttt{timeout} is caught), however, makes it necessary to use PRISM’s process-algebraic expressions to specify the exact composition of the modules, precisely as displayed in Figure 3. Second, in our examples, process \texttt{WaitDone} contained its enabling flag \( e_2 \). Since a module can only assign to its local variables, \( e_2 \) should actually be a local variable in \texttt{CommProtocol}.
number of processes created due to dynamic parallelism, which enables us to represent the code using PRISM’s static parallel composition of modules without any limitations on the nesting of parallel composition, loops, or exception handling constructs.

IV. IMPLEMENTATION

We now present some interesting aspects of the implementation in mcpta concerning its architecture and the optimisations it uses to generate readable, succinct and efficient PRISM models.

Since portability was an important concern, we used the C# programming language, which allows the mcpta binary to run on any platform where Mono\(^3\) 2.0 (or newer) is available, which includes Windows, Linux, and Mac OS. Our language parser for Modest is generated using the yacc/lex-based GPPG/GPLEX\(^4\) parser framework, which is written in and integrates well with C# while also providing the foundations for a possible future integration of Modest into Microsoft Visual Studio.

A. Architecture

Figure 13 illustrates the context mcpta works in and its internal architecture. Let us begin by explaining the interaction between Modest, mcpta and PRISM in more detail:

**Context:** Basically, mcpta translates Modest code corresponding to PTA into guarded commands for use with PRISM, applying the digital clocks semantics for time. To make the analysis of Modest models fully automated, though, the process of property specification has to be addressed as well. We therefore extend Modest with a set of constructs that allow the user to specify probabilistic timed properties in a Modest model itself, such as

\[
\text{property } P(\diamond (\text{did}(a) \lor \text{throw}(e)) \land i == 2) \leq 0.8;
\]

to specify the property “is the maximum probability of reaching a state where \(i\) equals 2 some time after action \(a\) has been executed or exception \(e\) has been thrown less than or equal to 0.8”. mcpta picks these up and, aside from the guarded commands representing the behaviour of the modelled systems, generates a property file in PRISM’s property specification language. If necessary, the guarded commands are suitably instrumented to support the properties, e.g. by adding reward structures for expected-time reachability properties.

**Internal architecture:** mcpta is designed to provide a generic infrastructure of reusable, loosely coupled components. It therefore has a layered architecture, depicted inside the mcpta box in Figure 13.

The most basic component that all others depend on, Syntax, provides a general representation of Modest code as an abstract syntax tree that is not limited to the subset corresponding to PTA as well as a collection of useful operations on Modest processes such as an expression optimiser. To make the development of additional operations easier, Syntax also provides different base classes for the implementation of the Visitor design pattern [17] using Reflective Visitors [18].

The Parser is responsible for converting textual Modest input into abstract syntax trees, while the Automaton and GuardedCommands components provide two additional representation layers: The Automaton layer applies Modest’s operational semantics on an abstract syntax tree, transforming it into a stochastic timed automaton, for which it supplies a representation. Similarly, GuardedCommands works on top of this, generating an abstract representation of a set of guarded command modules corresponding to a given automaton. It is here that the translations presented in Section III are implemented. Note that there is a strict layering of dependencies, which allows e.g. the implementation of a tool working on stochastic timed automata using the Automaton and Syntax components, but without depending on the Parser.

Finally, mcpta links all the components together to create a simple command-line façade that actually invokes all the components to translate Modest into guarded commands.

B. Optimisations

mcpta also implements several optimisations of the input models in order to arrive at readable and efficient PRISM code. First of all, expressions are optimised, including the removal of redundant subexpressions as well as constant folding and propagation. Intermediate, but unreachable locations, such as error locations introduced by throw that are of no relevance if the exception is caught, are removed. These basic operations already result in PRISM code that is rather clean and looks close to being written by hand.

Two more optimisations actually improve the model-checking time and memory usage of the generated models:

**Variable ordering:** Due to the use of symbolic representations based on binary decision diagrams, the model-checking performance of PRISM is highly dependent on a "good" variable ordering. In experiments performed with different variable orderings in our case studies, we saw reductions in terms of model-checking time between 40 and 50 % between the worst and the best orderings we found. Models where clock variables followed by the state variable appeared first
in the modules appeared to consistently perform very well, so mcpta sorts all variables according to this order.

**Redundant clocks:** The one optimisation that showed the most dramatic improvements, however, was the removal of redundant clocks: A clock variable \( c \) is redundant in a location \( l \) of a PTA iff, on all paths starting in \( l \), it is reset before it appears in a guard or in an invariant. If this is the case, we do not need to keep track of the actual value of \( c \) in \( l \). In the digital clocks semantics, we must keep the transitions labelled \( \text{tick} \) in \( l \) to allow time to progress, but we do not need to increment \( c \) on them.

mcpta implements the removal of redundant clocks, albeit currently in a weaker form (it only considers a clock redundant in a location if it is immediately reset on all outgoing edges and not used in their guards or the invariant). However, this still results in reductions of the underlying state-space by up to 90% in the case studies.

V. EXPERIMENTAL RESULTS

We consider three case studies to validate the correctness and usefulness of mcpta. All three of them concern communication protocols that combine probabilistic and time-dependent behaviour. We focus on different modelling techniques with Modest, investigate the performance impact of the redundant clocks optimisation, and compare the PRISM models generated by mcpta to hand-written, manually optimised ones. All experiments have been performed on an Intel Core 2 Duo T9300 (2.5 GHz) system running PRISM 3.2 on Windows Vista x64, using the PRISM engine that performed best in each case. We only present an overview of the actual model-checking and performance results; detailed lists and complete source files can be found on the mcpta website\(^5\).

A. Bounded Retransmission Protocol

The Bounded Retransmission Protocol (BRP) is an industrial data link protocol proposed by Philips for transmitting files, partitioned into chunks, over unreliable channels [19]. It is a variant of the Alternating Bit Protocol with a bounded number of retransmissions per chunk. The BRP has been studied for over fifteen years, with modelling and verification being carried out with a multitude of different formalisms and methods. In particular, timed and functional properties have been checked using a TA model [20], while a PA model has been studied both using the RAPTURE verification tool [21] and PRISM\(^6\). A probabilistic timed model has not been studied before.

For the protocol’s correctness, timing is essential: The transmission of data may incur a delay \( TD \), and senders and receivers rely on timeouts to detect message loss. The TA model can faithfully represent this, and was consequently used to, for example, find safe bounds for timeouts. On the other hand, the decision to lose a message had to be modelled using nondeterministic choice. The TA modelling therefore includes the unrealistic scenario of communication channels that lose every message. A PA modelling offers the more appropriate probabilistic choice here, allowing the channel to lose messages with some probability \( p \), and could thus be used to obtain probabilities for events such as the successful transmission of a complete file. However, time and timeouts are not available in PA, so modelling tricks have to be used to actually detect message loss.

Using Modest, we are able to specify a probabilistic timed model of the BRP. This makes it possible to check the properties mentioned before on a single model. In addition, it enables us to check, apparently for the first time, probabilistic timed properties such as “what is the worst case expected time until a file transfer completes” and “what is the worst-case probability that a file transfer completes within \( n \) seconds”.

**Modelling:** We already introduced models of different variants of a very simple communication protocol in the examples of the previous sections. These contain all the basic ingredients necessary to model the BRP, such as an unreliable communication layer with delays, timeouts, and bounded numbers of retransmissions. The two Modest models of the full BRP we built are thus very similar:

The first is composed of otherwise flat processes for the partners involved such as sender and receiver. This model, while functionally correct, is thus similar in structure to the example in Figure 4, but more complicated, and so turns out to be rather difficult to understand, debug and extend.

For this reason, we have built a second, more modular model in the spirit of the examples introduced in Section III—the parent process of Figure 10 almost literally appears there—which is easy to understand and extend, though slightly less efficient due to additional silent edges and scoping issues. These currently prevent the redundant clocks optimisation from being as efficient as it could be.

**Checking properties:** We check four distinct kinds of properties for the BRP: First, timed invariant properties that have been used with the timed model, such as “there are no premature timeouts”, and standard probabilistic reachability properties from the PRISM case study, including e.g. the maximum probability of the sender reporting an unsuccessful transmission after more than 8 chunks have been sent. For the latter set of properties, we reproduce exactly the same probabilities. Second, having a probabilistic timed model allows us to check properties of the BRP from two new classes, namely probabilistic time-bounded and expected-time reachability. We can therefore derive e.g. the maximum probability of the sender reporting a successful transmission within \( n \) time units, or the maximum expected time until the transfer of the first file is finished.

To check time-bounded properties, an additional module that keeps track of the time that has passed globally is needed; mcpta generates this automatically when it encoun-

\(^5\)http://www.modestchecker.net/
\(^6\)http://www.prismmodelchecker.org/casestudies/brp.php
maximum transmission delay \( TD \) has been investigated in a PRISM case study to the wireless medium and reduce the number of collisions. The Collision Avoidance (CSMA/CA) protocol to manage access to the channel's maximum transmission delay \( TD \). The model \( (64, 5, 1) \) is the largest model considered in the case study on the PRISM website. The performance results are summarised in Tables II and III for the flat and modular models, respectively, where the times given are the sum of model construction time and model-checking time for all properties of the respective kind. The overhead incurred by the modularisation currently results in 18 to 39 % larger state-spaces. The respective kind. The overhead incurred by the modularisation currently results in 18 to 39 % larger state-spaces. The results are summarised in Table IV. We observe that the redundant clocks optimisation has a dramatic effect on the state-space size, reducing the number of states by over 90 % in comparison to the unoptimised Modest model. We check the probabilistic, time-bounded and expected-time reachability properties from the PRISM code generated by mcpta with and without redundant clocks optimisation—to that of a manually generated and optimised PRISM model. The results are summarised in Table IV. We observe that the redundant clocks optimisation has a dramatic effect on the state-space size, reducing the number of states by over 90 % for the standard and by more than 50 % for the deadline models, compared to the unoptimised Modest model. The PRISM case study. Aside from being able to reproduce the exact results again, we now have the opportunity to compare the performance of the PRISM code generated by mcpta—with and without redundant clocks optimisation—to that of a manually generated and optimised PRISM model. The results are summarised in Table IV. We observe that the redundant clocks optimisation has a dramatic effect on the state-space size, reducing the number of states by over 90 % for the standard and by more than 50 % for the deadline models, compared to the unoptimised Modest model. But even compared to the PRISM model, where the very same optimisation was apparently built-in by hand at some points, we obtain a reduction of 74 % and 63 %, respectively. Signal propagation times or the time the station needs to switch from listening to sending mode. Modelling: In the original PRISM case study, the channel and the station models are given as both PTA and PRISM code. The PTA for the stations is relatively complex, so instead of using a probably unreadable Modest model relying on nested loops and careful use of exceptions, we make use of a standard transformation from PTA to Modest. Every location is represented as a Modest process consisting of a nondeterministic choice \( alt \) over the outgoing edges. The location invariant is preserved in an invariant construct on that choice, while every outgoing edge \((l,g,a,\mu)\) can be encoded as \(\text{when}(g)\ a\ \text{followed by a probabilistic choice} \ palt \over the destination locations—i.e. tail-recursive calls to the respective processes—and clock resets according to \( \mu \). In the case of the station model, the probabilistic branching in the PTA, which is induced by the exponential backoff process, is hidden in an assignment corresponding to

\[ \text{backoff} := \text{DiscreteUniform}(0, 2^{bc+4} - 1) \]

in Modest. In the PRISM language, this has to be represented by a lengthy guarded command for every possible value of \( 2^{bc+4} - 1 \), which amounts to 150 lines of code the equivalent of which mcpta generates automatically.

Checking properties: We check the probabilistic, time-bounded and expected-time reachability properties from the PRISM case study. Aside from being able to reproduce the exact results again, we now have the opportunity to compare the performance of the PRISM code generated by mcpta—with and without redundant clocks optimisation—to that of a manually generated and optimised PRISM model. The results are summarised in Table IV. We observe that the redundant clocks optimisation has a dramatic effect on the state-space size, reducing the number of states by over 90 % for the standard and by more than 50 % for the deadline models, compared to the unoptimised Modest model. But even compared to the PRISM model, where the very same optimisation was apparently built-in by hand at some points, we obtain a reduction of 74 % and 63 %, respectively.

When using PRISM, however, a reduction of the state space does not necessarily lead to reduced model-checking times; it may well be that a larger model has a better structure that allows faster model-checking. For this case study, though, the size savings do translate into time savings: Compared to the unoptimised Modest models, we save 86 %
and 65%; compared to the PRISM model, the time savings of 25% and 39% are not as drastic, but still significant.

C. IEEE 802.3 CSMA/CD Protocol

The IEEE 802.3 standards define portions of the Ethernet wired LAN technology, which uses Carrier Sense Multiple Access with Collision Detection (CSMA/CD) to allow transmissions to be aborted early when a collision is detected. As before, probabilities are introduced by an exponential backoff process, there are numerous timing parameters, and we model a network of two stations with an initial collision.

Modelling: Our Modest model of the CSMA/CD protocol is based on the probabilistic timed automata given in the corresponding PRISM case study\(^8\). However, there is no PRISM code available this time that could be used for comparison purposes. Again, we benefit from mcpta’s support for finite probability distribution sampling when modelling the exponential backoff process.

Checking properties: Since the PRISM case study was designed for use with the prototype implementation of the backwards reachability approach (cf. Section I), it just includes probabilistic and time-bounded reachability properties. We check these as well as the expected minimum and maximum times until both stations correctly deliver their packets; the performance results are summarised in Table V. The significant state-space explosion in the deadline models is worth noting; this is due to higher maximum constants for some clocks and the relatively long deadline of 1800 µs.

We see that the redundant clocks optimisation has a noticeable effect in this case as well, reducing the state-space by 39% for the standard and 13% for the deadline models. The reductions in model-checking time of 35% and 38% are about as high or higher than the state-space reduction.

VI. Conclusion

This paper has presented a fully automated checking approach for probabilistic timed automata. Models are written in Modest, and the tool mcpta translates this input into the input format of the PRISM model checker, which then can be used to study probabilistic, time-bounded probabilistic and expected reachability properties. We have explained how mcpta is able to handle dynamic parallelism and exception handling constructs despite the static module structure digestable by PRISM. Three case studies have exemplified the principal viability of this approach, and also highlighted several strengths and weaknesses. We consider the results to be very encouraging, and are eagerly working on improving and extending the mcpta tool.

References

\(^8\)http://www.prismmodelchecker.org/casestudies/csma.php