Explicit Algorithms for Probabilistic Model Checking

Dottorando
Igor Melatti

Tutore
Prof. Benedetto Intrigila
Two explicitalgorithms for probabilistic model checking are proposed.

- Formal description
- Proof of correctness
- Implementation (FHP-Mur

- Experimental results
- Comparison with state-of-the-art algorithms (PRISM)
- Verification of a “real-world” system

Formal analysis of the proposed Markov Chain description language
• Two explicit algorithms for probabilistic model checking are proposed
Two explicit algorithms for probabilistic model checking are proposed

- Formal description
- Proof of correctness
- Implementation (FHP-Murφ)
Two explicit algorithms for probabilistic model checking are proposed

- Formal description
- Proof of correctness
- Implementation (FHP-Murϕ)
- Experimental results
  - Comparison with state-of-the-art algorithms (PRISM)
  - Verification of a “real-world” system
Two explicit algorithms for probabilistic model checking are proposed

- Formal description
- Proof of correctness
- Implementation (FHP-Murφ)
- Experimental results
  - Comparison with state-of-the-art algorithms (PRISM)
  - Verification of a “real-world” system

Formal analysis of the proposed Markov Chain description language
Probabilistic Model Checking

Given the description of a Markov Chain, it verifies a PCTL property:

- **PCTL**: Probabilistic CTL

\[
\begin{aligned}
\exists \mathbf{t} U (\mathbf{t} : \mathbf{t} U 1) &> 0 \\
BPCTL: Bounded PCTL &\subset PCTL \\
\exists \mathbf{t} U k_1 (\mathbf{t} : \mathbf{t} U k_2) &> 0 \\
\end{aligned}
\]
• Markov Chain analysis
• Markov Chain analysis

• Given the description of a Markov Chain, it verifies a PCTL property
• Markov Chain analysis

• Given the description of a Markov Chain, it verifies a PCTL property

• PCTL: Probabilistic CTL
  
  \[- [tt \ U (\neg \phi \land \neg [tt \ U \phi]_{\geq 1})]_{\leq 0} \]
Markov Chain analysis

Given the description of a Markov Chain, it verifies a PCTL property

PCTL: Probabilistic CTL

\[ [tt \ U (\neg \phi \land \neg [tt \ U \phi]_{\geq 1})] \leq 0 \]

BPCTL: Bounded PCTL

- Proper subset of PCTL
- All Until (U) must be bounded

\[ [tt \ U^{\leq k_1} (\neg \phi \land \neg [tt \ U^{\leq k_2} \phi]_{\geq 1})] \leq 0 \]

\[ [tt \ U^{\leq k_1} (\phi_{\text{und}} \land \neg [tt \ U^{\leq k_2} \neg \phi_{\text{err}}]_{\geq 1})] \leq 0 \]
• FiniteHorizonProbabilistic-Murφ
• Finite Horizon Probabilistic-Murφ

• Explicit probabilistic model checker
- Finite Horizon Probabilistic-Murφ
- Explicit probabilistic model checker
  - explicit verification often outperforms symbolic verification in non-probabilistic model checking
• Finite Horizon Probabilistic-Murφ

• Explicit probabilistic model checker
  – explicit verification often outperforms symbolic verification in non-probabilistic model checking
  – we will show that this holds also for probabilistic model checking
• Finite\textsuperscript{Horizon}Probabilistic-\textit{Mur}φ

• Explicit probabilistic model checker
  
  – explicit verification often \textit{outperforms} symbolic verification in non-probabilistic model checking
  
  – we will show that this holds also for probabilistic model checking

• Murφ modified in the input language and in the verification algorithm
• Finite Horizon Probabilistic-Murφ

• Explicit probabilistic model checker
  – explicit verification often outperforms symbolic verification in non-probabilistic model checking
  – we will show that this holds also for probabilistic model checking

• Murφ modified in the input language and in the verification algorithm

• Two explicit algorithms developed
  – BF visit: only for finite horizon safety properties
    * Able to compute error probabilities
  – DF visit: all BPCTL formulas
• We want to verify if $s_0 \models [tt \ U^{\leq 2} \ \phi]_{\geq 0.5}$

• $\phi$ holds in $s_1, s_4, s_7$

• The searched probability is: 0
We want to verify if $s_0 \models [tt \ U^{\leq 2} \ \phi]_{\geq 0.5}$

- $\phi$ holds in $s_1, s_4, s_7$
- The searched probability is: $\frac{1}{3} \times 1$
- We want to verify if $s_0 \models [tt \ U \leq 2 \ \phi] \geq 0.5$.
- $\phi$ holds in $s_1, s_4, s_7$.
- The searched probability is: $\frac{1}{3} + \frac{1}{3} \times \ldots$
We want to verify if $s_0 \models [tt \ U^{\leq 2} \ \phi]_{\geq 0.5}$.

$\phi$ holds in $s_1, s_4, s_7$.

The searched probability is: $\frac{1}{3} + \frac{1}{3} \times (\frac{1}{2} \times 0 + \ldots)$.
- We want to verify if $s_0 \models [tt \ U^{\leq 2} \ \phi]_{\geq 0.5}$
- $\phi$ holds in $s_1$, $s_4$, $s_7$
- The searched probability is: $\frac{1}{3} + \frac{1}{3} \times \left( \frac{1}{2} \times 0 + \frac{1}{2} \times 1 \right)$
We want to verify if $s_0 \models [tt \cup^{\leq 2} \phi]_{\geq 0.5}$.

- $\phi$ holds in $s_1, s_4, s_7$

- The searched probability is: $\frac{1}{3} + \frac{1}{3} \times \frac{1}{2} + \frac{1}{3} \times \ldots$
We want to verify if $s_0 \models [tt \ U^{\leq 2} \ \phi]_{\geq 0.5}$.

- $\phi$ holds in $s_1$, $s_4$, $s_7$
- The searched probability is: $\frac{1}{3} + \frac{1}{3} \times \frac{1}{2} + \frac{1}{3} \times 0$
We want to verify if $s_0 \models [tt \ U^{\leq 2} \phi]_{\geq 0.5}$.

- $\phi$ holds in $s_1$, $s_4$, $s_7$.

- The searched probability is: $\frac{1}{3} + \frac{1}{3} \times \frac{1}{2} + \frac{1}{3} \times 0$.

- Finally, we have $\frac{1}{3} + \frac{1}{3} \times \frac{1}{2} = \frac{1}{2} \geq 0.5$, so the property is verified.
We want to verify if \( s_0 \models F \), being 
\[
F \equiv [\Phi \ U^{\leq k} \Psi]_{\leq 0.5}
\]

The cache stores 4-tuples \( \{s, F, h, p\} \)

- \( p \) is the probability of \( \Phi \ U^{\leq h} \Psi \)
When the DF visit of $s_3$ is completed, $\{s_3, F, k - 3, p_3\}$ is inserted in the cache

- $p_3$ is the probability value computed by the DF on $s_3$
- $k$ is decremented of 3 because $s_3$ is reached in 3 steps from $s_0$
When the DF visit of $s_3$ is completed, $\{s_3, F, k - 3, \rho_3\}$ is inserted in the cache

- $\rho_3$ is the probability value computed by the DF on $s_3$
- $k$ is decremented of 3 because $s_3$ is reached in 3 steps from $s_0$

Analogously, $\{s_2, F, k - 2, \rho_2\}$ is inserted in the cache
In this way, the DF visit of $s_4$ can directly compute $p_4 = p_3 \times 1$.

- $p_3$ is not computed, but it is found on the cache.

Then, $\{s_4, F, k - 2, p_4\}$ is inserted in the cache.
• Analogously, when the DF visit of $s_5$ starts, the nested DF visit of $s_4$ is skipped
  – $p_4$ is not computed, but it is found on the cache

• The result of the DF visit of $s_6$ will be multiplied by $\frac{1}{2}$ and then added to $\frac{1}{2} \times p_4$
Experimental results were carries out on two kind of systems:
Experimental results were carries out on two kind of systems:

**Probabilistic dining philosophers** Pnueli-Zuck (PZ) and Lehmann-Rabin (LR) protocols
Experimental results were carries out on two kind of systems:

**Probabilistic dining philosophers** Pnueli-Zuck (PZ) and Lehmann-Rabin (LR) protocols

- In the version found on the PRISM distribution, PRISM works better
Experimental results were carried out on two kinds of systems:

**Probabilistic dining philosophers** Pnueli-Zuck (PZ) and Lehmann-Rabin (LR) protocols

- In the version found on the PRISM distribution, PRISM works better.
- If they are modified in order to verify *quality-of-service* properties, FHP-Murφ works better.
Experimental Results

Experimental results were carries out on two kind of systems:

**Probabilistic dining philosophers** Pnueli-Zuck (PZ) and Lehmann-Rabin (LR) protocols

- In the version found on the PRISM distribution, PRISM works better
- If they are modified in order to verify *quality-of-service* properties, FHP-Murφ works better

**Hybrid systems** Verification of a turbogas control system, assuming a probability distribution on the user demand
Experimental results were carries out on two kind of systems:

**Probabilistic dining philosophers** Pnueli-Zuck (PZ) and Lehmann-Rabin (LR) protocols

- In the version found on the PRISM distribution, PRISM works better
- If they are modified in order to verify *quality-of-service* properties, FHP-Mur\(\varphi\) works better

**Hybrid systems** Verification of a turbogas control system, assuming a probability distribution on the user demand

- Probabilistic Safety Verification
Experimental results were carried out on two kinds of systems:

**Probabilistic dining philosophers** Pnueli-Zuck (PZ) and Lehmann-Rabin (LR) protocols

- In the version found on the PRISM distribution, PRISM works better.
- If they are modified in order to verify *quality-of-service* properties, FHP-Murφ works better.

**Hybrid systems** Verification of a turbogas control system, assuming a probability distribution on the user demand.

- Probabilistic Safety Verification
- Probabilistic Robustness Verification
Experimental Results: PZ and LR Protocols

<table>
<thead>
<tr>
<th>NPHIL</th>
<th>MAX_WAIT</th>
<th>Result</th>
<th>Mur$\phi$ Mem (MB)</th>
<th>PRISM Mem (MB)</th>
<th>Mur$\phi$ Time (s)</th>
<th>PRISM Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Result</td>
<td>Mur$\phi$ Mem (MB)</td>
<td>PRISM Mem (MB)</td>
<td>Mur$\phi$ Time (s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| **Modified Pnueli-Zuck** | | | | \begin{align} 
\text{Modified Lehmann-Rabin} & \\
3 & 4 & true & 5.0e+2 & 7.014830e+01 & 5.00634000e+03 & 5.359870e+02 \\
4 & 3 & true & 5.0e+2 & N/A & 1.11480680e+05 & N/A \\
& & & & & & |
| | | | & & & & |

**Property** verified:

- When an undesired state $s$ is reached, then the system almost always reaches, from $s$ and in a few steps, a non-error state.

- There is a low probability that we reach in $k_1$ steps an undesired state $s$, and there is not a high probability of reaching, from $s$ and in $k_2 = \frac{k_1}{10}$ steps, a non-error state.

- $[tt U^{\leq k_1} (\phi_{und} \land \neg [tt U^{\leq k_2} \neg \phi_{err}] \geq 1)] \leq 0$.

**NPHIL, MAX_WAIT:** protocol parameters
ICARO: 2MWElectric Co-generative Power Plant, in operation at the ENA Research Center of Casaccia (Italy)

The most important module is the Turbogas Control System (TCS) — it is also the most complex one.

It is a hybrid system: it has both continuous (e.g., power and user demand) and discrete variables (execution modality) — this kind of systems are hard to analyze with OBDD-based model checkers.

Thus, there is no hope to verify TCS with PRISM.
ICARO: 2MW Electric Co-generative Power Plant, in operation at the ENEA Research Center of Casaccia (Italy)
ICARO: 2MW Electric Co-generative Power Plant, in operation at the ENEA Research Center of Casaccia (Italy)

The most important module is the Turbogas Control System (TCS)

- It is also the most complex one
ICARO: 2MW Electric Co-generative Power Plant, in operation at the ENEA Research Center of Casaccia (Italy)

The most important module is the Turbogas Control System (TCS)
  - It is also the most complex one

It is an hybrid system: it has both continuous (e.g., power and user demand) and discrete variables (execution modality)
  - This kind of systems are hard to analyze with OBDD-based model checkers
  - Thus, there is no hope to verify TCS with PRISM
A Turbogas Control System

- Electric Power Generated by the Alternator
- Turbine Rotation Speed
- User Demand (u)
- Fuel Valve Opening
- Compressor Pression
- Exhaust Smokes Temperature

TCS

TCS is an electronic circuit, its detail are known.

The turbogas is modeled by a set of ODEs.

The user demand is modeled as a nondeterministic disturbance. Its variation is bounded by a verification parameter ($\text{MAX}_D\text{U}$).
TCS is an electronic circuit, its detail are known
- TCS is an electronic circuit, its detail are known
- The turbogas is modeled by a set of ODEs
- TCS is an electronic circuit, its detail are known
- The turbogas is modeled by a set of ODEs
- The user demand is modeled as a nondeterministic disturbance
  - Its variation is bounded by a verification parameter $(\text{MAX}_D\text{U})$
To automatically verify TCS, we added finite precision real numbers to Mur'\textsuperscript{'}.

Then, the ODEs are discretized with a sampling step of 10 ms and translated in the Mur'\textsuperscript{'} input language.

The property to be verified is that the main TCS parameters are maintained close to their setpoints values by the controller – this has to hold for every value of the user demand.

As a result, if the user demand varies too much rapidly (i.e. MAX\textsubscript{D\textsubscript{U}} is too high), the controller fails.
To automatically verify TCS, we added finite precision real numbers to Mur.$

$\phi$

• To automatically verify TCS, we added finite precision real numbers to Murφ

• Then, the ODEs are discretized with a sampling step of 10 ms and translated in the Murφ input language
• To automatically verify TCS, we added finite precision real numbers to Murφ.

• Then, the ODEs are discretized with a sampling step of 10 ms and translated in the Murφ input language.

• The property to be verified is that the main TCS parameters are maintained close to their setpoints values by the controller.
  – This has to hold for every value of the user demand.
• To automatically verify TCS, we added finite precision real numbers to $\text{Mur} \varphi$

• Then, the ODEs are discretized with a sampling step of 10 ms and translated in the $\text{Mur} \varphi$ input language

• The property to be verified is that the main TCS parameters are maintained close to their setpoints values by the controller
  – This has to hold for every value of the user demand

• As a result, if the user demand varies too much rapidly (i.e. $\text{MAX}_D \text{U}$ is too high), the controller fails
The TCS and Turbogas behaviors, obviously, remain deterministic. On the other hand, the user demand now has a probabilistic distribution.

Let

\[ p(u,i) = \begin{cases} 
0 & : i = 1 \\
0 & : i = +1 \\
\left( u - M^2 \right) / u & : i = 0 
\end{cases} \]

Then

\[ u(t+1) = \begin{cases} 
\max(u(t), 0) & \text{with prob. } p(u(t), 1) \\
\min(u(t) + M^2, 0) & \text{with prob. } p(u(t), 0) 
\end{cases} \]
• The TCS and Turbogas behaviors, obviously, remain deterministic
● The TCS and Turbogas behaviors, obviously, remain deterministic

● On the other hand, the user demand now have a probabilistic distribution
• The TCS and Turbogas behaviors, obviously, remain deterministic

• On the other hand, the user demand now have a probabilistic distribution
  
  Let

  \[ p(u, i) = \begin{cases} 
  0.4 + \beta \frac{(u - \frac{M}{2})|u - \frac{M}{2}|}{M^2} & \text{if } i = -1 \\
  0.2 & \text{if } i = 0 \\
  0.4 + \beta \frac{(\frac{M}{2} - u)|u - \frac{M}{2}|}{M^2} & \text{if } i = +1 
  \end{cases} \]
- The TCS and Turbogas behaviors, obviously, remain deterministic
- On the other hand, the user demand now have a probabilistic distribution
  - Let
    \[
    p(u, i) = \begin{cases} 
    0.4 + \beta \frac{(u - \frac{M}{2})|u - \frac{M}{2}|}{M^2} & \text{if } i = -1 \\
    0.2 & \text{if } i = 0 \\
    0.4 + \beta \frac{\left(\frac{M}{2} - u\right)|u - \frac{M}{2}|}{M^2} & \text{if } i = +1
    \end{cases}
    \]
  - Then
    \[
    u(t + 1) = \begin{cases} 
    \max(u(t) - \alpha, 0) & \text{with prob. } p(u(t), -1) \\
    u(t) & \text{with prob. } p(u(t), 0) \\
    \min(u(t) + \alpha, M) & \text{with prob. } p(u(t), +1)
    \end{cases}
    \]
• We compute which is the error probability in at most $k$ steps
  – finite horizon safety property
• We compute which is the error probability in at most $k$ steps
  – finite horizon safety property

• $\text{MAX}_D\_U$ has a value that force the non-probabilistic verification to fail
• We compute which is the error probability in at most $k$ steps
  – finite horizon safety property

• $\text{MAX\_D\_U}$ has a value that force the non-probabilistic verification to fail

<table>
<thead>
<tr>
<th>$\text{MAX_D_U}$</th>
<th>Reachable States</th>
<th>Finite Horizon</th>
<th>CPU Time</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>3018970</td>
<td>1600</td>
<td>68562.570</td>
<td>7.373291768e-05</td>
</tr>
<tr>
<td>35</td>
<td>2226036</td>
<td>1400</td>
<td>50263.020</td>
<td>1.076644427e-04</td>
</tr>
<tr>
<td>45</td>
<td>1834684</td>
<td>1300</td>
<td>41403.150</td>
<td>9.957147381e-05</td>
</tr>
<tr>
<td>50</td>
<td>83189</td>
<td>900</td>
<td>2212.360</td>
<td>3.984375e-03</td>
</tr>
</tbody>
</table>
• Verification of a robustness property
• Verification of a robustness property

• Informally: if the system reaches an undesired state, then it is able to return to a more safe state in a few time
• Verification of a robustness property

• Informally: if the system reaches an undesired state, then it is able to return to a more safe state in a few time

• A state is *undesired* if the critical parameters are near to their critical values
  – if the system remains too much time in an undesired state, it will crash
• Verification of a robustness property

• Informally: if the system reaches an undesired state, then it is able to return to a more safe state in a few time

• A state is \textit{undesired} if the critical parameters are near to their critical values
  – if the system remains too much time in an undesired state, it will crash

• More formally: there is a low probability of reaching an undesired state \( s \), such that there is not an high probability of reaching (in a few number of steps) a non-undesired state from \( s \)
• Verification of a robustness property

• Informally: if the system reaches an undesired state, then it is able to return to a more safe state in a few time

• A state is *undesired* if the critical parameters are near to their critical values – if the system remains too much time in an undesired state, it will crash

• More formally: there is a low probability of reaching an undesired state $s$, such that there is not an high probability of reaching (in a few number of steps) a non-undesired state from $s$

• The formula is $[tt \ U^{\leq k_1} (\neg \phi_{und} \land \neg[tt \ U^{\leq k_2} \phi_{und}]_{\geq 1})]_{\leq 0}$
• Verification of a robustness property

• Informally: if the system reaches an undesired state, then it is able to return to a more safe state in a few time

• A state is *undesired* if the critical parameters are near to their critical values
  – if the system remains too much time in an undesired state, it will crash

• More formally: there is a low probability of reaching an undesired state \( s \), such that there is not an high probability of reaching (in a few number of steps) a non-undesired state from \( s \)

• The formula is \([tt \ U^{\leq k_1} (\neg \phi_{und} \land \neg [tt \ U^{\leq k_2} \phi_{und}]_{\geq 1}])]_{\leq 0}\)

• \( k_1 \) is sufficient to reach an undesired state
Verification of a robustness property

Informally: if the system reaches an undesired state, then it is able to return to a more safe state in a few time

A state is *undesired* if the critical parameters are near to their critical values

– if the system remains too much time in an undesired state, it will crash

More formally: there is a low probability of reaching an undesired state $s$, such that there is not an high probability of reaching (in a few number of steps) a non-undesired state from $s$

The formula is $[tt \ U^{\leq k_1} (\neg \phi_{und} \land \neg [tt \ U^{\leq k_2} \phi_{und}]_{\geq 1})]_{\leq 0}$

$k_1$ is sufficient to reach an undesired state

$k_2 = \frac{k_1}{100}$
Results on a machine with 2 processors (both INTEL Pentium III 500Mhz) and 2GB of RAM.

Mur\(\varphi\) options used: \(-m500\) (use 500 MB of RAM)

<table>
<thead>
<tr>
<th>MAX_D_U</th>
<th>Visited States</th>
<th>(k_1)</th>
<th>CPU Time (s)</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>1.159160e+05</td>
<td>800</td>
<td>3.702400e+03</td>
<td>4.104681e-03</td>
</tr>
<tr>
<td>45</td>
<td>4.098000e+04</td>
<td>700</td>
<td>1.313900e+03</td>
<td>1.792883e-02</td>
</tr>
<tr>
<td>50</td>
<td>4.067700e+04</td>
<td>700</td>
<td>1.307850e+03</td>
<td>3.825000e-02</td>
</tr>
</tbody>
</table>
More features for FHP-Mur$^\varphi$ and then comparison with PRISM
More features for FHP-Mur$\varphi$ and then comparison with PRISM

- Continuous Markov Chains (with CSL logic)
  - Approximable to Discrete Time Markov Chain with an exponential distribution
  - The smaller the sampling step
    * the lowest the approximation error
    * the higher the execution time