University of Utah

Explicit Algorithms for Probabilistic Model Checking

Igor Melatti

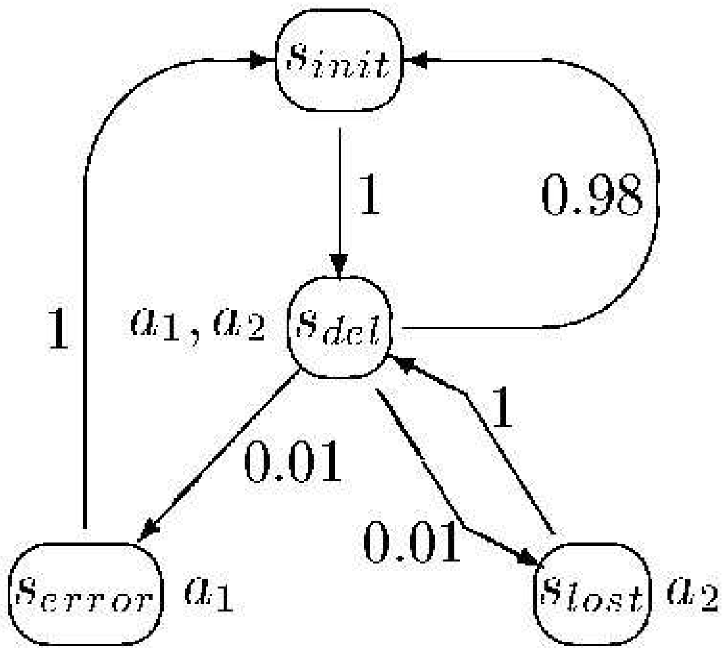
Thesis Contents

- Two explicit algorithms for probabilistic model checking are proposed

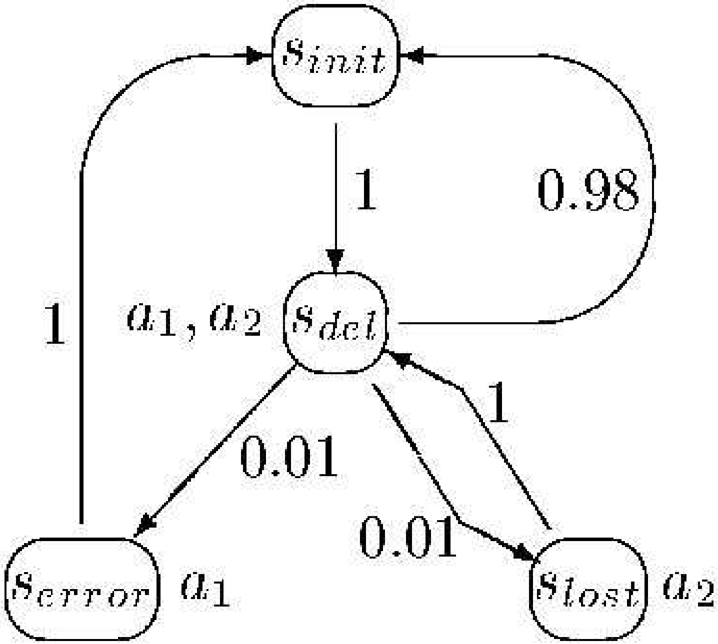
- Two explicit algorithms for probabilistic model checking are proposed
  - Formal description
  - Proof of correctness
  - Implementation (FHP-Murφ)

- 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

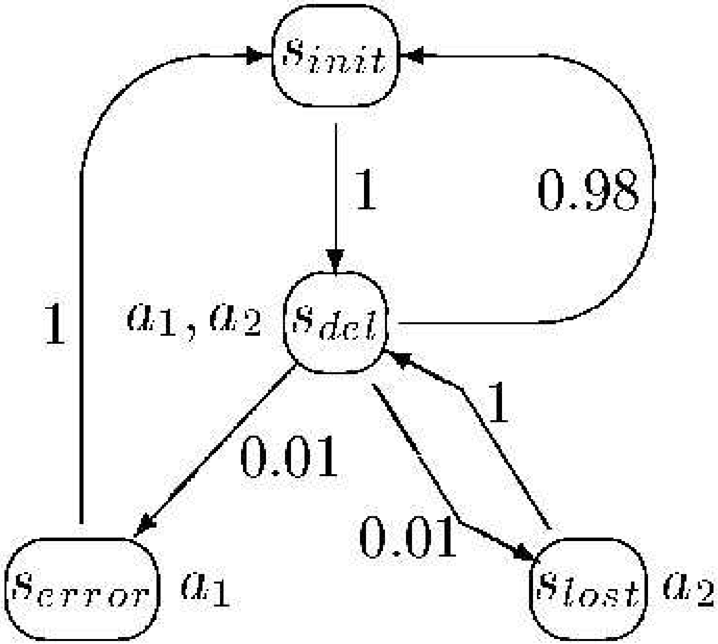


Baier *et al.*, "Symbolic model checking for probabilistic processes", *ICALP'97*, LNCS 1256



Baier *et al.*, "Symbolic model checking for probabilistic processes", *ICALP'97*, LNCS 1256

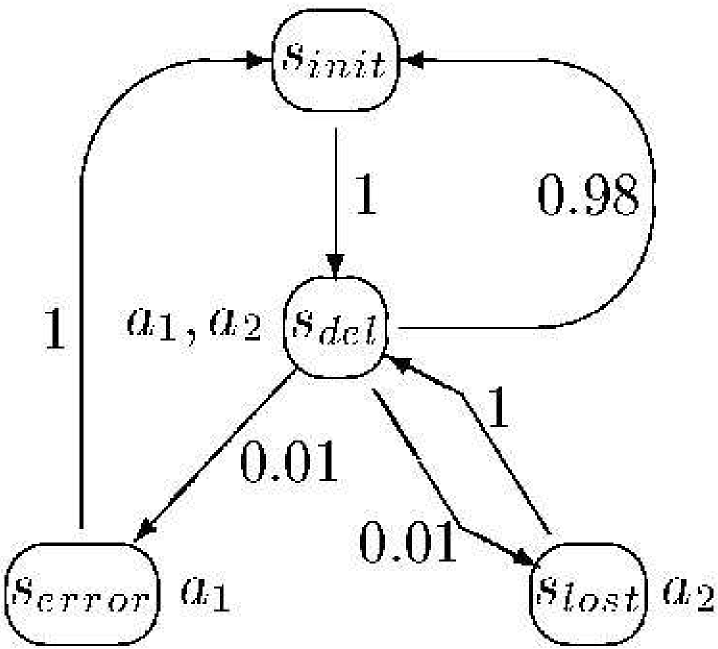
s_{init} the state in which the sender passes the message to the medium



Baier *et al.*, "Symbolic model checking for probabilistic processes", *ICALP'97*, LNCS 1256

s_{init} the state in which the sender passes the message to the medium

s_{del} the state in which the medium tries to deliver the message

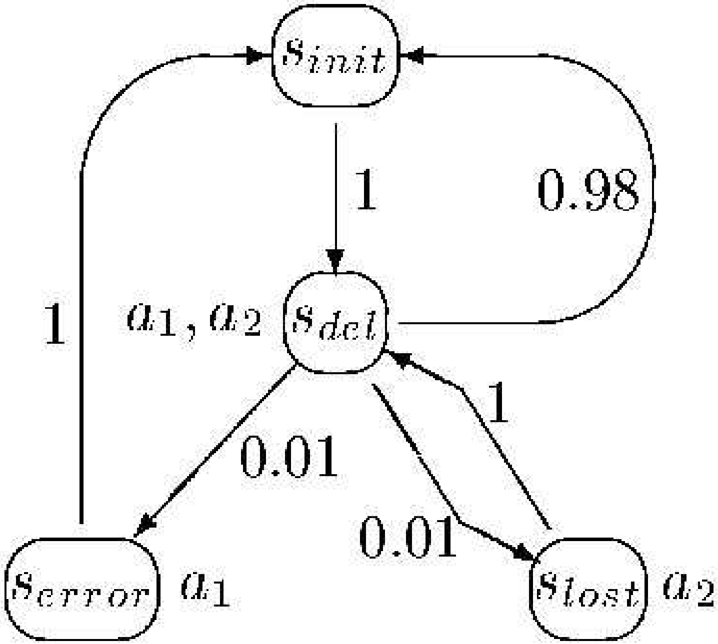


Baier *et al.*, "Symbolic model checking for probabilistic processes", *ICALP'97*, LNCS 1256

s_{init} the state in which the sender passes the message to the medium

s_{del} the state in which the medium tries to deliver the message

s_{lost} the state reached when the message is lost



Baier *et al.*, "Symbolic model checking for probabilistic processes", *ICALP'97*, LNCS 1256

s_{init} the state in which the sender passes the message to the medium

s_{del} the state in which the medium tries to deliver the message

s_{lost} the state reached when the message is lost

s_{error} the state reached when the message is corrupted

- A *Discrete Time Markov Chain* is a triple $\mathcal{M} = (S, \mathbf{P}, q)$ where

- A *Discrete Time Markov Chain* is a triple $\mathcal{M} = (S, \mathbf{P}, q)$ where
  - $S = \{s_0, \ldots, s_n\}$ is a finite set of states and $q \in S$ is the *initial state*

- A *Discrete Time Markov Chain* is a triple $\mathcal{M} = (S, \mathbf{P}, q)$ where
  - $S = \{s_0, \ldots, s_n\}$ is a finite set of states and $q \in S$ is the *initial state*
  - $\mathbf{P} : S \times S \to [0, 1]$ is a *stochastic matrix*
    - for all $s \in S$, $\sum_{t \in S} \mathbf{P}(s, t) = 1$

- A *Discrete Time Markov Chain* is a triple $\mathcal{M} = (S, \mathbf{P}, q)$ where
  - $S = \{s_0, \ldots, s_n\}$ is a finite set of states and $q \in S$ is the *initial state*
  - $\mathbf{P} : S \times S \rightarrow [0, 1]$ is a *stochastic matrix*
    * for all $s \in S$, $\sum_{t \in S} \mathbf{P}(s, t) = 1$
- An *execution sequence* (or *path*) in $\mathcal{M}$ is

$$\pi = r_0 r_1 r_2 \ldots$$

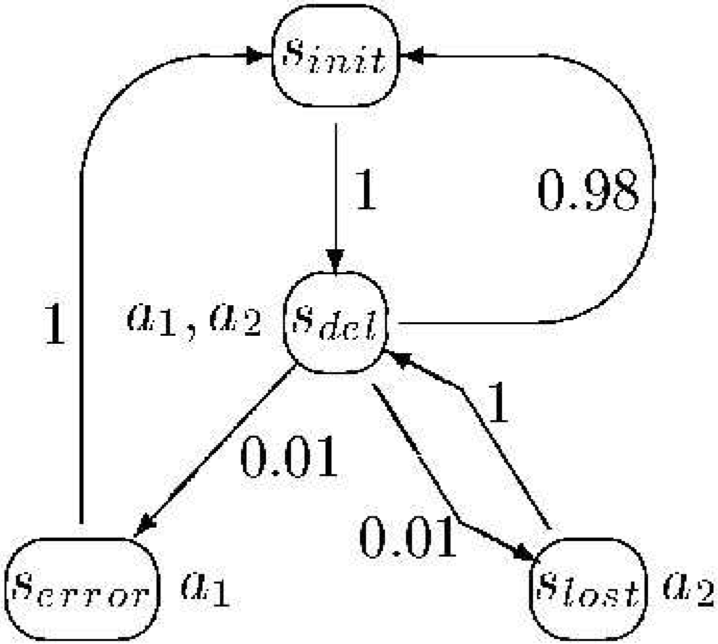
where, for all $i \geq 0$, $r_i \in S$ and $\mathbf{P}(r_i, r_{i+1}) > 0$.

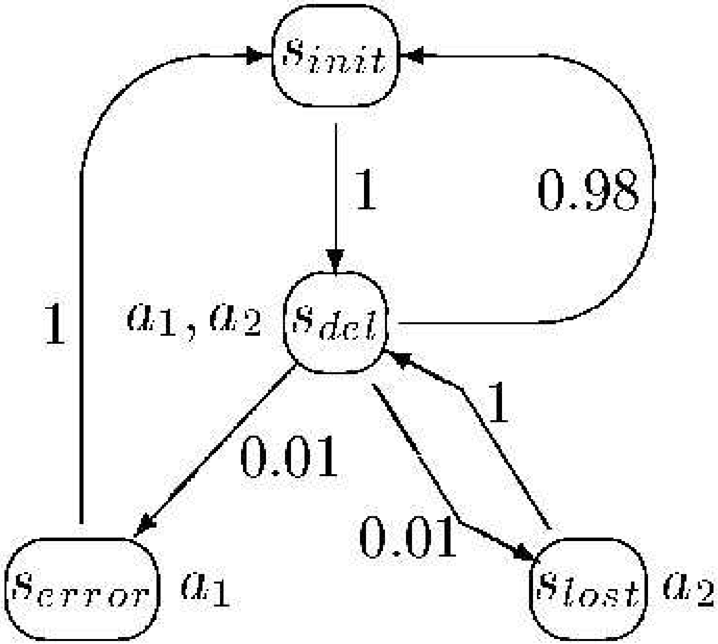
- A *Discrete Time Markov Chain* is a triple $\mathcal{M} = (S, \mathbf{P}, q)$ where
  - $S = \{s_0, \ldots, s_n\}$ is a finite set of states and $q \in S$ is the *initial state*
  - $\mathbf{P} : S \times S \to [0, 1]$ is a *stochastic matrix*
    * for all $s \in S$, $\sum_{t \in S} \mathbf{P}(s, t) = 1$
- An *execution sequence* (or *path*) in $\mathcal{M}$ is

$$\pi = r_0 r_1 r_2 \ldots$$

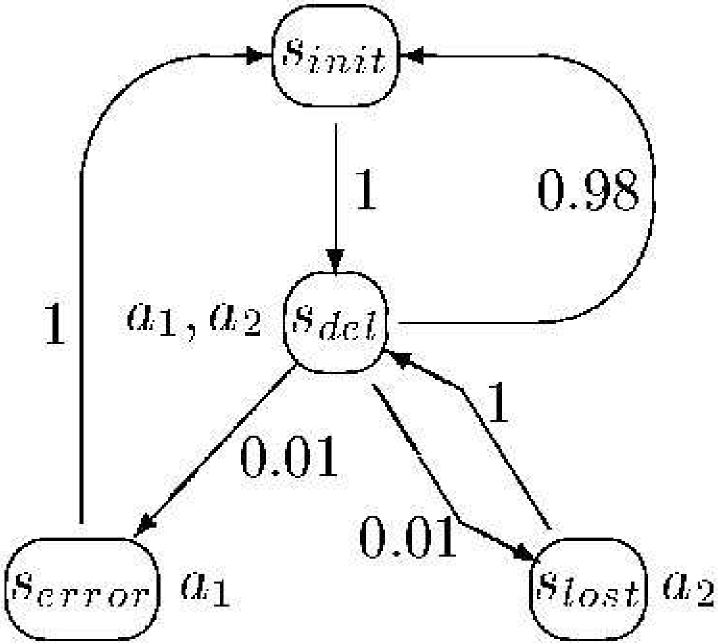
where, for all $i \geq 0$, $r_i \in S$ and $\mathbf{P}(r_i, r_{i+1}) > 0$.

- Probability of a finite path ρ: $\mathbf{P}(\rho) = \prod_{i=0}^{|\rho|-1} \mathbf{P}(\rho(i), \rho(i+1))$.

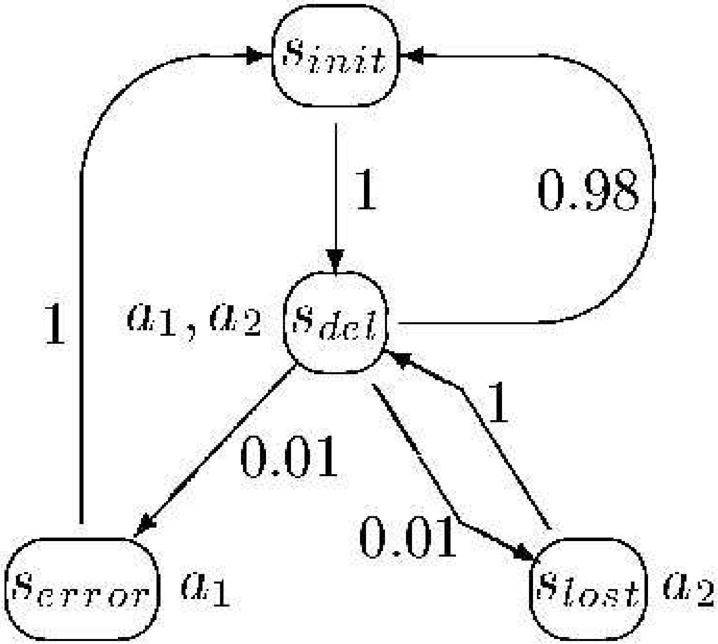

s_{init}
1
0.98
1
a_1, a_2
s_{del}
1
0.01
0.01
s_{error}
a_1
s_{lost}
a_2



- $S = \{s_{init}, s_{del}, s_{lost}, s_{error}\}$

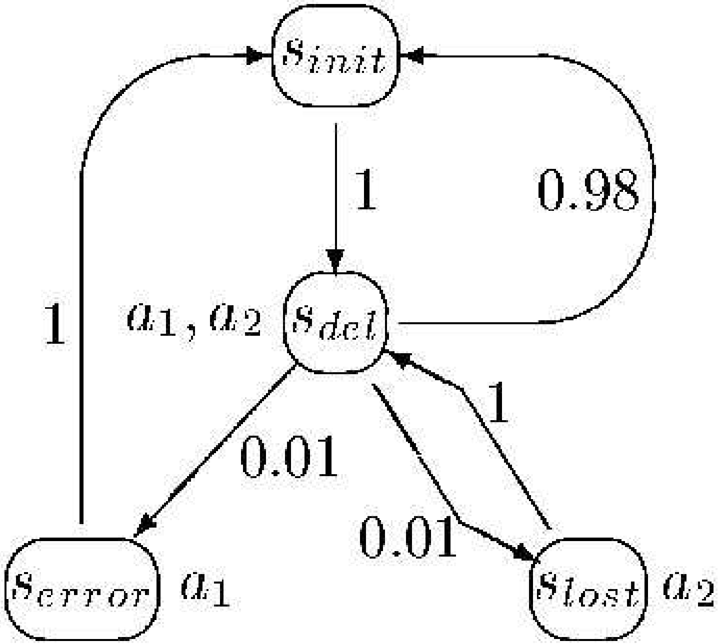


- $S = \{s_{init}, s_{del}, s_{lost}, s_{error}\}$
- $$\mathbf{P} = \begin{array}{cccc} s_{init} & s_{del} & s_{lost} & s_{error} \end{array} \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0.98 & 0 & 0.01 & 0.01 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$



- $S = \{s_{init}, s_{del}, s_{lost}, s_{error}\}$
- $\mathbf{P} = \begin{array}{cccc} s_{init} & s_{del} & s_{lost} & s_{error} \end{array} \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0.98 & 0 & 0.01 & 0.01 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$

- 2 possible paths and their probabilities:
  - $\mathbf{P}(s_{init}s_{del}s_{error}s_{init}) = 1 \cdot \frac{1}{100} \cdot 1 = \frac{1}{100}$
  - $\mathbf{P}(s_{init}(s_{del}s_{lost})^k s_{del}s_{init}) = 1 \cdot (\frac{1}{100} \cdot 1)^k \cdot \frac{98}{100} = \frac{98}{10^{2(k+1)}}$



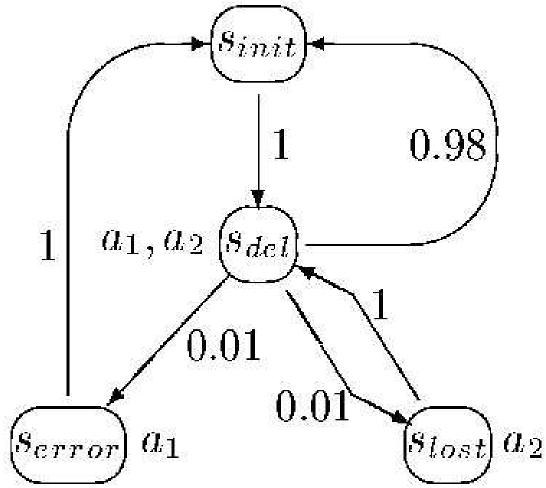
- $S = \{s_{init}, s_{del}, s_{lost}, s_{error}\}$
- $\mathbf{P} = \begin{pmatrix} s_{init} & s_{del} & s_{lost} & s_{error} \\ 0 & 1 & 0 & 0 \\ 0.98 & 0 & 0.01 & 0.01 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$
- 2 possible paths and their probabilities:
  - $\mathbf{P}(s_{init}s_{del}s_{error}s_{init}) = 1 \cdot \frac{1}{100} \cdot 1 = \frac{1}{100}$
  - $\mathbf{P}(s_{init}(s_{del}s_{lost})^k s_{del}s_{init}) = 1 \cdot (\frac{1}{100} \cdot 1)^k \cdot \frac{98}{100} = \frac{98}{10^{2(k+1)}}$
- Impossible path: $s_{init}s_{del}s_{error}s_{del}$

- Markov Chain analysis

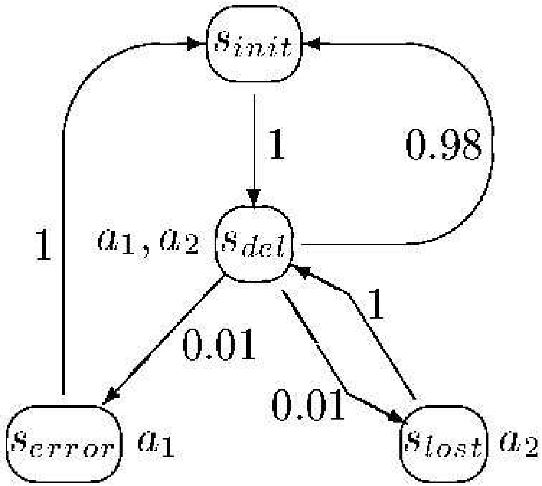
- Markov Chain analysis
- Given the description of a Markov Chain, it verifies a PCTL property

- Markov Chain analysis
- Given the description of a Markov Chain, it verifies a PCTL property
- PCTL: Probabilistic CTL
  - $[tt\ \mathbf{U}\ (\neg\phi \wedge \neg[tt\ \mathbf{U}\phi]_{\geq 1})]_{\leq 0}$

- Markov Chain analysis
- Given the description of a Markov Chain, it verifies a PCTL property
- PCTL: Probabilistic CTL
  - $[tt\ \mathbf{U}\ (\neg\phi \wedge \neg[tt\ \mathbf{U}\phi]_{\geq 1})]_{\leq 0}$
- BPCTL: Bounded PCTL
  - Proper subset of PCTL
  - All Untils (**U**) must be bounded
  - $[tt\ \mathbf{U}^{\leq k_1}\ (\neg\phi \wedge \neg[tt\ \mathbf{U}^{\leq k_2}\phi]_{\geq 1})]_{\leq 0}$
  - $[tt\ \mathbf{U}^{\leq k_1}\ (\phi_{und} \wedge \neg[tt\ \mathbf{U}^{\leq k_2}\neg\phi_{err}]_{\geq 1})]_{\leq 0}$

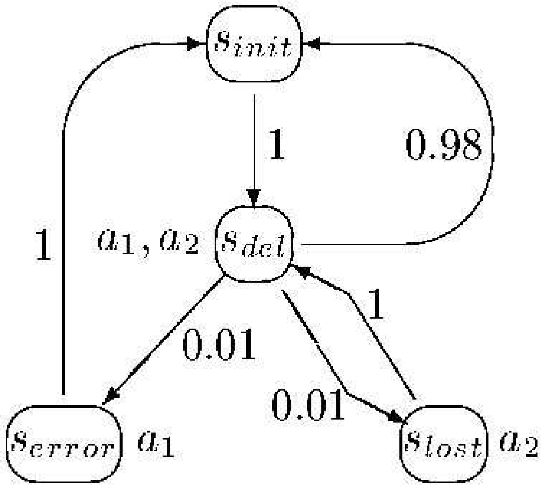


Property: $[\text{Try\_to\_deliver}\ U^{\leq 100}\ \text{Correctly\_delivered}]_{\geq 0.9}$



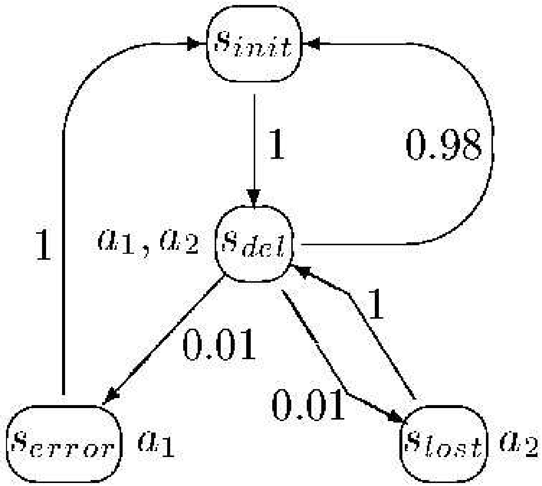
Property: $[\text{Try\_to\_deliver } U^{\leq 100} \text{ Correctly\_delivered}]_{\geq 0.9}$

- Try_to_deliver (T in the following) is true if we are in state s_{del} or s_{lost}
- Correctly_delivered (C in the following) is true if we are in state s_{init}
- Initial state is s_{del}



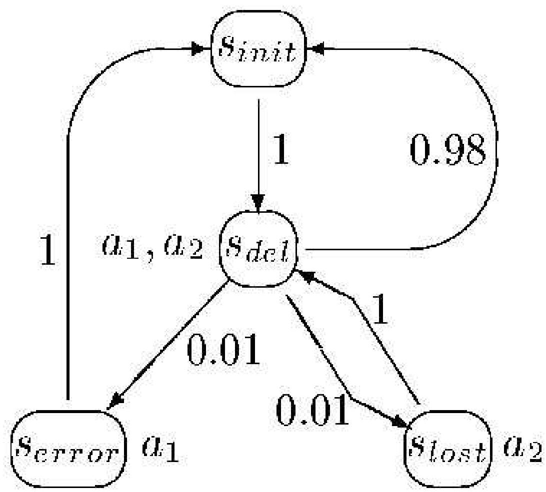
Property: $[\mathrm{Try\_to\_deliver}\ U^{\leq 100}\ \mathrm{Correctly\_delivered}]_{\geq 0.9}$

- Is the probability of the paths of the form T^kC ($0 \leq k \leq 100$) at least 0.9?



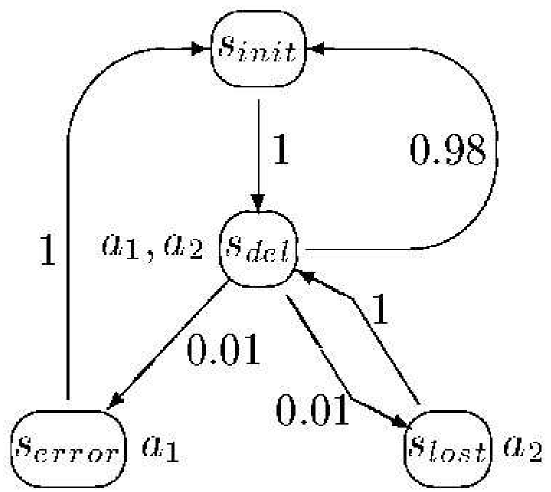
Property: $[\text{Try\_to\_deliver } U^{\leq 100} \text{ Correctly\_delivered}]_{\geq 0.9}$

- Is the probability of the paths of the form T^kC ($0 \leq k \leq 100$) at least 0.9?
  - A path of the form T^kC corresponds to an execution of the system in which, after a bounded trials, the message is finally transmitted



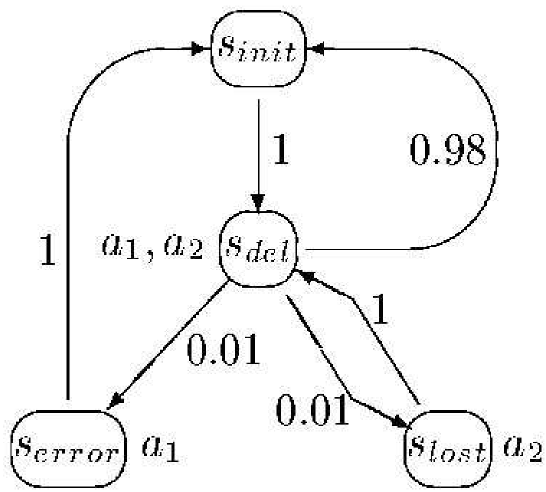
Property: $[\text{Try\_to\_deliver } U^{\leq 100} \text{ Correctly\_delivered}]_{\geq 0.9}$

- Is the probability of the paths of the form $T^k C$ ($0 \leq k \leq 100$) at least 0.9?
  - A path of the form $T^k C$ corresponds to an execution of the system in which, after a bounded trials, the message is finally transmitted
  - Thus, we are requiring the probability of a "correct behavior" to be high enough (i.e. ≥ 0.9)



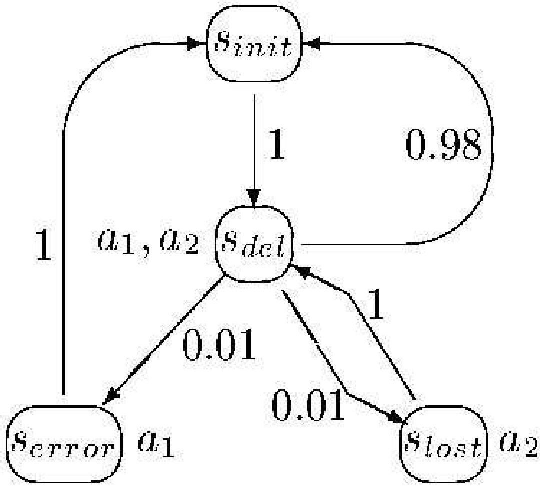
Property: $[\text{Try\_to\_deliver } U^{\leq 100} \text{ Correctly\_delivered}]_{\geq 0.9}$

- In a more mathematic speech, pick a path π at random, the probability that $\pi = T^k C$ for some $k \leq 100$ has to be ≥ 0.9



Property: $[\text{Try\_to\_deliver } U^{\leq 100} \text{ Correctly\_delivered}]_{\geq 0.9}$

- In a more mathematic speech, pick a path π at random, the probability that $\pi = T^k C$ for some $k \leq 100$ has to be ≥ 0.9
- In our framework, $P[T\ U^{\leq 100}\ C] = \sum_{\pi \mid \exists k \leq 100:\ \pi = T^k C} \mathbf{P}(\pi)$ holds



Property: $[\text{Try\_to\_deliver } U^{\leq 100} \text{ Correctly\_delivered}]_{\geq 0.9}$

- In a more mathematic speech, pick a path π at random, the probability that $\pi = T^k C$ for some $k \leq 100$ has to be ≥ 0.9
- In our framework, $P[T\ U^{\leq 100}\ C] = \sum_{\pi | \exists k \leq 100:\ \pi = T^k C} \mathbf{P}(\pi)$ holds
- Given a BPCTL formula $[\Phi]_{\geq 0.9}$, our algorithms computes $P[\Phi] = \sum_{\pi | \pi \models \Phi} \mathbf{P}(\pi)$

- Existing approaches to probabilistic model checking

- Existing approaches to probabilistic model checking
  - All based on symbolic computations

- Existing approaches to probabilistic model checking
  - All based on symbolic computations
  - $\mathbf{P}$ (and the results of the computations on it) is represented with the MTBDD data structure

- Existing approaches to probabilistic model checking
  - All based on symbolic computations
  - P (and the results of the computations on it) is represented with the MTBDD data structure
  - Idea derivated from the OBDD of standard model checking

- Existing approaches to probabilistic model checking
  - All based on symbolic computations
  - $\mathbf{P}$ (and the results of the computations on it) is represented with the MTBDD data structure
  - Idea derivated from the OBDD of standard model checking
  - This is ok if $\mathbf{P}$ is "regular"

- Existing approaches to probabilistic model checking
  - All based on symbolic computations
  - $\mathbf{P}$ (and the results of the computations on it) is represented with the MTBDD data structure
  - Idea derivated from the OBDD of standard model checking
  - This is ok if $\mathbf{P}$ is "regular"
  - If it is not, an exponential amount of RAM memory is needed

- Existing approaches to probabilistic model checking
  - All based on symbolic computations
  - $\mathbf{P}$ (and the results of the computations on it) is represented with the MTBDD data structure
  - Idea derivated from the OBDD of standard model checking
  - This is ok if $\mathbf{P}$ is “regular”
  - If it is not, an exponential amount of RAM memory is needed
  - Our approach tries to avoid this, at least for some classes of Markov Chains

- $\text{F}_{\text{inite}}\text{H}_{\text{orizon}}\text{P}_{\text{robabilistic}}\text{-Mur}\varphi$

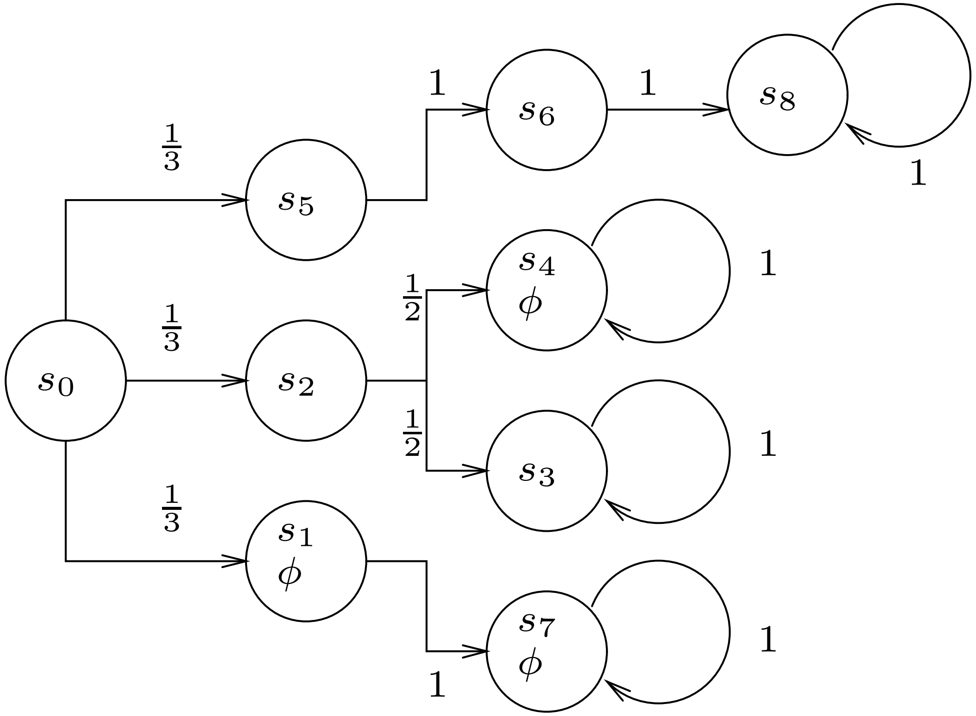
- F$_{\text{inite}}$H$_{\text{orizon}}$P$_{\text{robabilistic}}$-Murφ
- Explicit probabilistic model checker

- Finite H orizon P robabilistic -Murφ
- Explicit probabilistic model checker
  - explicit verification often outperforms symbolic verification in non-probabilistic model checking

- $\mathrm{F}_{\mathrm{inite}}\mathrm{H}_{\mathrm{orizon}}\mathrm{P}_{\mathrm{robabilistic}}\text{-Mur}\varphi$
- 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

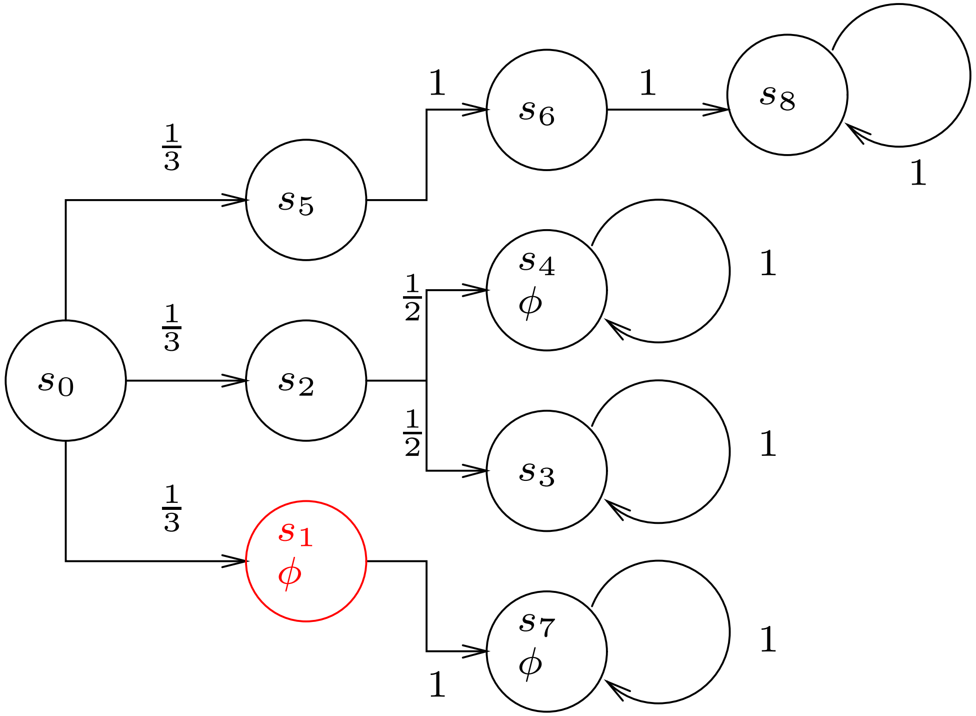
- $^{\text{F}}$inite$^{\text{H}}$orizon$^{\text{P}}$robabilistic$^{\text{-Mur}\varphi}$
- Explicit probabilistic model checker
  - explicit verification often outperforms symbolic verification in non-probabilistic model checking
  - we will show that this holds also for probabilistic model checking
- Murφ modified in the input language and in the verification algorithm

- ^{F}inite^{H}orizon^{P}robabilistic^{-}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



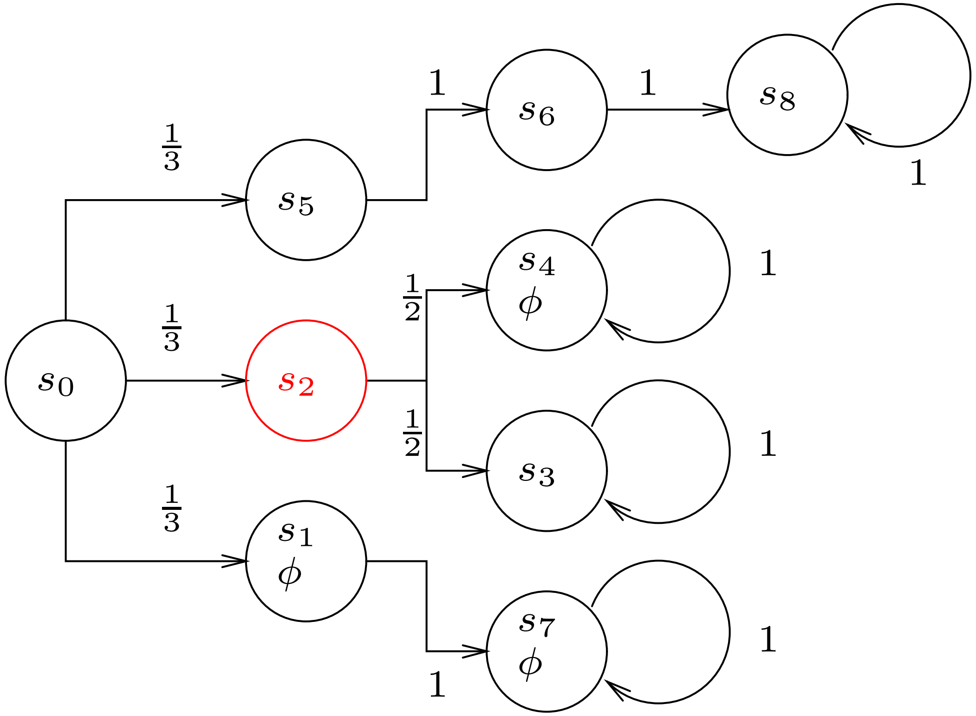
- We want to verify if $s_0 \models [tt\ \mathbf{U}^{\leq 2}\ \phi]_{\geq 0.5}$
- ϕ holds in s_1, s_4, s_7
- The searched probability is: 0

The Basic DF Visit



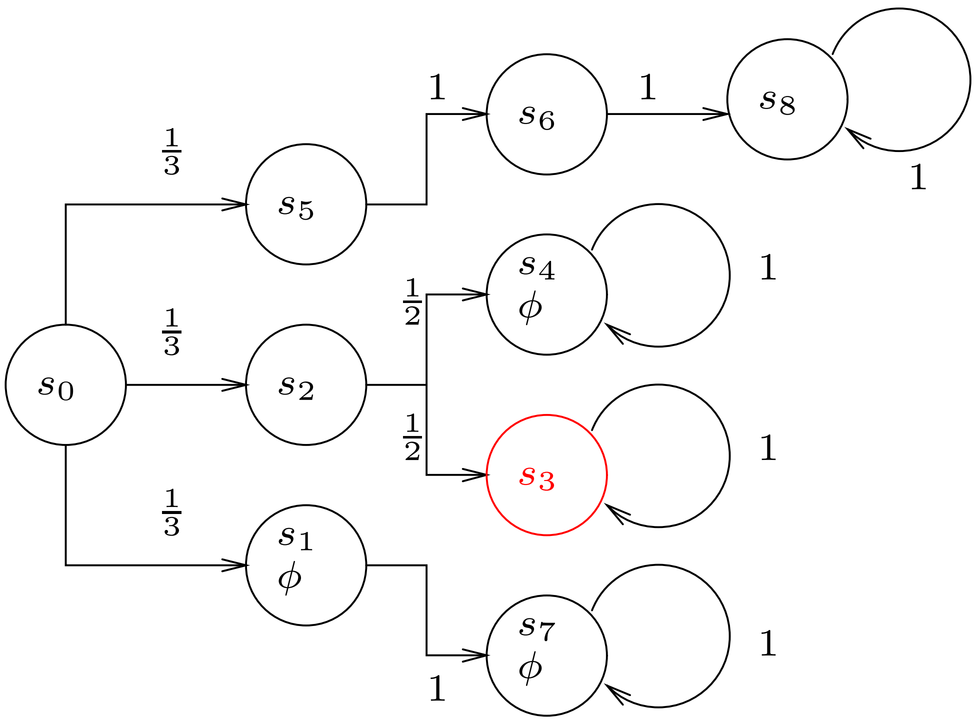
- We want to verify if $s_0 \models [tt \; \mathbf{U}^{\leq 2} \; \phi]_{\geq 0.5}$
- ϕ holds in s_1, s_4, s_7
- The searched probability is: $\frac{1}{3} \times 1$

The Basic DF Visit



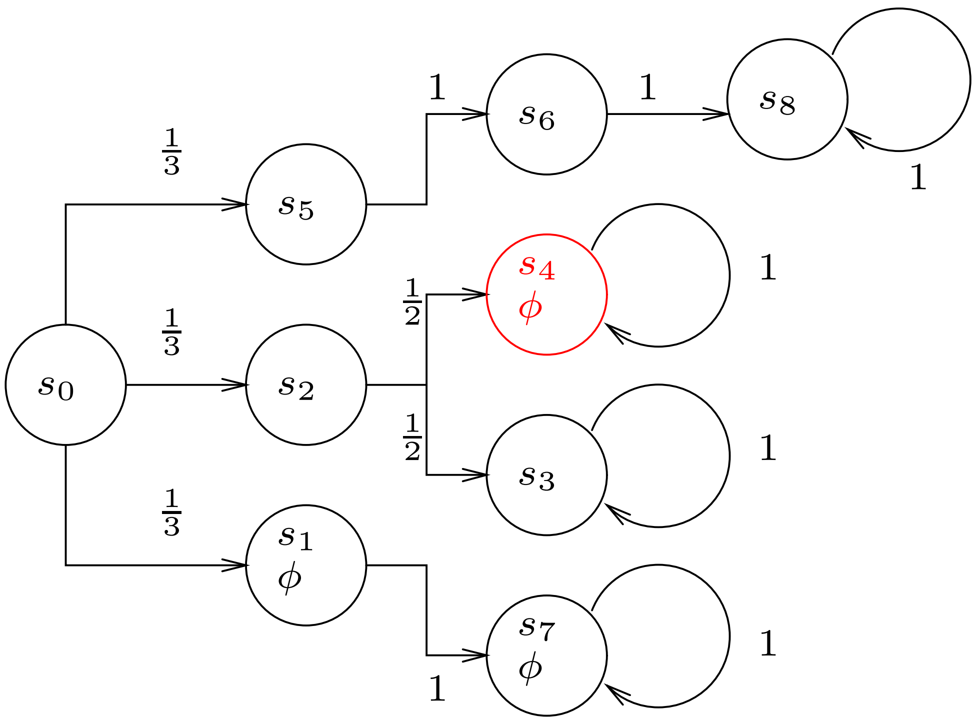
- We want to verify if $s_0 \models [tt\ \mathbf{U}^{\leq 2}\ \phi]_{\geq 0.5}$
- ϕ holds in s_1, s_4, s_7
- The searched probability is: $\frac{1}{3} + \frac{1}{3} \times \ldots$

The Basic DF Visit



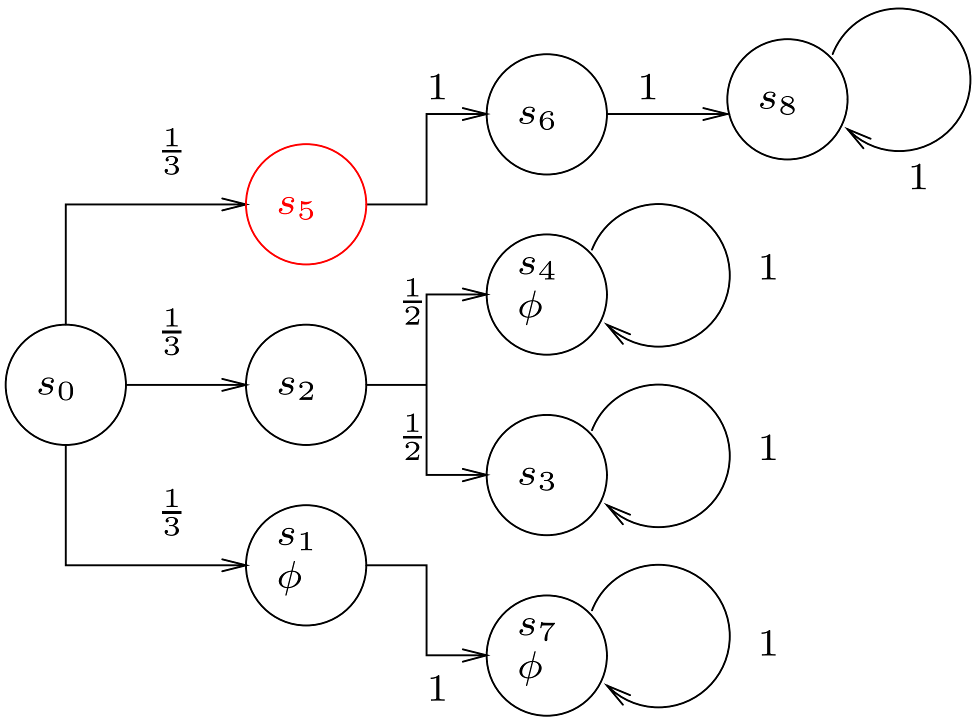
- We want to verify if $s_0 \models [tt\ \mathbf{U}^{\leq 2}\ \phi]_{\geq 0.5}$
- ϕ holds in s_1, s_4, s_7
- The searched probability is: $\frac{1}{3} + \frac{1}{3} \times (\frac{1}{2} \times 0 + \ldots$

The Basic DF Visit

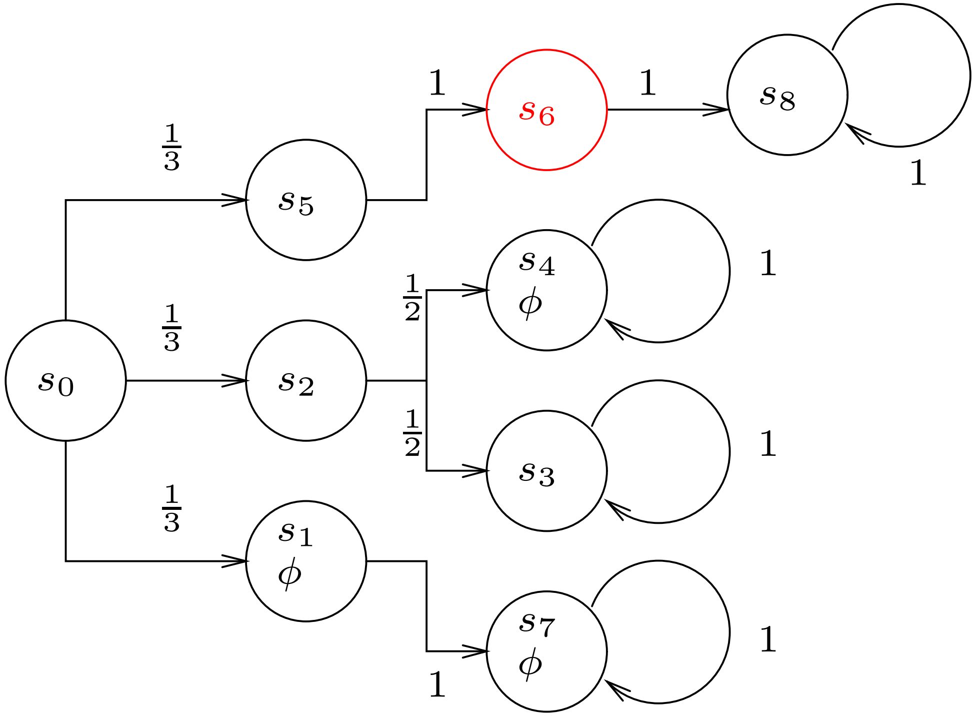


- We want to verify if $s_0 \models [tt\ \mathbf{U}^{\leq 2}\ \phi]_{\geq 0.5}$
- ϕ holds in s_1, s_4, s_7
- The searched probability is: $\frac{1}{3} + \frac{1}{3} \times (\frac{1}{2} \times 0 + \frac{1}{2} \times 1)$

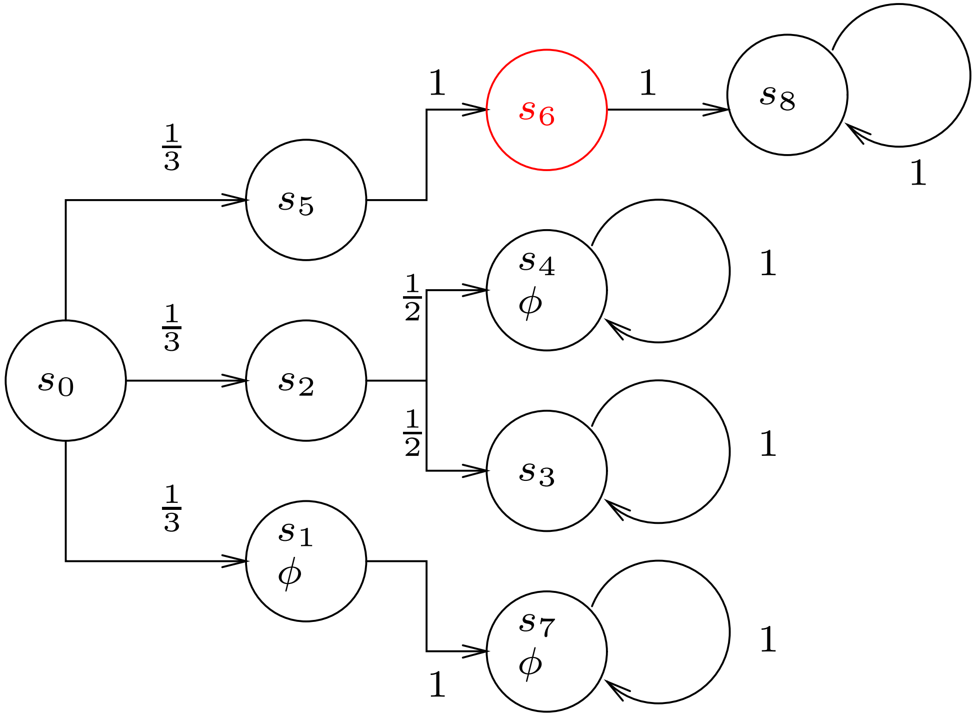
The Basic DF Visit



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

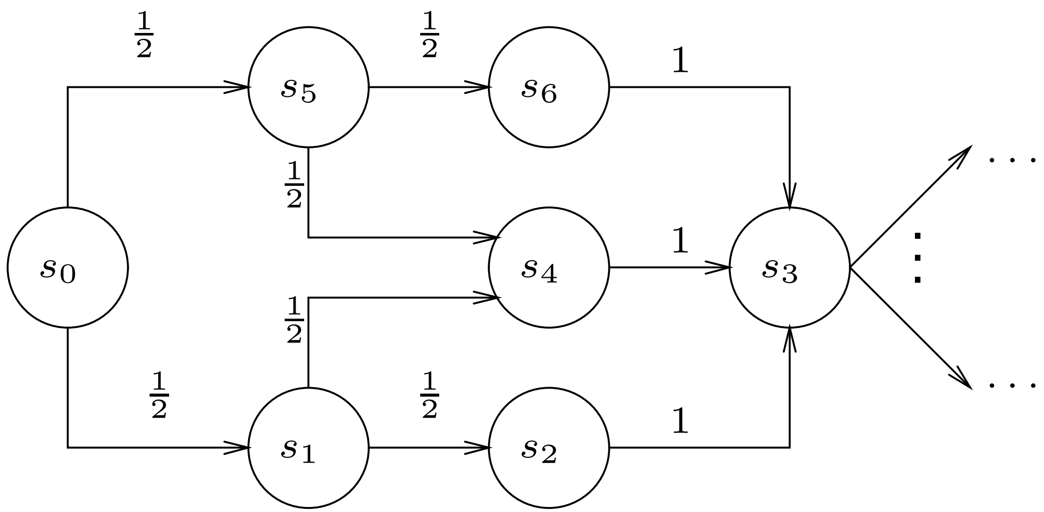


- We want to verify if $s_0 \models [tt\ \mathbf{U}^{\leq 2}\ \phi]_{\geq 0.5}$
- ϕ holds in s_1, s_4, s_7
- The searched probability is: $\frac{1}{3} + \frac{1}{3} \times \frac{1}{2} + \frac{1}{3} \times 0$



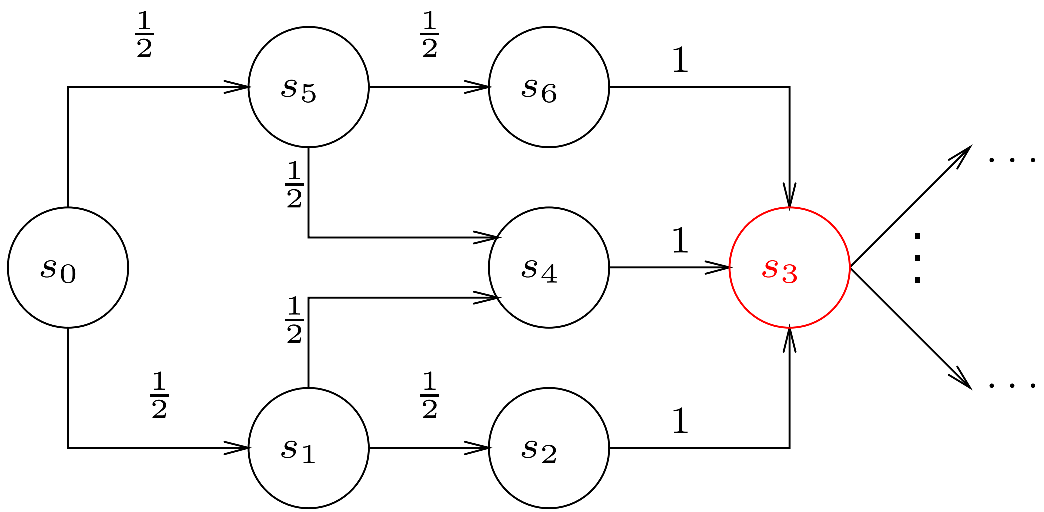
- We want to verify if $s_0 \models [tt\ \mathbf{U}^{\leq 2}\ \phi]_{\geq 0.5}$
- ϕ holds in s_1, s_4, s_7
- The searched probability is: $\frac{1}{3} + \frac{1}{3} \times \frac{1}{2} + \frac{1}{3} \times 0$
- Finally, we have $\frac{1}{3} + \frac{1}{3} \times \frac{1}{2} = \frac{1}{2} \geq 0.5$, so the property is verified

Cache Usage (1)



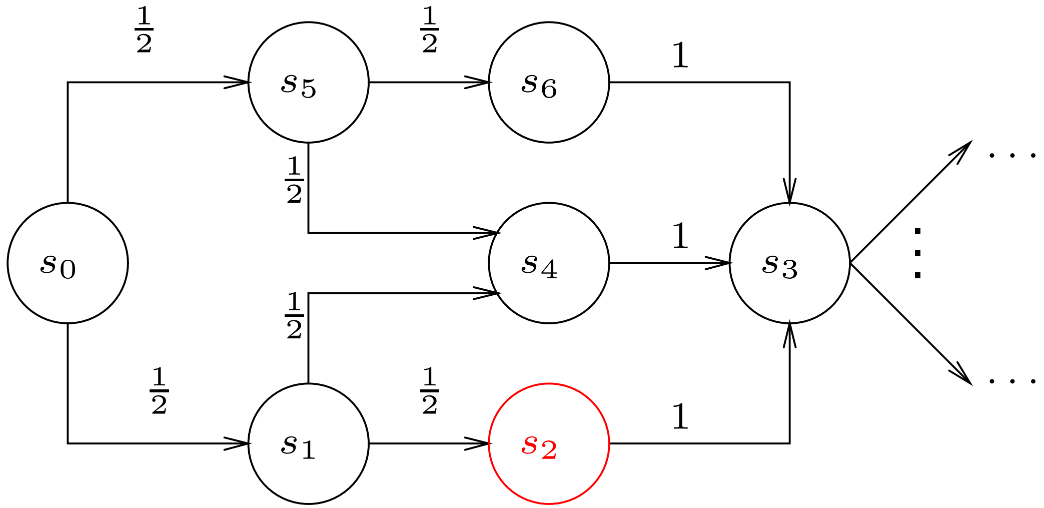
- We want to verify if $s_0 \models F$, being $F \equiv [\Phi \ \mathbf{U}^{\leq k} \ \Psi]_{\leq 0.5}$
- The cache stores 4-tuples $\{s, F, h, p\}$
  - p is the probability of $\Phi \ \mathbf{U}^{\leq h} \ \Psi$

Cache Usage (1)



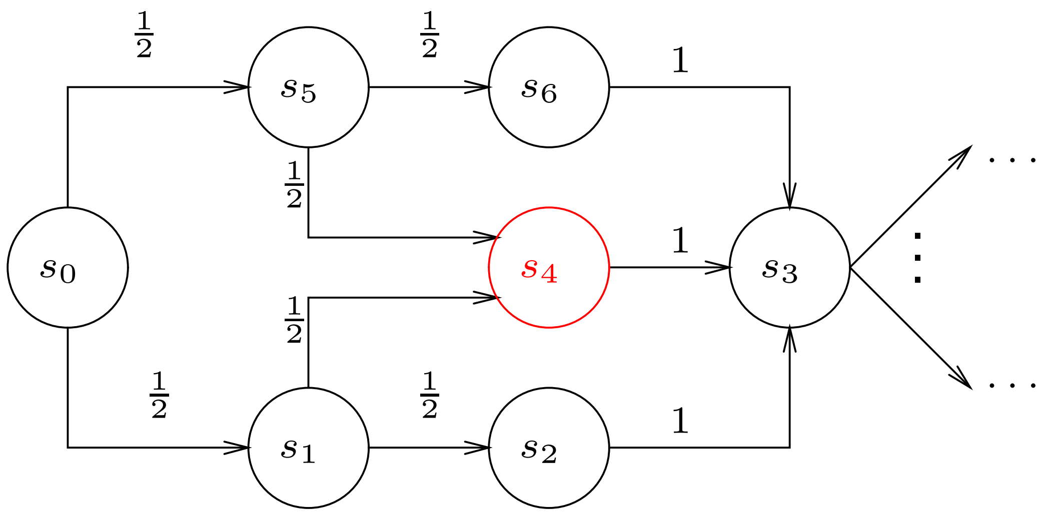
- When the DF visit of s_3 is completed, $\{s_3, F, k-3, p_3\}$ is inserted in the cache
  - p_3 is the probability value computed by the DF on s_3
  - k is decremented of 3 because s_3 is reached in 3 steps from s_0

Cache Usage (1)



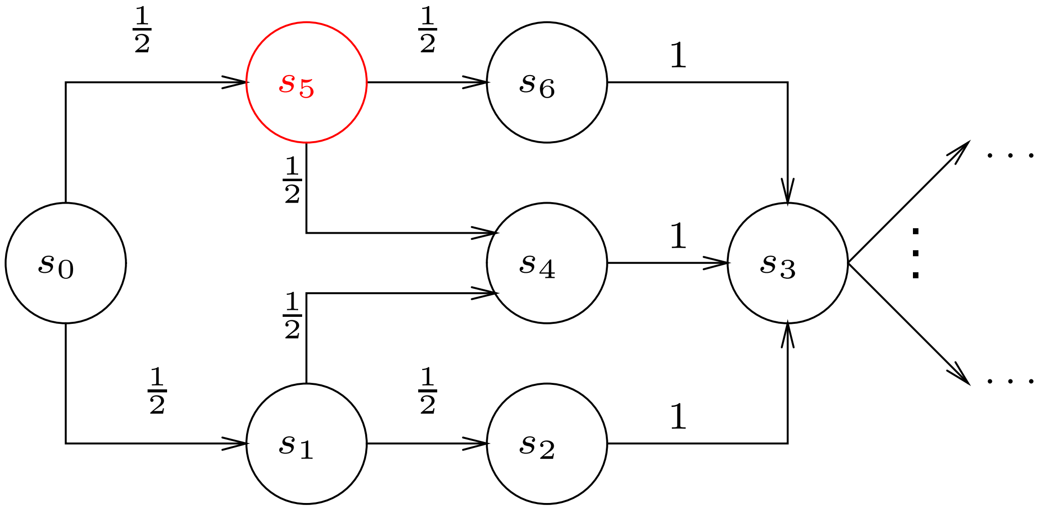
- When the DF visit of s_3 is completed, $\{s_3, F, k-3, p_3\}$ is inserted in the cache
  - p_3 is the probability value computed by the DF on s_3
  - k is decremented of 3 because s_3 is reached in 3 steps from s_0
- Analogously, $\{s_2, F, k-2, p_2\}$ is inserted in the cache

Cache Usage (1)



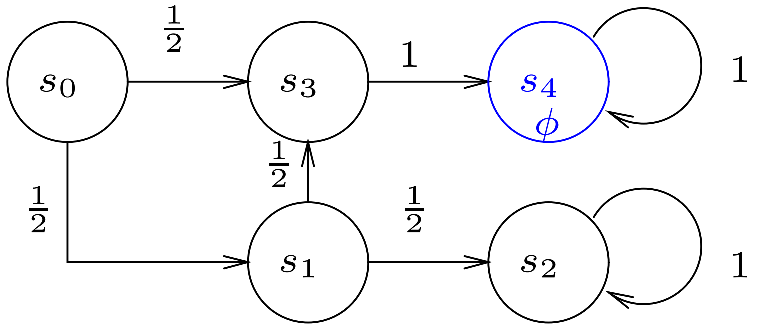
- In this way, the DF visit of s_4 can directly compute $p_4 = p_3 \times 1$
  - p_3 is not computed, but it is found on the cache
- Then, $\{s_4, F, k-2, p_4\}$ is inserted in the cache

Cache Usage (1)



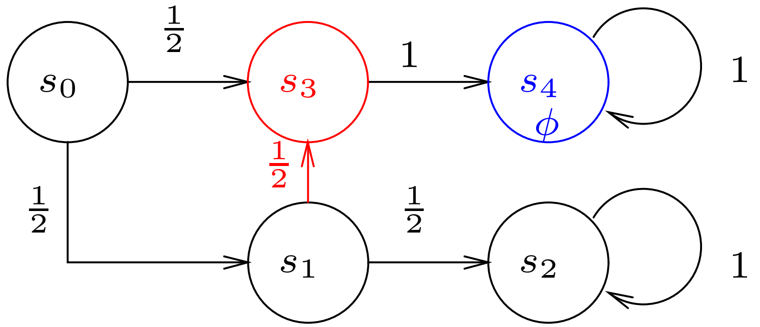
- Analogously, when the DF visit of s_5 starts, the nested DF visit of s_4 is skipped
  - p_4 is not computed, but it is found on the cache
- The result of the DF visit of s_6 will be multiplied by $\frac{1}{2}$ and then added to $\frac{1}{2} \times p_4$

Cache Usage (2)



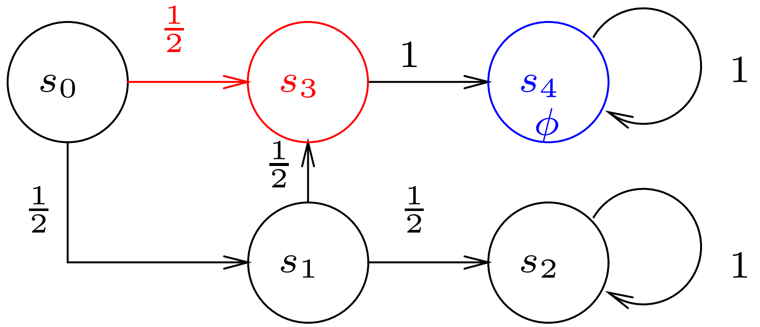
- We want to verify if $s_0 \models F$
  - $F \equiv [tt\ \mathbf{U}^{\leq 2}\ \Phi]_{\leq 0}$
  - $\Phi \equiv [tt\ \mathbf{U}^{\leq 2}\ \phi]_{\geq 1}$
  - $\phi(s_4) = 1, \forall s \neq s_4.\ \phi(s) = 0.$

Cache Usage (2)



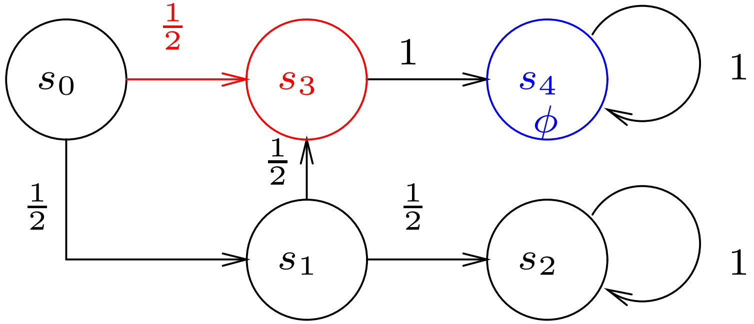
- We want to verify if $s_0 \models F$,
  - $F \equiv [tt\ \mathbf{U}^{\leq 2}\ \Phi]_{\leq 0}$
  - $\Phi \equiv [tt\ \mathbf{U}^{\leq 2}\ \phi]_{\geq 1}$
  - $\phi(s_4) = 1, \forall s \neq s_4.\ \phi(s) = 0$.
- s_3 is visited for the first time as a successor of s_1
  - The 4-tuple $< s_3, \Phi, 1, 1.0 >$ is stored on the cache

Cache Usage (2)



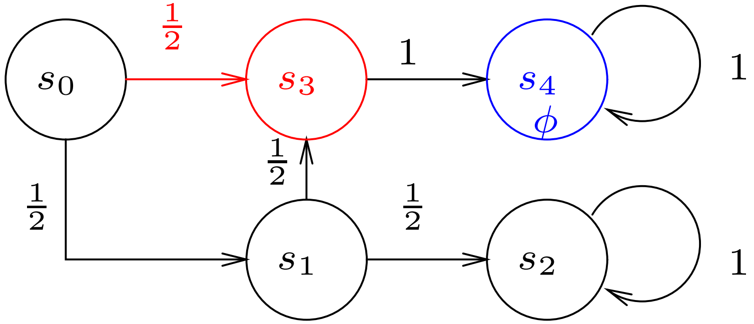
- s_3 is visited for the second time as a successor of s_0

Cache Usage (2)

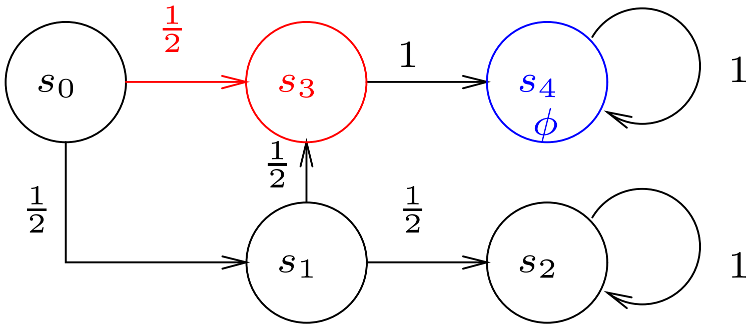


- s_3 is visited for the second time as a successor of s_0
  - The required 4-tuple $< s_3, \Phi, 2, p >$ is not on the cache

Cache Usage (2)

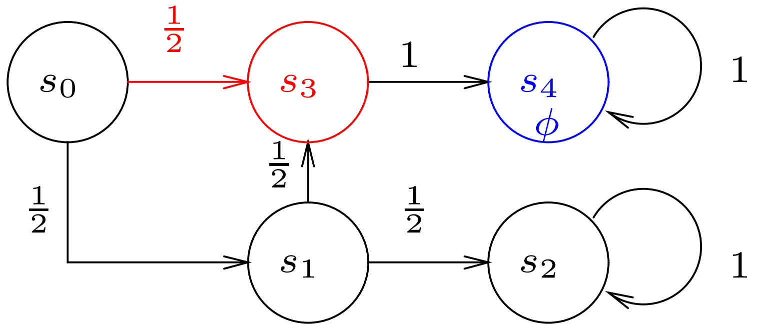


- s_3 is visited for the second time as a successor of s_0
  - The required 4-tuple $< s_3, \Phi, 2, p >$ is not on the cache
  - However, there is the 4-tuple $< s_3, \Phi, 1, 1.0 >$ (so, with a smaller horizon)



- s_3 is visited for the second time as a successor of s_0
  - The required 4-tuple $< s_3, \Phi, 2, p >$ is not on the cache
  - However, there is the 4-tuple $< s_3, \Phi, 1, 1.0 >$ (so, with a smaller horizon)
  - The stored probability 1.0 is already in the right relation with the probability bound given in $\Phi \equiv [tt \ \mathbf{U}^{\leq 2} \ \phi]_{\boxed{\geq 1}}$

Cache Usage (2)



- s_3 is visited for the second time as a successor of s_0
  - The required 4-tuple $< s_3, \Phi, 2, p >$ is not on the cache
  - However, there is the 4-tuple $< s_3, \Phi, 1, 1.0 >$ (so, with a smaller horizon)
  - The stored probability 1.0 is already in the right relation with the probability bound given in $\Phi \equiv [tt\ \mathbf{U}^{\leq 2}\ \phi]_{\boxed{\geq 1}}$
  - So, the second DF visit on s_3 is avoided

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 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

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-Murφ works better

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

- Probabilistic Safety Verification

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

- Probabilistic Safety Verification
- Probabilistic Robustness Verification

NPHIL	MAX_WAIT	Result	Murφ Mem (MB)	PRISM Mem (MB)	Murφ Time (s)	PRISM Time (s)
Modified Pnueli-Zuck						
5	3	false	5.0e+2	9.168246e+02	1.28381900e+04	1.196793e+03
5	4	false	5.0e+2	N/A	1.27377300e+04	N/A
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:

- If a philosopher risks to die, then it will eat soon
- $[tt \ \mathbf{U}^{\leq k_1} \ (\phi_{und} \wedge \neg[tt \ \mathbf{U}^{\leq k_2} \neg\phi_{err}]_{\geq 1})]_{\leq 0}$.

`NPHIL`, `MAX_WAIT`: protocol parameters

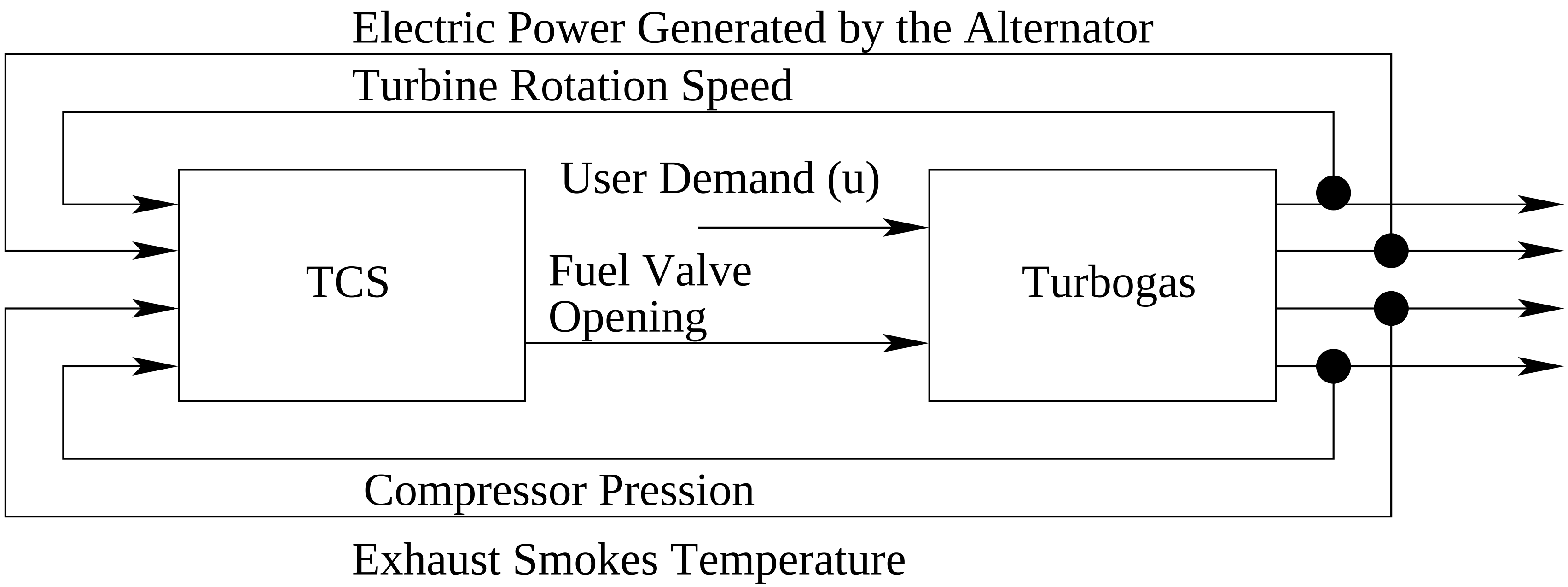
- ICARO: 2MW Electric Co-generative Power Plant, in operation at the ENEA Research Center of Casaccia (Italy)

- 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

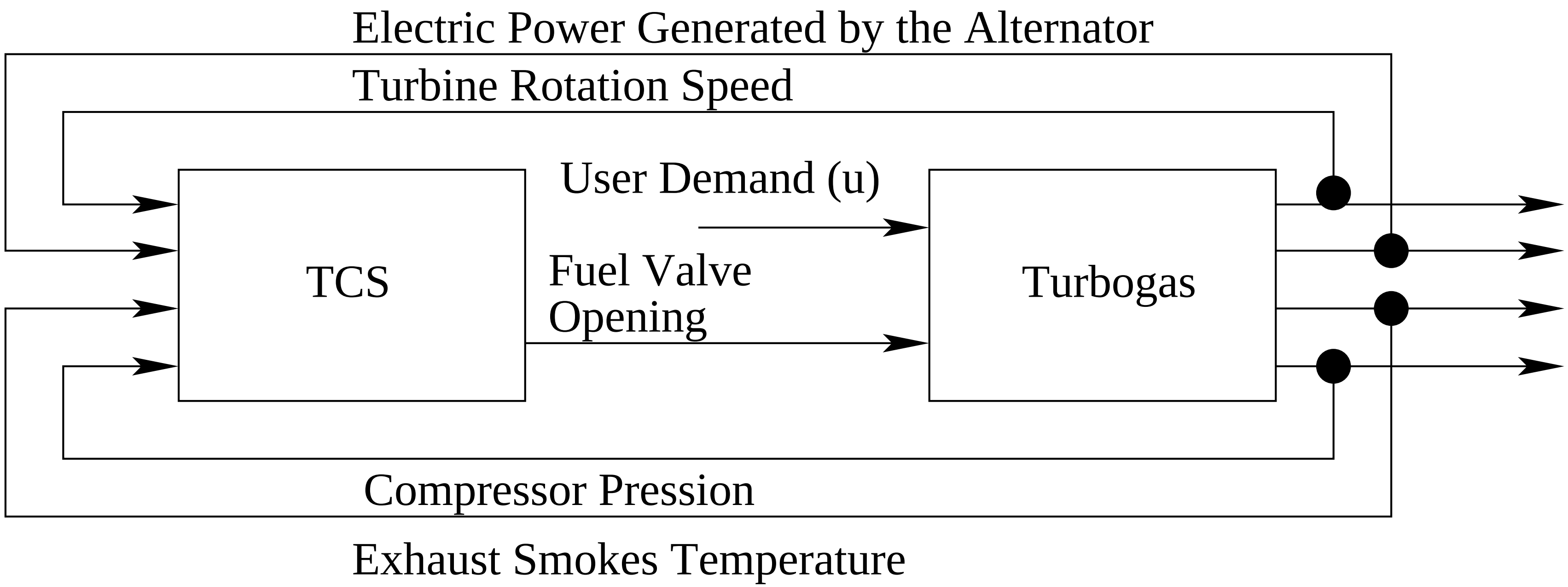
A Turbogas Control System

- 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

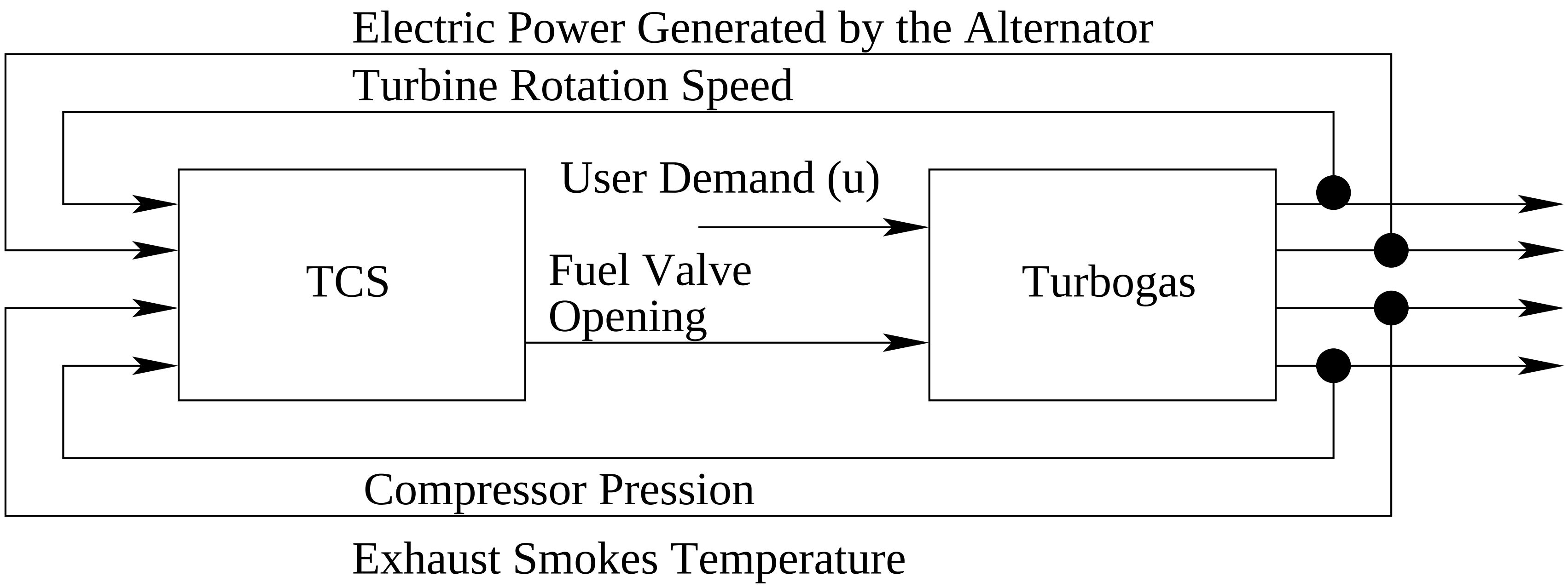


A Turbogas Control System

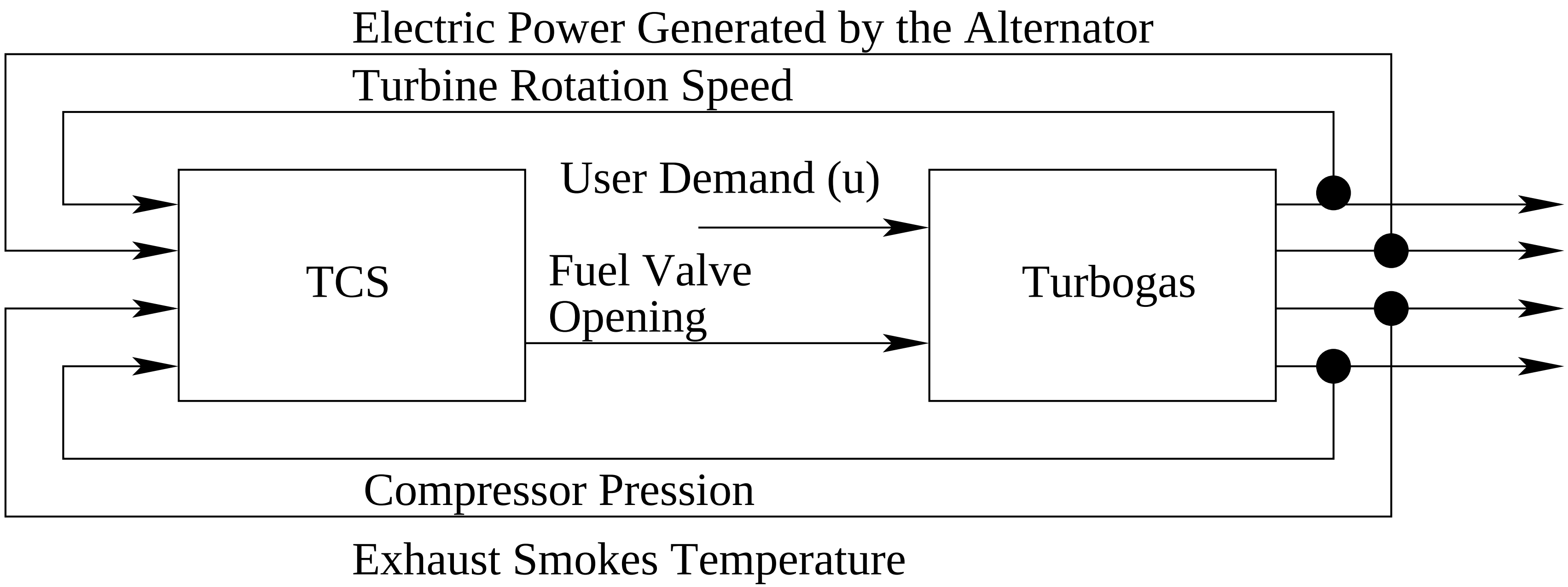


- TCS is an electronic circuit, its detail are known

A Turbogas Control System



- 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 (MAX_D_U)

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

- 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 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
- As a result, if the user demand varies too much rapidly (i.e. MAX_D_U is too high), the controller fails

- 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

$$p(u,i) = \begin{cases} 0.4 + \beta\frac{(u-\frac{M}{2})|u-\frac{M}{2}|}{M^2} & \text{if } i = -1 \\ 0.2 & \text{if } i = 0 \\ 0.4 + \beta\frac{(\frac{M}{2}-u)|u-\frac{M}{2}|}{M^2} & \text{if } i = +1 \end{cases}$$

  - 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*
- `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*
- `MAX_D_U` has a value that force the non-probabilistic verification to fail

MAX_D_U	Reachable States	Finite Horizon	CPU Time	Probability
25	3018970	1600	68562.570	7.373291768e-05
35	2226036	1400	50263.020	1.076644427e-04
45	1834684	1300	41403.150	9.957147381e-05
50	83189	900	2212.360	3.984375e-03

- 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 *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 $[tt\ \mathbf{U}^{\leq k_1}\ (\neg\phi_{und} \wedge \neg[tt\ \mathbf{U}^{\leq k_2}\phi_{und}]_{\geq 1})]_{\leq 0}$

- 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 $[tt\ \mathbf{U}^{\leq k_1}\ (\neg\phi_{und} \wedge \neg[tt\ \mathbf{U}^{\leq k_2}\phi_{und}]_{\geq 1})]_{\leq 0}$
- 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 $[tt\ \mathbf{U}^{\leq k_1}\ (\neg\phi_{und} \wedge \neg[tt\ \mathbf{U}^{\leq k_2}\phi_{und}]_{\geq 1})]_{\leq 0}$
- k_1 is sufficient to reach an undesired state
- $k_2 = \frac{k_1}{100}$

`MAX_D_U`	Visited States	k_1	CPU Time (s)	Probability
35	1.159160e+05	800	3.702400e+03	4.104681e-03
45	4.098000e+04	700	1.313900e+03	1.792883e-02
50	4.067700e+04	700	1.307850e+03	3.825000e-02

Results on a machine with 2 processors (both INTEL Pentium III 500Mhz) and 2GB of RAM.

Murφ options used: `-m500` (use 500 MB of RAM)

Future Works

More features for FHP-Murφ and then comparison with PRISM

More features for FHP-Murφ 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

More features for FHP-Murφ 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

Improving performances

More features for FHP-Murφ 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

Improving performances

- Try to apply symmetry reduction (to be investigated)