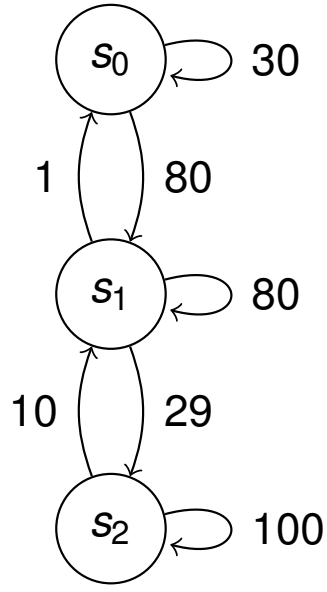
Symblicit Abstraction of Continuous-Time Markov Models for Transient Rewards

Ernst Moritz Hahn, Saarland University, Germany
Ralf Wimmer, Albert-Ludwigs-University Freiburg, Germany
Bernd Becker, Albert-Ludwigs-University Freiburg, Germany

ROCKS, April 27th, 2012

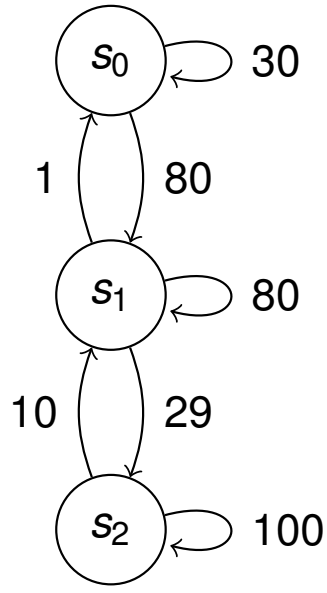
Problem



$$\begin{pmatrix} 30 & 80 & 0 \\ 1 & 80 & 29 \\ 0 & 10 & 100 \end{pmatrix}$$

- continuous-time Markov chains occur in many areas
  - performance evaluation
  - biology
  - ...

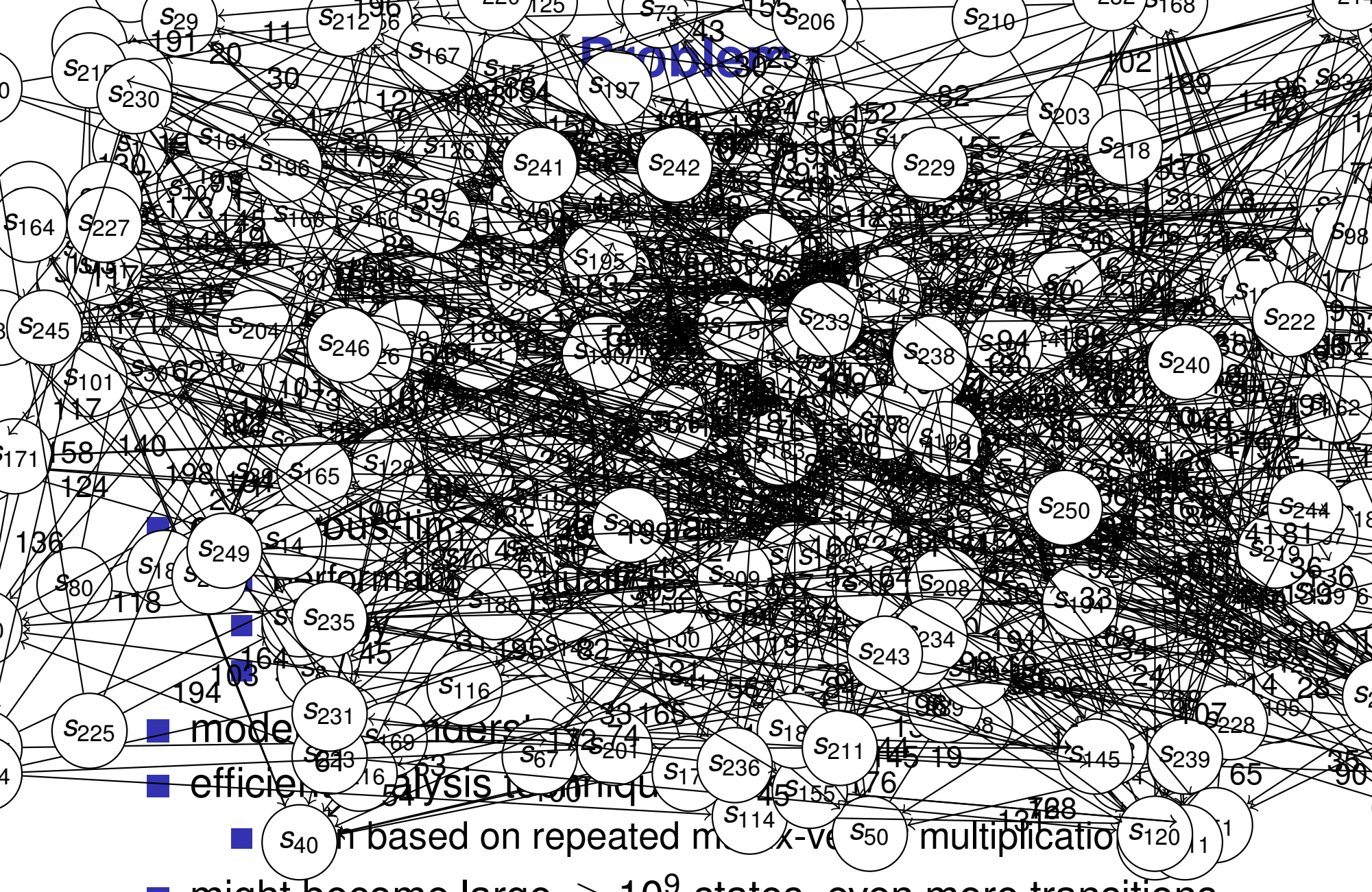
Problem



$$\begin{pmatrix} 30/110 & 80/110 & 0 \\ 1/110 & 80/110 & 29/110 \\ 0 & 10/110 & 100/110 \end{pmatrix} \cdot \begin{pmatrix} 0.4 \\ 0.7 \\ 0.2 \end{pmatrix} = \begin{pmatrix} next_0 \\ next_1 \\ next_2 \end{pmatrix}$$

- continuous-time Markov chains occur in many areas
  - performance evaluation
  - biology
  - ...
- model well understood
- efficient analysis techniques exist
  - often based on repeated matrix-vector multiplications

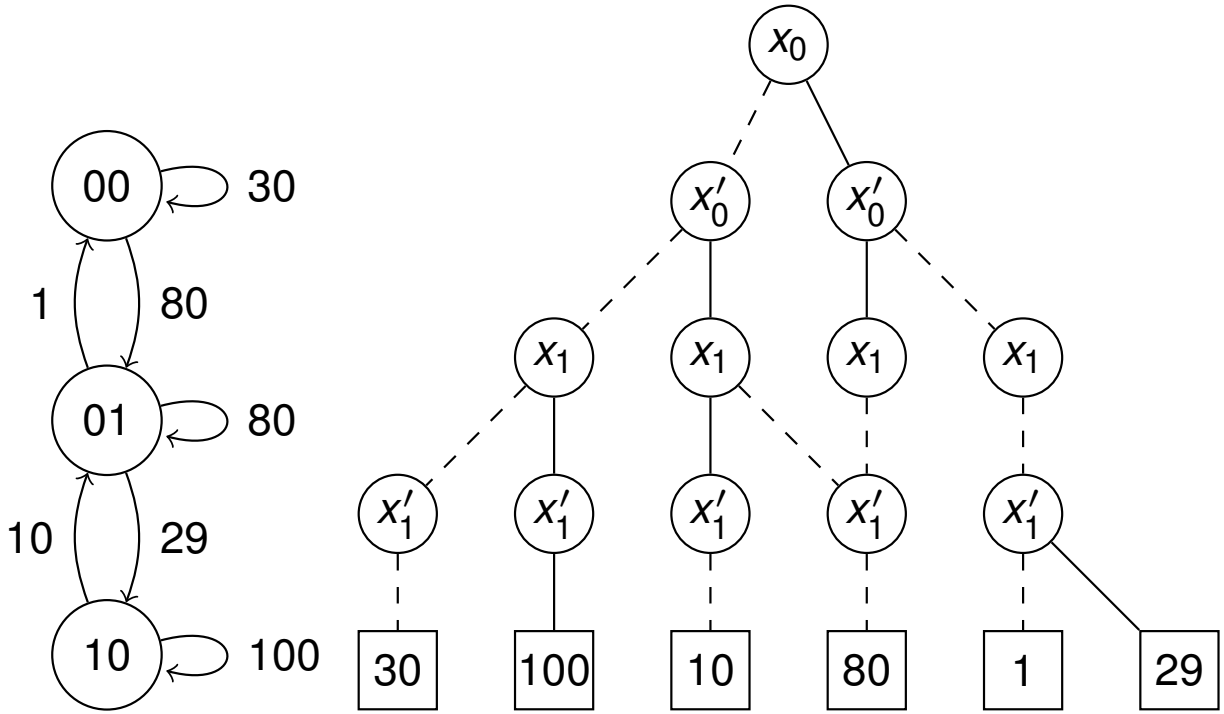
Problem



- model checkers' performance
- efficient analysis is important
  - based on repeated matrix-vector multiplication
- might become large, $\geq 10^9$ states, even more transitions
- $\Rightarrow$ direct analysis often not possible

Existing methods (1/2)

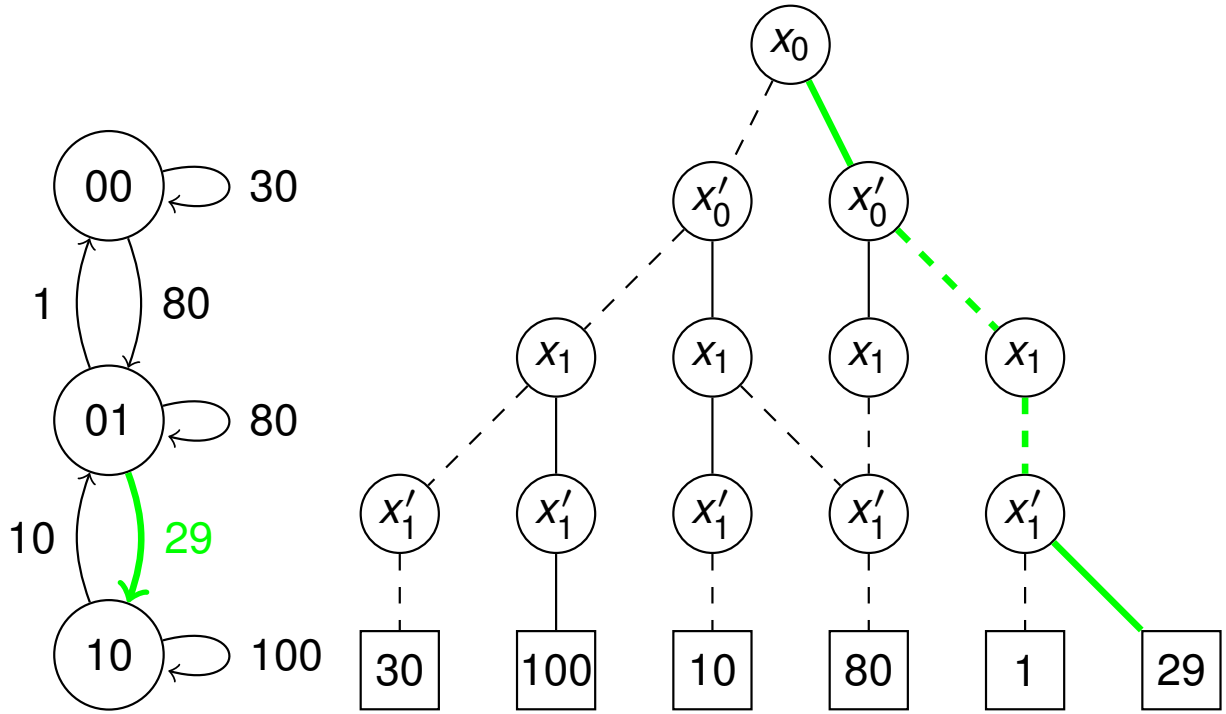
Multi-terminal binary decision diagrams (MTBDDs)



- extension of BDDs with more than two terminal nodes

Existing methods (1/2)

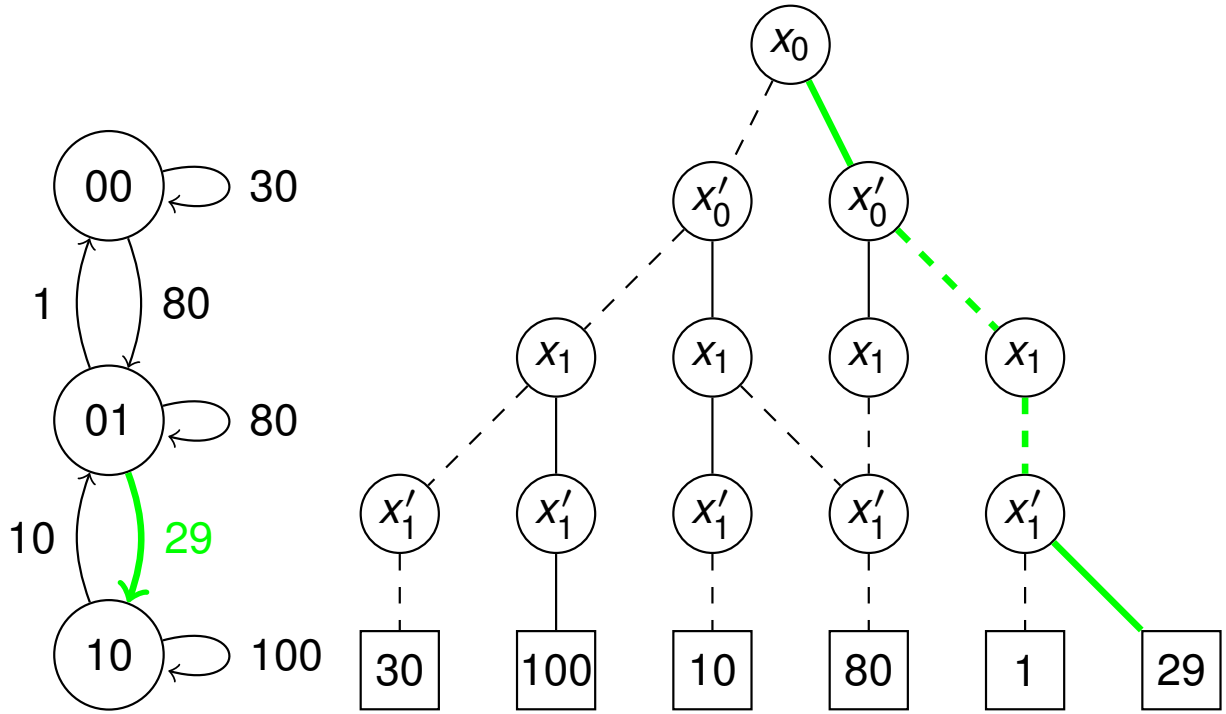
Multi-terminal binary decision diagrams (MTBDDs)



- extension of BDDs with more than two terminal nodes
- to store transition matrix and for matrix-vector mult.

Existing methods (1/2)

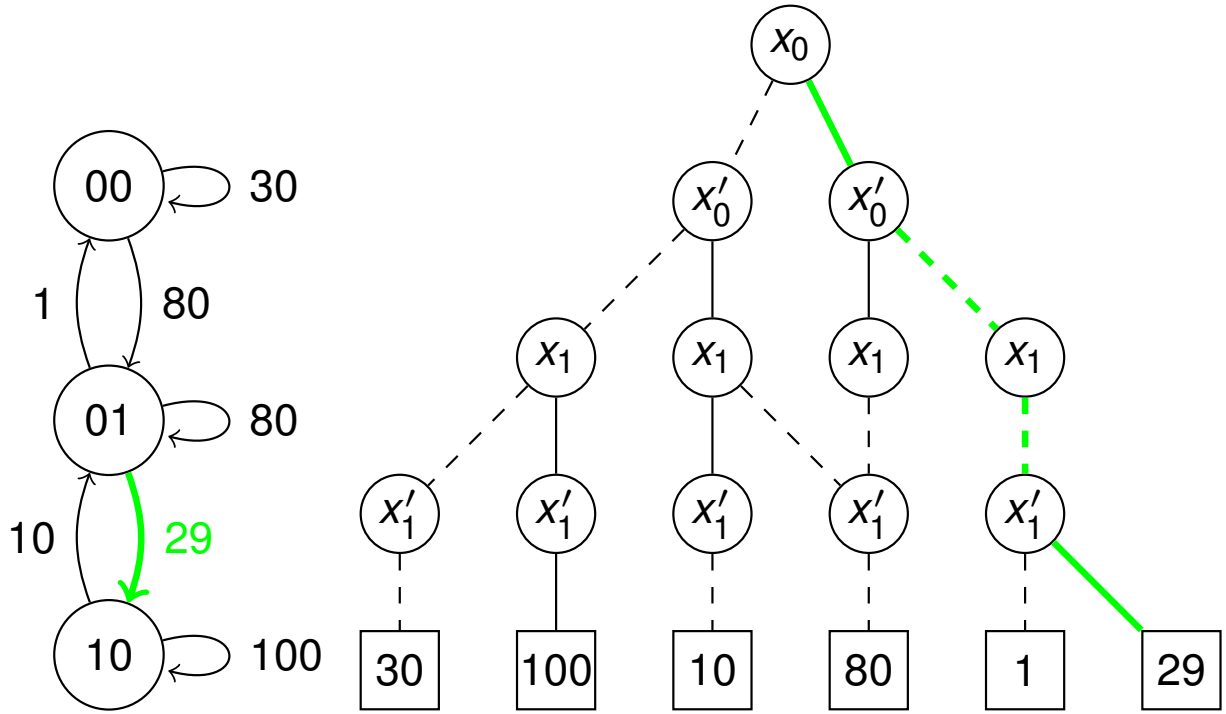
Multi-terminal binary decision diagrams (MTBDDs)



- extension of BDDs with more than two terminal nodes
- to store transition matrix and for matrix-vector mult.
- works very well in some cases
  - even for $\geq 10^9$ states

Existing methods (1/2)

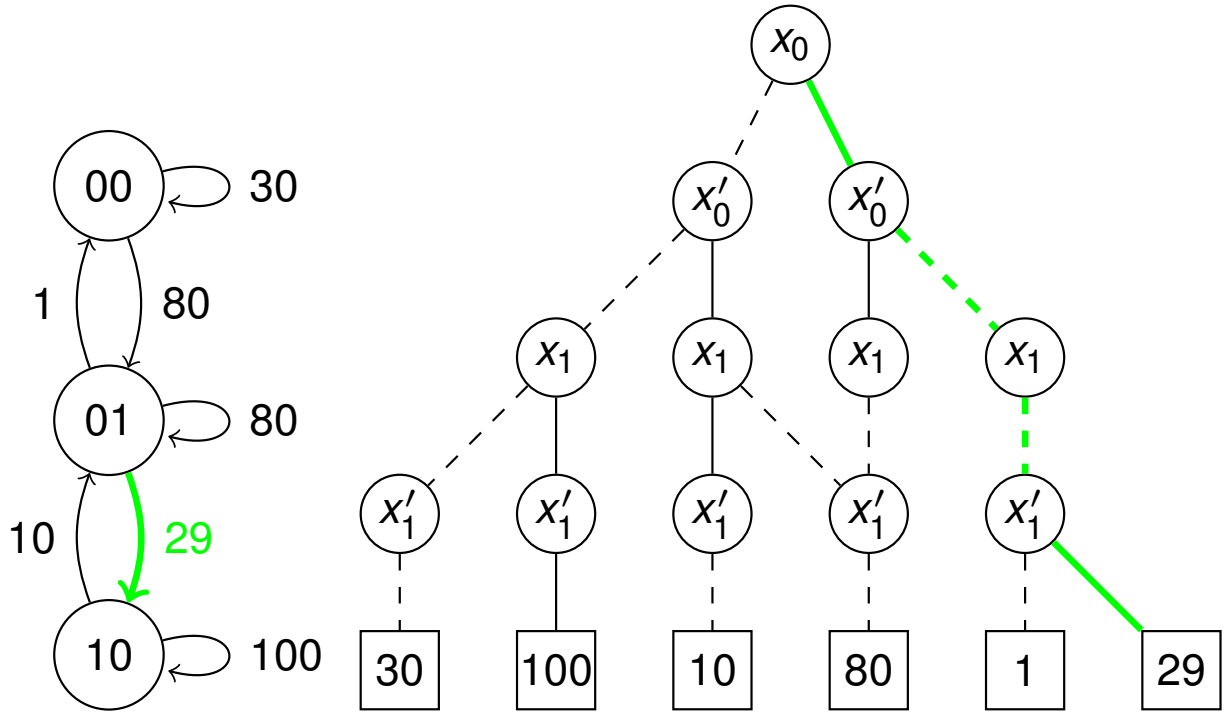
Multi-terminal binary decision diagrams (MTBDDs)



- extension of BDDs with more than two terminal nodes
- to store transition matrix and for matrix-vector mult.
- works very well in some cases
  - even for $\geq 10^9$ states
- does not work well if many different rates/probabilities

Existing methods (1/2)

Multi-terminal binary decision diagrams (MTBDDs)



- extension of BDDs with more than two terminal nodes
- to store transition matrix and for matrix-vector mult.
- works very well in some cases
  - even for $\geq 10^9$ states
- does not work well if many different rates/probabilities
  - happens often in value iteration

Existing methods (2/2)

Hybrid method

$$\begin{pmatrix} 0.4 \\ 0.7 \\ 0.2 \end{pmatrix}$$
$$\begin{pmatrix} next_0 \\ next_1 \\ next_2 \end{pmatrix}$$

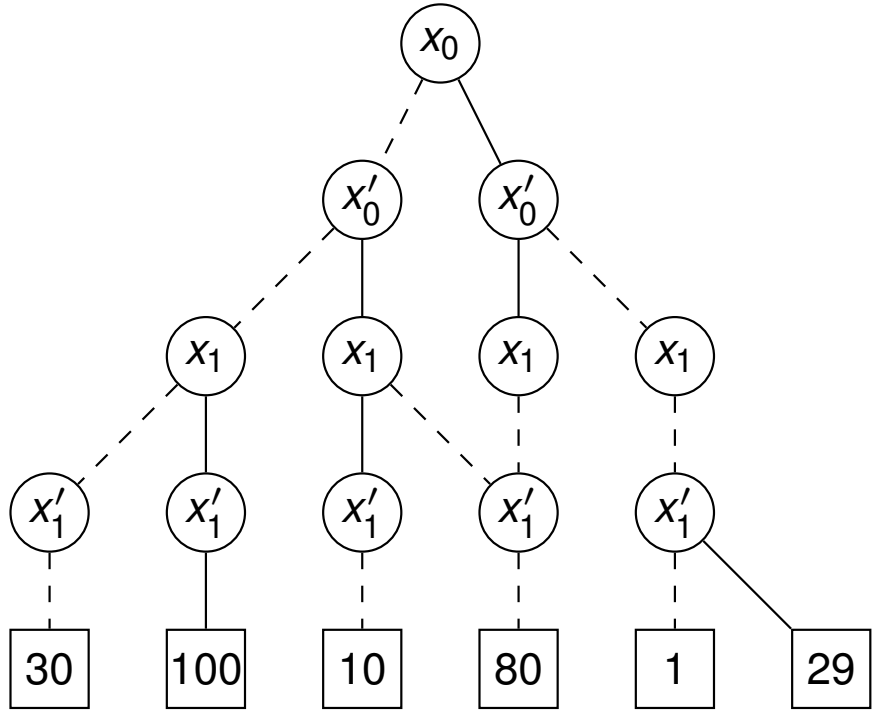
- stores state values explicitly

Existing methods (2/2)

Hybrid method

$$\begin{pmatrix} 0.4 \\ 0.7 \\ 0.2 \end{pmatrix}$$

$$\begin{pmatrix} next_0 \\ next_1 \\ next_2 \end{pmatrix}$$



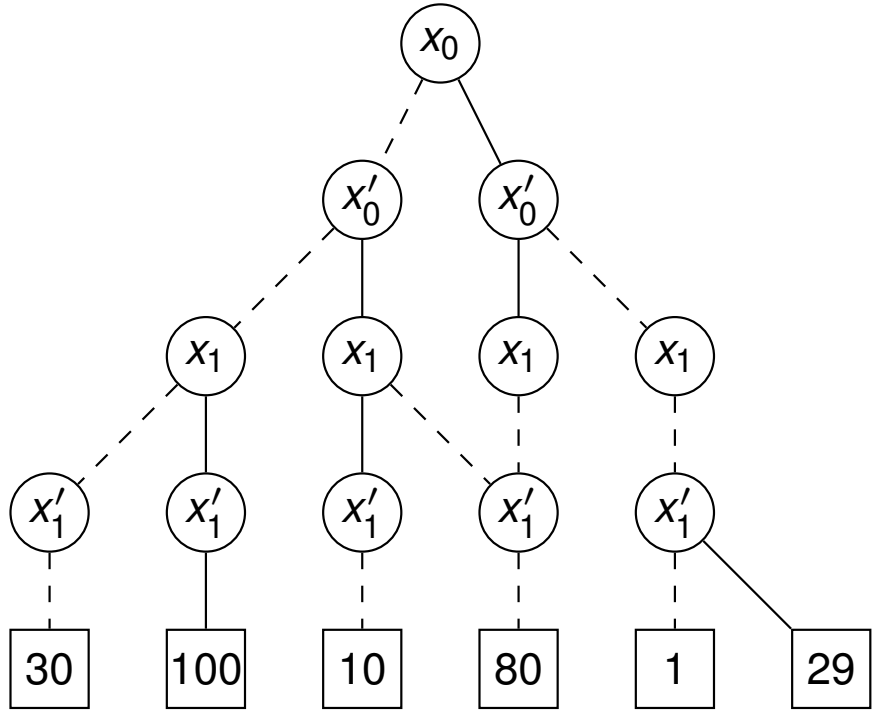
- stores state values explicitly
- and transition matrix in BDD-like way

Existing methods (2/2)

Hybrid method

$$\begin{pmatrix} 0.4 \\ 0.7 \\ 0.2 \end{pmatrix}$$

$$\begin{pmatrix} next_0 \\ next_1 \\ next_2 \end{pmatrix}$$



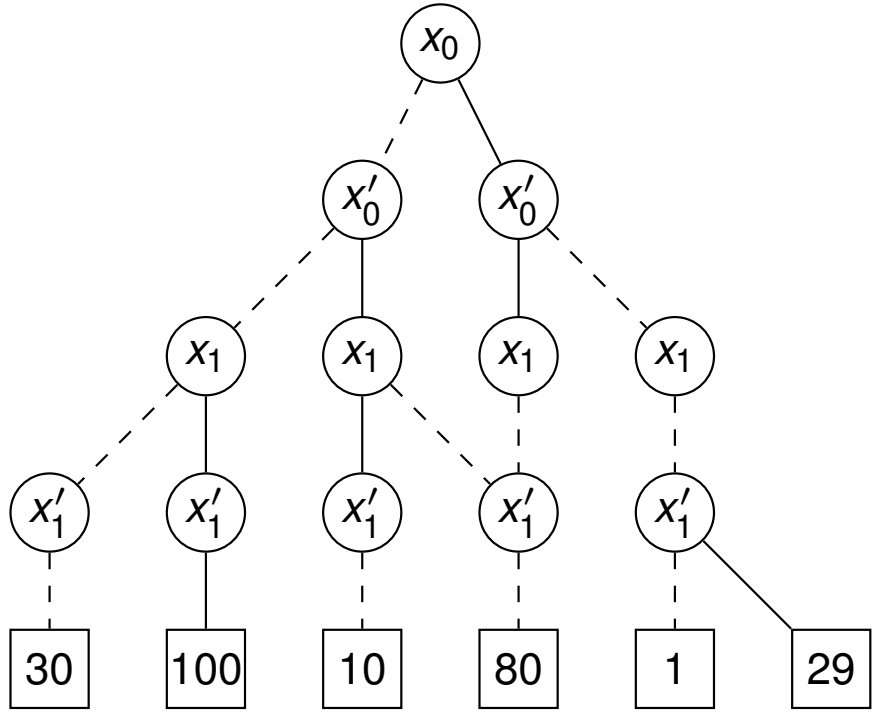
- stores state values explicitly
- and transition matrix in BDD-like way
- can reduce memory usage for transitions

Existing methods (2/2)

Hybrid method

$$\begin{pmatrix} 0.4 \\ 0.7 \\ 0.2 \end{pmatrix}$$

$$\begin{pmatrix} next_0 \\ next_1 \\ next_2 \end{pmatrix}$$

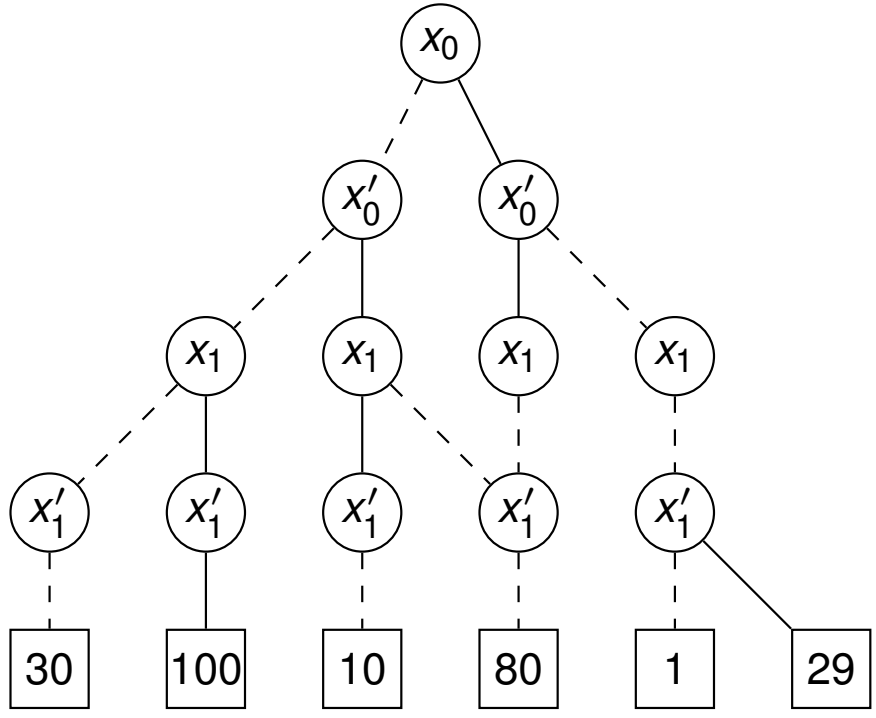


- stores state values explicitly
- and transition matrix in BDD-like way
- can reduce memory usage for transitions
- might be slow

Existing methods (2/2)

Hybrid method

$$\begin{pmatrix} 0.4 \\ 0.7 \\ 0.2 \end{pmatrix} \quad \begin{pmatrix} next_0 \\ next_1 \\ next_2 \end{pmatrix}$$



- stores state values explicitly
- and transition matrix in BDD-like way
- can reduce memory usage for transitions
- might be slow
- not applicable in case of very large state sets

Idea

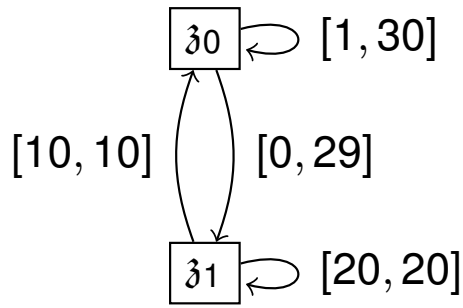
- reduce state space

Idea

- reduce state space
- do not store all individual rates

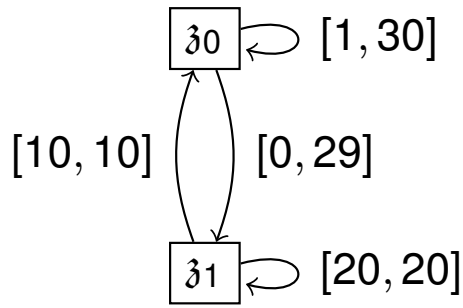
Idea

- reduce state space
- do not store all individual rates
- $\Rightarrow$ **abstract continuous-time Markov chains (ACTMCs)** (by Klink et al.) to abstract CTMCs, special case of continuous-time Markov decision processes (CTMDPs)



Idea

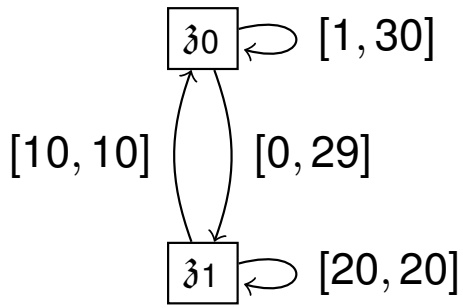
- reduce state space
- do not store all individual rates
- $\Rightarrow$ **abstract continuous-time Markov chains (ACTMCs)** (by Klink et al.) to abstract CTMCs, special case of continuous-time Markov decision processes (CTMDPs)



- good:
  - have algorithm for time-bounded reachability
  - have initial works for CTMC abstraction

Idea

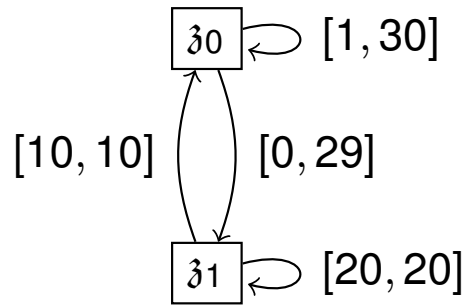
- reduce state space
- do not store all individual rates
- $\Rightarrow$ **abstract continuous-time Markov chains (ACTMCs)** (by Klink et al.) to abstract CTMCs, special case of continuous-time Markov decision processes (CTMDPs)



- good:
  - have algorithm for time-bounded reachability
  - have initial works for CTMC abstraction
- however:

Idea

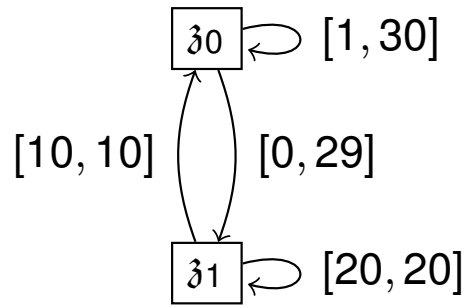
- reduce state space
- do not store all individual rates
- $\Rightarrow$ **abstract continuous-time Markov chains (ACTMCs)** (by Klink et al.) to abstract CTMCs, special case of continuous-time Markov decision processes (CTMDPs)



- good:
  - have algorithm for time-bounded reachability
  - have initial works for CTMC abstraction
- however:
  - (A) want to handle other properties than just reachability e.g. also final and cumulative rewards

Idea

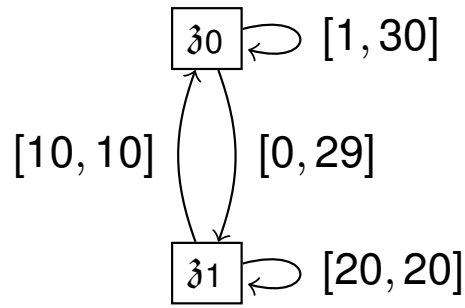
- reduce state space
- do not store all individual rates
- $\Rightarrow$ **abstract continuous-time Markov chains (ACTMCs)** (by Klink et al.) to abstract CTMCs, special case of continuous-time Markov decision processes (CTMDPs)



- good:
  - have algorithm for time-bounded reachability
  - have initial works for CTMC abstraction
- however:
  - (A) want to handle other properties than just reachability e.g. also final and cumulative rewards
  - (B) unclear how build abstractions of CTMCs efficiently

Idea

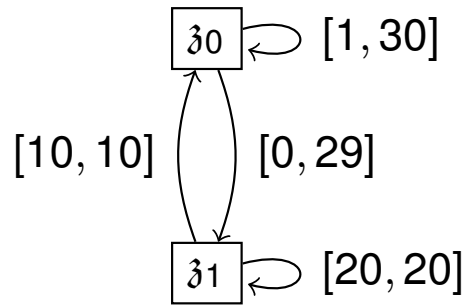
- reduce state space
- do not store all individual rates
- $\Rightarrow$ **abstract continuous-time Markov chains (ACTMCs)** (by Klink et al.) to abstract CTMCs, special case of continuous-time Markov decision processes (CTMDPs)



- good:
  - have algorithm for time-bounded reachability
  - have initial works for CTMC abstraction
- however:
  - (A) want to handle other properties than just reachability e.g. also final and cumulative rewards
  - (B) unclear how build abstractions of CTMCs efficiently
    - need avoid constructing explicit model first

Idea

- reduce state space
- do not store all individual rates
- $\Rightarrow$ **abstract continuous-time Markov chains (ACTMCs)** (by Klink et al.) to abstract CTMCs, special case of continuous-time Markov decision processes (CTMDPs)



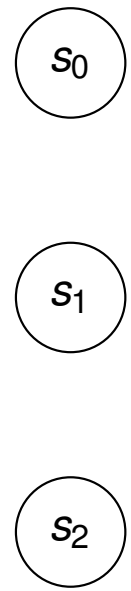
- good:
  - have algorithm for time-bounded reachability
  - have initial works for CTMC abstraction
- however:
  - (A) want to handle other properties than just reachability e.g. also final and cumulative rewards
  - (B) unclear how build abstractions of CTMCs efficiently
    - need avoid constructing explicit model first
    - which states to subsume?

(A) Properties: CTMCs

- CTMC $\mathcal{C}$

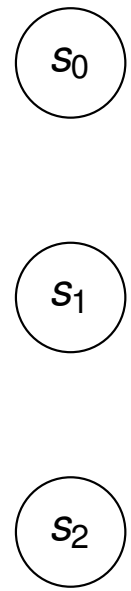
(A) Properties: CTMCs

- CTMC $\mathcal{C}$
  - S: set of **states**



(A) Properties: CTMCs

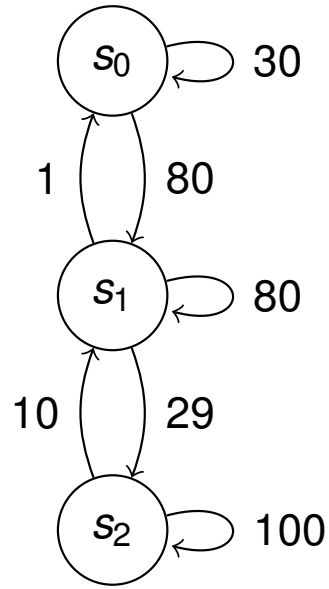
- CTMC $\mathcal{C}$
  - S: set of **states**
  - $\mathbf{u} > 0$: **uniformisation rate**



$\mathbf{u} = 110$

(A) Properties: CTMCs

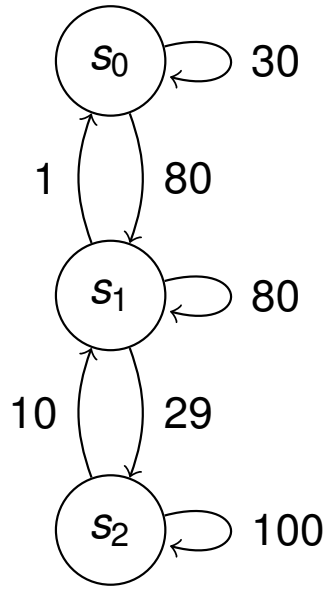
- CTMC $\mathcal{C}$
  - S: set of **states**
  - $\mathbf{u} > 0$: **uniformisation rate**
  - $\mathbf{R}\colon (S \times S) \to [0, \mathbf{u}]$: **rate matrix**, for all $s \in S$ have $\sum_{s'} \mathbf{R}(s, s') = \mathbf{u}$



$\mathbf{u} = 110$

(A) Properties: CTMCs

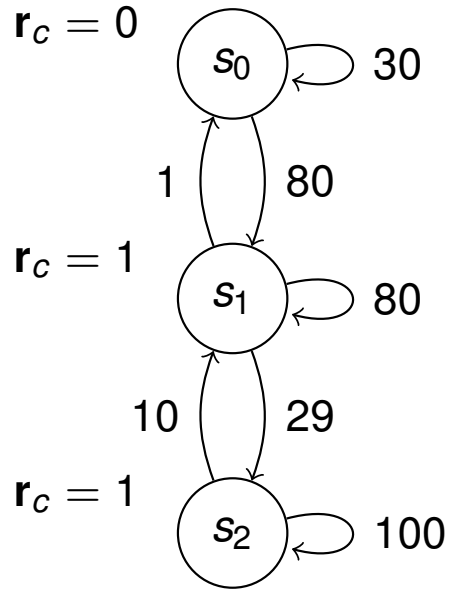
- CTMC $\mathcal{C}$
  - S: set of **states**
  - $\mathbf{u} > 0$: **uniformisation rate**
  - $\mathbf{R}\colon (S \times S) \to [0, \mathbf{u}]$: **rate matrix**, for all $s \in S$ have $\sum_{s'} \mathbf{R}(s, s') = \mathbf{u}$
- reward structure **r**



$\mathbf{u} = 110$

(A) Properties: CTMCs

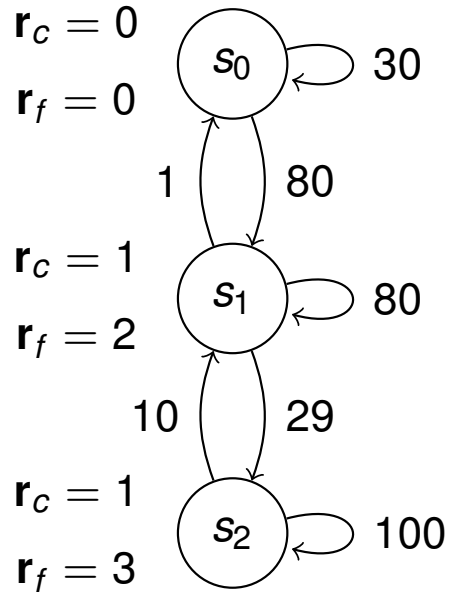
- CTMC $\mathcal{C}$
  - S: set of **states**
  - $\mathbf{u} > 0$: **uniformisation rate**
  - $\mathbf{R}\colon (S \times S) \to [0, \mathbf{u}]$: **rate matrix**, for all $s \in S$ have $\sum_{s'} \mathbf{R}(s, s') = \mathbf{u}$
- reward structure $\mathbf{r}$
  - $\mathbf{r}_c\colon S \to \mathbb{R}$: **cumulative rewards**



$\mathbf{u} = 110$

(A) Properties: CTMCs

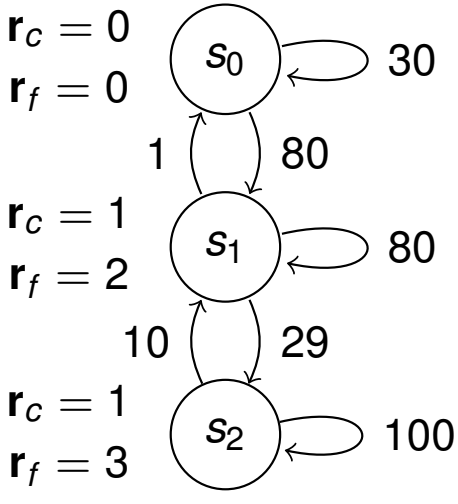
- CTMC $\mathcal{C}$
  - S: set of **states**
  - $\mathbf{u} > 0$: **uniformisation rate**
  - $\mathbf{R}\colon (S \times S) \to [0, \mathbf{u}]$: **rate matrix**, for all $s \in S$ have $\sum_{s'} \mathbf{R}(s, s') = \mathbf{u}$
- reward structure $\mathbf{r}$
  - $\mathbf{r}_c\colon S \to \mathbb{R}$: **cumulative rewards**
  - $\mathbf{r}_f\colon S \to \mathbb{R}$: **final rewards**

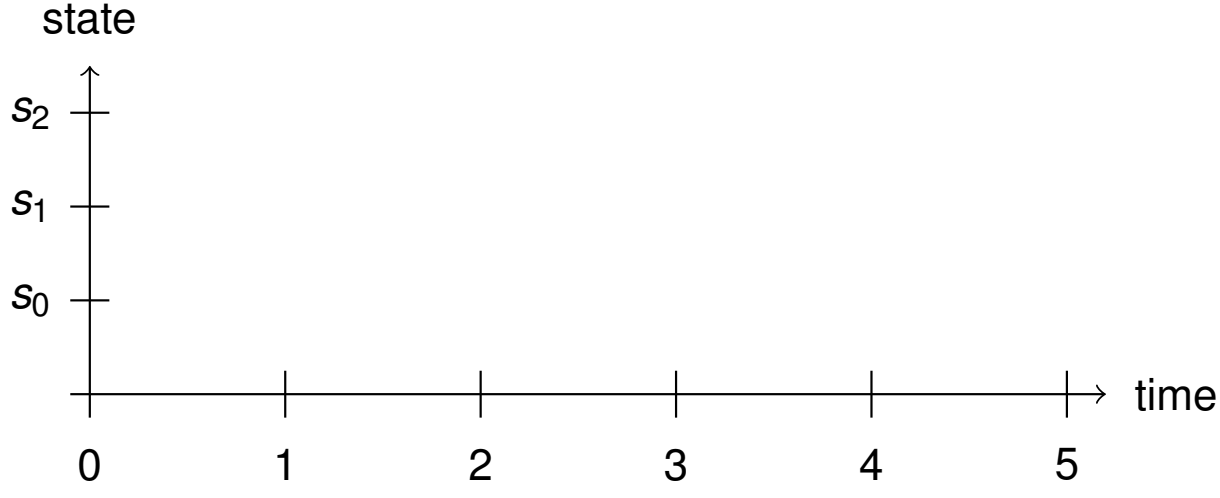


$\mathbf{u} = 110$

(A) Properties: Behaviour and Value

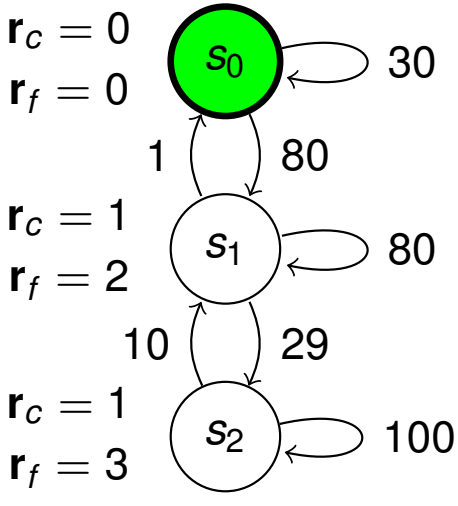
- model behaviour X_t

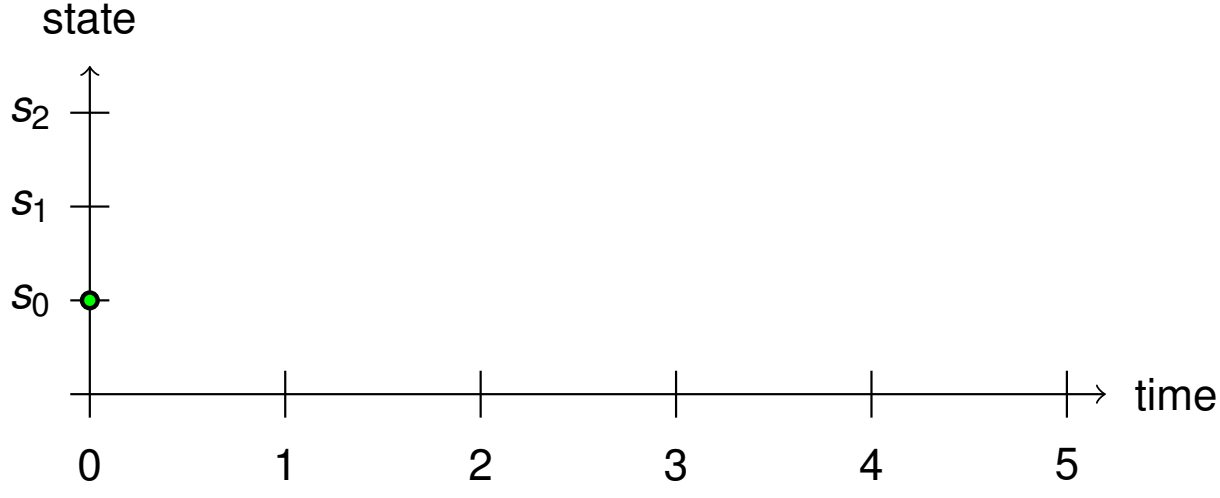




(A) Properties: Behaviour and Value

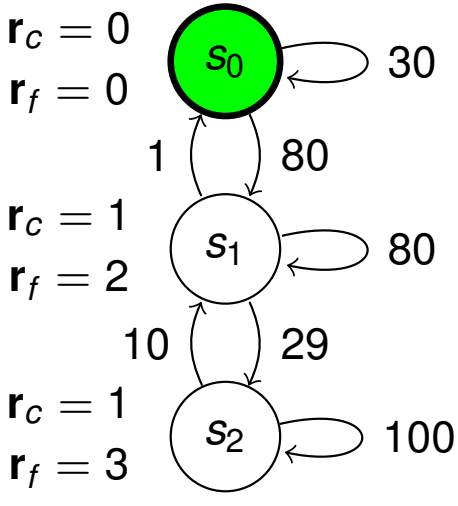
- model behaviour X_t
  - start in given s_0

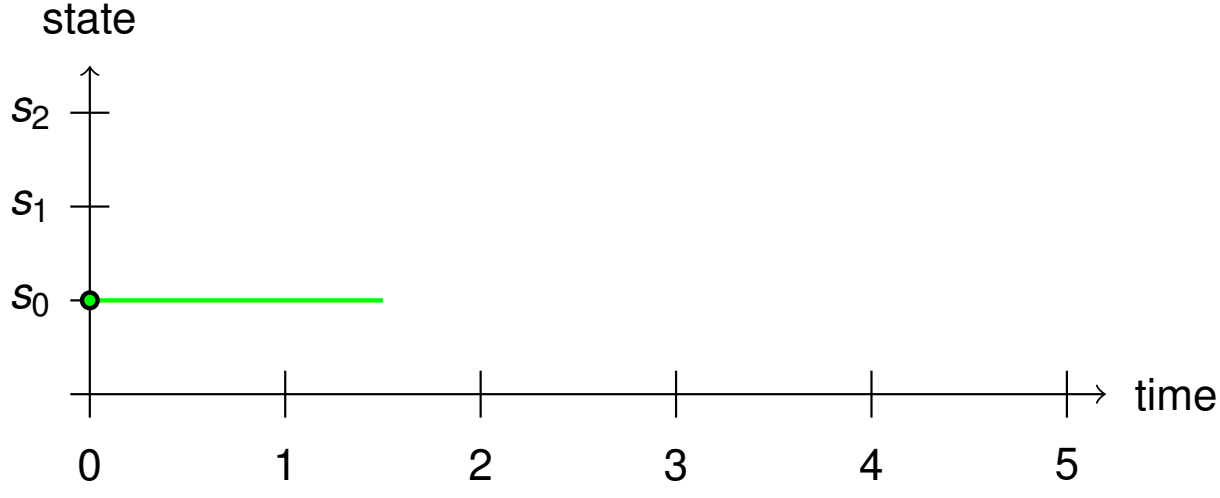




(A) Properties: Behaviour and Value

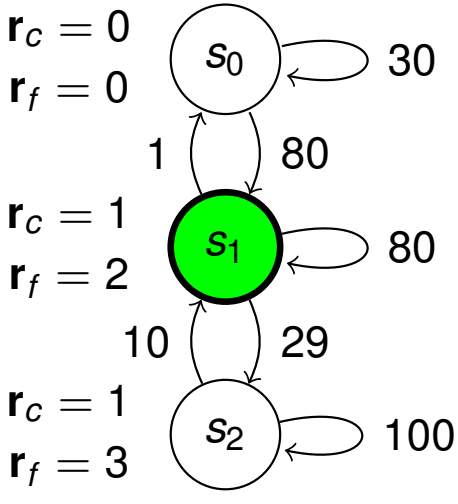
- model behaviour X_t
  - start in given s_0
  - wait for t_0 chosen by exponential distribution with param. **u**

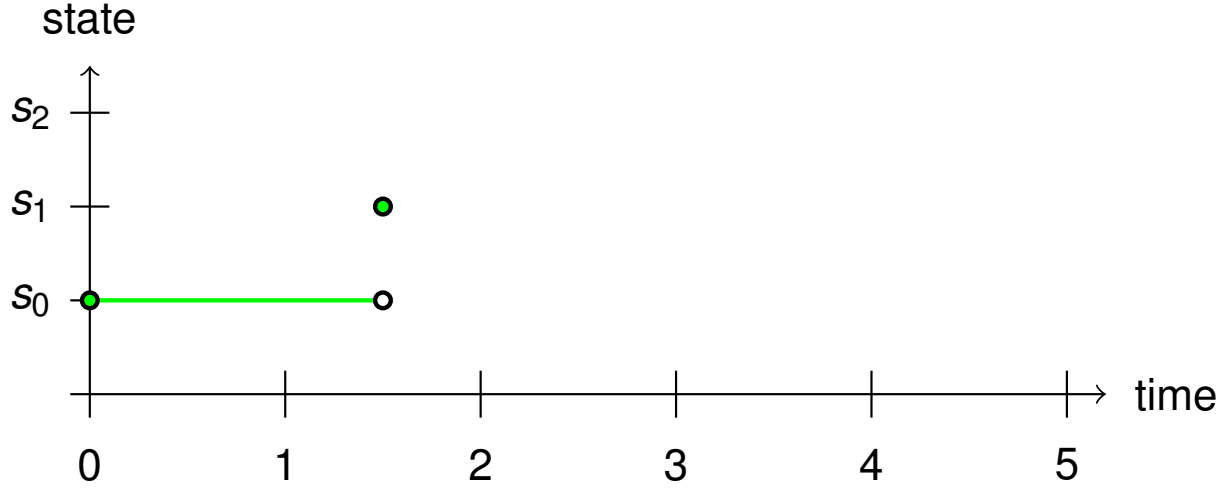




(A) Properties: Behaviour and Value

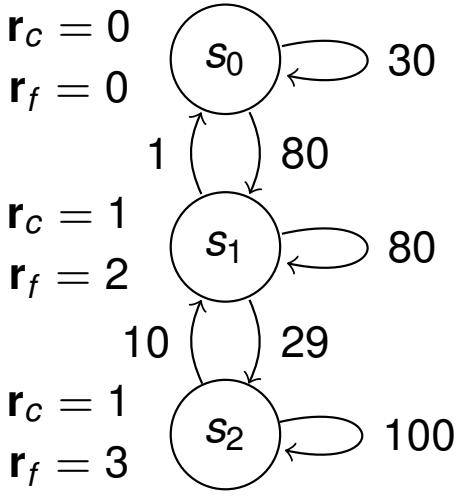
- model behaviour X_t
  - start in given s_0
  - wait for t_0 chosen by exponential distribution with param. $\mathbf{u}$
  - move to s_1 with prob. $\mathbf{R}(s_0, s_1)/\mathbf{u}$

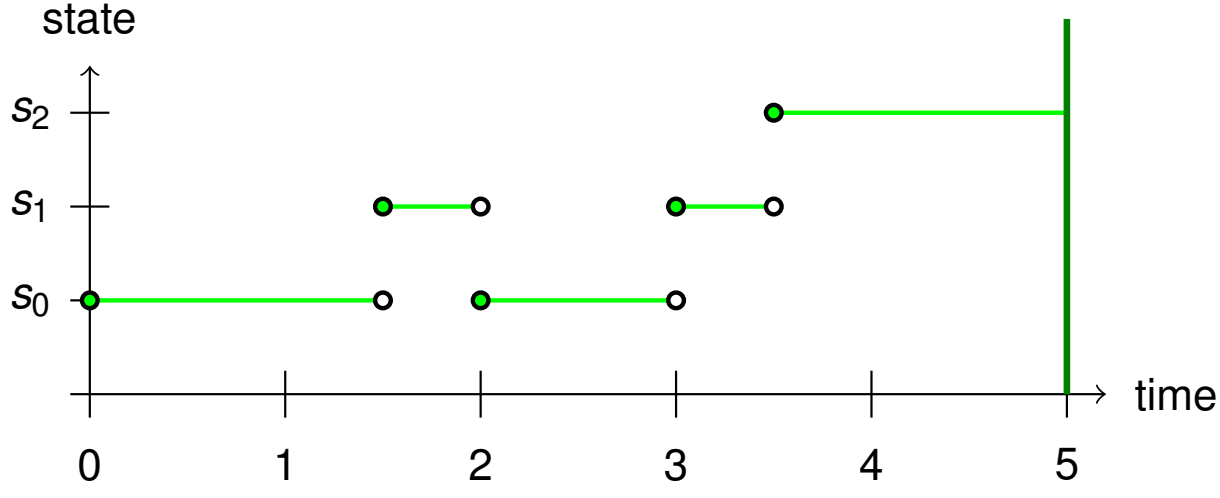




(A) Properties: Behaviour and Value

- model behaviour X_t
  - start in given s_0
  - wait for t_0 chosen by exponential distribution with param. **u**
  - move to s_1 with prob. $\mathbf{R}(s_0, s_1)/\mathbf{u}$
  - until some given $\mathbf{t} > 0$

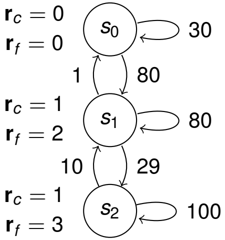


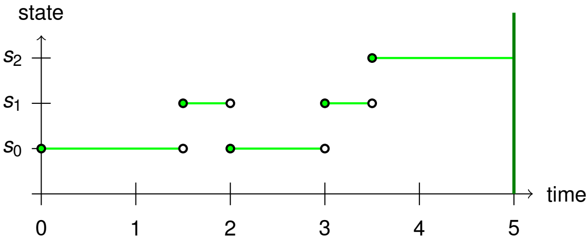


(A) Properties: Behaviour and Value

- model behaviour X_t
  - start in given s_0
  - wait for t_0 chosen by exponential distribution with param. $\mathbf{u}$
  - move to s_1 with prob. $\mathbf{R}(s_0, s_1)/\mathbf{u}$
  - until some given $\mathbf{t} > 0$
- problem: compute **value**

$$\underbrace{\mathbf{E}}_{\text{over paths}} \left[\qquad\qquad\qquad\qquad \right]$$

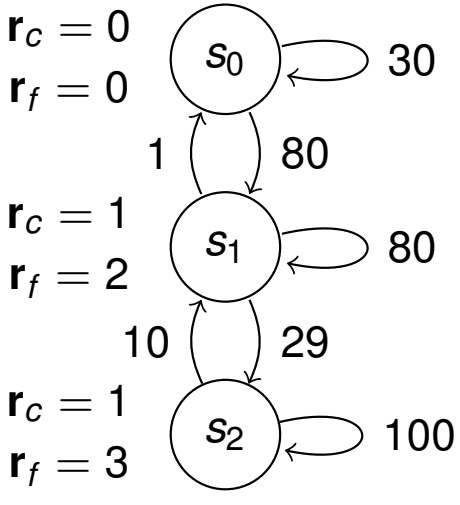


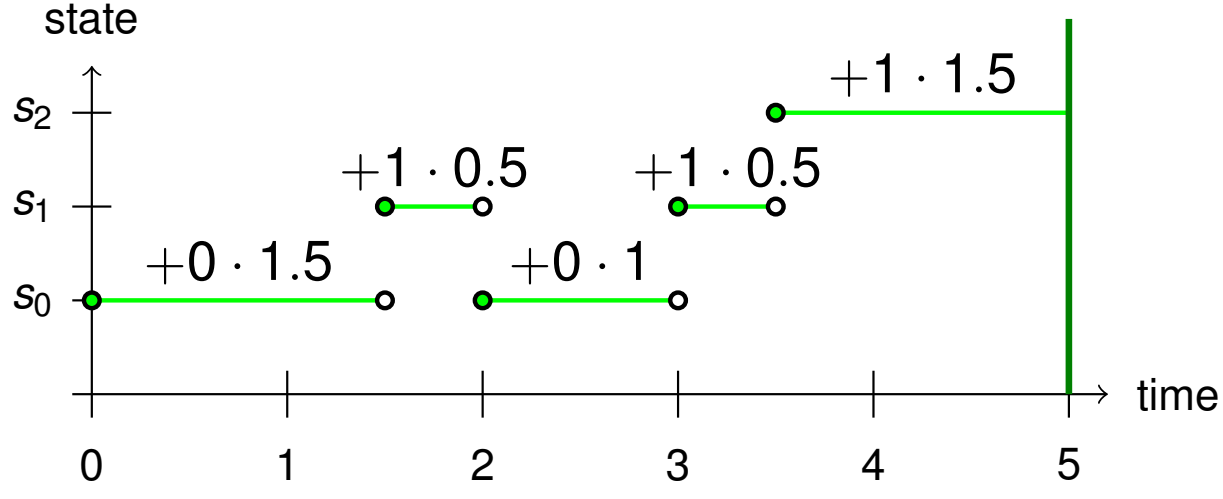


(A) Properties: Behaviour and Value

- model behaviour X_t
  - start in given s_0
  - wait for t_0 chosen by exponential distribution with param. **u**
  - move to s_1 with prob. $\mathbf{R}(s_0, s_1)/\mathbf{u}$
  - until some given $\mathbf{t} > 0$
- problem: compute **value**

$$\underbrace{\mathbf{E}}_{\text{over paths}} \left[\underbrace{\int_0^{\mathbf{t}} \mathbf{r}_c(X_u)\,\mathrm{d}u}_{\text{accumulated until } \mathbf{t}} \right]$$

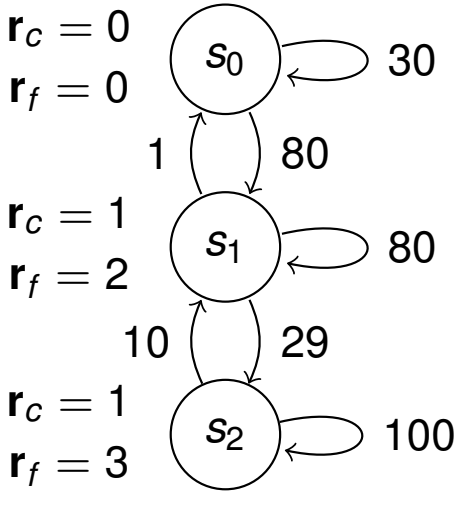


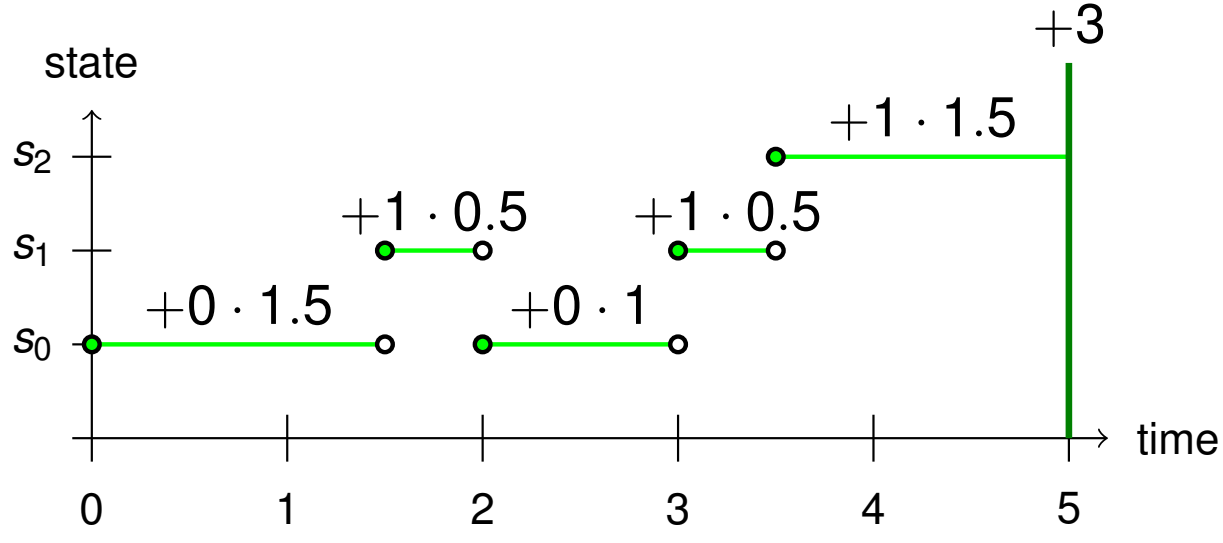


(A) Properties: Behaviour and Value

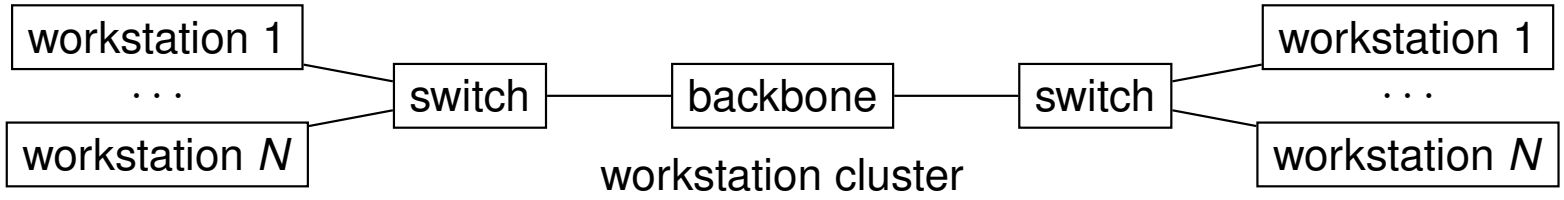
- model behaviour X_t
  - start in given s_0
  - wait for t_0 chosen by exponential distribution with param. **u**
  - move to s_1 with prob. $\mathbf{R}(s_0, s_1)/\mathbf{u}$
  - until some given $\mathbf{t} > 0$
- problem: compute **value**

$$\underbrace{\mathbf{E}}_{\text{over paths}} \left[\underbrace{\int_0^{\mathbf{t}} \mathbf{r}_c(X_u)\,\mathrm{d}u}_{\text{accumulated until } \mathbf{t}} + \underbrace{\mathbf{r}_f(X_{\mathbf{t}})}_{\text{obtained at } \mathbf{t}} \right]$$

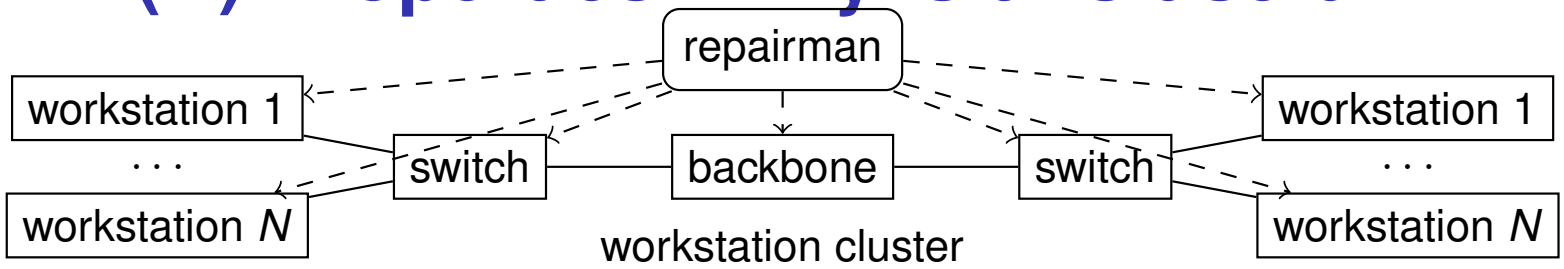




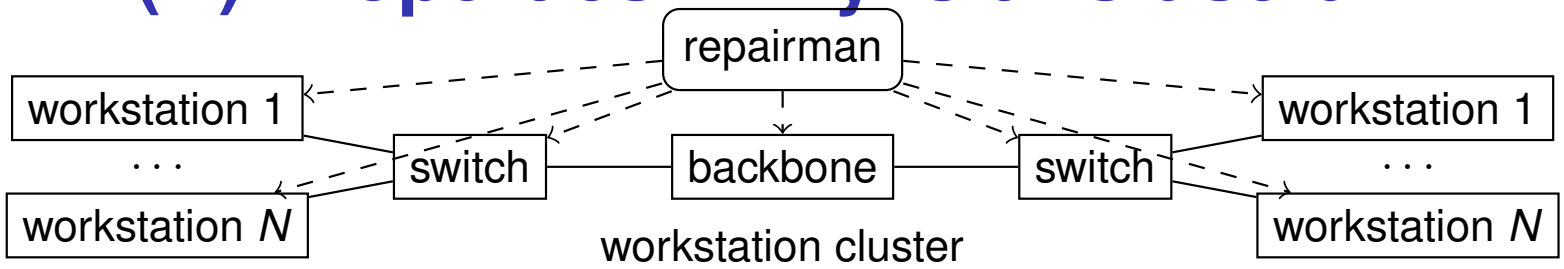
(A) Properties: Why is this useful?



(A) Properties: Why is this useful?

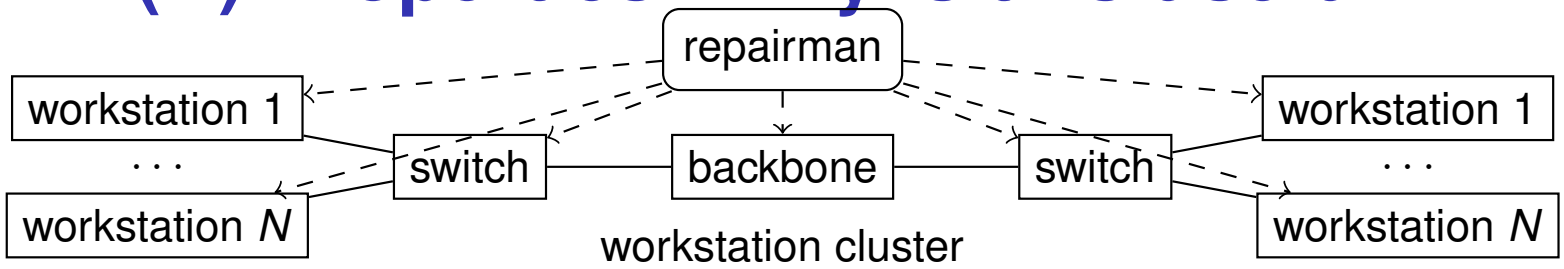


(A) Properties: Why is this useful?



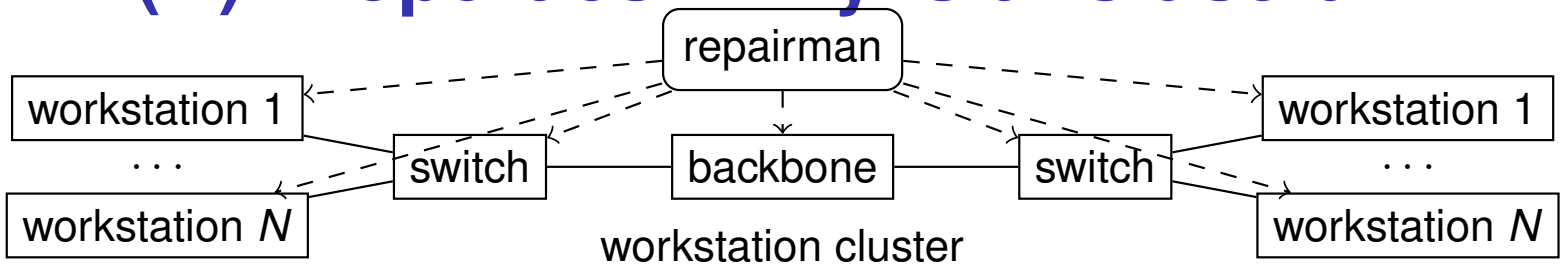
- cumulative reward
  - #repairs until **t**

(A) Properties: Why is this useful?



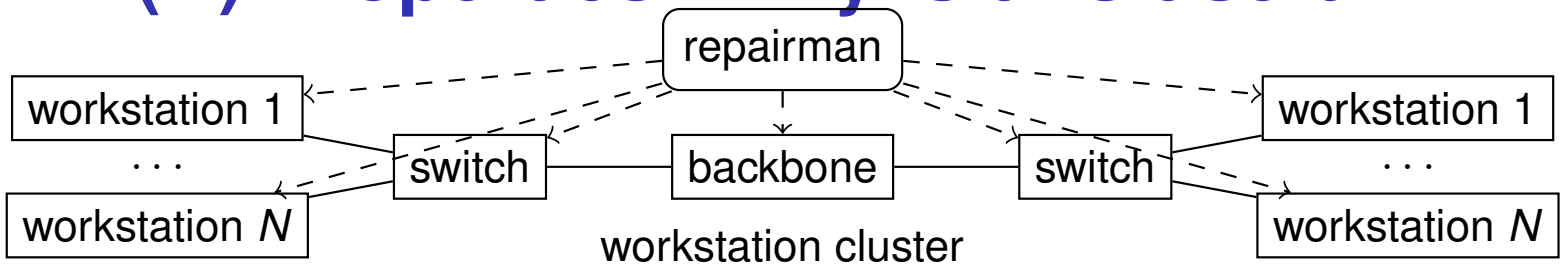
- cumulative reward
  - #repairs until **t**
- final reward
  - #workstations up and running at **t**

(A) Properties: Why is this useful?



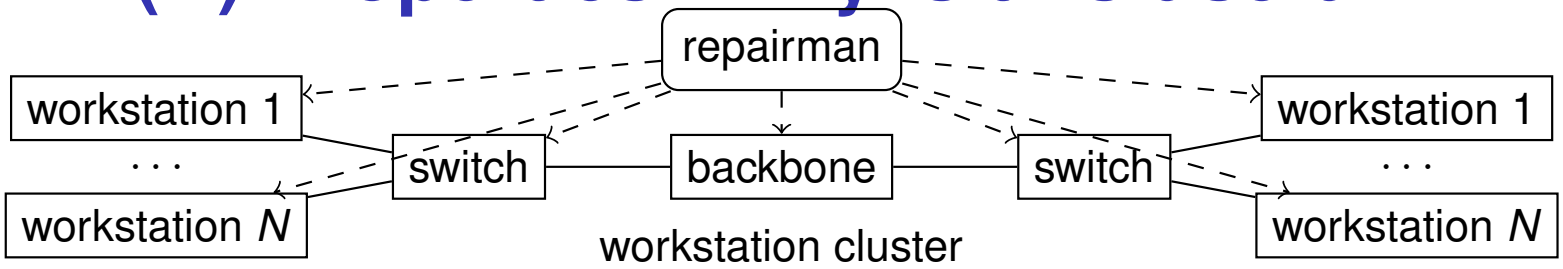
- cumulative reward
  - #repairs until **t**
- final reward
  - #workstations up and running at **t**
- allows to express CSL until properties
  - probability of failure until **t**

(A) Properties: Why is this useful?

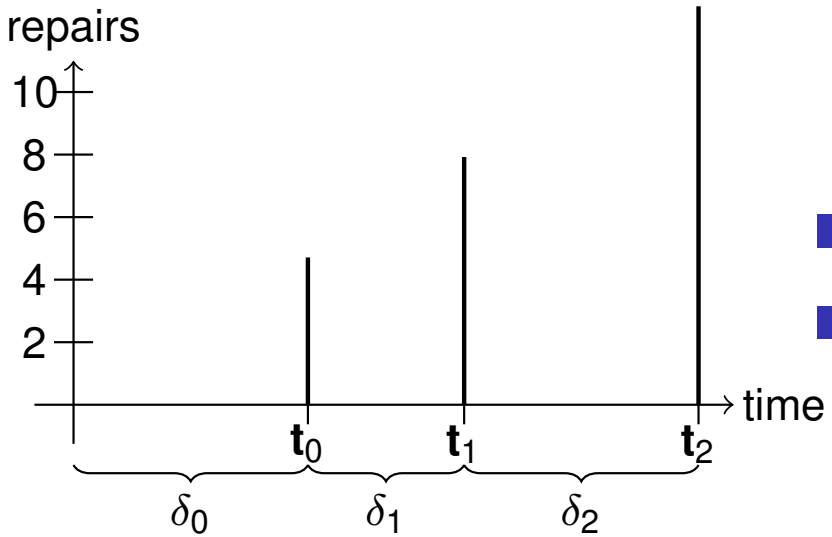


- cumulative reward
  - #repairs until **t**
- final reward
  - #workstations up and running at **t**
- allows to express CSL until properties
  - probability of failure until **t**
- combined cumulative and final:
  efficient computation for multiple time points

(A) Properties: Why is this useful?



- cumulative reward
  - #repairs until **t**
- final reward
  - #workstations up and running at **t**
- allows to express CSL until properties
  - probability of failure until **t**
- combined cumulative and final:
  efficient computation for multiple time points



- analyses with δ_i instead of $\mathbf{t}_i$
- more efficient

(A) Properties: Solution method (1/3)

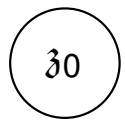
- abstract CTMC to ACTMC
  (special CTMDP, see later in (B))

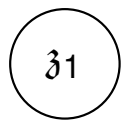
(A) Properties: Solution method (1/3)

- abstract CTMC to ACTMC
  (special CTMDP, see later in (B))
- CTMDP $\mathcal{C}$

(A) Properties: Solution method (1/3)

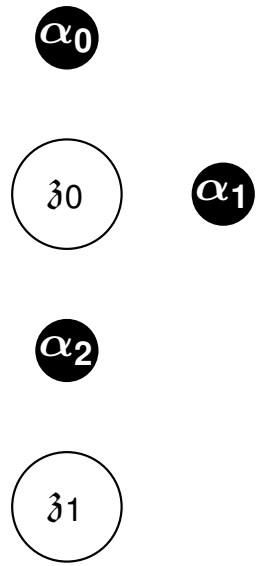
- abstract CTMC to ACTMC
  (special CTMDP, see later in (B))
- CTMDP $\mathcal{C}$
  - $S, \mathbf{u}, \mathbf{r}$: as before





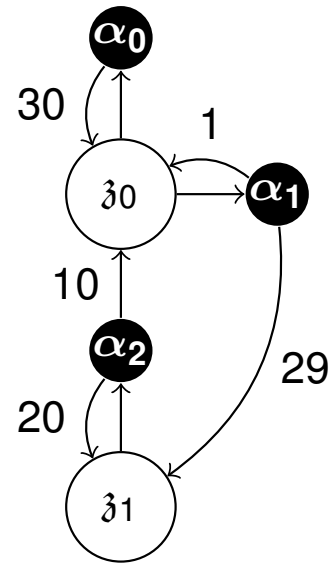
(A) Properties: Solution method (1/3)

- abstract CTMC to ACTMC
  (special CTMDP, see later in (B))
- CTMDP $\mathcal{C}$
  - $S, \mathbf{u}, \mathbf{r}$: as before
  - *Act*: set of **actions**



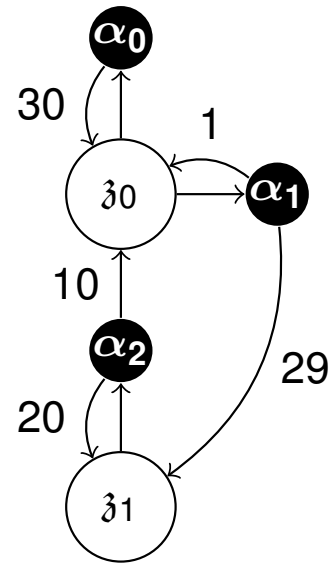
(A) Properties: Solution method (1/3)

- abstract CTMC to ACTMC
  (special CTMDP, see later in (B))
- CTMDP $\mathcal{C}$
  - $S, \mathbf{u}, \mathbf{r}$: as before
  - Act: set of **actions**
  - $\mathbf{R}$: $(S \times Act \times S) \to [0, \mathbf{u}]$,
    fa. $s \in S, \alpha \in Act$:
    $\sum_{s'} \mathbf{R}(s, \alpha, s') \in \{0, \mathbf{u}\}$



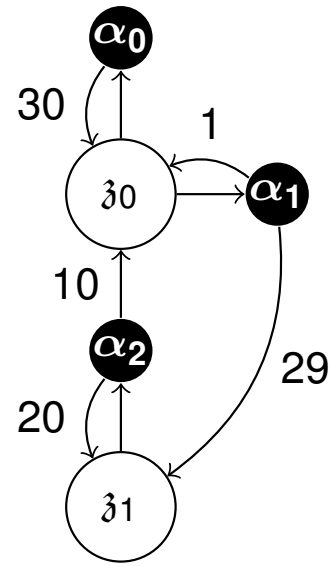
(A) Properties: Solution method (1/3)

- abstract CTMC to ACTMC
  (special CTMDP, see later in (B))
- CTMDP $\mathcal{C}$
  - $S, \mathbf{u}, \mathbf{r}$: as before
  - Act: set of **actions**
  - $\mathbf{R}$: $(S \times Act \times S) \to [0, \mathbf{u}]$,
    fa. $s \in S, \alpha \in Act$:
    $\sum_{s'} \mathbf{R}(s, \alpha, s') \in \{0, \mathbf{u}\}$
- model behaviour X_t^σ



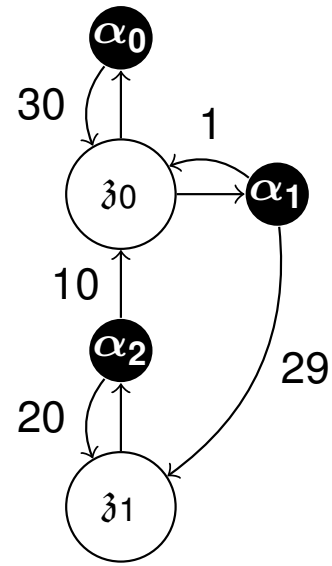
(A) Properties: Solution method (1/3)

- abstract CTMC to ACTMC
  (special CTMDP, see later in (B))
- CTMDP $\mathcal{C}$
  - $S, \mathbf{u}, \mathbf{r}$: as before
  - Act: set of **actions**
  - $\mathbf{R}$: $(S \times Act \times S) \to [0, \mathbf{u}]$,
    fa. $s \in S, \alpha \in Act$:
    $\sum_{s'} \mathbf{R}(s, \alpha, s') \in \{0, \mathbf{u}\}$
- model behaviour X_t^σ
  - σ: **scheduler**, chooses actions



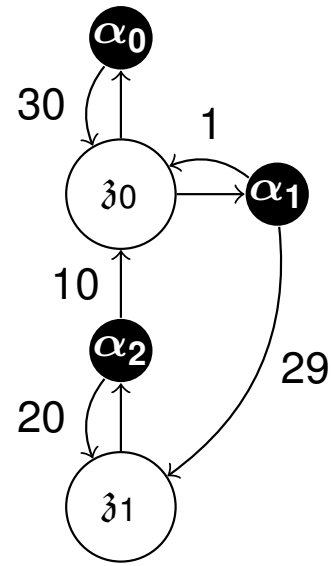
(A) Properties: Solution method (1/3)

- abstract CTMC to ACTMC
  (special CTMDP, see later in (B))
- CTMDP $\mathcal{C}$
  - $S, \mathbf{u}, \mathbf{r}$: as before
  - Act: set of **actions**
  - $\mathbf{R}$: $(S \times Act \times S) \to [0, \mathbf{u}]$,
    fa. $s \in S, \alpha \in Act$:
    $\sum_{s'} \mathbf{R}(s, \alpha, s') \in \{0, \mathbf{u}\}$
- model behaviour X_t^σ
  - σ: **scheduler**, chooses actions
  - this induces a CTMC



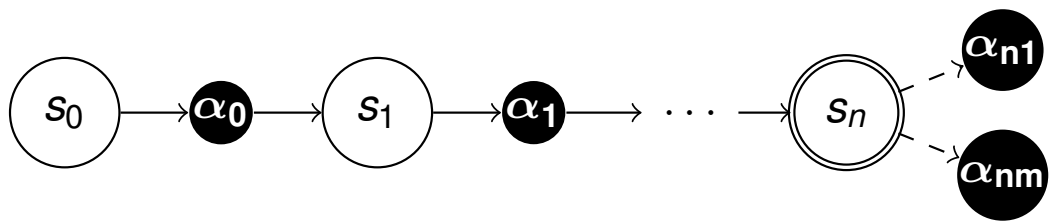
(A) Properties: Solution method (1/3)

- abstract CTMC to ACTMC
  (special CTMDP, see later in (B))
- CTMDP $\mathcal{C}$
  - $S, \mathbf{u}, \mathbf{r}$: as before
  - Act: set of **actions**
  - $\mathbf{R}$: $(S \times Act \times S) \to [0, \mathbf{u}]$,
    fa. $s \in S, \alpha \in Act$:
    $\sum_{s'} \mathbf{R}(s, \alpha, s') \in \{0, \mathbf{u}\}$
- model behaviour X_t^σ
  - σ: **scheduler**, chooses actions
  - this induces a CTMC
  - value of CTMDP depends on scheduler



(A) Properties: Solution method (2/3)

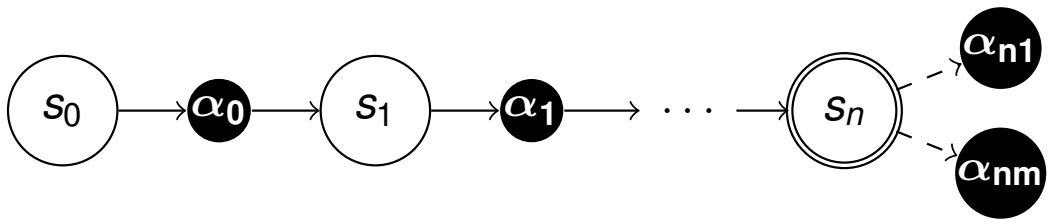
Compute maximal (minimal) value over σ which



(A) Properties: Solution method (2/3)

Compute maximal (minimal) value over σ which

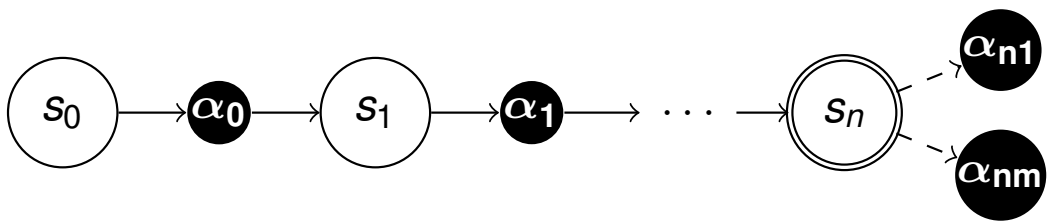
- can apply a probabilistic choice over actions



(A) Properties: Solution method (2/3)

Compute maximal (minimal) value over σ which

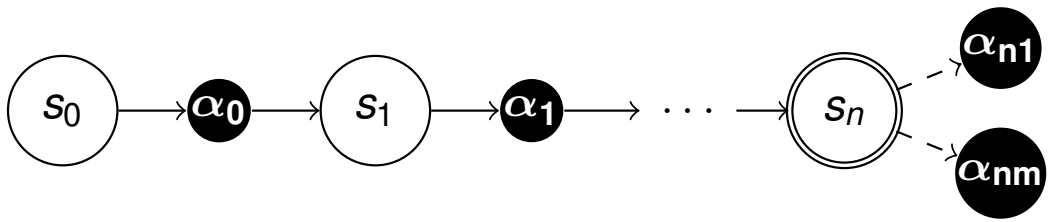
- can apply a probabilistic choice over actions
- know sequence of past actions and states



(A) Properties: Solution method (2/3)

Compute maximal (minimal) value over σ which

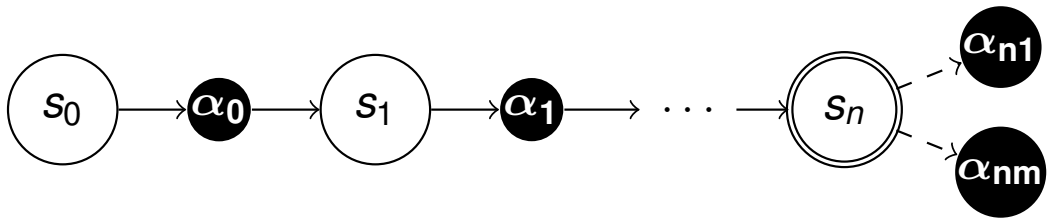
- can apply a probabilistic choice over actions
- know sequence of past actions and states
- does not know exact time points



(A) Properties: Solution method (2/3)

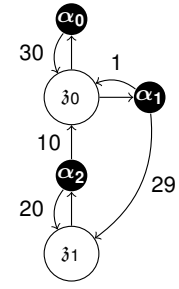
Compute maximal (minimal) value over σ which

- can apply a probabilistic choice over actions
- know sequence of past actions and states
- does not know exact time points
- can not change decision before leaving state



(A) Properties: Solution method (3/3)

Algorithm, based on [1] and [2]:



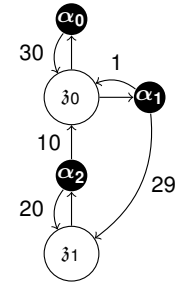
[1] C. Baier, H. Hermanns, J.-P. Katoen, and B. R. Haverkort, “Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes”,
[2] M. Kwiatkowska, G. Norman, and A. Pacheco, “Model checking expected time and expected reward formulae with random time bounds”

(A) Properties: Solution method (3/3)

Algorithm, based on [1] and [2]:

let k large enough, max. number jumps considered

$\phi(i)$ Poisson distr, $\psi(i) = \sum_{j>i} \phi(j)$ mixed Poisson



[1] C. Baier, H. Hermanns, J.-P. Katoen, and B. R. Haverkort, “Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes”,

[2] M. Kwiatkowska, G. Norman, and A. Pacheco, “Model checking expected time and expected reward formulae with random time bounds”

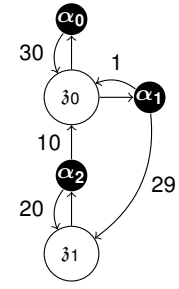
(A) Properties: Solution method (3/3)

Algorithm, based on [1] and [2]:

let k large enough, max. number jumps considered

$\phi(i)$ Poisson distr, $\psi(i) = \sum_{j>i} \phi(j)$ mixed Poisson

$\mathbf{P}(\cdot,\cdot) := \mathbf{R}(\cdot,\cdot)/\mathbf{u}$, $q_{k+1} := 0$



[1] C. Baier, H. Hermanns, J.-P. Katoen, and B. R. Haverkort, “Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes”,
[2] M. Kwiatkowska, G. Norman, and A. Pacheco, “Model checking expected time and expected reward formulae with random time bounds”

(A) Properties: Solution method (3/3)

Algorithm, based on [1] and [2]:

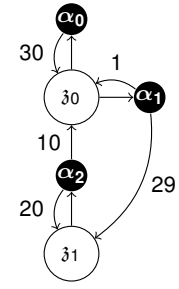
let k large enough, max. number jumps considered

$\phi(i)$ Poisson distr, $\psi(i) = \sum_{j>i} \phi(j)$ mixed Poisson

$\mathbf{P}(\cdot,\cdot) := \mathbf{R}(\cdot,\cdot)/\mathbf{u}$, $q_{k+1} := 0$

forall the $i = k, k-1, \ldots, 0$ **do**

  forall the $s \in S$ **do**



[1] C. Baier, H. Hermanns, J.-P. Katoen, and B. R. Haverkort, “Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes”,

[2] M. Kwiatkowska, G. Norman, and A. Pacheco, “Model checking expected time and expected reward formulae with random time bounds”

(A) Properties: Solution method (3/3)

Algorithm, based on [1] and [2]:

let k large enough, max. number jumps considered

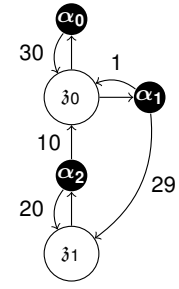
$\phi(i)$ Poisson distr, $\psi(i) = \sum_{j>i} \phi(j)$ mixed Poisson

$\mathbf{P}(\cdot,\cdot) := \mathbf{R}(\cdot,\cdot)/\mathbf{u}$, $q_{k+1} := 0$

forall the $i = k, k-1, \ldots, 0$ **do**

  forall the $s \in S$ **do**

    $q_i(s) := \max_\alpha \sum_{s'} \mathbf{P}(s, \alpha, s') q_{i+1}(s')$



[1] C. Baier, H. Hermanns, J.-P. Katoen, and B. R. Haverkort, “Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes”,

[2] M. Kwiatkowska, G. Norman, and A. Pacheco, “Model checking expected time and expected reward formulae with random time bounds”

(A) Properties: Solution method (3/3)

Algorithm, based on [1] and [2]:

let k large enough, max. number jumps considered

$\phi(i)$ Poisson distr, $\psi(i) = \sum_{j>i} \phi(j)$ mixed Poisson

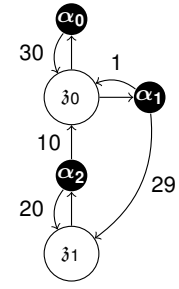
$\mathbf{P}(\cdot,\cdot) := \mathbf{R}(\cdot,\cdot)/\mathbf{u}$, $q_{k+1} := 0$

forall the $i = k, k-1, \ldots, 0$ **do**

  forall the $s \in S$ **do**

    $q_i(s) := \max_\alpha \sum_{s'} \mathbf{P}(s,\alpha,s') q_{i+1}(s')$

    $q_i(s) := q_i(s) \; + \; \phi_{\mathbf{ut}}(i) \cdot \mathbf{r}_f(s) \; + \; \psi_{\mathbf{ut}}(i) \cdot \mathbf{r}_c(s)/\mathbf{u}$



[1] C. Baier, H. Hermanns, J.-P. Katoen, and B. R. Haverkort, “Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes”,

[2] M. Kwiatkowska, G. Norman, and A. Pacheco, “Model checking expected time and expected reward formulae with random time bounds”

(A) Properties: Solution method (3/3)

Algorithm, based on [1] and [2]:

let k large enough, max. number jumps considered

$\phi(i)$ Poisson distr, $\psi(i) = \sum_{j>i} \phi(j)$ mixed Poisson

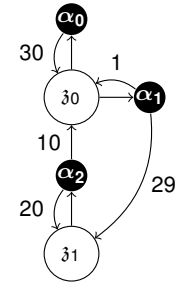
$\mathbf{P}(\cdot,\cdot) := \mathbf{R}(\cdot,\cdot)/\mathbf{u}$, $q_{k+1} := 0$

forall the $i = k, k-1, \ldots, 0$ **do**

  forall the $s \in S$ **do**

    $q_i(s) := \max_\alpha \sum_{s'} \mathbf{P}(s,\alpha,s') q_{i+1}(s')$

    $q_i(s) := q_i(s) \; + \; \phi_{\mathbf{ut}}(i) \cdot \mathbf{r}_f(s) \; + \; \psi_{\mathbf{ut}}(i) \cdot \mathbf{r}_c(s)/\mathbf{u}$



algorithm also shows that for maximising schedulers

[1] C. Baier, H. Hermanns, J.-P. Katoen, and B. R. Haverkort, “Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes”,

[2] M. Kwiatkowska, G. Norman, and A. Pacheco, “Model checking expected time and expected reward formulae with random time bounds”

(A) Properties: Solution method (3/3)

Algorithm, based on [1] and [2]:

let k large enough, max. number jumps considered

$\phi(i)$ Poisson distr, $\psi(i) = \sum_{j>i} \phi(j)$ mixed Poisson

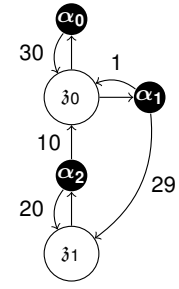
$\mathbf{P}(\cdot,\cdot) := \mathbf{R}(\cdot,\cdot)/\mathbf{u}$, $q_{k+1} := 0$

forall the $i = k, k-1, \ldots, 0$ **do**

  forall the $s \in S$ **do**

    $q_i(s) := \max_\alpha \sum_{s'} \mathbf{P}(s,\alpha,s') q_{i+1}(s')$

    $q_i(s) := q_i(s) \; + \; \phi_{\mathbf{ut}}(i) \cdot \mathbf{r}_f(s) \; + \; \psi_{\mathbf{ut}}(i) \cdot \mathbf{r}_c(s)/\mathbf{u}$



algorithm also shows that for maximising schedulers

- probabilistic choice over actions not needed

[1] C. Baier, H. Hermanns, J.-P. Katoen, and B. R. Haverkort, “Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes”,

[2] M. Kwiatkowska, G. Norman, and A. Pacheco, “Model checking expected time and expected reward formulae with random time bounds”

(A) Properties: Solution method (3/3)

Algorithm, based on [1] and [2]:

let k large enough, max. number jumps considered

$\phi(i)$ Poisson distr, $\psi(i) = \sum_{j>i} \phi(j)$ mixed Poisson

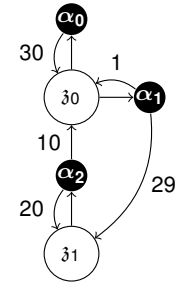
$\mathbf{P}(\cdot,\cdot) := \mathbf{R}(\cdot,\cdot)/\mathbf{u}$, $q_{k+1} := 0$

forall the $i = k, k-1, \ldots, 0$ **do**

  forall the $s \in S$ **do**

    $q_i(s) := \max_\alpha \sum_{s'} \mathbf{P}(s,\alpha,s') q_{i+1}(s')$

    $q_i(s) := q_i(s) \; + \; \phi_{\mathbf{ut}}(i) \cdot \mathbf{r}_f(s) \; + \; \psi_{\mathbf{ut}}(i) \cdot \mathbf{r}_c(s)/\mathbf{u}$



algorithm also shows that for maximising schedulers

- probabilistic choice over actions not needed
- history not needed, knowing step number suffices

[1] C. Baier, H. Hermanns, J.-P. Katoen, and B. R. Haverkort, “Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes”,
[2] M. Kwiatkowska, G. Norman, and A. Pacheco, “Model checking expected time and expected reward formulae with random time bounds”

(B) Abstraction: ACTMCs

- ACTMC $\mathcal{C}$

(B) Abstraction: ACTMCs

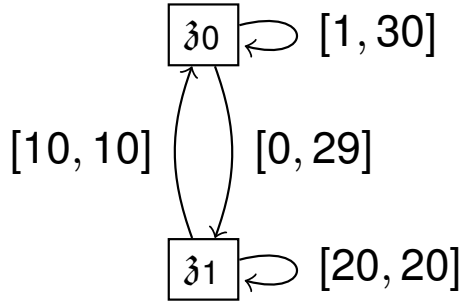
- ACTMC $\mathcal{C}$
  - $S, \mathbf{u}, \mathbf{r}$: as for CTMCs and CTMDPs

$\mathfrak{Z}0$

$\mathfrak{Z}1$

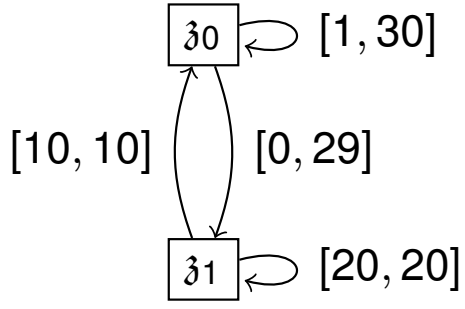
(B) Abstraction: ACTMCs

- ACTMC $\mathcal{C}$
  - $S, \mathbf{u}, \mathbf{r}$: as for CTMCs and CTMDPs
  - $\mathbf{l}^{\ell}, \mathbf{l}^{\mathbf{u}} : (S \times S) \to [0, \mathbf{u}]$



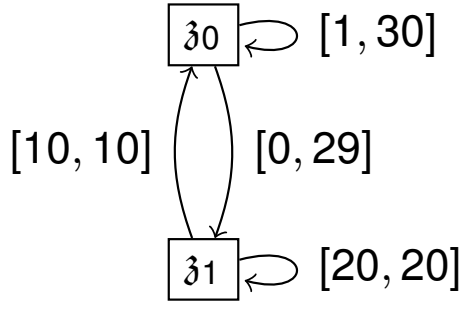
(B) Abstraction: ACTMCs

- ACTMC $\mathcal{C}$
  - $S, \mathbf{u}, \mathbf{r}$: as for CTMCs and CTMDPs
  - $\mathbf{l}^{\ell}, \mathbf{l}^{\mathbf{u}} : (S \times S) \to [0, \mathbf{u}]$
- special case of CTMDP



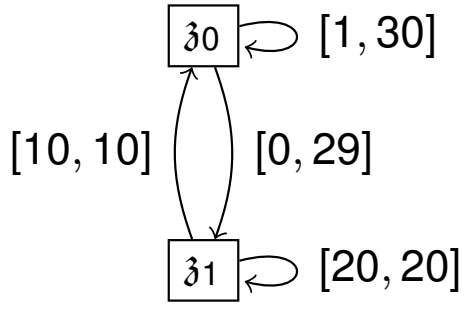
(B) Abstraction: ACTMCs

- ACTMC $\mathcal{C}$
  - $S, \mathbf{u}, \mathbf{r}$: as for CTMCs and CTMDPs
  - $\mathbf{l}^{\ell}, \mathbf{l}^{\mathbf{u}} : (S \times S) \to [0, \mathbf{u}]$
- special case of CTMDP
  - $\mathbf{l}^{\ell}, \mathbf{l}^{\mathbf{u}}$ induce *Act* and $\mathbf{R}$
    choose rates within intervals,
    with their sum being $\mathbf{u}$



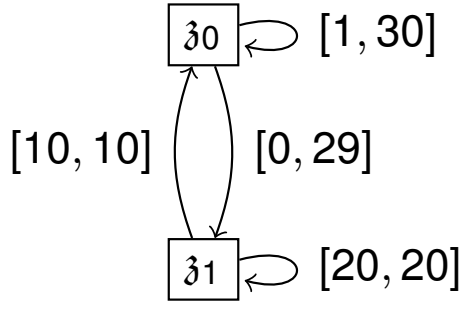
(B) Abstraction: ACTMCs

- ACTMC $\mathcal{C}$
  - $S, \mathbf{u}, \mathbf{r}$: as for CTMCs and CTMDPs
  - $\mathbf{l}^{\ell}, \mathbf{l}^{\mathbf{u}} : (S \times S) \to [0, \mathbf{u}]$
- special case of CTMDP
  - $\mathbf{l}^{\ell}, \mathbf{l}^{\mathbf{u}}$ induce *Act* and $\mathbf{R}$
    choose rates within intervals,
    with their sum being $\mathbf{u}$
  - $|Act| = \infty$



(B) Abstraction: ACTMCs

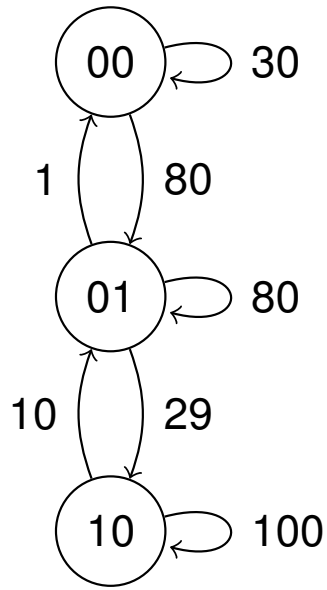
- ACTMC $\mathcal{C}$
  - $S, \mathbf{u}, \mathbf{r}$: as for CTMCs and CTMDPs
  - $\mathbf{l}^{\ell}, \mathbf{l}^{\mathbf{u}} \colon (S \times S) \to [0, \mathbf{u}]$
- special case of CTMDP
  - $\mathbf{l}^{\ell}, \mathbf{l}^{\mathbf{u}}$ induce *Act* and $\mathbf{R}$
    choose rates within intervals,
    with their sum being $\mathbf{u}$
  - $|Act| = \infty$
  - but $\max_{\alpha} \ldots$ still computable efficiently



(B) Abstraction: State partitioning

- start with `PRISM` (guarded-command like) description

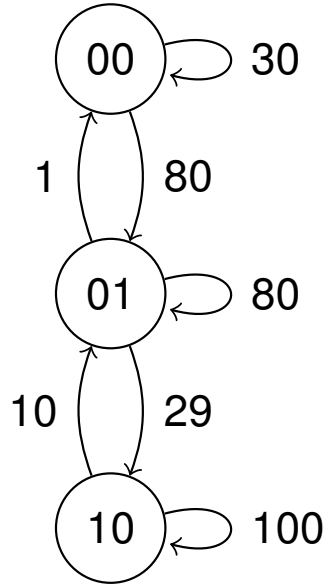
```
x0,x1 : bool init f;
...
[e]x0&!x1->29:x0'=f&x1'=f;
...
```



(B) Abstraction: State partitioning

- start with `PRISM` (guarded-command like) description
- remove rates

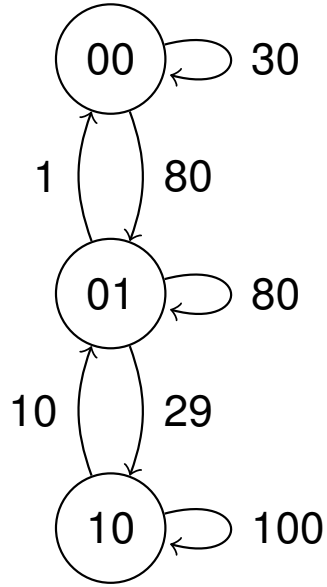
```
x0,x1 : bool init f;
...
[e]x0&!x1->x0'=f&x1'=t;
...
```

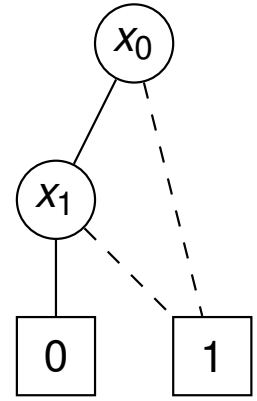


(B) Abstraction: State partitioning

- start with PRISM (guarded-command like) description
- remove rates
- compute reachable states as BDD (e.g. NuSMV)

```
x0,x1 : bool init f;
...
[e]x0&!x1->x0'=f&x1'=t;
...
```

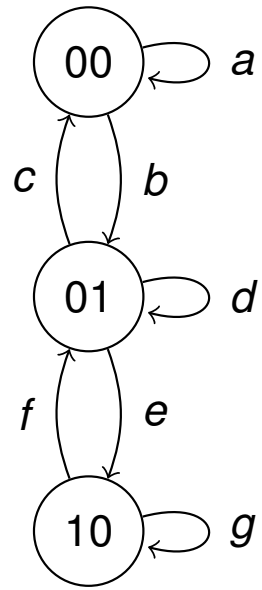


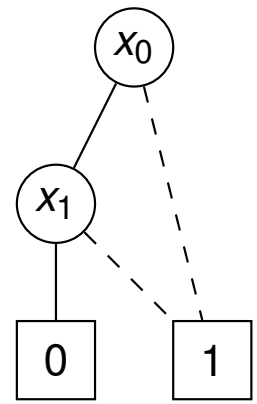


(B) Abstraction: State partitioning

- start with `PRISM` (guarded-command like) description
- remove rates
- compute reachable states as BDD (e.g. `NuSMV`)
- consider commands as labels *a b*, etc. of labelled transition system

```
x0,x1 : bool init f;
...
[e]x0&!x1->x0'=f&x1'=t;
...
```

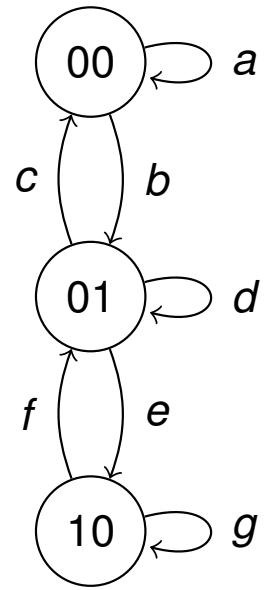


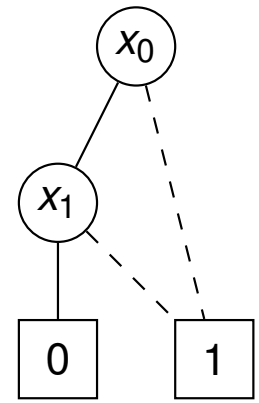


(B) Abstraction: State partitioning

- start with `PRISM` (guarded-command like) description
- remove rates
- compute reachable states as BDD (e.g. `NuSMV`)
- consider commands as labels *a b*, etc. of labelled transition system
- partitioning using symbolic `sig`nature `ref`inement

```
x0,x1 : bool init f;
...
[e]x0&!x1->x0'=f&x1'=t;
...
```

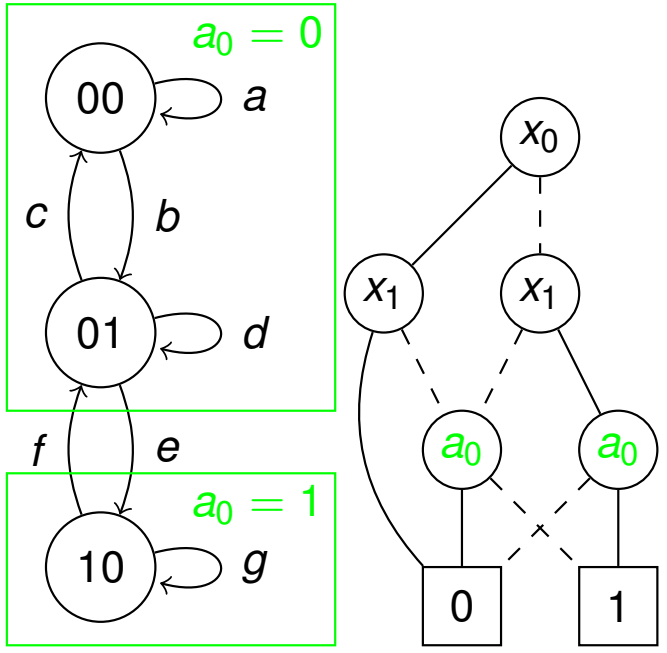




(B) Abstraction: State partitioning

- start with `PRISM` (guarded-command like) description
- remove rates
- compute reachable states as BDD (e.g. `NuSMV`)
- consider commands as labels *a b*, etc. of labelled transition system
- partitioning using symbolic `sig`nature `ref`inement
- by introducing additional BDD vars

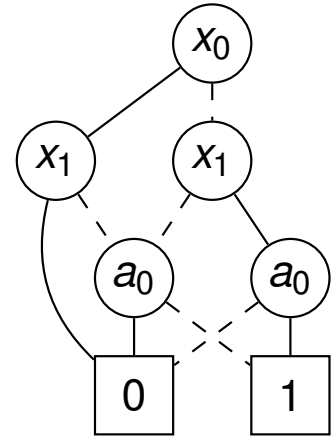
```
x0,x1 : bool init f;
...
[e]x0&!x1->x0'=f&x1'=t;
...
```



(B) Abstraction: Building ACTMCs

- prepare states of ACTMC

state partitioning:



PRISM model:

```
x0&!x1->
 29:x0'=f&x1'=t;
```

ACTMC to build:

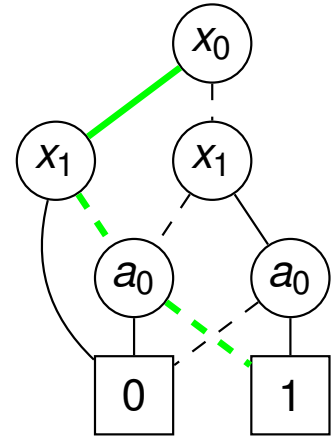
$a_0 = 0$

$a_0 = 1$

(B) Abstraction: Building ACTMCs

- prepare states of ACTMC
- consider each valid variable valuation

state partitioning:



`PRISM` model:

```
x0&!x1->
 29:x0'=f&x1'=t;
```

ACTMC to build:

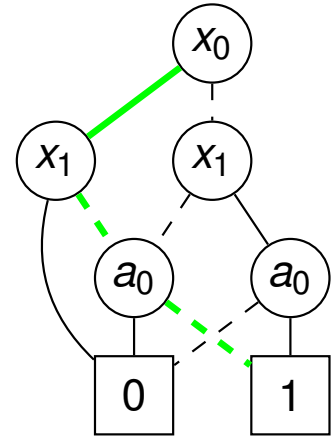
$a_0 = 0$

$a_0 = 1$

(B) Abstraction: Building ACTMCs

- prepare states of ACTMC
- consider each valid variable valuation
- get abstract state

state partitioning:



`PRISM` model:

```
x0&!x1->
 29:x0'=f&x1'=t;
```

ACTMC to build:

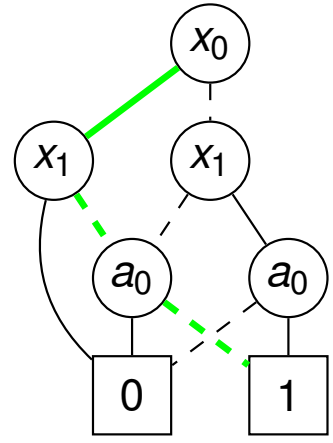
$a_0 = 0$

$a_0 = 1$

(B) Abstraction: Building ACTMCs

- prepare states of ACTMC
- consider each valid variable valuation
- get abstract state
- get contained concrete state

state partitioning:



PRISM model:

```
x0&!x1->
 29:x0'=f&x1'=t;
```

ACTMC to build:

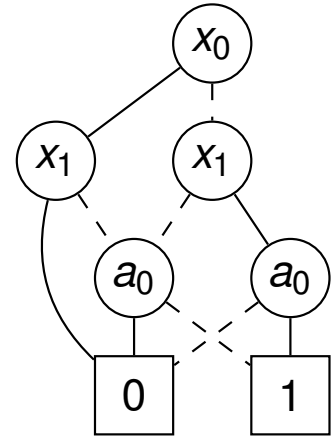
$a_0 = 0$

$a_0 = 1$

(B) Abstraction: Building ACTMCs

- prepare states of ACTMC
- consider each valid variable valuation
- get abstract state
- get contained concrete state
- `PRISM` model to get successor rates and states

state partitioning:



`PRISM` model:

```
x0&!x1->
 29:x0'=f&x1'=t;
```

ACTMC to build:

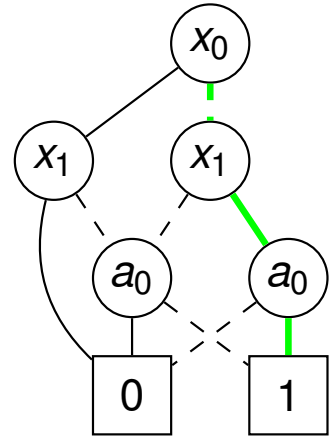
$a_0 = 0$

$a_0 = 1$

(B) Abstraction: Building ACTMCs

- prepare states of ACTMC
- consider each valid variable valuation
- get abstract state
- get contained concrete state
- `PRISM` model to get successor rates and states
- partitioning to find containing abstract states

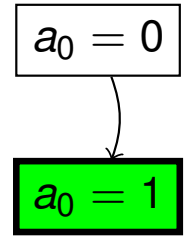
state partitioning:



`PRISM` model:

```
x0&!x1->
 29:x0'=f&x1'=t;
```

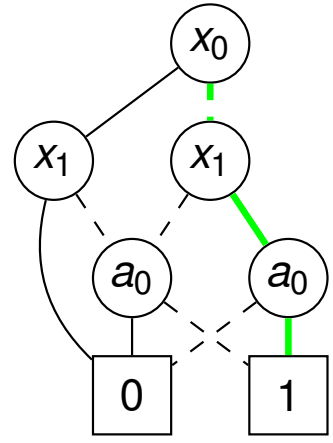
ACTMC to build:



(B) Abstraction: Building ACTMCs

- prepare states of ACTMC
- consider each valid variable valuation
- get abstract state
- get contained concrete state
- PRISM model to get successor rates and states
- partitioning to find containing abstract states
- widen rates of ACTMC

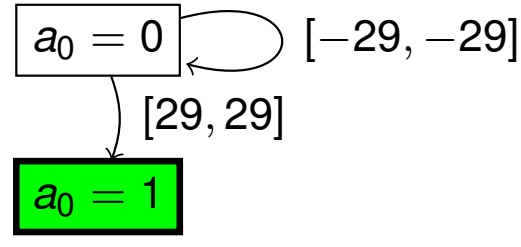
state partitioning:



PRISM model:

```
x0&!x1->
 29:x0'=f&x1'=t;
```

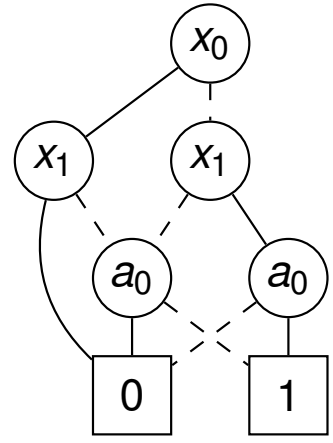
ACTMC to build:



(B) Abstraction: Building ACTMCs

- prepare states of ACTMC
- consider each valid variable valuation
- get abstract state
- get contained concrete state
- `PRISM` model to get successor rates and states
- partitioning to find containing abstract states
- widen rates of ACTMC

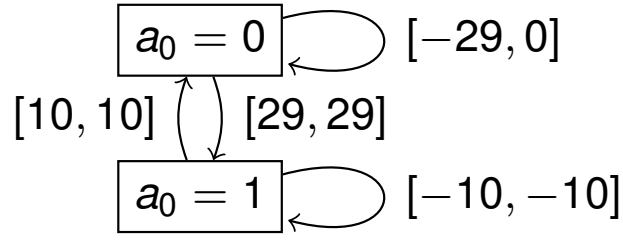
state partitioning:



`PRISM` model:

```
x0&!x1->
 29:x0'=f&x1'=t;
```

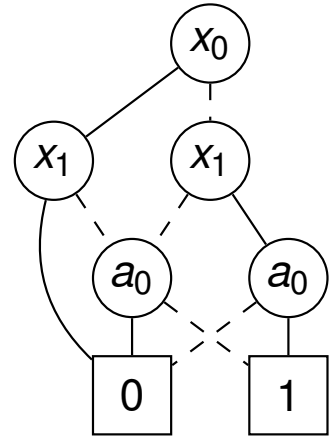
ACTMC to build:



(B) Abstraction: Building ACTMCs

- prepare states of ACTMC
- consider each valid variable valuation
- get abstract state
- get contained concrete state
- `PRISM` model to get successor rates and states
- partitioning to find containing abstract states
- widen rates of ACTMC
- rewards similar, but simpler

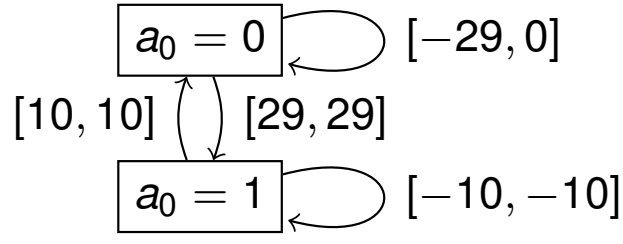
state partitioning:



`PRISM` model:

```
x0&!x1->
 29:x0'=f&x1'=t;
```

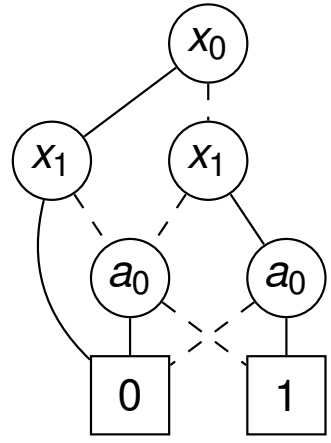
ACTMC to build:



(B) Abstraction: Building ACTMCs

- prepare states of ACTMC
- consider each valid variable valuation
- get abstract state
- get contained concrete state
- `PRISM` model to get successor rates and states
- partitioning to find containing abstract states
- widen rates of ACTMC
- rewards similar, but simpler
- finally, need some fixes

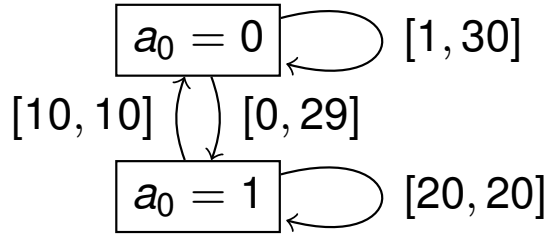
state partitioning:



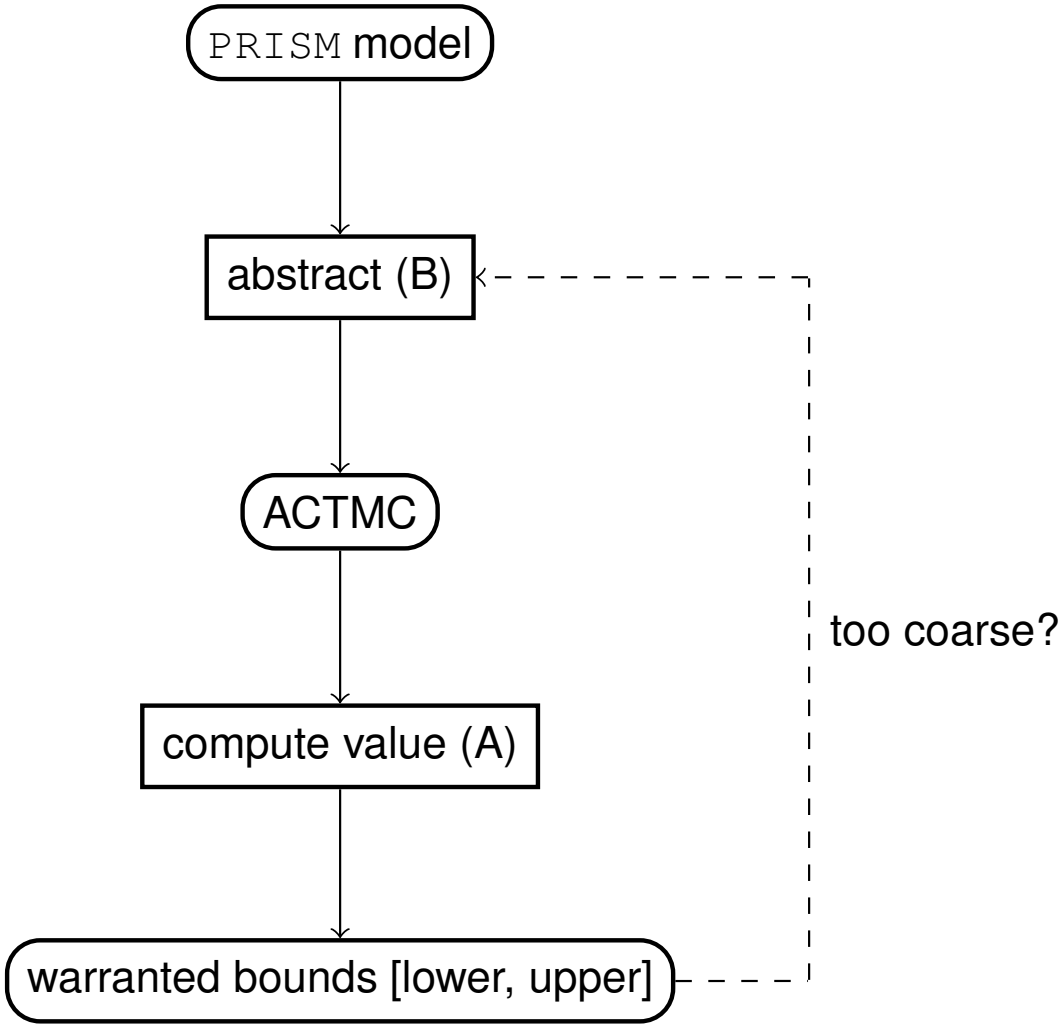
`PRISM` model:

```
x0&!x1->
 29:x0'=f&x1'=t;
```

ACTMC to build:



Overall scheme

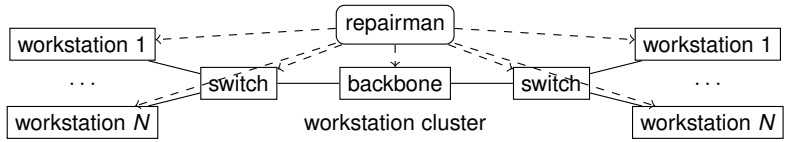


Case studies

- implementation of basic technique
- applied on

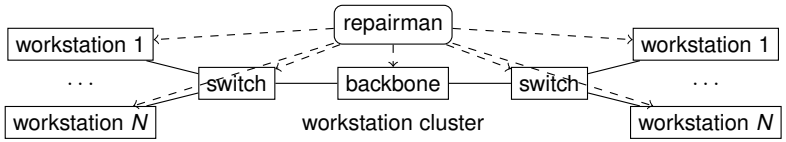
Case studies

- implementation of basic technique
- applied on
  - Workstation cluster



Case studies

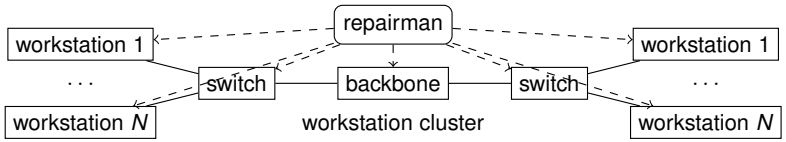
- implementation of basic technique
- applied on
  - Workstation cluster



  - Google File System

Case studies

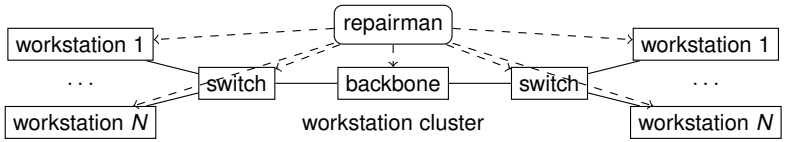
- implementation of basic technique
- applied on
  - Workstation cluster



  - Google File System
- results promising, e.g. for cluster

Case studies

- implementation of basic technique
- applied on
  - Workstation cluster



  - Google File System
- results promising, e.g. for cluster

		PRISM		A(E)CTMC Results			
N	$\|S\|$	T(m)	MB	$\|\mathfrak{A}\|$	T(m)	MB	Interval
64	$1.51 \cdot 10^5$	1	42	19420	2	89	$[127.98, 128.49]$
1024	$3.78 \cdot 10^7$	425	1015	19420	6	110	$[905.02, 905.20]$
8192	$2.42 \cdot 10^9$	**– Time out –**		19420	484	134	$[906.04, 906.04]$

Conclusion

- done so far
  - developed method to optimise general transient properties over relevant class of schedulers of CTMDPs
  - used for CTMC abstraction
  - discussed how to speed up abstraction
  - applied method on two case studies
- possible future works includes
  - improve refinement
  - abstraction of models already involving nondeterminism
  - integration into larger model checking context