Introduction to Model Checking

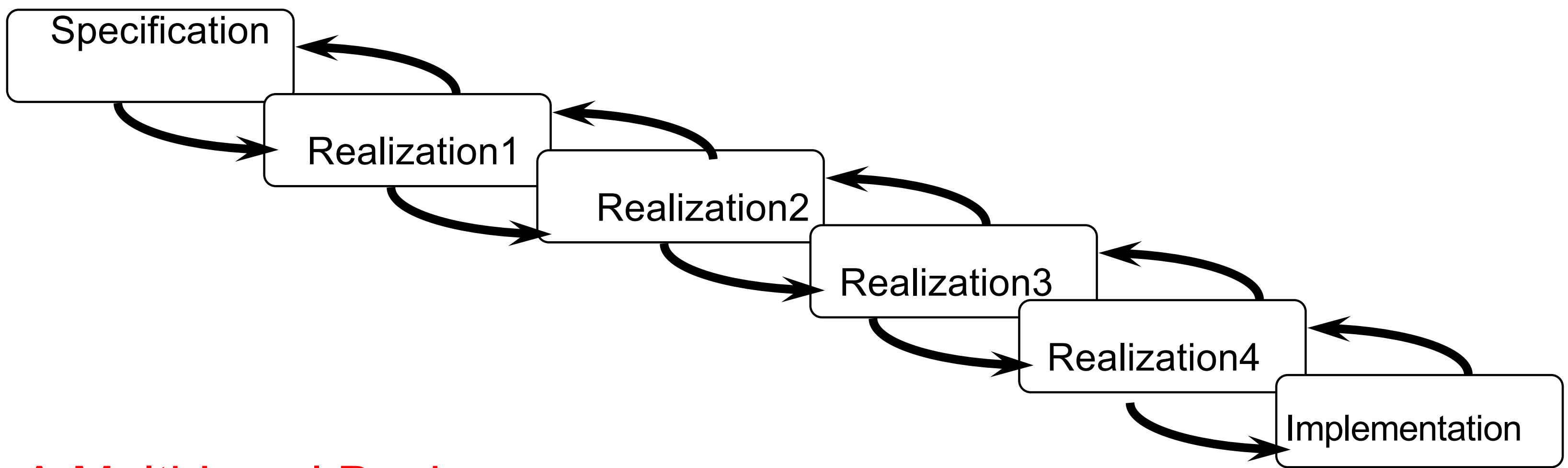
Ali Movaghar

Overview

- Basic concepts
- Probabilistic Model Checking
- Markov decision processes (MDPs)
- Adversaries
- PCTL
- PCTL model checking
- Costs and rewards

Design and Validation

A design is a process of getting a (more detailed) realization from a given specification.



A Multi-Level Design

and Validation

An implementation can be viewed as the most detailed realization.
https://web.eecs.umich.edu/~movaghar/Taxanomy-Dependable-Computing-2004.pdf

UNIVERSITY OF MICHIGAN

Design

- Design is a process of getting a (more detailed) realization from a given (higher-level) specification.
- The design of a complex system may happen on many levels.
- The implementation may be viewed as the lowest level of the design.

Validation

- Validation is a process of ensuring that a realization satisfies its specification.
- Validation is a process of ensuring that a design is correct.
- Validation is mainly used in system design and development.

Validation Methods

Validation has three main methods:

- Verification
- Evaluation
- Testing

Verification

- Verification is a formal mathematical method to prove that a realization satisfies its specification.

Evaluation

- Evaluation is a method for finding how well a system behaves.

Testing

- Testing is a method of proving that a realization does not satisfy its specification.

Integrated Validation Methods

- Testing, Verification, and Evaluation are usually complementary.

Methods for Evaluation

- Measurement
- Analytical Modeling
- Simulation Modeling
- Hybrid Modeling

So, why not test?

Testing only shows the presence of bugs, not their absence!

Methods for Testing

- Unit Testing: Validates that individual components or units of the software work correctly.
- Integration Testing: Ensures that different modules or services used by your application work well together.
- Functional Testing: Checks the software against the functional requirements/specifications.
- System Testing: Verifies that the complete and integrated software system meets the specified requirements.

Methods for Testing (Cont'd)

- Stress Testing: Determines the robustness of software by testing beyond the limits of normal operation.
- Performance Testing: Checks if the software performs well under their expected workload.
- Usability Testing: Evaluate the user-friendliness and ease of use of the software.
- Security Testing: Identifies vulnerabilities within the software and ensures that the data and resources are protected.

Methods for Testing (Cont'd)

- Acceptance Testing: Confirms that the software is ready for delivery by validating it against business requirements.
- Regression Testing: Ensures that new code changes do not adversely affect existing functionalities.
- Mutation testing: This helps ensure that the test cases are effective at finding potential bugs and that they cover the necessary aspects of the software's functionality.

What are formal methods?

- Techniques for analyzing systems, based on some mathematics.
- This does not mean that the user must be a mathematician.
- Some of the work is done informally, due to complexity.

Formal Methods

- Mathematically-based techniques for describing properties of systems
- Provide framework for
  - Specifying systems (and thus the notion of correctness)
  - Developing systems
  - Verifying correctness
    - Of implementation w.r.t. the specification
    - Equivalence of different implementations
- Reasoning is based on logic
  - Amenable to machine analysis and manipulation
  - In principle, can verify everything is true in the system!
    - Given enough time, skill, and patience

Formal Verification

Formal verification seeks to establish a mathematical proof that a system works correctly.

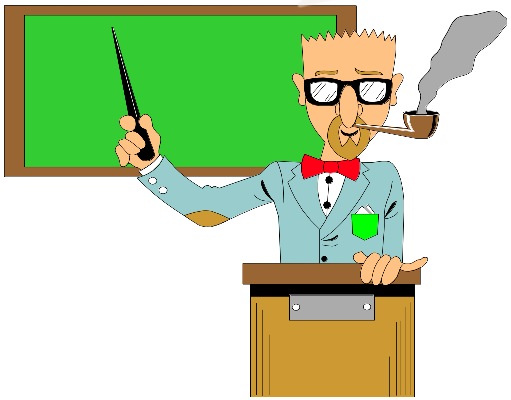
Formal Verification (Cont’d)

A formal approach provides:

- A system model (language) to describe the system,
- A specification model (language) to describe the correctness requirement,
- An analysis technique to verify that the system meets its specifications.

Why aren't FMs used more?

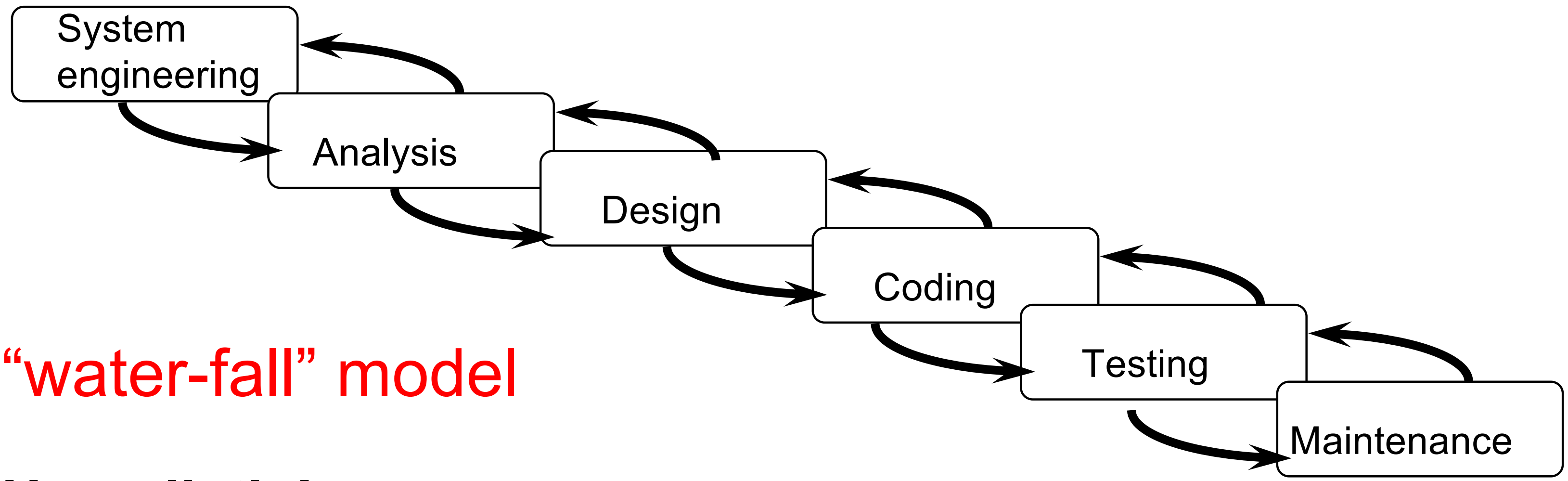
"Formal methods can revolutionize development!"



"Formal methods are difficult, expensive, not widely useful and for safety-critical systems only"

… and one more problem

Need to know what to build (specification) before you start building



“water-fall” model

Unrealistic!

- **May need to discover what to build iteratively**
- **Software changes all the time**

Formal Methods “Light”

- Partial application of formal methods
  - only parts of systems are specified
- Emphasis on analysis of some properties
  - security, fairness, deadlock freedom, rather than complete verification
- Debugging rather than assurance
- Automation

Most successful lightweight technique:
Model-Checking

Methods of Verification

There are two major methods for verification:

- Deductive Method
- Model Checking

Deductive Method

- In the deductive method, the problem is formulated as proving a theorem in a mathematical proof system.

Model Checking

- In the method of model checking, the behavior of the system is checked algorithmically through an exhaustive search of all reachable states.

Reactive Systems

- A reactive system is a system whose role is to maintain an ongoing interaction with its environment.
- The family of reactive systems includes most of the classes of systems whose correct and dependable construction is to be considered to be particularly challenging, including concurrent and real-time systems, embedded and process control systems, and operating systems.

Reactive Systems Properties

Reactive systems have usually the following properties:

- Concurrency
- Timeliness
- High performance, dependability, and security requirements

System Models

- Transition Systems (Automata)
- Process Algebras and their extensions
- Communicating Sequential Processes (CSP)
- Calculus of Communicating Systems (CCS)
- Actors
- Petri Nets and their extensions
- Deep Neural Networks (DNNs)
- Markov Decision Processes (MDPs)
- Other more recent models

https://web.eecs.umich.edu/~movaghar/pi-calculus.pdf
https://web.eecs.umich.edu/~movaghar/cspbook.pdf

Specification Models

- Temporal Logics and their Extensions
- Linear Temporal Logic (LTL)
- Computational Tree Logic (CTL)
- CTL*
- PCTL
- PCTL*
- CSL
- HyperLTL and HyperCTL*
- Other more recent models

Popular Tools

NuSMV

PRISM

SPIN

Dafny

Many Tools for DNNs

Motivation for Model Checking

- Safety-critical systems
  - Airplanes
  - Space shuttles
  - Railways
- Expensive mistakes
  - Chip design
  - Critical software
- Want to guarantee safe behavior over
- unbounded time

https://web.eecs.umich.edu/~movaghar/CACM_Article-2008.PDF
https://web.eecs.umich.edu/~movaghar/CACM_Article-2010.PDF

Smart vehicles

Motivation for Model Checking

Toyota Recalls 1.9 Million Prius Hybrids Over Software Flaw

By Jeremy Hsu

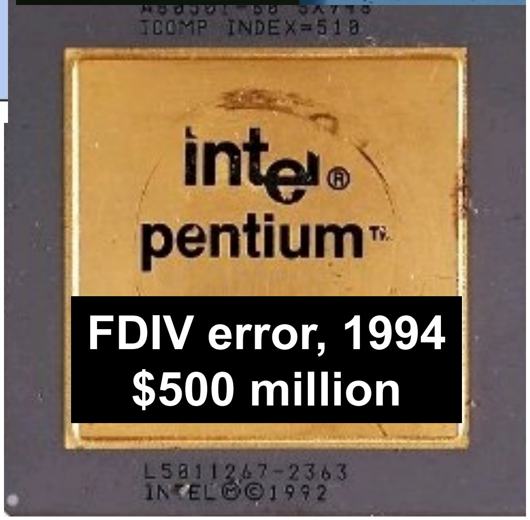
Posted 12 Feb 2014 | 21:55 GMT

32

```
<msblast.exe> (the primary executable of the exploit)
I just want to say LOVE YOU SAN!!
billy gates why do you make this possible ? Stop
making money and fix your software!!
windowsupdate.com
start %s
tftp -i %s GET %s
%d.%d.%d.%d
%i.%i.%i.%i
```

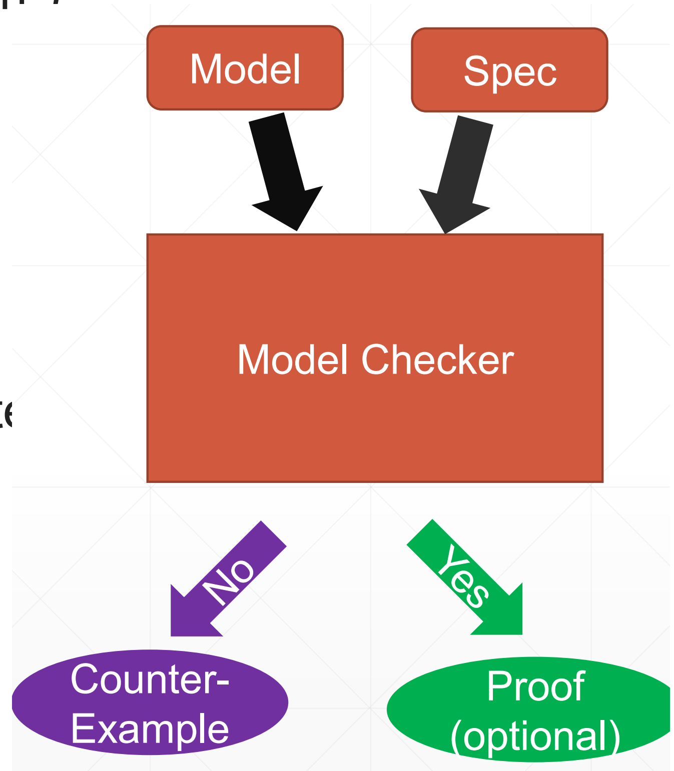
Estimated worst-case worm cost: > $50 billion

Bugs cost Time, Money, Lives, ...



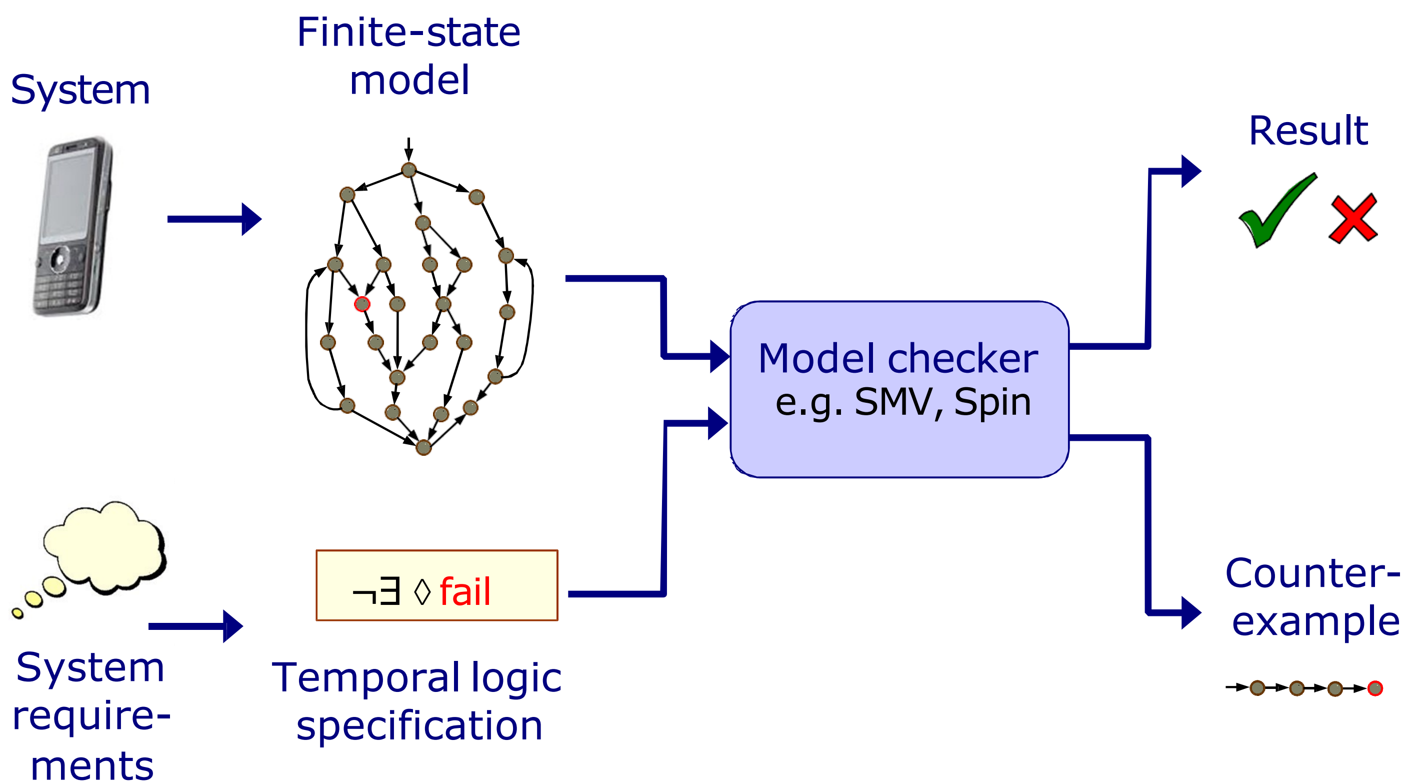
What is Model Checking?

- An approach for verifying the temporal behavior of a system
- Primarily fully-automated (“push-button”) techniques
- Model
  - Representation of the system
  - Need to decide the right level of granularity
- Specification
  - High-level desired property of syste
  - Considers infinite sequences
- PSPACE-complete for FSMs



Model Checking

Automated formal verification for finite-state models

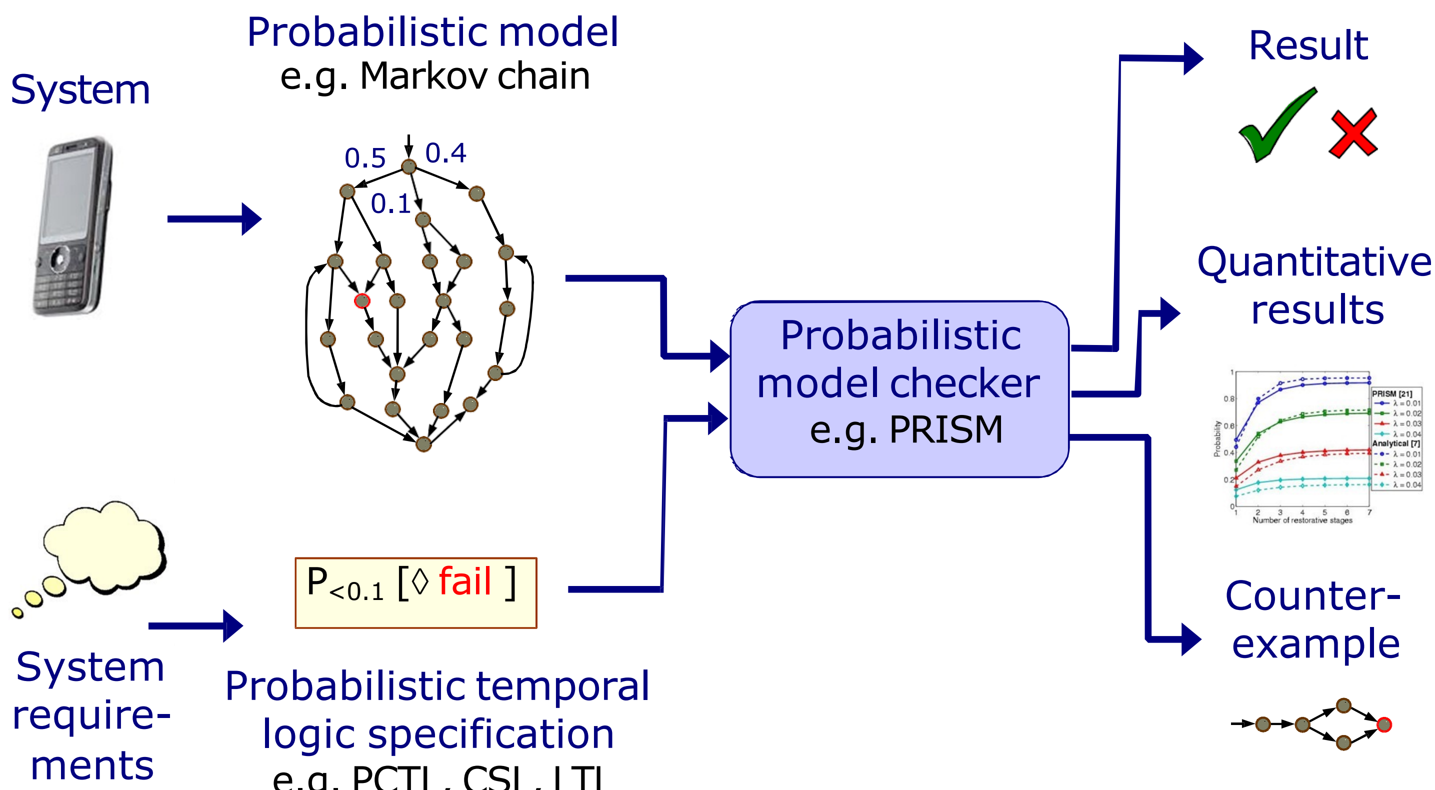


Overview

- Basic concepts
- Probabilistic Model Checking
- Markov decision processes (MDPs)
- Adversaries
- PCTL
- PCTL model checking
- Costs and rewards

Probabilistic Model Checking

Automatic verification of systems with probabilistic behaviour



Why Probability?

- Some systems are inherently probabilistic…

- Randomisation, e.g. in distributed coordination algorithms
  - as a symmetry breaker, in gossip routing to reduce flooding

- Examples: real-world protocols featuring randomisation:
  - Randomised back-off schemes
    - CSMA protocol, 802.11 Wireless LAN
  - Random choice of waiting time
    - IEEE1394 Firewire (root contention), Bluetooth (device discovery)
  - Random choice over a set of possible addresses
    - IPv4 Zeroconf dynamic configuration (link-local addressing)
  - Randomised algorithms for anonymity, contract signing, …

Why Probability? (Cont'd)

- Some systems are inherently probabilistic…
- Randomisation, e.g. in distributed coordination algorithms
  - as a symmetry breaker, in gossip routing to reduce flooding
- To model uncertainty and performance
  - to quantify rate of failures, express Quality of Service
- Examples:
  - computer networks, embedded systems
  - power management policies
  - nano-scale circuitry: reliability through defect-tolerance

Why Probability? (Cont'd)

- Some systems are inherently probabilistic…
- Randomisation, e.g. in distributed coordination algorithms
  - as a symmetry breaker, in gossip routing to reduce flooding
- To model uncertainty and performance
  - to quantify rate of failures, express Quality of Service
- To model biological processes
  - reactions occurring between large numbers of molecules are naturally modelled in a stochastic fashion

Verifying Probabilistic Systems

- We are not just interested in correctness

- We want to be able to quantify:
  - — security, privacy, trust, anonymity, fairness
  - — safety, reliability, performance, dependability
  - — resource usage, e.g. battery life
  - — and much more…

- Quantitative, as well as qualitative requirements:
  - — how reliable is my car's Bluetooth network?
  - — how efficient is my phone's power management policy?
  - — is my bank's web-service secure?
  - — what is the expected long-run percentage of protein X?

Probabilistic Models

- Markov Decision Process (MDP)
  - — probabilistic and nondeterministic behavior
  - — the semantic base for extended models below
- Probabilistic Timed Automata (PTA)
  - — extend MDPs with clocks to express timed behavior
- Probabilistic Hybrid Automata (PHA)
  - — extend clocks of PTAs to more general continuous variables
  - — often described by differential equations
- Stochastic Activity Networks (SAN)
- Hybrid Stochastic Activity Networks (HSAN)

Nondeterminism

- Some aspects of a system may not be probabilistic and should not be modeled probabilistically; for example:
- Concurrency - scheduling of parallel components
  - —e.g. randomized distributed algorithms - multiple probabilistic processes operating asynchronously
- Underspecification - unknown model parameters
  - —e.g. a probabilistic communication protocol designed for message propagation delays of between d_{min} and d_{max}
- Unknown environments
  - —e.g. probabilistic security protocols - unknown adversary
- Decision-making and control
  - —e.g. optimal resource management and optimal control

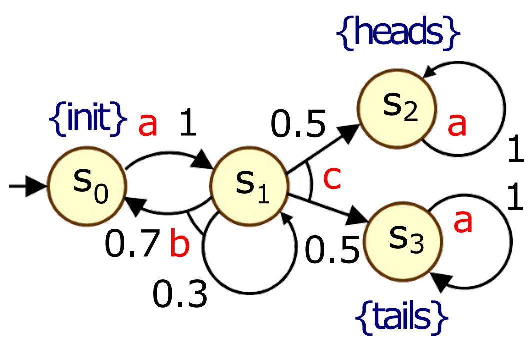
Overview

- Basic concepts
- Probabilistic Model Checking
- Markov decision processes (MDPs)
- Adversaries
- PCTL
- PCTL model checking
- Costs and rewards

Markov Decision Processes

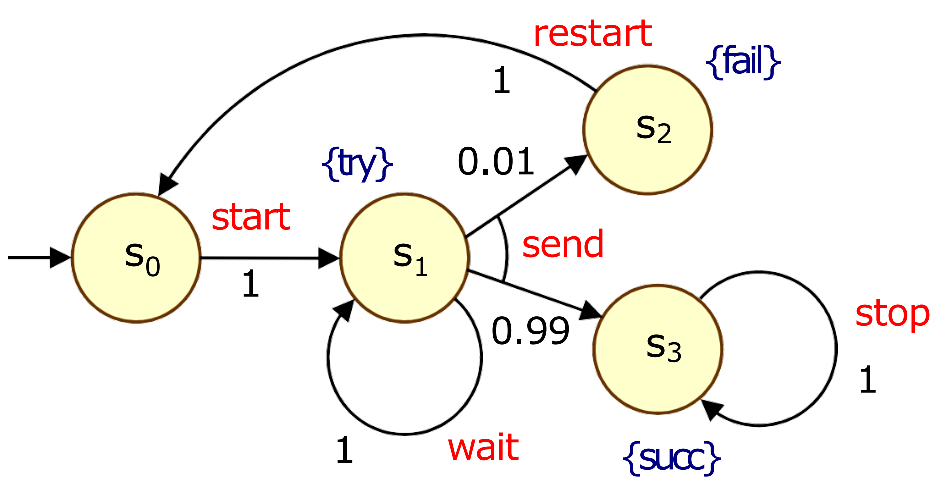
- Formally, an MDP M is a tuple (S,s_{init},Steps,L) where:
  - S is a finite set of states (“state space”)
  - $s_{init} \in S$ is the initial state
  - Steps : $S \rightarrow 2^{Act \times Dist(S)}$ is the transition probability function where Act is a set of actions and Dist(S) is the set of discrete probability distributions over the set S
  - L : $S \rightarrow 2^{AP}$ is a labelling with atomic propositions

- Notes:
  - Steps(s) is always non-empty, i.e. no deadlocks
  - the use of actions to label distributions is optional



Simple MDP Example

- Simple communication protocol
  - after one step, process starts trying to send a message
  - then, a nondeterministic choice between: (a) waiting a step because the channel is unready; (b) sending the message
  - if the latter, with probability 0.99 send successfully and stop
  - and with probability 0.01, message sending fails, restart



Modeling MDPs

- Guarded Commands modeling language
  - simple, textual, state-based language
  - based on Reactive Modules basic components: modules, variables, and commands

- Modules:
  - components of the system being modelled
  - a module represents a single MDP

```
module example
    ...
endmodule
```

Modeling MDPs

- Guarded Commands modeling language
  - simple, textual, state-based language
  - based on Reactive Modules basic components: modules, variables, and commands
- Variables:
  - finite-domain (bounded integer ranges or Booleans)
  - local or global – anyone can read, only the owner can modify
  - variable valuation = state of the MDP

```
module example
    s : [0..3] init 0;
    ...
endmodule
```

Modeling MDPs

- Guarded Commands modeling language
  - simple, textual, state-based language
  - based on Reactive Modules
  - basic components: modules, variables, and commands
- Commands:
  - describe the transitions between the states

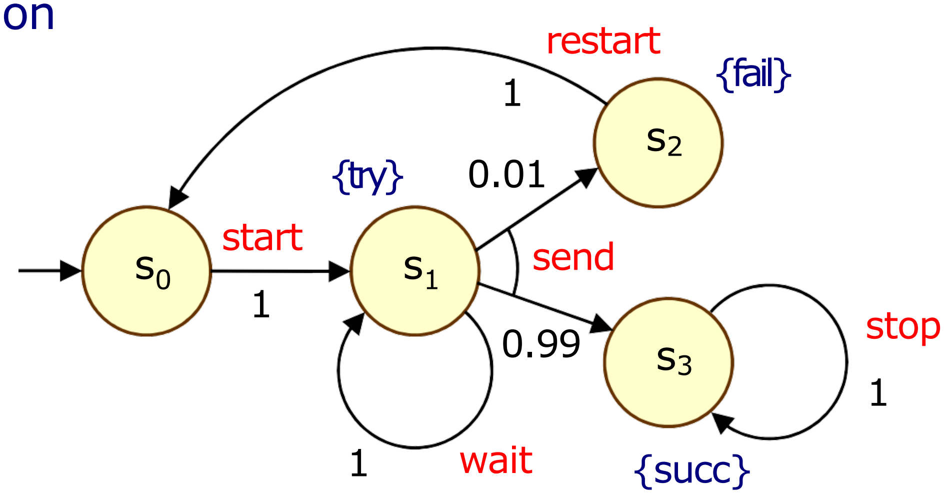
```
[act] exp -> p_1: asgn_11 & asgn_12 & ... + ... + p_n: asgn_n1 & ... ;
```

action guard probability update probability update

```
module example
    s : [0..3] init 0;
    ...
    [send] (s = 1) -> 0.01: (s' = 2) + 0.99: (s' = 3);
    ...
endmodule
```

Simple MDP Example

- Simple communication protocol

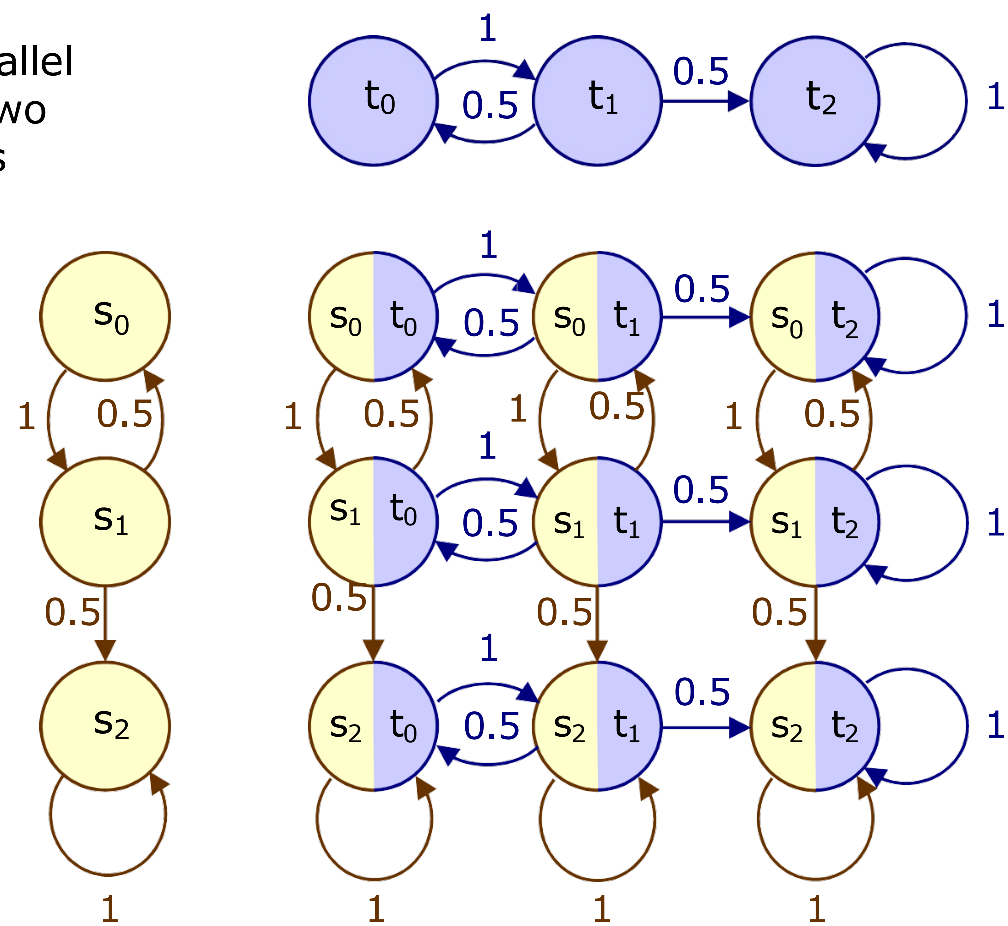



```
module example
    s : [0..3] init 0;
    [start]   (s = 0) -> (s' = 1);
    [wait]    (s = 1) -> true;
    [send]    (s = 1) -> 0.01: (s' = 2) + 0.99: (s' = 3);
    [restart] (s = 2) -> (s' = 0);
    [stop]    (s = 3) -> true;
endmodule
```

Example - Parallel Composition

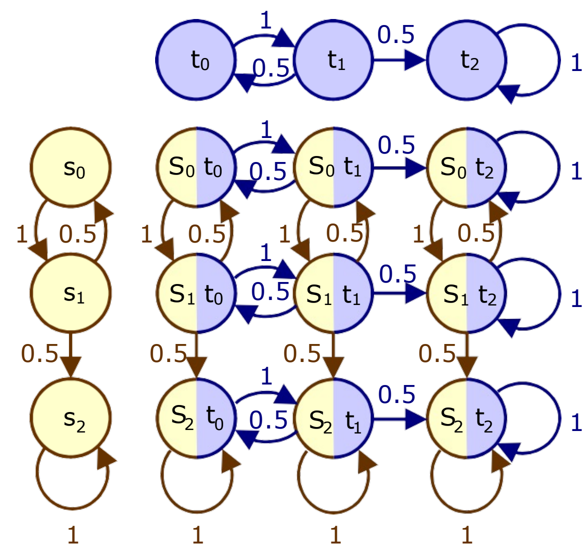
Asynchronous parallel composition of two 3-state DTMCs

Action labels omitted here



Example - Parallel Composition

Asynchronous parallel composition of two 3-state DTMCs




```
module threestate
    s : [0..2] init 0;
    [] s = 0 -> (s' = 1);
    [] s = 1 -> 0.5: (s' = s - 1)
              + 0.5: (s' = s + 1);
    [] s > 1 -> true;
endmodule

module copy = threestate[s = t] endmodule

system
    threestate || copy
endsystem
```

Default parallel composition on matching action labels – can be omitted

Paths and Probabilities

- A (finite or infinite) path through an MDP
  - — is a sequence of states and action/distribution pairs
  - — e.g. $s_0(a_0,\mu_0)s_1(a_1,\mu_1)s_2\ldots$
  - — such that $(a_i,\mu_i) \in Steps(s_i)$ and $\mu_i(s_{i+1}) > 0$ for all $i \geq 0$
  - — represents an execution (i.e. one possible behaviour) of the system which the MDP is modelling
  - — note that a path resolves both types of choices: nondeterministic and probabilistic

- To consider the probability of some behaviour of the MDP
  - — first need to resolve the nondeterministic choices
  - — …which results in a Markov chain (DTMC)
  - — …for which we can define a probability measure over paths

Overview

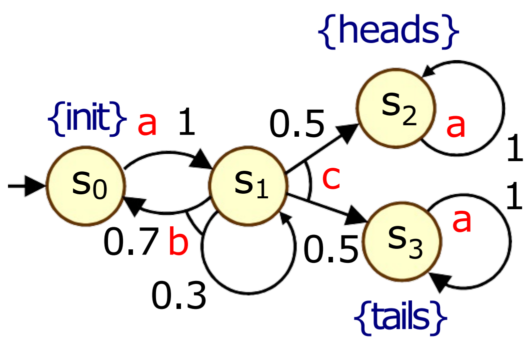
- Basic concepts
- Probabilistic Model Checking
- Markov decision processes (MDPs)
- Adversaries
- PCTL
- PCTL model checking
- Costs and rewards

Adversaries

- An adversary resolves nondeterministic choice in an MDP
  - also known as "schedulers", "strategies" or "policies"
- Formally:
  - an adversary A of an MDP M is a function mapping every finite path $\omega = s_0(a_1,\mu_1)s_1 \ldots s_n$ to an element of $Steps(s_n)$
- For each A can define a probability measure Pr^A_s over paths
  - constructed through an infinite state Markov chain (DTMC)
  - states of the DTMC are the finite paths of A starting in state s
  - the initial state is s (the path starting in s of length 0)
  - $\mathbf{P}^A_s(\omega,\omega') = \mu(s)$ if $\omega' = \omega(a, \mu)s$ and $A(\omega)=(a,\mu)$
  - $\mathbf{P}^A_s(\omega,\omega') = 0$ otherwise

Adversaries - Examples

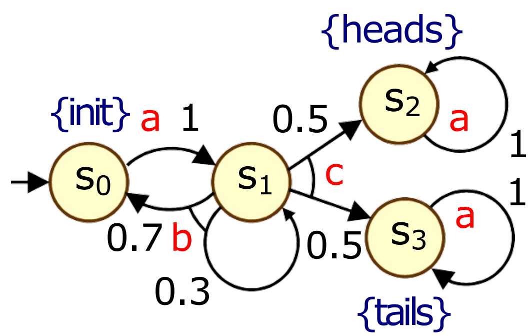
- Consider the simple MDP below
  - note that s_1 is the only state for which |Steps(s)| > 1
  - i.e. s_1 is the only state for which an adversary makes a choice
  - let μ_b and μ_c denote the probability distributions associated with actions b and c in state s_1

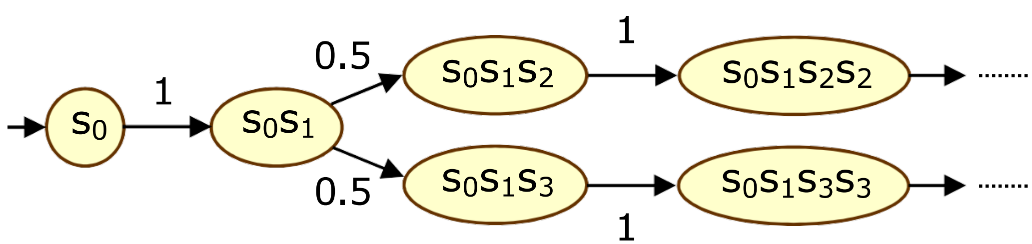


- Adversary A_1
  - picks action c the first time
  - $A_1(s_0 s_1)=(c,\mu_c)$
- Adversary A_2
  - picks action b the first time, then c
  - $A_2(s_0 s_1)=(b,\mu_b)$, $A_2(s_0 s_1 s_1)=(c,\mu_c)$, $A_2(s_0 s_1 s_0 s_1)=(c,\mu_c)$

Adversaries - Examples

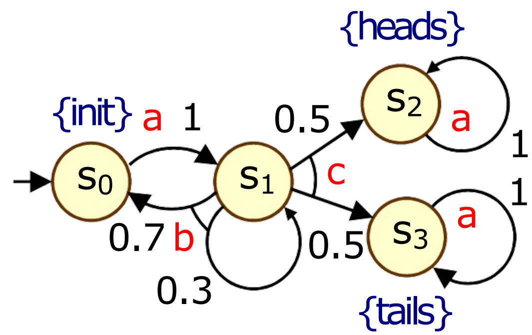
- Fragment of DTMC for adversary A_1
  - — A_1 picks action c the first time

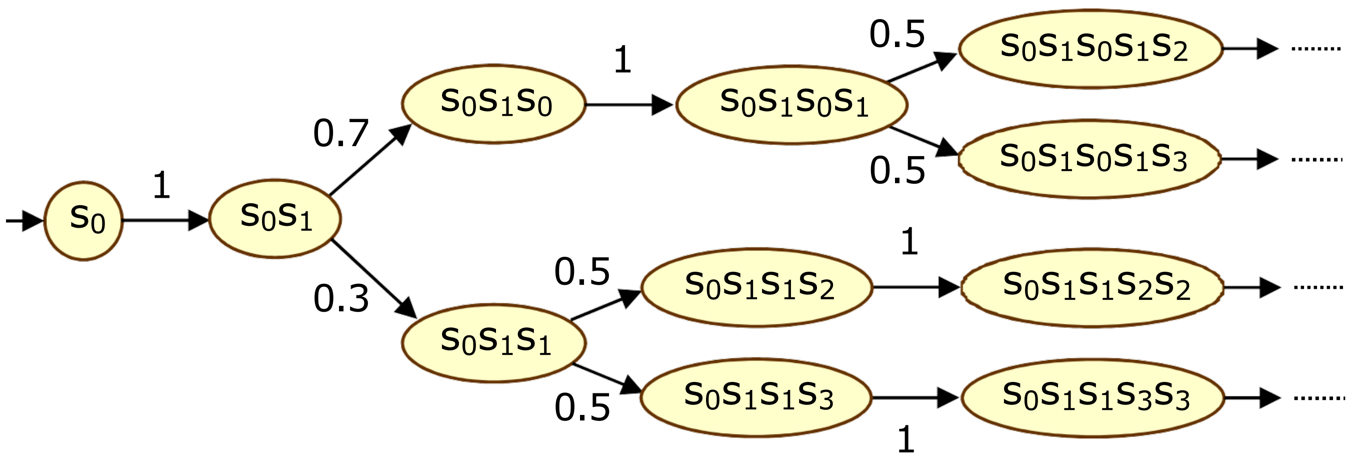




Adversaries - Examples

- Fragment of DTMC for adversary A_2
  - — A_2 picks action b, then c

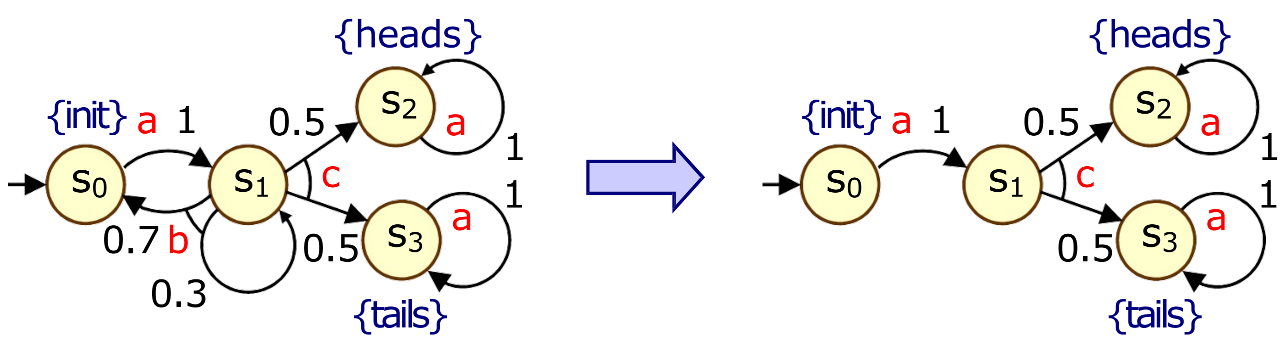




Memoryless Adversaries

- Memoryless adversaries always pick same choice in a state
  - — also known as: positional, Markov, simple
  - — formally, for adversary A:
  - — $A(s_0(a_1,\mu_1)s_1\ldots s_n)$ depends only on s_n
  - — resulting DTMC can be mapped to a |S|-state DTMC

- From previous example:
  - — adversary A_1 (picks c in s_1) is memoryless, A_2 is not



Overview

- Basic concepts
- Probabilistic Model Checking
- Markov decision processes (MDPs)
- Adversaries
- PCTL
- PCTL model checking
- Costs and rewards

PCTL

- Temporal logic for describing properties of MDPs
  - PCTL = Probabilistic Computation Tree Logic
- Extension of (non-probabilistic) temporal logic CTL
  - key addition is probabilistic operator P
  - quantitative extension of CTL's A and E operators
- Example
  - send → $P_{\geq 0.95}$ [true $U^{\leq 10}$ deliver]
  - "if a message is sent, then the probability of it being delivered within 10 steps is at least 0.95"

PCTL Syntax

- PCTL syntax:
  - $\varphi ::= \text{true} \mid a \mid \varphi \wedge \varphi \mid \neg\varphi \mid P_{\sim p}[\,\psi\,]$ (state formulas)

    ψ is true with probability ~p

  - $\psi ::= \bigcirc\, \varphi \mid \varphi\, U^{\leq k}\, \varphi \mid \varphi\, U\, \varphi$ (path formulas)

    "next" "bounded until" "until"

  - where a is an atomic proposition, used to identify states of interest, $p \in [0,1]$ is a probability, $\sim \in \{<,>,\leq,\geq\}$, $k \in \mathbb{N}$

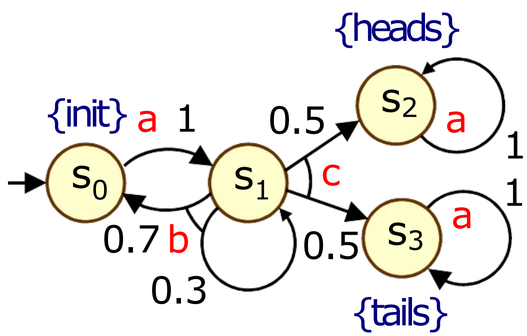
- A PCTL formula is always a state formula
  - path formulas only occur inside the P operator

PCTL Semantics for MDPs

- PCTL formulas interpreted over states of an MDP
  - $s \vDash \varphi$ denotes φ is "true in state s" or "satisfied in state s"

- Semantics of (non-probabilistic) state formulas:
  - for a state s of the MDP $(S, s_{init}, \mathbf{P}, L)$:
  - $s \vDash a$ - $a \in L(s)$
  - $s \vDash \varphi_1 \wedge \varphi_2$ - $s \vDash \varphi_1$ and $s \vDash \varphi_2$
  - $s \vDash \neg\varphi$ - $s \vDash \varphi$ is false

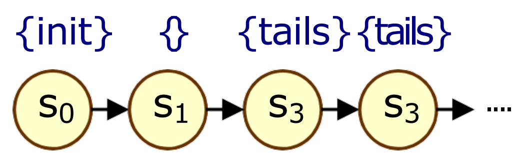
- Examples
  - $s_3 \vDash tails$
  - $s_2 \vDash heads \wedge \neg init$



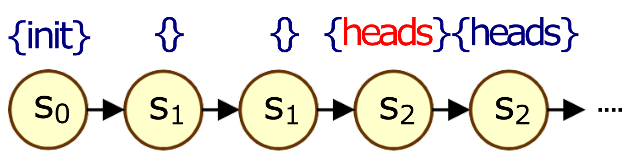
PCTL Semantics for MDPs

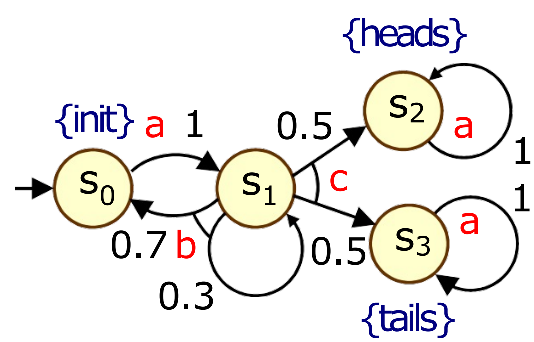
- Semantics of path formulas:
  - for a path $\omega = s_0 s_1 s_2 \ldots$ in the MDP:
  - $\omega \vDash \bigcirc \varphi$ - $s_1 \vDash \varphi$
  - $\omega \vDash \varphi_1 \, U^{\leq k} \, \varphi_2$ - $\exists i \leq k$ such that $s_i \vDash \varphi_2$ and $\forall j < i$, $s_j \vDash \varphi_1$
  - $\omega \vDash \varphi_1 \, U \, \varphi_2$ - $\exists k \geq 0$ such that $\omega \vDash \varphi_1 \, U^{\leq k} \, \varphi_2$

- Some examples of satisfying paths:
  - $\bigcirc \neg init$



  - $\neg tails \; U \; heads$

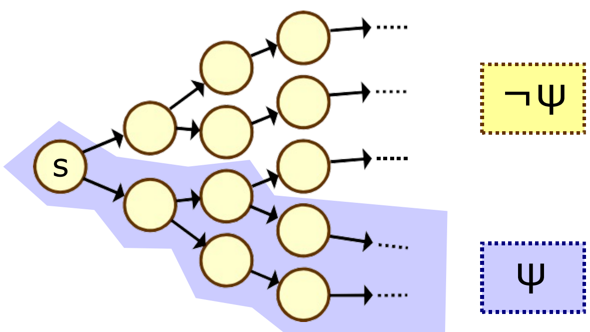




PCTL Semantics for MDPs

- Semantics of the probabilistic operator P
  - can only define probabilities for a specific adversary A
  - $s \models P_{\sim p} [\ \psi\ ]$ means "the probability, from state s, that ψ is true for an outgoing path satisfies ~p for all adversaries A"
  - formally $s \models P_{\sim p} [\ \psi\ ] \Leftrightarrow Prob^A(s, \psi) \sim p$ for all adversaries A

    where $Prob^A(s, \psi) = Pr^A_s \{\omega \in Path^A(s) \mid \omega \models \psi \}$



$Prob^A(s, \psi) \sim p$

Minimum and Maximum Probabilities

- Letting:
  - $p_{max}(s, \psi) = \sup_A \text{Prob}^A(s, \psi)$
  - $p_{min}(s, \psi) = \inf_A \text{Prob}^A(s, \psi)$
- We have:
  - if $\sim \in \{\geq, >\}$, then $s \vDash P_{\sim p} [\ \psi\ ]$ - $p_{min}(s, \psi) \sim p$
  - if $\sim \in \{<, \leq\}$, then $s \vDash P_{\sim p} [\ \psi\ ]$ - $p_{max}(s, \psi) \sim p$
- Model checking $P_{\sim p}[\ \psi\ ]$ reduces to the computation over all adversaries of either:
  - the minimum probability of ψ holding
  - the maximum probability of ψ holding
- Crucial result for model checking PCTL on MDPs
  - memoryless adversaries suffice, i.e. there are always memoryless adversaries A_{min} and A_{max} for which:
  - $\text{Prob}^{A_{min}}(s, \psi) = p_{min}(s, \psi)$ and $\text{Prob}^{A_{max}}(s, \psi) = p_{max}(s, \psi)$

Overview

- Basic concepts
- Probabilistic Model Checking
- Markov decision processes (MDPs)
- Adversaries
- PCTL
- PCTL model checking
- Costs and rewards

PCTL Model Checking

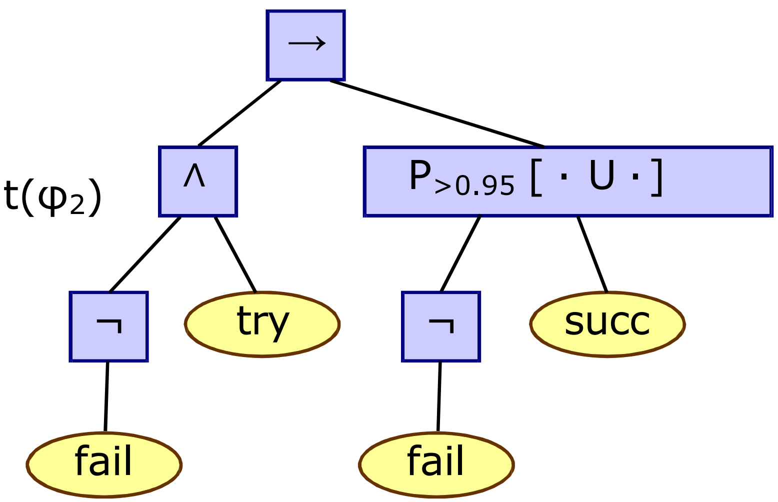
- Algorithm for PCTL model checking
- inputs: MDP $M=(S,s_{init},Steps,L)$, PCTL formula φ
  - output: $Sat(\varphi) = \{s \in S \mid s \vDash \varphi \}$ = set of states satisfying φ

- What does it mean for an MDP D to satisfy a formula φ?
  - sometimes, want to check that $s \vDash \varphi\ \forall\ s \in S$, i.e. $Sat(\varphi) = S$
  - sometimes, just want to know if $s_{init} \vDash \varphi$, i.e. if $s_{init} \in Sat(\varphi)$

- Sometimes, focus on quantitative results
  - e.g. compute the result of Pmax=? [◊ error]
  - e.g. compute result of Pmax=? [$\lozenge^{\leq k}$ error] for $0 \leq k \leq 100$

PCTL Model Checking for MDPs

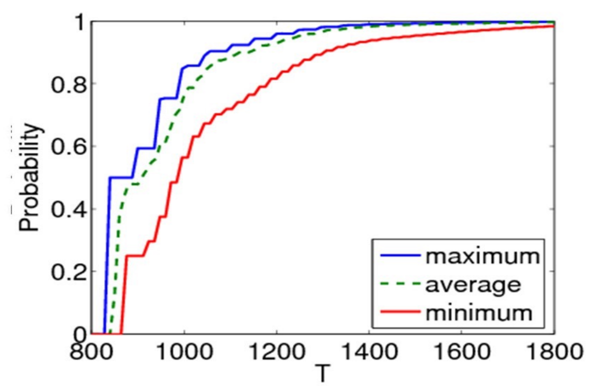
- Basic algorithm proceeds by induction on parse tree of φ
  - example: $\varphi = (\neg fail \wedge try) \rightarrow P_{>0.95}[\, \neg fail \; U \; succ \,]$
- For the non-probabilistic operators:
  - $Sat(true) = S$
  - $Sat(a) = \{s \in S \mid a \in L(s)\}$
  - $Sat(\neg\varphi) = S \setminus Sat(\varphi)$
  - $Sat(\varphi_1 \wedge \varphi_2) = Sat(\varphi_1) \cap Sat(\varphi_2)$
- For the $P_{\sim p}[\, \psi \,]$ operator
  - need to compute the probabilities $Prob(s, \psi)$ for all states $s \in S$
  - focus here on the "until" case: $\psi = \varphi_1 \; U \; \varphi_2$



Quantitative Properties

- For PCTL properties with P as the outermost operator
  - quantitative form (two types): $Pmin_{=?}$ [ψ] and $Pmax_{=?}$ [ψ]
  - i.e. "what is the minimum/maximum probability (over all adversaries) that path formula ψ is true?"
  - corresponds to an analysis of best-case or worst-case behaviour of the system
  - model checking is no harder since compute the values of $p_{min}(s, \psi)$ or $p_{max}(s, \psi)$ anyway
  - useful to spot patterns/trends

- Example: CSMA/CD protocol
  - "min/max probability that a message is sent within the deadline"



Some Real PCTL Examples

- Byzantine agreement protocol
  - $Pmin_{=?}$ [◊ (agreement ∧ rounds≤2)]
  - "what is the minimum probability that agreement is reached within two rounds?"
- CSMA/CD communication protocol
  - $Pmax_{=?}$ [◊ collisions=k]
  - "what is the maximum probability of k collisions?"
- Self-stabilisation protocols
  - $Pmin_{=?}$ [$◊^{\leq t}$ stable]
  - "what is the minimum probability of reaching a stable state within k steps?"

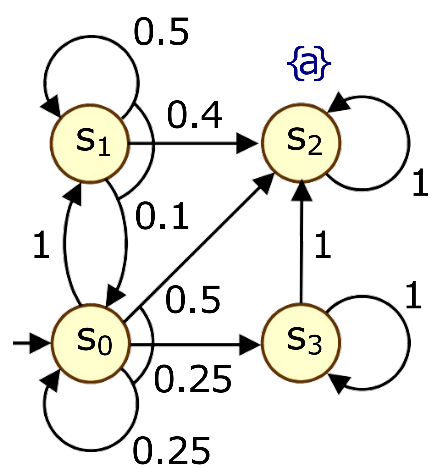
PCTL Until for MDPs

- Computation of probabilities $p_{min}(s, \varphi_1 \text{ U } \varphi_2)$ for all $s \in S$
- First identify all states where the probability is 1 or 0
  - "precomputation" algorithms, yielding sets S^{yes}, S^{no}
- Then compute (min) probabilities for remaining states ($S^?$)
  - either: solve linear programming problem
  - or: approximate with an iterative solution method

Example: $P_{\geq p}$ [◊ a]

≡

$P_{\geq p}$ [true U a]

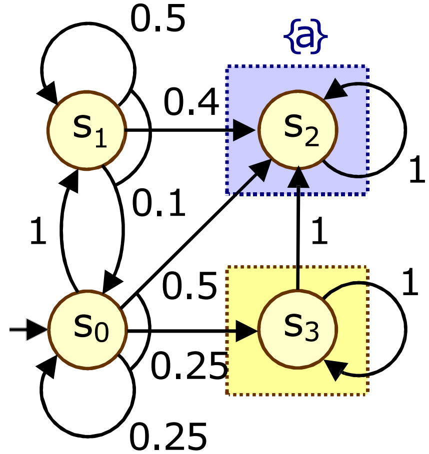


PCTL Until - Precomputation

- Identify all states where $p_{min}(s, \varphi_1 \text{ U } \varphi_2)$ is 1 or 0
  - $S^{yes} = Sat(P_{\geq 1}[\ \varphi_1 \text{ U } \varphi_2\ ])$, $S^{no} = Sat(\neg\ P_{>0}[\ \varphi_1 \text{ U } \varphi_2\ ])$
- Two graph-based precomputation algorithms:
  - algorithm Prob1A computes S^{yes}
    - for all adversaries the probability of satisfying $\varphi_1 \text{ U } \varphi_2$ is 1
  - algorithm Prob0E computes S^{no}
    - there exists an adversary for which the probability is 0

Example:

$P_{\geq p}[\lozenge\ a\ ]$



$S^{yes} = Sat(P_{\geq 1}[\lozenge\ a\ ])$

$S^{no} = Sat(\neg P_{>0}[\lozenge\ a\ ])$

Method 1 - Linear Programming

- Probabilities $p_{min}(s, \varphi_1 \cup \varphi_2)$ for remaining states in the set $S^? = S \setminus (S^{yes} \cup S^{no})$ can be obtained as the unique solution of the following linear programming (LP) problem:

$$\text{maximize} \sum_{s \in S^?} x_s \text{ subject to the constraints:}$$

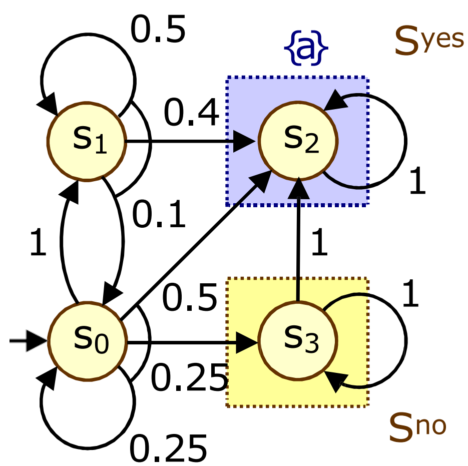
$$x_s \leq \sum_{s' \in S^?} \mu(s') \cdot x_{s'} + \sum_{s' \in S^{yes}} \mu(s')$$

$$\text{for all } s \in S^? \text{ and for all } (a,\mu) \in Steps(s)$$

- Simple case of a more general problem known as the stochastic shortest path problem

- This can be solved with standard techniques
  - e.g. Simplex, ellipsoid method, branch-and-cut

Example - PCTL Until (LP)



Let $x_i = p_{min}(s_i, \Diamond\, a)$

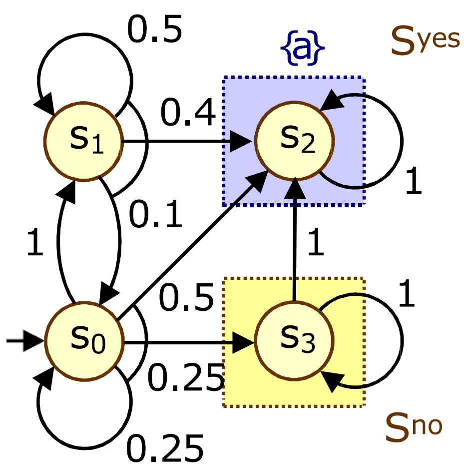
S^{yes}: $x_2=1$, S^{no}: $x_3=0$

For $S^? = \{x_0, x_1\}$:

Maximise x_0+x_1 subject to constraints:

- $x_0 \leq x_1$
- $x_0 \leq 0.25 \cdot x_0 + 0.5$
- $x_1 \leq 0.1 \cdot x_0 + 0.5 \cdot x_1 + 0.4$

Example - PCTL Until (LP)

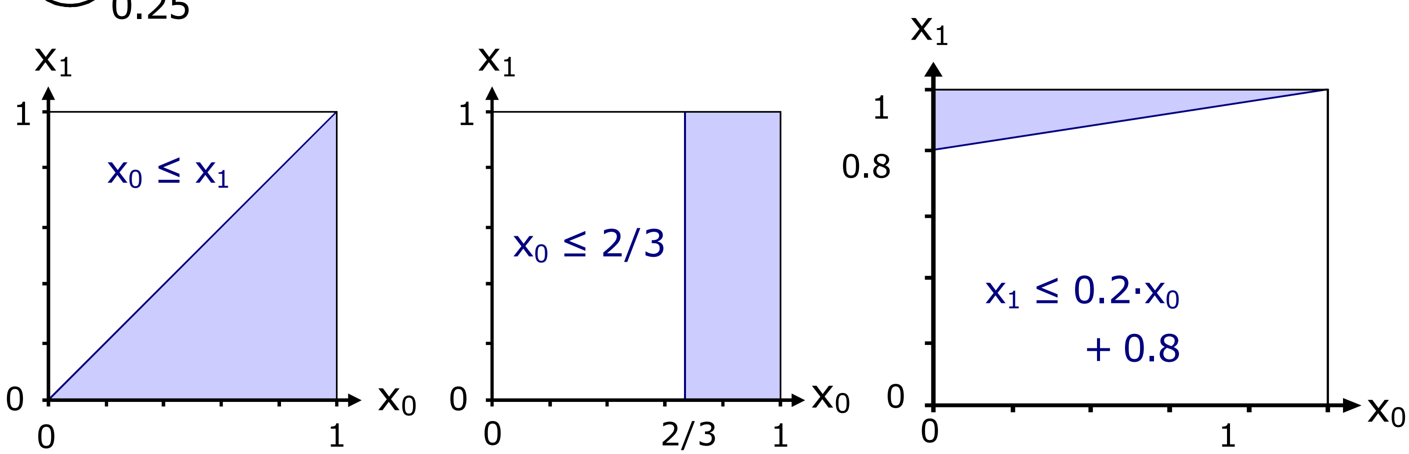


Let $x_i = p_{min}(s_i, \Diamond a)$ S^{yes}: $x_2=1$,

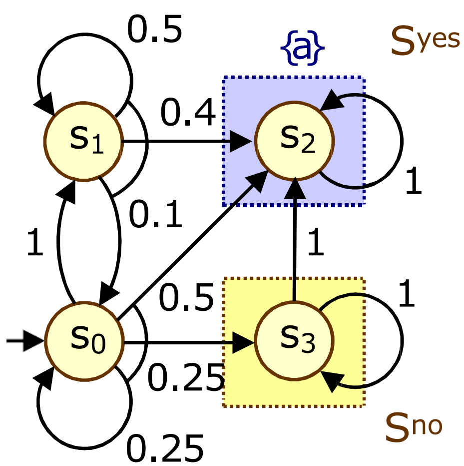
S^{no}: $x_3=0$ For $S^? = \{x_0, x_1\}$:

Maximise x_0+x_1 subject to constraints:

- $x_0 \leq x_1$
- $x_0 \leq 2/3$
- $x_1 \leq 0.2 \cdot x_0 + 0.8$



Example - PCTL Until (LP)



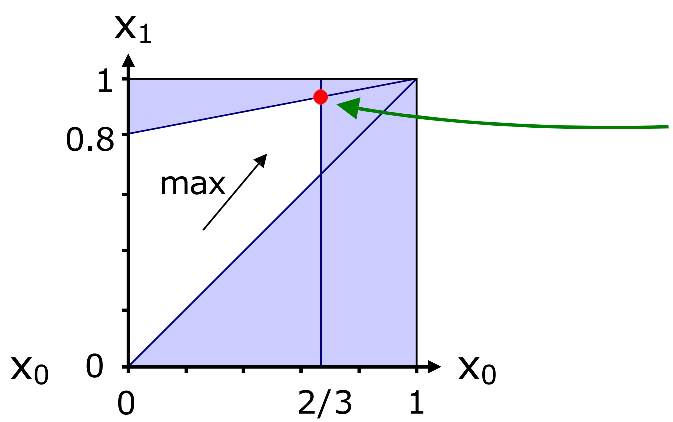
Let $x_i = p_{min}(s_i, F\ a)$ S^{yes}:

$x_2=1$, S^{no}: $x_3=0$

For $S^? = \{x_0, x_1\}$:

Maximise x_0+x_1 subject to constraints:

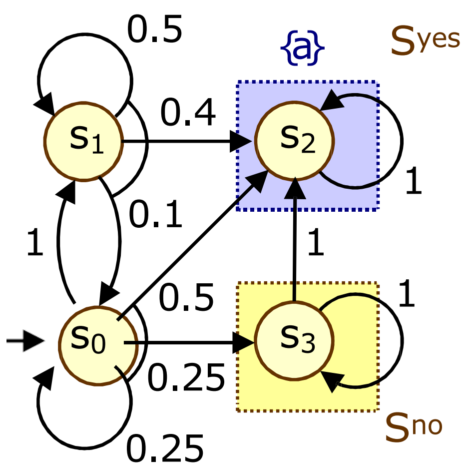
- $x_0 \leq x_1$
- $x_0 \leq 2/3$
- $x_1 \leq 0.2 \cdot x_0 + 0.8$



Solution: (x_0, x_1)

$= (2/3, 14/15)$

Example - PCTL Until (LP)



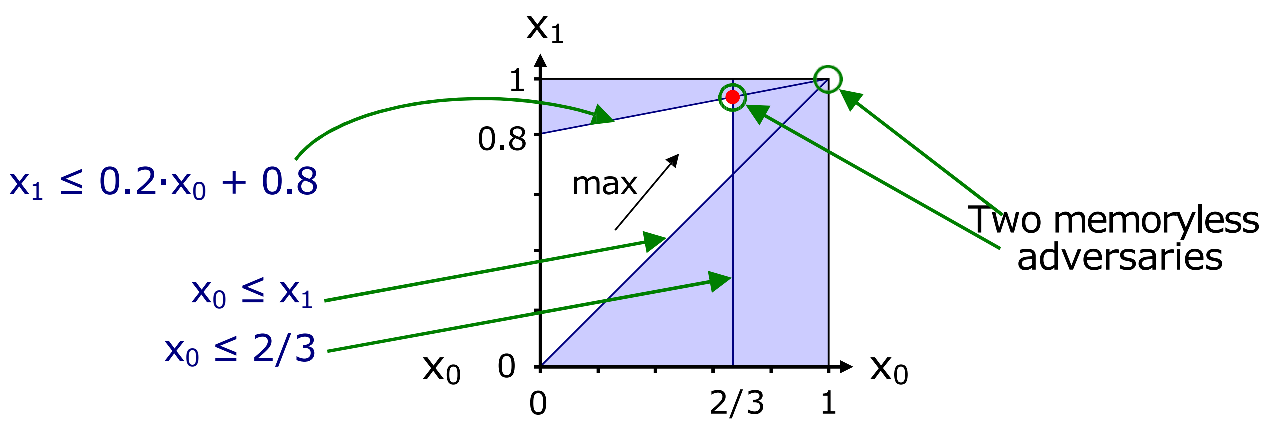
Let $x_i = p_{min}(s_i, \Diamond\, a)$

S^{yes}: $x_2=1$, S^{no}: $x_3=0$

For $S^? = \{x_0, x_1\}$:

Maximise x_0+x_1 subject to constraints:

- $x_0 \leq x_1$
- $x_0 \leq 2/3$
- $x_1 \leq 0.2 \cdot x_0 + 0.8$



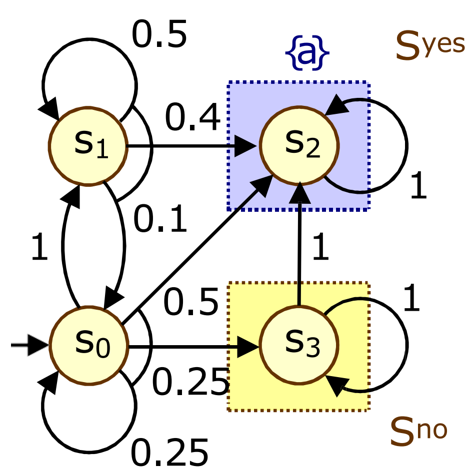
Method 2 – Value Iteration

- For probabilities $p_{min}(s, \varphi_1 \text{ U } \varphi_2)$ it can be shown that:
  - $p_{min}(s, \varphi_1 \text{ U } \varphi_2) = \lim_{n\to\infty} x_s^{(n)}$ where:

$$x_s^{(n)} = \begin{cases} 1 & \text{if } s \in S^{yes} \\ 0 & \text{if } s \in S^{no} \\ 0 & \text{if } s \in S^{?} \text{ and } n = 0 \\ \min_{(a,\mu)\in Steps(s)} \left(\sum_{s' \in S} \mu(s') \cdot x_{s'}^{(n-1)} \right) & \text{if } s \in S^{?} \text{ and } n > 0 \end{cases}$$

- This forms the basis for an (approximate) iterative solution
  - iterations terminated when solution converges sufficiently

Example - PCTL Until (Value Iteration)



Compute: $p_{min}(s_i, \Diamond a)$

$S^{yes} = \{x_2\}$, $S^{no} = \{x_3\}$, $S^? = \{x_0, x_1\}$

$[\ x_0^{(n)}, x_1^{(n)}, x_2^{(n)}, x_3^{(n)}\ ]$

n=0: [0, 0, 1, 0]

n=1 : [min(0,0.25·0+0.5), 0.1·0+0.5·0+0.4, 1, 0]

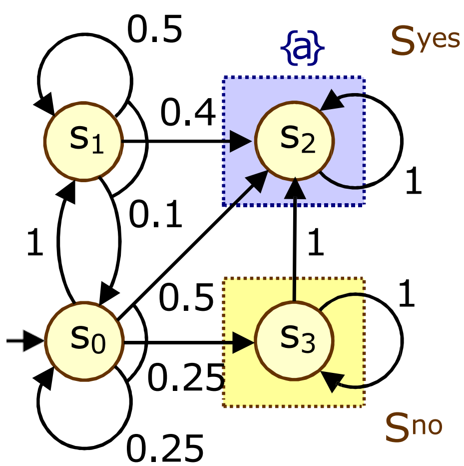
= [0, 0.4, 1, 0]

n=2 : [min(0.4,0.25·0+0.5), 0.1·0+0.5·0.4+0.4, 1, 0]

= [0.4, 0.6, 1, 0]

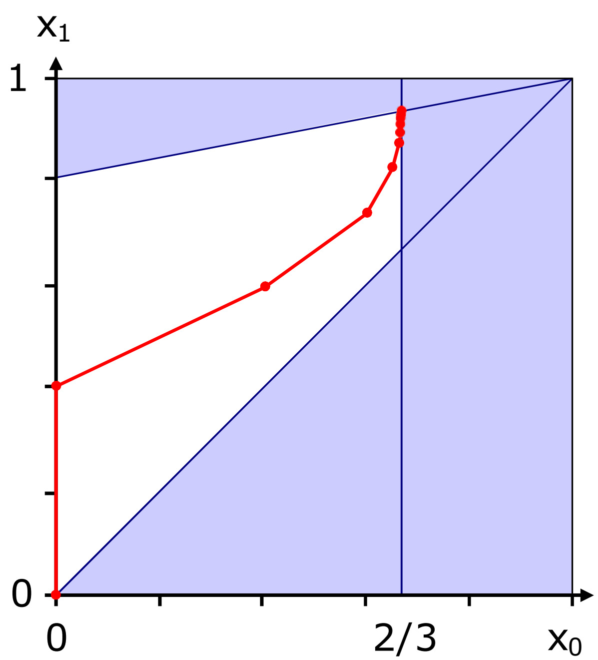
n=3: …

Example - PCTL Until (Value Iteration



	$[\ x_0^{(n)}, x_1^{(n)}, x_2^{(n)}, x_3^{(n)}\ ]$
n=0:	[0.000000, 0.000000, 1, 0]
n=1:	[0.000000, 0.400000, 1, 0]
n=2:	[0.400000, 0.600000, 1, 0]
n=3:	[0.600000, 0.740000, 1, 0]
n=4:	[0.650000, 0.830000, 1, 0]
n=5:	[0.662500, 0.880000, 1, 0]
n=6:	[0.665625, 0.906250, 1, 0]
n=7:	[0.666406, 0.919688, 1, 0]
n=8:	[0.666602, 0.926484, 1, 0]
n=9:	[0.666650, 0.929902, 1, 0]
	…
n=20 :	[0.666667, 0.933332, 1, 0]
n=21 :	[0.666667, 0.933332, 1, 0]
	≈ [2/3, 14/15, 1, 0]

Example - Value Iteration + LP



	$[\, x_0^{(n)}, x_1^{(n)}, x_2^{(n)}, x_3^{(n)} \,]$
n=0:	[0.000000, 0.000000, 1, 0]
n=1:	[0.000000, 0.400000, 1, 0]
n=2:	[0.400000, 0.600000, 1, 0]
n=3:	[0.600000, 0.740000, 1, 0]
n=4:	[0.650000, 0.830000, 1, 0]
n=5:	[0.662500, 0.880000, 1, 0]
n=6:	[0.665625, 0.906250, 1, 0]
n=7:	[0.666406, 0.919688, 1, 0]
n=8:	[0.666602, 0.926484, 1, 0]
n=9:	[0.666650, 0.929902, 1, 0]
	…
n=20:	[0.666667, 0.933332, 1, 0]
n=21:	[0.666667, 0.933332, 1, 0]
	≈ [2/3, 14/15, 1, 0]

PCTL Model Checking - Summary

- Computation of set Sat(Φ) for MDP M and PCTL formula Φ
  - recursive descent of parse tree
  - combination of graph algorithms, numerical computation
- Probabilistic operator P:
  - ○ Φ : one matrix-vector multiplication, $O(|S|^2)$
  - $\Phi_1 \, U^{\leq k} \, \Phi_2$: k matrix-vector multiplications, $O(k|S|^2)$
  - $\Phi_1 \, U \, \Phi_2$: linear programming problem, polynomial in |S| (assuming use of linear programming)
- Complexity:
  - linear in |Φ| and polynomial in |S|
  - S is states in MDP, assume |Steps(s)| is constant

Overview

- Basic concepts
- Probabilistic Model Checking
- Markov decision processes (MDPs)
- Adversaries
- PCTL
- PCTL model checking
- Costs and rewards

Costs and Rewards

- We augment DTMCs with rewards (or, conversely, costs)
  - real-valued quantities assigned to states and/or transitions
  - these can have a wide range of possible interpretations
- Some examples:
  - elapsed time, power consumption, size of message queue, number of messages successfully delivered, net profit, …
- Costs? or rewards?
  - mathematically, no distinction between rewards and costs
  - when interpreted, we assume that it is desirable to minimise costs and to maximise rewards
  - we will consistently use the terminology "rewards" regardless

Reward-Based Properties

- Properties of MDPs augmented with rewards
  - allow a wide range of quantitative measures of the system
  - basic notion: expected value of rewards
  - formal property specifications will be in an extension of PCTL
- More precisely, we use two distinct classes of property…
- Instantaneous properties
  - the expected value of the reward at some time point
- Cumulative properties
  - the expected cumulated reward over some period

PCTL and Rewards

- Extend PCTL to incorporate reward-based properties
  - add an R operator, which is similar to the existing P operator

expected reward is ~r

- φ ::= … | $P_{\sim p}$ [ψ] | $R_{\sim r}$ [$I^{=k}$] | $R_{\sim r}$ [$C^{\leq k}$] | $R_{\sim r}$ [$\Diamond\, \varphi$]

"instantaneous" "cumulative" "reachability"

  - where $r \in \mathbb{R}_{\geq 0}$, $\sim \in \{<,>,\leq,\geq\}$, $k \in \mathbb{N}$

- $R_{\sim r}$ [·] means "the expected value of · satisfies ~r"

Types of Reward Formulas

- Instantaneous: $R_{\sim r}$ [$I^{=k}$]
  - "the expected value of the state reward at time-step k is ~r"
  - e.g. "the expected queue size after exactly 90 seconds"

- Cumulative: $R_{\sim r}$ [$C^{\leq k}$]
  - "the expected reward cumulated up to time-step k is ~r"
  - e.g. "the expected power consumption over one hour"

- Reachability: $R_{\sim r}$ [◊ φ]
  - "the expected reward cumulated before reaching a state satisfying φ is ~r"
  - e.g. "the expected time for the algorithm to terminate"

Model Checking MDP Reward Formulas

- Instantaneous: $R_{\sim r}$ [$I^{=k}$]
  - — similar to the computation of bounded until probabilities
  - — solution of recursive equations

- Cumulative: $R_{\sim r}$ [$C^{\leq k}$]
  - — extension of bounded until computation
  - — solution of recursive equations

- Reachability: $R_{\sim r}$ [$\lozenge$ φ]
  - — similar to the case for P operator and until
  - — graph-based precomputation (identify ∞ - reward states)
  - — then linear programming problem (or value iteration)

Summary

- Basic concepts
  - Design and Validation
  - Formal Verification
  - Model Checking
- Probabilistic Model Checking
- Markov Decision Processes (MDPs)
  - probabilistic as well as nondeterministic behaviors
  - to model concurrency, underspecification, …
  - easy to model using guarded commands
- Adversaries Resolve Nondeterminism in an MDP
  - induce a probability space over paths
  - consider minimum/maximum probabilities over all adversaries
- Property Specifications
  - probabilistic extensions of temporal logic, e.g. PCTL
  - also: the expected value of costs/rewards
  - quantify overall adversaries
- Model Checking Algorithms
  - covered two basic techniques for MDPs: linear programming or value iteration