Probabilistic model checking

Marta Kwiatkowska

Department of Computer Science, University of Oxford

POPL 2015 tutorial, Mumbai, January 2015
What is probabilistic model checking?

- Probabilistic model checking...
  - is model checking applied to probabilistic models

- Probabilistic models...
  - can be derived from high-level specification or extracted from probabilistic programs
Model checking

System

Finite-state model

Temporal logic specification

¬EF fail

Model checker e.g. SMV, Spin

Result

Counter-example

System requirements
Probabilistic model checking

System

Probabilistic model
e.g. Markov chain

System requirements

Probabilistic temporal logic specification
e.g. PCTL, LTL

Result

Quantitative results

Counter-example

Probabilistic model checker
e.g. PRISM

$P_{<0.1} [F \text{ fail}]$
Why probability?

- Some systems are inherently probabilistic…
- **Randomisation**, e.g. in wireless coordination protocols
  - as a symmetry breaker
    
    ```
    bool short_delay = Bernoulli(0.5) // short or long delay
    ```
  - **Modelling uncertainty**
    - to quantify rate of failures
      
      ```
      bool fail = Bernoulli(0.001) // success wp 0.999 or failure
      ```
- **Modelling performance and biological processes**
  - reactions occurring between large numbers of molecules are naturally modelled in a stochastic fashion
    
    ```
    float binding_rate = exp(2.5) // exponentially distributed
    ```
Probability example

• Modelling a 6–sided die using a fair coin
  – algorithm due to Knuth/Yao:
  – start at 0, toss a coin
  – upper branch when H
  – lower branch when T
  – repeat until value chosen

• Probability of obtaining a 4?
  – THH, TTHH, TTTTHH, ...
  – Pr("eventually 4")
    \[ = \frac{1}{2}^3 + \frac{1}{2}^5 + \frac{1}{2}^7 + \ldots = \frac{1}{6} \]
  – expected number of coin flips needed = 11/3
  – NB termination guaranteed
Probabilistic models

dtmc
module die
// local state s : [0..7] init 0;
// value of the dice d : [0..6] init 0;
[] s=0 -> 0.5 : (s'=1) + 0.5 : (s'=2);
...
[] s=3 ->
  0.5 : (s'=1) + 0.5 : (s'=7) & (d'=1);
[] s=4 ->
  0.5 : (s'=7) & (d'=2) + 0.5 : (s'=7) & (d'=3);
...
[] s=7 -> (s'=7);
endmodule
rewards "coin_flips"
[] s<7 : 1;
endrewards

• Given in PRISM’s guarded commands modelling notation
int s, d;

s = 0; d = 0;
while (s < 7) {
    bool coin = Bernoulli(0.5);
    if (s == 0)
        if (coin) s = 1 else s = 2;
    ...
    else if (s == 3)
        if (coin) s = 1 else {s = 7; d = 1;}
    else if (s == 4)
        if (coin) {s = 7; d = 2} else {s = 7; d = 3;}
    ...
} return (d)

- Given as a (loopy) probabilistic program
Relation to programming languages

• **Probabilistic model checking (PMC)**
  – probabilistic models, state based, where transition relation is probabilistic
  – nonterminating behaviour
  – focus on *computing probability or expectation* of an event, or repeated events, typically via numerical methods
  – considers models with *nondeterminism*

• **Probabilistic programming (PP)**
  – imperative or functional programming extended with random assignment, interpreted as *distribution transformers*
  – terminating behaviour
  – focus on *probabilistic inference* (computing representation of the denoted probability distribution), typically via sampling
  – no nondeterminism, but *conditioning* on observations
• Excellent potential for cross-fertilisation
  – PMC and PP different communities
  – yet shared models (Markov chains) and methods (symbolic MTBDD/ADD-based solvers)

• PMC: maturing field
  – variety of models, incl. nondeterministic, timed, hybrid, etc
  – good for compact model representations, efficient automata-based and controller synthesis methods
  – can benefit from machine learning, cf ATVA 2014

• PP: emerging field
  – variety of efficient sampling-based MC methods
  – good for representing and computing distributions
  – can benefit from nondeterminism, useful for under-specification and input nondeterminism

Outline

0. Motivation

1. **Model checking for discrete-time Markov chains**
   - Definition, paths & probability spaces
   - PCTL model checking
   - Costs and rewards

2. **Model checking for Markov decision processes**
   - Definition & adversaries
   - PCTL model checking
   - Note on LTL model checking

3. **Probabilistic programs as Markov decision processes**
   - How to verify probabilistic programs

4. **PRISM**
   - Functionality, supported models and logics

5. **Summary and further reading**
Part 1

Discrete-time Markov chains
Discrete-time Markov chains

- **Discrete-time Markov chains (DTMCs)**
  - state-transition systems augmented with probabilities

- **States**
  - discrete set of states representing possible configurations of the system being modelled

- **Transitions**
  - transitions between states occur in discrete time-steps

- **Probabilities**
  - probability of making transitions between states is given by discrete probability distributions
Discrete-time Markov chains

- Formally, a DTMC $D$ is a tuple $(S, s_{\text{init}}, P, L)$ where:
  - $S$ is a finite set of states ("state space")
  - $s_{\text{init}} \in S$ is the initial state
  - $P : S \times S \rightarrow [0,1]$ is the transition probability matrix
    where $\sum_{s' \in S} P(s,s') = 1$ for all $s \in S$
  - $L : S \rightarrow 2^{\text{AP}}$ is function labelling states with atomic propositions

- Note: no deadlock states
  - i.e. every state has at least one outgoing transition
  - terminating behaviour represented by adding self loops
Simple DTMC example

\[D = (S, s_{\text{init}}, P, L)\]

\[S = \{s_0, s_1, s_2, s_3\}\]
\[s_{\text{init}} = s_0\]

\[P = \begin{bmatrix}
0 & 1 & 0 & 0 \\
0 & 0.01 & 0.01 & 0.98 \\
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}\]

\[AP = \{\text{try, fail, succ}\}\]
\[L(s_0) = \emptyset,\]
\[L(s_1) = \{\text{try}\},\]
\[L(s_2) = \{\text{fail}\},\]
\[L(s_3) = \{\text{succ}\}\]
DTMCs: An alternative definition

- **Alternative definition**... a DTMC is:
  - a family of random variables \( \{ X(k) \mid k=0,1,2,... \} \)
  - where \( X(k) \) are observations at discrete time-steps
  - i.e. \( X(k) \) is the state of the system at time-step \( k \)
  - which satisfies...

- **The Markov property** ("memorylessness")
  - \( \Pr( X(k)=s_k \mid X(k-1)=s_{k-1}, ..., X(0)=s_0 ) \)
  - \( = \Pr( X(k)=s_k \mid X(k-1)=s_{k-1} ) \)
  - for a given current state, future states are independent of past

- **This allows us to adopt the “state-based” view presented so far** (which is better suited to this context)
Other assumptions made here

• We consider time-homogenous DTMCs
  – transition probabilities are independent of time
  – \( P(s_{k-1}, s_k) = \Pr( X(k) = s_k \mid X(k-1) = s_{k-1} ) \)
  – otherwise: time-inhomogenous

• We will (mostly) assume that the state space \( S \) is finite
  – in general, \( S \) can be any countable set

• Initial state \( s_{\text{init}} \in S \) can be generalised…
  – to an initial probability distribution \( s_{\text{init}} : S \to [0,1] \)

• Transition probabilities are reals: \( P(s, s') \in [0,1] \)
  – but for algorithmic purposes, are assumed to be rationals
Paths and probabilities

- A (finite or infinite) path through a DTMC
  - is a sequence of states $s_0s_1s_2s_3...$ such that $P(s_i,s_{i+1}) > 0 \; \forall i$
  - represents an execution (i.e. one possible behaviour) of the system which the DTMC is modelling

- To reason (quantitatively) about this system
  - need to define a probability space over paths

- Intuitively:
  - sample space: $\text{Path}(s) =$ set of all infinite paths from a state $s$
  - events: sets of infinite paths from $s$
  - basic events: cylinder sets (or “cones”)
  - cylinder set $C(\omega)$, for a finite path $\omega$
    - set of infinite paths with the common finite prefix $\omega$
  - for example: $C(ss_1s_2)$
Probability space over paths

- **Sample space** $\Omega = \text{Path}(s)$
  - set of infinite paths with initial state $s$

- **Event set** $\Sigma_{\text{Path}(s)}$
  - the **cylinder set** $C(\omega) = \{ \omega' \in \text{Path}(s) \mid \omega \text{ is prefix of } \omega' \}$
  - $\Sigma_{\text{Path}(s)}$ is the **least $\sigma$-algebra** on $\text{Path}(s)$ containing $C(\omega)$ for all finite paths $\omega$ starting in $s$

- **Probability measure** $\Pr_s$
  - define probability $P_s(\omega)$ for finite path $\omega = ss_1...s_n$ as:
    - $P_s(\omega) = 1$ if $\omega$ has length one (i.e. $\omega = s$)
    - $P_s(\omega) = P(s,s_1) \cdot ... \cdot P(s_{n-1},s_n)$ otherwise
  - define $\Pr_s(C(\omega)) = P_s(\omega)$ for all finite paths $\omega$
  - $\Pr_s$ extends **uniquely** to a probability measure $\Pr_s : \Sigma_{\text{Path}(s)} \rightarrow [0,1]$

- See [KSK76] for further details
- Can also derive the probability space for finite and infinite sequences
Probability space – Example

- **Paths where sending fails the first time**
  - \( \omega = s_0s_1s_2 \)
  - \( C(\omega) = \) all paths starting \( s_0s_1s_2 \ldots \)
  - \( P_{s_0}(\omega) = P(s_0, s_1) \cdot P(s_1, s_2) = 1 \cdot 0.01 = 0.01 \)
  - \( Pr_{s_0}(C(\omega)) = P_{s_0}(\omega) = 0.01 \)

- **Paths which are eventually successful and with no failures**
  - \( C(s_0s_1s_3) \cup C(s_0s_1s_1s_3) \cup C(s_0s_1s_1s_1s_3) \cup \ldots \)
  - \( Pr_{s_0}( C(s_0s_1s_3) \cup C(s_0s_1s_1s_3) \cup C(s_0s_1s_1s_1s_3) \cup \ldots ) \)
    - \( = P_{s_0}(s_0s_1s_3) + P_{s_0}(s_0s_1s_1s_3) + P_{s_0}(s_0s_1s_1s_1s_3) + \ldots \)
    - \( = 1 \cdot 0.98 + 1 \cdot 0.01 \cdot 0.98 + 1 \cdot 0.01 \cdot 0.01 \cdot 0.98 + \ldots \)
    - \( = 0.9898989898 \ldots = 98/99 \)
PCTL

• Temporal logic for describing properties of DTMCs
  – PCTL = Probabilistic Computation Tree Logic [HJ94]
  – essentially the same as the logic pCTL of [ASB+95]

• Extension of (non–probabilistic) temporal logic CTL
  – key addition is probabilistic operator $P$
  – quantitative extension of CTL’s A and E operators

• Example
  – send → $P \geq 0.95 \ [\text{true} \ U \leq 10 \ \text{deliver}]$
  – “if a message is sent, then the probability of it being delivered within 10 steps is at least 0.95”
PCTL syntax

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

  - $\psi ::= X \phi \mid \phi U^{\leq k} \phi \mid \phi U \phi$ (path formulas)

- define $F \phi \equiv \text{true} U \phi$ (eventually), $G \phi \equiv \neg (F \neg \phi)$ (globally)

- 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 DTMCs

• PCTL formulas interpreted over states of a DTMC
  – s ⊨ φ denotes φ is “true in state s” or “satisfied in state s”

• Semantics of (non–probabilistic) state formulas:
  – for a state s:
    – s ⊨ a ⇔ a ∈ L(s)
    – s ⊨ φ₁ ∧ φ₂ ⇔ s ⊨ φ₁ and s ⊨ φ₂
    – s ⊨ ¬φ ⇔ s ⊨ φ is false

• Semantics of path formulas:
  – for a path ω = s₀s₁s₂...:
    – ω ⊨ X φ ⇔ s₁ ⊨ φ
    – ω ⊨ φ₁ U φ₂ ⇔ ∃ i such that sᵢ ⊨ φ₂ and ∀ j < i, sⱼ ⊨ φ₁
PCTL semantics for DTMCs

- **Semantics of the probabilistic operator P**
  - Informal definition: $s \models P_\sim p [ \psi ]$ means that “the probability, from state $s$, that $\psi$ is true for an outgoing path satisfies $\sim p$”
  - Example: $s \models P_{<0.25} [ X \text{ fail } ] \iff$ “the probability of atomic proposition fail being true in the next state of outgoing paths from $s$ is less than 0.25”
  - Formally: $s \models P_\sim p [ \psi ] \iff \text{Prob}(s, \psi) \sim p$
  - Where: $\text{Prob}(s, \psi) = \Pr_s \{ \omega \in \text{Path}(s) \mid \omega \models \psi \}$
  - (Sets of paths satisfying $\psi$ are always measurable [Var85])
Quantitative properties

- Consider a PCTL formula $P_{p \sim p} [ \psi ]$
  - if the probability is unknown, how to choose the bound $p$?
- When the outermost operator of a PTCL formula is $P$
  - we allow the form $P =? [ \psi ]$
  - “what is the probability that path formula $\psi$ is true?”
- Model checking is no harder: compute the values anyway
- Useful to spot patterns, trends

Example
  - $P =? [ F \text{err/total}>0.1 ]$
  - “what is the probability that 10% of the NAND gate outputs are erroneous?”
PCTL model checking for DTMCs

- Algorithm for PCTL model checking [CY88,HJ94,CY95]
  - inputs: DTMC $D=\langle S, s_{\text{init}}, P, L \rangle$, PCTL formula $\phi$
  - output: $\text{Sat}(\phi) = \{ s \in S \mid s \models \phi \} = \text{set of states satisfying } \phi$

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

- Sometimes, focus on quantitative results
  - e.g. compute result of $P=? [ F \text{ error } ]$
  - e.g. compute result of $P=? [ F_{\leq k} \text{ error } ]$ for $0 \leq k \leq 100$
PCTL model checking for DTMCs

- Basic algorithm proceeds by induction on parse tree of $\phi$
  - example: $\phi = (\neg\text{fail} \land \text{try}) \rightarrow P_{>0.95} [\neg\text{fail} \cup \text{succ}]$

- For the non-probabilistic operators:
  - $\text{Sat}(\text{true}) = S$
  - $\text{Sat}(a) = \{ s \in S \mid a \in L(s) \}$
  - $\text{Sat}(\neg\phi) = S \setminus \text{Sat}(\phi)$
  - $\text{Sat}(\phi_1 \land \phi_2) = \text{Sat}(\phi_1) \cap \text{Sat}(\phi_2)$

- For the $P_{\neg p} [\psi]$ operator
  - need to compute the probabilities $\text{Prob}(s, \psi)$ for all states $s \in S$
  - focus here on “until” case: $\psi = \phi_1 \cup \phi_2$
PCTL until for DTMCs

- Computation of probabilities $\text{Prob}(s, \phi_1 \ U \phi_2)$ for all $s \in S$
- First, identify all states where the probability is 1 or 0
  - $S^{\text{yes}} = \text{Sat}(P_{\geq 1} [ \phi_1 \ U \phi_2 ])$
  - $S^{\text{no}} = \text{Sat}(P_{\leq 0} [ \phi_1 \ U \phi_2 ])$
- Then solve linear equation system for remaining states

- We refer to the first phase as “precomputation”
  - two algorithms: $\text{Prob}_0$ (for $S^{\text{no}}$) and $\text{Prob}_1$ (for $S^{\text{yes}}$)
  - algorithms work on underlying graph (probabilities irrelevant)
- Important for several reasons
  - reduces the set of states for which probabilities must be computed numerically (which is more expensive)
  - gives exact results for the states in $S^{\text{yes}}$ and $S^{\text{no}}$ (no round-off)
  - for $P_{\sim p}[\cdot]$ where $p$ is 0 or 1, no further computation required
PCTL until – Linear equations

• Probabilities \( \text{Prob}(s, \phi_1 U \phi_2) \) can now be obtained as the unique solution of the following set of linear equations:

\[
\text{Prob}(s, \phi_1 U \phi_2) = \begin{cases} 
1 & \text{if } s \in S^\text{yes} \\
0 & \text{if } s \in S^\text{no} \\
\sum_{s' \in S} P(s,s') \cdot \text{Prob}(s', \phi_1 U \phi_2) & \text{otherwise}
\end{cases}
\]

– can be reduced to a system in \( |S^?| \) unknowns instead of \( |S| \)
where \( S^? = S \setminus (S^\text{yes} \cup S^\text{no}) \)

• This can be solved with (a variety of) standard techniques
  – direct methods, e.g. Gaussian elimination
  – iterative methods, e.g. Jacobi, Gauss–Seidel, ...
    (preferred in practice due to scalability)
  – PRISM works with a compact MTBDD–based matrix
PCTL until – Example

- Example: $P_{>0.8} [\neg a U b ]$
PCTL until – Example

- Example: $P_{>0.8} [\neg a \mathbf{U} b ]$

\[ S_{no} = Sat(P_{\leq 0} [\neg a \mathbf{U} b ]) \]

\[ S_{yes} = Sat(P_{\geq 1} [\neg a \mathbf{U} b ]) \]
**PCTL until – Example**

- Example: $P_{>0.8} [\neg a \cup b ]$

- Let $x_s = \text{Prob}(s, \neg a \cup b)$

- Solve:
  
  $x_4 = x_5 = 1$
  
  $x_1 = x_3 = 0$
  
  $x_0 = 0.1x_1 + 0.9x_2 = 0.8$
  
  $x_2 = 0.1x_2 + 0.1x_3 + 0.3x_5 + 0.5x_4 = \frac{8}{9}$

  $\text{Prob}(\neg a \cup b) = x = [0.8, 0, \frac{8}{9}, 0, 1, 1]$

  $\text{Sat}(P_{\geq 1} [\neg a \cup b ]) = \{ s_2, s_4, s_5 \}$

$S_{\text{no}} =$ Sat($P_{\leq 0} [\neg a \cup b ]$)

$S_{\text{yes}} =$ Sat($P_{\geq 1} [\neg a \cup b ]$)
PCTL model checking – Summary

- **Computation of set Sat(Φ) for DTMC D and PCTL formula \( Φ \)**
  - recursive descent of parse tree
  - combination of graph algorithms, numerical computation

- **Probabilistic operator \( P \):**
  - \( X \ Φ \) : one matrix–vector multiplication, \( O(|S|^2) \)
  - \( Φ_1 U^{≤k} Φ_2 \) : \( k \) matrix–vector multiplications, \( O(k|S|^2) \)
  - \( Φ_1 U Φ_2 \) : linear equation system, at most \( |S| \) variables, \( O(|S|^3) \)

- **Complexity:**
  - linear in \( |Φ| \) and polynomial in \( |S| \)
• We augment DTMCs with rewards (or, conversely, costs)
  – real-valued quantities assigned to states and/or transitions
  – allow a wide range of quantitative measures of the system
  – basic notion: expected value of rewards (or costs)
  – 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
Rewards in the PRISM language

- **Rewards** “total_queue_size”
  
  true : queue1 + queue2;
  
  endrewards

  (instantaneous, state rewards)

- **Rewards** “time”
  
  true : 1;
  
  endrewards

  (cumulative, state rewards)

- **Rewards** "dropped"
  
  [receive] q=q_max : 1;
  
  endrewards

  (cumulative, transition rewards)

  (q = queue size, q_max = max. queue size, receive = action label)

- **Rewards** “power”
  
  sleep=true : 0.25;
  
  sleep=false : 1.2 * up;
  
  [wake] true : 3.2;
  
  endrewards

  (cumulative, state/trans. rewards)

  (up = num. operational components, wake = action label)
DTMC reward structures

For a DTMC $(S, s_{\text{init}}, P, L)$, a reward structure is a pair $(\rho, \iota)$
- $\rho : S \rightarrow \mathbb{R}_{\geq 0}$ is the state reward function (vector)
- $\iota : S \times S \rightarrow \mathbb{R}_{\geq 0}$ is the transition reward function (matrix)

Example (for use with instantaneous properties)
- “size of message queue”: $\rho$ maps each state to the number of jobs in the queue in that state, $\iota$ is not used

Examples (for use with cumulative properties)
- “time-steps”: $\rho$ returns 1 for all states and $\iota$ is zero (equivalently, $\rho$ is zero and $\iota$ returns 1 for all transitions)
- “number of messages lost”: $\rho$ is zero and $\iota$ maps transitions corresponding to a message loss to 1
- “power consumption”: $\rho$ is defined as the per-time-step energy consumption in each state and $\iota$ as the energy cost of each transition
- **Extend PCTL to incorporate reward-based properties**
  - add an R operator, which is similar to the existing P operator

\[
\[ \phi ::= \ldots \mid P\_p[\psi] \mid R\_r[I=k] \mid R\_r[C\leq k] \mid R\_r[F\phi] \]
\]

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

- \( R\_r[\cdot] \) means “the expected value of \( \cdot \) satisfies \( \sim r \)”
Reward formula semantics

- **Formal semantics of the three reward operators**
  - based on random variables over (infinite) paths

- **Recall**:
  - \( s \models P_{\sim p}[\psi] \iff \Pr_s \{ \omega \in \text{Path}(s) \mid \omega \models \psi \} \sim p \)

- **For a state \( s \) in the DTMC (see [KNP07a] for full definition)**:
  - \( s \models R_{\sim r}[I^{=k}] \iff \text{Exp}(s, X_{I^{=k}}) \sim r \)
  - \( s \models R_{\sim r}[C^{\leq k}] \iff \text{Exp}(s, X_{C^{\leq k}}) \sim r \)
  - \( s \models R_{\sim r}[F \Phi] \iff \text{Exp}(s, X_{F\Phi}) \sim r \)

where: \( \text{Exp}(s, X) \) denotes the *expectation* of the random variable \( X : \text{Path}(s) \to \mathbb{R}_{\geq 0} \) with respect to the *probability measure* \( \Pr_s \)
Reward formula semantics

- **Definition of random variables:**
  - for an infinite path $\omega = s_0 s_1 s_2 ...$

  $$X_{\downarrow k}(\omega) = \rho(s_k)$$

  $$X_{C_{\leq k}}(\omega) = \begin{cases} 0 & \text{if } k = 0 \\ \sum_{i=0}^{k-1} \rho(s_i) + \iota(s_i, s_{i+1}) & \text{otherwise} \end{cases}$$

  $$X_{F_{\phi}}(\omega) = \begin{cases} 0 & \text{if } s_0 \in \text{Sat}(\phi) \\ \infty & \text{if } s_i \notin \text{Sat}(\phi) \text{ for all } i \geq 0 \\ \sum_{i=0}^{k_{\phi}-1} \rho(s_i) + \iota(s_i, s_{i+1}) & \text{otherwise} \end{cases}$$

  - where $k_{\phi} = \min \{ j \mid s_j \models \phi \}$
Model checking reward properties

- **Instantaneous:** $R_{\sim r} [ \text{I} = k ]$
- **Cumulative:** $R_{\sim r} [ \text{C} \leq k ]$
  - variant of the method for computing bounded until probabilities (not discussed)
  - solution of recursive equations

- **Reachability:** $R_{\sim r} [ \text{F} \phi ]$
  - similar to computing until probabilities
  - precomputation phase (identify infinite reward states)
  - then reduces to solving a system of linear equation

- For more details, see e.g. [KNP07a]
  - complexity not increased wrt classical PCTL
Part 2

Markov decision processes
Recap: Discrete–time Markov chains

- Discrete–time Markov chains (DTMCs)
  - state–transition systems augmented with probabilities
- Formally: DTMC $D = (S, s_{\text{init}}, P, L)$ where:
  - $S$ is a set of states and $s_{\text{init}} \in S$ is the initial state
  - $P : S \times S \rightarrow [0,1]$ is the transition probability matrix
  - $L : S \rightarrow 2^{\text{AP}}$ labels states with atomic propositions
  - define a probability space $Pr_s$ over paths $\text{Path}_s$

- Properties of DTMCs
  - can be captured by the logic PCTL
  - e.g. $\text{send} \rightarrow P_{\geq 0.95} [ F \text{ deliver} ]$
  - key question: what is the probability of reaching states $T \subseteq S$ from state $s$?
  - reduces to graph analysis + linear equation system
Nondeterminism

- Some aspects of a system may not be probabilistic and should not be modelled probabilistically; for example:

  - **Concurrency** – scheduling of parallel components
    - e.g. randomised distributed algorithms – multiple probabilistic processes operating **asynchronously**

  - **Underspecification** – unknown model parameters
    - e.g. a probabilistic communication protocol designed for message propagation delays of between $d_{\text{min}}$ and $d_{\text{max}}$

  - **Unknown environments** – unknown inputs
    - e.g. probabilistic security protocols – unknown adversary
Markov decision processes

- Markov decision processes (MDPs)
  - extension of DTMCs which allow nondeterministic choice

- Like DTMCs:
  - discrete set of states representing possible configurations of the system being modelled
  - transitions between states occur in discrete time-steps

- Probabilities and nondeterminism
  - in each state, a nondeterministic choice between several discrete probability distributions over successor states

\[s_0 \xrightarrow{0.7} s_1 \quad \quad s_1 \xrightarrow{0.5} s_2 \quad \quad s_1 \xrightarrow{0.3} s_3\]

\[s_2 \xrightarrow{0.5} s_1 \quad \quad s_3 \xrightarrow{1} s_3\]

\[\{\text{init}\} \xrightarrow{1} s_0 \quad \quad s_0 \xrightarrow{1} \{\text{heads}\}\]

\[\{\text{tails}\}\]

\[s_0 \rightarrow 0.5 s_1 \quad \quad s_1 \rightarrow 0.5 s_2 \quad \quad s_1 \rightarrow 0.5 s_3\]
Markov decision processes

- Formally, an MDP $M$ is a tuple $(S, s_{\text{init}}, \alpha, \delta, L)$ where:
  - $S$ is a set of states (“state space”)
  - $s_{\text{init}} \in S$ is the initial state
  - $\alpha$ is an alphabet of action labels
  - $\delta \subseteq S \times \alpha \times \text{Dist}(S)$ is the transition probability relation, where $\text{Dist}(S)$ is the set of all discrete probability distributions over $S$
  - $L : S \rightarrow 2^{\text{AP}}$ is a labelling with atomic propositions

- Notes:
  - we also abuse notation and use $\delta$ as a function
  - i.e. $\delta : S \rightarrow 2^{\alpha \times \text{Dist}(S)}$ where $\delta(s) = \{ (a, \mu) \mid (s, a, \mu) \in \delta \}$
  - we assume $\delta(s)$ is always non-empty, i.e. no deadlocks
  - MDPs, here, are identical to probabilistic automata [Segala]
    - usually, MDPs take the form: $\delta : S \times \alpha \rightarrow \text{Dist}(S)$
Simple MDP example

- A 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

```
s0 ----start----> s1 ----send----> s2 ----{fail}----> restart
          |               |               |
          v               v               v
          0.01             0.99           1

s1 ----{try}----> s0

s2 ----{fail}----> restart

s3 ----{succ}----> s1
```

- 1
- 0.01
- 0.99
- 1
- 1
Asynchronous parallel composition of two 3-state DTMCs

Action labels omitted here
Paths and strategies

• **A (finite or infinite) path through an MDP**
  - is a sequence \((s_0...s_n)\) of (connected) states
  - represents an execution of the system
  - resolves both the probabilistic and nondeterministic choices

• **A strategy \(\sigma\) (aka. “adversary” or “policy”) of an MDP**
  - is a resolution of nondeterminism only
  - is (formally) a mapping from finite paths to distributions on action–distribution pairs
  - induces a fully probabilistic model
  - i.e. an (infinite–state) Markov chain over finite paths
  - on which we can define a probability space over infinite paths
Classification of strategies

- Strategies are classified according to
  - randomisation:
    - \( \sigma \) is deterministic (pure) if \( \sigma(s_0...s_n) \) is a point distribution, and randomised otherwise
  - memory:
    - \( \sigma \) is memoryless (simple) if \( \sigma(s_0...s_n) = \sigma(s_n) \) for all \( s_0...s_n \)
    - \( \sigma \) is finite memory if there are finitely many modes such as \( \sigma(s_0...s_n) \) depends only on \( s_n \) and the current mode, which is updated each time an action is performed
    - otherwise, \( \sigma \) is infinite memory

- A strategy \( \sigma \) induces, for each state \( s \) in the MDP:
  - a set of infinite paths \( \text{Path}^{\sigma}(s) \)
  - a probability space \( \text{Pr}^{\sigma}_s \) over \( \text{Path}^{\sigma}(s) \)
Example strategy

- Fragment of induced Markov chain for strategy which picks \(b\) then \(c\) in \(s_1\)

![Diagram of the Markov chain]

finite-memory, deterministic
PCTL

• Temporal logic for properties of MDPs (and DTMCs)
  – extension of (non–probabilistic) temporal logic CTL
  – key addition is probabilistic operator $P$
  – quantitative extension of CTL’s A and E operators

• PCTL syntax:
  – $\phi ::= \text{true} \mid a \mid \phi \land \phi \mid \neg \phi \mid P_{\sim p} [\psi]$ (state formulas)
  – $\psi ::= X \phi \mid \phi U_{\leq k} \phi \mid \phi U \phi$ (path formulas)
  – where $a$ is an atomic proposition, used to identify states of interest, $p \in [0,1]$ is a probability, $\sim \in \{<,>,\leq,\geq\}$, $k \in \mathbb{N}$

• Example: $\text{send} \rightarrow P_{\geq 0.95} [\text{true U}_{\leq 10} \text{deliver}]$
PCTL semantics for MDPs

- **Semantics of the probabilistic operator $P$**
  - can only define probabilities for a specific strategy $\sigma$
  - $s \models P_{\sim p} [\psi]$ means “the probability, from state $s$, that $\psi$ is true for an outgoing path satisfies $\sim p$ for all strategies $\sigma$”
  - formally $s \models P_{\sim p} [\psi] \Leftrightarrow \Pr_s^\sigma(\psi) \sim p$ for all strategies $\sigma$
  - where we use $Pr_s^\sigma(\psi)$ to denote $Pr_s^\sigma \{ \omega \in \text{Path}_s^\sigma | \omega \models \psi \}$
Minimum and maximum probabilities

- Letting:
  - $\Pr_s^{\max}(\psi) = \sup_{\sigma} \Pr_s^{\sigma}(\psi)$
  - $\Pr_s^{\min}(\psi) = \inf_{\sigma} \Pr_s^{\sigma}(\psi)$

- We have:
  - if $\sim \in \{\geq, >\}$, then $s \vDash P_{\sim p}[\psi] \iff \Pr_s^{\min}(\psi) \sim p$
  - if $\sim \in \{<, \leq\}$, then $s \vDash P_{\sim p}[\psi] \iff \Pr_s^{\max}(\psi) \sim p$

- Model checking $P_{\sim p}[\psi]$ reduces to the computation over all strategies of either:
  - the minimum probability of $\psi$ holding
  - the maximum probability of $\psi$ holding

- Crucial result for model checking PCTL until on MDPs
  - memoryless strategies suffice, i.e. there are always memoryless strategies $\sigma_{\min}$ and $\sigma_{\max}$ for which:
    - $\Pr_s^{\sigma_{\min}}(\psi) = \Pr_s^{\min}(\psi)$ and $\Pr_s^{\sigma_{\max}}(\psi) = \Pr_s^{\min}(\psi)$
Quantitative properties

• For PCTL properties with P as the outermost operator
  – quantitative form (two types): $P_{\min}=\? [ \psi ]$ and $P_{\max}=\? [ \psi ]$
  – i.e. “what is the minimum/maximum probability (over all adversaries) that path formula $\psi$ 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 $Pr_s^{\min}(\psi)$ or $Pr_s^{\max}(\psi)$ anyway
  – useful to spot patterns/trends

• Example: CSMA/CD protocol
  – “min/max probability that a message is sent within the deadline”
PCTL model checking for MDPs

- **Algorithm for PCTL model checking** [BdA95]
  - inputs: MDP $M= (S, s_{init}, \alpha, \delta, L)$, PCTL formula $\phi$
  - output: $Sat(\phi) = \{ s \in S \mid s \models \phi \} = \text{set of states satisfying } \phi$

- **Basic algorithm same as PCTL model checking for DTMCs**
  - proceeds by induction on parse tree of $\phi$
  - non-probabilistic operators (true, $a$, $\neg$, $\land$) straightforward

- **Only need to consider $P_{\sim p}[\psi]$ formulas**
  - reduces to computation of $Pr_s^{\min}(\psi)$ or $Pr_s^{\max}(\psi)$ for all $s \in S$
  - dependent on whether $\sim \in \{\geq, >\}$ or $\sim \in \{<, \leq\}$
  - these slides cover the case $Pr_s^{\min}(\phi_1 U \phi_2)$, i.e. $\sim \in \{\geq, >\}$
  - case for maximum probabilities is very similar
PCTL until for MDPs

- Computation of probabilities $\Pr_s^{\min}(\phi_1 \ U \phi_2)$ for all $s \in S$
- First identify all states where the probability is 1 or 0
  - “precomputation” algorithms, yielding sets $S^{\text{yes}}, S^{\text{no}}$
- Then compute (min) probabilities for remaining states ($S^?$)
  - either: solve linear programming problem
  - or: approximate with an iterative solution method
  - or: use policy iteration

Example:

$$P_{\geq p}[F \ a] \equiv P_{\geq p}[\text{true } U \ a]$$
PCTL until – Precomputation

- Identify all states where $\text{Pr}_{s}^{\text{min}}(\phi_1 \cup \phi_2)$ is 1 or 0
  - $S_{\text{yes}} = \text{Sat}(P_{\geq 1} [ \phi_1 \cup \phi_2 ])$, $S_{\text{no}} = \text{Sat}(\neg P_{>0} [ \phi_1 \cup \phi_2 ])$

- Two graph-based precomputation algorithms:
  - algorithm Prob1A computes $S_{\text{yes}}$
    - for all strategies the probability of satisfying $\phi_1 \cup \phi_2$ is 1
  - algorithm Prob0E computes $S_{\text{no}}$
    - there exists a strategy for which the probability is 0

Example:
$P_{\geq p} [ F a ]$

Diagram:

\[ S_{\text{yes}} = \text{Sat}(P_{\geq 1} [ F a ]) \]
\[ S_{\text{no}} = \text{Sat}(\neg P_{>0} [ F a ]) \]
Method 1 – Linear programming

- Probabilities $Pr_{s}^{\text{min}}(\phi_1 \cup \phi_2)$ for remaining states in the set $S^? = S \setminus (S^{\text{yes}} \cup S^{\text{no}})$ can be obtained as the unique solution of the following linear programming (LP) problem:

$$\text{maximize } \sum_{s \in S^?} x_s \text{ subject to the constraints:}$$
$$x_s \leq \sum_{s' \in S^?} \mu(s') \cdot x_{s'} + \sum_{s' \in S^{\text{yes}}} \mu(s')$$

for all $s \in S^?$ and for all $(a, \mu) \in \delta(s)$

- Simple case of a more general problem known as the stochastic shortest path problem [BT91]

- This can be solved with standard techniques
  - e.g. Simplex, ellipsoid method, branch–and–cut
Example – PCTL until (LP)

Let $x_i = \Pr_{s_i \text{min}}(F a)$

$S_{\text{yes}}$: $x_2 = 1$, $S_{\text{no}}$: $x_3 = 0$

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

Maximise $x_0 + x_1$ subject to constraints:

- $x_0 \leq x_1$
- $x_0 \leq 0.25 \cdot x_0 + 0.5$
- $x_1 \leq 0.1 \cdot x_0 + 0.5 \cdot x_1 + 0.4$
Let $x_i = \text{Pr}_{s_i}^{\text{min}}(F \, a)$

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

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 \( x_i = \text{Pr}_{s_i}^{\text{min}}(F a) \)

\( S^\text{yes} \): \( x_2 = 1 \), \( S^\text{no} \): \( x_3 = 0 \)

For \( S = \{x_0, x_1\} \):

Maximise \( x_0 + x_1 \) subject to constraints:
- \( x_0 \leq x_1 \)
- \( x_0 \leq \frac{2}{3} \)
- \( x_1 \leq 0.2 \cdot x_0 + 0.8 \)

Solution: \((x_0, x_1) = (\frac{2}{3}, \frac{14}{15})\)
Example – PCTL until (LP)

Let $x_i = \Pr_{s_i}^{\min}(F \ a)$

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

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

Maximise $x_0 + x_1$ subject to constraints:

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

Two memoryless adversaries
Method 2 – Value iteration

• For probabilities $Pr_s^{\text{min}}(\phi_1 \cup \phi_2)$ it can be shown that:

$$Pr_s^{\text{min}}(\phi_1 \cup \phi_2) = \lim_{n \to \infty} x_s^{(n)}$$

where:

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

• This forms the basis for an (approximate) iterative solution
  – iterations terminated when solution converges sufficiently
Compute: $\Pr_{S_i}^{\min}(F a)$

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

\[
\begin{array}{c}
n=0: \quad [0, 0, 1, 0] \\
n=1: \quad [\min(0, 0.25\cdot 0 + 0.5), 0.1\cdot 0 + 0.5\cdot 0 + 0.4, 1, 0] \\
&= [0, 0.4, 1, 0] \\
n=2: \quad [\min(0.4, 0.25\cdot 0 + 0.5), 0.1\cdot 0 + 0.5\cdot 0.4 + 0.4, 1, 0] \\
&= [0.4, 0.6, 1, 0] \\
n=3: \quad \ldots
\end{array}
\]
Example – PCTL until (value iteration)

<table>
<thead>
<tr>
<th>n</th>
<th>[x_0^{(n)}, x_1^{(n)}, x_2^{(n)}, x_3^{(n)}]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[0.000000, 0.000000, 1, 0]</td>
</tr>
<tr>
<td>1</td>
<td>[0.000000, 0.400000, 1, 0]</td>
</tr>
<tr>
<td>2</td>
<td>[0.400000, 0.600000, 1, 0]</td>
</tr>
<tr>
<td>3</td>
<td>[0.600000, 0.740000, 1, 0]</td>
</tr>
<tr>
<td>4</td>
<td>[0.650000, 0.830000, 1, 0]</td>
</tr>
<tr>
<td>5</td>
<td>[0.662500, 0.880000, 1, 0]</td>
</tr>
<tr>
<td>6</td>
<td>[0.665625, 0.906250, 1, 0]</td>
</tr>
<tr>
<td>7</td>
<td>[0.666406, 0.919688, 1, 0]</td>
</tr>
<tr>
<td>8</td>
<td>[0.666602, 0.926484, 1, 0]</td>
</tr>
<tr>
<td>9</td>
<td>[0.666650, 0.929902, 1, 0]</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>[0.666667, 0.933332, 1, 0]</td>
</tr>
<tr>
<td>21</td>
<td>[0.666667, 0.933332, 1, 0]</td>
</tr>
</tbody>
</table>

\[ \approx [\frac{2}{3}, \frac{14}{15}, 1, 0] \]
Example – Value iteration + LP

\[ \begin{align*}
    &n=0: & [0.000000, 0.000000, 1, 0] \\
    &n=1: & [0.000000, 0.400000, 1, 0] \\
    &n=2: & [0.400000, 0.600000, 1, 0] \\
    &n=3: & [0.600000, 0.740000, 1, 0] \\
    &n=4: & [0.650000, 0.830000, 1, 0] \\
    &n=5: & [0.662500, 0.880000, 1, 0] \\
    &n=6: & [0.665625, 0.906250, 1, 0] \\
    &n=7: & [0.666406, 0.919688, 1, 0] \\
    &n=8: & [0.666602, 0.926484, 1, 0] \\
    &n=9: & [0.666650, 0.929902, 1, 0] \\
    &\vdots & & \\
    &n=20: & [0.666667, 0.933332, 1, 0] \\
    &n=21: & [0.666667, 0.933332, 1, 0] \\
\end{align*} \]

\[ \approx [\frac{2}{3}, \frac{14}{15}, 1, 0] \]
Method 3 – Policy iteration

- **Value iteration:**
  - iterates over (vectors of) probabilities
- **Policy iteration:**
  - iterates over strategies (“policies”)

1. Start with an arbitrary (memoryless) strategy $\sigma$
2. Compute the reachability probabilities $Pr^\sigma(F_a)$ for $\sigma$
3. Improve the strategy in each state
4. Repeat 2/3 until no change in strategy

**Termination:**
- finite number of memoryless strategies
- improvement in (minimum) probabilities each time
Method 3 – Policy iteration

1. Start with an arbitrary (memoryless) strategy $\sigma$
   - pick an element of $\delta(s)$ for each state $s \in S$

2. Compute the reachability probabilities $Pr^\sigma(F_a)$ for $\sigma$
   - probabilistic reachability on a DTMC
   - i.e. solve linear equation system

3. Improve the strategy in each state

   $\sigma'(s) = \arg\min \left\{ \sum_{s' \in S} \mu(s') \cdot Pr_s^\sigma(F_a) \mid (a, \mu) \in \delta(s) \right\}$

4. Repeat 2/3 until no change in strategy
Example – Policy iteration

Arbitrary strategy $\sigma$:

Compute: $Pr^\sigma(Fa)$

Let $x_i = Pr^\sigma_{si}(Fa)$

$x_2=1$, $x_3=0$ and:

• $x_0 = x_1$
• $x_1 = 0.1 \cdot x_0 + 0.5 \cdot x_1 + 0.4$

Solution:

$Pr^\sigma(Fa) = [1, 1, 1, 0]$

Refine $\sigma$ in state $s_0$:

$\min\{1(1), 0.5(1)+0.25(0)+0.25(1)\}$

$= \min\{1, 0.75\} = 0.75$
Example – Policy iteration

Refined strategy $\sigma'$:

Compute: $Pr^{\sigma'}(F \, a)$

Let $x_i = Pr_{si}^{\sigma'}(F \, a)$

$x_2 = 1, \ x_3 = 0$ and:

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

Solution:

$Pr^{\sigma'}(F \, a) = [\frac{2}{3}, \frac{14}{15}, 1, 0]$

This is optimal
Example – Policy iteration

\[ x_1 = 0.2 \cdot x_0 + 0.8 \]

\[ x_0 = x_1 \]

\[ x_0 = 2/3 \]
PCTL model checking – Summary

• Computation of set $\text{Sat}(\Phi)$ for MDP $M$ and PCTL formula $\Phi$
  – recursive descent of parse tree
  – combination of graph algorithms, numerical computation

• Probabilistic operator $P$:
  – $X \Phi$ : one matrix–vector multiplication, $O(|S|^2)$
  – $\Phi_1 \cup_{\leq k} \Phi_2$ : $k$ matrix–vector multiplications, $O(k|S|^2)$
  – $\Phi_1 \cup \Phi_2$ : linear programming problem, polynomial in $|S|$ (assuming use of linear programming)

• Complexity:
  – linear in $|\Phi|$ and polynomial in $|S|$
  – $S$ is states in MDP, assume $|\delta(s)|$ is constant
Costs and rewards for MDPs

- We can augment MDPs 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

- Extend logic PCTL with R operator, for “expected reward”
  - as for PCTL, either $R_{\sim r} [ \ldots ]$, $R_{\min =?} [ \ldots ]$ or $R_{\max =?} [ \ldots ]$

- Some examples:
  - $R_{\min =?} [ I = 90 ]$, $R_{\max =?} [ C \leq 60 ]$, $R_{\max =?} [ F \text{ “end” } ]$
  - “the minimum expected queue size after exactly 90 seconds”
  - “the maximum expected power consumption over one hour”
  - the maximum expected time for the algorithm to terminate
Limitations of PCTL

- PCTL, although useful in practice, has limited expressivity
  - essentially: probability of reaching states in X, passing only through states in Y (and within k time-steps)

- More expressive logics can be used, for example:
  - LTL [Pnu77] – the non-probabilistic linear-time temporal logic
  - PCTL* [ASB+95,BdA95] – which subsumes both PCTL and LTL
    - both allow path operators to be combined

- In PCTL, temporal operators always appear inside $P_{\neg p} [...]$
  - (and, in CTL, they always appear inside A or E)
  - in LTL (and PCTL*), temporal operators can be combined
• Same idea as PCTL: probabilities of sets of path formulae
  – for a state $s$ of a DTMC and an LTL formula $\psi$:
  – $\text{Prob}(s, \psi) = \Pr_s \{ \omega \in \text{Path}(s) \mid \omega \models \psi \}$
  – all such path sets are measurable (see later)

• For MDPs, we can again consider lower/upper bounds
  – $p_{\min}(s, \psi) = \inf_{\sigma \in \text{Adv}} \text{Prob}^\sigma(s, \psi)$
  – $p_{\max}(s, \psi) = \sup_{\sigma \in \text{Adv}} \text{Prob}^\sigma(s, \psi)$
  – (for LTL formula $\psi$)

• For DTMCs or MDPs, an LTL specification often comprises an LTL (path) formula and a probability bound
  – e.g. $P_{>0.99} [ F ( \text{req} \land X \text{ack} ) ]$
LTL model checking for DTMCs

- Model check LTL specification $P_{\sim p}[\psi]$ against DTMC $D$

- 1. Generate a deterministic Rabin automaton (DRA) for $\psi$
  - build nondeterministic Büchi automaton (NBA) for $\psi$ [VW94]
  - convert the NBA to a DRA [Saf88]

- 2. Construct product DTMC $D \otimes A$

- 3. Identify accepting BSCCs of $D \otimes A$

- 4. Compute probability of reaching accepting BSCCs
  - from all states of the $D \otimes A$

- 5. Compare probability for $\langle s, q_s \rangle$ against $p$ for each $s$

- Qualitative LTL model checking – no probabilities needed
**PCTL* model checking**

- **PCTL* syntax:**
  - \( \phi ::= \text{true} \mid a \mid \phi \land \phi \mid \neg \phi \mid P_{\neg p}[\psi] \)
  - \( \psi ::= \phi \mid \psi \land \psi \mid \neg \psi \mid X \psi \mid \psi U \psi \)

- **Example:**
  - \( P_{>p}[ GF (\text{send} \rightarrow P_{>0}[F \text{ack}]) ] \)

- **PCTL* model checking algorithm**
  - bottom-up traversal of parse tree for formula (like PCTL)
  - to model check \( P_{\neg p}[\psi] \):
    - replace maximal state subformulae with atomic propositions
    - (state subformulae already model checked recursively)
    - modified formula \( \psi \) is now an LTL formula
    - which can be model checked as for LTL
LTL model checking for MDPs

- Model check LTL specification $P_{\sim p} [ \psi ]$ against MDP $M$

- 1. Convert problem to one needing maximum probabilities
  - e.g. convert $P_{\geq p} [ \psi ]$ to $P_{<1-p} [ \neg \psi ]$
- 2. Generate a DRA for $\psi$ (or $\neg \psi$)
  - build nondeterministic Büchi automaton (NBA) for $\psi$ [VW94]
  - convert the NBA to a DRA [Saf88]
- 3. Construct product MDP $M \otimes A$
- 4. Identify accepting end components (ECs) of $M \otimes A$
- 5. Compute max. probability of reaching accepting ECs
  - from all states of the $D \otimes A$
- 6. Compare probability for $(s, q_s)$ against $p$ for each $s$
Complexity

- Complexity of model checking LTL formula $\psi$ on DTMC $D$
  - is doubly exponential in $|\psi|$ and polynomial in $|D|$
- Converting LTL formula $\psi$ to DRA $A$
  - for some LTL formulae of size $n$, size of smallest DRA is $2^{2^n}$
- In total: $\mathcal{O}(\text{poly}(|D|,|A|))$
- In practice: $|\psi|$ is small and $|D|$ is large
- Can be reduced to single exponential in $|\psi|$
  - see e.g. [CY88,CY95]

- Complexity of model checking LTL formula $\psi$ on MDP $M$
  - is doubly exponential in $|\psi|$ and polynomial in $|M|$
  - unlike DTMCs, this cannot be improved upon
Part 3

Probabilistic programs as MDPs
Probabilistic software

- Consider sequential ANSI C programs
  - support functions, pointers, arrays, but not dynamic memory allocation, unbounded recursion, floating point operations
- Add function `bool coin(double p)` for probabilistic choice
  - for modelling e.g. failures, randomisation
- Add function `int ndet(int n)` for nondeterministic choice
  - for modelling e.g. user input, unspecified function calls
- Aim: verify software with failures, e.g. wireless protocols
  - extract models as Markov decision processes
  - properties: maximum probability of unsuccessful data transmission, minimum expected number of packets sent
- Develop abstraction-refinement framework [VMCAI09]
Example – sample target program

```c
bool fail = false;
int c = 0;
int main ()
{
    // nondeterministic
    c = num_to_send ()
    while (! fail && c > 0)
    {
        // probabilistic
        fail = send_msg ();
        c --;
    }
}
```

Φ: “what is the minimum/maximum probability of the program terminating with fail being true?”
Example – simplified

bool fail = false;
int c = 0;
int main ()
{
    // nondeterministic
    c = ndet (3);
    while (! fail && c > 0)
    {
        // probabilistic
        fail = coin (0.1);
        c --;
    }
}
**Abstraction-refinement loop**

- **Model extraction: extension of goto-cc**
  - function inlining, constant/invariant propagation, side-effect free expressions, points-to analysis, etc.

- **Probabilistic program**
  - probabilistic control flow graph
  - Markov decision process (MDP) semantics
bool fail = false;
int c = 0;
int main ()
{
    // nondeterministic
    c = ndet (3);
    while (! fail && c > 0)
    {
        // probabilistic
        fail = coin (0.1);
        c --;
    }
}
Probabilistic program as MDP

minimum/maximum probability of the program terminating with fail being true is 0 and 0.19, respectively
Experimental results

• Successfully applied to several Linux network utilities:
  − TFTP (file-transfer protocol client)
  − 1 KLOC of non-trivial ANSI-C code
  − Loss of packets modelled by probabilistic choice
  − Linux kernel calls modelled by nondeterministic choice

• Example properties
  − “maximum probability of establishing a write request”
  − “maximum expected amount of data that is sent before timeout”
  − “maximum expected number of echo requests required to establish connectivity”

• Implemented through extension of CProver and PRISM
Part 4

PRISM
Tool support: PRISM

- **PRISM: Probabilistic symbolic model checker [CAV11]**
  - developed at Birmingham/Oxford University, since 1999
  - free, open source software (GPL), runs on all major OSs

- **Support for:**
  - models: DTMCs, CTMCs, MDPs, PTAs, SMGs, ...
  - properties: PCTL, CSL, LTL, PCTL*, costs/rewards, rPATL, ...

- **Features:**
  - simple but flexible high-level modelling language
  - user interface: editors, simulator, experiments, graph plotting
  - multiple efficient model checking engines (e.g. symbolic)
  - **New!** strategy synthesis, stochastic game models (SMGs), multiobjective verification, parametric models

- **See:** [http://www.prismmodelchecker.org/](http://www.prismmodelchecker.org/)
PRISM GUI: Editing a model
PRISM GUI: The Simulator
PRISM GUI: Model checking and graphs
Probabilistic verification in action

- Bluetooth device discovery protocol
  - frequency hopping, randomised delays
  - low-level model in PRISM, based on detailed Bluetooth reference documentation
  - numerical solution of 32 Markov chains, each approximately 3 billion states
  - identified worst-case time to hear one message, 2.5 seconds

- FireWire root contention
  - wired protocol, uses randomisation
  - model checking using PRISM
  - optimum probability of leader election by time T for various coin biases
  - demonstrated that a biased coin can improve performance
Probabilistic verification in action

• DNA transducer gate [Lakin et al, 2012]
  – DNA computing with a restricted class of DNA strand displacement structures
  – transducer design due to Cardelli
  – automatically found and fixed design error, using Microsoft’s DSD and PRISM

• Microgrid demand management protocol [TACAS12,FMSD13]
  – designed for households to actively manage demand while accessing a variety of energy sources
  – found and fixed a flaw in the protocol, due to lack of punishment for selfish behaviour
  – implemented in PRISM-games
Summary

• **Overview of probabilistic model checking**
  – discrete–time Markov chains and Markov decision processes
  – property specifications in temporal logics
  – model checking methods combine graph–theoretic techniques, automata–based methods, numerical equation solving and optimisation

• **Ongoing work (not discussed)**
  – further models (stochastic games, probabilistic timed/hybrid automata)
  – controller/strategy synthesis
  – runtime verification
  – multiobjective verification and synthesis
  – sampling–based exploration

• **Potential for connections to probabilistic programming**
  – integrate with probabilistic inference
Further material

• Reading

• See also
  – 20 lecture course taught at Oxford
  – http://www.prismmodelchecker.org/lectures/pmc/

• PRISM website www.prismmodelchecker.org