Applied Concurrency Theory
Lecture 5 : probabilistic models

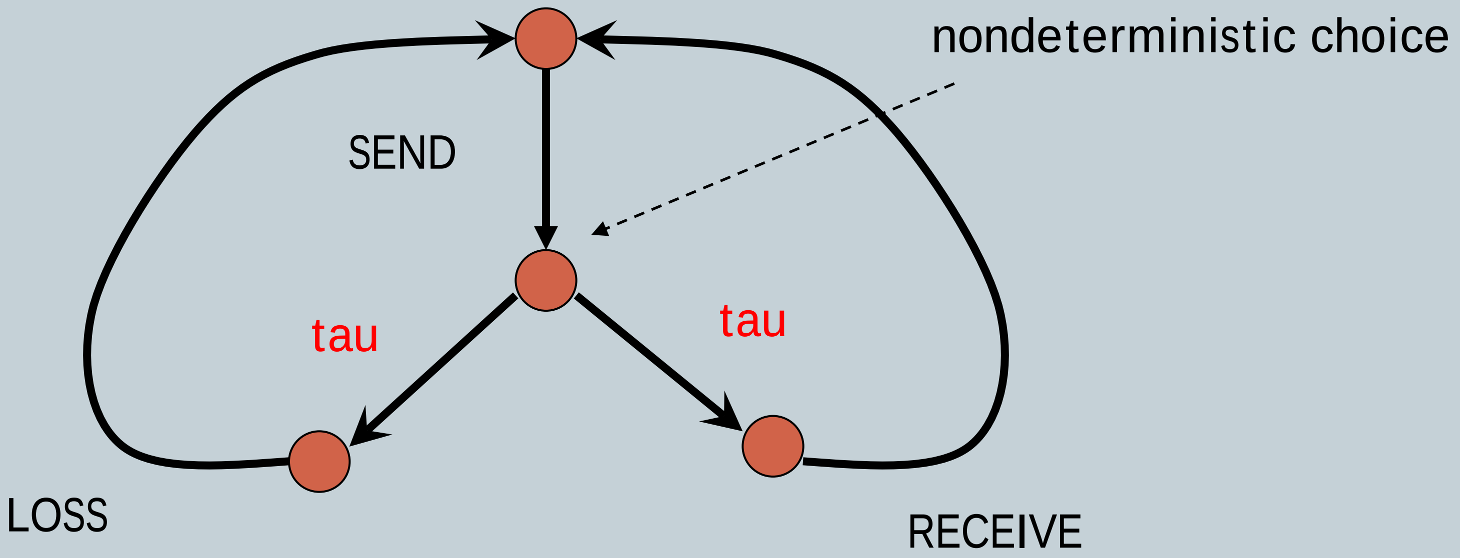
Hubert Garavel

Alexander Graf-Brill

Nondeterministic choice – probabilistic choice

Example 1: a lossy transmission channel

process P = SEND; (tau; RECEIVE; P [] tau; LOSS; P)

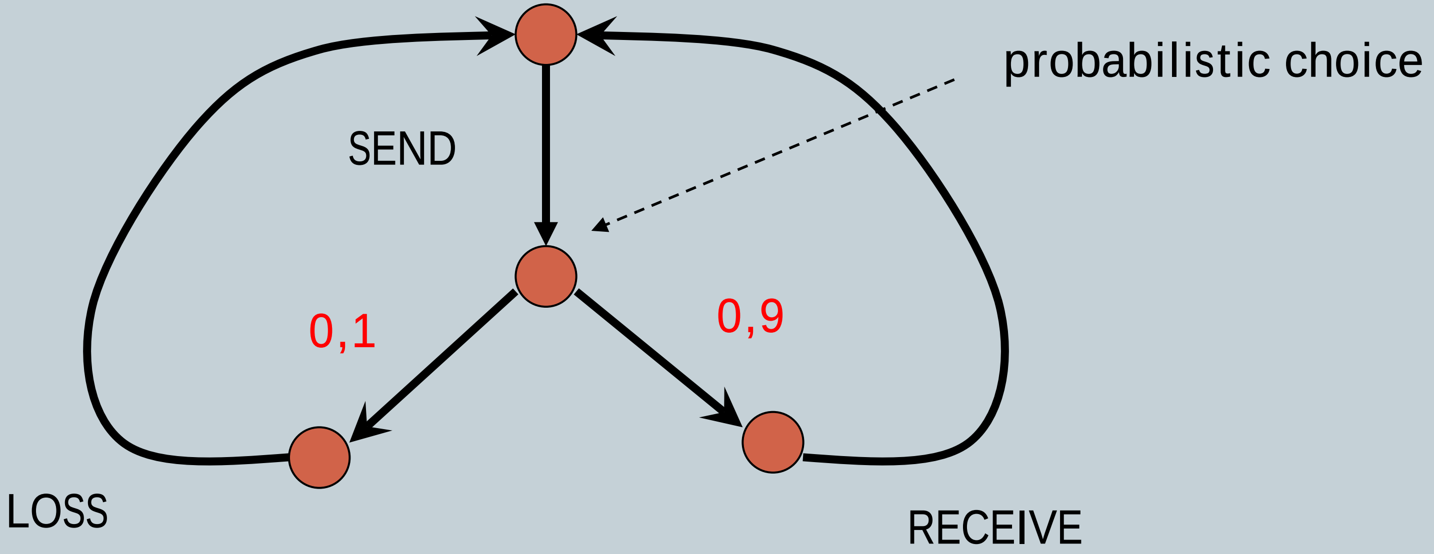


Nondeterminism is not optimal here…

- All branches have the same probability (or, more precisely, have an unspecified probability)
  - yet, in practice, we know that losses are not frequent
- Because this probability is unspecified, no numerical estimation can be done by tools
- Solution: switch to a probabilistic model, with explicitly specified probabilities

Lossy channel (probabilistic version)

process P = SEND; (0.9; RECEIVE; P [] 0.1; LOSS; P)



Randomized algorithms

- In the lossy channel example, probabilities will enable to compute useful data (e.g., the average percentage of messages lost on a long period).

- More generally, there are useful algorithms relying on random behaviour

  See Wikipedia: Randomized algorithm

  Other examples (taken from the PRISM tool library):

  - Randomised consensus
  - Self-stabilising algorithms
  - Bluetooth device discovery
  - Crowds anonymity protocol
  - Contract signing protocols

Discrete-time Markov chains

The simplest model

- DTMC (Discrete-time Markov chains) [Andrei Markov, 1906]
- A finite (or infinite) automaton
  - infinite DTMC are mathematically well-defined
  - but software tools mostly deal with finite-state DTMCS
- Each transition T is labelled with its probability to be fired
  - probability 0: firing T is impossible
  - probability 1: firing T is mandatory
- Constraint:
  - for each state S, the sum of probabilities attached to the transitions leaving S must be equal to 1
  - if sum less than 1, one sometimes assumes that one remains in S for the remaining probability

Example 2: the coin and the dice (1/2)

- Problem raised by D. E. Knuth and A. C. Yao:

  [KY76] The complexity of nonuniform random number generation. In J. F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, Academic Press, New York, 1976

- How to simulate a dice with 6 faces by using only a coin?
  - assuming that all coin tossing experiments are independent
  - and that the coin is fair, i.e., heads and tails have the same probability (50%-50%)

Example 2: the coin and the dice (2/2)

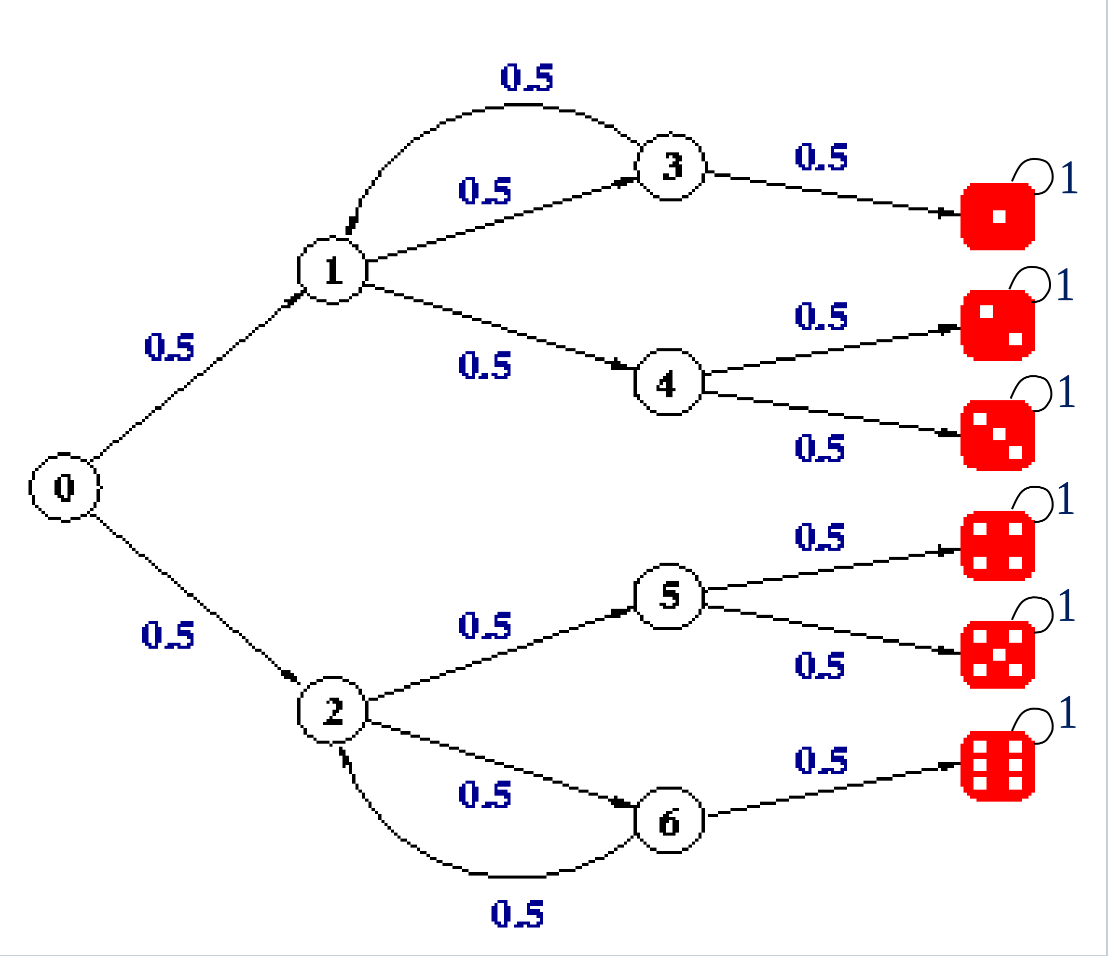
initial state : 0

heads = follow upper arrow

tails = follow lower arrow

one remains forever
in red states

How to prove that each red state is eventually reached with probability 1/6?



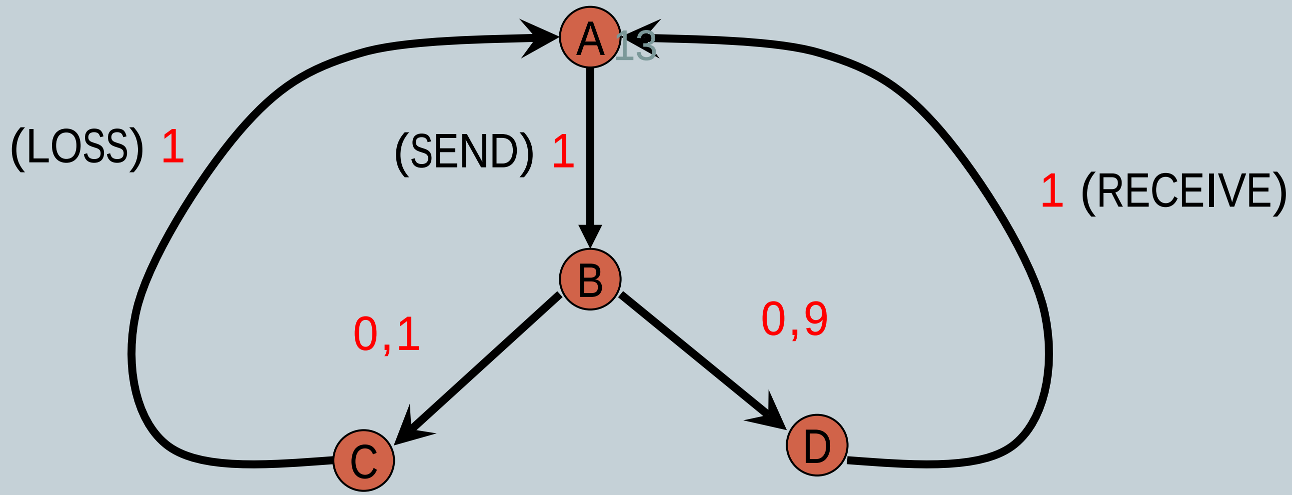
Matrix representation of a DTMC

- If the DTMC is finite with N states, then it can be represented by an N x N **transition matrix** (or **one-step matrix**, or **Markov matrix**)
- Element (*i*, *j*) of the matrix is the **probability** attached to the transition from state i to state j *(i : raw, j : column)*
- The sum of the elements on each line of the matrix must always be egal to 1
- If it is not the case, one might have forgotten the 'looping' transition that permits to remain in the same state (e.g., as with the red states of Example 2)

How does a DTMC work?

- Standard automaton:
  - An automaton evolves (its state changes) at discrete instants
  - At each instant, the automaton is in one and only one state
- DTMC:
  - A DTMC evolves (its state changes) at discrete instants
  - At each instant, the DTMC can be in one or several states, but with smaller probabilities than 1
- Physical metaphor:
  - automaton: the current state is a particle that cannot be divided
  - DTMC: the current state is a wave that splits and flows into several states

Example



Instant 1 : DTMC is in state A at 100%

Instant 2 : DTMC is in state B at 100%

Instant 3 : DTMC is in state C at 10% and/or D at 90%

Instant 4 : DTMC is in state A at 100%, etc.

Probability vectors

If the DTMC has N states, a probability vector at a given time instant is a vector V with N elements:

$$V = \begin{pmatrix} p_1 \\ p_2 \\ \dots \\ p_N \end{pmatrix} \quad \text{where} \quad p_1 + p_2 + \dots + p_n = 1$$

where p_i is the probability to be in the i-th state at this time instant.

A probability vector generalizes the notion of current state; for an ordinary automaton, one p_i would be 1 and all others would be 0.

Evolution of probabilities as time passes

If V is the probability vector describing the DTMC at a given time instant, the probability vector V' at the next time instant after a transition is given by the following equation

$V' = {}^{t}V \, . \, M$ (and not $V' = M.V$!)

where M is the transition matrix of the DTMC

${}^{t}V = (p_1 \; p_2 \; ... \; p_N)$ ${}^{t}V$: *transposed vector*

$$(p_1 \; p_2 \; p_3) \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = \begin{pmatrix} a.p_1 + d.p_2 + g.\,p_3 \\ b.p_1 + e.p_2 + h.\,p_3 \\ c.p_1 + f.p_2 + i.\,p_3 \end{pmatrix}$$

Steady-state probabilities (1/3)

- As time passes, the probability vector V evolves (‘transient probabilities’)

- Can one predict what will happen on the long run? (i.e., the limit of V when times tends to infinity)

- Stationary (or steady-state) behaviour:
  - there is an initial transient phase,
  - on the long run, an equilibrium is reached
  - probabilities are distributed among states and do not change (or converge to a limit) as time is passing

Steady-state probabilities (2/3)

- If such an equilibrium exists, the steady-state probability vector V should satisfy the following equation:

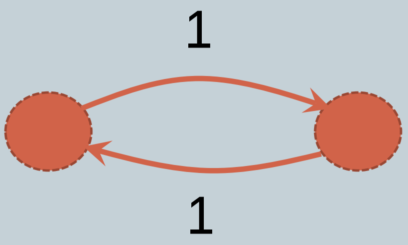
$$^{t}V \,.\, M = V$$ *(V is a left eigenvector of M)*

- Remarks:
  - M is not 'free' because the sum of each of its lines must be 1 (the last column is 1 minus the sum of other columns)
    => this gives one less equation
  - but the sum of all elements of V must be 1 too
    => this gives one more equation
  - So N variables and N equations

Steady-state probabilities (3/3)

Equilibrium equation ${}^{t}V \, . \, M = V$

- Does a solution always exist? No
- If it exists, is it unique? No



- Sufficient conditions exist for a unique solution
  - e.g., when matrix M is aperiodic and irreducible
  - the coin/dice DTMC does not meet these conditions, but admits a unique solution

- In certain cases, the solution does not depend on the initial probability vector ('self-stabilizing')
  - e.g., when matrix M is aperiodic and irreducible
  - the coin/dice DTMC solution depends on the initial state!

Mathematical definition of DTMCs

- Mathematical DTMCs vs Computer-Science DTMCs;
  - mathematical definition of DTMCs allows infinite state spaces
  - mathematical studies ignore parallel composition of DTMCs

- Basis: a sequence of random variables X_0, X_1, X_2, ... X_n … that give the current state of the DTMC at instant n
- Notations:
  - prob (X_n = s) : probability that the DTMC is in state s at instant n (i.e., an element of a probability vector)
  - prob (X_n = s | X_i) i<n : conditional probability knowing X_i that the DTMC is in state s at instant n
  - prob (X_n = s | X_i , X_j) i<n and j<n : conditional probability knowing X_i and X_j etc.

Markov property

- A DTMC satisfies the 'Markov property'

  $\text{prob}(X_{n+1} = s \mid X_0, X_1, X_2, \ldots X_n) = \text{prob}(X_{n+1} = s \mid X_n)$

- This property expresses that the future (i.e., the next state at instant n+1) only depends on the present (i.e., the current state at instant n) and not on the past (i.e., between instants 0 and n-1)
- Said differently, the present contains all the information needed to predict the future and one does not need to record the entire history from instant 0 to continue evolving
- Automata also have this property: their current state encodes all the history needed to take future decisions

Markov decision processes

Limitations of DTMCs

- A state-based model
  - All the ‘useful’ information is in the states
  - No visible information on the transitions (only probabilities)
  - This does not fit with the usual models of concurrency
- How to compose DTMCs in parallel?
  - This is mandatory to model concurrent components
  - Parallel composition of DTMCs is severely restricted: no message-passing communication, only shared variables
- How to model ‘true’ nondeterminism?
  - ‘True’ nondeterminism cannot be modelled using DTMCs
  - Concurrency introduces nondeterminism (due to interleaving)
  - => parallel composition of DTMCs is not a DTMC

Beyond DTMCs

- **Main goal**
  - Introduce transitions labelled with action names as in the LTS (Labelled Transition Systems) model used for CCS, CSP, LOTOS, pi-calculus, etc.
  - Keep the possibility of having probabilities on transitions
  - Have a meaningful definition of parallel composition
- **Different solutions:**
  - IPC (Interactive Probabilistic Chains) = LTS with normal transitions and probabilistic transitions
  - MDP (Markov Decision Processes) = IPC + alternation of normal and probabilistic transitions

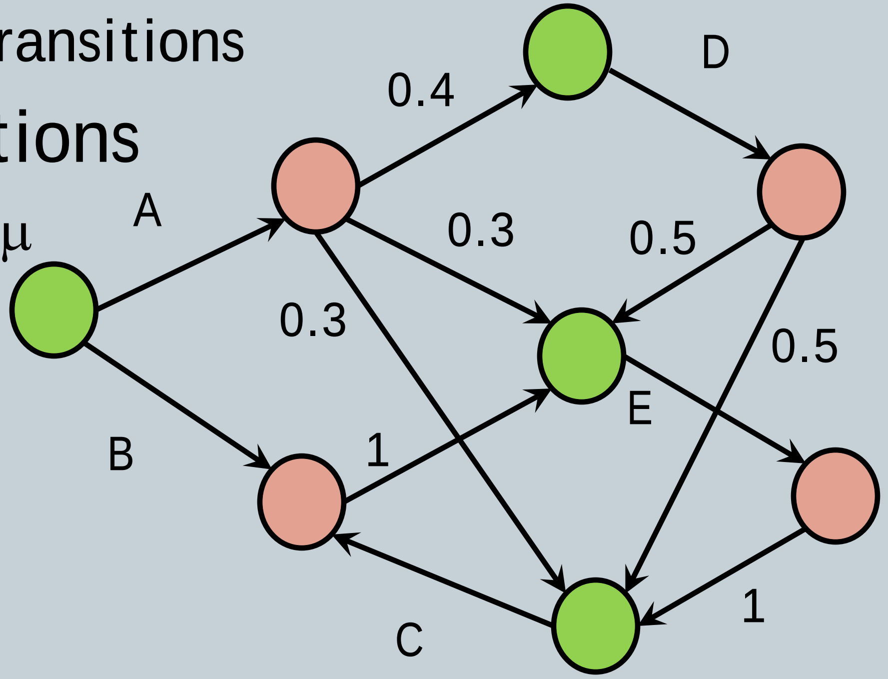
Markov Decision Processes (1/2)

- As with IPCs, MDP have 2 kinds of transitions:
  - normal transitions: ‘A’, ‘GET !2 !false’, τ, etc.
  - probabilistic transitions: 0.001, 0.25
- Additional constraints:
  - the sum of probabilistic transitions leaving a state must be 1 (aleady exists in DTMCs and IPCs)
  - no choice between a normal and a probabilistic transition
  - alternation (stronger constraint):
    on every execution path, normal and probabilistic transitions strictly alternate

Markov Decision Processes (2/2)

Consequences of alternation:

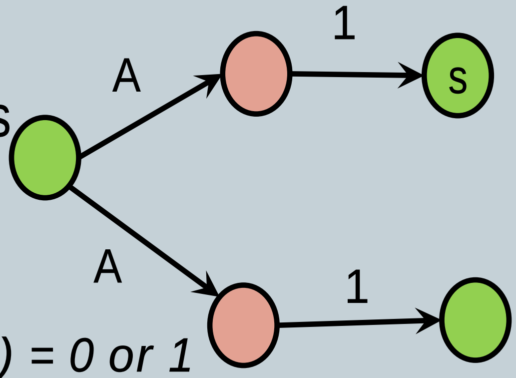
- Graphically: 2 kinds of vertices
  - ‘states’: before normal transitions
  - ‘nails’: before probabilistic transitions
- Mathematically: 2 definitions
  - transitions = state —(label)→μ
  - μ = probability distribution over states



Nondeterminism in MDP

- Nondeterminism is allowed in MDP
- Two causes:
  - local nondeterminism: choice between two identical transitions leading to different nails
  - global nondeterminism: coming from parallel composition and interleaving semantics

- Main consequence:
  - no unique probability vector as with DTMCs
  - one may only compute a [min, max] interval of probabilities



after an A-transition, prob (X=s) = 0 or 1

The PRISM tool

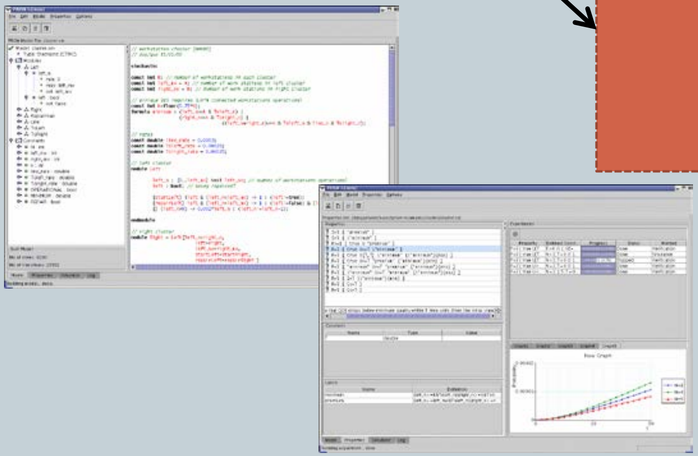
The PRISM tool

- Developed in Oxford (formerly: Birmingham)
- Web site: http://www.prismmodelchecker.org

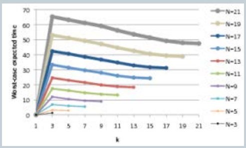
model

properties

PRISM



results of the property
+ probabilities
+ matrices
+ statistics



The PRISM modelling language

Motivation

PRISM offers a modelling language to describe:

- Sequential modules (~ processes):
  - DTMC (Discrete-Time Markov Chains)
  - MDP (Markov Decision Processes)
  - and also CTMC and PTA (see Lecture 6)
- Parallel composition of modules

Sequential modules from the outside (1/2)

- Mixed interfaces, which combine:
  - action labels (as in process calculi)
  - shared variables (as in thread-based programs)
- Action labels
  - permit synchronization between concurrent modules
  - no exchange of values (as ! and ? in CSP and LOTOS)
- State variables
  - local: writable by one module, readable by other modules
  - global: readable and writable by all modules
  - no notion of 'purely local' variable (≠ process calculi)

Sequential modules from the outside (2/2)

- Drawback: no syntactic way of declaring interfaces
  - no lists of gate and variable parameters as in LOTOS
  - one must read and analyze the body of each module!

- Exemple of PRISM module specification:

```
const int N = 10; // constant
global X:bool;    // global variable
module M
    Y:[0..N];       // local variable of module M
    ...
endmodule
```

Sequential modules from the inside (1/5)

- In most languages (e.g., LOTOS and LOTOS NT), the current state consists of two components:
  - a control part: the current program location (i.e., program counter in an assembly language)
  - a data part: the current values of variables
- In PRISM there is no control part: the current state of a module is entirely encoded in its variables
  - PRISM follows the idea of ‘guarded commands’ language
  - there is one single program location (= single state machine)
  - to encode an automaton with N states, one must declare a local variable of type [1..N] or [0..N-1]

Sequential modules from the inside (2/5)

- The body of a PRISM module combines 2 operators:
  - nondeterministic choice
  - probabilistic choice

- It is not a process calculus in the sense that these two operators must appear in a precise order and cannot be freely combined
  - first level, nondeterministic choice
  - second level, probabilistic choice

Sequential modules from the inside (3/5)

- Nondeterministic choice:

  [action_label$_1$] boolean_guard$_1$ -> branch$_1$;

  [action_label$_2$] boolean_guard$_2$ -> branch$_2$;

  ...

  [action_label$_n$] boolean_guard$_n$ -> branch$_n$;

  - (branches are defined below)
  - action_labels can be ommitted (e.g., in a DTMC) – taus ?
  - guards contain local (and from other modules) and global variables
  - as in LOTOS, boolean_guard may overlap (=> nondeterminism)
  - Caution! in a DTMC, Prism remplaces nondeterminism with an equiprobable probabilistic choice (with a warning?)

Sequential modules from the inside (4/5)

- Probabilistic choice (i.e. branches)

  branch ::= $prob_1$: $update_1$
  + $prob_2$: $update_2$
  + ...
  + $prob_n$: $update_n$

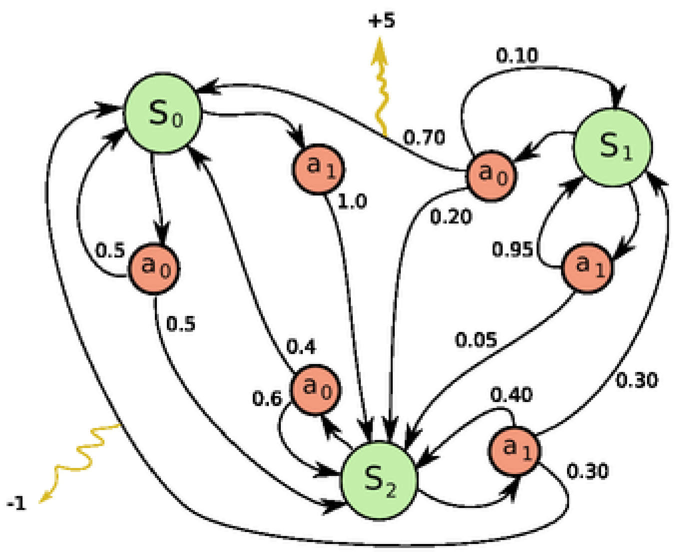
  - the $prob_i$ may use numbers or constants (defined by **const**)
  - their sum must be 1
  - the $update_i$ are assignments to variables, written using a strange syntax:

    (x'= 0) // parentheses and quote are mandatory

    (x'=1) & (y'=y+1) // & rather than ;

Sequential modules from the inside (5/5)

- PRISM syntax corresponds exactly to MDPs
  - in green: states (origin of nondeterministic choices)
  - in red: nails (origin of probabilistic choices)



Source: Wikipedia

```
module M
s : [0..2];
[] s=0->(s'=2);
[] s=0->0.5:(s'=0)+0.5:(s'=2);
[] s=1->0.7:(s'=0)+0.1:(s'=1)+0.2:(s'=2);
[] s=1->0.95:(s'=1)+0.05:(s'=2);
[] s=2->0.4:(s'=0)+0.6:(s'=2);
[] s=2->0.3:(s'=0)+0.3:(s'=1)+0.4:(s'=2);
endmodule
```

Parallel composition of modules

- Explicit parallel composition
  - using the three LOTOS parallel composition operators
  - || only synchronizes on common gates:
    in LOTOS, P || Q synchronizes on gates (P) ∪ gates (Q)
    in PRISM, P || Q synchronizes on gates (P) ∩ gates (Q)
  - another difference with LOTOS : shared variables!
  - global state = local states of each module + global variables
- Implicit parallel composition
  - just declaring modules together composes them with ||
- Hiding and renaming
  - M / {a,b,…} similar to (**hide** a, b... **in** M) in LOTOS
  - M {a<-b,c<-d,…} similar to process calls in LOTOS

The PRISM property specification language

Motivation

- The property language is used to ask questions about the state space
- In ‘traditional’ model checkers, these questions have a Boolean result:
  - can message M (X, Y) be received with X > Y?
  - is each SEND (X) message eventually followed by a RECV(X)?
- In probabilistic model checkers (such as PRISM), the questions may have a Boolean or numerical result
  - often questions about probabilities
  - (but also costs, rewards, elapsed time)

Properties in PRISM

- The property language of PRISM is rich (= complex)
- It merges several temporal logics:
  - standard temporal logics: LTL
  - probabilistic temporal logics: CSL, PCTL, PCTL*
- Depending on the form of the formulas to evaluate, different algorithms (‘engines’) are used by PRISM (e.g., ‘hybrid’, ‘MTBDD’, ‘sparse’)
  - Various restrictions regarding the type of PRISM models, the nature of formulas, and the search engine used.

Examples of Boolean properties

- P>=1 [F X=0]
  - With probability 1, eventually variable X becomes null
- P<0.1 [F<=1000 X=0]
  - With probability less than 0.1, variable X becomes null during the first 1000 time units
- S>=0.75 [X=0]
  - With (steady-state) probability greater than 75%, variable X is null on the long-run

Example of numerical properties

- P=? [F X=0]
  - Give the probability that variable X becomes null eventually
- P=? [F<=1000 X=0]
  - Give the probability that variable X becomes null during the first 1000 time units
- S=? [X=0]
  - Give (steady-state) probability that variable X is null on the long-run

(see the PRISM manual for many more examples)

Today's challenge

Getting started with PRISM

- Type in a file ‘dice.pm’ the PRISM specification of the coin/dice example (Example 2 above)
  - do not forget the loops on the red states
  - pre-check its correctness by launching the command

    $ **prism dice.pm**

- Write a file ‘dice.pctl’ containing PCTL formulas to check that the steady-state probability of each ‘red’ state is 1/6. Check it using PRISM.

- Generate the transition matrix in Matlab format and send it with ‘dice.pm’ and ‘dice.pctl’ to Alexander

References

PRISM language and tool

- PRISM Web site
  - http://www.prismmodelchecker.org/

- PRISM user manual (models and properties) (skip directly to Section "The PRISM Language“)
  - PDF: http://www.prismmodelchecker.org/doc/manual.pdf
  - HTML: http://www.prismmodelchecker.org/manual/Main/AllOnOnePage

- Brief semantics of the basic PRISM constructs
  - http://www.prismmodelchecker.org/doc/semantics.pdf

Markov chains

- Wikipedia: Markov chain
- Wikipedia: Markov decision process
- Real applications of Markov decision processes
  - http://www.it.uu.se/edu/course/homepage/aism/st11/MDPApplications1.pdf
  - http://www.it.uu.se/edu/course/homepage/aism/st11/MDPApplications2.pdf
  - http://www.it.uu.se/edu/course/homepage/aism/st11/MDPApplications3.pdf