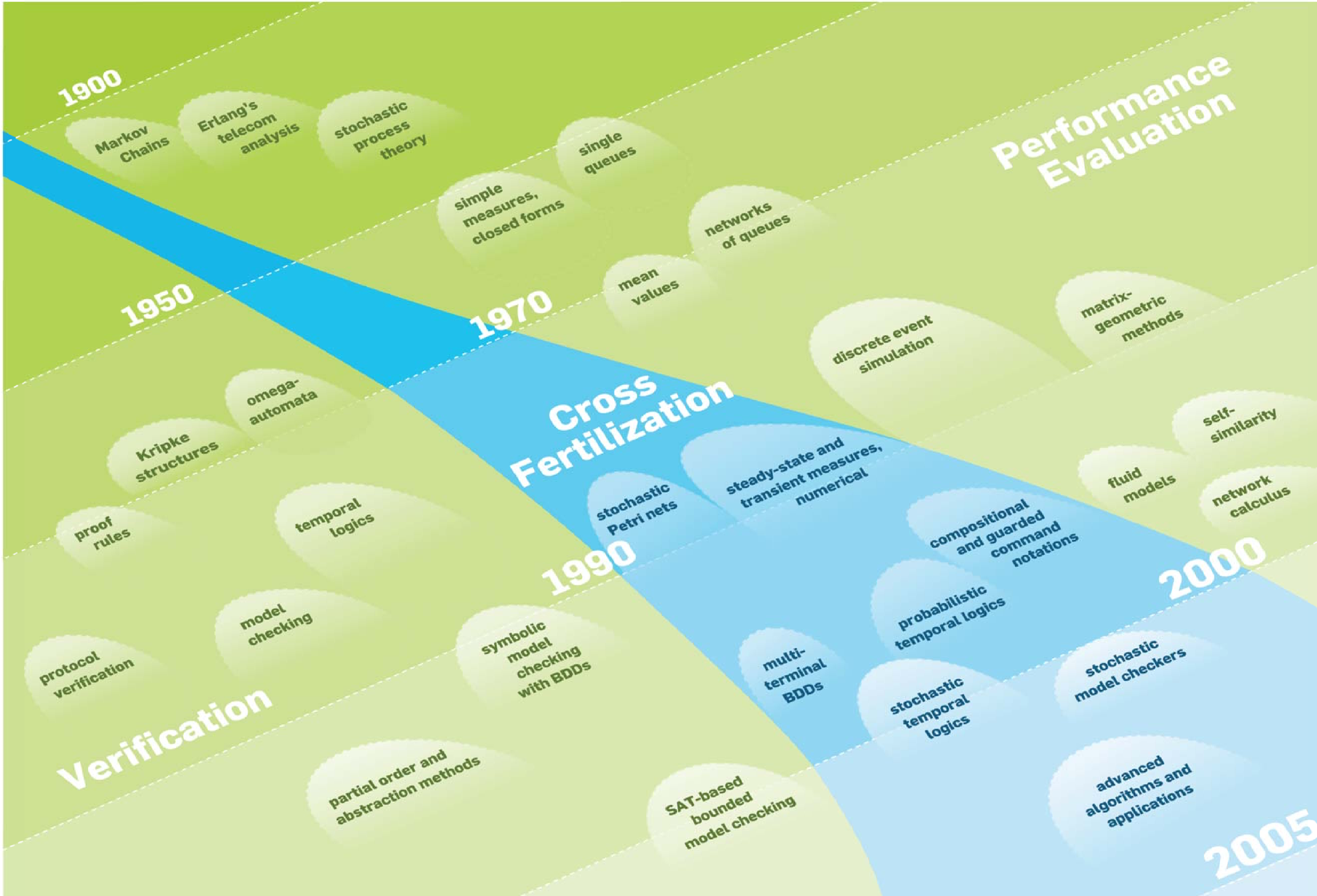

1900
Markov Chains
Erlang's telecom analysis
stochastic process theory
single queues
simple measures, closed forms
Performance Evaluation
networks of queues
1950
mean values
1970
matrix-geometric methods
discrete event simulation
omega-automata
Cross Fertilization
Kripke structures
steady-state and transient measures, numerical
self-similarity
fluid models
stochastic Petri nets
network calculus
proof rules
temporal logics
compositional and guarded command notations
1990
2000
probabilistic temporal logics
model checking
symbolic model checking with BDDs
protocol verification
multi-terminal BDDs
stochastic model checkers
stochastic temporal logics
Verification
partial order and abstraction methods
advanced algorithms and applications
SAT-based bounded model checking
2005

From Concurrency Models to Numbers: Performance, Dependability, Energy

Holger Hermanns

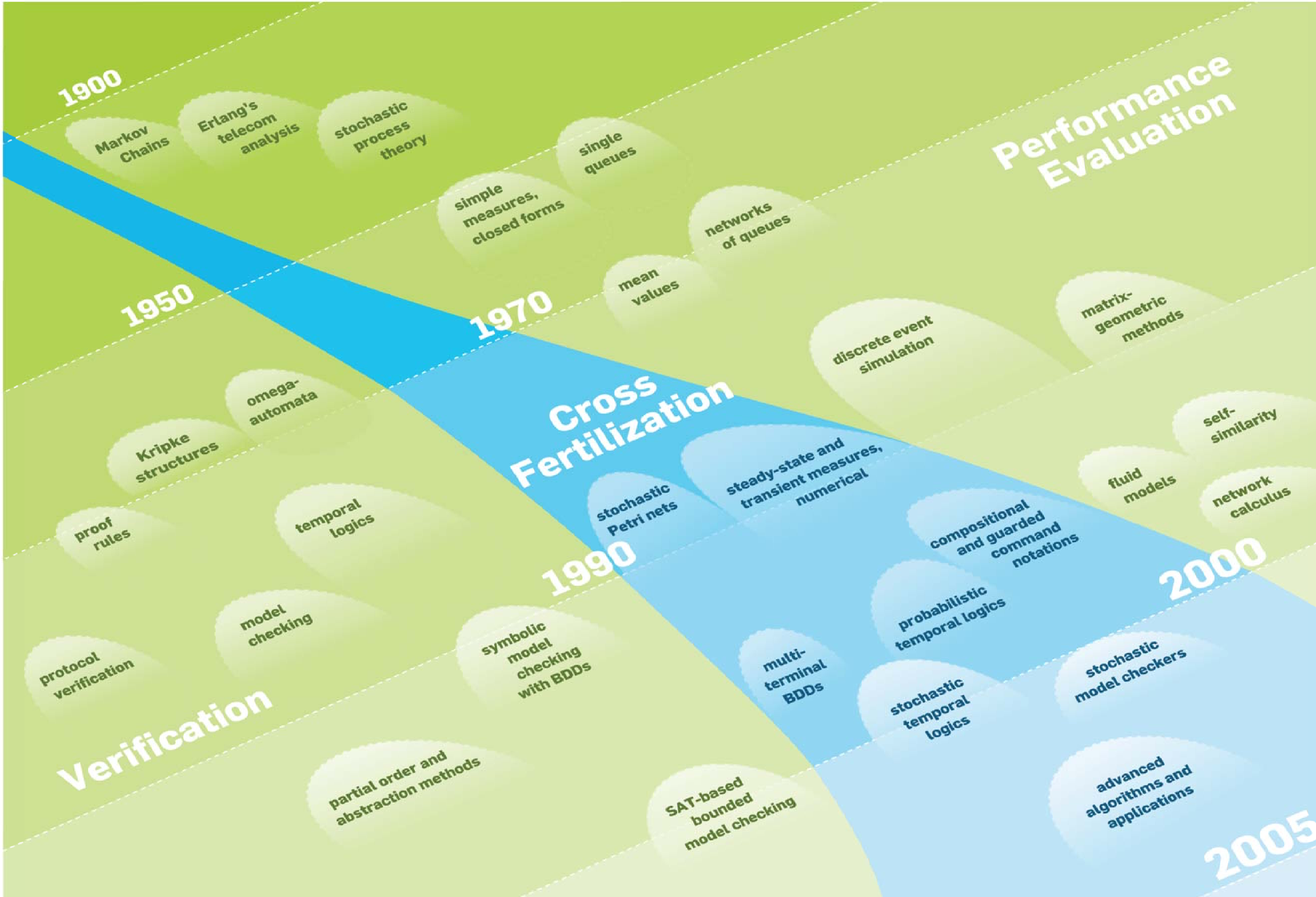
Saarland University – Computer Science, Germany

INRIA Grenoble – Rhône-Alpes, France

International Summer School Marktoberdorf
August, 2010

First Remarks

Probability?	Yes.
Continuous Time?	Yes.
Performance?	Yes.
Reliability?	Yes.
Security?	No.
Concurrency?	Yes.
Compositionality?	Yes.
Computability?	Yes.
Tools?	Several.
Applications?	Plenty.
Numerical Stability?	Huh?


1900
Markov Chains
Erlang's telecom analysis
stochastic process theory
single queues
simple measures, closed forms
networks of queues
mean values
Performance Evaluation
1950
1970
discrete event simulation
matrix-geometric methods
omega-automata
Kripke structures
Cross Fertilization
steady-state and transient measures, numerical
self-similarity
stochastic Petri nets
fluid models
network calculus
proof rules
temporal logics
compositional and guarded command notations
1990
2000
model checking
probabilistic temporal logics
symbolic model checking with BDDs
protocol verification
multi-terminal BDDs
stochastic model checkers
stochastic temporal logics
Verification
partial order and abstraction methods
advanced algorithms and applications
SAT-based bounded model checking
2005

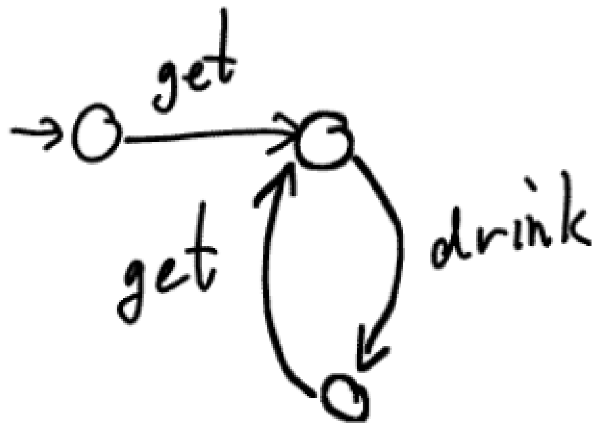
Setting the stage

Transition system

A transition system is a tuple

$$\mathcal{T} = (S, \text{Act}, \longrightarrow, s_0)$$

- S is the state space, i.e., set of states,
- Act is a set of actions,
- $\longrightarrow \subseteq S \times \text{Act} \times S$ is the transition relation,
  transitions are of the form $s \xrightarrow{\alpha} s'$
- $s_0 \in S$ the initial state.

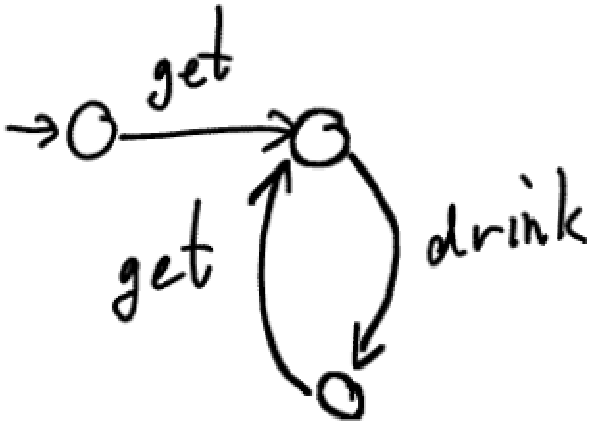


Transition system

A transition system is a tuple

$$\mathcal{T} = (S, \mathrm{Act}, \longrightarrow, s_0, AP, L)$$

- S is the state space, i.e., set of states,
- Act is a set of actions,
- $\longrightarrow \subseteq S \times \mathrm{Act} \times S$ is the transition relation, transitions are of the form $s \xrightarrow{\alpha} s'$
- $s_0 \in S$ the initial state,
- AP a set of atomic propositions,
- $L : S \to 2^{AP}$ the labeling function.



Concurrency and communication

"real" concurrent system

$$P = P_1 \parallel \ldots \parallel P_n$$

$$\updownarrow$$

transition system

$$\mathcal{T} = \mathcal{T}_1 \parallel \cdots \parallel \mathcal{T}_n$$

Holy Grail: define **semantic operators** on transition systems that model "real" concurrent behaviour

Operators for parallelism and communication

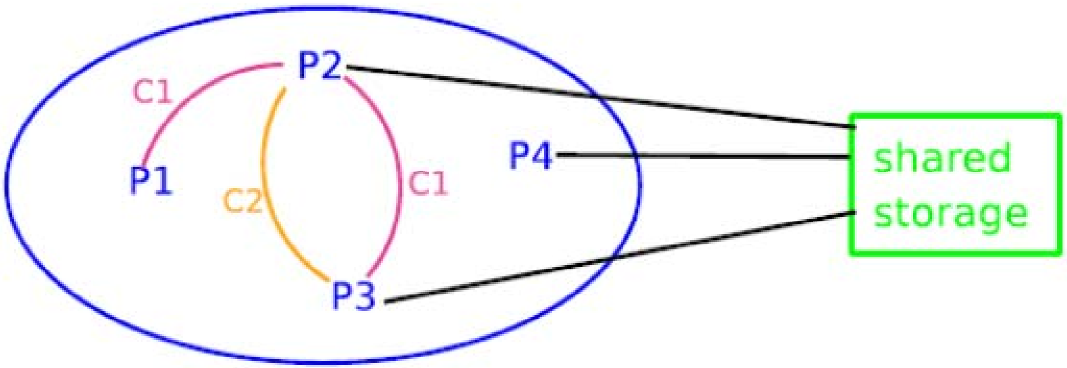
- pure concurrency

interleaving
for entirely independent systems
no communication, no dependencies

- synchronous message passing
- synchronous product

for parallel systems with fully synchronous execution
e.g. clocked hardware

. . . and the full monty . . .



Interleaving operator for transition systems

$\mathcal{T}_1 = (S_1, \mathrm{Act}_1, \longrightarrow_1, s_{01}, AP_1, L_1)$ $\mathcal{T}_2 = (S_2, \mathrm{Act}_2, \longrightarrow_2, s_{02}, AP_2, L_2)$

The composite transition system $\mathcal{T}_1 \,|||\, \mathcal{T}_2$ is:

$$\mathcal{T}_1 \,|||\, \mathcal{T}_2 \;=\; (S_1 \times S_2,\ \mathrm{Act}_1 \cup \mathrm{Act}_2,\ \longrightarrow,\ \langle s_{01}, s_{02}\rangle, AP, L)$$

where the transition relation $\longrightarrow$ is given by:

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1'}{\langle s_1, s_2\rangle \xrightarrow{\alpha} \langle s_1', s_2\rangle} \qquad \frac{s_2 \xrightarrow{\alpha}_2 s_2'}{\langle s_1, s_2\rangle \xrightarrow{\alpha} \langle s_1, s_2'\rangle}$$

atomic propositions: $AP = AP_1 \uplus AP_2$

labeling function: $L(\langle s_1, s_2\rangle) = L_1(s_1) \cup L_2(s_2)$

Synchronous product for transition systems

$\mathcal{T}_1 = (S_1, \mathsf{Act}_1, \longrightarrow_1, \ldots)$ $\qquad$ $\mathcal{T}_2 = (S_2, \mathsf{Act}_2, \longrightarrow_2, \ldots)$

The synchronous product $\mathcal{T}_1 \otimes \mathcal{T}_2$ is:

$$\mathcal{T}_1 \otimes \mathcal{T}_2 \;=\; (S_1 \times S_2, \mathsf{Act}, \longrightarrow, \ldots)$$

where the transition relation $\longrightarrow$ is given by:

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1' \;\wedge\; s_2 \xrightarrow{\beta}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha * \beta} \langle s_1', s_2' \rangle}$$

action set Act is given by a function

$$* : \mathsf{Act}_1 \times \mathsf{Act}_2 \longrightarrow \mathsf{Act}, \qquad (\alpha, \beta) \mapsto \alpha * \beta$$

for parallel systems with fully synchronous execution

Synchronous message passing for transition systems

$\mathcal{T}_1 = (S_1, \mathsf{Act}_1, \longrightarrow_1, \ldots)$ $\qquad$ $\mathcal{T}_2 = (S_2, \mathsf{Act}_2, \longrightarrow_2, \ldots)$

The concurrent execution with **synchronization** over all actions in Syn is:

$$\mathcal{T}_1 \parallel_{\mathsf{Syn}} \mathcal{T}_2 \;=\; (S_1 \times S_2, \mathsf{Act}_1 \cup \mathsf{Act}_2, \rightarrow, \ldots)$$

where $\mathsf{Syn} \subseteq \mathsf{Act}_1 \cap \mathsf{Act}_2$ set of synchronization actions

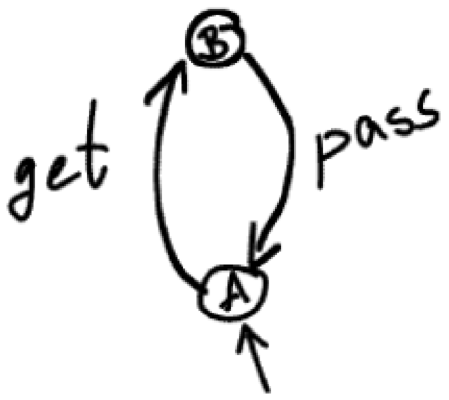
interleaving for $\alpha \in \mathsf{Act}_i \setminus \mathsf{Syn}$:

$$\frac{s_1 \overset{\alpha}{\longrightarrow}_1 s_1'}{\langle s_1, s_2 \rangle \overset{\alpha}{\longrightarrow} \langle s_1', s_2 \rangle} \qquad \frac{s_2 \overset{\alpha}{\longrightarrow}_2 s_2'}{\langle s_1, s_2 \rangle \overset{\alpha}{\longrightarrow} \langle s_1, s_2' \rangle}$$

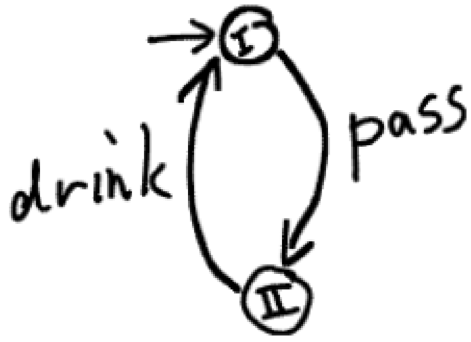
handshaking (rendezvous) for $\alpha \in \mathsf{Syn}$:

$$\frac{s_1 \overset{\alpha}{\longrightarrow}_1 s_1' \quad \wedge \quad s_2 \overset{\alpha}{\longrightarrow}_2 s_2'}{\langle s_1, s_2 \rangle \overset{\alpha}{\longrightarrow} \langle s_1', s_2' \rangle}$$

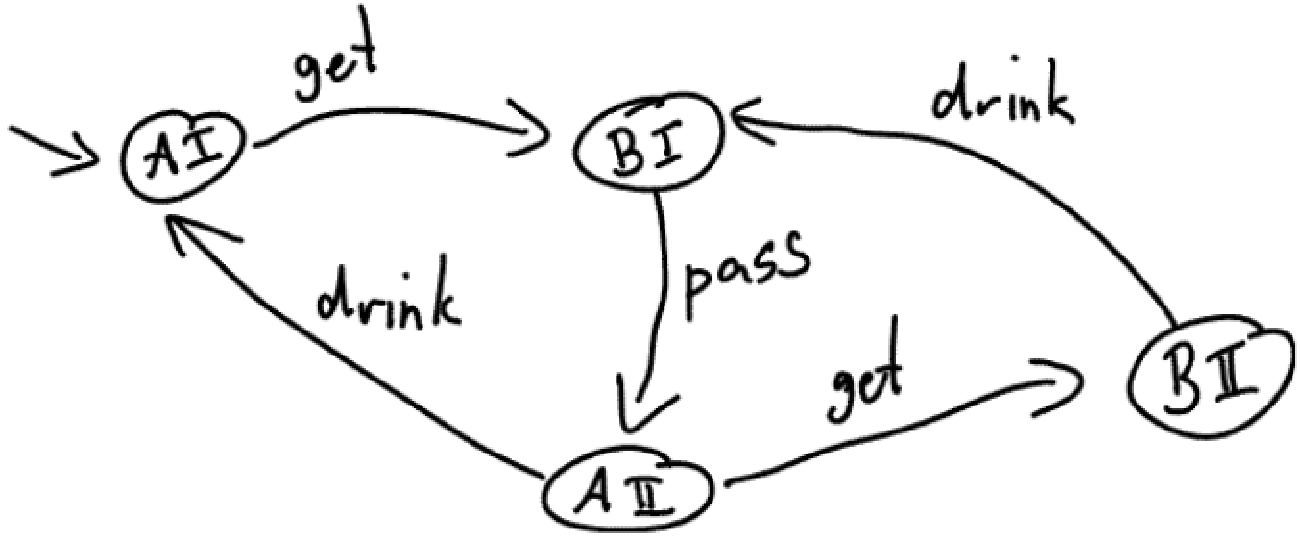
Lecturer



Student



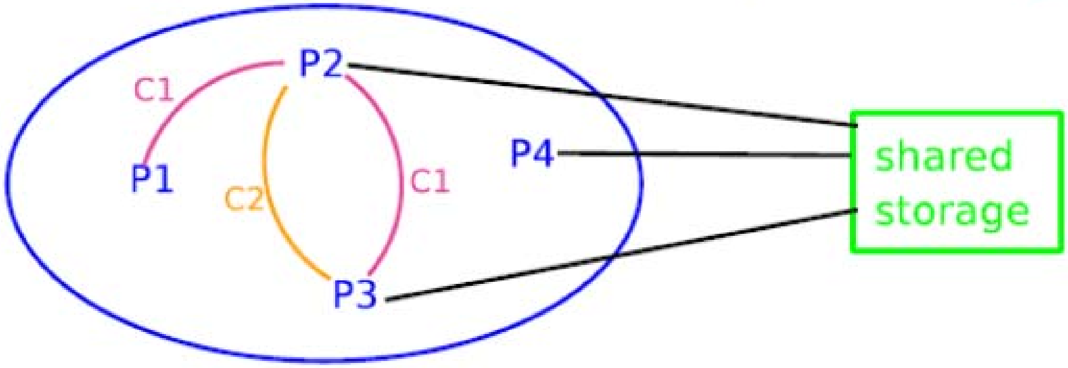
$\text{Lecturer} \parallel_{\{pass\}} \text{Student}$



Channel systems and shared variable systems

We want to represent data-dependent concurrent systems with

- communication over **shared variables**
- **synchronous message passing** (channels of **capacity 0**)
- **asynchronous message passing** (**capacity** $\geq$ **1**)



This can all be encoded into

- transition systems

  and synchronous message passing

Bisimulation, a natural equivalence

$\mathcal{T}_1 = (S_1, \mathsf{Act}_1, \longrightarrow_1, \ldots)$ $\qquad\qquad$ $\mathcal{T}_2 = (S_2, \mathsf{Act}_2, \longrightarrow_2, \ldots)$

A relation $\mathbf{R} \subseteq S_1 \times S_2$ is a bisimulation, if
for all $(s_1, s_2) \in \mathbf{R}$ and for all $\alpha \in \mathsf{Act}$:

(1) $s_1 \xrightarrow{\alpha}_1 s_1'$ implies $\exists\ s_2 \xrightarrow{\alpha}_2 s_2'$ such that $(s_1', s_2') \in \mathbf{R}$
(2) $s_2 \xrightarrow{\alpha}_2 s_2'$ implies $\exists\ s_1 \xrightarrow{\alpha}_1 s_1'$ such that $(s_1', s_2') \in \mathbf{R}$

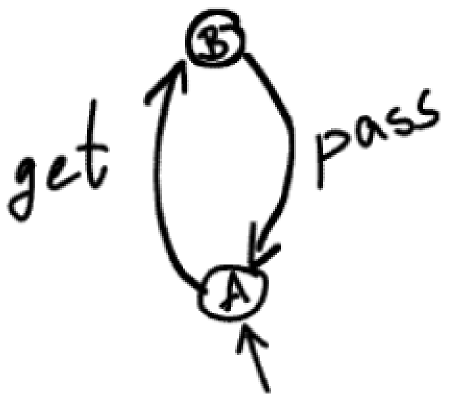
Bisimulation equivalence of $\mathcal{T}_1$ and $\mathcal{T}_2$ requires that
$\mathcal{T}_1$ and $\mathcal{T}_2$ can *simulate each other* in a stepwise manner

$\mathcal{T}_1 \sim \mathcal{T}_2$ iff there is a bisimulation $\mathbf{R}$ for $(\mathcal{T}_1, \mathcal{T}_2)$ relating the initial states.

Bisimulation equivalence is a congruence for $\|_{\mathsf{Syn}}$ (and $\otimes$, and ...)

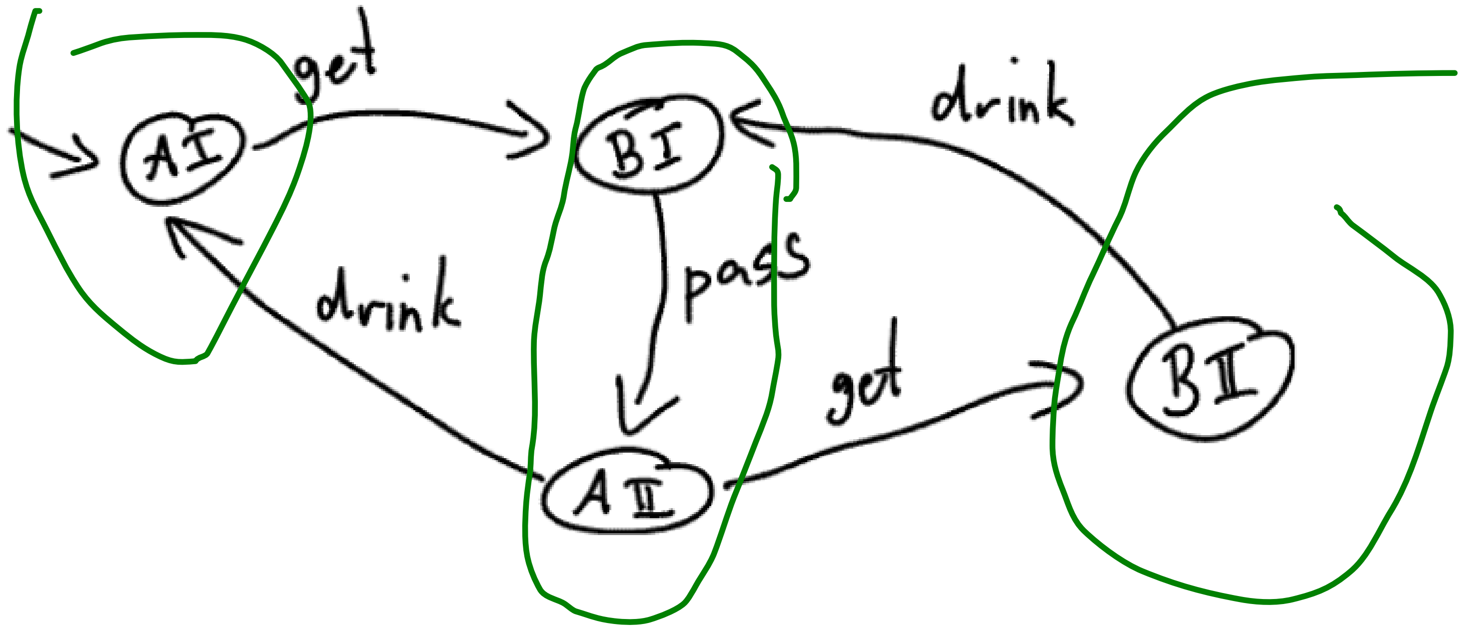
Can be weakened to ignore 'internal' moves, same principal properties.

Lecturer



Student

Lecturer $\|_{\{pass\}}$ Student



What we have

Transition systems.

A (set of) natural and expressive composition operator(s).

A natural congruence notion, bisimulation.

What does this buy us?

Principal understanding.

What we also have:

An abstraction operator (hiding).

Efficient minimisation algorithms for bisimulation.

Matching logics (CTL, sugared a-f mu-calculus).

What does this buy us?

Compositional minimisation. Practical verification.

Who sells that?

What is CADP?

- Home Page
- Tools Overview
- Current Status

Installation

- How to obtain CADP?
- Usage Statistics
- Issues & Patches

Documentation

- Tutorials
- Publications
- Manual Pages
- Demo Examples
- FAQ

CADP Newsletters

- Nr. 1 - Dec. 1996
- Nr. 2 - Jun. 1997
- Nr. 3 - Sep. 1997
- Nr. 4 - Jan. 1999
- Nr. 5 - Jul. 2001
- Nr. 6 - Apr. 2007

CADP Community

- Forum
- Education & Training
- Case Studies
- Research Tools

- demo_31: UPDATED SCSI-2 bus arbitration protocol
  Hubert Garavel, Holger Hermanns, Radu Mateescu, Christophe Joubert, and David Champelovier
  Tools used: CAESAR, CAESAR.ADT, BCG_MIN, BCG_STEADY, DETERMINATOR, EVALUATOR, SVL

- demo_32: Sequentially consistent, distributed cache memory
  Susanne Graf and Wendelin Serwe
  Tools used: CAESAR, CAESAR.ADT, BISIMULATOR, BCG_MIN, SVL

- demo_33: Randomized binary distributed consensus protocol
  Frédéric Tronel and Frédéric Lang
  Tools used: CAESAR, CAESAR.ADT, BCG_GRAPH, BISIMULATOR, PROJECTOR, SVL

- demo_34: Computer integrated manufacturing (CIM) architecture
  Radu Mateescu
  Tools used: BCG_MIN, CAESAR, CAESAR.ADT, EVALUATOR, SVL

- demo_35: Distributed summation algorithm using "n among m" synchronization
  Frédéric Lang
  Tools used: BCG_MIN, CAESAR, CAESAR.ADT, EXP.OPEN, SVL

- demo_36: Distributed Erathostenes sieve
  Frédéric Lang
  Tools used: BCG_LABELS, BCG_MIN, BISIMULATOR, CAESAR, CAESAR.ADT, EXP.OPEN, SVL

- demo_37: ODP (Open Distributed Processing) trader
  Frédéric Lang
  Tools used: BCG_MIN, BISIMULATOR, CAESAR.ADT, CAESAR, EXP.OPEN, PROJECTOR, SVL

- demo_38: New! Asynchronous circuit for the DES (Data Encryption Standard)
  Wendelin Serwe and Hubert Garavel
  Tools used: BCG_MIN, BISIMULATOR, CAESAR.ADT, CAESAR, EXEC/CAESAR, EXP.OPEN, PROJECTOR, SVL

- demo_39: New! Turntable system for drilling products
  Radu Mateescu
  Tools used: BCG_MIN, BCG_STEADY, BISIMULATOR, CAESAR, CAESAR.ADT, DETERMINATOR, EVALUATOR, SVL

- demo_40: New! Web services for stock management and on-line book auction
  Antonella Chirichiello, Gwen Salaun, and Wendelin Serwe

Random Basics

Stochastic Processes

Stochastic process

A stochastic process is a family of random variables $\{X(t) \mid t \in T\}$ defined on the same probability space $(\Omega, \mathcal{F}, P)$.

State space S

- For each t, $X(t) : \Omega \to S$ with S finite or countable.
- S is called the state space.

Time domain T

A stochastic process $\{X(t) \mid t \in T\}$ is called

- discrete-time if $T = \mathbb{N}$,
- continuous-time if $T = \mathbb{R}$.

Markov chains

Markov property:

the past influences the future only via the present. A stochastic process $\{X(t) \mid t \in \mathbb{R}\}$ is a Markov chain if it satisfies the Markov property: for all $0 = t_0 < t_1 < ... < t_n < t_{n+1}$ and $s_i \in S$:

$$\begin{aligned} P(X_{t_{n+1}} = s_{n+1} \mid X_{t_n} = s_n, X_{t_{n-1}} = s_{n-1}, \ldots, X_{t_0} = s_0) \\ = P(X_{t_{n+1}} = s_{n+1} \mid X_{t_n} = s_n) \end{aligned}$$

Discrete-time Markov chain:

For $T = \mathbb{N}$ we have an equivalent formulation:

$$\begin{aligned} P(X_{n+1} = s_{n+1} \mid X_n = s_n, X_{n-1} = s_{n-1}, \ldots, X_0 = s_0) \\ = P(X_{n+1} = s_{n+1} \mid X_n = s_n) \end{aligned}$$

Homogeneous DTMCs, graphically

We consider homogeneous Markov chains:

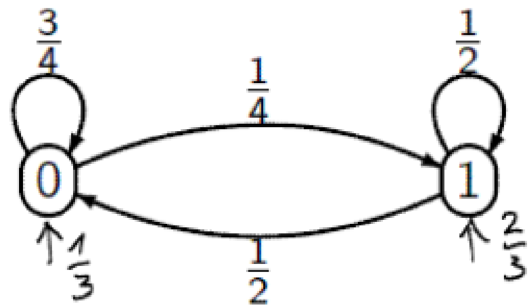
$$P(X_{n+1} = s' \mid X_n = s) = P(X_1 = s' \mid X_0 = s)$$

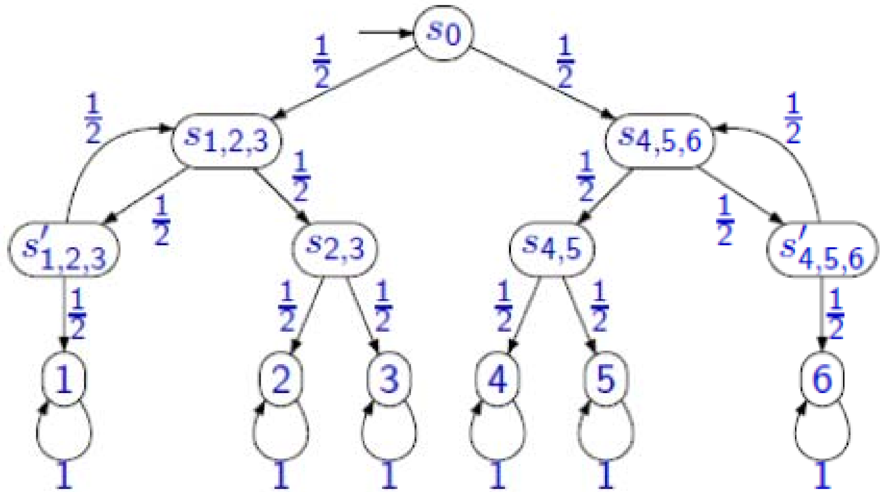
Graph-based definition

A homogeneous DTMC can be represented as a tuple: $(S, \mathbf{P}, \pi(0))$ where

- S is the set of states,
- $\mathbf{P} : S \times S \rightarrow [0, 1]$ with $\sum_{s' \in S} \mathbf{P}(s, s') = 1$ is the transition matrix,
- $\pi(0)$ is the initial distribution

This is the usual graphical representation.
Until further notice we restrict to finite S.





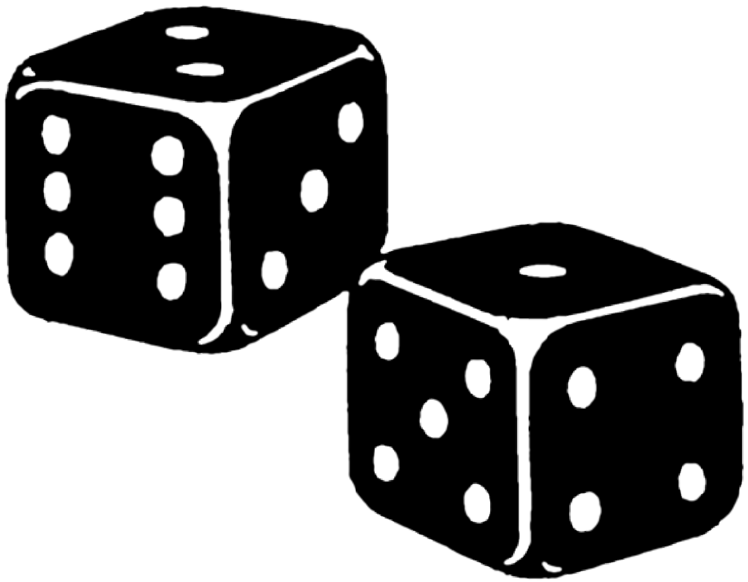
Example – Craps Gambling Game

First roll:

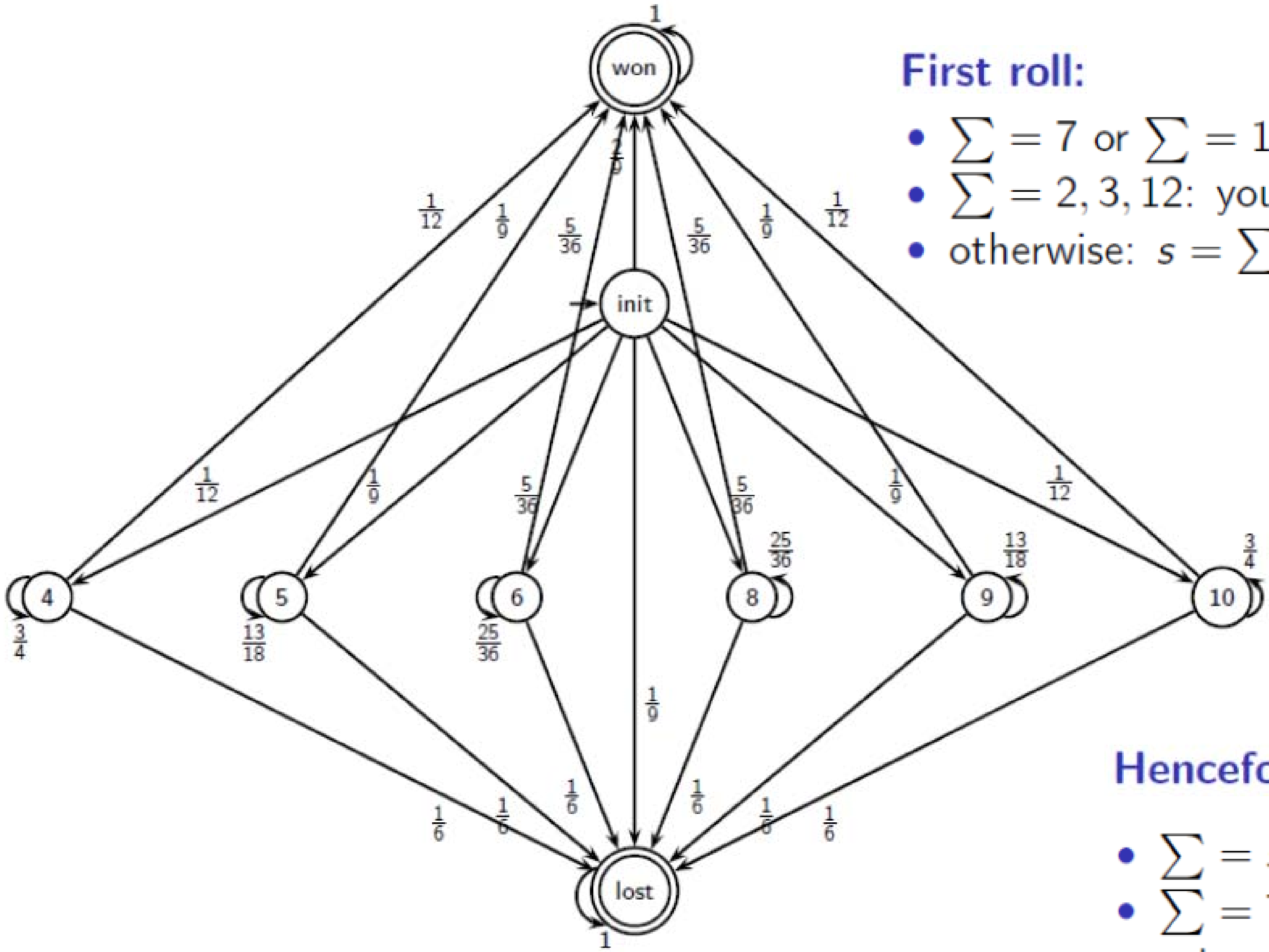
- $\sum = 7$ or $\sum = 11$: you **win**
- $\sum = 2, 3, 12$: you **lose**
- otherwise: $s = \sum$ is stored

Henceforth:

- $\sum = s$: you **win**
- $\sum = 7$: you **lose**
- otherwise: **repeat**



The Craps Gambling Game as a Markov Chain



First roll:

- $\sum = 7$ or $\sum = 11$: you **win**
- $\sum = 2, 3, 12$: you **lose**
- otherwise: $s = \sum$ is stored

Henceforth:

- $\sum = s$: you **win**
- $\sum = 7$: you **lose**
- otherwise: **repeat**

Real-world example: IPv4 Zeroconf Protocol

Why Zeroconf?

- Network administrators: assign addresses for IP hosts and network infrastructure
- Zeroconf: dynamic configuration of IPv4 Link-Local addresses
- even simple devices are able to communicate when attached
- simple and inexpensive for this form of networking

Zeroconf

- new hosts: randomly pick an address among the K (65024) addresses
- with m hosts in the network, collision probability is $\frac{m}{K}$
- the host asks other hosts whether they are using this address
- lossy channel: probability of no answer in case of collision is p

Zeroconf as a Markov chain