Seminar *Trends in Computer-Aided Verification*

Introduction

Winter Semester 2015/16; October 28, 2015

H. Bruintjes, S. Chakraborty, C. Jansen, T. Lange, T. Noll, F. Olmedo, H. Wu
Software Modeling and Verification Group
RWTH Aachen University

`http://moves.rwth-aachen.de/teaching/ws-1516/cav/`

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

Outline

Formal Verification Methods

Formal verification methods

- Rigorous, mathematically based techniques for the specification, development and verification of software and hardware systems
- Aim at improving correctness, reliability and robustness of such systems

Formal Verification Methods

Formal verification methods

- Rigorous, mathematically based techniques for the specification, development and verification of software and hardware systems
- Aim at improving correctness, reliability and robustness of such systems

Classifications

- According to design phase
  - specification, implementation, testing, ...
- According to specification formalism
  - source code, process algebras, timed automata, Markov chains, ...
- According to underlying mathematical theories
  - model checking, theorem proving, static analysis, ...

Outline

Aims of this Seminar

Goals

Aims of this seminar

- Independent understanding of a scientific topic
- Acquiring, reading and understanding scientific literature
- Writing of your own report on this topic
- Oral presentation of your results

Requirements on Report

Your report

- Independent writing of a report of $\approx$ 15 pages
- Complete set of references to all consulted literature
- Correct citation of important literature
- Plagiarism: taking text blocks (from literature or web) without source indication causes immediate exclusion from this seminar
- Font size 12pt with "standard" page layout
- Language: German or English
- We expect the correct usage of spelling and grammar
  - $\geq$ 10 errors per page $\Longrightarrow$ abortion of correction
- Report template will be made available on seminar web page

Requirements on Talk

Your talk

- Talk of about 45 (= 40 + 5) minutes
- Focus your talk on the audience
- Descriptive slides:
  - $\leq$ 15 lines of text
  - use (base) colors in a useful manner
- Language: German or English
- No spelling mistakes please!
- Finish in time. Overtime is bad
- Ask for questions

Final Preparations

Preparation of your talk

- Setup laptop and projector ahead of time
- Use a (laser) pointer
- Number your slides
- Multiple copies: laptop, USB, web
- Have backup slides ready for expected questions

Outline

Important Dates

Deadlines

- 30.11.2015: Detailed outline due
- 11.01.2016: Report due
- 01.02.2016: Slides due
- 11./12.02.2016 (???): Seminar

Important Dates

Deadlines

- 30.11.2015: Detailed outline due
- 11.01.2016: Report due
- 01.02.2016: Slides due
- 11./12.02.2016 (???): Seminar

Missing a deadline causes immediate exclusion from the seminar

Outline

Selecting Your Topic

Procedure

- You obtain(ed) a list of topics of this seminar.
- Indicate the preference of your topics (first, second, third).
- Return sheet by Friday (30 October) via e-mail to `tim.lange@cs.rwth-aachen.de` or to secretary.
- We do our best to find an adequate topic-student assignment.
- Disclaimer: no guarantee for an optimal solution.
- Assignment will be published on website by 2 November.
- Please give language preference
  - unsure $\Longrightarrow$ German

Selecting Your Topic

Procedure

- You obtain(ed) a list of topics of this seminar.
- Indicate the preference of your topics (first, second, third).
- Return sheet by Friday (30 October) via e-mail to `tim.lange@cs.rwth-aachen.de` or to secretary.
- We do our best to find an adequate topic-student assignment.
- Disclaimer: no guarantee for an optimal solution.
- Assignment will be published on website by 2 November.
- Please give language preference
  - unsure $\Longrightarrow$ German

Withdrawal

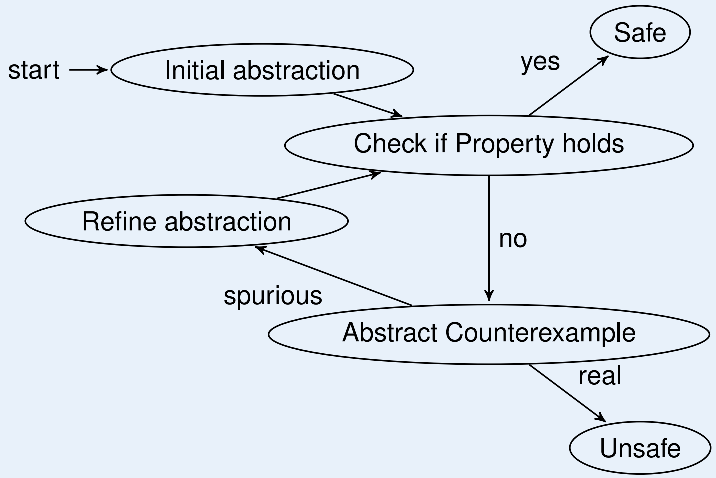
- You have up to three weeks to refrain from participating in this seminar.
- Later cancellation (by you or by us) causes a not passed for this seminar and reduces your (three) possibilities by one.

Outline

1: Efficient Abstraction Refinement

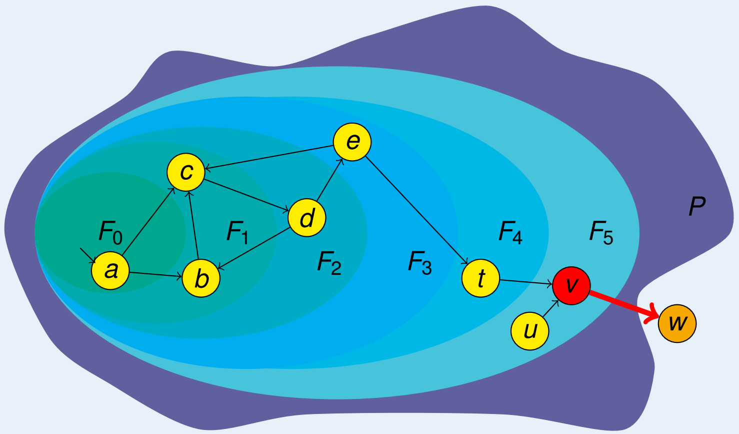
CEGAR (Software MC)

- Unroll transition relation until you find an abstract error state
- For every found error node: check full path



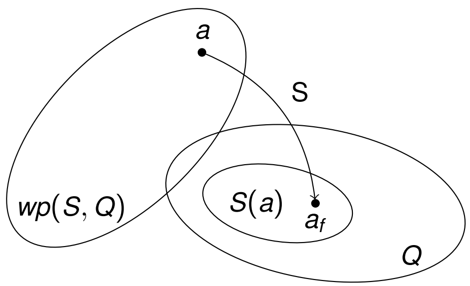
IC3 (Hardware MC)

- Construct stepwise refinement of reachable states
- Every Counterexample is a one-step counterexample



- Fuse: One step checks for refinement: CTIGAR

2: Efficient Computation of Weakest Preconditions



Weakest preconditions

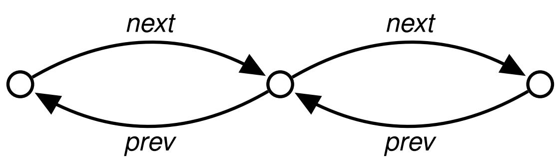
- Given a program statement S, and an execution state Q
- What state P can reach Q after executing S?

Important question in every software model checking algorithm.

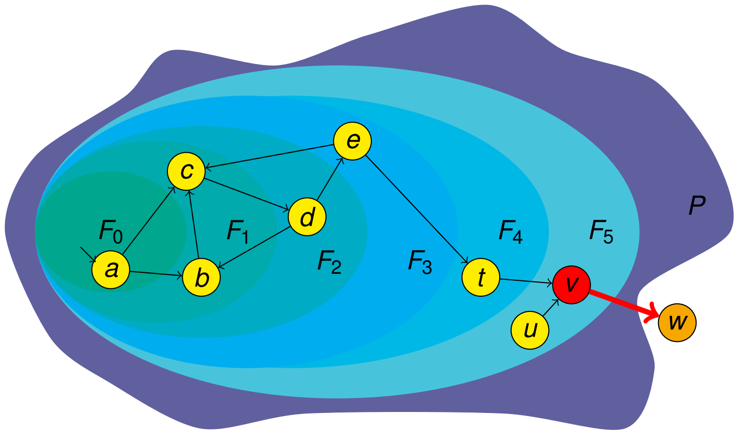
Naive algorithm

For every assignment $x := e$ of expression e to variable x, replace x with e in Q ($Q[x \mapsto e]$)
Problem: $\left|S_1; \ldots; S_n\right| = 2^n$ with $S_i = x := x + x$

3: Property-Directed Inference



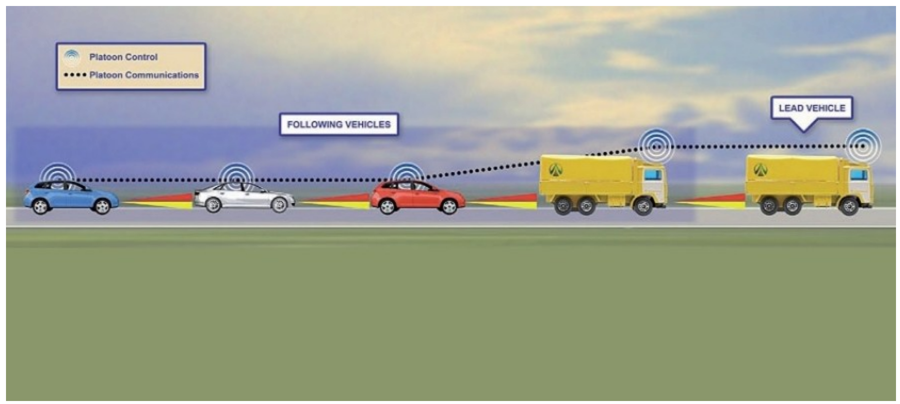
- Inference of Universal Invariants is a very hard problem
- Example: Sorted insert into arbitrary list, Memory safety
- Use modification of IC3 algorithm to infer invariants or prove their absence



Outline

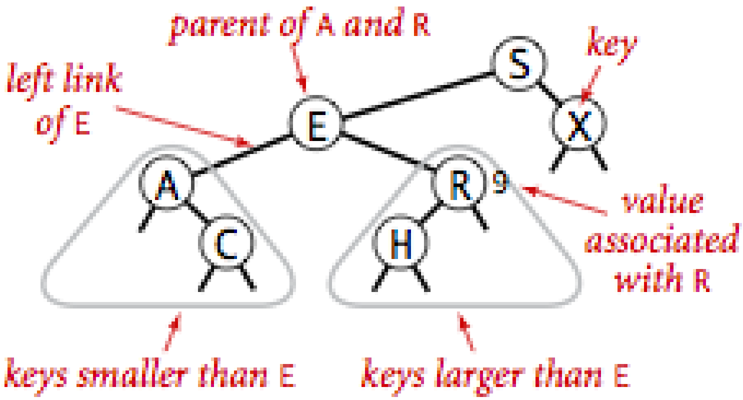
4: Analysis of infinite-state graph transformation systems

UNIVERSITÄT DES SAARLANDES



- analysis of distributed systems, e.g. protocols for car platooning or drone swarms
- system model as graph transformation system
- challenge adressed: unbounded numbers of agents, concurrency
- restriction: safety properties only
- solution: abstraction of graph transformation system

5: Analysis of heap structures with data



Anatomy of a binary search tree

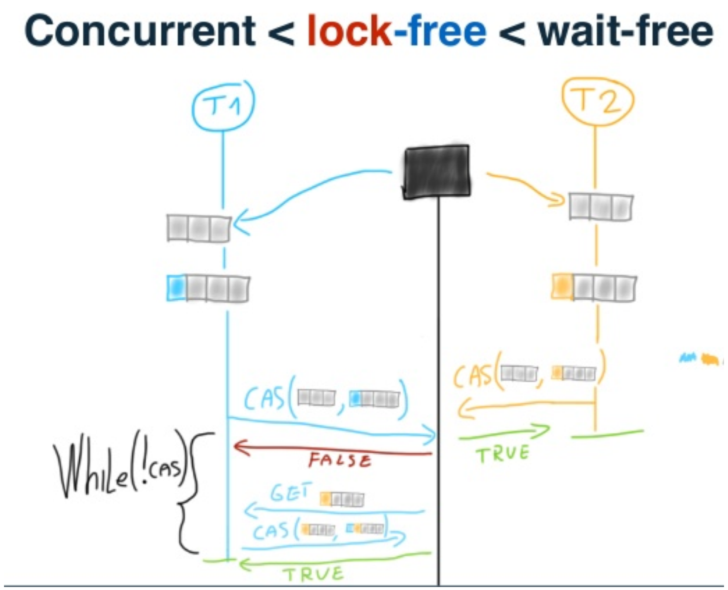
- analysis of pointer programs with values
- abstract interpretation-based
- aim: fully automatic inference of invariants
- tool Sample

6: Analysis of concurrent data structures

UPPSALA UNIVERSITET

BRNO UNIVERSITY OF TECHNOLOGY

LINKÖPING UNIVERSITY



- lock-free data structures are hard to write: verify them!
- specs as automata
- instrument program: generate sequence of events
- model checking: handle unboundedness by symbolic encoding

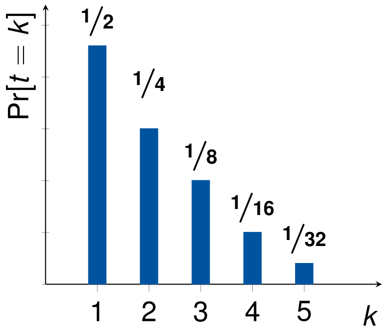
Outline

7: Cost-Based Analysis of Probabilistic Programs

- **Problem:** Determine the *average resource consumption* of a probabilistic program.

PROBABILISTIC PROGRAM + RESOURCE–CONSUMPTION ANNOTATIONS → AVERAGE RESOURCE–CONSUMPTION OF THE PROGRAM

$$\texttt{repeat}\ \{b := heads\}\ [1/2]\ \{b := tail\};\ \langle 1 \rangle\ \texttt{until}\ (b := heads)$$



$$\mathbb{E}[t] = 1 \cdot \frac{1}{2} + 4 \cdot \frac{1}{4} + 3 \cdot \frac{1}{8} + \cdots = \mathbf{2}$$

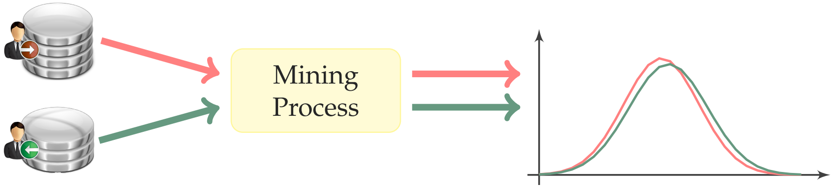
- **Solution Overview:** Reason *inductively on the program structure* by means of operator

$$\Delta(c) : \mathbb{S} \to \mathbb{R}^{\geq 0}_{\infty}$$

8: Relational Hoare Logic for Probabilistic Programs

- **Problem:** prove that two probabilistic programs produce "similar" outputs, and quantify this similarity.

Motivated by the notion of *differential privacy*, a confidentiality policy for the mining of sensitive data.



- **Solution Overview:** use a *quantitative relational Hoare logic* with judgments

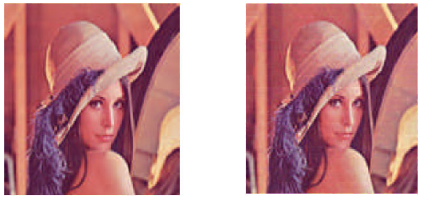
$$\{P\}\ c_1\ \sim_{\epsilon,\delta}\ c_2\ \{Q\}$$

c_1, c_2 : probabilistic programs
P, Q : relational pre– and post–condition
ϵ, δ : error bound

9: Correctness of Approximate Computations

- **Problem:** prove that an “approximate” version of a computation preserves its specification.

Motivated by the fact that the approximate version presents *increased performance* (at the expense of only a small precision loss).



exact approximated

- **Solution Overview:**
  - Consider an approximate program as a *non–deterministic abstraction* of the original program.
  - Embed the semantics of both (original and approximate) programs c_{o} and c_{a} in a single (compound) program $c_{(\mathrm{o},\mathrm{a})}$.
  - Use a Hoare logic with relational assertions to reason about the compound program $c_{(\mathrm{o},\mathrm{a})}$.

ORIGINAL PROGRAM c_{o} :

$a := A[i]$

COMPOUND PROGRAM $c_{(\mathrm{o},\mathrm{a})}$:

$a := A[i]\,;$
$\mathrm{orig\_a} := a\,;$
$\texttt{relax}\ (a)\ \texttt{st}\ (|a - \mathrm{orig\_a}| \leq \epsilon)$

HOARE TRIPLE :

$\{\mathrm{true}\}\ c_{(\mathrm{o},\mathrm{a})}\ \{a_{\langle o\rangle} = A[i] \wedge a_{\langle a\rangle} \leq A[i]+\epsilon\}$

Outline

10: Formal requirements engineering

Paper *Aligning Qualitative, Real-Time, and Probabilistic Property Specification Patterns Using a Structured English Grammar* by Marco Autili, Lars Grunske, Markus Lumpe, Patrizio Pelliccione and Antony Tang

- Properties are logic formulas that can be used to formally verify some behavior: Formally well defined, hard to use for non-experts.
- To make Specification of properties easier, patterns can be used, for example:
  If [something] happens, it will be followed by [something else].
- This paper takes existing categories of patterns and underlying logics, and extends them for more coverage
- Additionally, they are mapped to a (semi-)English grammar for ease of use.
- A tool is developed as well to aid in specifying properties using patterns.

11: Contract-based safety assessment

Paper *Formal Safety Assessment via Contract-Based Design* by Marco Bozzano, Alessandro Cimatti, Cristian Mattarei and Stefano Tonetta

- Considers two aspects of system design:
  - Hierarchical design: Decompose systems into subsystems, refine system requirements into sub-requirements etc.
  - Safety assessment: Analyze consequence of a fault (in a subsystem) on the system (e.g. causing a failure).
- Contract based design is used for the hierarchy: Define assumptions and guarantees of components to characterize systems.
- Enables the generation of hierarchical Fault Trees (as opposed to simple flat Fault Trees), which are a graphical representation of how low level faults can cause high level failures.

12: Probabilistic safety and liveness

Paper *Probably safe or live* by Joost-Pieter Katoen, Lei Song and Lijun Zhang.

- Safety and liveness: Something (bad) will not happen, or something (good) will eventually happen.
- Practical reason to distinguish the two: Safety properties can be analyzed easier with different model checking algorithms
- In this paper: Look at safety and liveness in the probabilistic setting:
  - Look at probabilistic properties: Check the probability of some behavior
  - Which properties can be classified as safe, which as live?

Outline

13: The Cyclic-Routing Problem

14: Expressive Completeness for Metric Temporal Logic

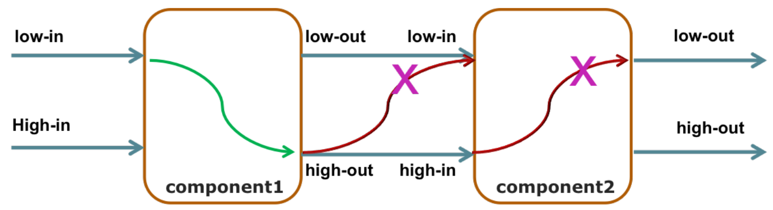
15: Solving Partial-Information Stochastic Parity Games

Outline

16: Type-based information flow analysis

Information flow security

- Confidentiality (secrets kept)
- Integrity (data not corrupted)



Example (encyption/decryption)

$$\frac{T \vdash e_1 : \tau \quad T \vdash e_2 : \texttt{key L}}{T \vdash \mathrm{encrypt}(e_1, e_2) : \texttt{enc}\ \tau\ \texttt{L}}$$

$$\frac{T \vdash e_1 : \texttt{enc}\ \tau\ \sigma \quad T \vdash e_2 : \texttt{key H}}{T \vdash \mathrm{decrypt}(e_1, e_2) : \tau^{\sigma}}$$

The type-based approach

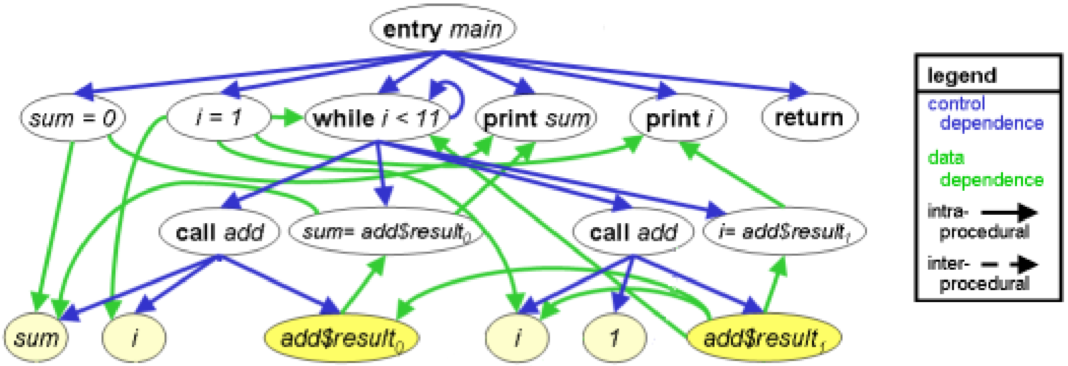
- Type system for tracking information flow in programs
- Associates security levels (L, H) with variables
- Program is secure if final value of low variables independent of initial value of the high variables
- Extension: cryptographic operations

17: Information flow analysis based on program dependence graphs

Information flow: which high inputs influence which low outputs?

Program dependence: which outputs depend on which inputs?

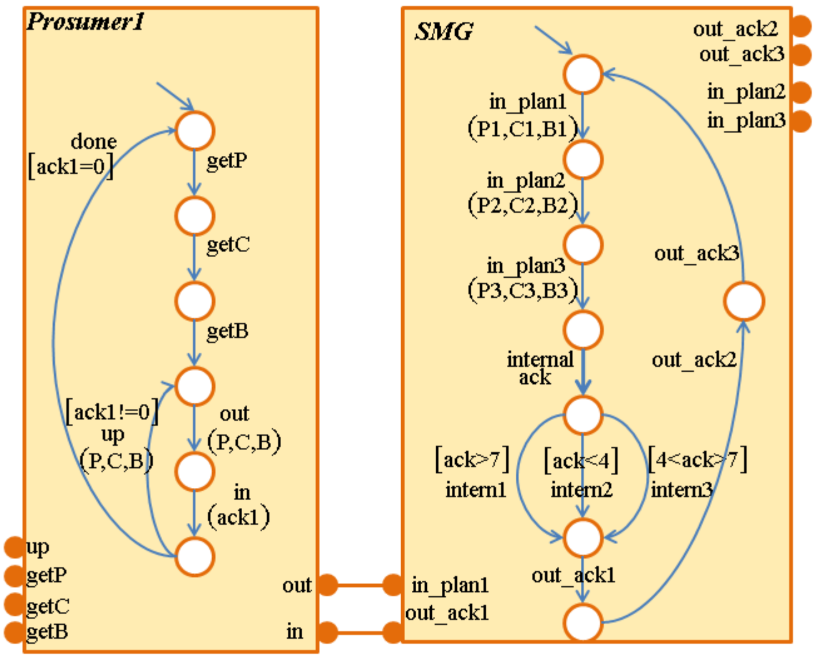
- interesting output values define slicing criterion
- backward analysis of information flow based on program dependence graph



Applications

- Debugging
- Testing
- Model checking
- Information flow security
  - if no high variable in the backward slice of any low output, then system is secure
  - interprocedural extension by context-sensitive slicing

18: Model-driven information flow analysis



- So far: analysis of information flow on source-code level
- Now: define and verify security policy from early steps of system design
- Here: BIP specification language (Behaviour-Interaction-Priority)
- Formal definition of two non-interference properties
  - event: observation of public events should not allow to deduce any information about occurrence of secret events
  - data: no leakage of secret data into public ones
- Automatic analysis of non-interference

Outline

Hao Wu: Formal Methods in System Design

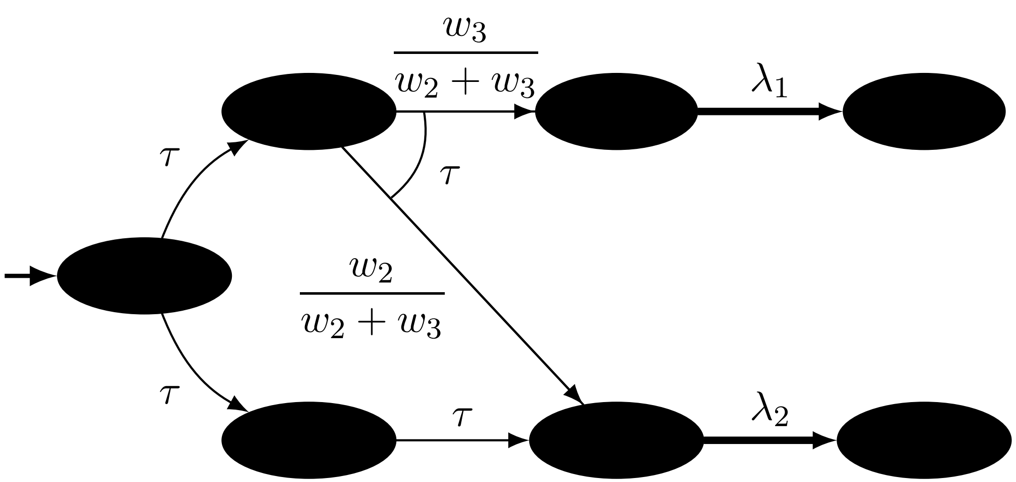
19: Model checking and performance of shared-memory mutex protocols

- Studied objects – mutual exclusion protocols (such as Peterson's/Dekker's algorithms)
- Two Interested aspects: functional correctness & performance evalution

  Functional properties:
  - Mutual exclusion.
  - Livelock freedom.
  - Starvation freedom.
  - Bounded overtaking.
  - etc.

  Performance evaluation is based on interactive Markov chain (IMC):
  - Throughtput
- All studied algorithms are modeled and verified using the CADP toolbox from Inria, France.

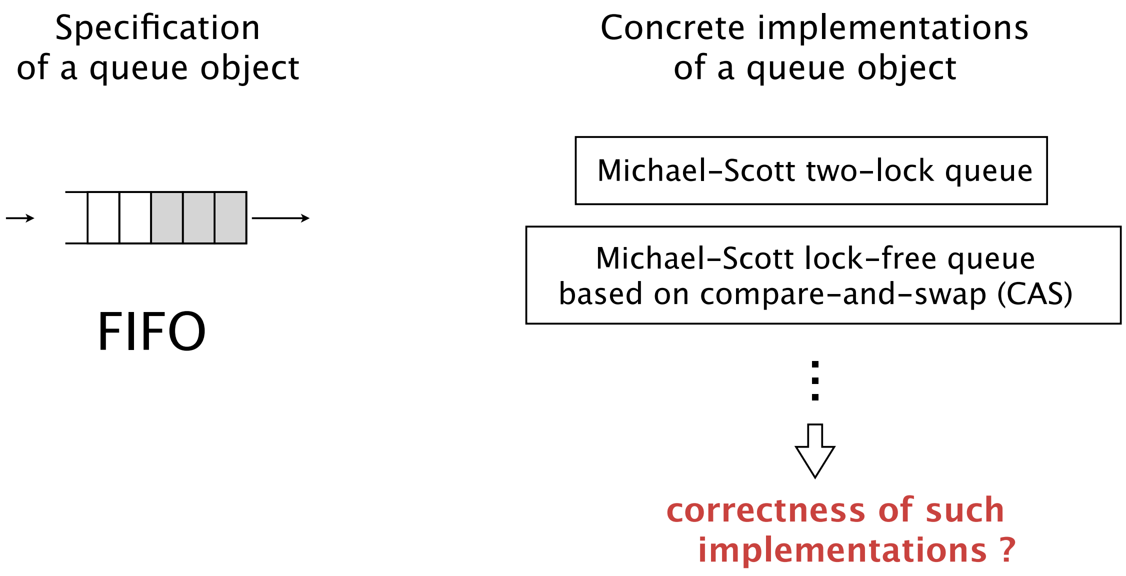
20: Modelling and Analysis of Markov Automata



- Markov automata (MAs) = LTS with random delays + probabilistic choices
- The expected time objectives considered in MA
  - The minimal/maximal expected time to reach a set of target states
- The long-run objectives considered in MA
  - The minimum/maximum long-run average time spend in a set of target states
- The timed reachability objectives in MA
  - The minimum/maximum probability to reach a set of target states in a given time interval

21: Model Checking Linearizability via Refinement

- Linearizability is an important correctness criterion for implementations of concurrent objects.
- Specification and implementations of a concurrent object



- This paper provide a new approach to automatically verify linearizability based on refinement relations from abstract specifications to concrete implementations.

Outline

Some Final Hints

Hints

- Take your time to understand your literature.
- Be proactive! Look for additional literature and information.
- Discuss the content of your report with other students.
- Be proactive! Contact your supervisor on time.
- Prepare the meeting(s) with your supervisor.
- Forget the idea that you can prepare a talk in a day or two.

Some Final Hints

Hints

- Take your time to understand your literature.
- Be proactive! Look for additional literature and information.
- Discuss the content of your report with other students.
- Be proactive! Contact your supervisor on time.
- Prepare the meeting(s) with your supervisor.
- Forget the idea that you can prepare a talk in a day or two.

We wish you success and look forward to an enjoyable and high-quality seminar!