Chapter 3

Specification models:

High-level modeling techniques and related analysis methods for the computer assisted verification of DES

Computer Engineering and Networks Laboratory

Discrete Event Systems

Fall 2008

Motivation (1)

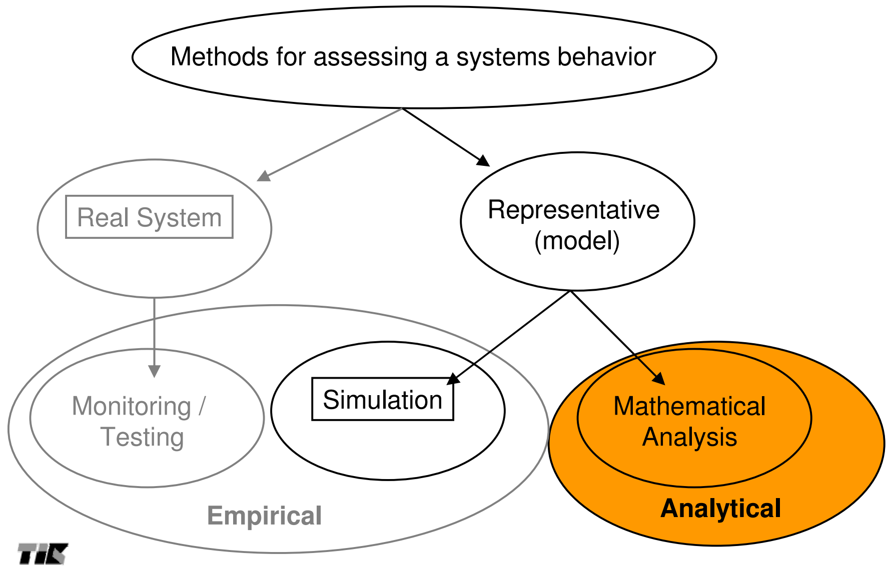
- **Modern systems consist of many different components (HW + SW !), yielding a very high degree of complexity**
- **Systems have to fulfill a set of requirements, defined by a specification as given by a contractor, customer or legislation**
- **What are these kinds of requirements?**
  - Functionality:
    Coke vending machines either delivers drink or returns my money
  - Performance:
    Voice-of-IP requires max. delay of a IP-packet < xxx msec.
  - Energy-consumption, heating characteristics
  - Reliability: 99,999% of emergency calls must be routed correctly
  - Safety / Security requirements (can been seen as part of functionality)
  - Economic requirements: costs, amortization etc.

Motivation (1)

Why do we need more sophisticated methods other than finite state machines?

- States are atomic
  - no hierarchic structuring possible
  - usage of variables
- Partitioning of systems in (parallel operating) components not possible
  - modularization?
- FSM are easily very large and thus not human readable anymore

Motivation (3)



Computer-assisted analysis/verification (1)

What's the general strategy for formally and automatically analyzing models?

This will be covered here

1. Take a design of the system behavior, given as some high-level model such as an SDL specification, PN, network of TA, or ……
2. Take (a set of) formal system requirements, e.g. given in terms of a set of Message Sequence Charts, as safety or progress properties, …
3. Validate that each requirement is indeed satisfied by the system design.

Computer-assisted analysis/verification (2)

What are the techniques for formally analyzing models?

1. Theorem proving
   - Strategy: generate a formal proof that *D* satisfies ϕ.
   - Applicable if design *D* can be represented in some adequate mathematical theory
2. **State space exploration**
   - **Strategy: check systematically and exhaustively each reachable state in *D* satisfies ϕ.**
   - **Applicable if the behavior of *D* can be finitely represented.**
   - **One is enabled to show the presence and absence of errors!**
3. Simulation or Testing of models
   - Strategy: Check whether ϕ holds on some executions of *D.*
   - Applicable if *D* is in some sense executable.
   - **One may be able to show the presence of errors, but not the absence!**

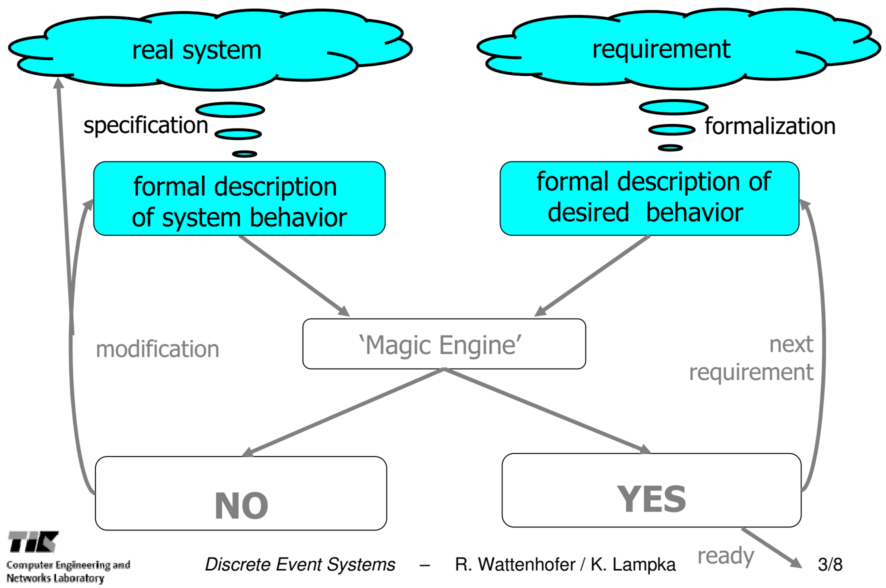
(State-based) Computer-assisted analysis/verification (3)

What are the requirements to be verified (by state-based methods)**?**

1. **Safety**: A safety property to be verified asserts that a system under analysis never reaches a (set of) dedicated state, e.g. like error states, or in particular a deadlock. The mutual exclusion property is one of the most prominent examples of a safety property. (constraint on finite behaviour)
2. **Liveness or progress**: A liveness property guarantees that a system under analysis is executing a (set) of dedicated activities infinitely often (constraint on infinite behaviour)
3. **Starvation** exists if there is an infinite run, where a dedicated action is never executed.

Dijkstra'71: Dining Philosophers Problem
en.wikipedia.org/wiki/Dining_philosophers_problem

(State-based) Computer-assisted analysis/verification (4)



(State-based) Computer-assisted analysis/verification (5)

Does a design *D* satisfy a requirement ϕ?

What are the practical obstacle? ☞ Space and CPU-time is limited

- State Space Explosion:
  The number of possible state combinations *exponentially*
  in the number of concurrent processes or independent activities.

- Captures your model the reality ? ☞ Validation of model
  – You can only assert those properties which are captured by the model.
  – Consistency check between model and reality (= model validation, not covert here).

What's the principal obstacle? ☞ Halting problem

- Decidability:
  full generality it is undecidable whether *D* satisfies ϕ.
  This depends on the specification, and the requirement.

Validation ↔ Verification

- Validation ("doing right things")
  – Comparing theory to reality
  – Make observation for providing evidence that the theory is correct, e.g.
    - show the absence of a specific error in different runs
    - show that program delivers correct result with respect to a given input
  – If possible try to find counter-examples (Falsification)

- Verification ("doing things right")
  – Proof correctness of system design
  – formal model with unique interpretation
  – formal system, i.e. formal model and unique set of deductive rules for operating on (or transforming) the model.
  – If incorrect behavior is detected, counter-example is automatically provided

Specification Models (at glance)

- **Systems have to fulfill a set of requirements, defined by a specification as given by a contractor, customer or legislation**
- **Showing (proving) a dedicated behavior of a system needs exhaustive analysis:**
  – **Testing, monitoring and simulation of real system or a model is in principle not sufficient** (rare events?)
- **Verification of systems require their formal description**
- **What does formal mean?**
  – **Model posses a unique interpretation (unique set of deductive rules to operate on the model)**

Early versions of UML-state charts ?

- **Drawback: Reality far from trivial**
  => level of detail bounded by capabilities of (mathematically) handling a model (run-time and memory is limited)
- Abstraction:
  Keep models simple as possible but as complex as necessary!
  You can only check what you have modeled!

What are we doing?

- **Lecture 23.10:**
  – **Specification and Description Language SDL**
  – **Message Sequence Charts**
  – Related Analysis methods: The TAU-Tool-suite
- **Lecture 30.10:**
  – **Petri Nets**
  – **Symbolic Analysis methods (for finite models)**
- **Lecture: 6.11:**
  – **Timed Automata**
  – **Introduction to model checking**