Universität Konstanz
Fachbereich Informatik & Informationswissenschaft
Stephan Heidinger

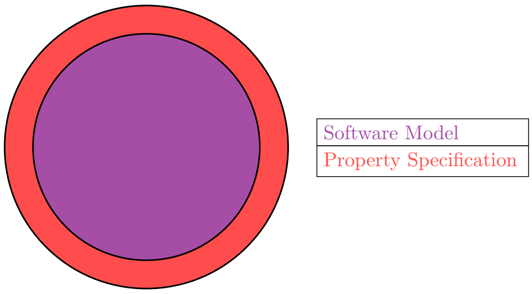
☉_☉ *Find any errors? Please send them back, I want to keep them!*

Basic Principle

given:

- Software Model M
- Property Specification S

does $M \models S$



Classification of Software Systems

Transformational Systems transforms set of (empty) input data into output data, function from state S_i to S_k
correctness:

- termination
- correctness of function $S_i \rightarrow S_k$
- correctness of input - output transformation

Reactive Systems ongoing interaction with environment, driven by environment
correctness:

- non-termination (normally)
- correctness of stimuli-response pairs

Embedded Systems usually reactive, directly connected to hardware

Cyber-Physical Systems integration of computation and physical processes

Real-Time Systems systems, where correctness depends on time a result is delivered

Soft Real-Time Systems missed deadlines will decrease result, not lead to failure, propabilities

Hard Real-Time Systems missed deadlines will lead to failure

Hybrid Systems state is characterized by discrete and continuous variables

Safety-Critical Systems systems failure may entail, death, serious injury, environmental harm, damage to property/assets

Requirement Specification

Natural Language

- ✚ very expressive
- ✚ understood by all parties

- ➘ ambiguous

Formal Language

- ✚ unambigous
- ✚ machine-analyzeable
- ➘ limited expressiveness
- ➘ hard to understand

Software Verification Method

Requirements

- formal foundation (automatic procedures)
- should be capable of relating artifacts from different stages in design cycle (formal vs informal req, design vs req, ...)
- should be easy to integrate in design cycle (high degree of automation, low degree of interaction)
- scalable

Model Checking Process

1. provide model (e.g. Promela), involves abstraction
2. simulate (check if model does, what you want)
3. elicit and formalize requirements ⇒ property specification
4. model execution: run model checker with model and specification
   outcome:
   - property is valid (check next property)
   - property is invalid (counterexample, check model for errors, check properties for errors, rethink design)
   - exhaust of memory (use more abstraction, state space reduction, incomplete methods)
   - exhaust of time (smaller model, faster computer)

Spin

- designed for communication protocols
- Open Source
- still in development
- Promela (PROtocol/PROcess MEta LAnguage)
  - concurrent modeling language
  - guarded commands
  - modeling of reactive systems

State-Based Modeling

State

- salient features of a system at given point of observation
- states can be observed, as long as features of interest unchanged

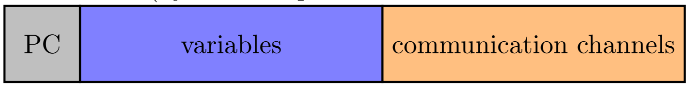
- features of interest

  point of control "program counter" (of all processes)

  values of local and global variables

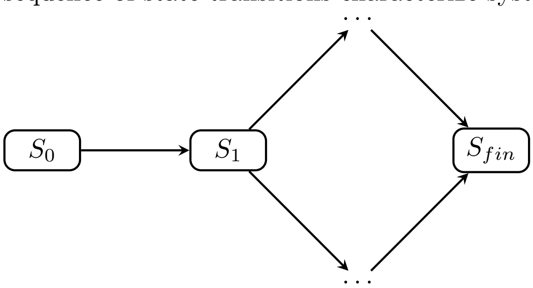
  communication channels (messages sent but not received)

- state vector (byte-wise representation of features of interest)



State Transition

- instantaneous change of observed features of the system ("something happens")
- represents computation step
- sequence of state transitions characterize system computation



- in real-time systems, time passes in states
- in stochastic systems, transitions are labeled with probability distributions
- in hybrid systems, continuous and discrete state variables change during state transition
- transitions show the possible events
- transition sequences are valid computations sequences
- transitions encode history (S_0 has been visited before S_1)

Devising State-Machines

- "programm a model"
- abstraction: focus on relevant, can lead to non-determinism
- simplicity: find most simple abstraction that still reveals phenomena

Deadlock

- concurrent processes wait for each other in a circular wait with no pre-emption
- highly undesired

Closed System Modeling

- model checker can only validate all possible system executions under assumed environment
- model includes also environment

Transition Systems

Transition System TS is a tuple $(S, Act, \rightarrow, I, AP, L$ where

S	is a set of states
Act	is a set of actions
$\rightarrow \subseteq S \times Act \times S$	is a transition relation
$I \subseteq S$	is a set of initial states
AP	is a set of atomic propositions
$L : S \rightarrow 2^A P$	is a labeling function

- where S and Act are finite or countably infinte
- we write $s \rightarrow^\alpha s'$ for $(s, \alpha, s') \in \rightarrow$
- Atomic Propositions: Facts that we want to observer/that are observable in the system in any given state.
- Labeling Function: shows, which atomic propositions hold in given state.

$$Post(s,\alpha) = \left\{ s' \in S | s \xrightarrow{\alpha} s' \right\} \qquad Post(s) = \bigcup_{\alpha \in Act} Post(s,\alpha)$$

$$Pre(s,\alpha) = \left\{ s' \in S | s' \xrightarrow{\alpha} s \right\} \qquad Pre(s) = \bigcup_{\alpha \in Act} Pre(s,\alpha)$$

$$Post(C,\alpha) = \bigcup_{s \in C} Post(s,\alpha), \qquad Post(C) = \bigcup_{s \in C} Post(s) \text{ for } C \subseteq S$$

$$Pre(C,\alpha) = \bigcup_{s \in C} Pre(s,\alpha), \qquad Pre(C) = \bigcup_{s \in C} Pre(s) \text{ for } C \subseteq S$$

a state is **terminal** or **final** iff $Post(s) = \varnothing$

0.0.1 (Non)Determinism

A transition system $TS = (S, Act, \rightarrow, I, AP, L)$ is **action-deterministic**, iff for all s, α

- $|I| \leq 1$ and
- $|Post(s), \alpha)| \leq 1$

else it is **action-nondeterministic**

there is at most one outgoing transition from each state labeled α

A transition system $TS = (S, Act, \rightarrow, I, AP, L)$ is **AP-deterministic**, iff for all $s, A \in 2^{AP}$

- $|I| \leq 1$ and
- $|Post(s) \cap \{s' \in S | L(s') = A\}| \leq 1$

else it is **AP-nondeterministic**

every successor of a state s has a unique AP labeling

- Nondeterminism can lead to potentially smaller representation.
- in Software Engineering/Modeling: Abstraction
  - avoid overspecification
  - what does the system do, not how is it done
  - concurrency
    - simulate concurrency by nondeterminsm
    - either nondeterministic action can be executed first

System Execution

Given a transition system $TS = (S, Act, \to, I, AP, L)$

finite execution fragment ϱ of TS is an alternating sequence of states and actions ending with a state:

$$\varrho = s_0\alpha_1 s_1\alpha_2 \dots \alpha_n s_n \text{ s.t. } s_i \to^{\alpha_{i+1}} s_{i+1} \text{ for all } 0 \leq i < n$$

infinite execution fragment ϱ of TS is an infinite, alternating sequence of states and actions:

$$\varrho = s_0\alpha_1 s_1\alpha_2 \dots \alpha_n s_n \text{ s.t. } s_i \to^{\alpha_{i+1}} s_{i+1} \text{ for all } 0 \leq i$$

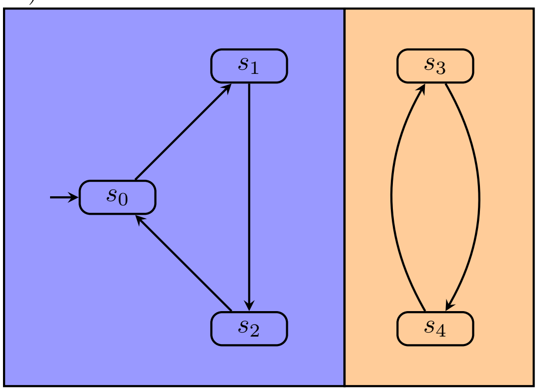
execution of TS is an inital, maximal execution fragment

- an execution fragment is maximal, iff it is either
  - finite and ending in a terminal state

  or
  - infinite
- an execution fragment is initial, iff $s_0 \in I$

reachable state is a state $s \in S$ if there exists an initial, finite execution fragment $s_0\alpha_1 s_1 \dots \alpha_n s_n$ so that $s_n = s$

Reach(TS) denotes the set of all reachable states in TS



Reach(TS) not reachable

Program Graphs PG

- want to include things like variables, assignements, etc $\Rightarrow$ Program Graphs
- introduce conditional transitions (transition can only be exectured, if condition is true)

$$s \xrightarrow{g:\alpha} s'$$

  - g: a boolean condition on data variables ("guard")
  - α: an action, that is possible, if g is satisfied
- assume domain of variables as infinite
  - practical implementations use variables over finite domains
- Stepwise unfolding of PGs lead to TS

valuation: assign values to variables (e.g. $\eta(x) = 17$)

propositional logic formulae

effect of actions:

$$Effect : Act \times Eval(Var) \to Eval(Var)$$

Effects define **operational semantics**

A **program graph** PG over set Var of typed variables is a tuple $(Loc, Act, Effect, \rightarrow, Loc_0, g_0)$ where

- Loc is a set of locations with initial locations $Loc_0 \subseteq Loc$
- Act is a set of actions
- $Effect : Act \times Eval(Var) \rightarrow Eval(Var)$ is the effect function
- $\rightarrow \subseteq Loc \times (Cond(Var) \times Act) \times Loc$, transition relation cond = boolean condition
- $g_0 \in Cond(Var)$ is the inital condition

Notation $\ell \xrightarrow{g:\alpha} \ell'$ denotes $(\ell, g, \alpha, \ell') \in \longrightarrow$

Semantics for Program Graphs

- Unfolding: Transformation of PG to equivalent TS
  - states:
    - state $< l, \eta >$: current control location l + data valuation η
    - initial state: initial location satisfying condition g_0
  - propositions:
    - at_l: control is at location l
    - $x \in D$ iff $D \subseteq dom(x)$
  - labeling
    - $< l, \eta >$ is labeled with at_l and all conditions that hold in η
- from transitions in PG to transitions in TS
  - if $\ell \xrightarrow{g:\alpha} \ell'$ and g holds in η, then $< l, \eta > \rightarrow^{\alpha} < l', Effect(\alpha, \eta) >$

Structured Operational Semantics Sos

- definition of **operational semantics** of a program in terms f computations steps defined by a transition system
- whether a step happens is determined by inference rules:

$$\frac{premise}{conclusion}$$

  - if the premise holds, the conclusion holds (and can trigger further inference rules)
  - if the premise is a **tautology**, it may be omited
    the rule is then called an **axiom**
- semantics is structural, because it applies inference rules recursively to syntactic structure

Transition Systems for Program Graphs

The transition system $TS(PG)$ of program graph

$$PG = (Loc, Act, Effect, \longrightarrow, Loc_0, g_0)$$

over set Var of variables is the tuple

$$(S, Act, \longrightarrow, I, AP, L)$$

where

- $S = Loc \times Eval(Var)$
- $\longrightarrow \subseteq S \times Act \times S$ is defined by $\dfrac{\ell \xrightarrow{g:\alpha} \ell' \ \wedge \ \eta \models g}{< \ell, \eta > \xrightarrow{\alpha} < \ell', Effect(\alpha, \eta) >}$
- $I = \{< \ell, \eta > | \ell \in Loc_0, \eta \models g_0\}$
- $AP = Loc \cup Cond(Var)$ and $L(< \ell, \eta >) = \{\ell\} \cup \{g \in Cond(Var) | \eta \models g\}$

Data in Transition Systems

- TS do not possess data variables, data values and their changes need to be encoded in states
- assume $i = 1, \ldots, n$ variables of domain s_i $\Rightarrow \Pi_{i=1}^{n} s_i$
  combinatorial, exponential state space explosion
- variables over infinite domains $\Rightarrow$infinite number of states