Transition Systems

Lecture #2 of Model Checking

Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling and Verification

E-mail: `katoen@cs.rwth-aachen.de`

October 22, 2008

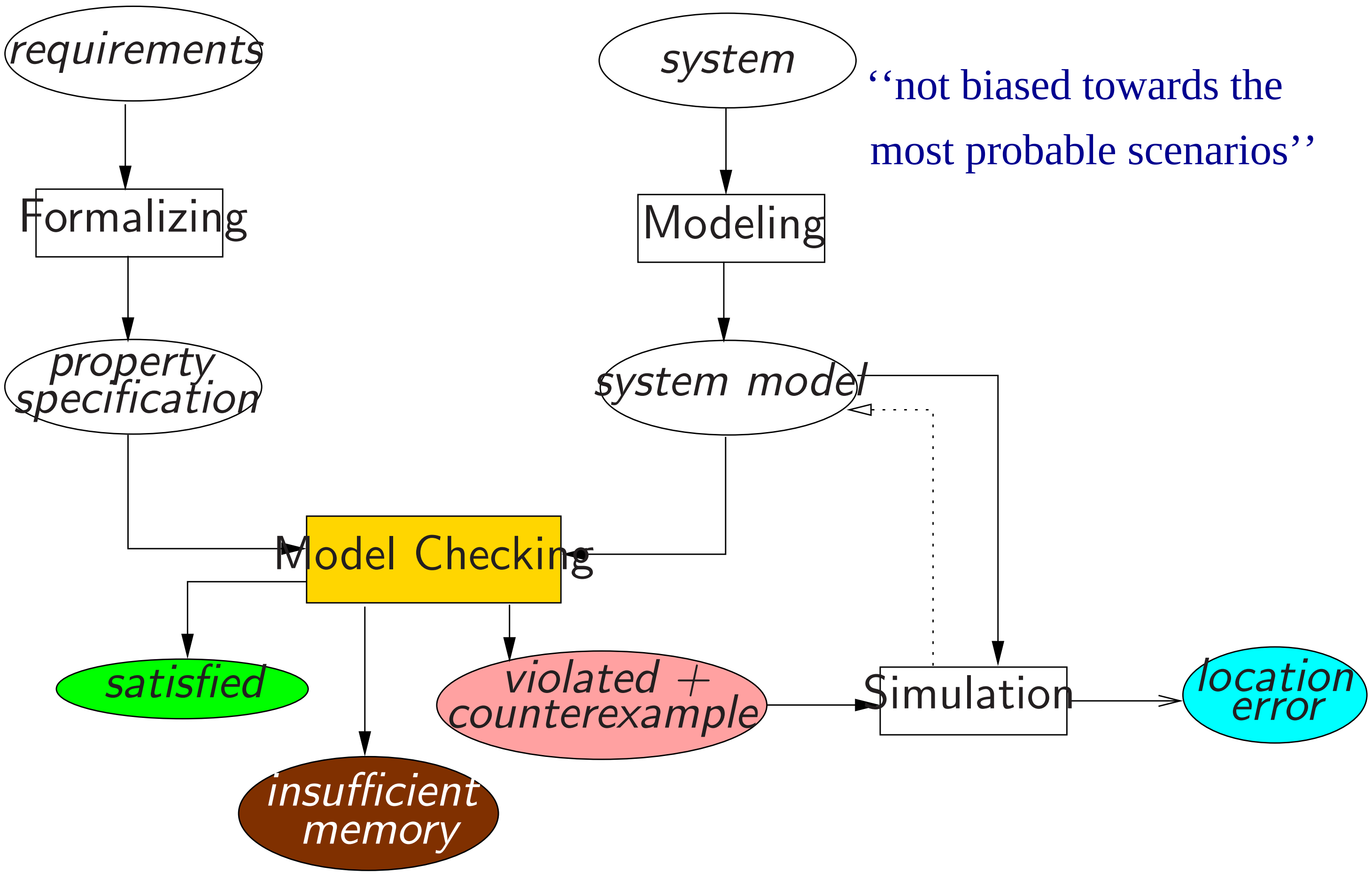
Overview Lecture #2

⇒ *Transition systems*

- Executions
- Modeling data-dependent systems

- *Parallelism and communication*
  - Interleaving
  - Shared variables

Recall model checking



''not biased towards the most probable scenarios''

Transition systems

- model to describe the behaviour of systems
- digraphs where nodes represent *states*, and edges model *transitions*
- state:
  - the current colour of a traffic light
  - the current values of all program variables + the program counter
  - the current value of the registers together with the values of the input bits
- transition: ("state change")
  - a switch from one colour to another
  - the execution of a program statement
  - the change of the registers and output bits for a new input

Transition system

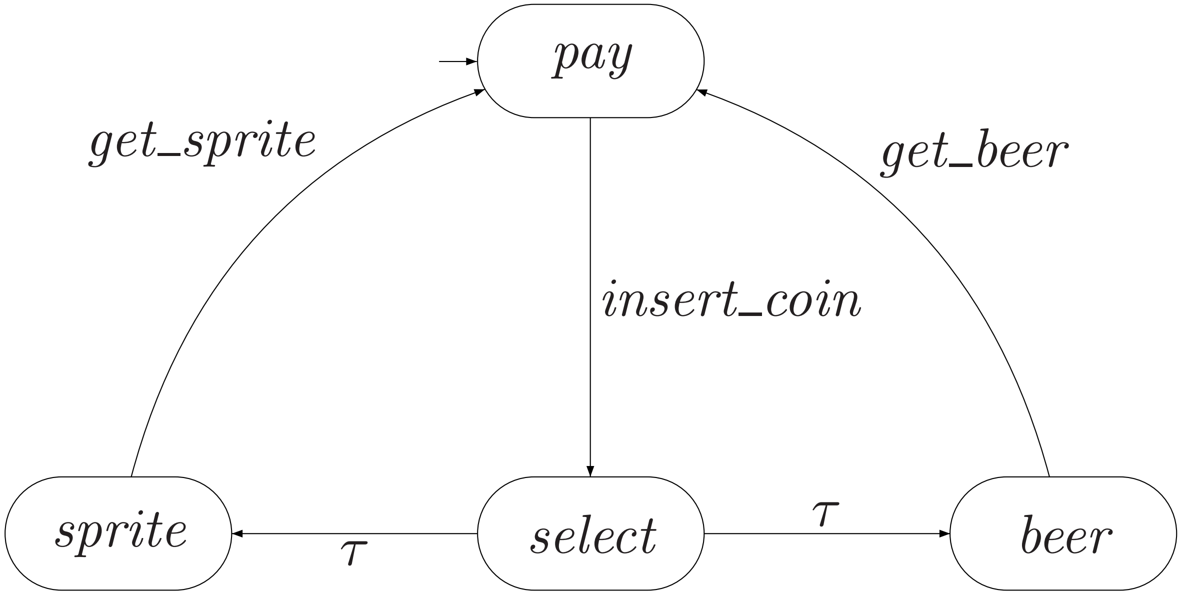
A *transition system* TS is a tuple $(S, Act, \to, I, AP, L)$ where

- S is a set of states
- Act is a set of actions
- $\longrightarrow \subseteq S \times Act \times S$ is a transition relation
- $I \subseteq S$ is a set of initial states
- AP is a set of atomic propositions
- $L : S \to 2^{AP}$ is a labeling function

S and Act are either finite or countably infinite

Notation: $s \xrightarrow{\alpha} s'$ instead of $(s, \alpha, s') \in \longrightarrow$

A beverage vending machine



states? actions?, transitions?, initial states?

Atomic propositions?

Direct successors and predecessors

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

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

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

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

State s is called *terminal* if and only if $Post(s) = \varnothing$

Action- and *AP*-determinism

Transition system $\mathit{TS} = (S, \mathit{Act}, \rightarrow, I, \mathit{AP}, L)$ is *action-deterministic* iff:

$$|\, I \,| \;\leqslant\; 1 \quad \text{and} \quad |\, \mathit{Post}(s, \alpha) \,| \;\leqslant\; 1 \qquad \text{for all } s, \alpha$$

Transition system $\mathit{TS} = (S, \mathit{Act}, \rightarrow, I, \mathit{AP}, L)$ is *AP-deterministic* iff:

$$|\, I \,| \;\leqslant\; 1 \;\text{ and }\; |\, \underbrace{\mathit{Post}(s) \,\cap\, \{\, s' \in S \mid L(s') = A \,\}}_{\text{equally labeled successors of } s} | \;\leqslant\; 1 \qquad \text{for all } s, A \in 2^{\mathit{AP}}$$

The role of nondeterminism

Here: nondeterminism is a feature!

- to model concurrency by interleaving
  - no assumption about the relative speed of processes
- to model implementation freedom
  - only describes what a system should do, not how
- to model under-specified systems, or abstractions of real systems
  - use incomplete information

in automata theory, nondeterminism may be exponentially more succinct
but that's not the issue here!

Executions

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

$$\varrho \;=\; s_0\,\alpha_1\,s_1\,\alpha_2\,\ldots\alpha_n\,s_n \text{ such that } s_i \xrightarrow{\alpha_{i+1}} s_{i+1} \text{ for all } 0 \leqslant i < n.$$

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

$$\rho \;=\; s_0\,\alpha_1\,s_1\,\alpha_2\,s_2\,\alpha_3\ldots \text{ such that } s_i \xrightarrow{\alpha_{i+1}} s_{i+1} \text{ for all } 0 \leqslant i.$$

- An *execution* of *TS* is an initial, maximal execution fragment
  - a *maximal* execution fragment is either finite ending in a terminal state, or infinite
  - an execution fragment is *initial* if $s_0 \in I$

Example executions

$$\rho_1 = pay \xrightarrow{coin} select \xrightarrow{\tau} sprite \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} sprite \xrightarrow{sget} \ldots$$

$$\rho_2 = select \xrightarrow{\tau} sprite \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} beer \xrightarrow{bget} \ldots$$

$$\varrho = pay \xrightarrow{coin} select \xrightarrow{\tau} sprite \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} sprite$$

Execution fragments ρ_1 and ϱ are initial, but ρ_2 is not

ϱ is not maximal as it does not end in a terminal state

Assuming that ρ_1 and ρ_2 are infinite, they are maximal

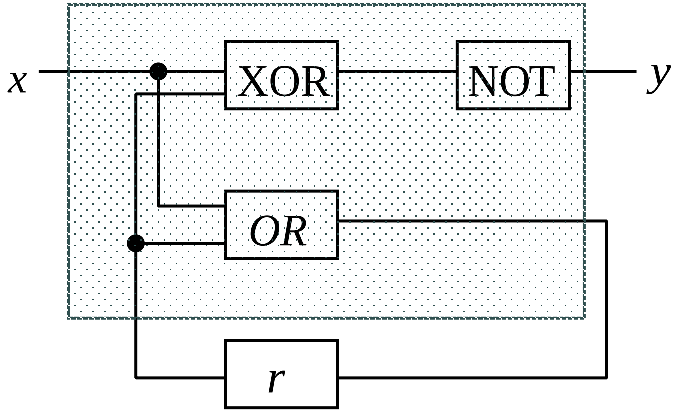
Reachable states

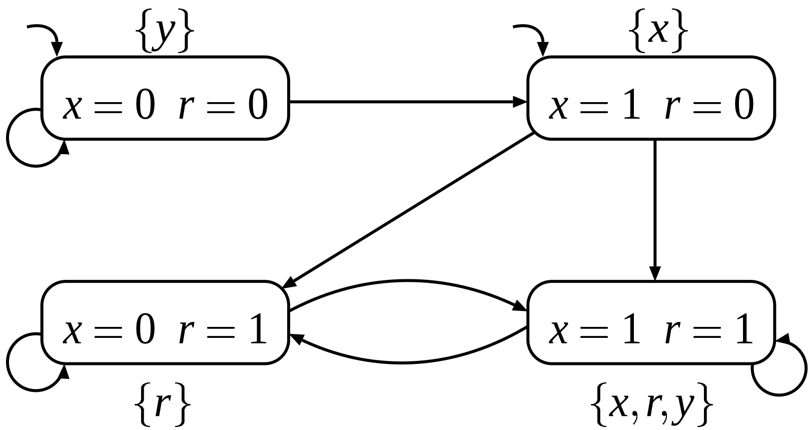
State $s \in S$ is called *reachable* in *TS* if there exists an initial, finite execution fragment

$$s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} s_n = s\ .$$

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

Modeling sequential circuits





Transition system representation of a simple hardware circuit

Input variable x, output variable y, and register r

Output function $\neg(x \oplus r)$ and register evaluation function $x \vee r$

Atomic propositions

Consider two possible state-labelings:

- Let $AP = \{ x, y, r \}$
  - $L(\langle x = 0, r = 1 \rangle) = \{ r \}$ and $L(\langle x = 1, r = 1 \rangle) = \{ x, r, y \}$
  - $L(\langle x = 0, r = 0 \rangle) = \{ y \}$ and $L(\langle x = 1, r = 0 \rangle) = \{ x \}$
  - property e.g., "once the register is one, it remains one"

- Let $AP' = \{ x, y \}$ – the register evaluations are now "invisible"
  - $L(\langle x = 0, r = 1 \rangle) = \varnothing$ and $L(\langle x = 1, r = 1 \rangle) = \{ x, y \}$
  - $L(\langle x = 0, r = 0 \rangle) = \{ y \}$ and $L(\langle x = 1, r = 0 \rangle) = \{ x \}$
  - property e.g., "the output bit y is set infinitely often"

Beverage vending machine revisited

"Abstract" transitions:

$$start \xrightarrow{true:coin} select \quad \text{and} \quad start \xrightarrow{true:refill} start$$

$$select \xrightarrow{nsprite>0:sget} start \quad \text{and} \quad select \xrightarrow{nbeer>0:bget} start$$

$$select \xrightarrow{nsprite=0 \wedge nbeer=0:ret\_coin} start$$

Action	Effect on variables
$coin$	
ret_coin	
$sget$	$nsprite := nsprite - 1$
$bget$	$nbeer := nbeer - 1$
$refill$	$nsprite := max;\ nbeer := max$

Program graph representation

Some preliminaries

- typed variables with a valuation that assigns values to variables
  - e.g., $\eta(x) = 17$ and $\eta(y) = -2$
- the set of Boolean conditions over *Var*
  - propositional logic formulas whose propositions are of the form "$\overline{x} \in \overline{D}$"
  - $(-3 < x \leqslant 5) \ \wedge \ (y = green) \ \wedge \ (x \leqslant 2 \cdot x')$
- effect of the actions is formalized by means of a mapping:

$$\mathit{Effect} : \mathit{Act} \times \mathit{Eval}(\mathit{Var}) \rightarrow \mathit{Eval}(\mathit{Var})$$

  - e.g., $\alpha \equiv x := y+5$ and evaluation $\eta(x) = 17$ and $\eta(y) = -2$
  - $\mathit{Effect}(\alpha, \eta)(x) \ = \ \eta(y) + 5 = 3,$ and $\mathit{Effect}(\alpha, \eta)(y) \ = \eta(y) = -2$

Program graphs

A *program graph* *PG* over set *Var* of typed variables is a tuple

$$(\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \longrightarrow, \mathit{Loc}_0, g_0) \quad \text{where}$$

- Loc is a set of *locations* with initial locations $\mathit{Loc}_0 \subseteq \mathit{Loc}$
- Act is a set of actions
- $\mathit{Effect} : \mathit{Act} \times \mathit{Eval}(\mathit{Var}) \to \mathit{Eval}(\mathit{Var})$ is the *effect* function
- $\longrightarrow \subseteq \mathit{Loc} \times (\underbrace{\mathit{Cond}(\mathit{Var})}_{\text{Boolean conditions over } \mathit{Var}} \times \mathit{Act}) \times \mathit{Loc}$, transition relation
- $g_0 \in \mathit{Cond}(\mathit{Var})$ is the initial *condition*.

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

Beverage vending machine

- $\mathit{Loc} = \{\, start, select \,\}$ with $\mathit{Loc}_0 = \{\, start \,\}$
- $\mathit{Act} = \{\, bget, sget, coin, ret\_coin, refill \,\}$
- $\mathit{Var} = \{\, nsprite, \;\; nbeer \,\}$ with domain $\{\, 0, 1, \ldots, max \,\}$
- $$\begin{array}{lcl} \mathit{Effect}(coin, \eta) & = & \eta \\ \mathit{Effect}(ret\_coin, \eta) & = & \eta \\ \mathit{Effect}(sget, \eta) & = & \eta[nsprite := nsprite{-}1] \\ \mathit{Effect}(bget, \eta) & = & \eta[nbeer := nbeer{-}1] \\ \mathit{Effect}(refill, \eta) & = & [nsprite := max, nbeer := max] \end{array}$$
- $g_0 = (nsprite = max \;\wedge\; nbeer = max)$

From program graphs to transition systems

- Basic strategy: *unfolding*
  - state = location (current control) ℓ + data valuation η
  - initial state = initial location satisfying the initial condition g_0

- Propositions and labeling
  - propositions: "at ℓ" and "$x \in D$" for $D \subseteq \mathit{dom}(x)$
  - $\langle \ell, \eta \rangle$ is labeled with "at ℓ" and all conditions that hold in η

- $\ell \xrightarrow{g:\alpha} \ell'$ and g holds in η then $\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \mathit{Effect}(\alpha, \eta) \rangle$

Structured operational semantics

- The notation $\dfrac{\text{premise}}{\text{conclusion}}$ means:

- If the proposition above the “solid line” (i.e., the premise) holds, then the proposition under the fraction bar (i.e., the conclusion) holds

- Such “if . . ., then . . .” propositions are also called *inference rules*

- If the premise is a tautology, it may be omitted (as well as the “solid line”)

- In the latter case, the rule is also called an *axiom*

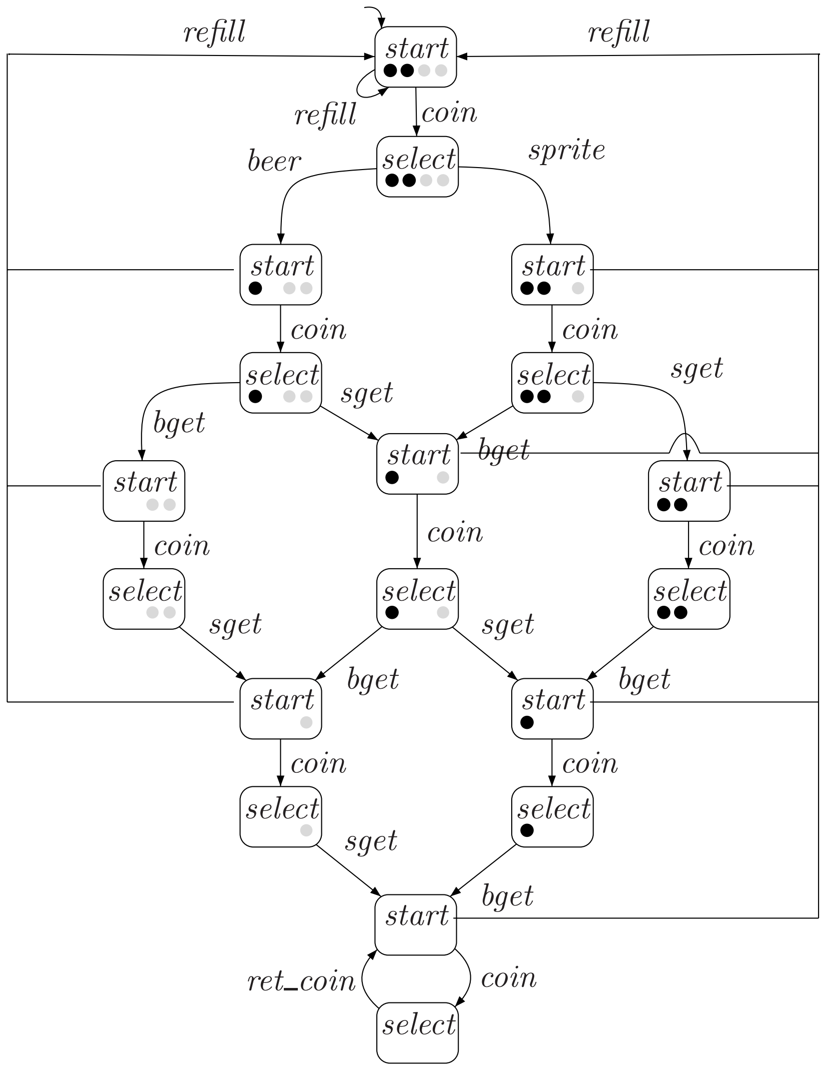
Transition systems for program graphs

The transition system $\mathit{TS}(\mathit{PG})$ of program graph

$$\mathit{PG} = (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \longrightarrow, \mathit{Loc}_0, g_0)$$

over set Var of variables is the tuple $(S, \mathit{Act}, \longrightarrow, I, \mathit{AP}, L)$ where

- $S = \mathit{Loc} \times \mathit{Eval}(\mathit{Var})$
- $\longrightarrow \subseteq S \times Act \times S$ is defined by the rule: $\dfrac{\ell \xrightarrow{g:\alpha} \ell' \quad \wedge \quad \eta \models g}{\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \mathit{Effect}(\alpha, \eta) \rangle}$
- $I = \{ \langle \ell, \eta \rangle \mid \ell \in \mathit{Loc}_0, \eta \models g_0 \}$
- $\mathit{AP} = \mathit{Loc} \cup \mathit{Cond}(\mathit{Var})$ and $L(\langle \ell, \eta \rangle) = \{\ell\} \cup \{ g \in \mathit{Cond}(\mathit{Var}) \mid \eta \models g \}$.


refill
refill
start
refill
coin
select
beer
sprite
start
start
coin
coin
select
select
sget
sget
bget
start
bget
start
start
coin
coin
coin
select
select
select
sget
sget
bget
bget
start
start
coin
coin
select
select
sget
bget
start
ret_coin
coin
select

Transition systems $\neq$ finite automata

As opposed to finite automata, in a transition system:

- there are *no* accept states
- set of states and actions may be countably infinite
- may have infinite branching
- actions may be subject to synchronization (cf. next lecture)
- nondeterminism has a different role

Transition systems are appropriate for reactive system behaviour