Introduction to Model Checking

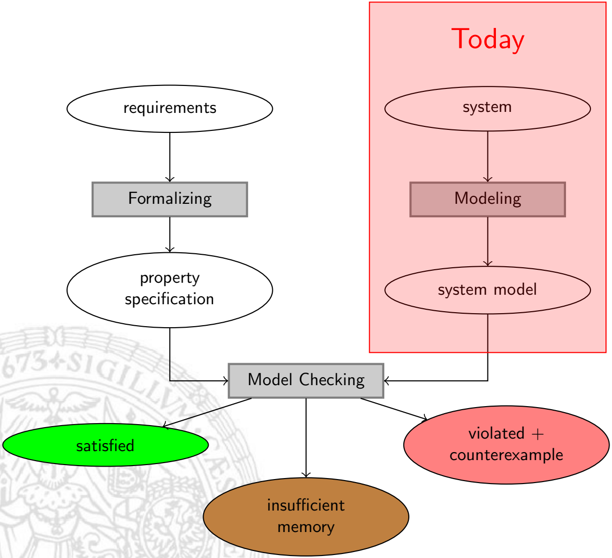
René Thiemann

Institute of Computer Science

University of Innsbruck

WS 2007/2008

Model checking overview



Transition systems

- model to describe the behaviour of systems
- digraphs where nodes represent states, and edges model transitions
- state:
  - the current phase of a traffic light
  - the current values of all program variables + the program counter
- transition: (“state change”)
  - a switch from one phase to the next one
  - the execution of a program statement

Transition system

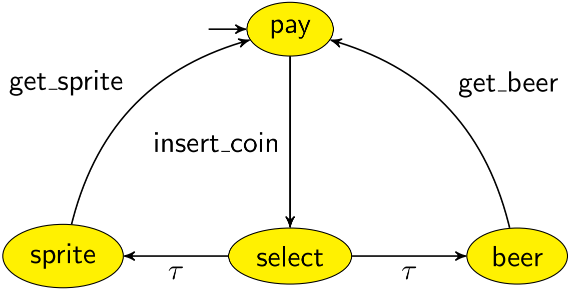
A *transition system* TS is a tuple $(S, Act, \rightarrow, I, AP, L)$ where

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

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

- An execution ϱ of TS is an alternating sequence of states and actions

$$\varrho \;=\; s_0\,\alpha_1\,s_1\,\alpha_2\,\ldots\alpha_n\,s_n\,\ldots$$

  such that
  - $s_i \xrightarrow{\alpha_{i+1}} s_{i+1}$ for all $0 \leqslant i \in \mathbb{N}$
  - $s_0 \in I$

  (W.l.o.g. consider only infinite executions)
- A trace of an execution is an infinite sequence of atomic propositions, i.e., $trace(\varrho) \in (2^{AP})^\omega$

$$trace(\varrho) = L(s_0)\;L(s_1)\;L(s_2)\;L(s_3)\;\ldots$$

- $Traces(TS)$ is the set of all traces of all executions of TS
  It defines the observable behaviour of TS.

Example

Beverage vending machine revisited

"Abstract" transitions:

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

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

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

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

Program graph representation

Some preliminaries

- typed variables with a valuation that assigns values to variables
  - e.g., $\eta(x) = 17$ and $\eta(y) = green$
- the set of Boolean conditions over Var
  - propositional logic formulas whose propositions are of the form "$\overline{x} \in \overline{D}$"
  - $(nsprite \geqslant 1) \;\wedge\; (y = blue) \;\wedge\; (x \leqslant 2{\cdot}x')$
- effect of the actions is formalized by means of a mapping:

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

  - e.g., for action α use update $x := y == blue \; ? \; 2 \cdot x : x - 1$, and evaluation η is given by $\eta(x) = 17$ and $\eta(y) = red$
  - $Effect(\alpha, \eta)(x) \;=\; \eta(x) - 1 = 16,$ and $Effect(\alpha, \eta)(y) \;= \eta(y) = red$

Program graphs

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

$$(Loc, Act, Effect, \longrightarrow, Loc_0, g_0) \quad \text{where}$$

- *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
- $\longrightarrow \subseteq Loc \times (\underbrace{Cond(Var)}_{\text{Boolean conditions over } Var} \times Act) \times Loc$, transition relation
- $g_0 \in Cond(Var)$ is the initial *condition*.

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

Beverage vending machine

- $Loc = \{ start, select \}$ with $Loc_0 = \{ start \}$
- $Act = \{ bget, sget, coin, ret\_coin, refill \}$
- $Var = \{ nsprite, \quad nbeer \}$ with domain $\{ 0, 1, \ldots, max \}$
- $$\begin{aligned} Effect(coin, \eta) &= \eta \\ Effect(ret\_coin, \eta) &= \eta \\ Effect(sget, \eta) &= \eta[nsprite := nsprite - 1] \\ Effect(bget, \eta) &= \eta[nbeer := nbeer - 1] \\ Effect(refill, \eta) &= [nsprite := max, nbeer := max] \end{aligned}$$
- $g_0 = (nsprite = max \wedge nbeer = max)$

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 $D \subseteq dom(x)$
  - $\langle \ell, \eta \rangle$ 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*

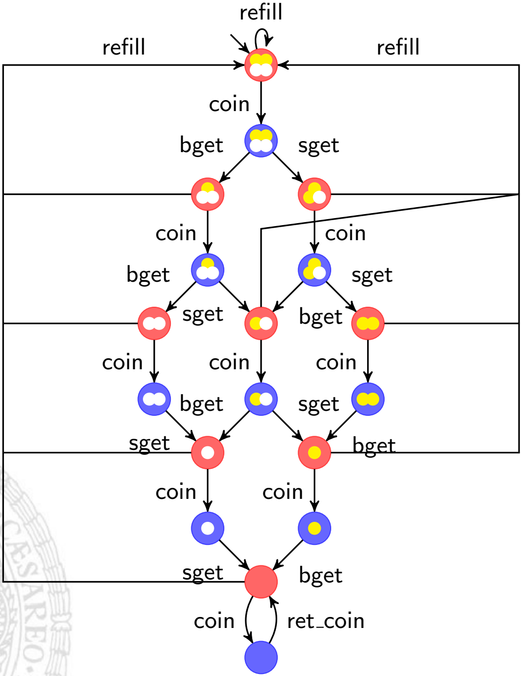
Transition systems for program graphs

The transition system $TS(PG)$ 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 the rule: $\dfrac{\ell \xrightarrow{g:\alpha} \ell' \quad \wedge \quad \eta \models g}{\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', Effect(\alpha, \eta) \rangle}$
- $I = \{ \langle \ell, \eta \rangle \mid \ell \in Loc_0, \eta \models g_0 \}$
- $AP = Loc \cup Cond(Var)$ and
  $L(\langle \ell, \eta \rangle) = \{\ell\} \cup \{ g \in Cond(Var) \mid \eta \models g \}$.


refill
refill
refill
coin
bget
sget
coin
coin
bget
sget
sget
bget
coin
coin
coin
bget
sget
sget
bget
coin
coin
sget
bget
coin
ret_coin

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)
- non-determinism has a different role

Transition systems are appropriate for reactive system behaviour

Exercise

- Modify the program graph of the vending machine such that the user can select the beverages. Additional actions select_beer and select_sprite may be helpful.
- Construct the corresponding transition system for $max = 2$.
- Does your system satisfy the following property?

Whenever the user infinitely often selects beer,
then she gets beer infinitely often.