Formal Specification and Verification

Formal specification (2)

29.11.2016

Viorica Sofronie-Stokkermans

e-mail: sofronie@uni-koblenz.de

Until now

- Logic
- Formal specification (generalities)

  Algebraic specification

Formal specification

- **Specification languages for describing programs/processes/systems**

  Model based specification

  transition systems, abstract state machines, specifications based on set theory

  Axiom-based specification

  algebraic specification

  Declarative specifications

  logic based languages (Prolog)

  functional languages, λ-calculus (Scheme, Haskell, OCaml, ...)

  rewriting systems (very close to algebraic specification): ELAN, SPIKE, ...

- **Specification languages for properties of programs/processes/systems**

  Temporal logic

Formal specification

- **Specification languages for describing programs/processes/systems**

  Model based specification

    transition systems, abstract state machines, specifications based on set theory

  Axiom-based specification

    algebraic specification

  Declarative specifications

    logic based languages (Prolog)

    functional languages, λ-calculus (Scheme, Haskell, OCaml)

    rewriting systems (very close to algebraic specification): ELAN, SPIKE

- **Specification languages for properties of programs/processes/systems**

  Temporal logic

Algebraic Specification

"A gentle introduction to CASL"

M. Bidoit and P. Mosses

`http://www.lsv.ens-cachan.fr/~bidoit/GENTLE.pdf`

Formal specification

- **Specification languages for describing programs/processes/systems**

  Model based specification

  transition systems, abstract state machines, specifications based on set theory

  Axiom-based specification

  algebraic specification

  Declarative specifications

  logic based languages (Prolog)

  functional languages, λ-calculus (Scheme, Haskell, OCaml)

  rewriting systems (very close to algebraic specification): ELAN, SPIKE

- **Specification languages for properties of programs/processes/systems**

  Temporal logic

Transition systems

Transition systems

- Executions
- Modeling data-dependent systems

Transition systems

- Model to describe the behaviour of systems
- Digraphs where nodes represent states, and edges model transitions
- **State:** Examples
  - 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"): Examples
  - 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 systems

Definition.

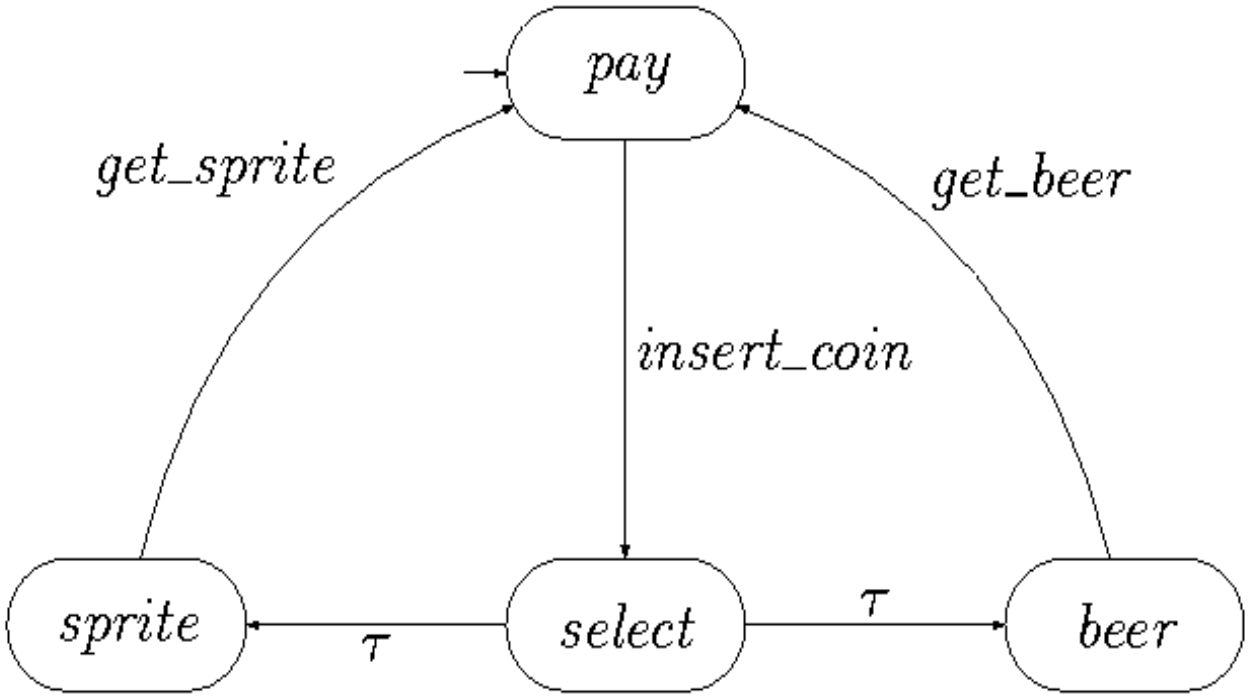
A transition system TS is a tuple $(S, Act, \rightarrow, I, AP, L)$ where:

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

S and Act are either finite or countably infinite

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

A beverage vending machine



states? actions?, transitions?, initial states?

Direct successors and predecessors

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

$Pre(s, \alpha) = \{s' \in S \mid s' \xrightarrow{\alpha} s\}$, $\quad Pre(s) = \bigcup_{\alpha \in Act} Pre(s, \alpha)$

$Post(C, \alpha) = \bigcup_{s \in C} Post(s, \alpha)$,

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

$Pre(C, \alpha) = \bigcup_{s \in C} Pre(s, \alpha)$,

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

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

Action- and AP-determinism

Definition. Transition system $TS = (S, Act, \rightarrow, I, AP, L)$ is action-deterministic iff:

$$| I | \leq 1 \text{ and } | Post(s, \alpha) | \leq 1 \text{ for all } s \in S, \alpha \in Act$$

(at most one initial state and for every action, a state has at most one successor)

Definition. Transition system $TS = (S, Act, \rightarrow, I, AP, L)$ is AP-deterministic iff:

$$| I | \leq 1 \text{ and } | Post(s) \cap \{s' \in S \mid L(s') = A\} | \leq 1 \text{ for all } s \in S, A \in 2^{AP}$$

(at most one initial state; for state and every $A : AP \rightarrow \{0, 1\}$ there exists at most a successor of s in which "satisfies A")

Non-determinism

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

Non-determinism

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!

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
- nondeterminism has a different role

Transition systems are appropriate for modelling reactive system behaviour

Executions

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

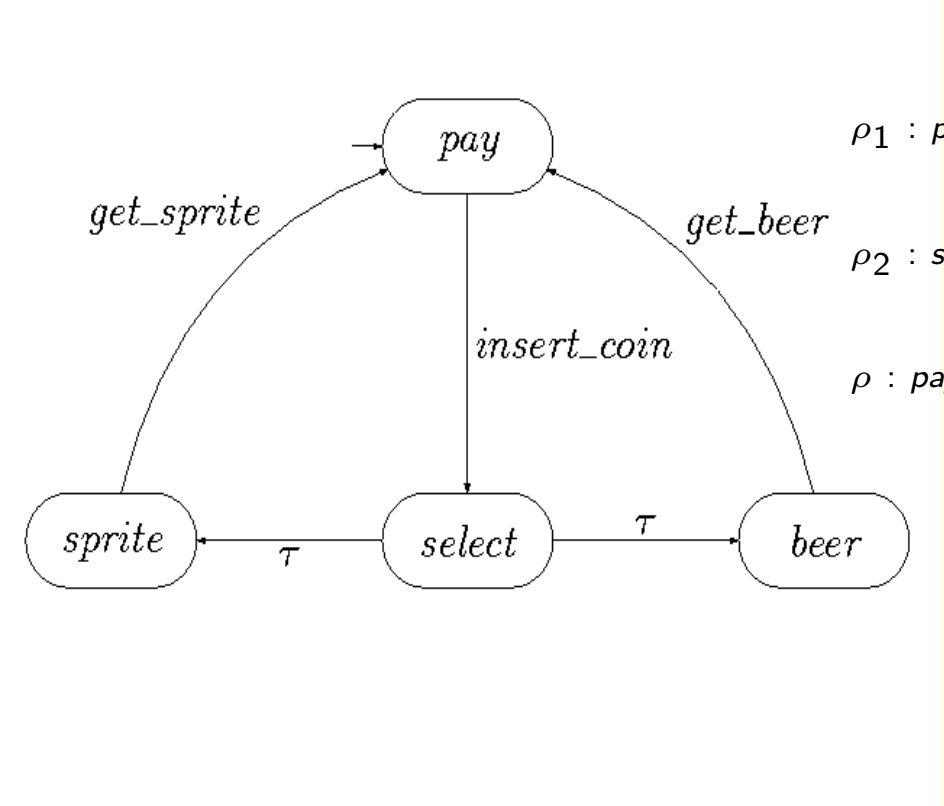
  $\rho = s_0\alpha_1 s_1\alpha_2 \ldots \alpha_n s_n$ such that $s_i \xrightarrow{\alpha_{i+1}} s_{i+1}$ for all $0 \leq i < n$.

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

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

- 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 $s_0 \in I$

Examples of Executions

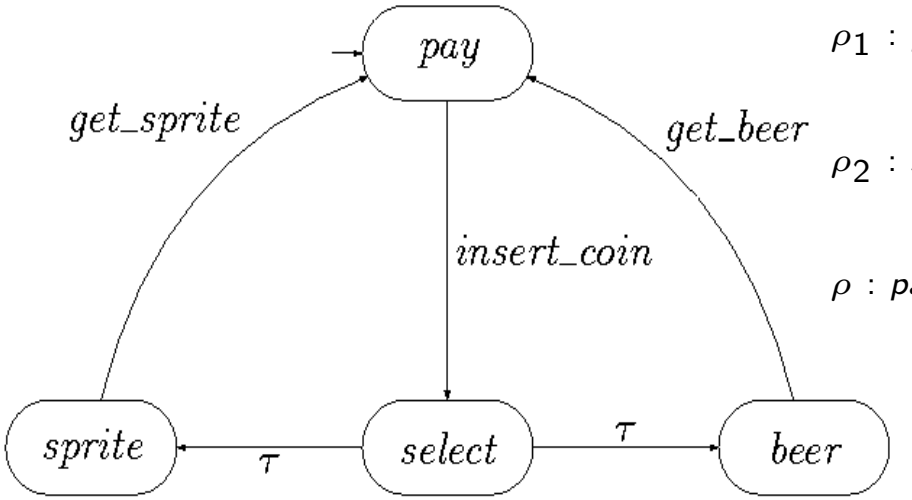


$\rho_1 : pay \xrightarrow{coin} select \xrightarrow{\tau} sprite \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} sprite \xrightarrow{sget} \ldots$

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

$\rho : pay \xrightarrow{coin} select \xrightarrow{\tau} sprite \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} sprite$

Examples of Executions



ρ_1 : $pay \xrightarrow{coin} select \xrightarrow{\tau} sprite \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} sprite \xrightarrow{sget} \ldots$

ρ_2 : $select \xrightarrow{\tau} sprite \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} beer \xrightarrow{bget} \ldots$

ρ : $pay \xrightarrow{coin} select \xrightarrow{\tau} sprite \xrightarrow{sget} pay \xrightarrow{coin} select \xrightarrow{\tau} sprite$

- 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

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

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

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