Labeled Transition Systems
Lecture #1 of Probabilistic Models for Concurrency

Joost-Pieter Katoen
Lehrstuhl II: Programmiersprachen u. Softwarevalidierung

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

March 12, 2005
Theme of the course

The theory of modelling and analysis of concurrent probabilistic systems
Course topics

- **Part I: Process algebra**
  - Transition systems
  - Behavioural equivalences and operational semantics

- **Part II: Probabilistic process algebra**
  - Markov chains and decision processes
  - Behavioural equivalences and operational semantics

- **Part III: General probabilistic process algebra**
  - (Stochastic) timed automata
  - Behavioural equivalences and operational semantics

- **Part IV: Case studies**
Overview Lecture #1

⇒ Transition systems
  – Trace behaviour
  – Modeling data-dependent systems

• Organisational matters
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 quadruple $(S, Act, \rightarrow, I)$ 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.

$S$ and $Act$ are either finite or countably infinite

Notation: $s \xrightarrow{\alpha} s'$ instead of $(s, \alpha, s') \in \rightarrow$
A beverage vending machine

- `sprite` to `select`
- `select` to `beer`
- `pay` to `get_sprite`
- `pay` to `get_beer`
- `insert_coin` to `select`
Direct successors and predecessors

\[\text{Post}(s, \alpha) = \left\{ s' \in S \mid s \xrightarrow{\alpha} s' \right\}, \quad \text{Post}(s) = \bigcup_{\alpha \in \text{Act}} \text{Post}(s, \alpha)\]

\[\text{Pre}(s, \alpha) = \left\{ s' \in S \mid s' \xrightarrow{\alpha} s \right\}, \quad \text{Pre}(s) = \bigcup_{\alpha \in \text{Act}} \text{Pre}(s, \alpha)\]

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

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

State \(s\) is called \textit{terminal} if and only if \(\text{Post}(s) = \emptyset\)
Deterministic transition systems

Transition system $TS = (S, Act, \rightarrow, L)$ is deterministic if and only if:

\[ |I| < 2 \quad \text{and} \quad |Post(s, \alpha)| < 2 \quad \text{for all} \quad s, \alpha \]

Otherwise, $TS$ is called non-deterministic.
Executions

• A **finite execution fragment** $\varrho$ of $TS$ is an alternating sequence of states and actions ending with a state:

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

such that $s_i \xrightarrow{\alpha_{i+1}} s_{i+1}$ for all $0 \leq i < n$.

• An **infinite execution fragment** $\rho$ 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 $s_0 \in I$
Traces

- The finite word $\alpha_1 \alpha_2 \ldots \alpha_n \in \text{Act}^*$ is a finite trace of TS whenever there is a finite execution fragment of $TS$

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

  – notation: $\alpha_1 \alpha_2 \ldots \alpha_n = \text{trace}(\varrho)$.

- The infinite word $\alpha_1 \alpha_2 \ldots \in \text{Act}^\omega$ is an (infinite) trace whenever there is an infinite execution fragment of $TS$

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

  – notation: $\alpha_1 \alpha_2 \alpha_3 \ldots = \text{trace}(\rho)$. 
Finite and infinite traces

- The set of traces of a set of executions is defined in the usual way:

  \[ \text{trace}(\{ \rho_1, \ldots, \rho_n \}) = \{ \text{trace}(\rho_1), \ldots, \text{trace}(\rho_n) \} \]

- \( \text{Traces}(s) = \text{trace}(\text{Execs}(s)) \) and \( \text{Traces}(TS) = \bigcup_{s \in I} \text{Traces}(s) \)

- \( \text{Traces}_{\text{fin}}(s) = \text{trace}(\text{Execs}_{\text{fin}}(s)) \) and \( \text{Traces}_{\text{fin}}(TS) = \bigcup_{s \in I} \text{Traces}_{\text{fin}}(s) \)

  - a trace of state \( s \) is the trace of an infinite execution fragment starting in \( s \)
  - a finite trace of \( s \) is the trace of a finite execution fragment that starts in \( s \)
Beverage vending machine revisited

“Abstract” transitions:

\[
\begin{align*}
\text{start} & \xrightarrow{\text{true:coin}} \text{select} & \quad \text{and} & \quad \text{start} \xrightarrow{\text{true:refill}} \text{start} \\
\text{select} & \xrightarrow{\text{nsprite} > 0: \text{sget}} \text{start} & \quad \text{and} & \quad \text{select} \xrightarrow{\text{nbeer} > 0: \text{bget}} \text{start} \\
\text{select} & \xrightarrow{\text{nsprite} = 0 \land \text{nbeer} = 0: \text{ret}_\text{coin}} \text{start}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Action</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>coin</td>
<td>(\text{nsprite} := \text{nsprite} - 1)</td>
</tr>
<tr>
<td>ret_coin</td>
<td>(\text{nsprite} := \text{nsprite} - 1)</td>
</tr>
<tr>
<td>sget</td>
<td>(\text{nbeer} := \text{nbeer} - 1)</td>
</tr>
<tr>
<td>bget</td>
<td>(\text{nbeer} := \text{nbeer} - 1)</td>
</tr>
<tr>
<td>refill</td>
<td>(\text{nsprite} := \text{max}; \text{nbeer} := \text{max})</td>
</tr>
</tbody>
</table>
Transition system representation
Program graphs

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

Notation: $\ell \xrightarrow{g, \alpha} \ell'$ denotes $(\ell, g, \alpha, \ell') \in \rightarrow$
Beverage vending machine

- \( \text{Loc} = \{ \text{start}, \text{select} \} \) with \( \text{Loc}_0 = \{ \text{start} \} \)

- \( \text{Act} = \{ \text{bget}, \text{sget}, \text{coin}, \text{ret\_coin}, \text{refill} \} \)

- \( \text{Var} = \{ \text{nsprite}, \text{nbeer} \} \) with domain \( \{ 0, 1, \ldots, \text{max} \} \)

\begin{align*}
\text{Effect}(\text{coin, } \eta) & = \eta \\
\text{Effect}(\text{ret\_coin, } \eta) & = \eta \\
\text{Effect}(\text{sget, } \eta) & = \eta[\text{nsprite} := \text{nsprite} - 1] \\
\text{Effect}(\text{bget, } \eta) & = \eta[\text{nbeer} := \text{nbeer} - 1] \\
\text{Effect}(\text{refill, } \eta) & = [\text{nsprite} := \text{max}, \text{nbeer} := \text{max}] \\
\end{align*}

- \( g_0 = (\text{nsprite} = \text{max} \land \text{nbeer} = \text{max}) \)
Transition systems for program graphs

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

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

over set $Var$ of variables is the tuple $(S, Act, \rightarrow, I)$ where

- $S = Loc \times Eval(Var)$
- $\rightarrow \subseteq S \times Act \times S$ is defined by the following rule:

$$\ell \xrightarrow{\text{g:}\alpha} \ell' \land \eta \models g$$

$$\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \text{Effect}(\alpha, \eta) \rangle$$

- $I = \{ \langle \ell, \eta \rangle \mid \ell \in Loc_0, \eta \models g_0 \}$. 
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)

*Transition systems are appropriate for reactive system behaviour*
Reactive behaviour

- Buttons can react either
  - by sometimes going down when pushed, or
  - by sometimes being blocked when pushed

- Interaction = trying to push buttons in some order

- (Identity of) states are unobservable

- Can only observe the sequence of actions that is performed

  *this is called the black-box view*
Overview Lecture #1

- Transition systems
  - Trace behaviour
  - Modeling data-dependent systems

⇒ Organisational matters
Course organization

- **Lectures**: twice per week (check course web-page!)

- **Exercises**: every other week
  - marked exercises (50% of points needed + one example on board)

- **Course material**:
  - slides of the lectures
  - literature indicated on web-page (in library)
  - there are no lecture notes

- **Exam**: written exam on *July 22, 2005*
  - 2 page summary at June 1 + 2 page summary July 15
  - own summaries are only allowed material at exam
Preliminary planning

• **Lectures**: Tue AH II + Wed AH VI
  - April 12, 13, 20, 26, 27
  - May 4, 10, 11, 24, 25, 31
  - June 1, 7, 21, 22
  - July 5, 6, 12, 13, 20

• **Exercises**:
  - April 13, 27
  - May 11, 25
  - June 1, 15, 22
  - July 6, 20