Theoretical Foundations of the UML

Lecture 17: Statecharts Semantics (2)

Joost-Pieter Katoen

Lehrstuhl für Informatik 2
Software Modeling and Verification Group

http://moves.rwth-aachen.de/i2/uml09100/

20. Januar 2013

Statecharts

Definition (Statecharts)

A statechart SC is a triple $(N, E, Edges)$ with:

1. N is a set of nodes (or: states) structured in a tree
2. E is a set of events
   - pseudo-event $after(d) \in E$ denotes a delay of $d \in \mathbb{R}_{\geqslant 0}$ time units
   - $\perp \notin E$ stands for "no event available"
3. $Edges$ is a set of (hyper-) edges, defined later on.

Definition (System)

A system is described by a finite collection of statecharts $(SC_1, \ldots, SC_k)$.

What does a single StateChart mean?

- The semantics is given as a Mealy machine:
- State = a set of nodes ("current control") + the values of variables
- Edge is enabled if all events are present and guard holds in current state
- Executing edge $X \xrightarrow{e[g]/A} Y$ = perform actions A, consume event e
  - leave source nodes X and switch to target nodes Y

  $\Rightarrow$ events are unordered, and considered as a set
- Principle: execute as many non-conflicting edges at once

  $\Rightarrow$ the execution of such maximal set is a macro step

States and configurations

Definition (Configuration)

A configuration of $SC = (N, E, Edges)$ is a set $C \subseteq N$ of nodes satisfying:

- root $\in C$
- $x \in C$ and $type(x) = \text{OR}$ implies $|children(x) \cap C| = 1$
- $x \in C$ and $type(x) = \text{AND}$ implies $children(x) \subseteq C$

Let $Conf$ denote the set of configurations of SC.

Definition (State)

State of $SC = (N, E, Edges)$ is a triple (C, I, V) where

- C is a configuration of SC
- $I \subseteq V$ is a set of events ready to be processed
- V is a valuation of the variables.

Enabling of an edge

Definition (Enabledness)

Edge $X \xrightarrow{e[g]/A} Y$ is enabled in state (C, I, V) whenever:

- $X \subseteq C$, i.e. all source nodes are in configuration C
- $(\underbrace{(C_1, \ldots, C_n)}_{\text{configurations}}, \underbrace{(V_1, \ldots, V_n)}_{\text{variable valuations}}) \models g$, i.e., guard g is satisfied
- either $e \neq \bot$ implies $e \in I$, or $e = \bot$

Let $En(C, I, V)$ denote the set of enabled edges in state (C, I, V).

RWTH AACHEN UNIVERSITY

Macro steps

- On receiving an input e, several edges in SC may become enabled
- Then, a maximal and consistent set of enabled edges is taken
- If there are several such sets, choose one nondeterministically
- Edges in concurrent components can be taken simultaneously
- But edges in other components cannot; they are inconsistent
- To resolve nondeterminism (partly), priorities are used

Least common ancestor

Definition (Least common ancestor)

For $X \subseteq N$, the least common ancestor, denoted $lca(X)$, is the node $y \in N$ such that:

$$(\forall x \in X. \, x \trianglelefteq y) \quad \text{and} \quad \forall z \in N. \, (\forall x \in X. \, x \trianglelefteq z) \text{ implies } y \trianglelefteq z.$$

Intuition

Node y is an ancestor of any node in X (first clause), and is a descendant of any node which is an ancestor of any node in X (second clause).

Orthogonality of nodes

Definition (Orthogonality of nodes)

Nodes $x, y \in N$ are orthogonal, denoted $x \perp y$, if

$$\neg(x \trianglelefteq y) \quad \text{and} \quad \neg(y \trianglelefteq x) \quad \text{and} \quad \mathit{type}(\mathit{lca}(\{\, x, y \,\})) = \text{AND}.$$

Orthogonality captures the notion of independence. Orthogonal nodes can execute enabled edges independently, and thus concurrently.

Scope

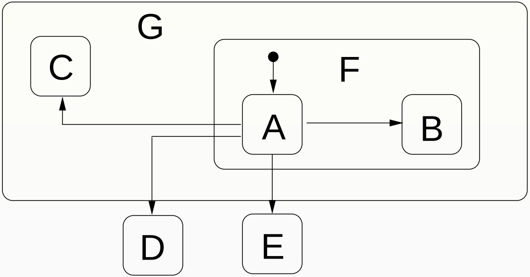
Definition (Scope of edge)

The scope of edge $X \overset{\cdots}{\longrightarrow} Y$ is the most nested OR-node that is an ancestor of both X and Y.

Intuition

The scope of edge $X \overset{\cdots}{\longrightarrow} Y$ is the most nested OR-node that is unaffected by executing the edge $X \overset{\cdots}{\longrightarrow} Y$.

Scope: example



$$scope(A \to D) = \text{root} \quad \text{and} \quad scope(A \to C) = G \quad \text{and} \quad scope(A \to B) = F$$

Consistency: formal definition

Definition (Consistency)

1. Edges $ed, ed' \in Edges$ are consistent if:

$$ed = ed' \quad \text{or} \quad scope(ed) \perp scope(ed').$$

2. $T \subseteq Edges$ is consistent if all edges in T are pairwise consistent. $Cons(T)$ is the set of edges that are consistent with all edges in $T \subseteq Edges$

$$Cons(T) = \{ed \in Edges \mid \forall ed' \in T : ed \text{ is consistent with } ed'\}$$

Example

On the black board.

What is now a macro step?

A macro step is a set T of edges such that:

- all edges in step T are enabled
- all edges in T are pairwise consistent, that is:
  - they are identical or
  - scopes are (descendants of) different children of the same AND-node
- enabled edge ed is not in step T implies

  there exists $ed' \in T$ such that ed is inconsistent with ed', and
  the priority of ed' is not smaller than ed
- step T is maximal (wrt. set inclusion)

Priorities

Priorities restrict (but do not abandon) nondeterminism between multiple enabled edges.

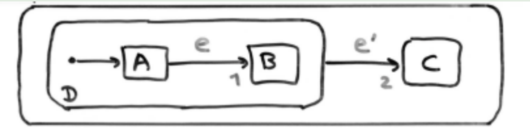
Definition (Priority relation)

The priority relation $\preceq \subseteq Edges \times Edges$ is a partial order defined for $ed, ed' \in Edges$ by:

$$ed \preceq ed' \quad \text{if} \quad scope(ed') \trianglelefteq scope(ed)$$

So, ed' has priority over ed if its scope is a descendant of ed's scope.

Example:



$2 \preceq 1$ since $scope(1) = D \trianglelefteq scope(2) = \text{root}$.

Priority: examples

Nondeterminism

Priorities rule out some nondeterminism, but not necessarily all.

What is now a macro step?

A macro step is a set T of edges such that:

- all edges in step T are enabled
- all edges in T are pairwise consistent
  - they are identical or
  - scopes are (descendants of) different children of the same AND-node
- step T is maximal (wrt. set inclusion)
  - T cannot be extended with any enabled, consistent edge
- priorities: enabled edge ed is not in step T implies

  $\exists ed' \in T.\ (ed \text{ is inconsistent with } ed' \wedge \neg(ed' \preceq ed))$

A macro step — formally

A macro step is a set T of edges such that:

- enabledness: $T \subseteq En(C, I, V)$
- consistency: $T \subseteq Cons(T)$
- maximality: $En(C, I, V) \cap Cons(T) \subseteq T$
- priority: $\forall ed \in En(C, I, V) - T$ we have
  $(\exists ed' \in T.\ (ed \text{ is inconsistent with } ed' \wedge \neg(ed' \preceq ed)))$

Note:

The first three points yield: $T = En(C, I, V) \cap Cons(T)$.

Computing the set T of macro steps in state (C, I, V)

function $nextStep(C, I, V)$

$T := \varnothing$

while $T \subset \ En(C, I, V) \cap \ Cons(T)$

do let $ed \in High\,((En(C, I, V) \cap \ Cons(T)) - T)\,;$

$\quad T := T \cup \{ed\}$

od

return T.

where $High(T) \ = \ \{ed \in T \mid \neg(\exists ed' \in T.\, ed \preceq ed')\}$

Correctness

Theorem:

For any state (C, I, V), $nextStep(C, I, V)$ is a macro step.

Proof.

The proof goes in two steps:

1. We prove enabledness, consistency, and maximality by applying some standard results from fixed point theory, in particular Tarski's-Kleene fixpoint theorem;
2. Then we consider priority and use some monotonicity argument.

□

Intermezzo on fixed point theory

Step execution

What happens in performing a step?

For a single statechart, executing a step results in performing the actions of all the edges in the step, and changing "control" to the target nodes of these edges.

Interference

Actions in statechart SC_j may influence the sets of events of other statecharts, e.g., SC_i with $i \neq j$ if action *send* $i.e$ is performed by SC_j in a step.

Thus:

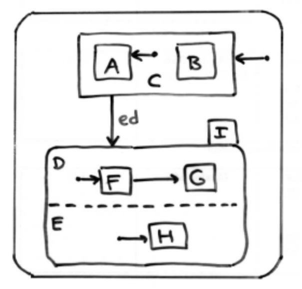
Execution of steps is considered on the system $(SC_1, \ldots, SC_n)$.

Default completion

Definition (Default completion)

The default completion C' of some set C of nodes is the canonical superset of C such that C' is a configuration. If C' contains an OR-node x and $children(x) \cap C = \varnothing$ implies $default(x) \in C'$.

Example:



1. Default completion of $C = \{\text{root}, I\}$ is $C' = C \cup \{D, E, F, H\}$
2. Default completion of $C = \{\text{root}, C\}$ is $C' = C \cup \{A\}$.

Step execution by a single statechart

- Let C_j be the current configuration of statechart SC_j
- Let $T_j \subseteq \mathit{Edges}_j$ be a step for SC_j
- The next state (C'_j, I'_j, V'_j) of statechart SC_j is given by:
  1. C'_j is the default completion of

$$\bigcup_{X \xrightarrow{e[g]/A} Y \in T_j} Y \;\cup\; \{x \in C_j \mid \forall X \to Y \in T_j.\, \neg(x \trianglelefteq \mathit{scope}(X \to Y))\}$$

  2. $I'_j = \bigcup_{k=1}^{n} \{e \mid \exists X \xrightarrow{e[g]/A} Y \in T_k.\, \mathit{send}\ j.e \in A\}$
  3. $V'_j(v) = \begin{cases} V_j(v) & \text{if } \forall X \xrightarrow{e[g]/A} Y \in T_j.\, v := \ldots \notin A \\ \mathit{val}(\text{expr}) & \text{if } \exists X \xrightarrow{e[g]/A} Y \in T_j.\, v := \text{expr} \in A \end{cases}$

Mealy machines [Mealy, 1953]

Definition (Mealy machine)

A Mealy machine $\mathcal{A} = (Q, q_0, \Sigma, \Gamma, \delta, \omega)$ with:

- Q is a finite set of states with initial state $q_0 \in Q$
- Σ is the input alphabet
- Γ is the output alphabet
- $\delta : Q \times \Sigma \to Q$ is the deterministic (input) transition function, and
- $\omega : Q \times \Sigma \to \Gamma$ is the output function

Intuition

A Mealy machine (or: finite-state transducer) is a finite-state automaton that produces output on a transition, based on current input and state.

Moore machines

In a Moore machine $\omega : Q \to \Gamma$, output is purely state-based.

From statecharts to a Mealy machine (1)

States

A state q is a tuple of the (local) states of SC_1 through SC_n.

Input and output events

Any input is a set of events, and any output is a set of events.

Next-state function δ

Defines the effect of executing a step.

Output function ω

Defines all events sent to some SC outside the system $(SC_1, \ldots, SC_n)$.

From statecharts to a Mealy machine (2)

States

A state q is a tuple of the (local) states of SC_1 through SC_k.

Formally:

- $Q = \prod_{k=1}^{n} (Conf_k \times 2^{E_k} \times Val_k)$ is the set of states
  - where $Conf_k$ is the set of configurations of SC_k,
  - E_k is the set of the events of SC_k,
  - and Val_k is the set of variable valuations of SC_k
- $q_0 = \prod_{k=1}^{n} (C_{0,k}, \varnothing, Val_{0,k})$ is the initial state
  - where $C_{0,k}$ is the default completion of the set $\{\text{root}\}$
  - the initial set of events is empty
  - $Val_{0,k}$ is the initial variable valuation of SC_k

From statecharts to a Mealy machine (3)

Input and output events

Any input is a set of events, and any output is a set of events.

Formally,

- Input alphabet: $\Sigma = 2^E - \{ \varnothing \}$
  - where $E = \bigcup_{k=1}^{n} E_k$ is the set of events in all statecharts

- Output alphabet: $\Gamma = 2^{E'}$
  - with $E' = \underbrace{\left\{ send\ j.e \in \bigcup\limits_{k=1}^{n} SC_k \mid j \notin \{1, \ldots, n\} \right\}}_{\text{all outputs that cannot be consumed}}$

From statecharts to a Mealy machine (4)

Next-state function δ

Defines the effect of executing a step.

Formally,

- $(s'_1, \ldots, s'_n) \in \delta((s_1, \ldots, s_n), E)$ where
  - $s''_i = (C'_i, I''_i, V'_i)$ is the next state after executing $T_i = \mathit{nextStep}(C_i, I_i, V_i)$
  - and $s'_i = (C'_i, I''_i \cup (E \cap E_i), V'_i)$

From statecharts to a Mealy machine (5)

Output function ω

Defines all events sent to some SC outside the system $(SC_1, \ldots, SC_n)$.

Formally,

- $\omega((s_1, \ldots, s_n), E) =$
  $\left\{ send\ j.e \mid j \notin \{1, \ldots, n\} \wedge \exists i.\, \exists X \xrightarrow{e[g]/send\ j.e} Y \in nextStep(C_i, I_i, V_i) \right\}$