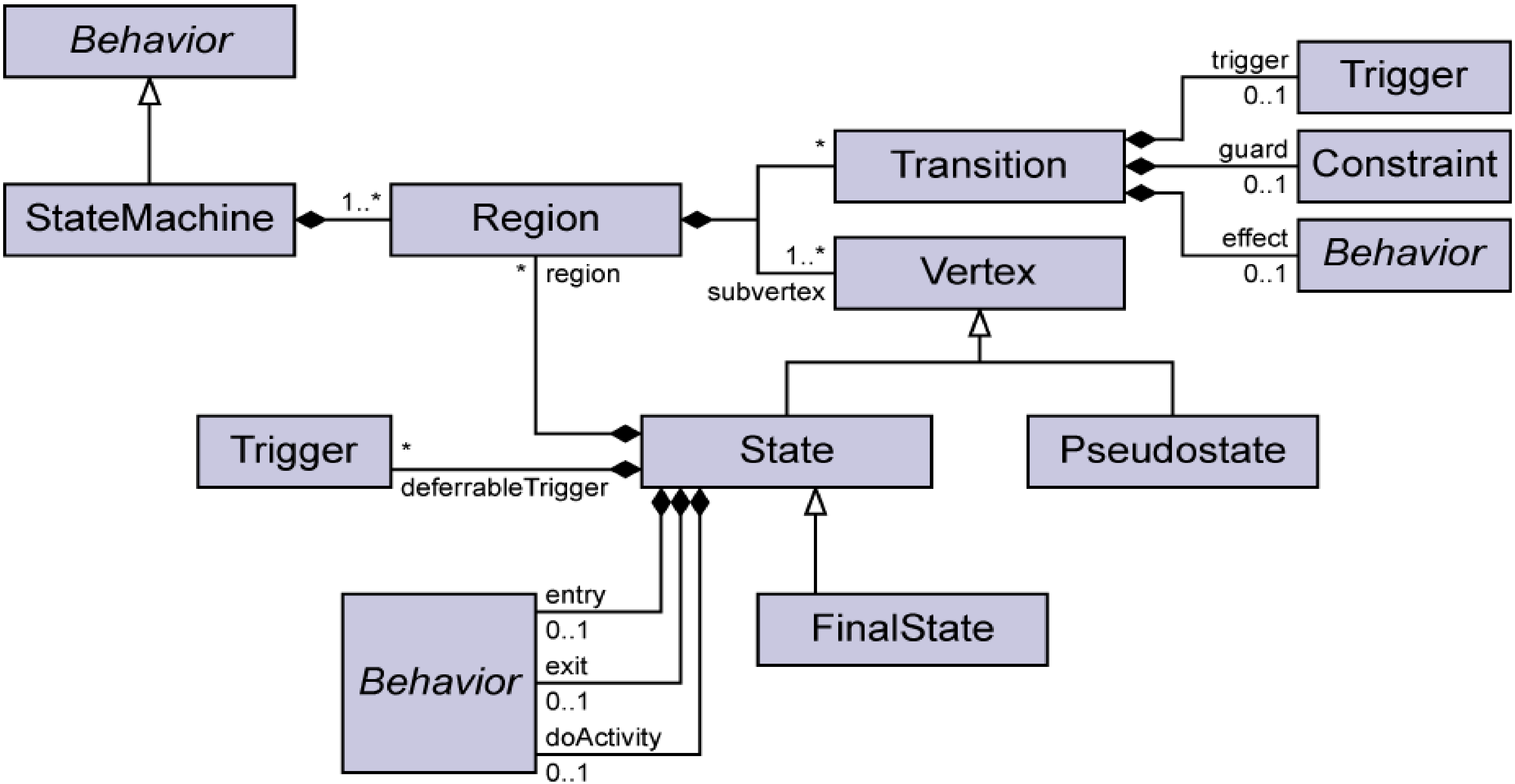
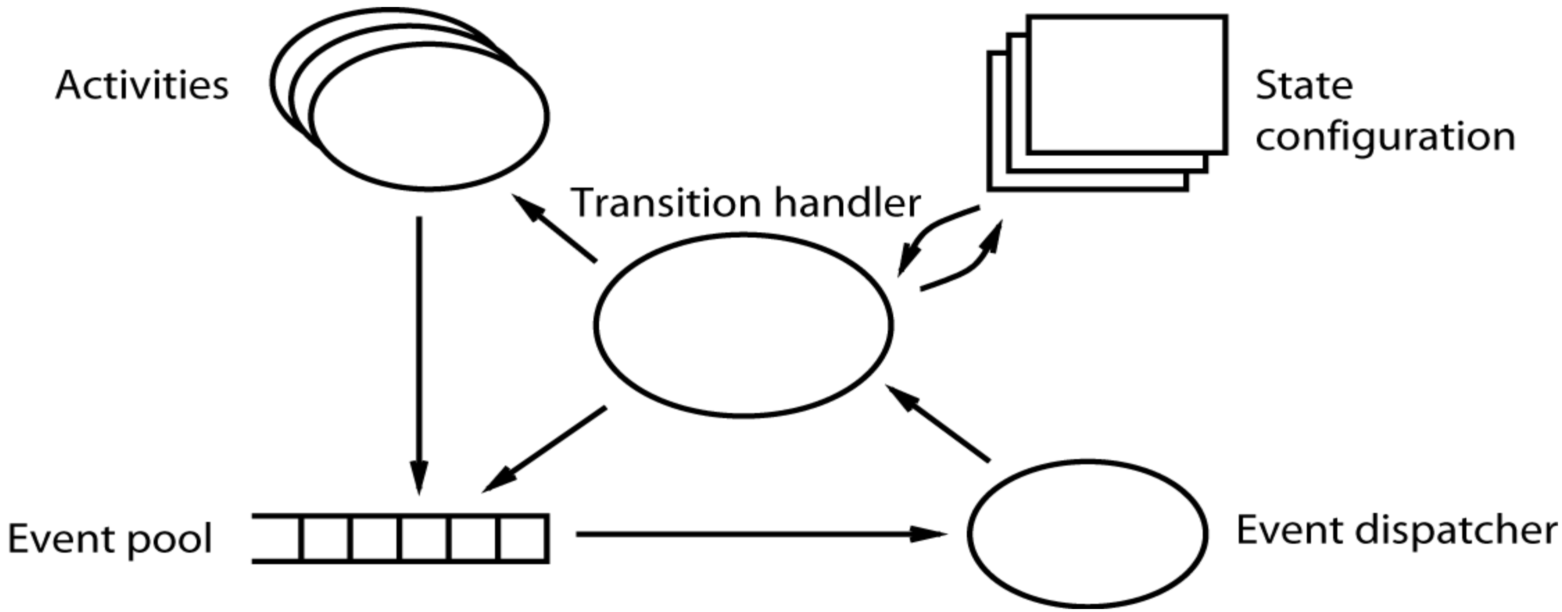
Metamodel



Behavior
StateMachine
1..*
Region
*
region
*
Transition
1..*
subvertex
Vertex
trigger
0..1
Trigger
guard
0..1
Constraint
effect
0..1
Behavior
Trigger
*
deferrableTrigger
State
Pseudostate
FinalState
Behavior
entry
0..1
exit
0..1
doActivity
0..1

Run-to-Completion Step: Overview

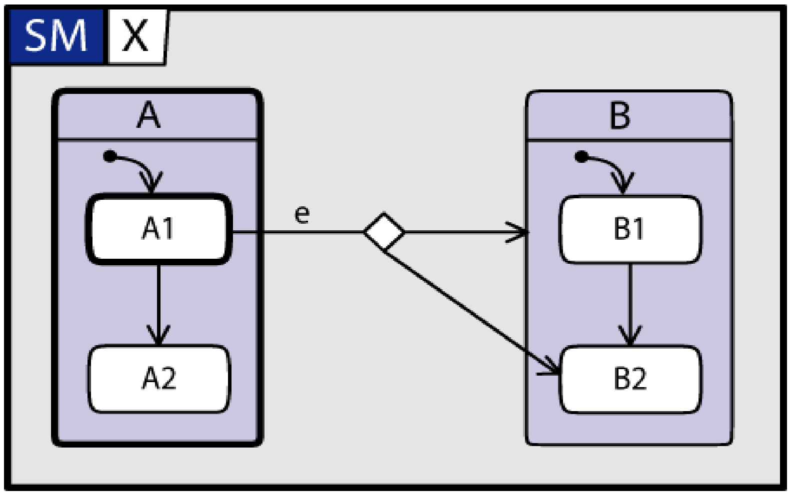


- Choose an **event** from the event pool (queue)
- Choose a **maximal**, **conflict-free**, **prioritized**, set of transitions enabled by the event
- Execute set of transitions
  - exit source states (inside-out)
  - execute transition effects
  - enter target states (outside-in)

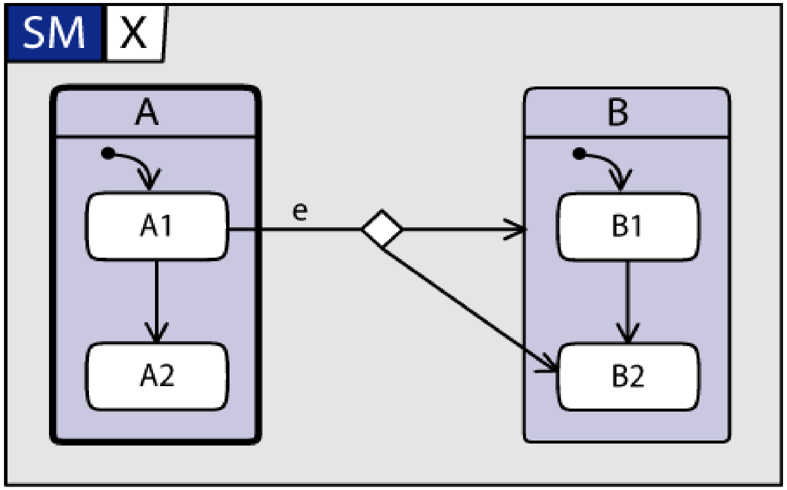
  thereby generating new events and activities

Run-to-Completion Step: Preliminaries (1)

- **Active state configuration**
  - the states the state machine currently is in
  - forms a tree
    - if a composite state is active, all its regions are active
- **Least-common-ancestor** (LCA) of states s_1 and s_2
  - the least region or orthogonal state (upwards) containing s_1 and s_2



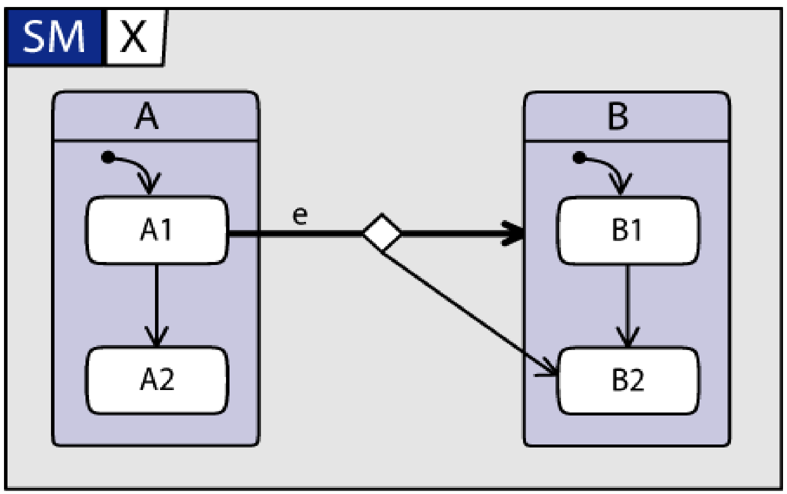
bold: active state configuration

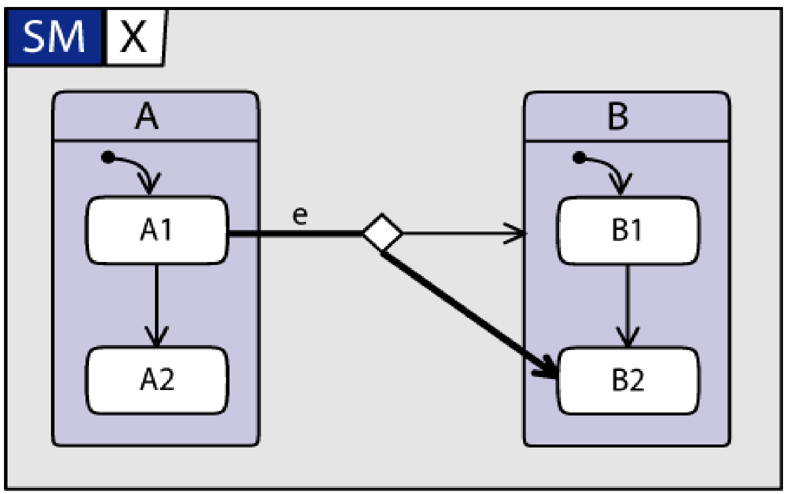


bold: LCA of states A1 and A2

Run-to-Completion Step: Preliminaries (2)

- **Compound** transitions
  - transitions for an event are “chained” into compound transitions
    - eliminating pseudostates like junction, fork, join, entry, exit
    - this is not possible for choice pseudostates where the guard of outgoing transitions are evaluated dynamically (in contrast to junctions)
  - several source and target states





Run-to-Completion Step: Preliminaries (3)

- **Main source** / **target state** m of compound transition t
  - Let s be LCA of all source and target states of t
  - If s region: m = direct subvertex of s containing all source states of t
  - If s orthogonal state: $m = s$
  - Similarly for main target state
  - All states between main source and explicit source states are exited, all state between main target and explicit target states are entered.

- **Conflict** of compound transitions t_1 and t_2
  - intersection of states exited by t_1 and t_2 not empty

- **Priority** of compound transition t_1 over t_2
  - s_i “deepest” source state of transition t_i
  - s_1 (direct or transitive) substate of s_2

Run-to-Completion Step (1)

RTC(*env*, *conf*) ≡
  ⌈*event* ← `fetch`()
    step ← choose steps(*conf*, *event*)
    if *step* = ∅ ∧ *event* ∈ *deferred*(*conf*)
    then `defer`(*event*)
    fi
    for *transition* ∈ *step* do
      conf ← handleTransition(*env*, *conf*, *transition*)
    od
    if *isCall* (*event*) ∧ *event* ∉ *deferred*(*conf*)
    then `acknowledge`(*event*)
    fi
    conf ⌋

Run-to-Completion Step (2)

steps(env, $conf$, $event$) $\equiv$

$\ulcorner$ $transitions \leftarrow enabled(env, conf, event)$

$\{step \mid (guard, step) \in$ steps$(conf, transitions) \wedge env \vDash guard\}$ $\lrcorner$

steps($conf$, $transitions$) $\equiv$

$\ulcorner$ $steps \leftarrow \{(\text{true}, \emptyset)\}$

for $transition \in transitions$ do

for $(guard, step) \in$ steps$(conf, transitions \setminus \{transition\})$ do

if $inConflict(conf, transition, step)$

then if $higherPriority(conf, transition, step)$

then $guard \leftarrow guard \wedge \neg guard(transition)$ fi

else $step \leftarrow step \cup \{transition\}$

$guard \leftarrow guard \wedge guard(transition)$ fi

$steps \leftarrow steps \cup \{(guard, step)\}$ od od

$steps$ $\lrcorner$

Run-to-Completion Step (3)

```
handleTransition(conf, transition) ≡
 ⌈for state ∈ insideOut(exited(transition)) do
      uncomplete(state)
      for timer ∈ timers(state) do stopTimer(timer) od
      execute(exit(state))
      conf ← conf \ {state}
  od
  execute(effect(transition))
  for state ∈ outsideIn(entered(transition)) do
      execute(entry(state))
      for timer ∈ timers(state) do startTimer(timer) od
      conf ← conf ∪ {state}
      complete(conf, state)
  od
  conf ⌋
```

Semantic variation points

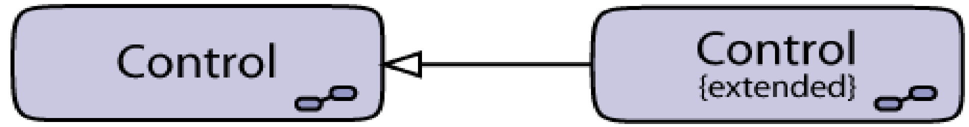
- Some semantic variation points have been mentioned before.
  - delays in event pool
  - handling of deferred events
  - entering of composite states without default entry

- Which events are prioritized?
  - completion events only
  - all internal events (completion, time, change)

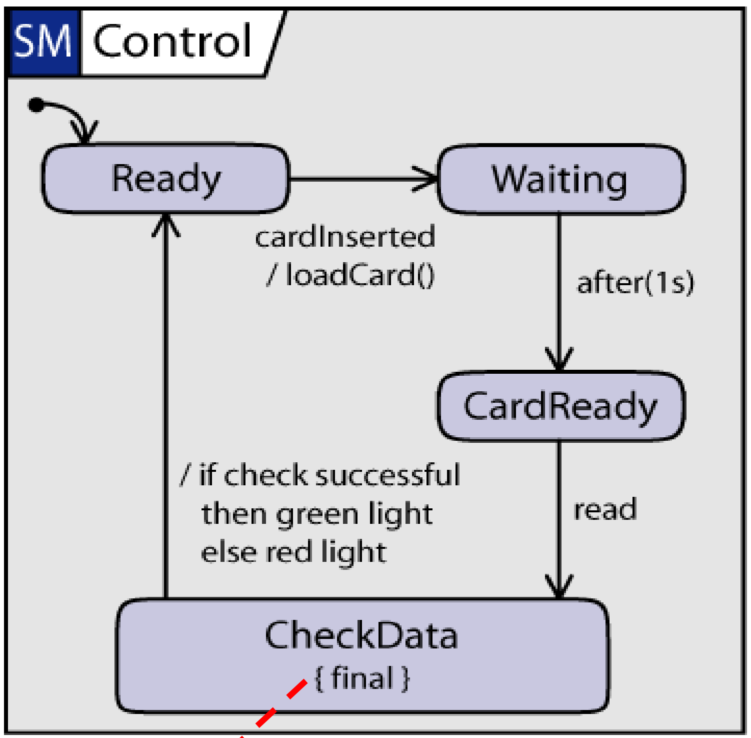
- Which (additional) timing assumptions?
  - delays in communication
  - time for run-to-completion step
    - zero-time assumption

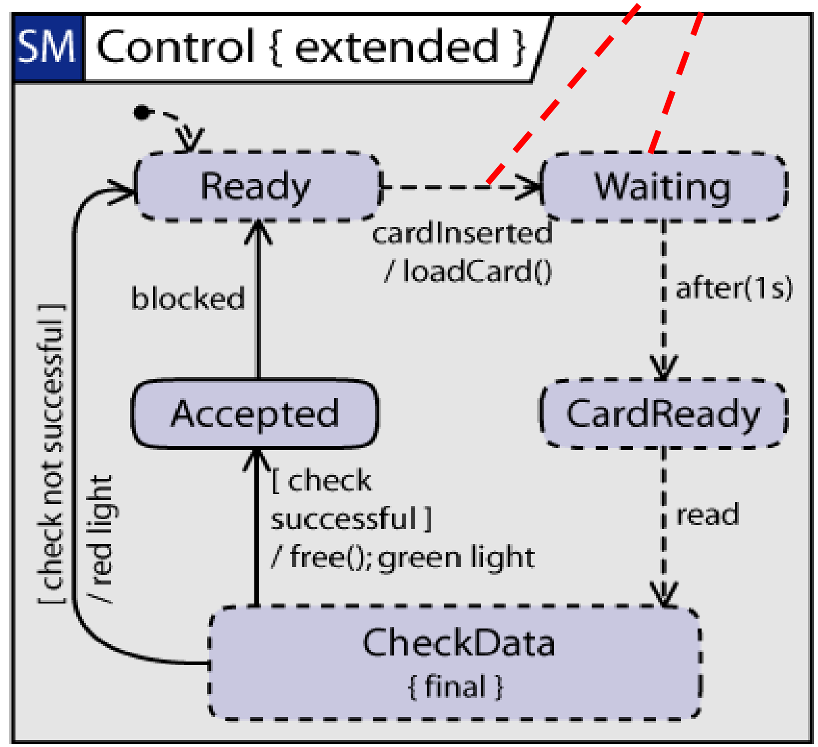
State machine refinement

- State machines are behaviors and may thus be refined.



not refined (may be omitted)

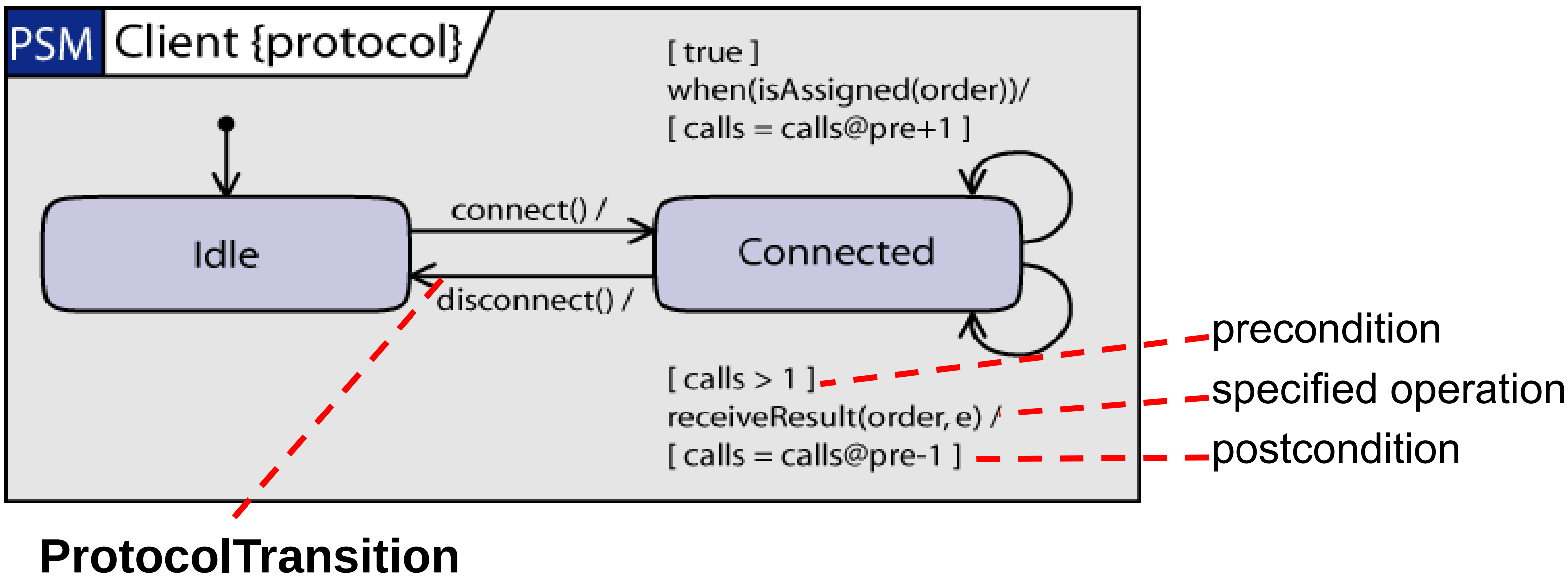




no refinement possible

Protocol state machines

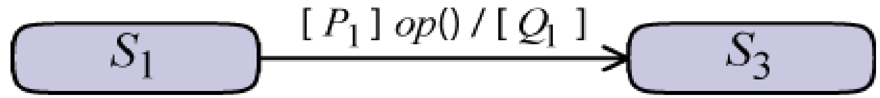
- Protocol state machines specify which behavioral features of a classifier can be called in which state and under which condition and what effects are expected.
  - particularly useful for object life cycles and ports
  - no effects on transitions, only effect descriptions



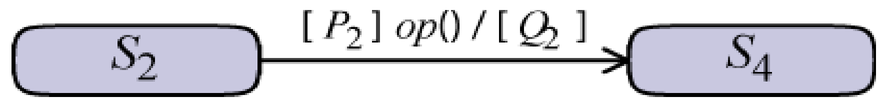
Protocol state machines

Several operation specifications are combined conjunctively:

context C::op()
pre: inState(S_1) **and** P_1
post: Q_1 **and** inState(S_3)



context C::op()
pre: inState(S_2) **and** P_2
post: Q_2 **and** inState(S_4)



results in

context C::op()
pre: (inState(S_1) **and** P_1) **or** (inState(S_2) **and** P_2)
post: (inState@pre(S_1) **and** P_1@pre) **implies** (Q_1 **and** inState(S_3))
and (inState@pre(S_2) **and** P_2@pre) **implies** (Q_2 **and** inState(S_4))

How things work together

- Static structure
  - sets the scene for state machine behavior
  - state machines refer to
    - properties
    - behavioral features (operations, receptions)
    - signals
- Interactions
  - may be used to exemplify the communication of state machines
  - refer to event occurrences used in state machines
- OCL
  - may be used to specify guards and pre-/post-conditions
  - refers to actions of state machines (`OclMessage`)
- Protocols and components
  - state machines may specify protocol roles

Wrap up

- State machines model behaviour
  - object and use case life cycles
  - control automata
  - protocols
- State machines consist of
  - Regions and …
  - … (Pseudo)States (with entry, exit, and do-activities) …
  - connected by Transitions (with triggers, guards, and effects)
- State machines communicate via event pools.
- State machines are executed by run-to-completion steps.