David Harel and Hillel Kugler

The Rhapsody Semantics of Statecharts (or, On the Executable Core of the UML)

Lecture Notes in Computer Science, Volume 3147,
Jan 2004, Pages 325 - 354.
In Integration of Software Specification Techniques for Application in Engineering 2004.

Summarized by Jingwu Li
for COMP 762

Introduction

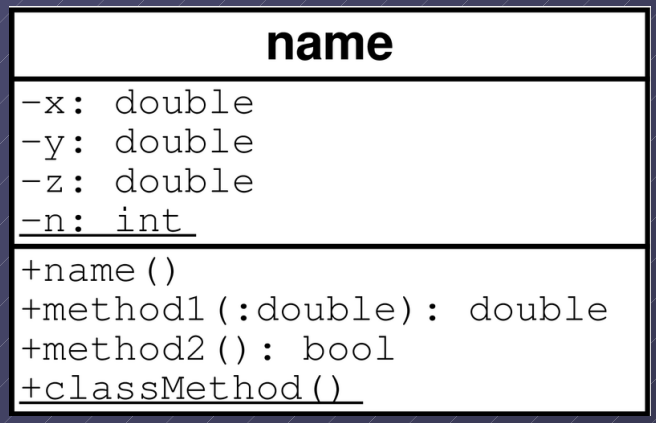
In Object-Oriented world view objects have:

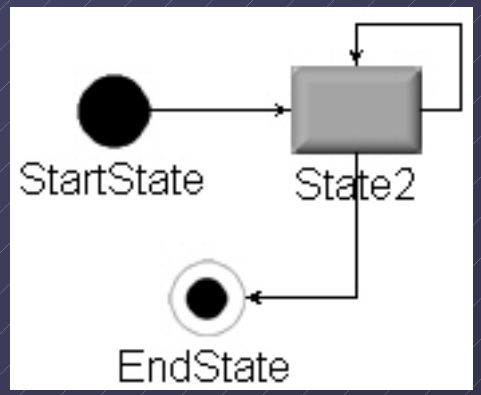
- Structure

  Class Diagram: describes structure of a system in terms of classes and associations between them.

- Behaviour

  - **Statecharts:** describes behaviour of a system in terms of possible states and transitions





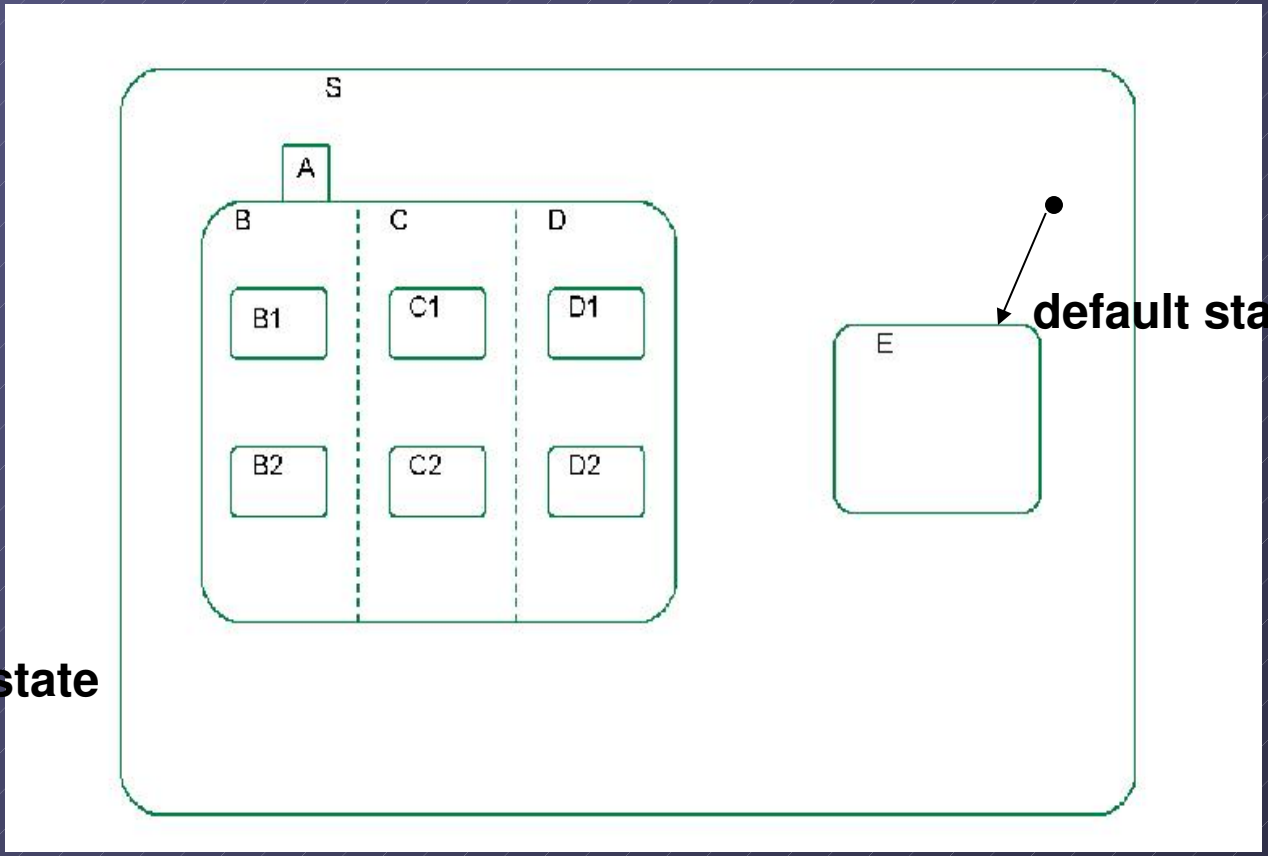
Terminology

- State
  represents the status of an object
- Transition
  the change process from one state to another
- Condition
  determines whether a transition can happen or not
- Message/event
  instantaneous change in environmental or internal condition of a system
- Action
  operation carried out or event generated upon executing a transition
- Reactive class
  class with associated Statechart describing its behaviour
- Configuration
  a set of states in which an object can reside

State

Three type of states

- OR-State (superstate): can only be in exactly one sub-state
- AND-State (superstate) : orthogonal components
- Basic state : have no substates

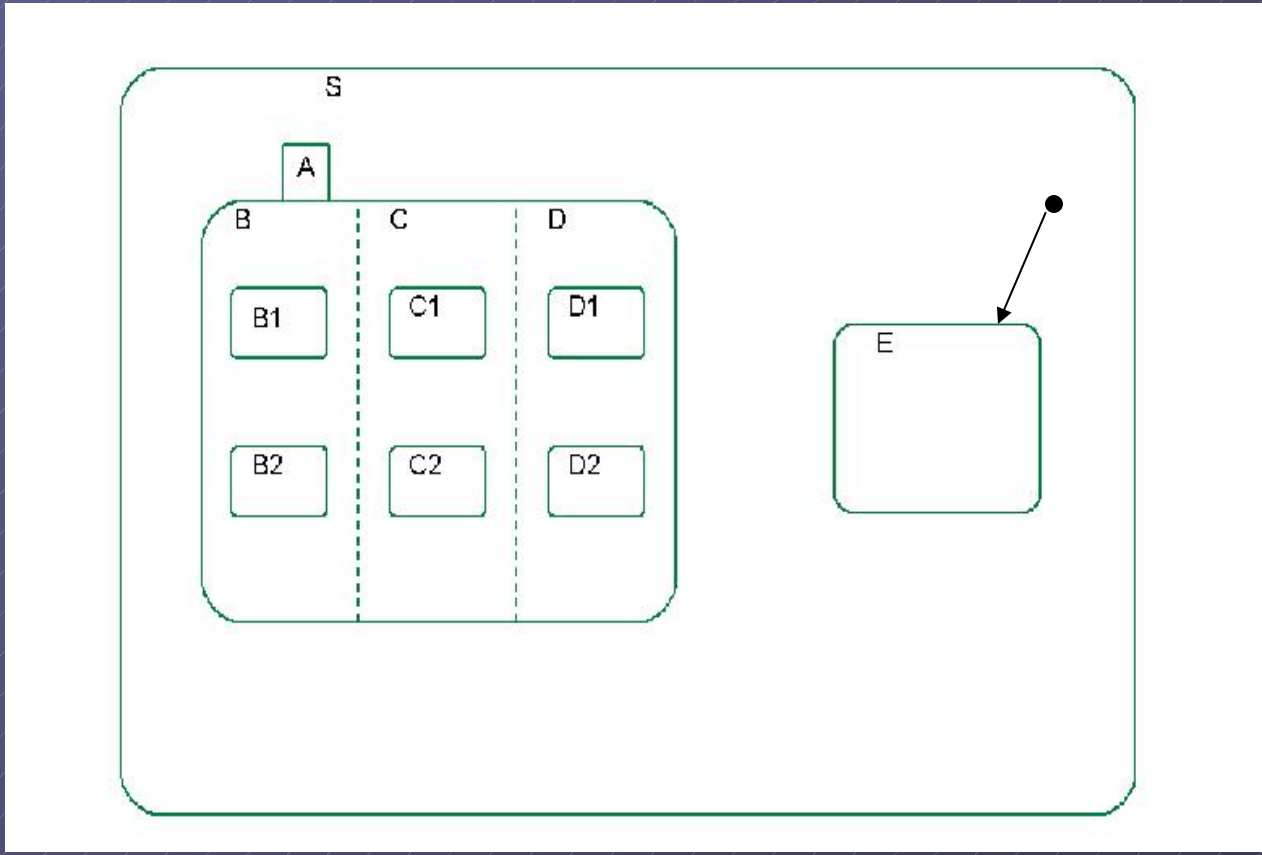


Configuration

Maximal set of states that object can be in simultaneously:

- root state
- exactly one substate for each OR-state
- all substates for each AND-state

Configuration



{ B1, B, C1, C, D2, D, A, S }

{ B2, B, C2, C, D2, D, A, S }

{ E, S }

Transition

Transition label syntax: m[c]/a

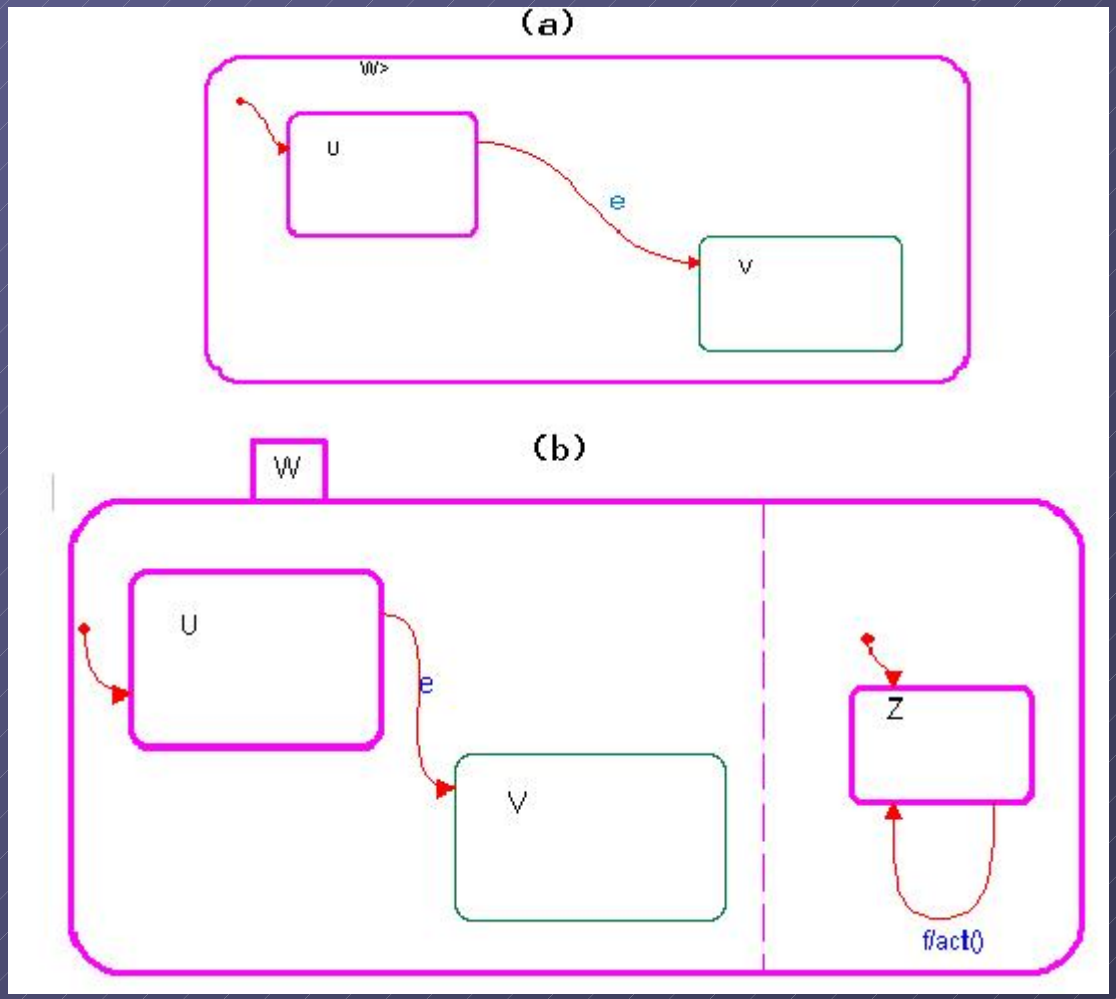
- m: message
  - event (asynchronous)
  - triggered operation (synchronous)
  - timeout event tm(t)
- c: condition
- a: action
  - generate event
  - invoke triggered operation
  - invoke primitive operation

Null transition: /a

Static Reaction

Have same format as transition labels: m[c]/a

A state has static reaction denoted by >



w>f/act()

Semantics: virtual substate which is orthogonal to ordinary substates and other SRs

Elements associated with state

- Exit Action
- Transition
- Static Reaction (SR)
- Entry Action

Behaviour

- Runs
- Run
  - Status (snapshot)
  - Step (run-to-completion)
  - Microstep

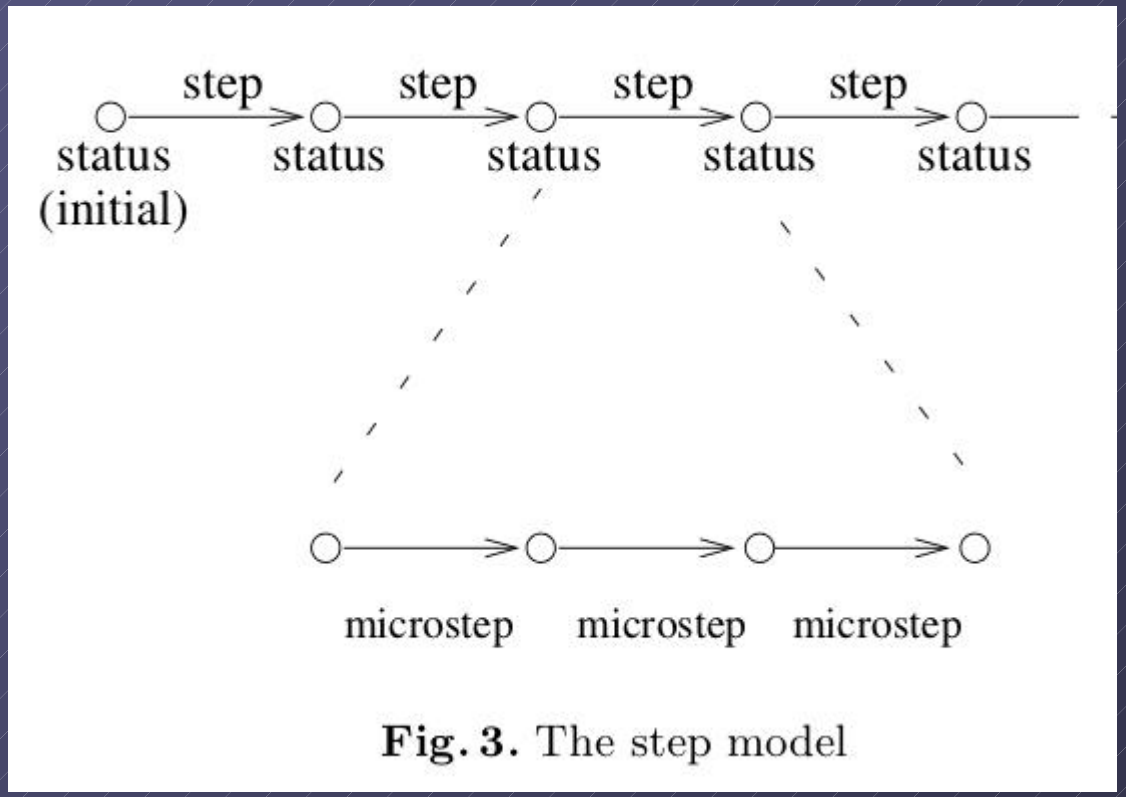
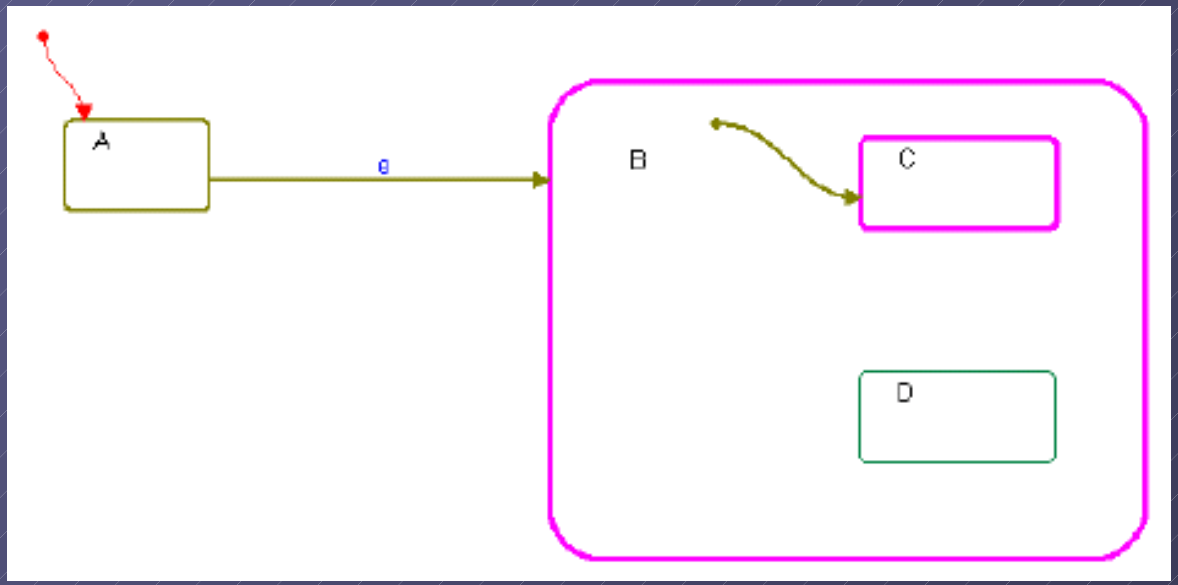


Fig. 3. The step model

Default Transition

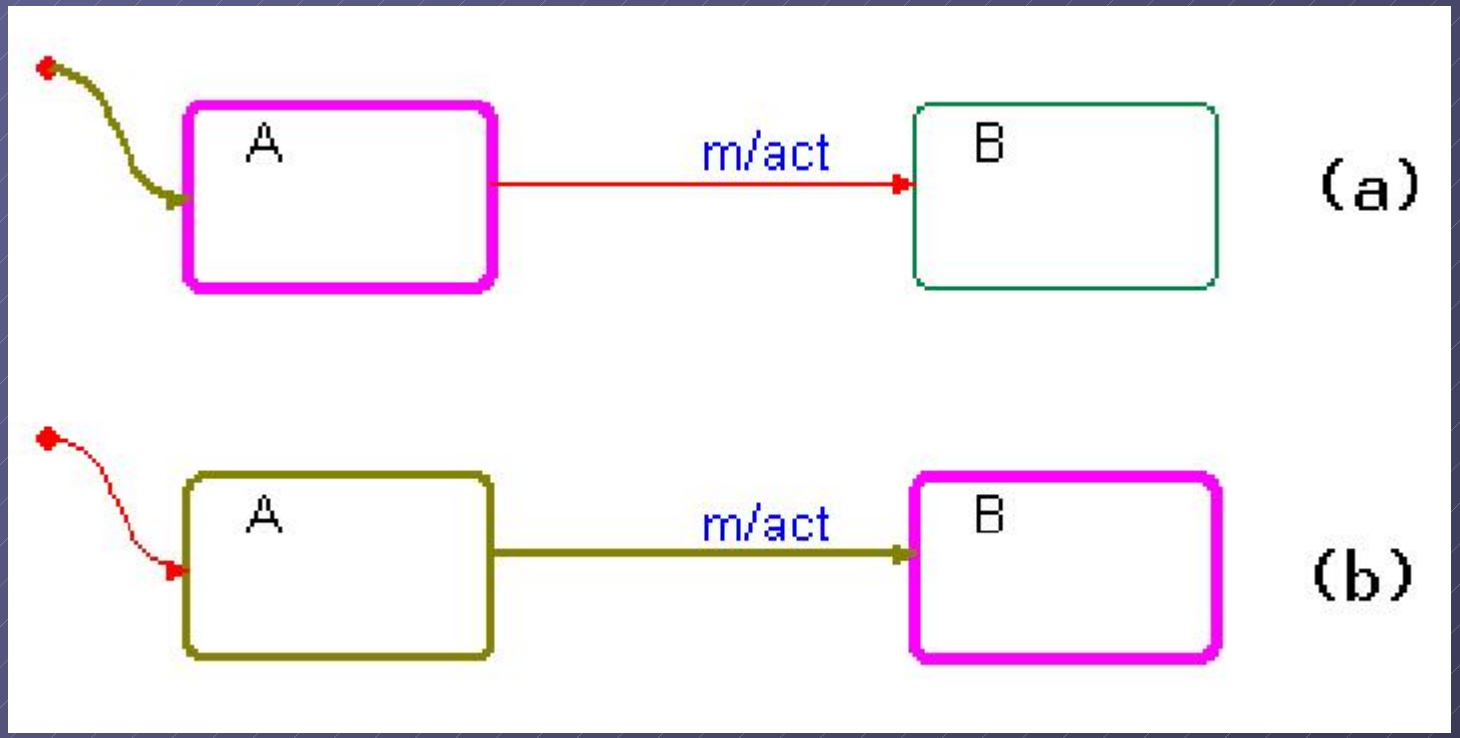
Default transition is regarded as an microstep.



Main Principles

- Changes occur in a step are sensed in the **same** step.
- Once an event is dispatched to the statechart it will live for the duration of **one step only**
- Calculations in one step are based on the **current** values of data members and the state configuration.
- **Greediness** property
- The execution of a step can take more than zero time.

State change process



(i) The exit action of state A is performed.
(ii) The action act specified by the transition is performed.
(iii) The entry action of state B is performed.
(iv) The active configuration is updated and the object is placed in state B.

act: act1;act2;act3;…; actn execute in sequence order.

Communication

- Asynchronous Communication:

  Event: o → GEN(event(p1, p2, … pN))
  : GEN(event(p1, p2, … pN))
  Event can be sub-classed.
  Event queue
  Dispatcher

- Synchronous Communication:

  Triggered operation: result = O → t(p1, p2, … pN)
  : t/reply(17)
  The return value for a triggered operation must be set within the transition.

Asynchronous Communication

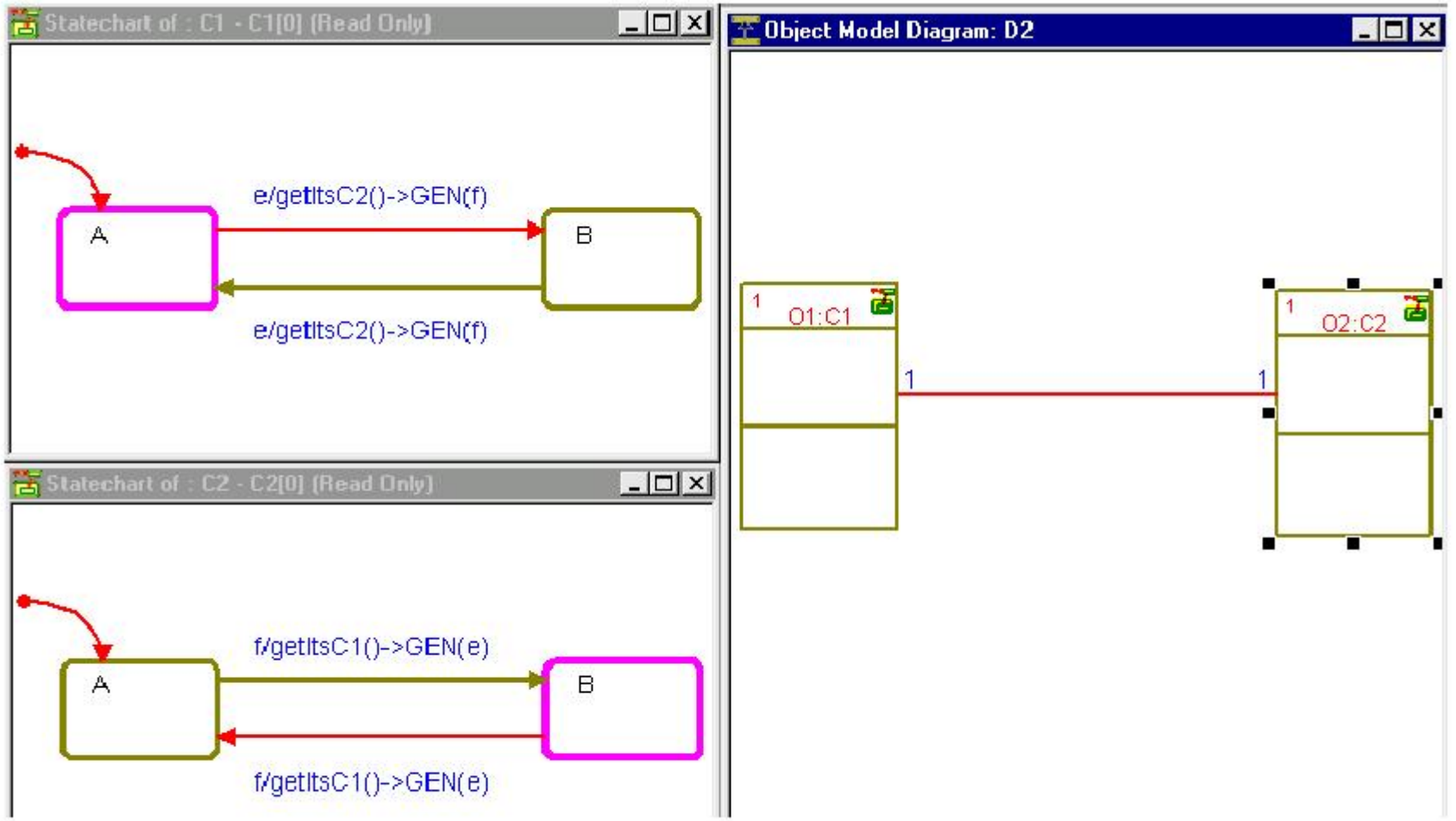
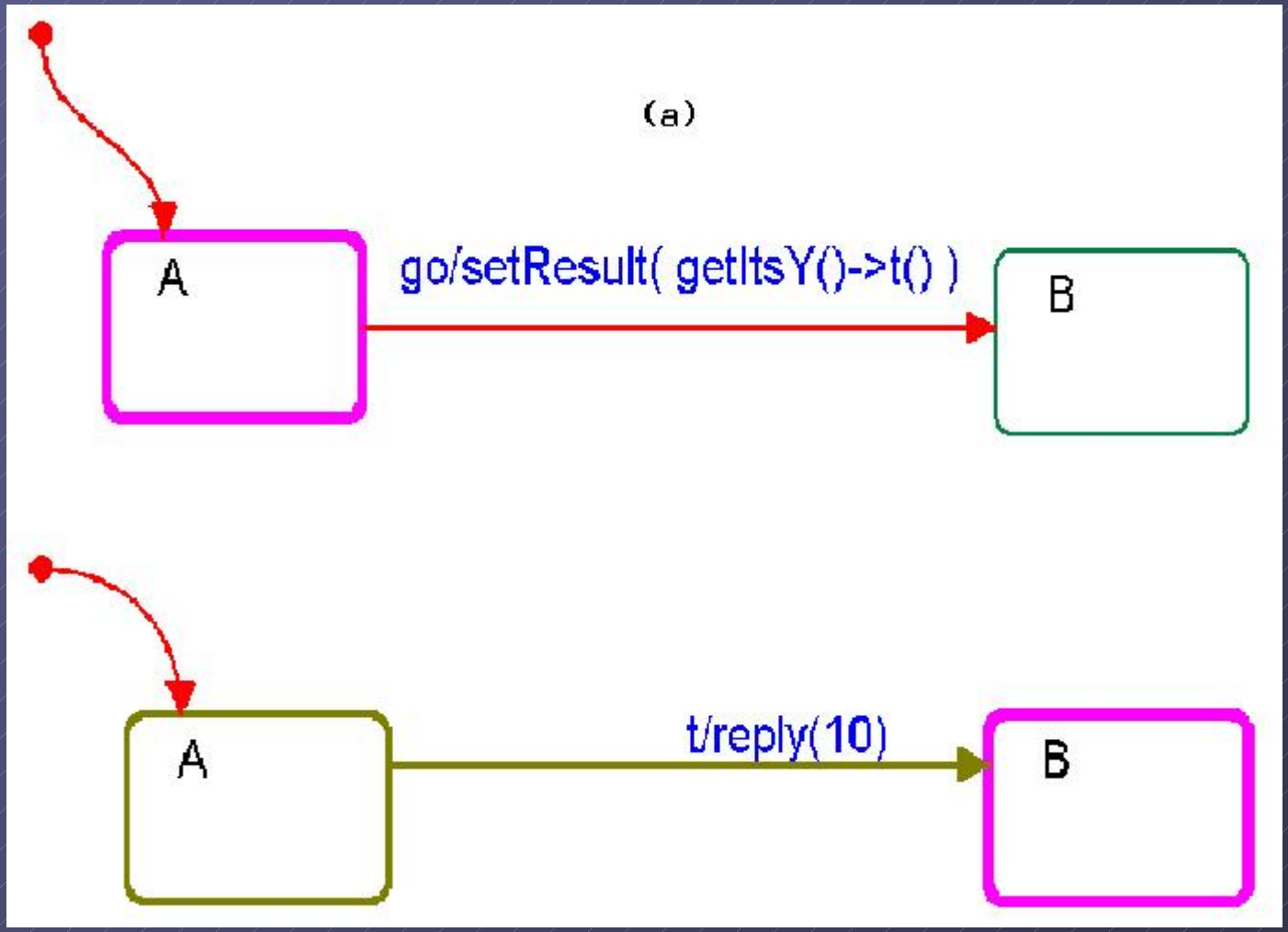


Fig. 5. Communication using events

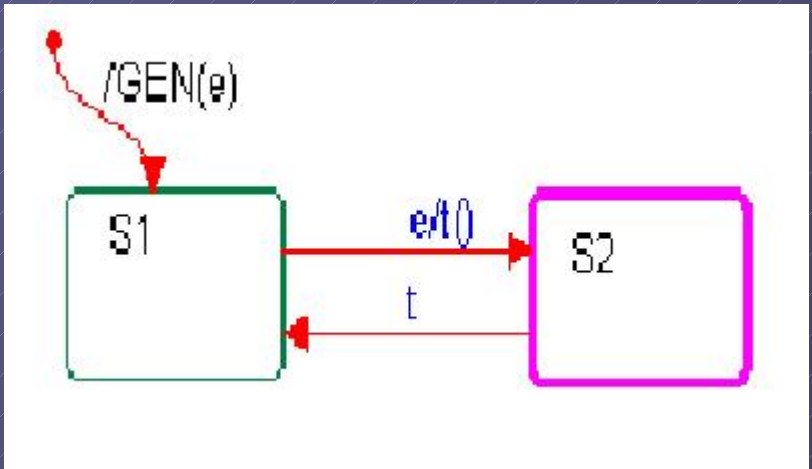
Synchronous Communication

X class

Y class



What happens when triggered operation is called when an object is not in a stable state?

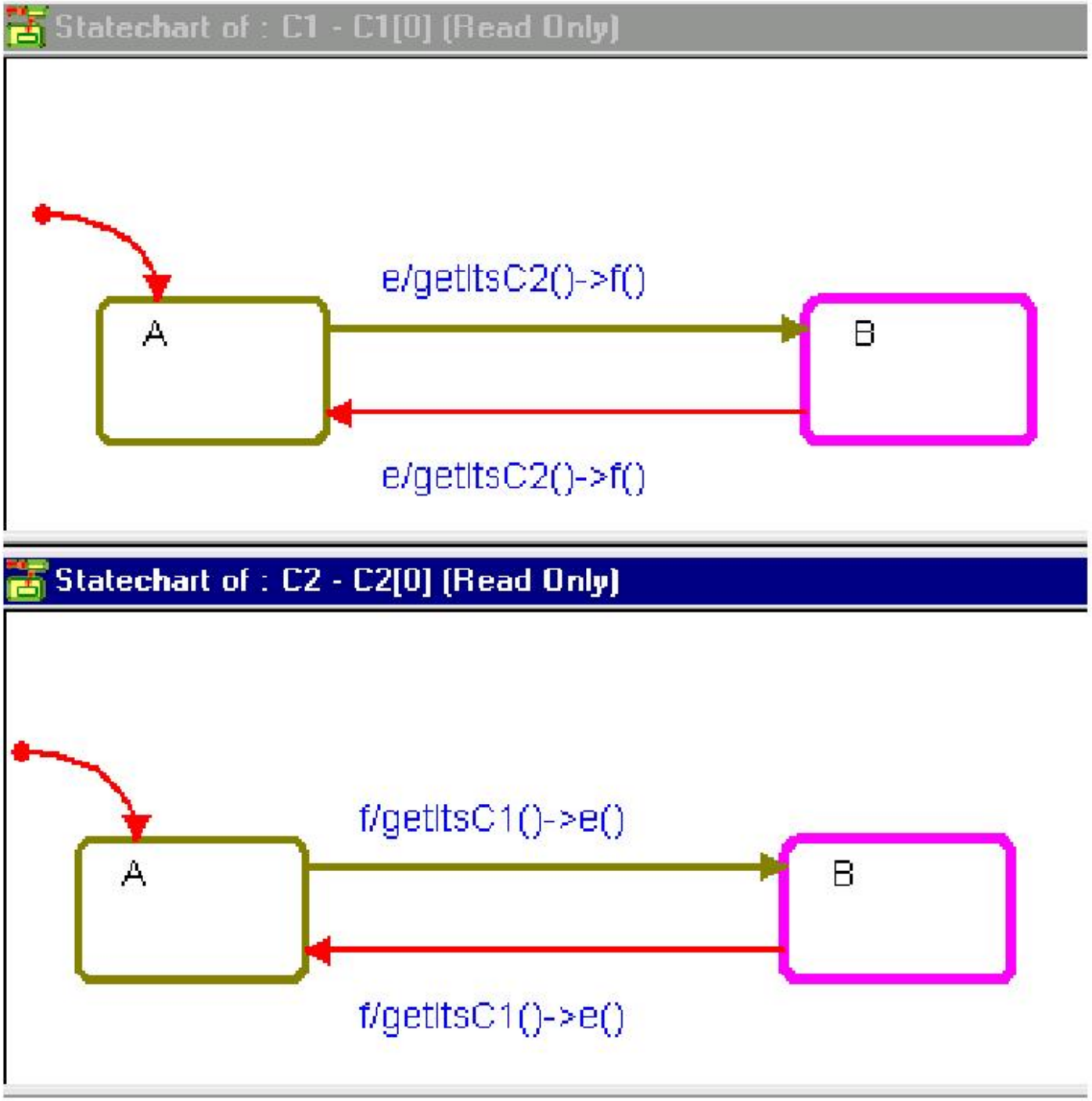


Three Ways to handle this case:

(1) Treat as deadlock

(2) Allow the transition to be completed, then to process t

(3) No effect, so stay in S2


Statechart of : C1 - C1[0] (Read Only)
e/getItsC2()->f()
A
B
e/getItsC2()->f()
Statechart of : C2 - C2[0] (Read Only)
f/getItsC1()->e()
A
B
f/getItsC1()->e()

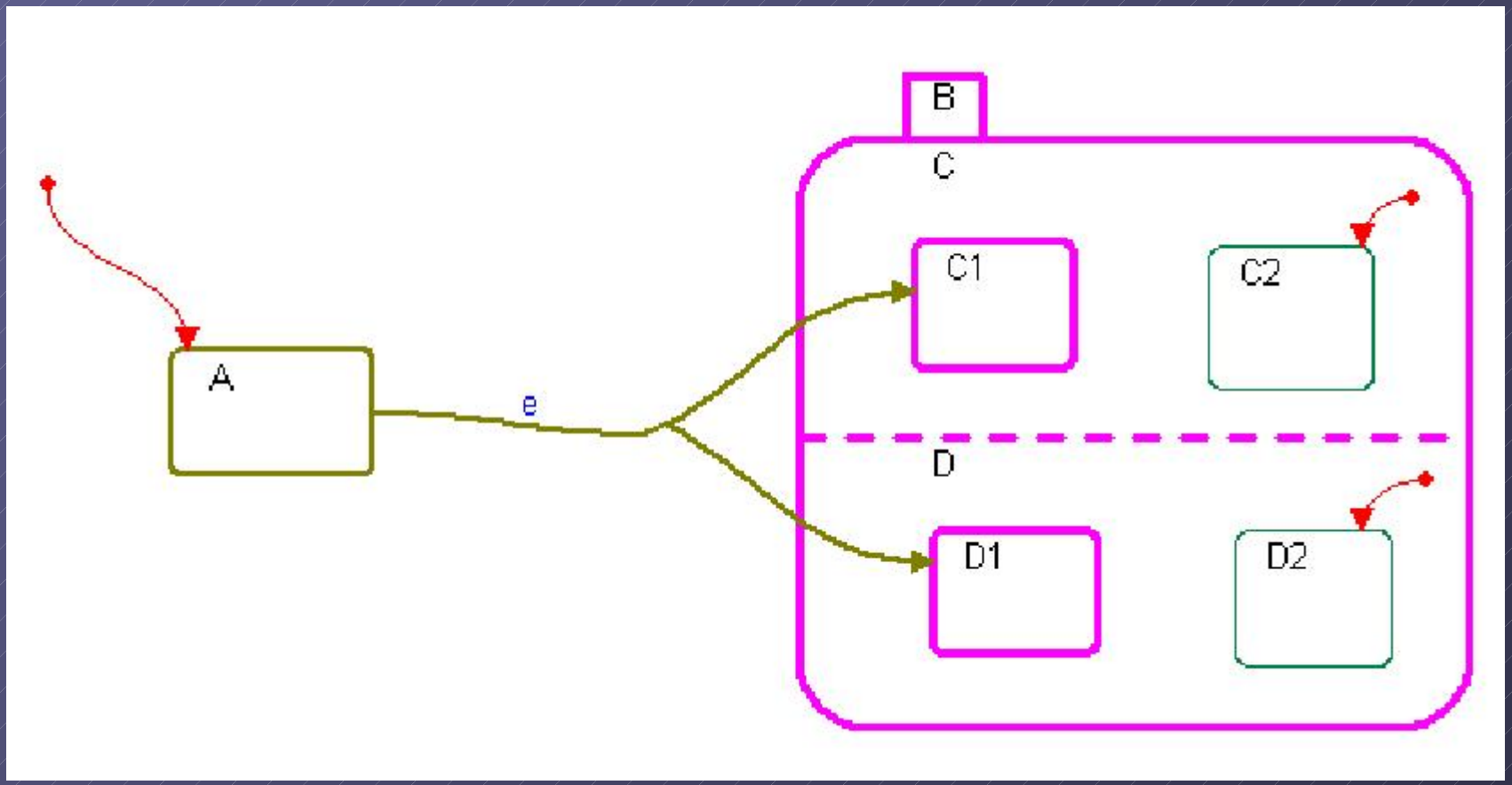
Types of Connectors

There are two types of transition connectors:

- AND-Connector
  - join connector
  - fork connector
- OR-Connector
  - junction connector
  - condition connector

Fork Connector

Split into several processes



Join Connector

Synchronize processes

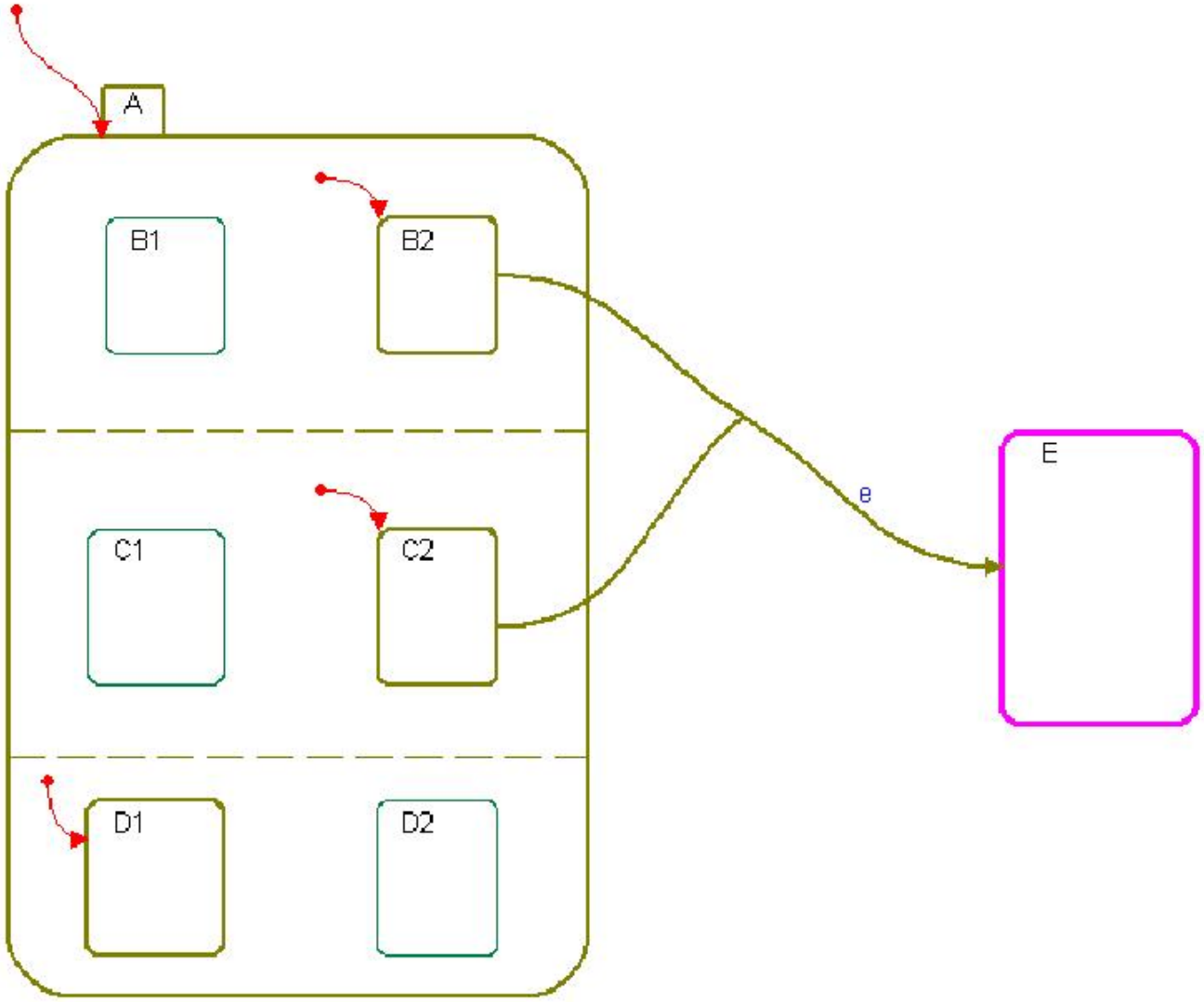


Fig. 11. A join connector

Junction Connector

Either … or…

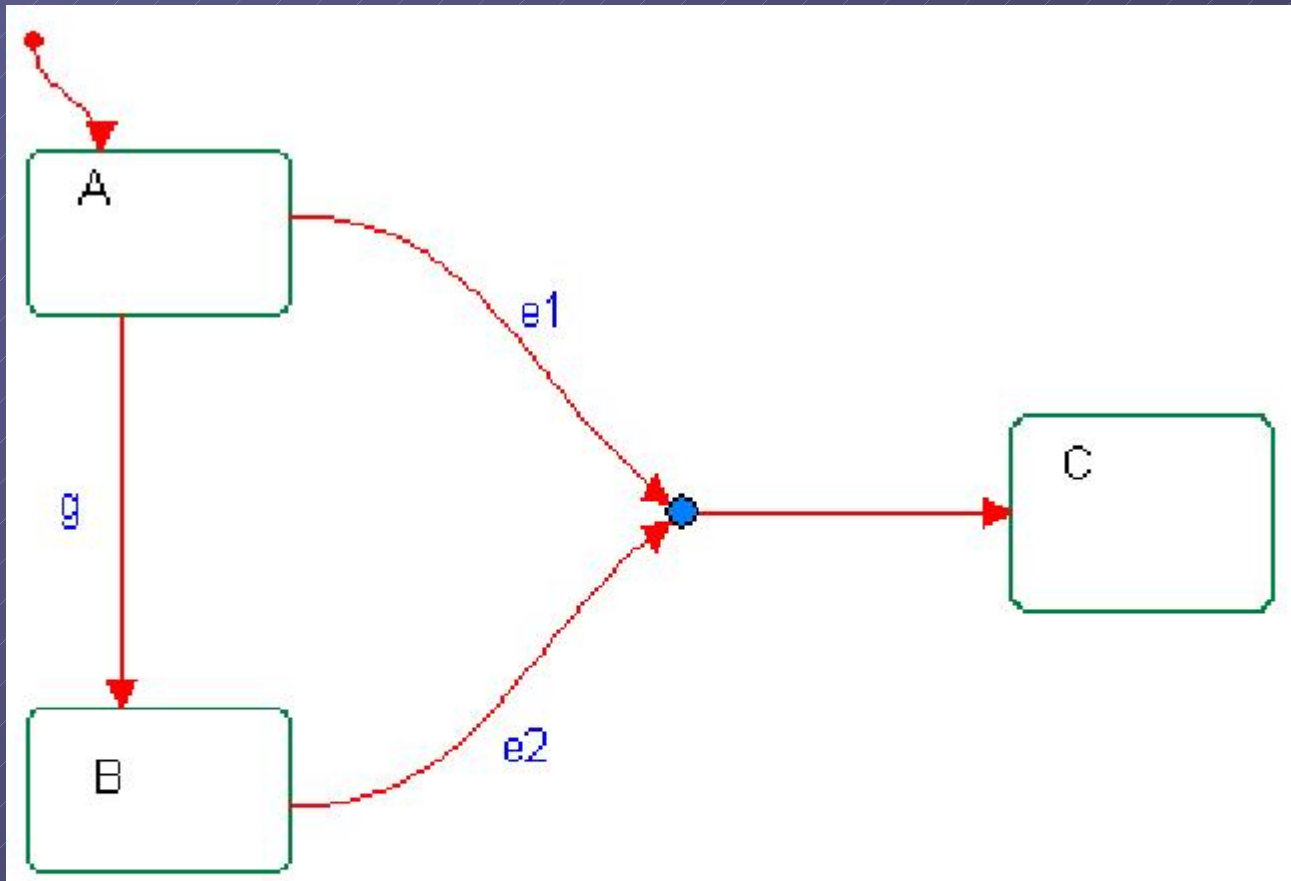


Fig. 12. A junction connector

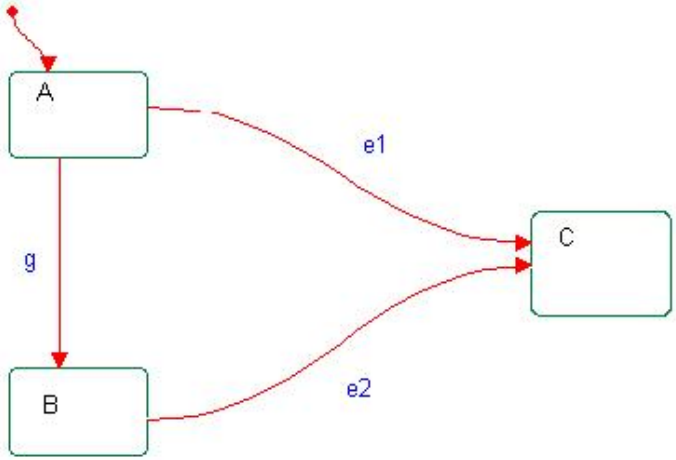


Fig. 13. A construct equivalent to a junction connector

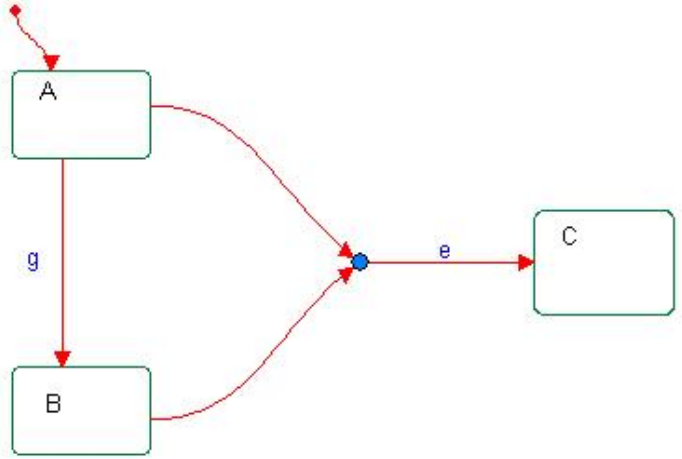


Fig. 14. A junction connector with a common label

Condition Connector

Branch

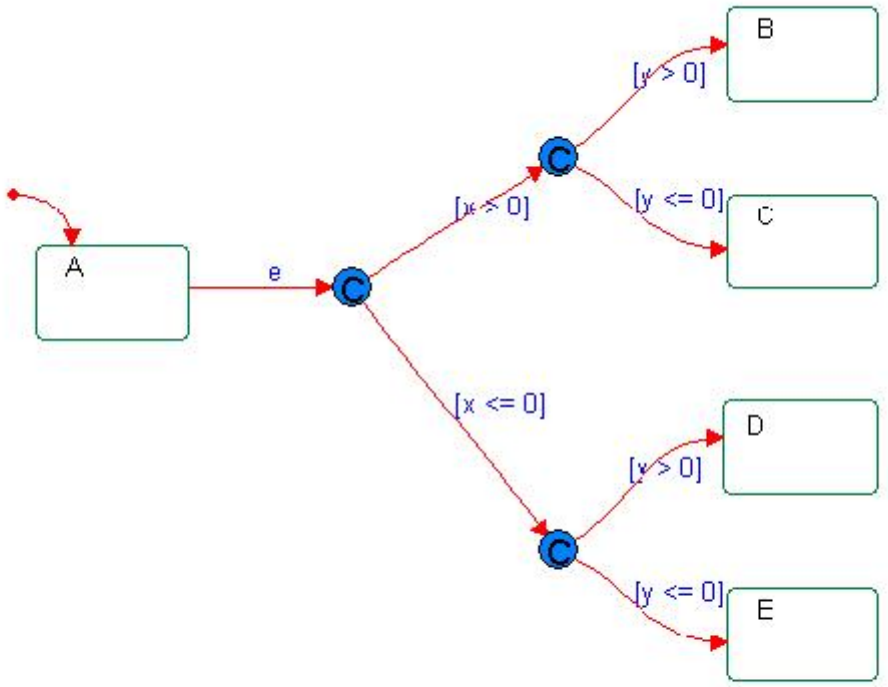


Fig. 15. Nested condition connectors with a common label

Condition Connector (ctn)

When taking a transition, first all guards are evaluated, and only then are the actions performed.

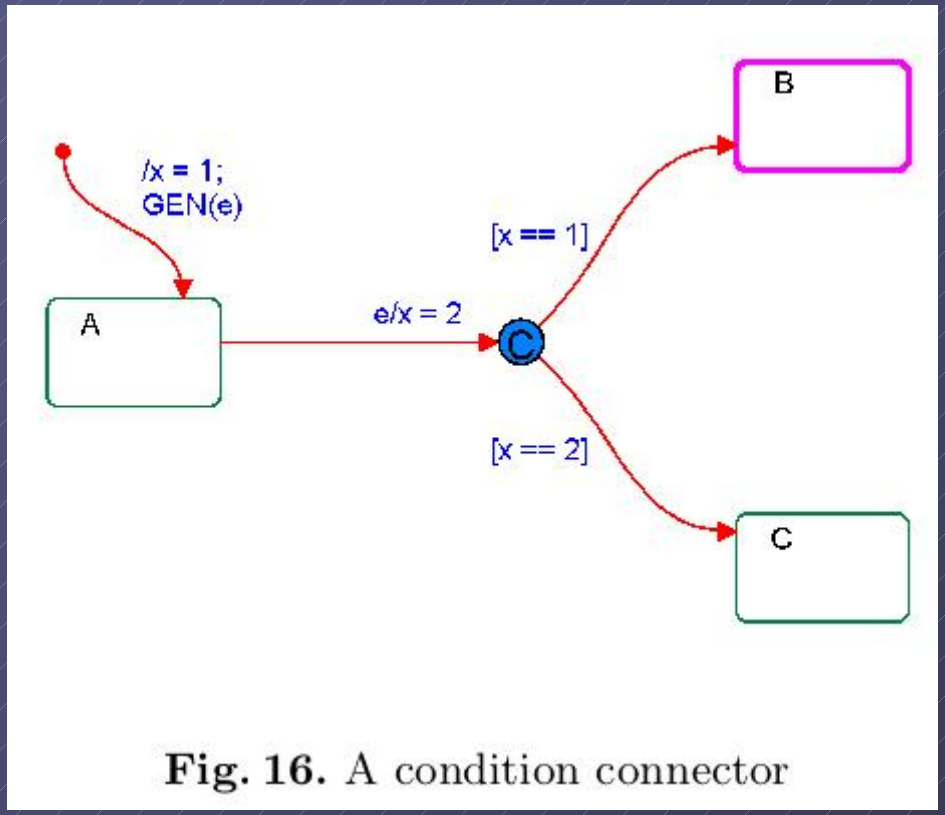


Fig. 16. A condition connector

Entering state A, → B

Compound Transition

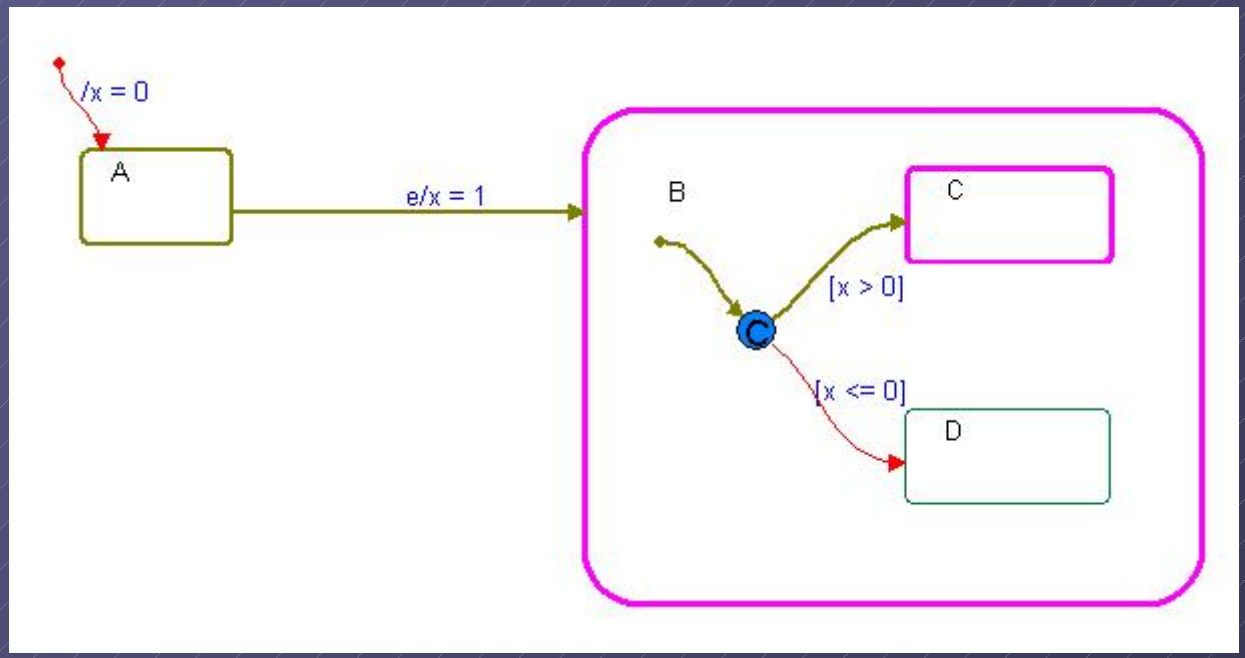
Compound Transition consists

- Transition Segments (connected by Connectors)
- Transitions

Full CTs : always leads from one legal state configuration to another. (Step)

statechart cannot be in a non-basic state without the ability to enter appropriate substates

Default transition is regarded as an microstep.



In state A, e occurs → C

Transition Scope

- Intent: Scope is used to determine which states should be exited and which entered while taking a CT.
- Def: The scope of a CT is the **lowest OR state** in the hierarchy of states that is a **proper common ancestor** of all the source and target states.
- Result: Taking the CT will result in a change of the active configuration involving only substates in the scope.

Example 1

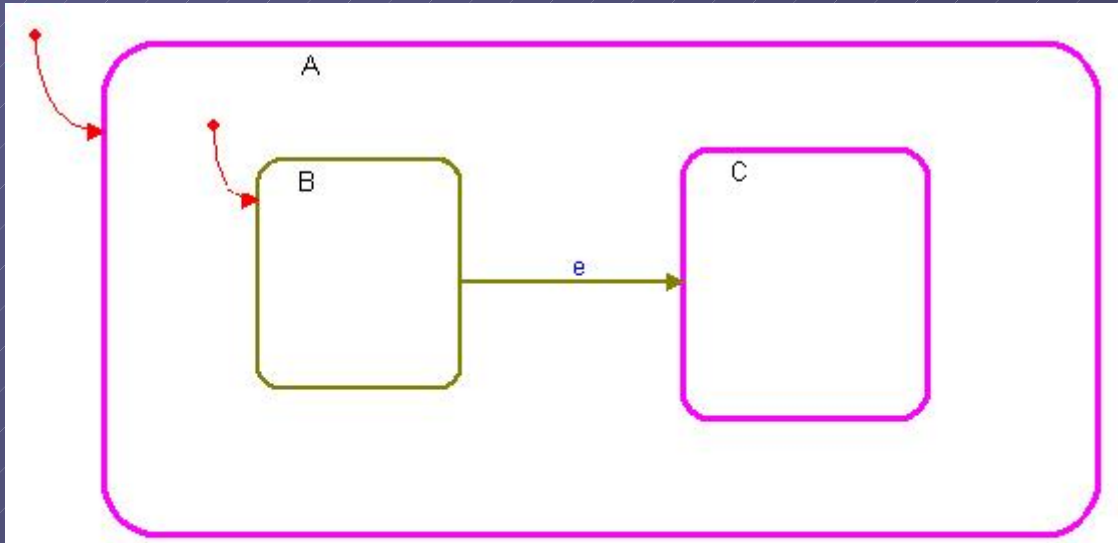


Fig. 20. The scope of a transition

A is the scope of transition e.

Example 2

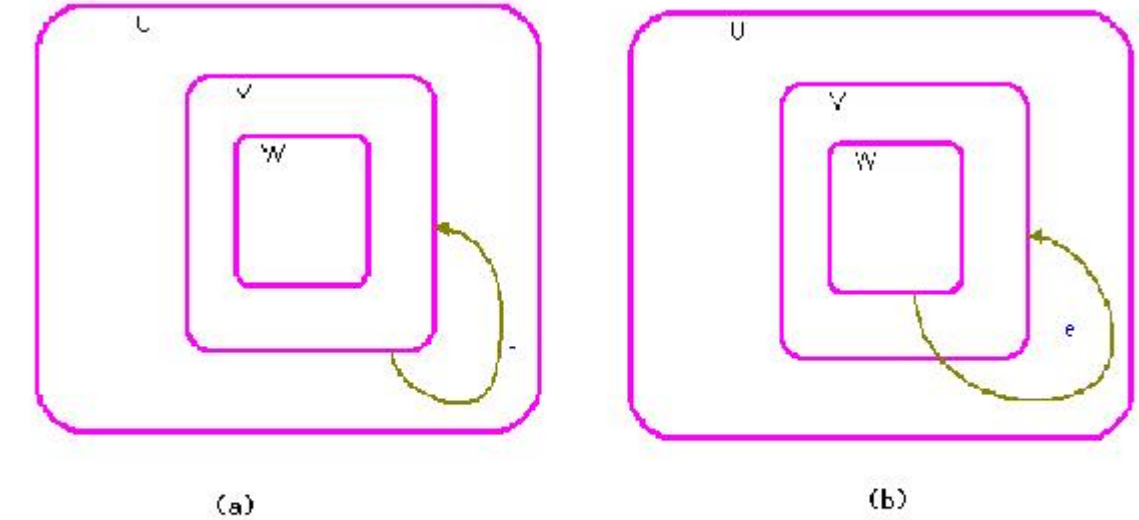


Fig. 21. The scope of a transition

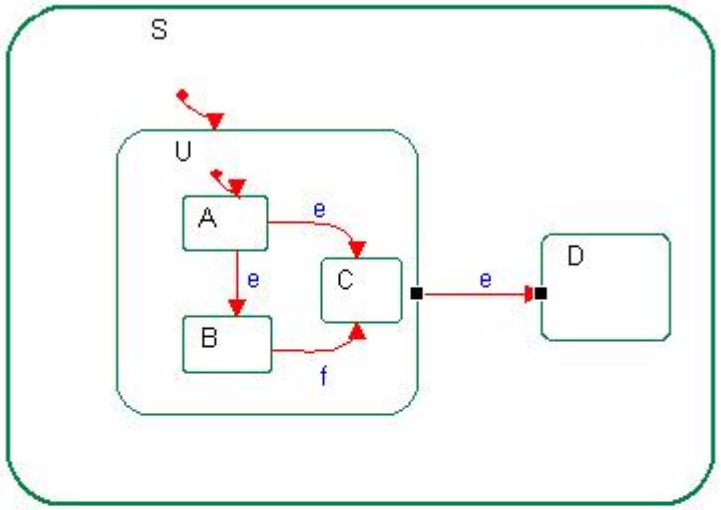
U is the scope of transition e.
(exit w, v, enter v, w)

Transition Conflict

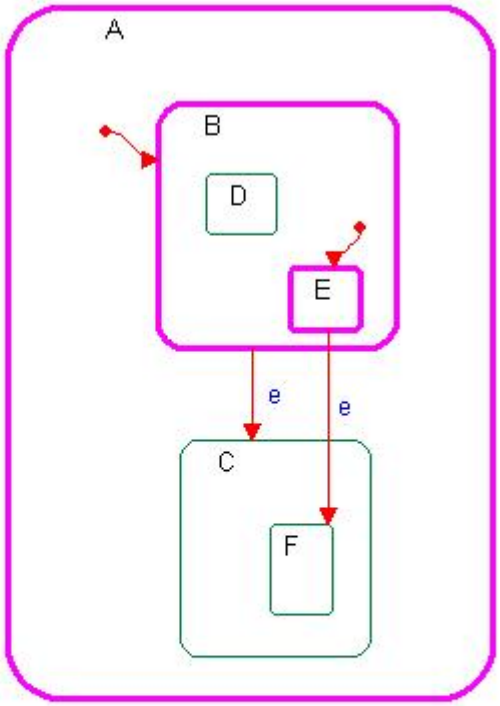
We say that two transitions are in conflict if there is some common state that would be exited if either of them were to be taken.

How to deal with conflict (two case):

- When a message can trigger several conflicting transitions priority is given to lower level source states
- Detects nondeterminism during code generation and does not allow them



(a)



(b)

Fig. 23. Conflicting transitions

U → D conflicts with
A → C, A → B, B → C

outside-in priority
E → F

History Connector

- A history connector is used to store the most recent active configuration of a state.
- Each state can have at most one history connector.
- The semantics of the history connector is that when the connector is the source of a CT, the statechart transitively enters the most recently visited active state(s).

History Connector

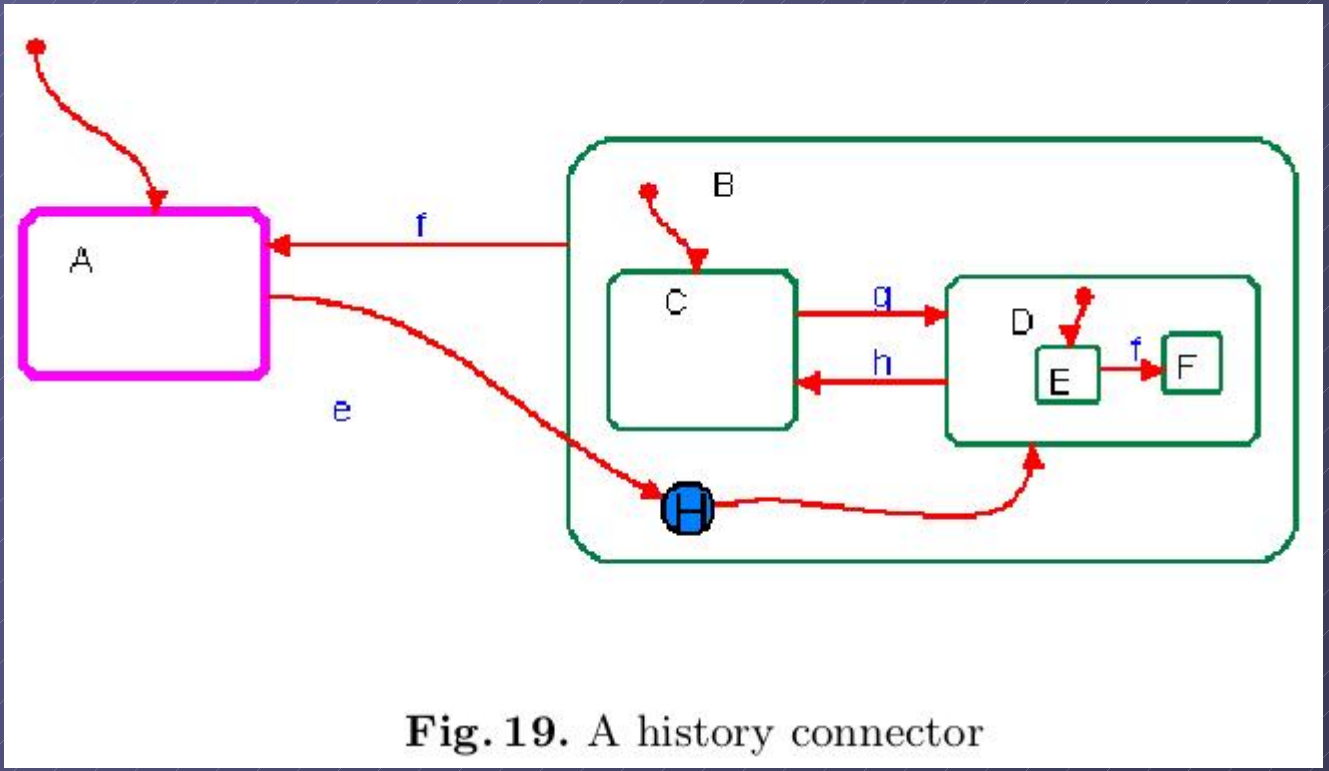


Fig. 19. A history connector

A , e, D, E, f, F, f , A (H), e, D F
No enter D action, enter F action, History action

Step Algorithm

```
procedure StepCycle ()
begin
  loop forever
    while Event-Queue ≠ empty do
      ev ← Get-Event-From-Queue
      dest ← Get-Destination-Of-Event
      if dest still exists then
        dest → takeEvent(ev)
      else
        Ignore ev
      end if
    end while
  end loop
end
```

takeEvent()

- Determine CTs/SRs that will fire due to the event;
- Perform those CTs/SRs

  For each transition do:
  (1) Update histories of exited states;
  (2) Perform exit actions of exited states, from inner to outer state;
  (3) Perform actions on the CT/SR sequentially based on the order in which they are written on the transition;
  (4) Perform entry actions of the entered states, from outer to inner state;
  (5) For lowest level states entered, which are not basic states, perform default transitions (recursively) until basic states are reached;
  (6) Update the active configuration;
- Process null transitions;
- Control returns to the dispatcher and new messages can be dispatched.