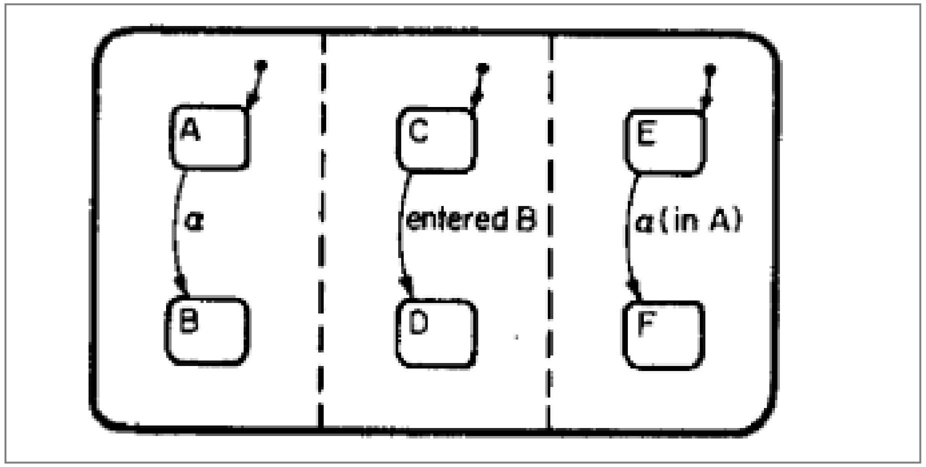
The Statemate Semantics of Statecharts

- There is no Statecharts standard, thus no official semantics.
- Behavior of the SUD is a set of possible runs, each representing the response of the system to a sequence of external stimuli generated by the environment. A run is composed of a sequence of steps, the snapshots of the system state between runs being called status.
- Various semantics proposals; main source of discourse: Should changes that occur in a given step take effect in the current step, or the next one?

When do changes take effect?



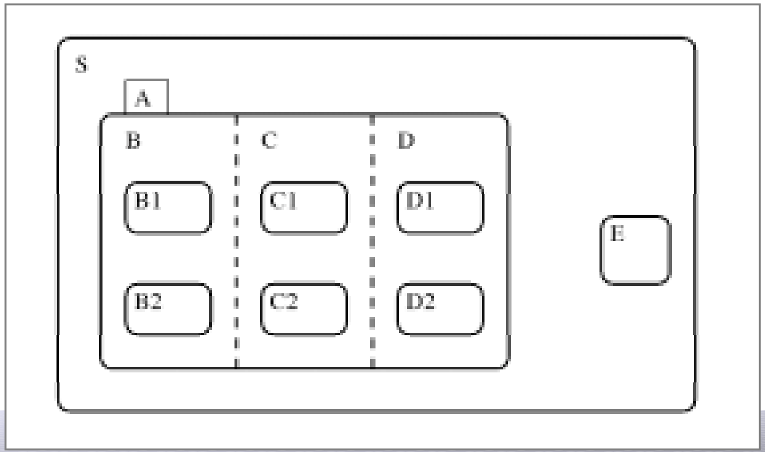
Note: The execution of a step takes zero time.

Principles of the Semantics

1. Reactions to external and internal events, and changes that occur in a step, can be sensed only after completion of the step.
2. Events live for the duration of one step only (ie the one following that in which they occur) and are not remembered in subsequent steps.
3. Calculations in one step are based on the situation at the beginning of the step.
4. Greediness property: A maximal subset of nonconflicting transitions and SRs is always executed.
5. The execution of a step takes zero time.
6. The time interval between the executions of two consecutive steps is not part of the step semantics.

Basic Definitions

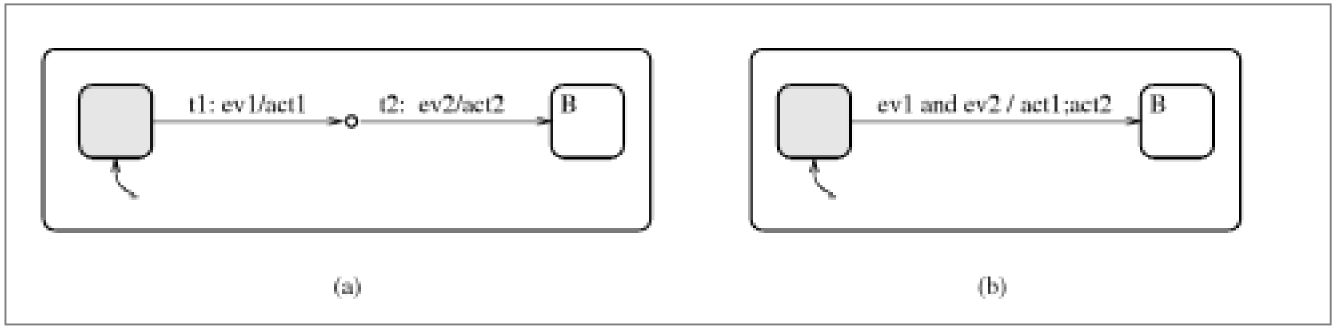
- Configuration: maximal set of states the system can be in simultaneously.
- Let R be a root state. A configuration C obeys the following rules:
  1. C contains R
  2. If C contains a state A of type OR, it must also contain exactly one of A's substates
  3. if C contains a state A of type AND, it must also contain all of A's substates
  4. The only states in C are those required by 1-3.
- Basic configuration: maximal set of basic states in a legal configuration.



Source [1]

Compound Transitions

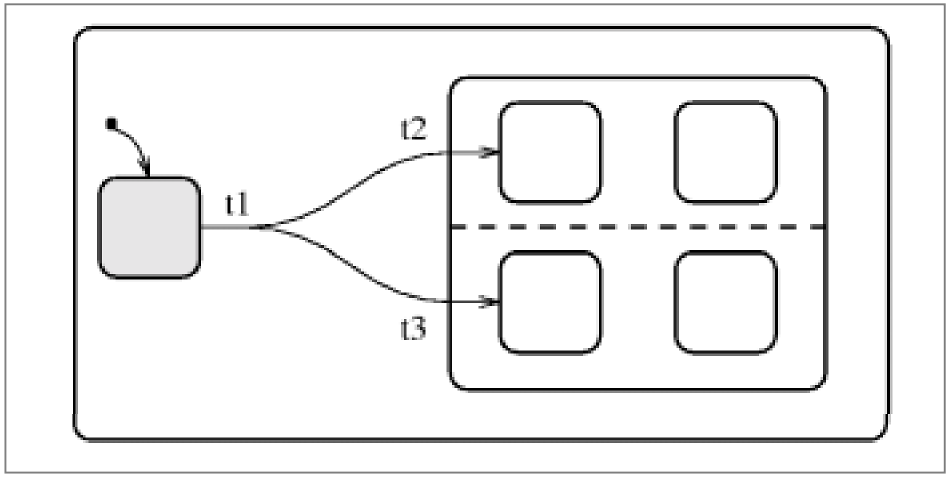
- Transition segment: labelled arrows
- Basic compound transition (CT): maximal chain of transition segments, linked by connectors that are executable simultaneously as a single transition.
- Trigger of a CT: conjunction of the triggers of its constituting segments, action is concatenation of the actions thereof.

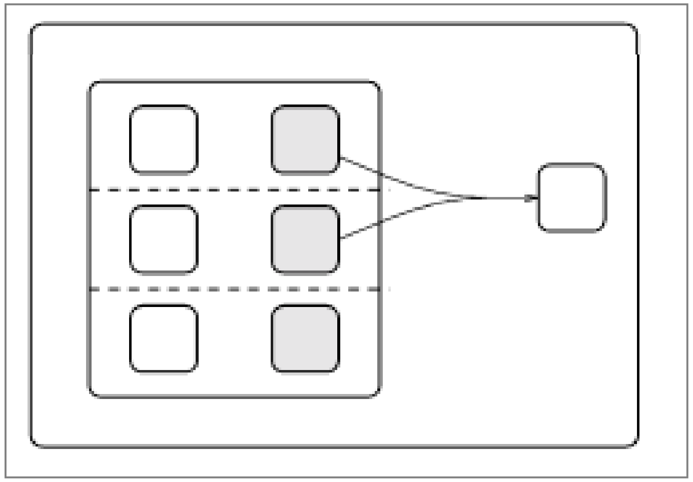


Source [1]

Compound Transitions

- Two types of connectors:
  1. AND connectors: transition segments connected to an AND connector will all participate in the same CT.
     Types of AND connectors: joint, fork.



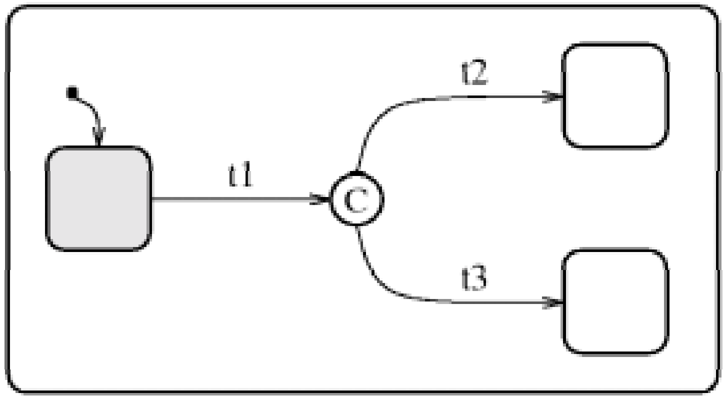


Source [1]

Compound Transitions

- Two types of connectors:

2. OR connectors. Let T1 and T2 be the sets of transition segments leading to and emanating from an OR connector C. Any CT that contains a segment from T1∪T2 must contain exactly one segment from T1 and one from T2.
   Types of OR connectors: condition, selection, junction.



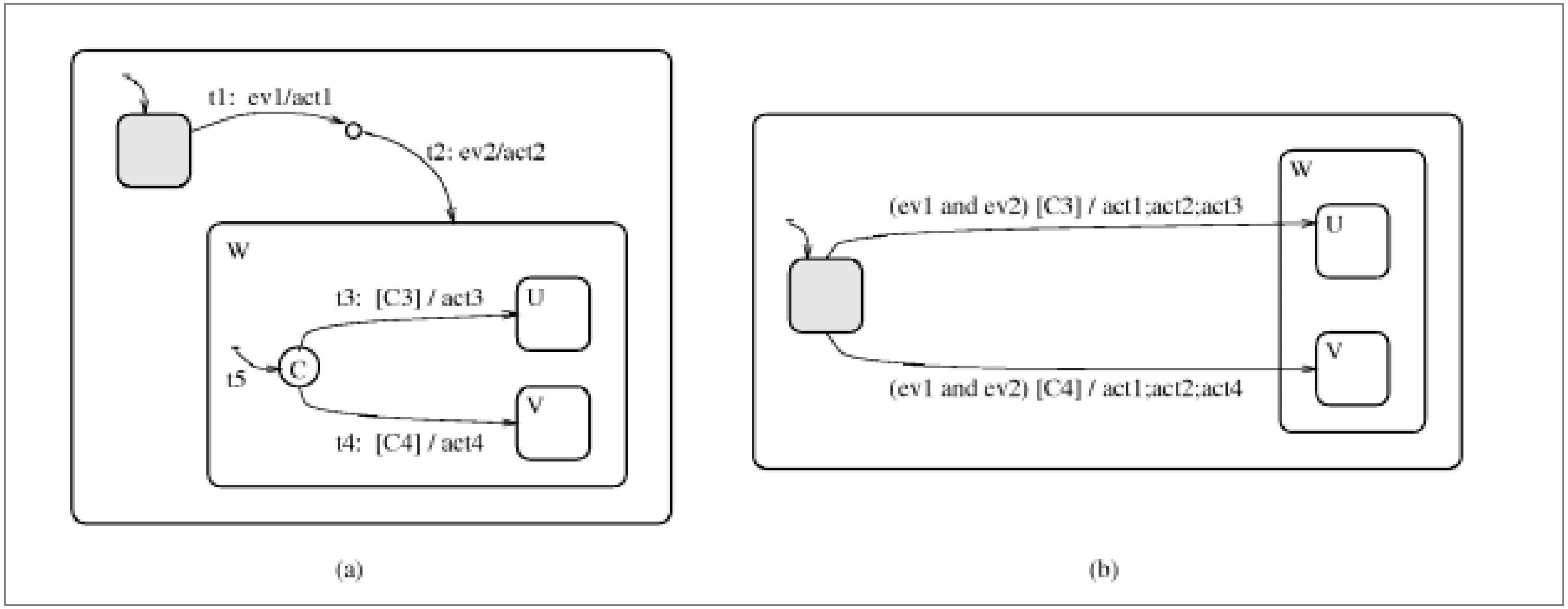
Source [1]

Compound Transitions

- Initial CT: source is a state.
- Continuation CT: source is a default or history connector.
- Full CT: combination of one initial CT and continuation CTs which, when executed, lead the system to a full basic configuration.
- Remark: a CT can have several sources and several targets.
- The target set must be maximal: if it contains a descendant of a component of an AND state, it contains descendants of all of its other components too.
- CT is enabled: at the beginning of the step the system is in all the states of its source sets, and its trigger is true.

Compound Transitions

- Complex Example:

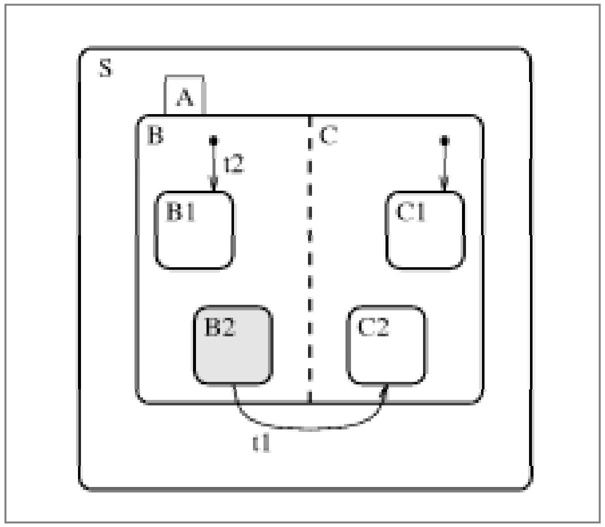

t1: ev1/act1
t2: ev2/act2
W
t3: [C3] / act3
U
C
t5
V
t4: [C4] / act4
(a)
W
(ev1 and ev2) [C3] / act1;act2;act3
U
V
(ev1 and ev2) [C4] / act1;act2;act4
(b)

Source [1]

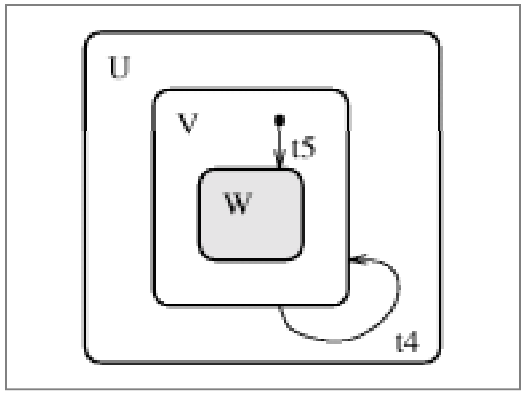
Compound Transitions: Scope

- The scope of a CT t_r is the lowest OR-state in the hierarchy of states that is a proper common ancestor of all the sources and targets of t_r including non-basic states that are explicit sources or targets of transition arrows appearing in t_r. History connectors that are targets of such arrows are represented by the states in which they are acutally drawn.
- Thus: the scope is the lowest state in which the system stays without exiting and reentering when taking the transition.
- Remark: the notion of scope does not depend on the way the arrow itself is drawn, but on its sources and targets only.

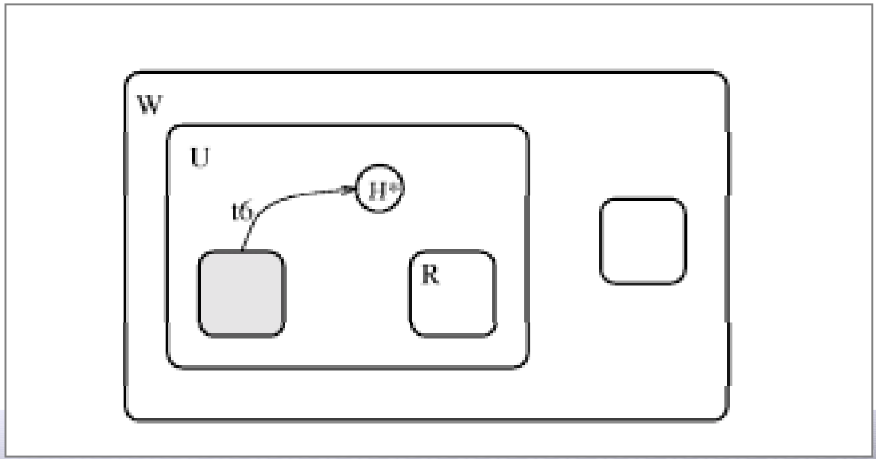
Compound Transitions: Scope



a



b

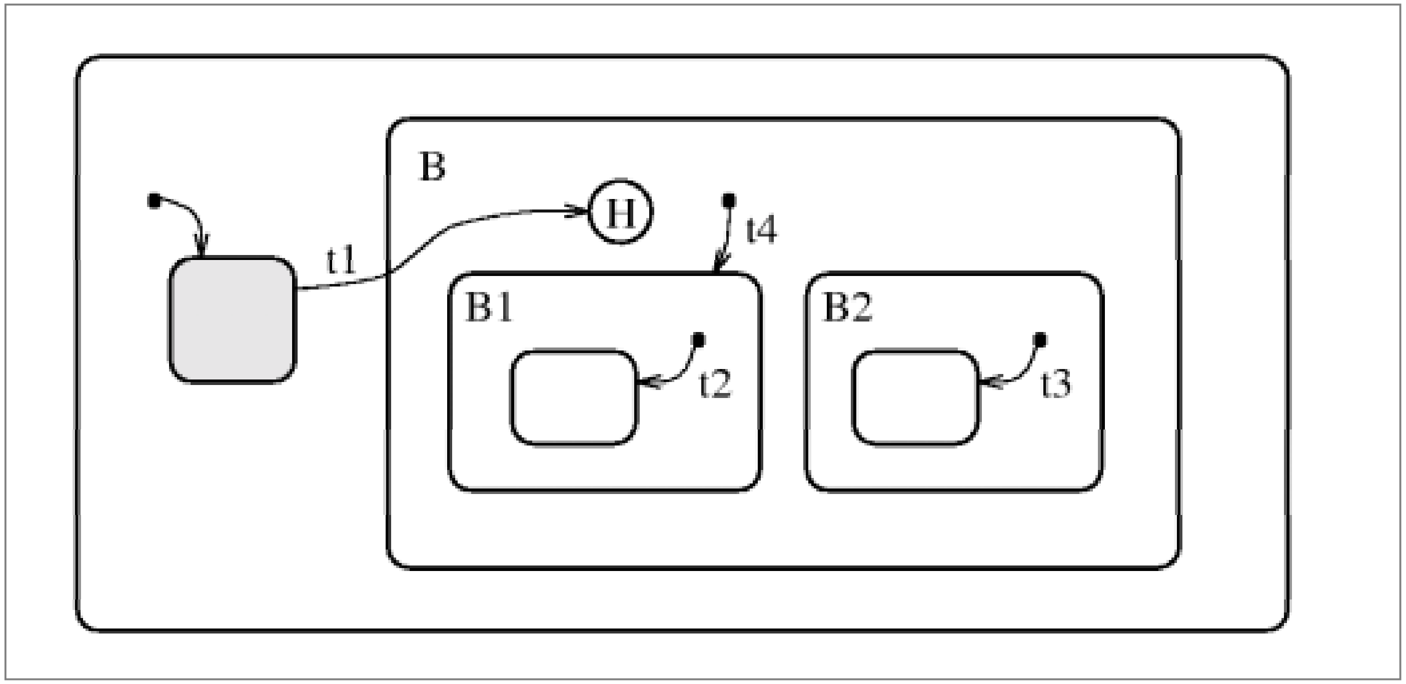


c

Evaluating History

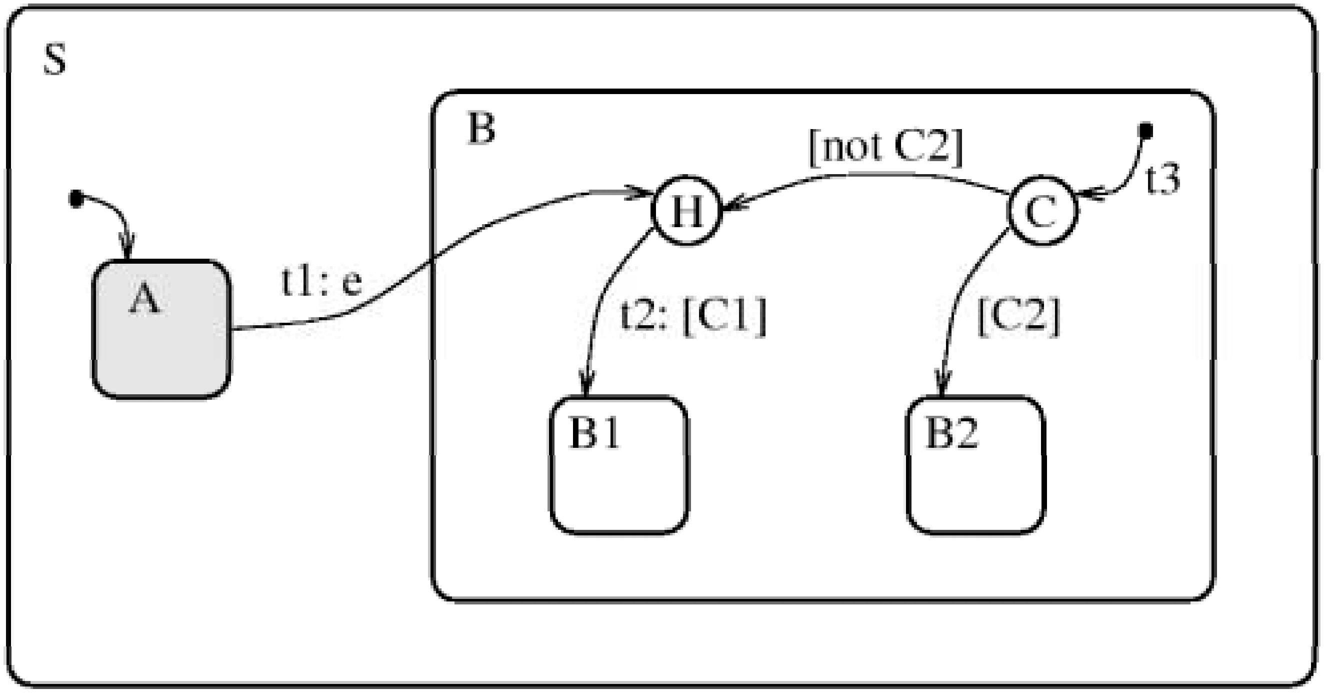
```
Suppose a Compound Transition t1 is executed whose target is a
   history connector h of state S.
If S has history
   if h is an H connector
        let S' be the substate of S which the system was in
        when most recently in S; t1 is treated as if its
        target is S'.
   else /* h is a H* connector */
        let S' be the basic configuration relative to S
        which the system was in when it was most recently
        in S. t1 is treated as if its targets are all the
        states in S'.
else /* system was never in S, or S's history was erased by
         clh since it was last in S */
   t1 is treated as if its target is S; however, if there are
   transitions emanating from h, then these have priority
   higher than those emanating from the default connector of
   S.
```

Evaluating History – Example 1



Source [1]

Evaluating History – Example 2

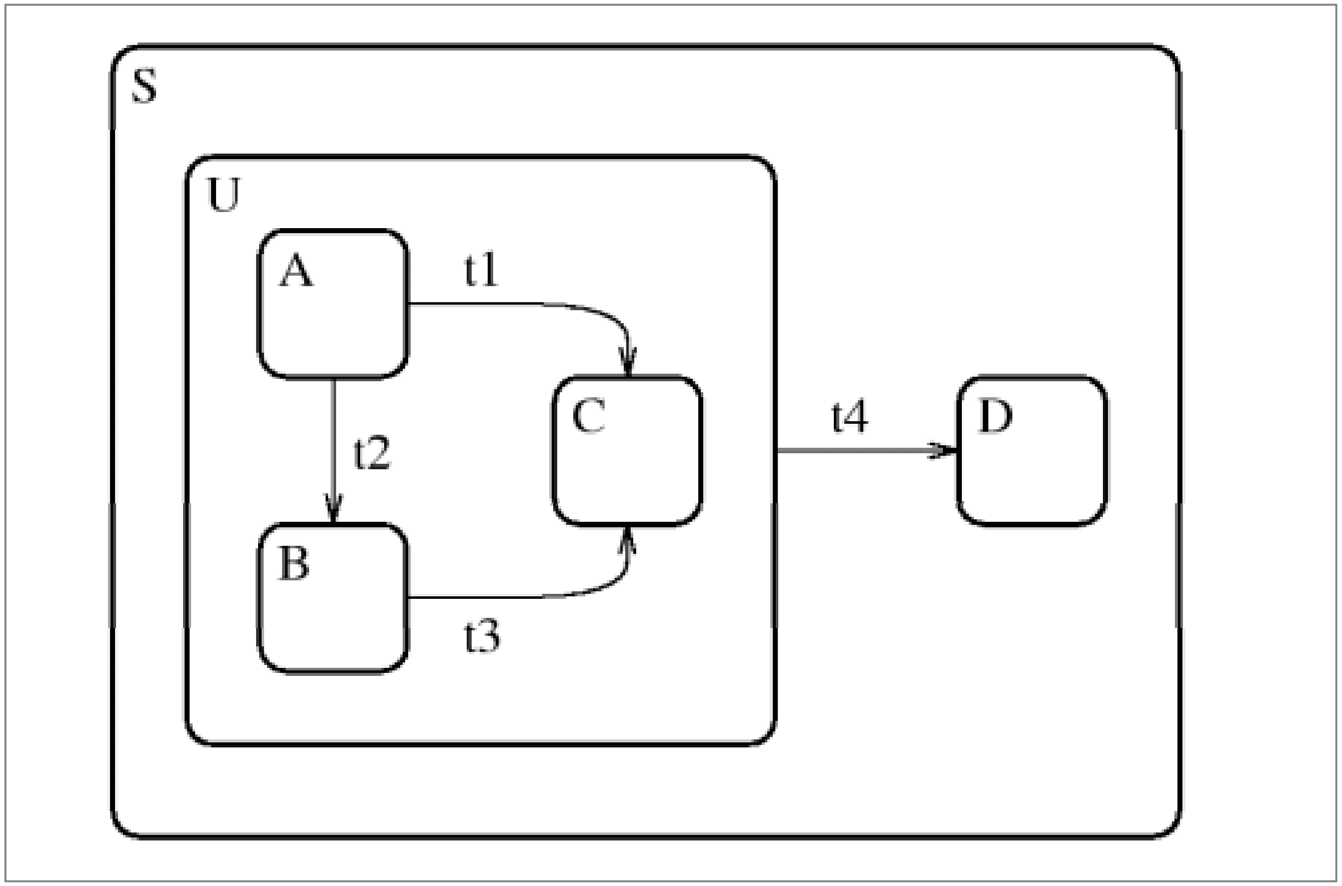


Source [1]

Conflicting Transitions

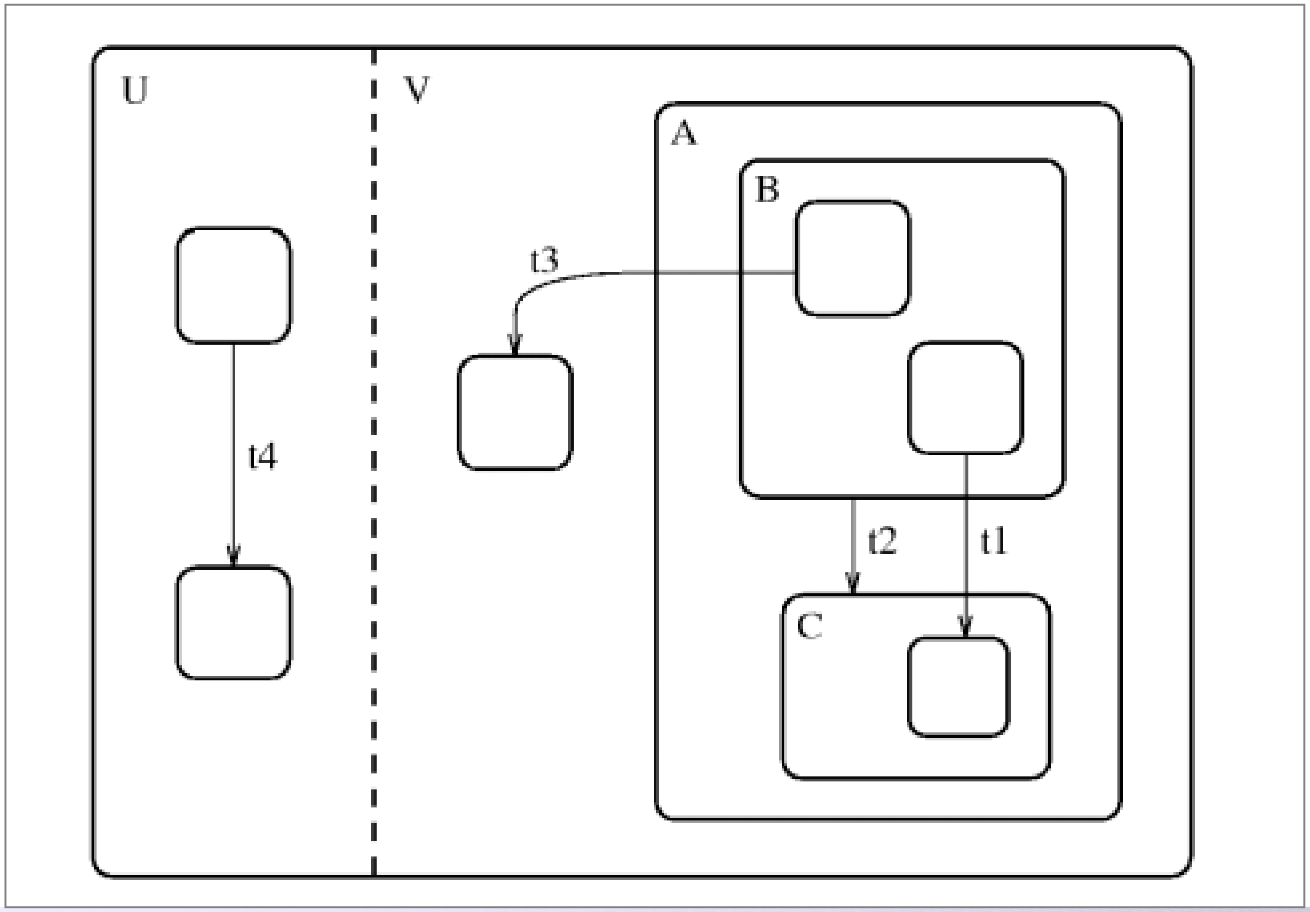
- Two CTs are in conflict if there is some common state that would be exited if any one of them were to be taken.
- Let t_x and t_y be two conflicting transitions and let S_x and S_y be their scopes, respectively. If one of the scopes is an ancestor of the other in the state hierarchy, priority is given to the transition whose scope is higher in the hierarchy. If S_x=S_y, non-determinism occurs.
- An SR defined in state S is in conflict with all the CTs that exit S or one of S's ancestors.
- CTs have higher priority than SRs, since if a state S is exited as a result of some CT, its SRs are not executed.

Examples for Conflicting Transitions - 1



Source [1]

Examples for Conflicting Transitions - 2



Source [1]

The Step Algorithm (1)

- Inputs:
  - state of the system:
    - list of states the system is currently in
    - list of activities currently active
    - current values of conditions and data-items
    - list of events generated internally in the previous step
    - list of scheduled actions and their time for execution
    - list of timeout events and their time for occurrence
    - information on the history of states
  - current time
  - list of external changes presented by the environment since last step.
- Output: New system state.

The Step Algorithm (2)

```
Step preparation:
   Add the external events to the list of internally generated events.
   Execute all the actions implied by the external changes (e.g. changing
   the values of data items, conditions, activities, but not states).
   For each pair (a,t) in the list of scheduled actions do
        if (t <= current_time)
                then carry out a and remove (a,t) from the list
   For each pair (E,t) in the timeout event list, with E=tm(e,d) do
        if e is generated
        then t:=current_time + d;
        else if t <= current_time
                then generate E and set t=infty.
```

The Step Algorithm (3)

Compute the Contents of the Step (CTs and SRs to be executed are marked):

1. Compute the set of enabled CTs.
2. Remove from this set all the CTs that are in conflict with an enabled CT of higher priority.
3. Split the set of enabled CTs into maximal nonconflicting sets.
4. For each set of CTs, compute the set of enabled SRs defined in states that are currently active and are not being exited by any of the CTs in the set.
5. If there are no enabled CTs or SRs

```
    then step is empty
    else if stage 3. resulted in a single set
            then this constitutes the step
            else we have non-determinism and any one of the sets
    can be chosen as step.
```

The Step Algorithm (4)

Execute the CTs and SRs:

Let EN be the set of enabled CTs and SRs.

- For each SR X in EN, execute the action associated with X.
- For each CT X in EN, let Sx and Sn ben the sets of states exited and entered by X, respectively:
  - Update the history of all the parents of states in Sx
  - Delete the states in Sx from the list of states in which the system resides
  - Execute the actions associated with exiting the states in Sx
  - Execute the actions of X
  - Execute the actions associated with entering the states in Sn
  - Add to the list of states in which the system resides all the states in Sn.

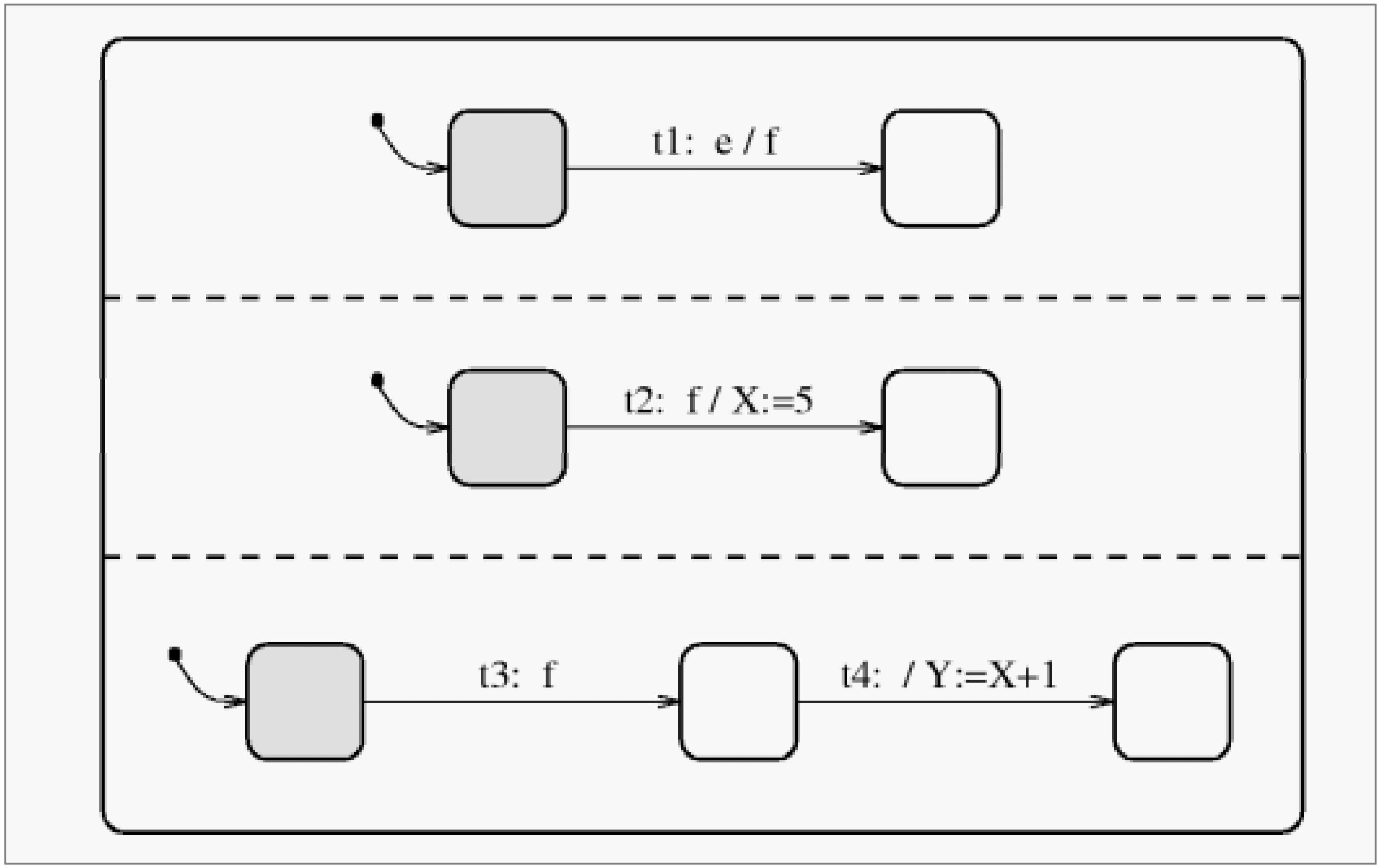
Models of Time

- When is the internal clock advanced relative to the execution of steps?
- How long do steps take in terms of the clock?
- Models of time supported by STATEMATE:
  - synchroneous: system executes a single step every time unit, reacting to all the external changes that occur in the one time unit elapsed since the completion of the previous step.
  - asnchroneous: system reacts whenever an external change occurs, allowing for several external changes to occur simultaneously and allowing several steps to take place within a single point in time (super-step). However, while the steps in a super-step are assumed to take zero time they are considered to be executed in order.
- In both models: execution of step viewed as taking zero time.

Racing Conditions

- Racing condition: value of an element is modified more than once, or is both modified and used at a single point in time.
- Due to super-steps, racing problems can arise also between transitions or actions that are executed in different steps.
- In each step and super-step, several transitions may be enabled. A race situation is one in which, were the enabled transitions executed in a different order (yet a legal one according to the enabling order) different results are obtained in one or more of the data items or conditions.

Racing Condition - Example



Source [1]

Multiple statecharts

- Multiple active statecharts are treated like orthogonal components, except that when one of its statecharts becomes non-active the other chart continues to be active.
- In the asynchroneous time model the steps in multiple statecharts are carried out in synchronization.
- In the synchroneous time model within a chart all the orthogonal components remain synchronized but each chart is synchronized with its own clock and not necessarily with the other charts.