Section 8. The Basic Step Algorithm

- Inputs
  - The status of the system
  - The current time
  - A list of external changes presented by the environment since the last step

  Comments
  - Scheduled action appears in the list as *(a, next-a)*
    - *a*, appeared in an action expression of the form *sc(a,d)* that has already been carried out
    - *next-a*, is the time at which the scheduled a is to be executed
  - Timeout event *E* of the form *tm(e,d)* appears as *(E, next-E)*
- Outputs
  - A new system status

Inputs

- A list of states in which the system currently resides
- A list of activities that are currently active
- Current values of conditions and data-items
- A list of events that were generated internally in the previous step
- A list of schedules actions and their time for execution
- A list of timeout events and their time for occurrence
- Relevant information on the history of states

The Algorithm

- Stage 1: Step Preparation
  a. Add the external events to the list of internally generated events
  b. Execute all action implied by external changes
  c. For each pair (*a*, *next-a*) do the following:

     $$\text{if } next-a \le current\_time$$

     - Then carry out *a* and remove (*a*, *next-a*) from the list
  d. For each pair (*E*, *next-E*) in the timeout even list, with *E* = *tm*(e,*d*) do the following:

```
if e is generated
then set $next-E := current\_time + d$;
$else\ if\ next-E \le current\_time$ then
generate $E$ and set $next-E := \infty$
```

The Algorithm – cont'd

- Stage 2: Compute the Contents of the Step
  a. Compute the set of enabled CTs
  b. Remove from this set all the CTs that are in conflict with an enabled CT of higher priority
  c. Split the set of enabled CTs into maximal non-conflicting sets
  d. 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 hte CTs in the set
  e. If there are no enabled CTs or SRs, then the step is empty.
     - Else if state (c) above resulted in a single set then the set constitutes the step.
     - Else state (c) produced more than one set and we have nondeterminism

Example of Non-conflicting Set

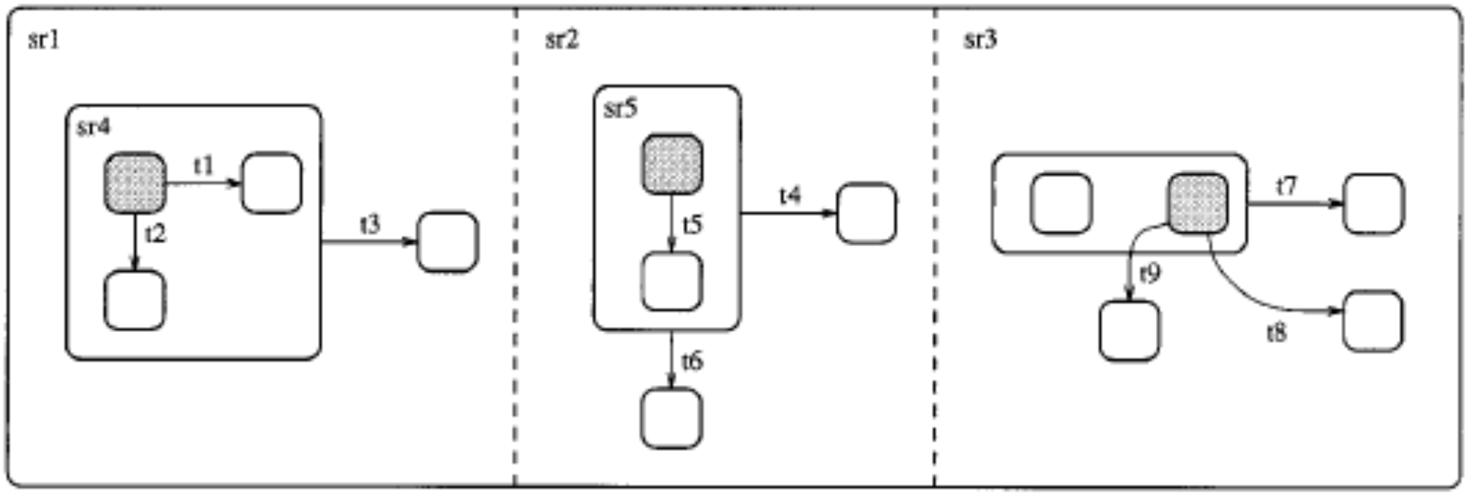


Figure 22.

- Various *sr* denote static reactions associated with the corresponding states
- t3 has higher priority than t1 and t2; t4 and t6 have higher priority than t5; and the three transitions in the right-hand component have the same priority.

$\{t3,\ t4,\ t7,\ sr1,\ sr2,\ sr3\}$
$\{t3,\ t4,\ t8,\ sr1,\ sr2,\ sr3\}$
$\{t3,\ t4,\ t9,\ sr1,\ sr2,\ sr3\}$
$\{t3,\ t6,\ t7,\ sr1,\ sr2,\ sr3\}$
$\{t3,\ t6,\ t8,\ sr1,\ sr2,\ sr3\}$
$\{t3,\ t6,\ t9,\ sr1,\ sr2,\ sr3\}$

The Algorithm – cont'd

- Stage 3: Execute the CTs and SRs . *Let EN be the set of enabled CTs and SRs computing in 2*
  a. For each SR X in EN, execute the action associated with X
  b. For each CT X in EN, Let *Sx* and *Sn* be the sets of states exited and entered by X, respectively;
- Notes and comments
  - In order to implement the semantics of a step, assignments are carried out in 2 stages.
  - **write-write racing**, the order of action execution can affect the results of the step

b.

- 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 action in *X*
- Execute the actions associated with entering the states in *Sn*
- Add to the list of states in which the system resides all states in *Sn*

Section 9. Two Models of Time

- when is the internal clock advanced relative to the execution of steps, and how long do steps take in terms of the clock?
- STATEMATE Supports:
  - Synchronous time model
  - Asynchronous time model
    - **Superstep:** several steps that take place within a single point in time under asynchronous time model
  - Execution of a step may be viewed as taking zero time as far as the environment is concerned (no external changes have effect during execution of step)

Section 9. Two Models of Time: cont'd

- synchronous fits models that are highly synchronous assuming that the previous step was executed at time t, issuing a GO command during a STATEMATE simulation works as follows:

  execute all external changes reported since completion of the previous step;

  increment the clock by one time-unit;

  execute all timeout events and scheduled actions whose due time falls inside $(t, t + 1]$ (that is, the interval that excludes t but includes $t + 1$);

  execute one step.

- Asynchronous time model
  - The simulator's operator or environment must advance time explicitly. Several GO commands for this. For example GO-Repeat:

  execute all external changes reported since completion of the previous step;

  execute all timeout events and scheduled actions whose time is due (up to and including the current time);

  repeatedly execute one step until the system is in a stable state (i.e., there are no generated events and no enabled CTs or SRs).

GO-REPEAT

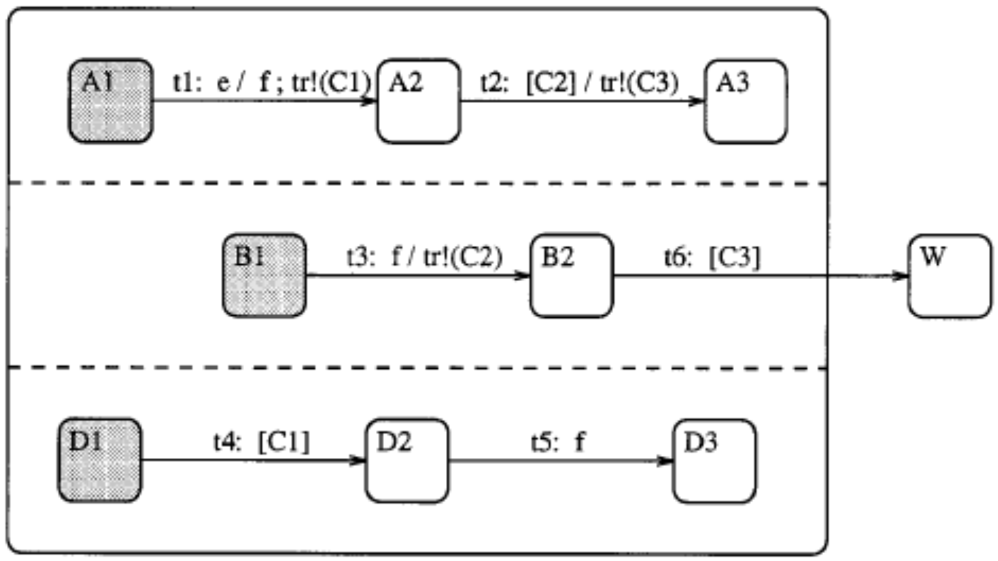


Figure 24.

* Assume C1, C2 and C3 are false and event e is generated

(1) transition $t1$ is taken, leading to the basic configuration $\{A2, B1, D1\}$, making $C1$ true and generating f;

(2) transitions $t3$ and $t4$ are taken, leading to the basic configuration $\{A2, B2, D2\}$, and making $C2$ true;

(3) transition $t2$ is taken, leading to the basic configuration $\{A3, B2, D2\}$, and making $C3$ true. Note that $t5$ is not taken because event f is "alive" only during step (2); in step (3) it no longer exists.

(4) $t6$ is taken, leading to the basic configuration $\{W\}$.

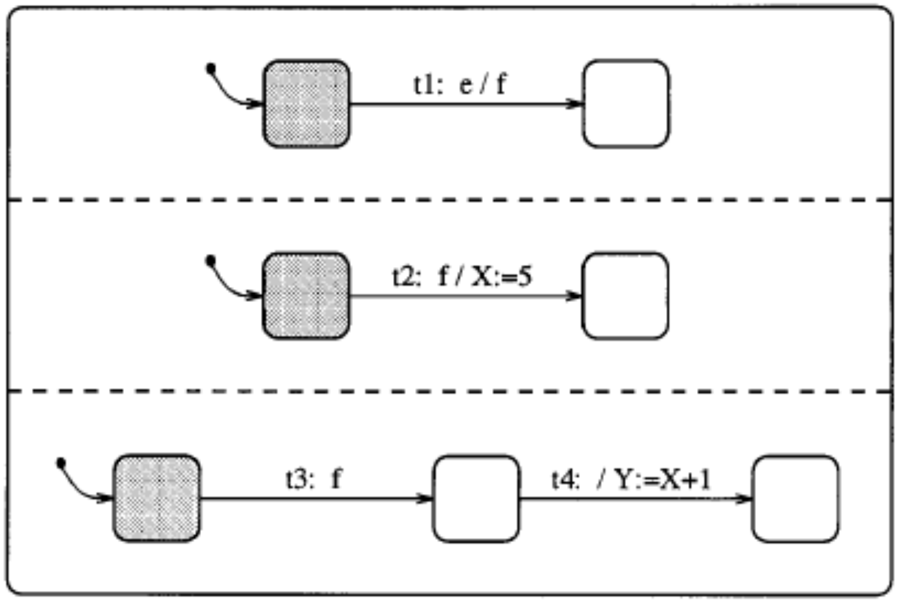
Go Commands

- GO-REPEAT
- GO-ADVANCE
  - Designed to be used in conjunction with GO-REPEAT, to advance the clock
- GO-STEP
  - Executes one step without advancing time. This commands enables easier debugging of the model
- GO-NEXT
  - Advances the clock to the time of the next timeout event or scheduled action without carrying out a step. Before the time is actually advanced, all the steps that can be executed are executed
- GO-EXTENDED
  - A combination of GO-NEXT and GO-REPEAT, which is intended to execute the next superstep that really does something

- STATEMATE's code generator lets user choose between:
  - **RTL code style,** generation of VHDL or Verilog code (synchronous)
  - **behavioral code,** HDL code generated (asynchronous)
- Code generators generate one style of code (CPU clock and simulated clock).

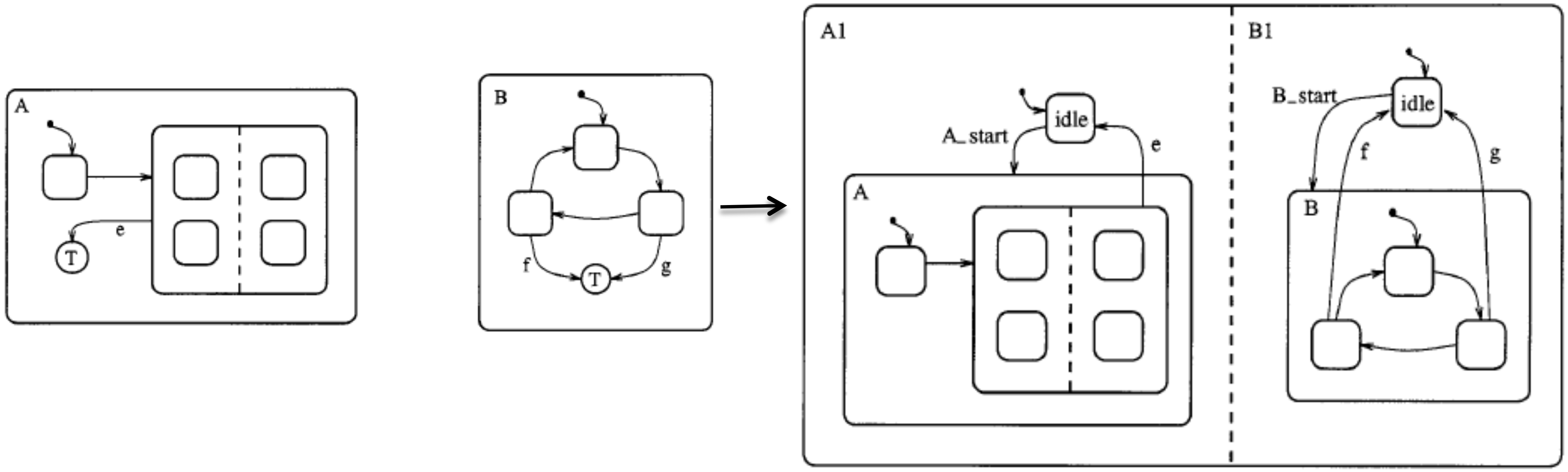
Section 10. Racing Conditions

- Arise when the value of an element is modified more than once or is both modified and used at a single point in time.



Section 11. The Multiple Statecharts

- Multiple active statecharts are treated basically as orthogonal components at the highest level of a single statechart, except when one of the statecharts becomes nonactive



Appendix

- A. Comparison with Other Works
  - RSML Language of Leveson et al. [1995]
    - Does not support history connector
    - Directed comm. Over broadcast comm
    - Allows AND/OR tables for entering condition expressions
    - Does not support disjunctions of trigger events
    - Different terminology
  - Von der Beek [1994]
    - Lists 19 issues relevant to proposals for the semantics of statecharts (see paper)

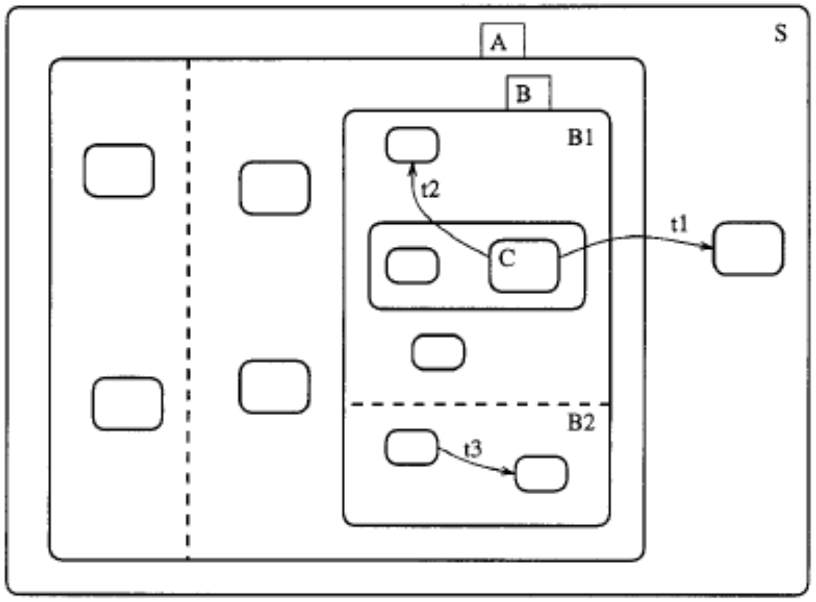
- Perfect-Synchrony Hypothesis
- Self-Triggering, Causality
- Negated Trigger Event
- Effect of Transition Executing is Contradictory to Its Cause
- Interlevel Transition
- State Reference
- Compositional Semantics, Self-Termination
- Operational versus Denotational Semantics
- Durability of Events
- Parallel Execution of Transitions
- Transition Refinement
- Multiple Entering or Exiting of an Instantaneous State
- Infinite Sequence of Transition Executions at an Instant of Time
- Determinism

Appendix

- B. Combinational Assignments
  - Supported by both HW and SW of STATEMATE
  - Associated with an entire chart
  - Viewed as continuous implementation of a function
  - When and how CAs are executed:
    - *When?* During any step in which an operand is changed, either externally or internally, all the relevant CAs are executed
    - Modified Section 8 to include stage 4 to execute all CAs

Appendix

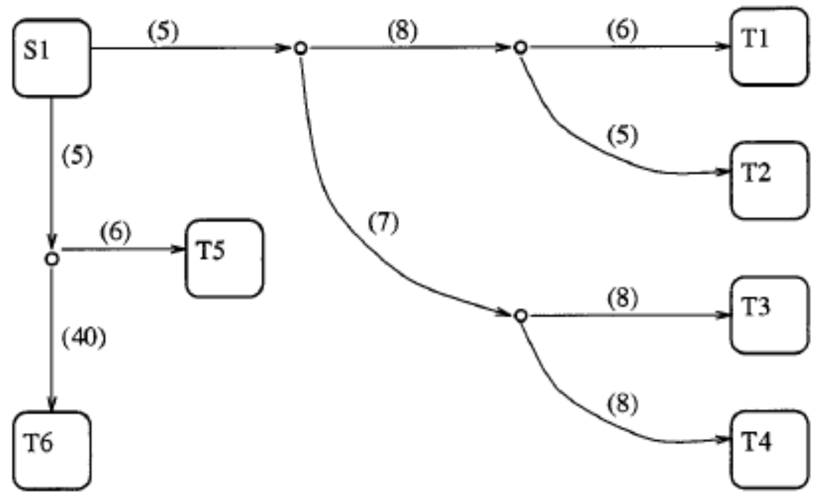
- Priority of Transitions
  - **Structural Priority** (SP): Main criterion for determining the overall priority of a transition. SP is defined for initial CTs only. The determinant of an initial CT *tr* is defined by its scope



Appendix

- Transitions whose determinants are neither identical nor mutually ancestoral are never in conflict. Higher determinant of a CT in the state hierarchy, the higher its SP.
- **Priority number** (PN), is an integer one can affix to transition segments. The priority of a transition segment is compared with the priorities of other segments with the same source or the same priority determinant.

Appendix



	$T2$	$T3$	$T4$	$T5$	$T6$
$T1$	$>$	$>$	$>$	$=$	$=$
$T2$		$>$	$>$	$=$	$=$
$T3$			$=$	$=$	$=$
$T4$				$=$	$=$
$T5$					$<$

- Joint or Fork connectors: view each combination of segments incident to the connector as a single virtual segment
  - Definition is carried out recursively, until all the fork and joint connectors have been removed