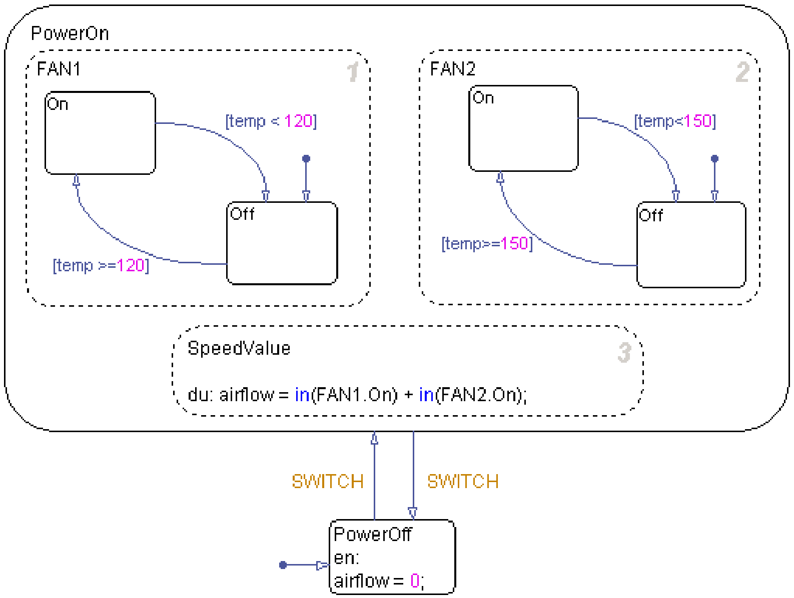
Semantics of Statecharts

Michael Whalen

Program Director
University of Minnesota Software Engineering Center

Statecharts

- Popular notation for implementing complex state machines
- Proposed by Harel in 1987
- Statecharts =
  state diagrams +
  depth (hierarchy) +
  orthogonality (parallelism)
  broadcast-communication



Statecharts (my history)

- 1997-99: Worked on simulation and translation tools for the Requirements State Machine Language (RSML)
- 1999-2002: Developed the semantics of the Requirements State Machine Language without Events ($RSML^{-e}$) – Masters thesis
- 2002-2004: Created Compiler for $RSML^{-e}$ to SIMP (fully-specified subset of C) and proved its correctness – PhD thesis
  - **Vowed to quit working on Statecharts ☺**
- 2007-2009: Created formal analysis and compiler tools for Simulink Stateflow & worked on formal semantics
- 2010 - ??? Working with NASA Ames & Vanderbilt U on parameterized analysis of Statecharts dialects

Statecharts Formalisms

- Classical Statecharts (STATEMATE)
- Rhapsody Statecharts
- UML Statecharts
- MATLAB Stateflow
- SyncCharts (ESTEREL)
- Requirements State Machine Language (RSML)
- …about 100 other variants

What happens when event ‘e’ occurs?

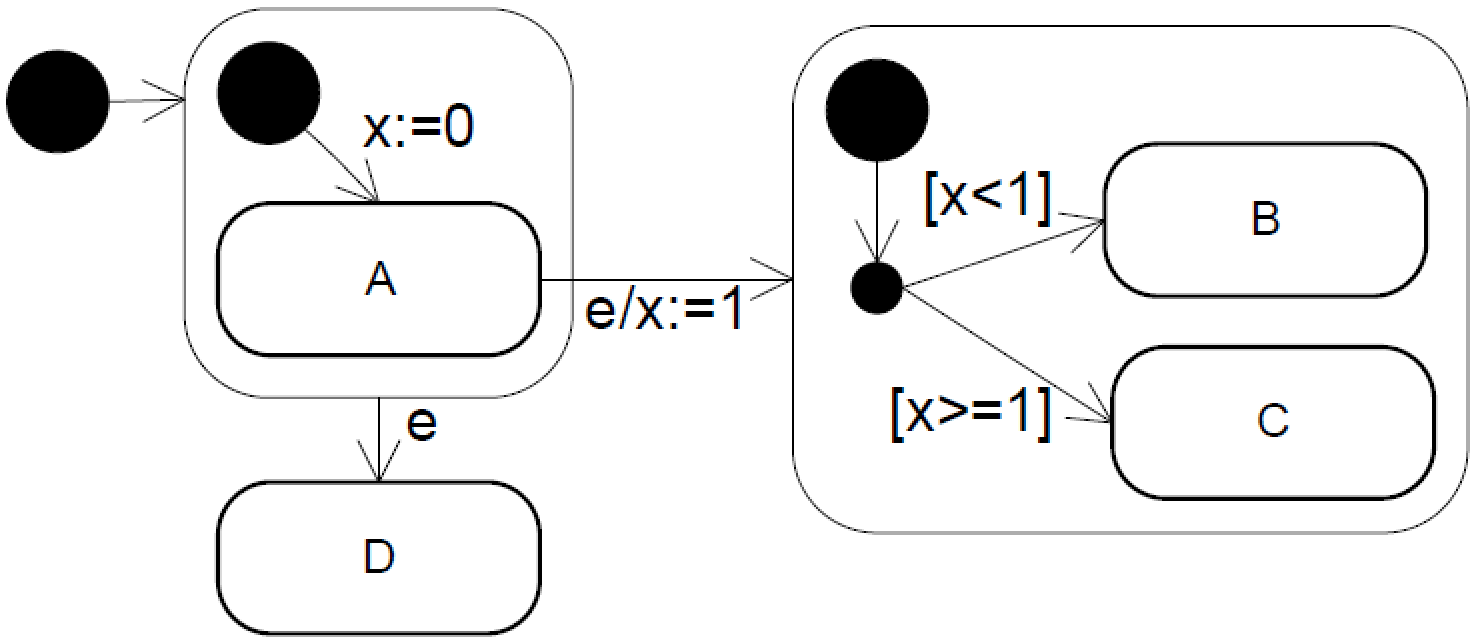


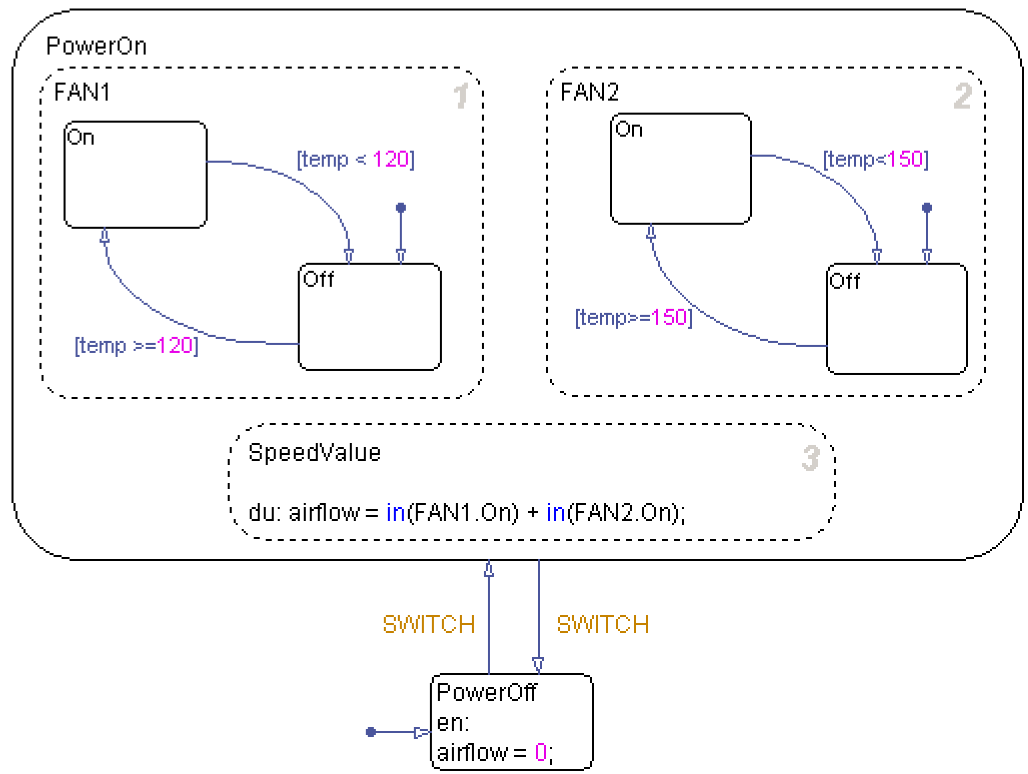
Figure from: Michelle L. Crane and Juergen Dingel, UML vs. Classical vs. Rhapsody Statecharts: Not All Models are Created Equal, *Proceedings of MoDELS2005*, Montego Bay, Jamaica, October, 2005

Analyzing Statecharts

- Statecharts are used to design embedded systems
  - Sometimes *safety-critical* embedded software
- Some dialects are *underspecified*
  - UML & Rhapsody: parallel evaluation, conflicting active transitions, event ordering all underspecified
- Large projects use multiple dialects
  - NASA Constellation project: Rhapsody, UML, and Stateflow
  - Engineers familiar with different dialects read same diagram differently!
- Want to determine: when are charts "safe"?
  - Within a dialect
  - Across dialects

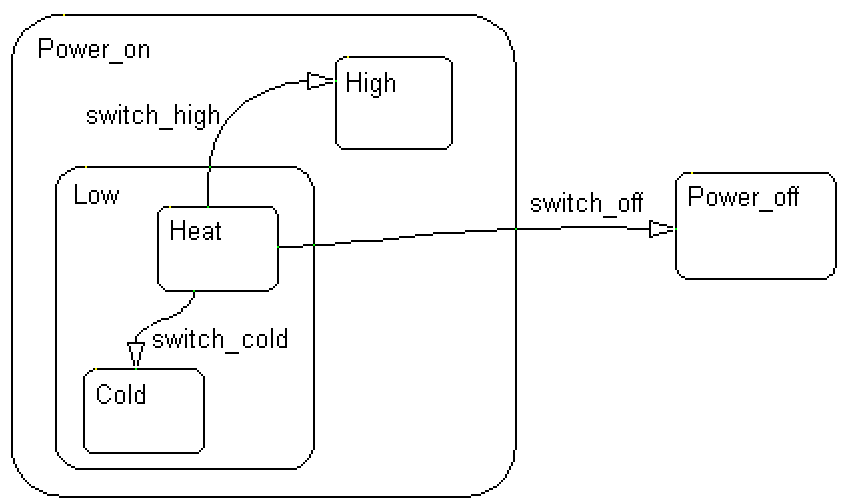
Syntax of Statecharts

- States:
  - AND (parallel)
  - OR (hierarchical)
- Transitions
  - Event-triggered
  - Conditional
- Events
  - "Basic"
  - Valued
- Non-graphical variables



Syntax of Statecharts

- Transition labels of the form:
  Event [condition] / action
  - Possible to omit one or more components
- Boundary Crossing Transitions



Syntax of Statecharts

- Fork and Join: mechanisms for simplifying complex or redundant transitions

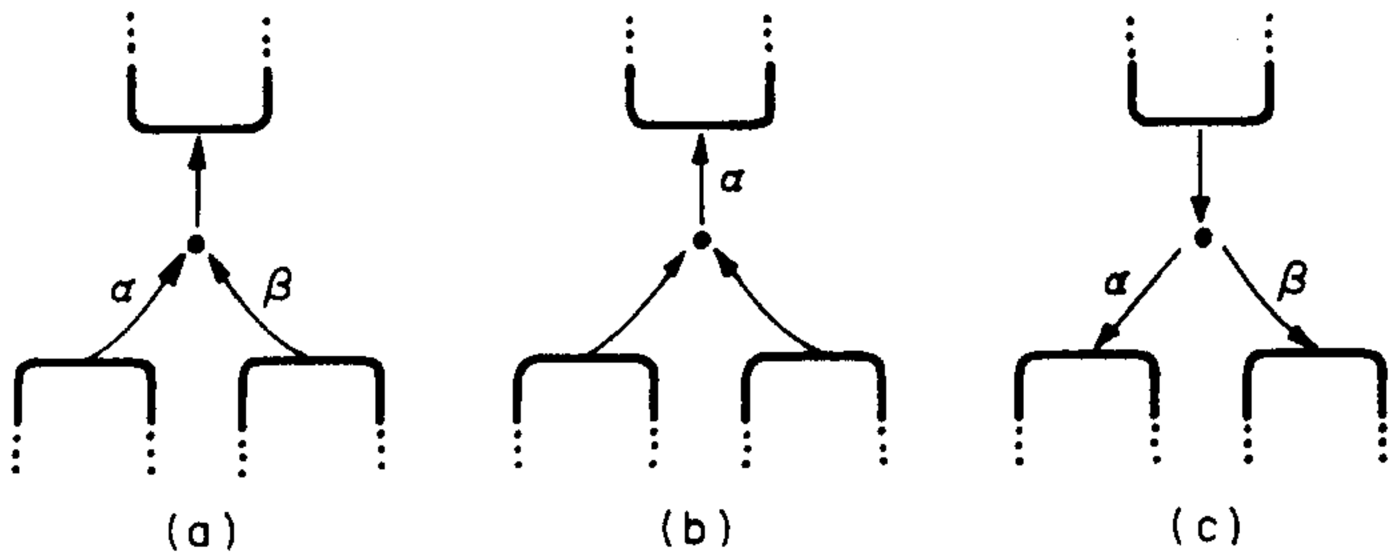


Figure from: David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming 8*, 1987.

Syntax of Statecharts

- History Junctions
  - Allow restoration of child state
  - Can either be "single level" or transitive

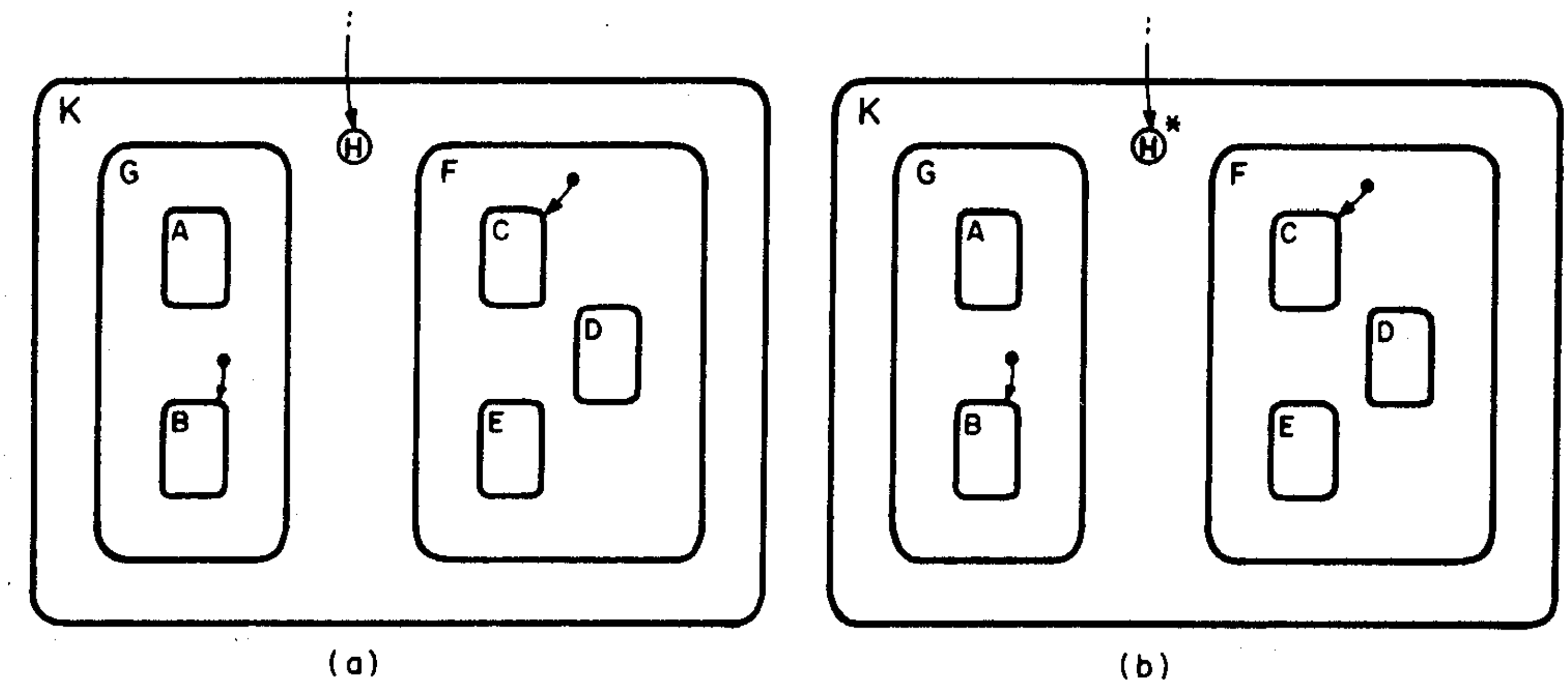


Figure from: David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming 8*, 1987.

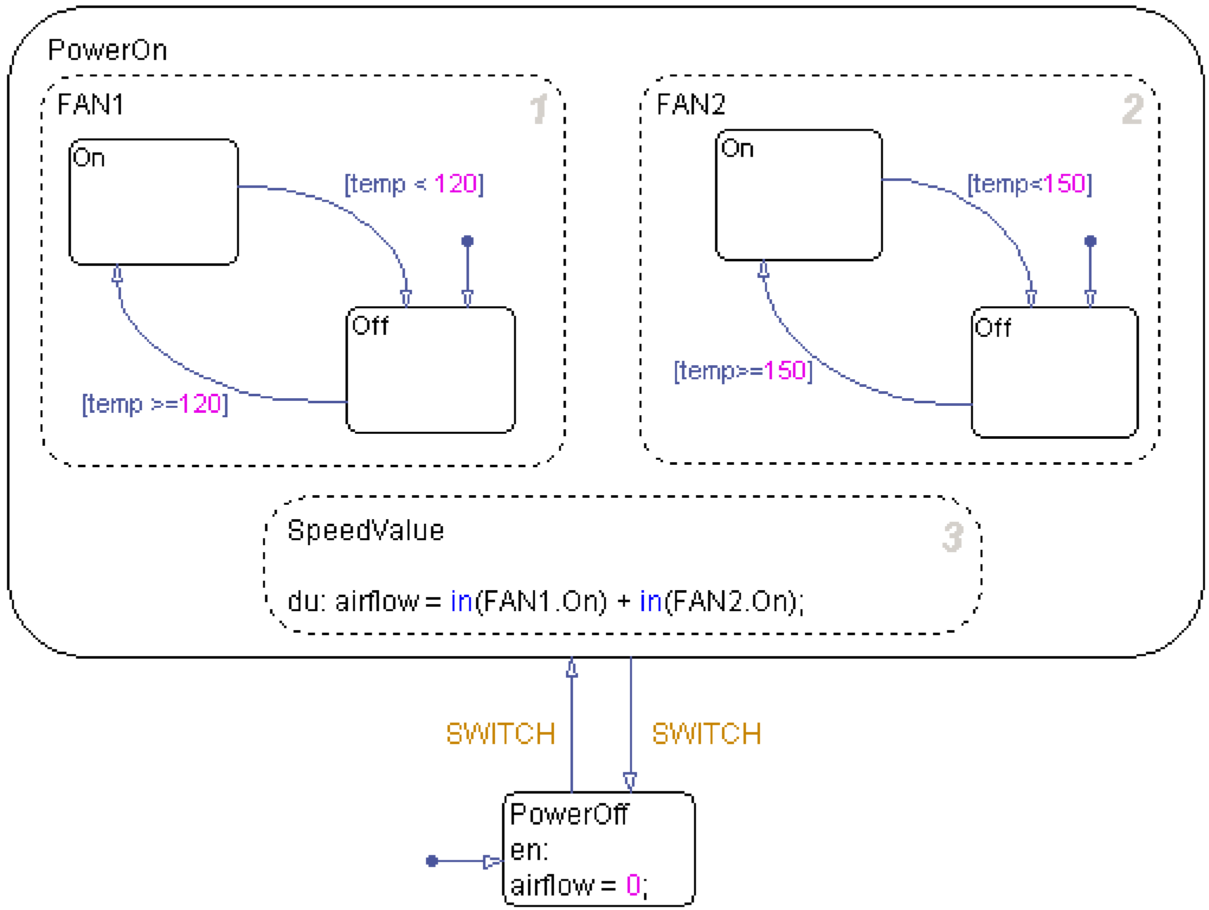
Semantics Sketch

- All semantics have a notion of a *step*
  - An external event causes a chart to evaluate
    - Event can be implicit (time tick)
    - External event becomes initial *active* event
- Transitions are evaluated
  - Transition is *enabled* if
    - The source state of the transition is occupied
    - The triggering event of the transition (if any) matches an active event
    - The condition on the transition (if any) evaluates to true

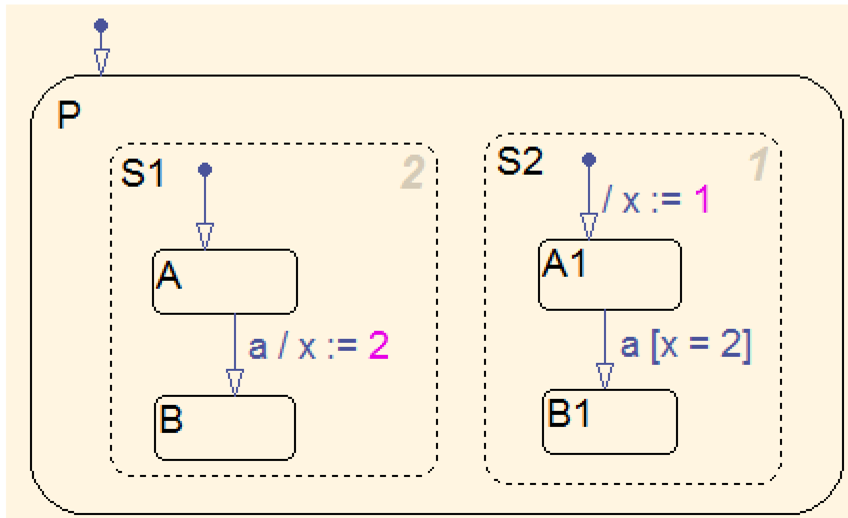
Semantics Sketch

- Subset of enabled transitions *fire*
  - Change from source to destination state
  - May generate actions including additional events
    - Semantics of event propagation differ between Statecharts dialects
- Step evaluation is completed when all events have been processed

Example


PowerOn
FAN1
1
On
[temp < 120]
Off
[temp >=120]
FAN2
2
On
[temp<150]
Off
[temp>=150]
SpeedValue
3
du: airflow = in(FAN1.On) + in(FAN2.On);
SWITCH
SWITCH
PowerOff
en:
airflow = 0;

Parallel State Evaluation



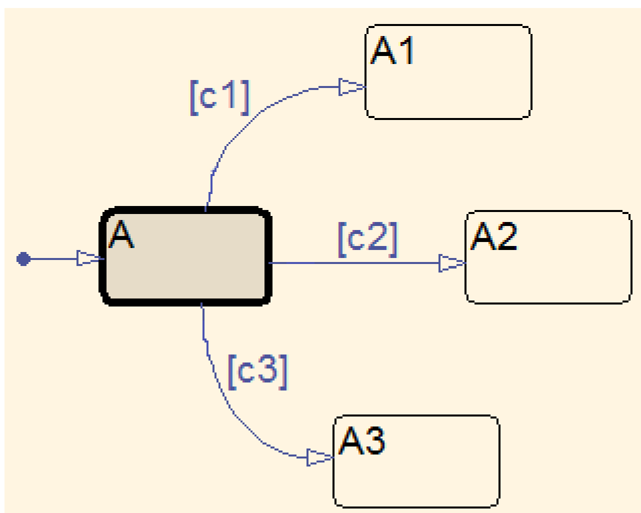
UML, Rhaspody, STATEMATE: No order specified by semantics; semantics are **tool dependent** in case of conflicts

Stateflow: deterministic user-specified sequential order

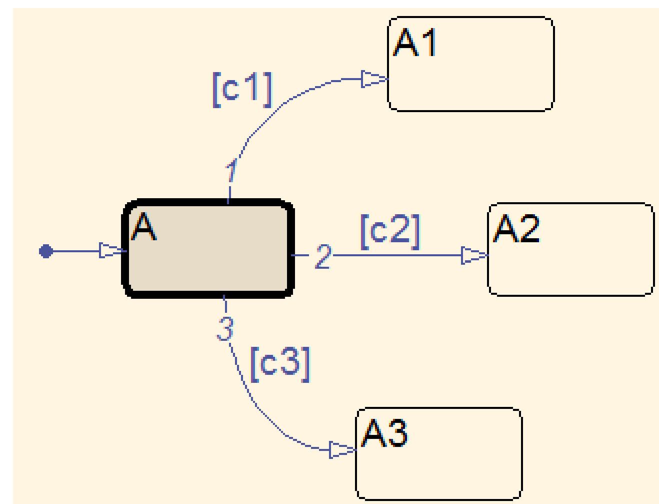
SyncCharts: semantics-determined partial order. Variables cannot be shared between parallel state machines, so this model would be rejected

Simultaneously Enabled Transitions

- Some dialects do not define an ordering on transitions at a particular level of hierarchy



UML, Rhaspody, STATEMATE: No order specified by semantics; semantics are **tool dependent** in case of conflicts



Stateflow, SyncCharts: deterministic user-specified sequential order

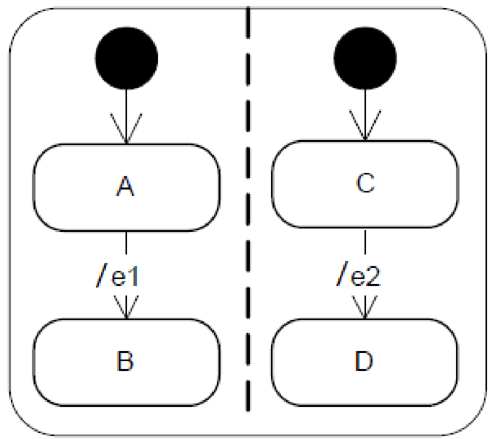
DISCREPANCIES BETWEEN DIALECTS

Event Processing

- Most Statecharts semantics split step into *microsteps*
  - Each microstep handles one round of event processing
  - If current round generates new events via transition actions, re-run chart until no further events are generated
- Stateflow uses *function call* semantics
  - Event action interrupts current chart processing and re-runs chart on generated event

Simultaneous Events

- Can multiple simultaneous events occur?



Classical, SyncCharts: multiple events to be "true" at the same instant

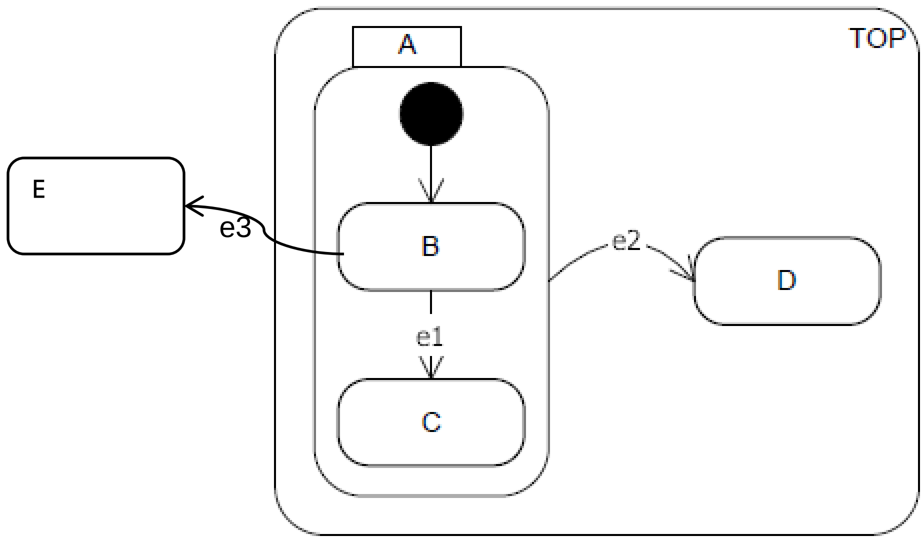
UML / Rhapsody: queue up events (in arbitrary order) and execute one at a time.

Stateflow: only one event due to function call semantics

Figure from: Michelle L. Crane and Juergen Dingel, UML vs. Classical vs. Rhapsody Statecharts: Not All Models are Created Equal, *Proceedings of MoDELS2005*, Montego Bay, Jamaica, October, 2005

Transition Ordering

- How does semantics choose between simultaneously enabled transitions?

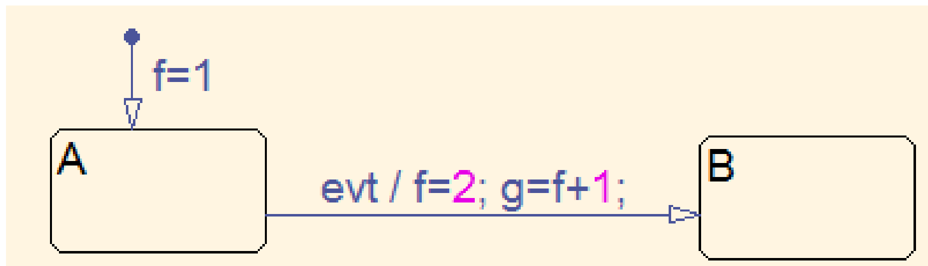


Classical: transitions with highest *scope* have highest priority. Scope is largest state that contains portion of transition arc (go to state E)

UML / Rhapsody: transitions in smallest substate have priority (go to state C).

Stateflow, SyncCharts: evaluation is “top down” based on transition source (go to state D)

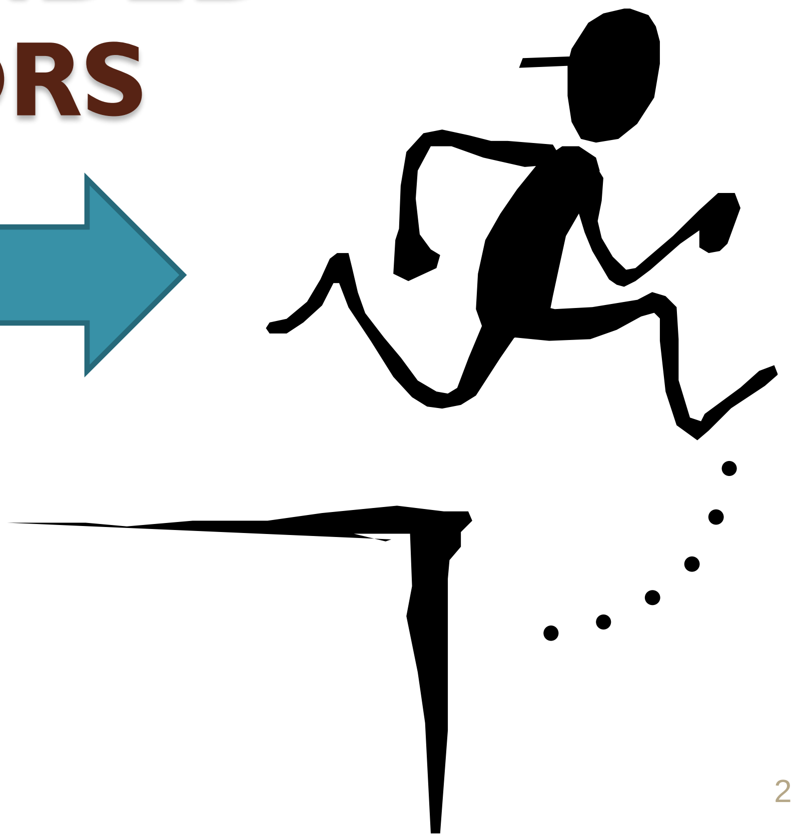
Execution of Actions



Classical: Assignment actions within a microstep are considered simultaneous. Transition result: f = 2, g = 2

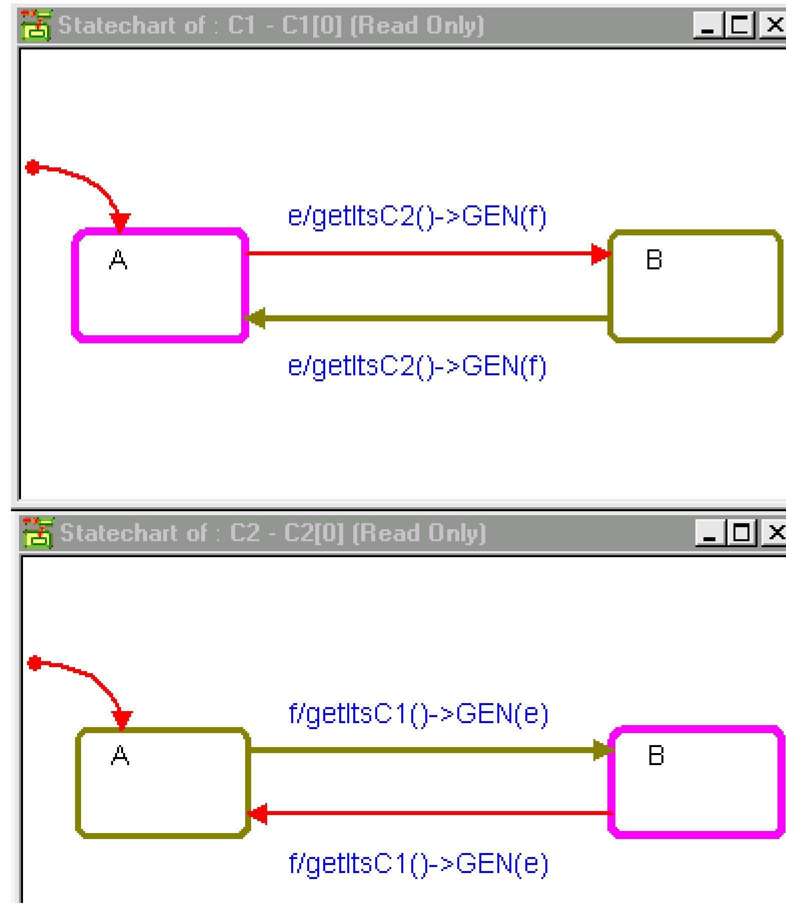
UML, Rhapsody, Stateflow, Esterel: Assignment actions are sequential. Transition result : f = 2, g = 3.

UNINTENDED BEHAVIORS



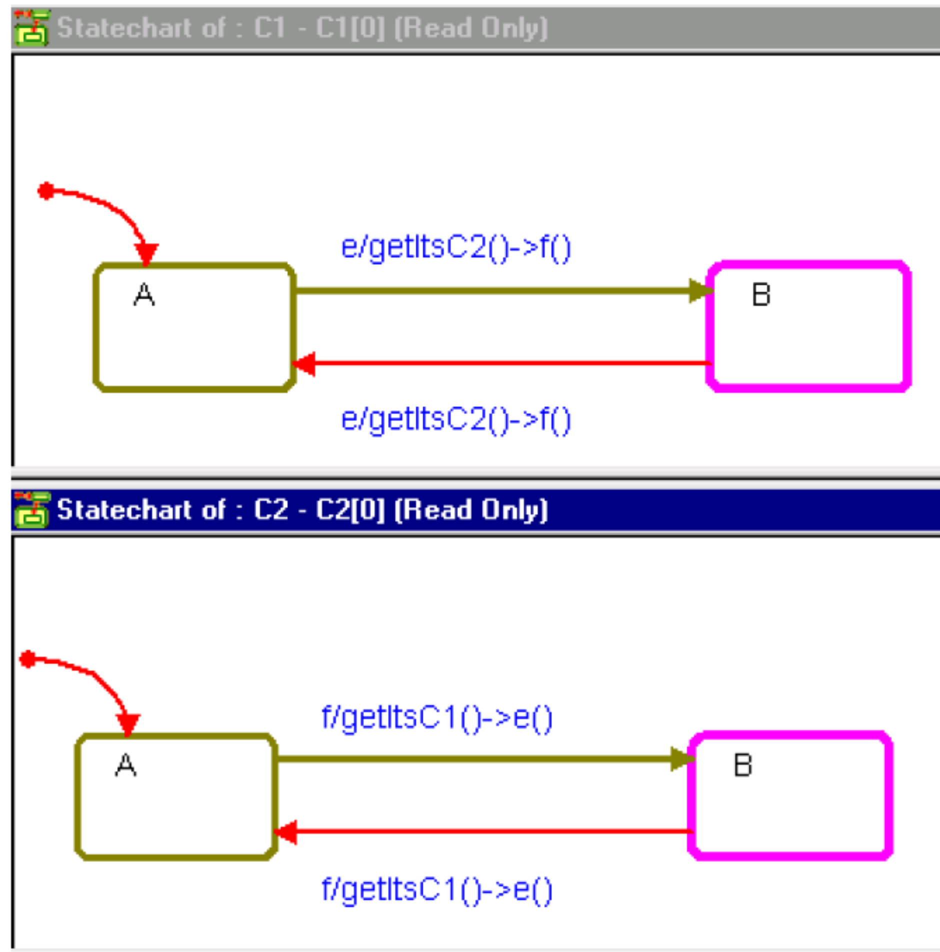
Infinite Loops (Rhapsody)

- Example of GEN leading to infinite loop
- C1 queues message for C2 which queues message for C1 which …

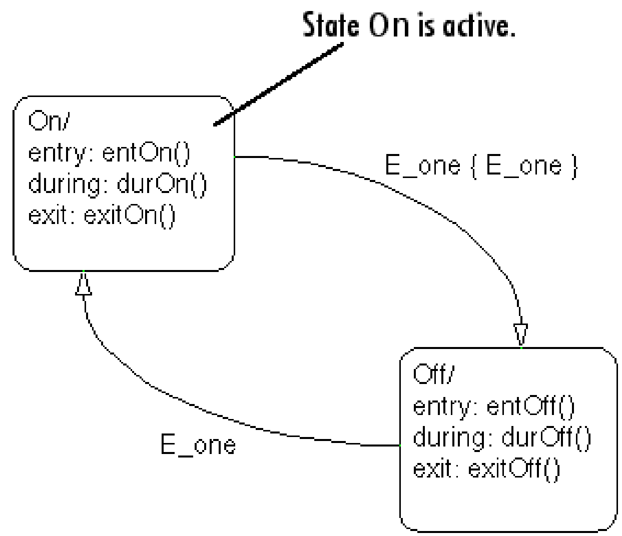


Bad Return Policies (Rhapsody)

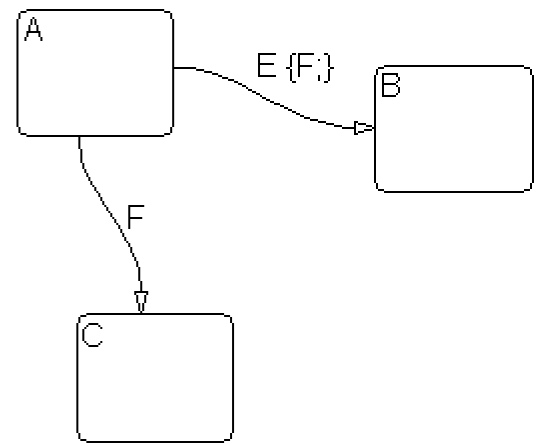
- Trigger Example
- Rhapsody policy: triggered messages received while evaluating a message are dropped.
- So, no infinite loop here.
- Triggers can return values.
  - If trigger is dropped, return value is not defined by Rhapsody semantics.



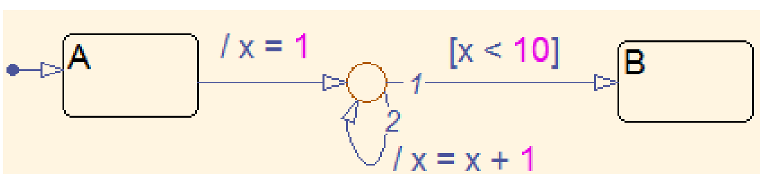
Strange Charts (Stateflow)



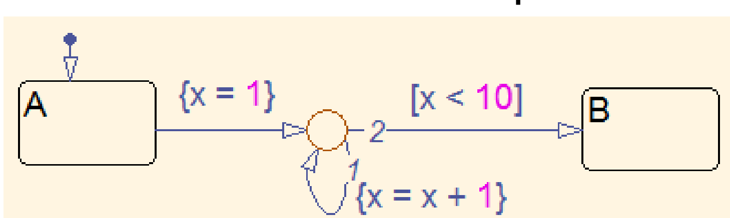
Infinite Event Loop



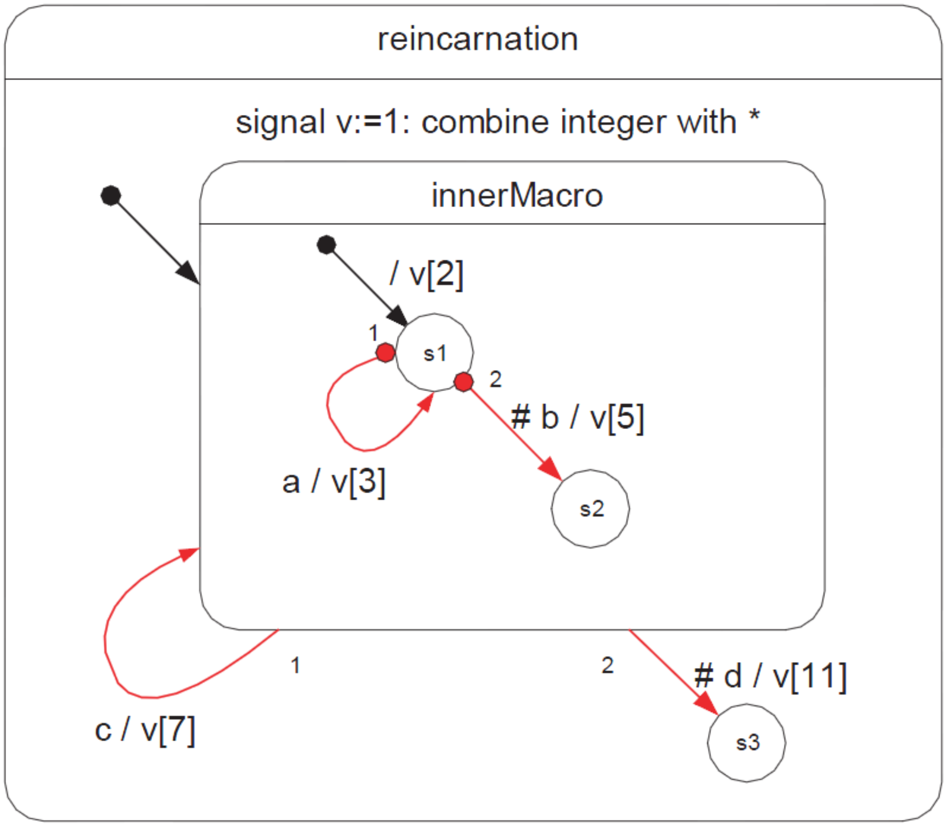
Early Return Logic



Infinite Junction Loops



Multiple Entries (SyncCharts)



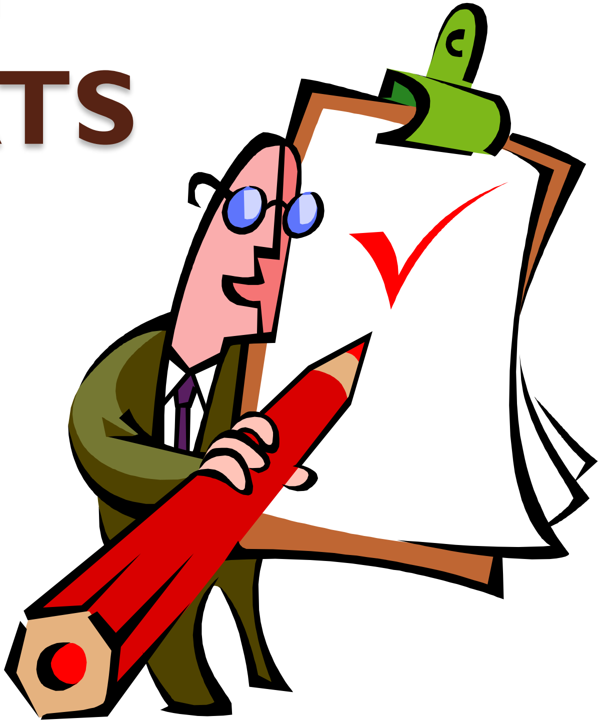
After eval starting in s1:
v = 11,550

1. Start in s1
2. Queue transition ‘c’ (weak abort)
3. Take transition ‘a’ (v=3)
4. Take immediate transition b to s2 (v=15)
5. Take transition ‘c’ (v=105)
6. Re-enter s1 (v=210)
7. Take immediate transition b (v=1050)
8. Take immediate transition d (v=11,550) to state s3

- SyncCharts adds ‘strong abort’ vs. ‘weak abort’ transitions
- Also ‘immediate’ vs ‘delayed’ transitions
- Valued signals can be combined using commutative operator

Figure from: Charles Andre, Computing SyncCharts Reactions, *Electronic Notes in Computer Science Volume 88 (2003)*

ANALYZING STATECHARTS



Stateflow Semantics

- Stateflow User Manual is 1400 pages
- Transition semantics alone is 7 pages of pseudocode
- Two attempts at formalization
  - Gregoire Hamon
    - Operational Semantics [SRI 2003]
      - Large; incomplete
    - Denotational Semantics [Chalmers 2006]
      - Based on continuations
      - Elegant, relatively complete, slightly incorrect
      - Worked with Gregoire to correct errors, complete definition

Stateflow Semantics

- Denotational semantics distills 1400 page manual into 1 ½ pages of formalism
- In: Gregoire Hamon. A Denotational Semantics of Stateflow, *EMSOFT 2006*
- Handful of errors in EMSOFT paper w.r.t. to boundary crossing transitions, transition actions, flowcharts
  - I fixed these and added support for a few remaining issues: history states and early return logic
  - Gregoire and I need to submit this for publication!

Stateflow Semantics

Syntax:

$$\begin{aligned}
&\mathit{Program}\ P ::= (s, [src_0, ..., src_n], I, O, L, K)\\
&\mathit{SrcComp}\ src ::= p : sd \mid j : T \mid f : fd\\
&\mathit{StateDef}\ sd ::= ((a_e, a_d, a_x), (L, K), T, C)\\
&\mathit{FunctionDef}\ fd ::= ((I, L), T)\\
&\mathit{Comp}\ C ::= Or\ (T, [s_0, ...s_n]) \mid And\ ([s_0, ...s_n])\\
&\mathit{Trans}\ t ::= (e, c, (a_c, a_t), d)\\
&\mathit{Dest}\ d ::= p \mid p.j\\
&\mathit{TransLst}\ T ::= \emptyset \mid t.T\\
&\mathit{Path}\ p ::= \emptyset \mid s.p
\end{aligned}$$

Environments:

$$\begin{aligned}
&\mathit{Env}\ \rho ::= (I, O, K, S, V, (SI,SL,SO).L)\\
&\mathit{Kenv}\ \theta ::=\\
&\quad \{\ p_0 : (S[[p_0 : sd_0]]e\ \theta, S[[p_0 : sd_0]]d\ \theta, S[[p_0 : sd_0]]x\ \theta),\\
&\quad\quad ...\\
&\quad\quad p_n : (S[[p_n : sd_n]]e\ \theta, S[[p_n : sd_n]]d\ \theta, S[[p_n : sd_n]]x\ \theta),\\
&\quad\quad p_0.j_0 : T\ [[T_0]]\ \theta\ p_0, ..., p_k.j_k : T\ [[T_k]]\ \theta\ p_k\}
\end{aligned}$$

Stateflow Transition Semantics

τ : *trans* → *kenv* → *env* → *path list* → *k-* → *k+* → *k-* → *event* → *env*

τ [[(*et*, *c*, (*ac*, *at*), *d*)]] θ ρ *P transact complete fail e* =
if (*et* = *e*) ∧ (*B*[[*c*]] ρ) then
let *transact'* =
$\lambda\rho_t$. transact (*A*[[a_t]] θ ρ_t) in
D[[*d*]] θ (*A*[[a_c]] θ ρ) *P transact' complete fail e*
else
fail ρ

T: TransList → KEnv → env → path list → k- → k+ → k- → event → env

T [[Ø]] θ ρ *P transact complete fail e* = *complete* ρ []
T [[*t*.Ø]] θ ρ *P transact complete fail e* = τ [[*t*]] θ *P* ρ *transact complete fail e*
T [[*t*.*t'*.*T*]] θ ρ *P transact complete fail e* =
let *fail'* = λ ρ_f.*T* [[*t'*.*T*]] θ ρ_f *P transact complete fail e* in
τ [[*t*]] θ ρ *P transact complete fail' e*

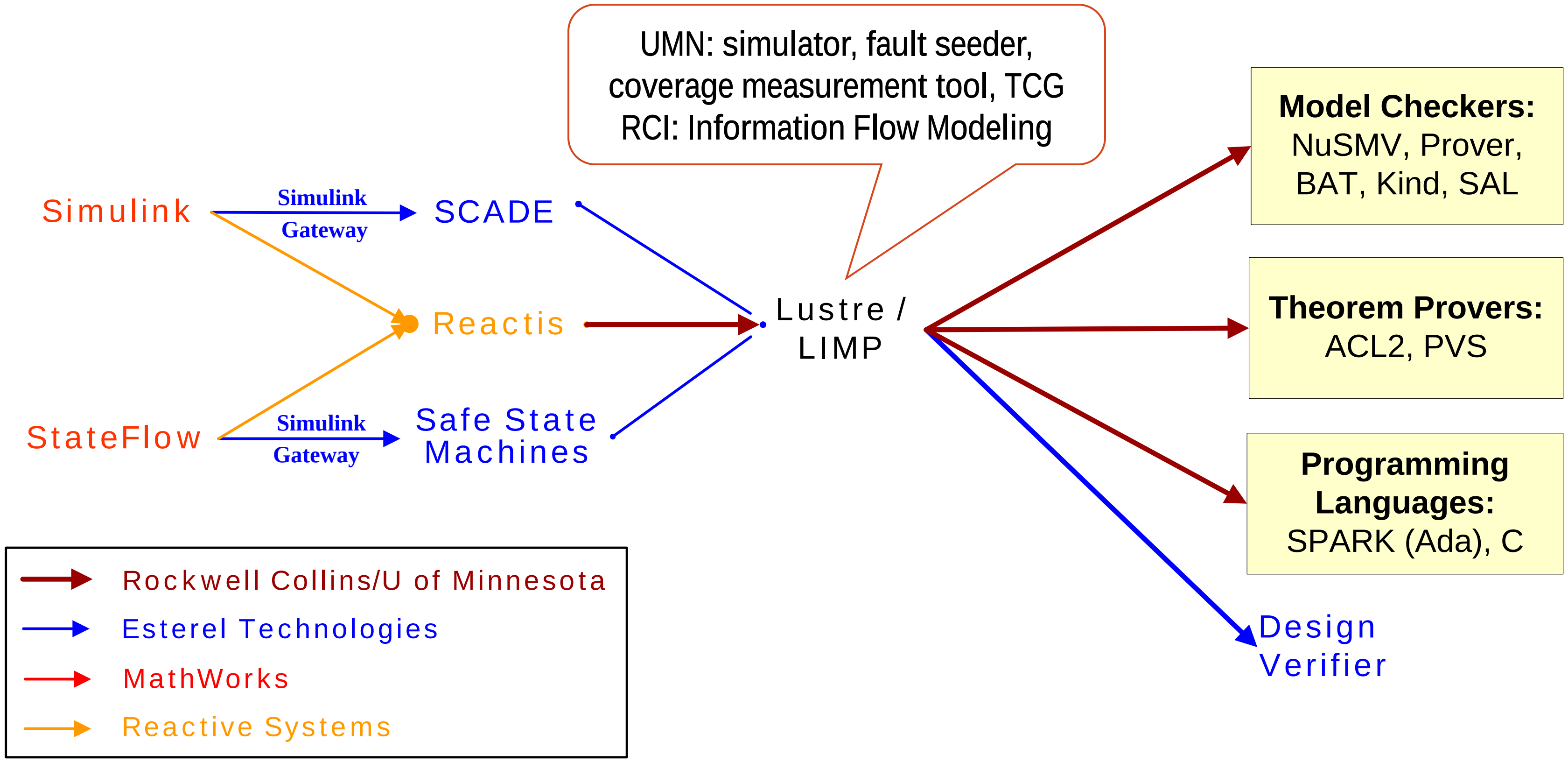
Stateflow Destination / State Semantics

D: Destination → KEnv → env → path list → k- → k+ → k- → event → env
S: StateDef → KEnv → env → P → event → env
open_path: KEnv → env → path list → k- → k- → event → env

$D[[p]]\ \theta\ \rho\ P\ transact\ complete\ fail\ e = success\ transact\ \rho\ P.p$
$D[[j]]\ \theta\ \rho\ P\ transact\ complete\ fail\ e = \theta(j)\ P.p\ \rho\ transact\ complete\ fail\ e$

$S[[p : ((a_e, a_d, a_x), T, C)]]_e\ \theta\ \rho\ P\ e = C[[C]]_e\ \theta\ (A[[a_e]]\ \theta\ (\text{open}\ \rho\ p))\ P\ e$
$S[[p : ((a_e, a_d, a_x), T, C)]]_d\ \theta\ \rho\ e =$
let
during = $\lambda\rho_d\ .\ (A[[a_d]]\ \theta\ \rho_d)$
fail = $\lambda\rho_f\ .\ C[[C]]d\ \theta\ (\text{during}\ \rho_f)\ e$
complete = $\lambda\rho_c.\ \lambda p_c.\ \lambda t_c.$ open_path $\theta\ \rho_c\ p_c\ t_c$ during fail e
transact = id (* identity function *)
in
T [[T]] $\theta\ \rho$ transact complete fail e
end
$S[[p : ((a_e, a_d, a_x), T,C)]]_x\ \theta\ \rho\ P\ e = \text{close}\ p \circ A[[a_x]]\ \theta \circ C[[C]]_x\ \theta\ \rho\ P\ e$

Implementation in Gryphon Tool Family



M. Whalen, D. Greve, L. Wagner, Model Checking Information Flow, In: *Design and Verification of Microprocessor Systems for High-Assurance Applications,* D. Hardin, Ed., Springer, March 2010.

S. Miller, M. Whalen, D. Cofer, Software Model Checking Takes Off, *Communications of the ACM*, February 2010

D. Hardin, D.R. Johnson, L. Wagner, and M. Whalen. Development of Security Software: A High-Assurance Methodology, *ICFEM 2009*, Rio de Janeiro, Brazil, December, 2009.

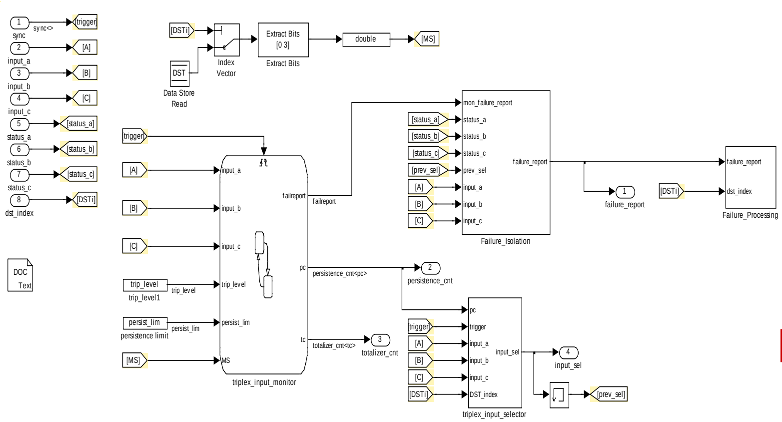
Functional Analysis of Stateflow

CerTA FCS Phase I

- Sponsored by AFRL
  - Wright Patterson VA Directorate
- Compare FM & Testing
  - Testing team & FM team
- Lockheed Martin UAV
  - Adaptive Flight Control System
  - Redundancy Management Logic
  - Modeled in Simulink
  - Translated to NuSMV model checker

	Subsystem/ Blocks	Charts / Transitions / TT Cells	Reachable State Space	Properties
Triplex voter	10 / 96	3 / 35 / 198	$6.0 * 10^{13}$	48
Failure processing	7 / 42	0 / 0 / 0	$2.1 * 10^{4}$	6
Reset manager	6 / 31	2 / 26 / 0	$1.32 * 10^{11}$	8
Totals	23 / 169	5 / 61 / 198	N/A	62

… for each of ten control surfaces



Phase I Results

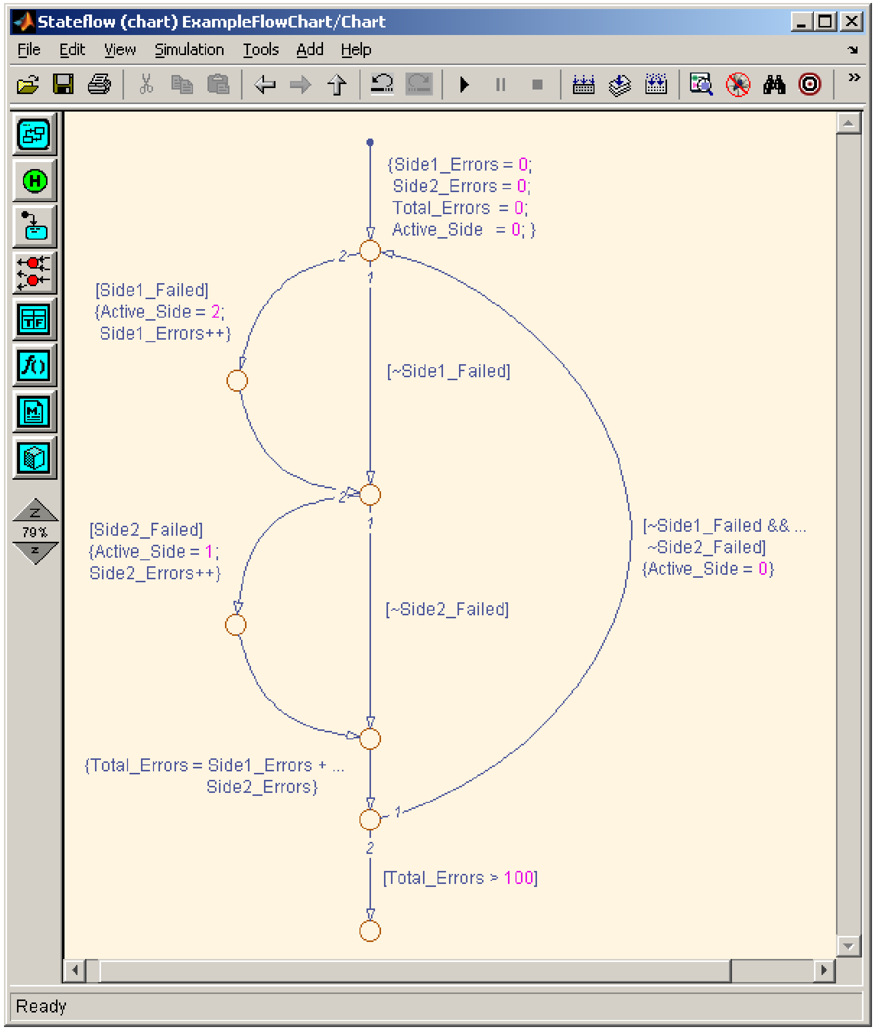
	Effort (% total)	Errors Found
Testing	60%	0
Model-Checking	40%	12

Functional Analysis of Stateflow

CerTA FCS Phase II – Verification of Stateflow Flowcharts

- Stateflow Flowcharts
  - No explicit states
  - Stateflow junctions
  - Cyclic paths
  - Transitions modify local state variables
  - Imperative programming

- Solution
  - Extension to translator to support flowcharts
  - Require a parameter that specifies the maximum times any cycle will be executed
  - This bound becomes property to check



Analysis of RCI State Machine Notation

FCS 5000 Flight Control Mode Logic

Mode Controller A

Converted to Simulink

Translated to NuSMV

6.8×10^{21} Reachable States

Mode Controller B

Example Requirement

Mode A1 => Mode B1

Counterexample Found in Less than Two Minutes

Found 27 Errors

RCI Stateflow analysis

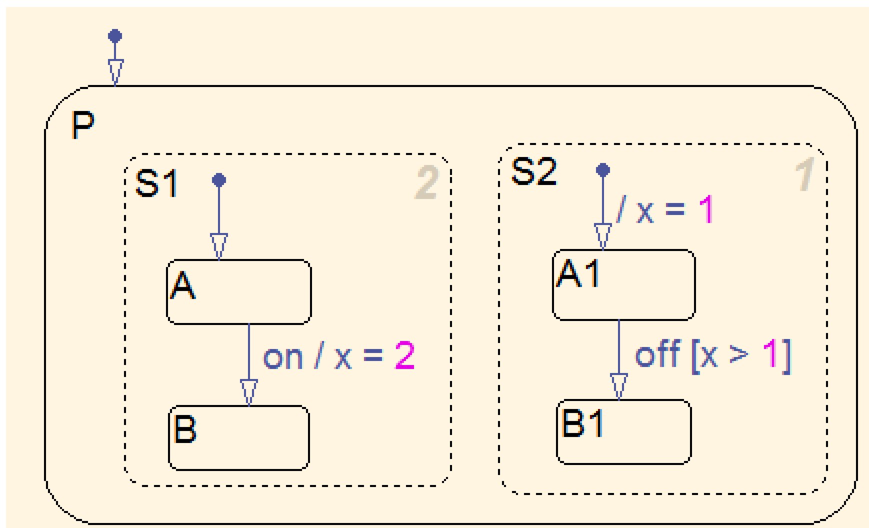
- Focused on *functional analysis*
  - Prove functional and safety requirments of mixed Simulink/Stateflow models
- Based on Stateflow: deterministic notation
- Autogenerated some "well-formedness" properties
  - State consistency
  - Absence of *early return logic*
  - Junction loop bounds

New Work with NASA Ames and Vanderbilt University

- Examining well-formedness properties
  - Consistency of evaluation
    - Parallel state machines
    - Multiple enabled transitions
  - Finiteness of intra-step event graph
  - Chart state consistency
- Preservation properties across dialects
  - Creation of parameterized semantics for multiple dialects
  - Equivalence
  - Preservation of functional properties

Parallel State Consistency

- Syntactic mechanisms check disjointness of parallel charts (SyncCharts)



Example chart rejected by SyncCharts because x is assigned or tested by both parallel states S1 and S2.

Semantic Parallel State Consistency

- Attempt all interleavings for given state using incremental SAT solver
  - Create next-step transition relation in parts
    - Start from "leaf" parallel machines
  - Given current state, show equivalence of parallel machines for current step
    - If we can't show equivalence, flag an error
    - A little bit like partial order reduction
  - Choose arbitrary interleaving and compose up to next level

Predicate for Initial State $\wedge$ Predicate for Previous Execution Steps $\Rightarrow$ (AB $\Leftrightarrow$ BA)

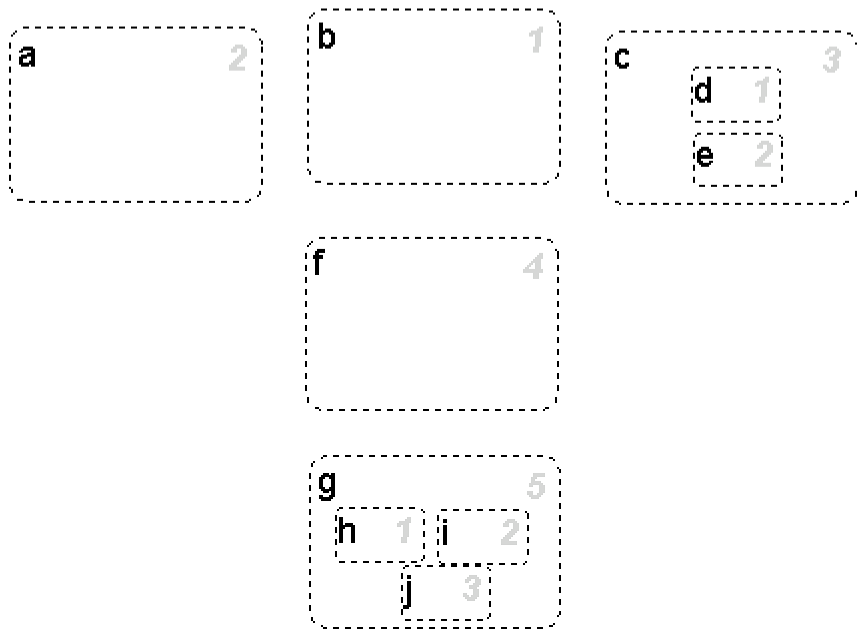
Conclusions

- Each of the examined semantics has quirks
- Be wary of assuming a particular semantics just given the visual notation
  - Bigger problem for groups that use more than one dialect (e.g. NASA) in same system
- Formal analysis is *very helpful* for finding latent bugs in charts
- Working on *parameterized semantics* for multiple dialects (derived from Hamon's, Atlee's work)
- Starting to explore analysis over multiple dialects

BACKUP SLIDES

Unintended Orderings (Stateflow)

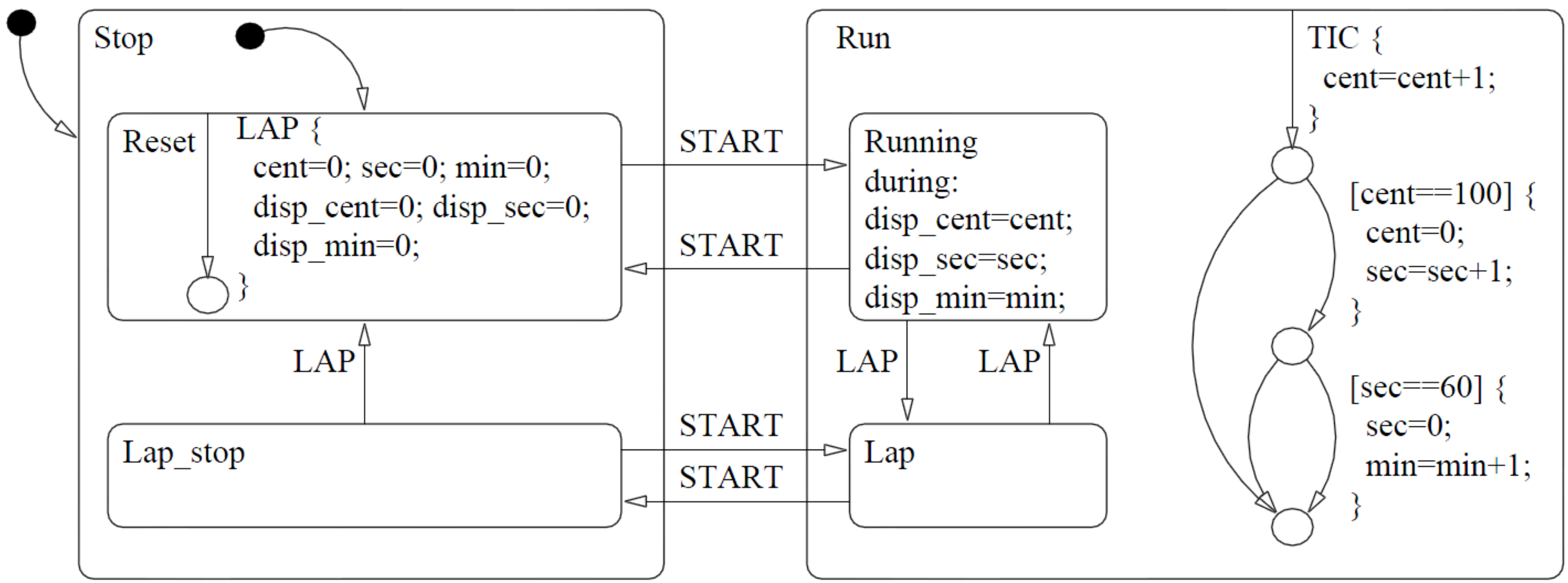
- Order of evaluation of parallel charts



A little history

- Chart determinism
  - Mats Heimdahl: Completeness and Consistency of RSML [1993-1996]
    - conservative
    - Not sound in the presence of multiple simultaneous events
- Functional properties
  - William Chan: Model Checking Large Software Specifications [1996-99]

Example Chart



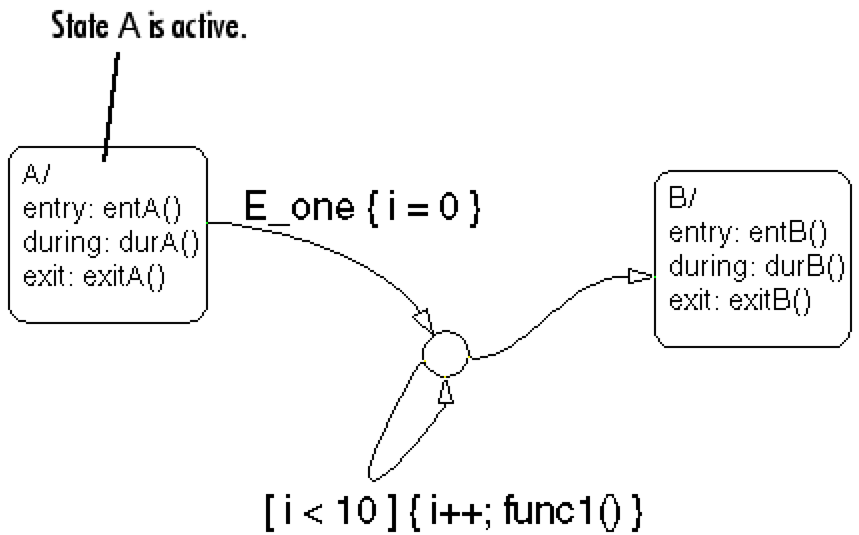
Stateflow Semantic Formalization

- SRI – Operational semantics
  - Large, Complex
  - Several facets of the language not covered
- Gregoire Hamon [Mathworks] – Denotational semantics
  - Small
  - Relatively complete
  - Not quite right
- I've been working with Gregoire on completeness and corrections

Stateflow Semantics Problems

- Two different kinds of actions: *transition* actions and *condition* actions
  - Condition actions occur upon satisfying condition for a transition *segment*
  - Transition actions only occur when transition reaches an end state
- Possible to use flowcharts to create poorly structured programming language

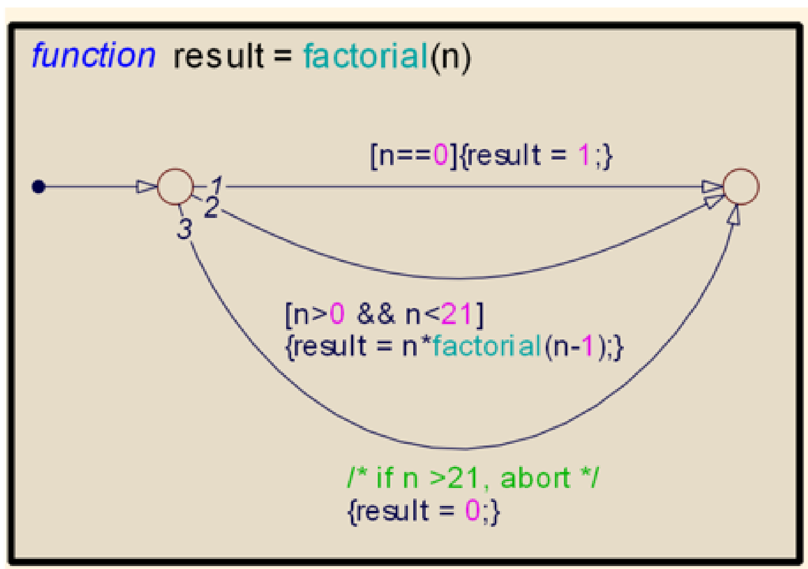
Strange looking charts



For Loop Chart

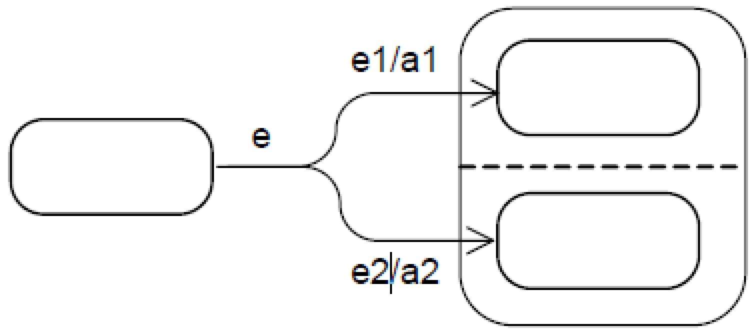
Syntax of Statecharts

- Non-graphical variables
- Functions
  - UML: Calls to functions / methods defined in a class
  - Stateflow: *Graphical Functions*



Discrepancies: Fork and Join

- What happens when forks reference multiple events?

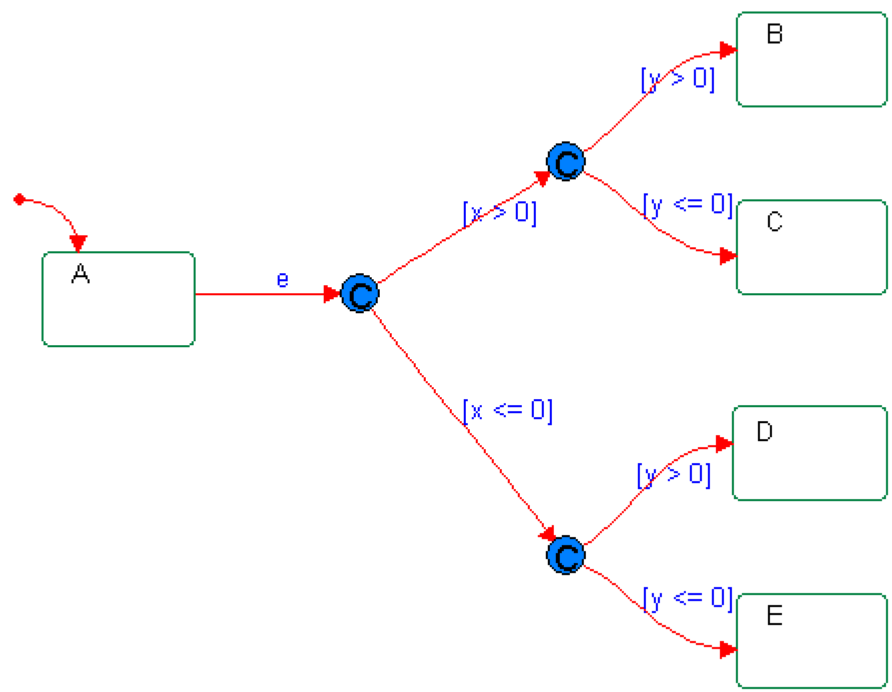


Classical, SyncCharts: multiple simultaneous events are possible, so the transitions have meaning

UML, Rhapsody, Stateflow: only one event at-a-time due to queueing; the transition cannot fire.

Rhapsody Semantics

- Conditional connectors allow splitting transitions based on condition
  - If >1 condition is simultaneously true one is selected arbitrarily.
  - All guards are evaluated *simultaneously* prior to actions.



Rhapsody Semantics

- Statecharts embedded within classes
- Each chart is assigned a thread
  - Multiple charts can share a thread
  - Thread operates as "event dispatcher" to its objects
- Event communication has two forms
  - Asynchronous queueing: GEN method
    - Can queue to self
  - Synchronous invocation: TRIGGER method
    - Function call semantics

Problems with Rhapsody

- Several parts of semantics are unspecified (according to Harel06)
- Event queuing allows possible interleaving between "internal" and "external" events
- Ordering of evaluation on parallel state machines is undefined

Chart Transition Consistency

- Local consistency: can > 1 outgoing transition fire from a given state?
  - Necessary for determinism within UML, Rhapsody, STATEMATE dialects
  - Sufficient to show determinism when paired with parallel state consistency
- Hierarchical consistency: can > 1 outgoing transition fire from state hierarchy?
  - Necessary (but not sufficient) to show determinism between different dialects