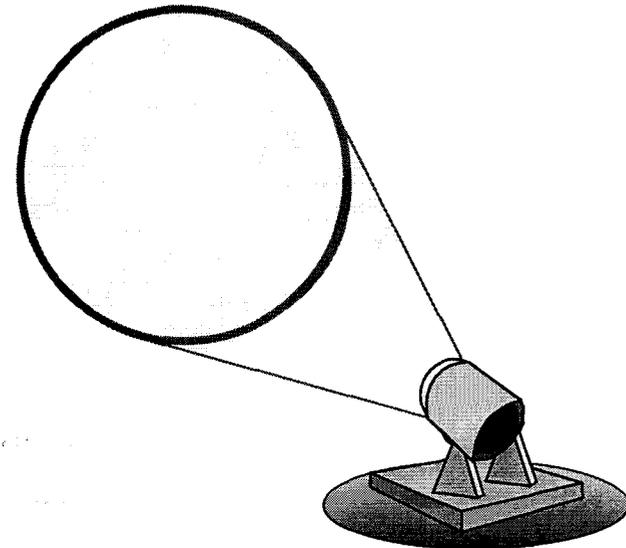




Automated Statechart Model Checking

Dr. Erich Mikk & Paula Pingree
Wednesday, August 28, 2002
12:00 - 1:00 PM
Conference Room 167





Center for Space Mission Information and Software Systems



The State of Software Quality





Task Outline

Objectives:

- Extend capabilities for flight software verification by introducing formal method model checking
- Evaluate and implement software tools that will help to automate the process
- Apply method and tools to Fault Protection (FP) flight software (FSW) implemented in StateFlow[®] statecharts as a prototype
- ***Infuse this verification technology in future projects***

Team:

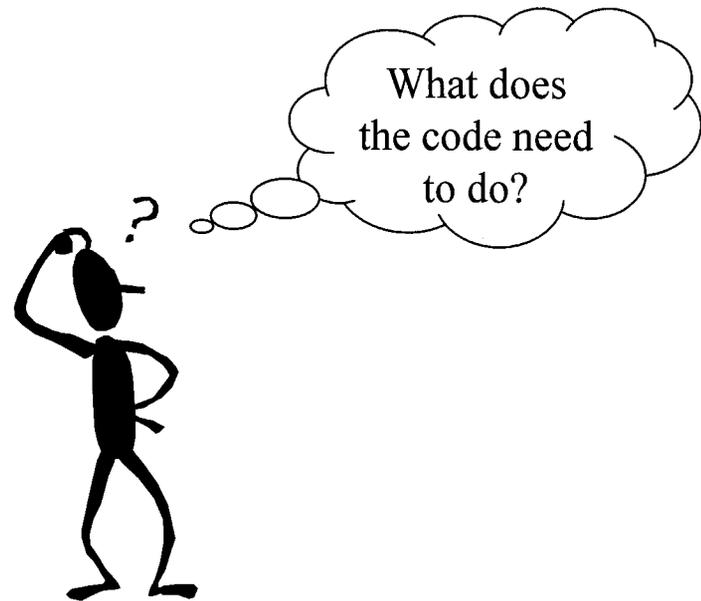
- Paula J. Pingree (JPL), Lead
*Systems Engineering & Technology Infusion Group
Autonomy & Control Section (345)*
- Erich Mikk (Erlangen, Germany), Independent Consultant
Developer of Extended Hierarchical Automata (EHA)
- Gerard Holzmann, Margaret Smith, Dennis Dams (Bell Labs, Murray Hill, NJ), Co-Investigators
Computer Principles Research Department

Funding:

\$90K in FY02 (SQI & CSMISS)

How We Develop Flight Software

- Systems Engineers specify FSW requirements
 - Textual, informal
 - Decomposition and traceability may be incomplete
 - Schedule may require that code development begin before requirements are finalized



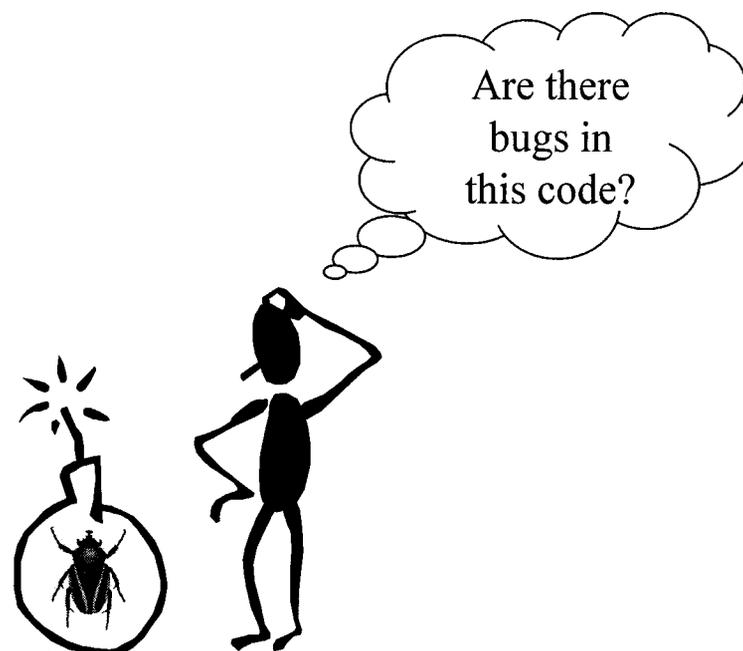
How We Develop Flight Software

- Systems Engineers specify FSW requirements
- Developers design the code
 - Requirements are interpreted
 - Derived requirements are created
 - Code reviews may be limited



How We Develop Flight Software

- Systems Engineers specify FSW requirements
- Developers design the code
- Test Engineers develop Test Cases & run Test Procedures
 - Human process prone to inconsistency
 - Limited resources (time, \$, personnel, testbeds)



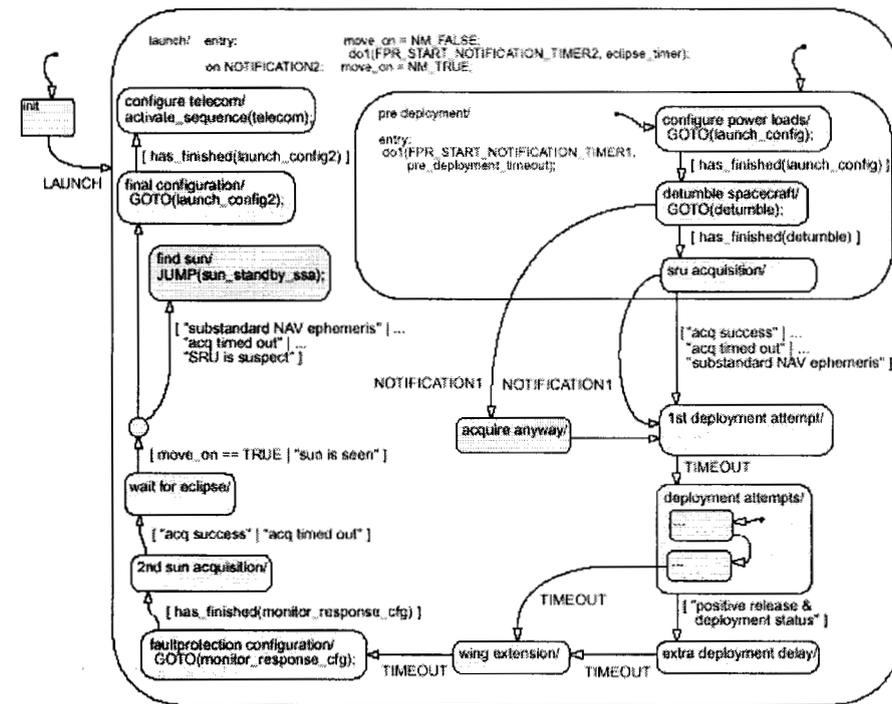
How We Develop Flight Software

- Systems Engineers specify FSW requirements
- Developers design the code
- Test Engineers develop Test Cases & Procedures
- Verification & Validation results
 - Some bugs are found & fixed
 - Some bugs remain



FSW Development Using Statecharts

- DS1's "13th Technology"
 - Model-based code generation of Fault Protection (FP) Monitors & Responses
 - Accomplished using Stateflow® and Stateflow Coder by The Mathworks
 - Highly successful implementation



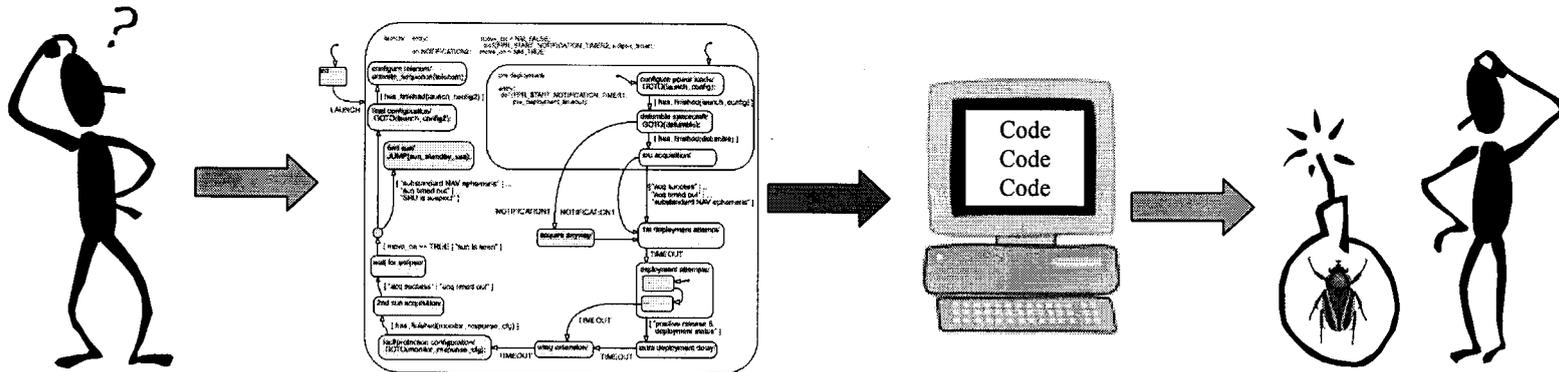
A section of the launch statechart, showing sun acquisition and pre-deployment of the DS1 solar array panels.



FSW Development Using Statecharts

- Benefits:
 - Enforces standard diagrammatic conventions
 - Allows design & implementation by Systems Engineers
 - Provides concise design notation for easier review
 - Open and customizable architecture exists for auto-code generation
- Other Implementations in development:
 - Deep Impact Fault Protection
 - MDS Threading Policies

FSW Design Using Statecharts

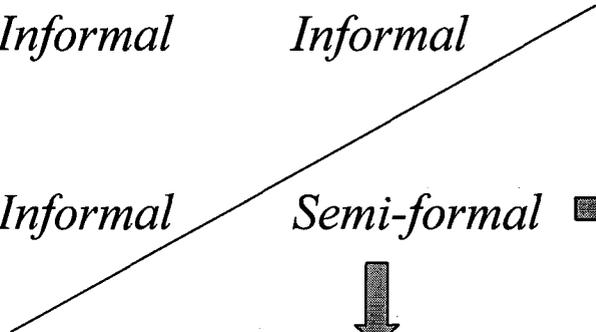


- Validation of auto-generated code follows traditional testing methods
 - Iteration for bug fixes occurs downstream in development cycle
 - Still never sure if bugs remain in design
- Can we take advantage of Model-based Validation Methods?

A New Approach: Model Checking

- Apply “Lightweight” Formal Methods to FSW Validation
- Use the SPIN Model Checker
 - Developed at Bell Labs by Dr. Gerard Holzmann
 - SPIN can exhaustively examine the state space of a model and detect violations of the user-specified properties, e.g. unreachable states

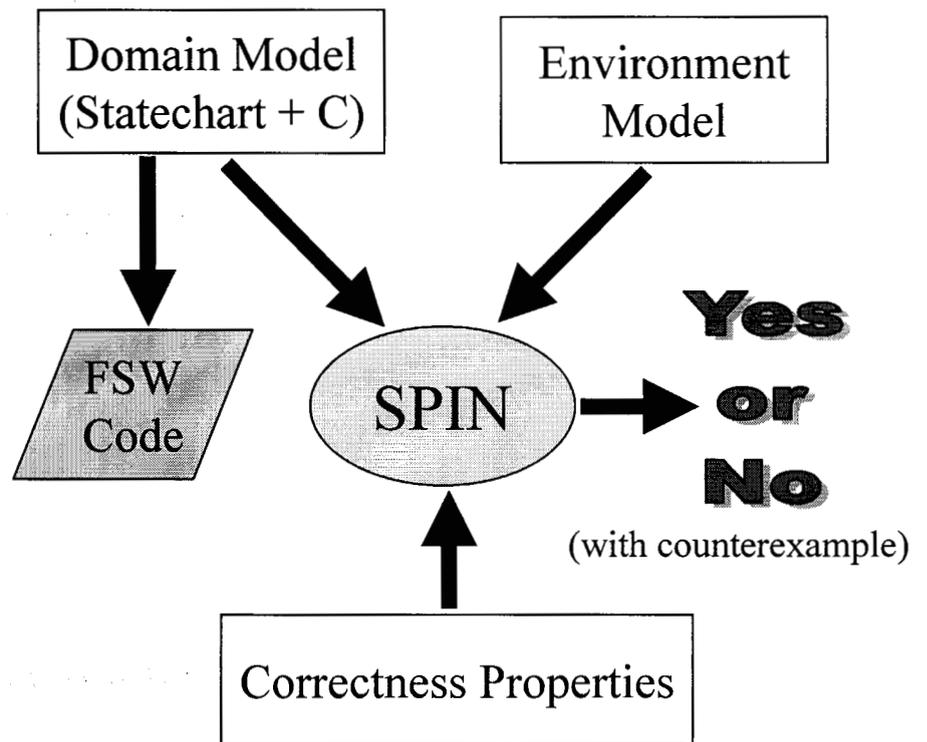
	Traditional	Statecharts	Model Checking
Requirements	<i>Informal</i>	<i>Informal</i>	<i>Formal (LTL)</i>
Design	<i>Informal</i>	<i>Semi-formal</i> →	<i>Formal (Promela)</i>
Code	<i>Formal</i>	<i>Formal</i>	





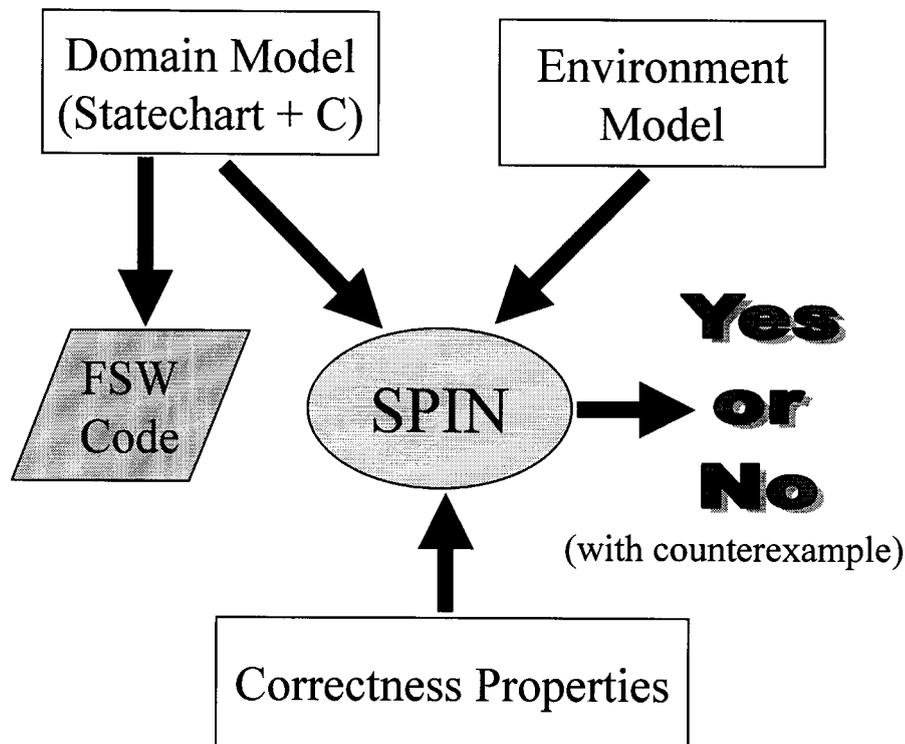
Applying Model Checking to FSW

- We provide automated translation of the statechart model from Stateflow to Promela, the input modeling language of SPIN
- Key Benefits:
 - SPIN validation model and FSW code, now both auto-generated, have the same source (the Stateflow statechart)
 - Validation of statechart design can occur earlier in development cycle and without use of valuable testbed resources



- Based on Requirements
- Expressed in Linear Temporal Logic (LTL)

Code Generation for a Model Checker



- Based on Requirements
- Expressed in Linear Temporal Logic (LTL)

Generating code for a model checker is different

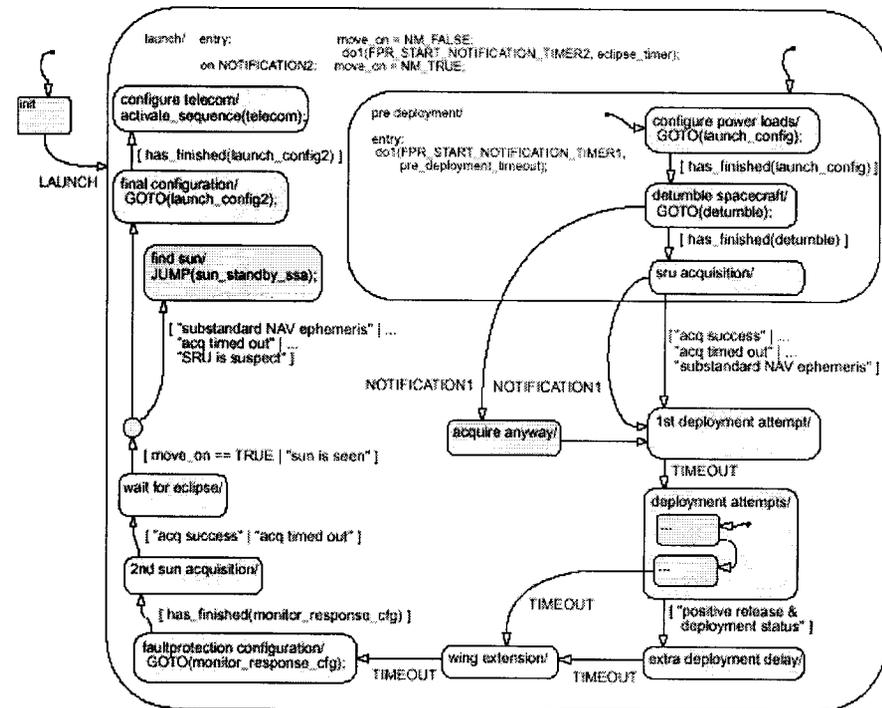
- Which behavior we are going to observe in the model checker?
- What are the properties that every model under verification should satisfy?
- Can we generalize the semantics such that the verification results hold even if verification object or the tools processing it change?

What is in the Step?

What would we like to observe on this statechart?

1. Every action, i.e. state entries and exits, variable changes, ...
2. Reaction to one "tick"
3. Complete system reaction to an external trigger

There is no golden way ... we have to keep it flexible





Properties of Reactive Systems - Responsiveness

We adopt synchrony hypothesis due to G. Berry

1. Software system is infinitely faster than its environment
2. Software system is responsive

Responsiveness - software system reacts to every input from the environment

This means among others:

- (*) Software system accepts every environment input
- (*) It executes at least one action or transition as a reaction
- (*) It returns to the state where it can accept forthcoming input events

If our models are responsive:

1. No looping behavior
2. No non-reactions



Non-Determinism as a Means of Generalization

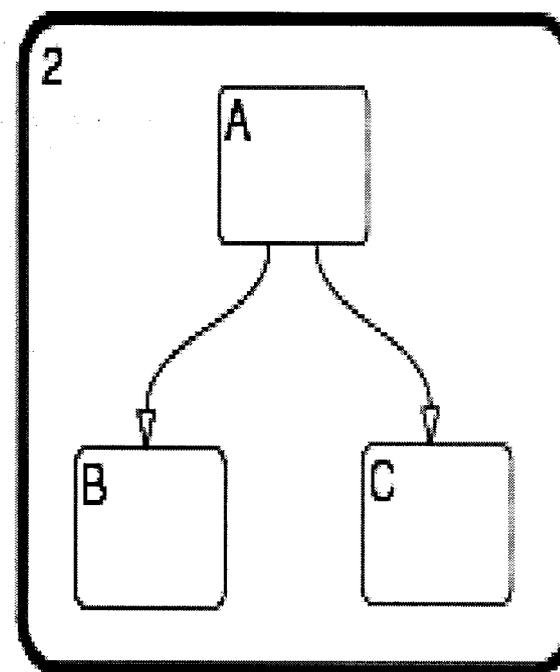
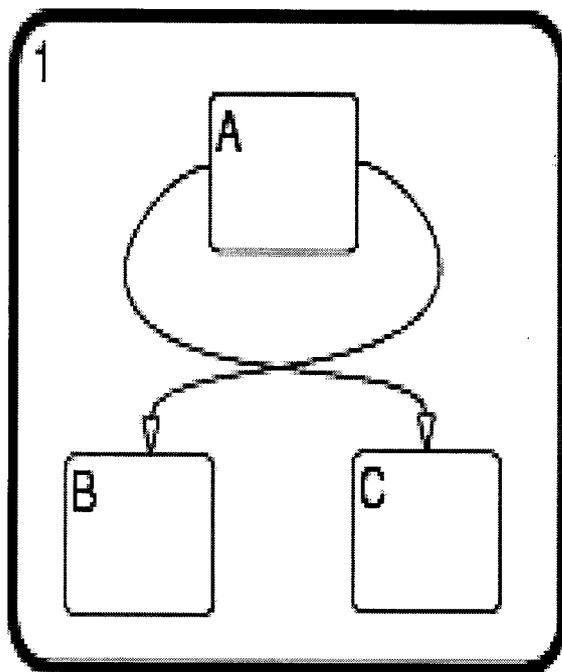
Stateflow determines

- (*) The execution order of AND-states
- (*) The execution order of transitions emerging from one state
- (*) The backtracking order of transition cascades

This means that system properties rely on the

1. Mutual positions of AND-states
2. The place where the emerging transition is attached at the source transition
3. ...

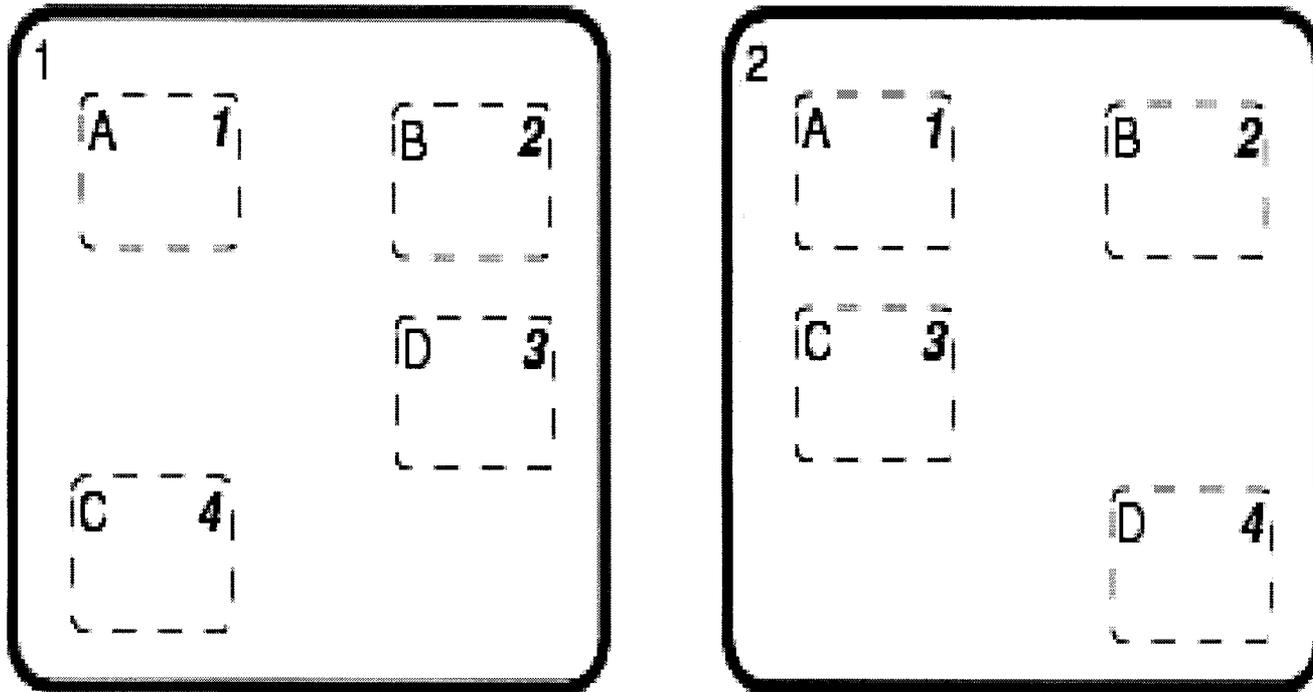
Generalization of Emerging Transitions



Stateflow semantics: 1: $A \rightarrow B$ 2: $A \rightarrow C$

Generalization: in 1 & 2: $A \rightarrow B$ and $A \rightarrow C$

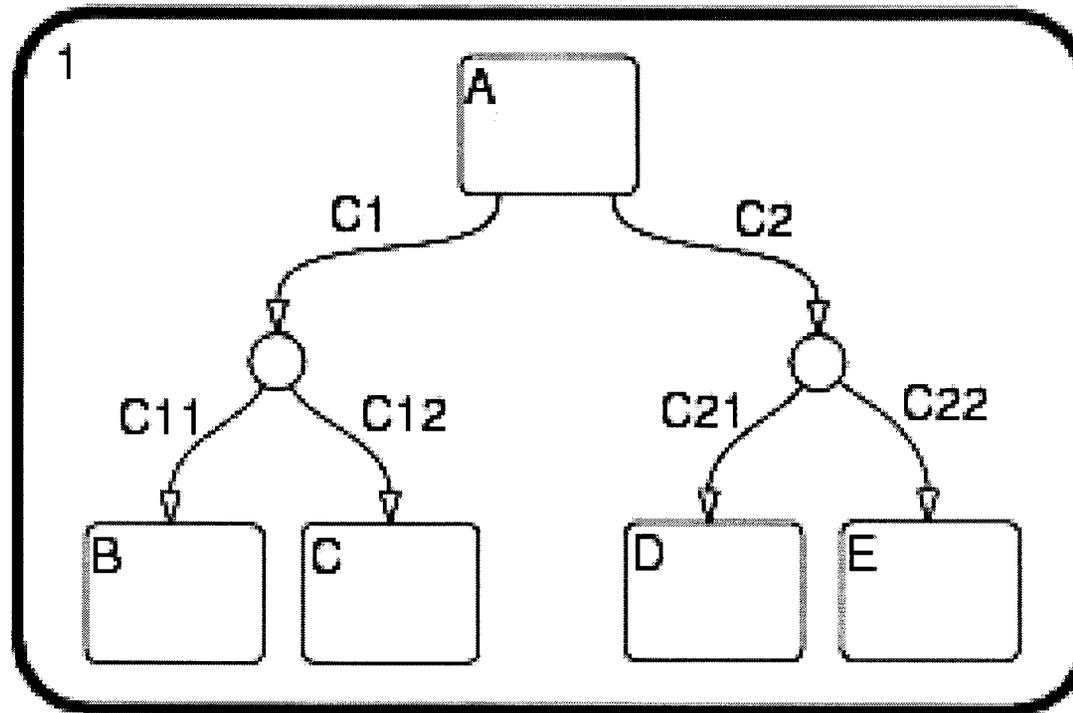
Generalization of AND-states



Stateflow semantics: 1: A, B, D, C 2: A, B, C, D

Generalization: in 1& 2: all combinations of A, B, C, D

Generalization of Backtracking



Stateflow semantics: C2, C22, C21

Generalization: C2, C22, C21 and C2, C22, C1...



Task Continuation

- Product Peer Review to be held in September
 - Final Report on FY02 Task
 - Tool Set and User's Guide delivered
- R&TD Proposal submitted for FY03 (\$400K):
 - “Rapid Adoption of Model-Based Validation for Mission-Critical Flight Software Architectures & Domains”
 - Recommended by the Advanced Software Technology and Methods Initiative (ASTMI)
- Continued support in FY03 from SQI
 - Tool maintenance and improvement
 - Technology infusion for future projects



Technology Infusion - Make It Happen

- Potential Applications
 - All projects using Stateflow or other statechart representations or any other precise description of dynamic behavior
 - Domains: Fault Protection (FP), Protocol validation...
 - Supports software reuse
- Demonstrate relevance of increased software quality with these tools and methods
 - DS1 prototype, Deep Impact FP
- Make tools available to JPL community

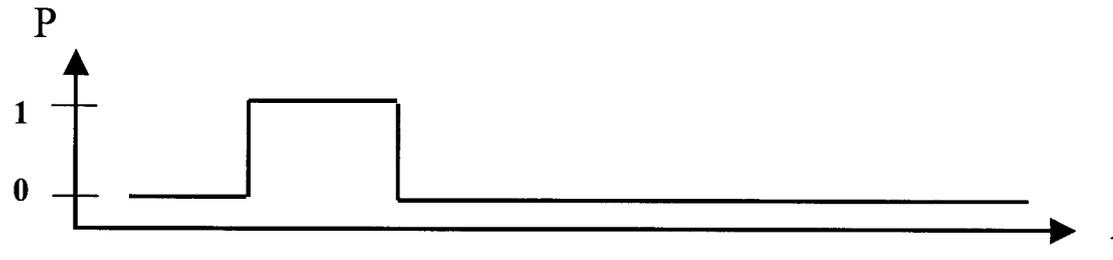


References

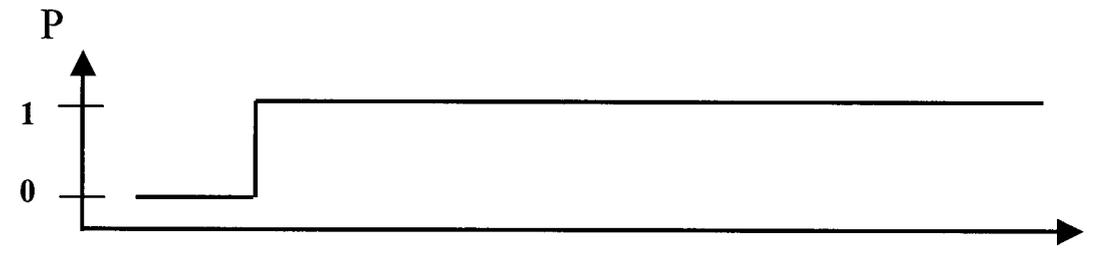
- P. Pingree, E. Mikk, G. Holzmann, M. Smith, D. Dams, Validation of Mission Critical Software Design And Implementation Using Model Checking. *Accepted for the 21st Digital Avionics Systems Conference, October 2002*
<http://eis/~ppingree/pubs.html>
- E. Mikk, Semantics and Verification of Statecharts. PhD Thesis. *Technical Report of Christian-Albrechts-University in Kiel, October 2000*
- E. Mikk, Y. Lakhnech, M. Siegel and G. Holzmann, Implementing Statecharts in PROMELA/SPIN. In *Proceedings of the 2nd IEEE Workshop on Industrial-Strength Formal Specification Techniques*. pages 90-101. IEEE Computer Society 1999
- G.J. Holzmann, The model checker Spin, *IEEE Trans. on Software Eng.*, 5(23):279-295, 1997

Appendix-1

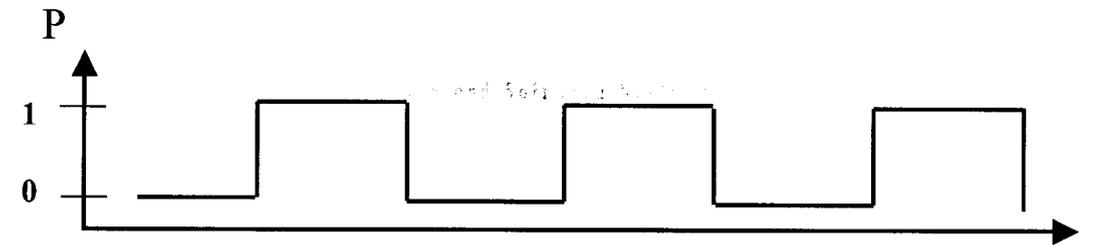
$\langle \rangle P$



$\langle \rangle [] P$



$[] \langle \rangle P$



Appendix-2

