Semantic Translation of Simulink/Stateflow Models to Hybrid Automata using Graph Transformations

A. Agarwal, Gy. Simon, *G. Karsai*
ISIS, Vanderbilt University
Nashville, TN 37235, USA

Overview

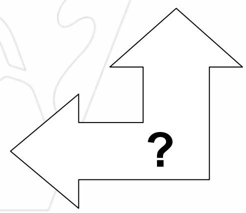
- Translation problem
- Tool used: GReAT
- Algorithm with example
- Summary

The translation problem

Motivation

Simulink/Stateflow

- *De facto* prototyping and simulation environment for dynamic systems
  - E.g.: Embedded controllers for automobiles
- Large legacy libraries
- No formal verification capability

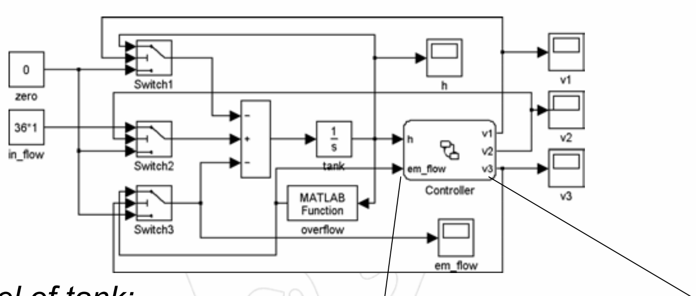


Hybrid automata

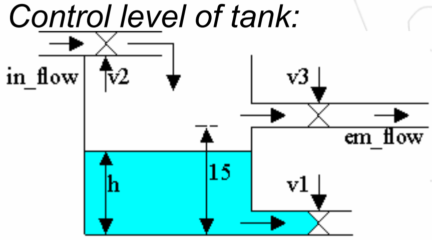
- Mathematical modeling technique
- Formal foundations
- Few real-life examples
- Verification capability

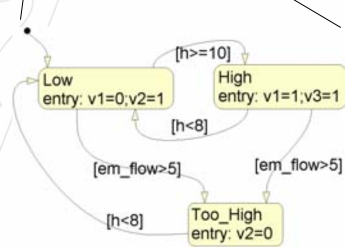
The translation problem

Simulink/Stateflow Example (Input)



Control level of tank:





The translation problem

Hybrid Automata (Output)

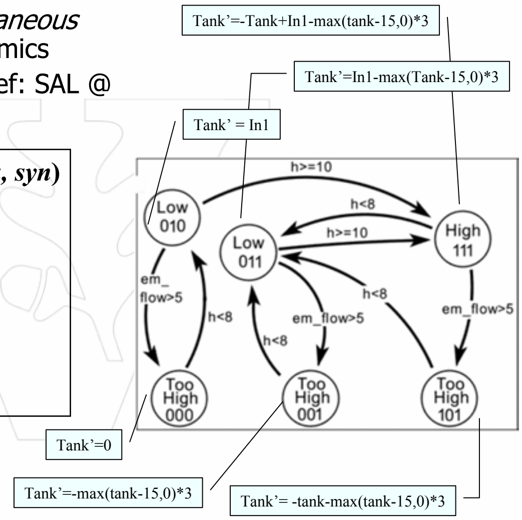
Hybrid Systems:

Dynamic systems with *simultaneous* continuous and discrete dynamics

Hybrid Automata Example (ref: SAL @ U Penn)

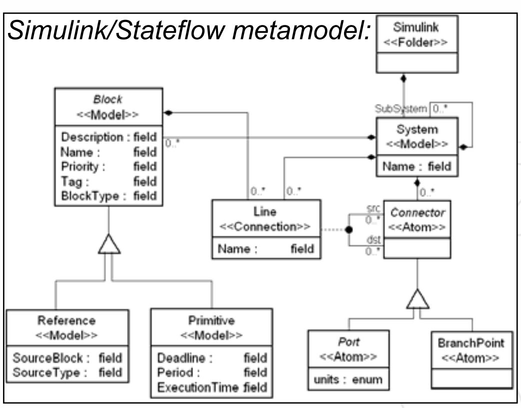
A = (*X, V, flow,inv, init, E, jump, å, syn*)

- — ***X*** **: a set of real-valued variables**
- — ***V*** **: a set of control modes**
- — ***flow*** **: a flow condition over** ***X***
- — ***inv*** **: a set of invariant over X**
- — **init : an initial condition**
- — ***E : a set of transitions***
- — ***jump*** **: a condition for transition**
- — ***å*** **: a set of events**
- — ***syn*** **: a set of synchronization labels**

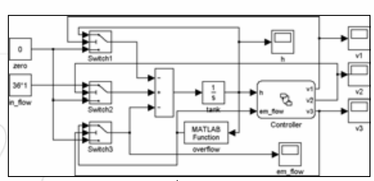


Background

Metamodels as graph grammars



Simulink/Stateflow model:



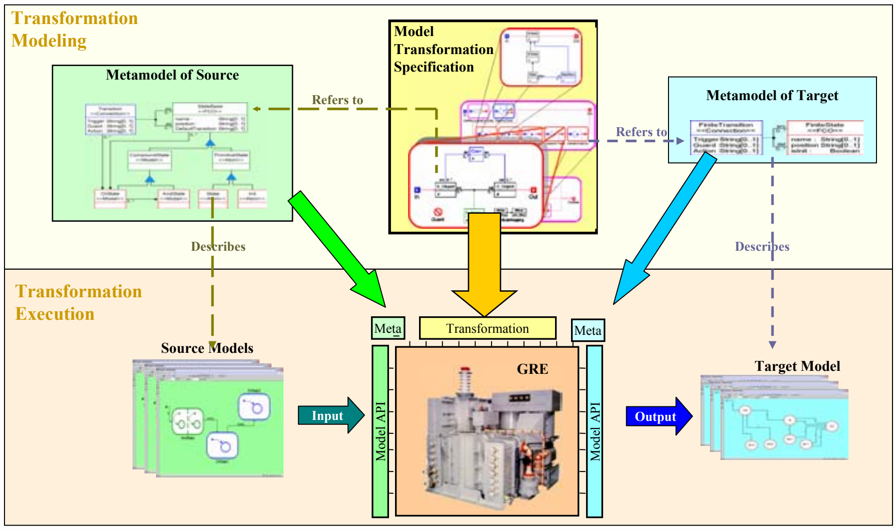
Visualization

Model Object Network

Graph grammar - - - - - - - - - - - - → Graph

UML-based metamodels for:
Simulink/Stateflow *and* ***HSIF*** *(HA interchange language)*

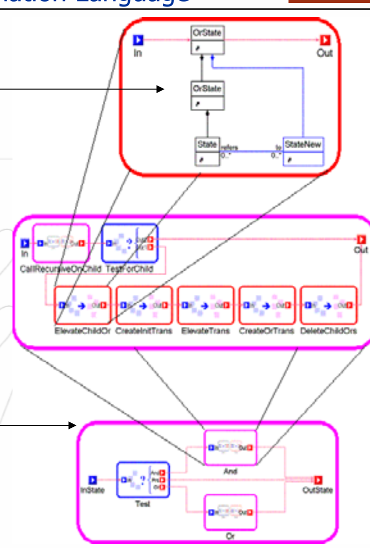
Tool used: GReAT



GReAT:

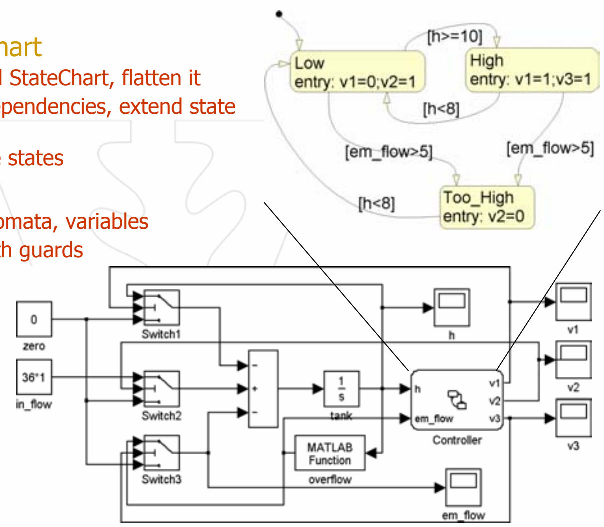
UMT: A Simple Model Transformation Language

1. Pattern specification
    - Pattern variables are typed with their UML classes
    - Cardinality of association-ends is checked
    - Extra (OCL) constraints define guard conditions
2. Graph transformation and rewrite
    - Create new/delete/modify objects
    - Attribute mapping (procedural)
    - "Cross-links": edges between old/new objects
    - Input/output ports: pre-bound pattern variables
3. "High-level" control flow over the rules
    - Port connections imply "data flow" and control flow
    - Hierarchy/Sequencing/Recursion/Branching



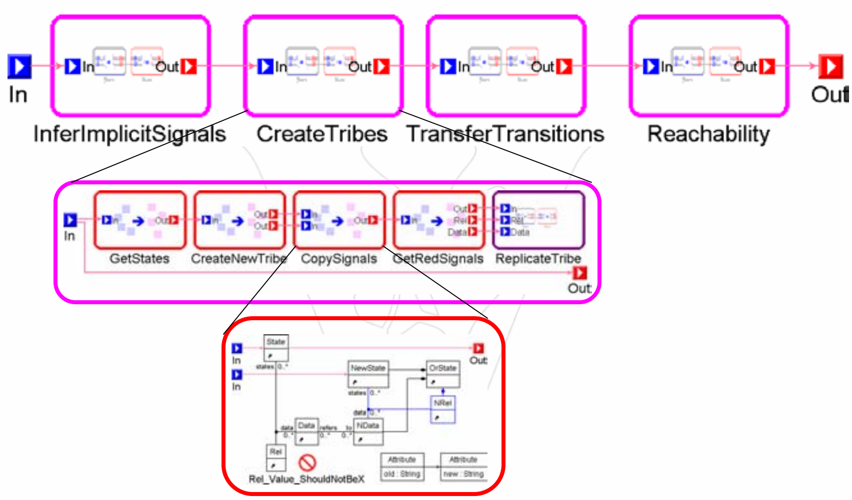
Algorithm

1. Stateflow Part
   1. Convert to StateChart
      1. Create Hierarchical StateChart, flatten it
      2. Determine data dependencies, extend state machines
      3. Prune unreachable states
   2. Convert to HSIF
      1. Create Hybrid Automata, variables
      2. Add transitions with guards
2. Simulink Part
   1. Locate associated Hybrid Automata
   2. Add variables as needed
   3. Derive and add equations



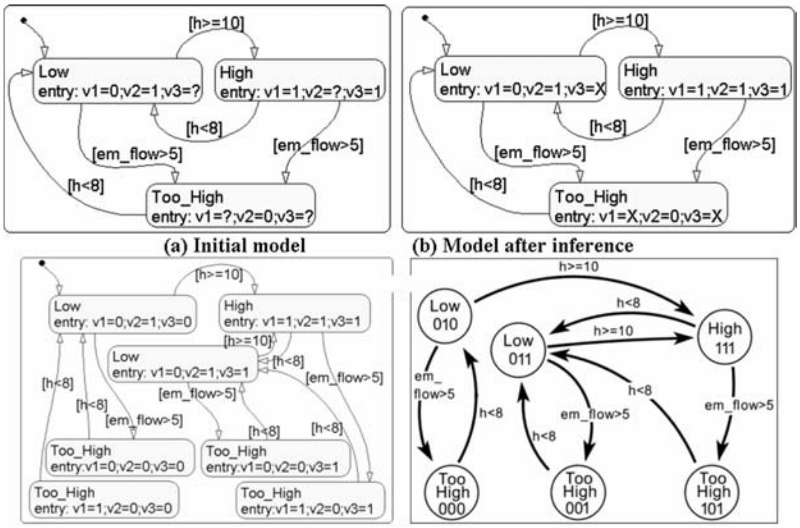
Algorithm

Determine data dependencies, extend state machines



Algorithm

Inferring signals, extending state machines



(a) Initial model

(b) Model after inference

(c) Transferring the transitions

(d) Final Hybrid Automata States

Status, metrics

- A hierarchal Simulink diagram with the following primitives:
  - Continuous : Integrator
  - Math : Product/Sum/Gain/Abs/Min/Max/Signum/Saturate
  - Signal and Systems : Mux/Demux/Ground
  - Source and Sinks : Constant/Workspace variables
  - Nonlinear : Controlled Switch/Manual Switch
- The Simulink diagram can contain any number of Stateflow diagram.
- Stateflow diagram can be hierarchical.
- The Stateflow diagram receives signals from Simulink and can only produce switching signals that control the switches.
- Switches cannot be controlled by any other Simulink block.
- In Stateflow, the switch control action can only be performed in the entry action.

Complexity

- Most algorithms are of polynomial complexity
- Some parts are worst-case exponential:
  - State-splitting
  - Flattening

Size

- Primitive rules: 154
- Complex rules: 43
- C++ code: ~6000 lines

GReAT in Action

Problem	Developer	GReAT		Hand code
		Primitive Rules #/ Compound Rules #	Man hours	LOC
Hierarchical Data Flow (HDF) to Flat Data Flow (FDF)	Staff Eng	11/3	~3	~200
KHORUS to GUDML	MSc Student	19/10	~8	~500
Hierarchical Concurrent State Machine (HCSM) to Finite State Machine (FSM)	PhD Student	21/5	~8	~500
Simulink Stateflow to C code	PhD Researcher	70/50	~25	~2.5K
Matlab Simulink/ Stateflow to Hybrid Automata	PhD Student	154/43	~60	~6K

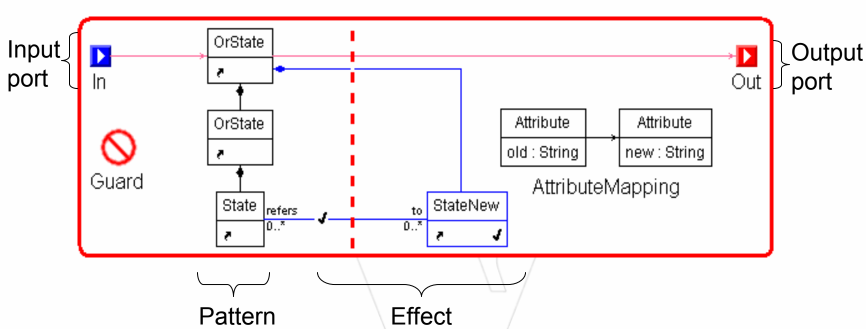
Summary, further work

- Tool integration requires translators that convert models created in one tool into semantically equivalent models in other tools. Translators are essential for design automation…*but difficult to build*.
- Graph transformations can be used to solve practical translation problems … *if good supporting tools are available.*
- Modeling a transformation using GT programs offers an opportunity for reasoning about the transformations --- *A great potential area for research.*

Background slides

UMT

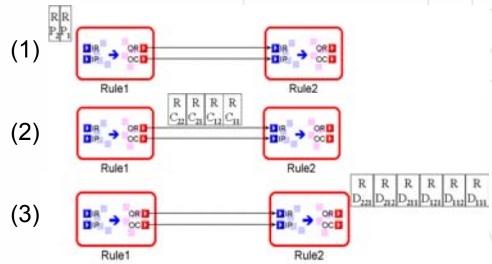
A Transformation Rule



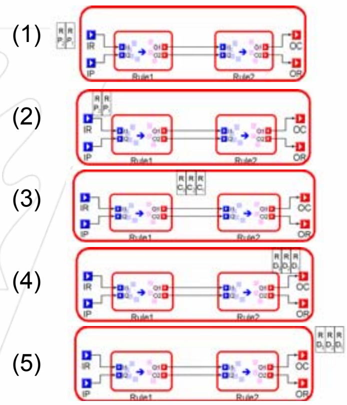
UMT

Rule execution – Rules and Blocks

Rules produce multiple matches: "packets". By default, all packets are consumed by a rule, and new set of packets is produced



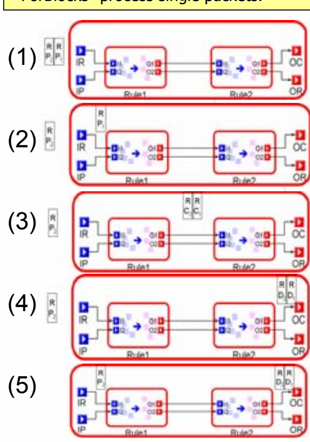
"Blocks" are composite rules, with simple composition semantics.



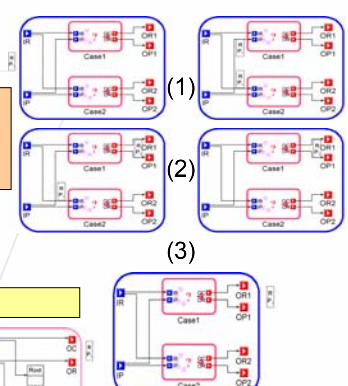
UMT

Rule execution- ForBlocks and Test/Cases

"ForBlocks" process single packets.



Tests are conditional control structures built from *Cases*.



Also supported

- Recursion
- Non-deterministic execution

A single *Case*: