Translation Validation for Stateflow Code Generator*

Prahladavaradan Sampath†

Joint work with Rajeev A.C † and Ramesh S‡

* This work was carried out when the authors were part of General Motors India Science Lab

† Mathworks

‡ ECI Lab, General Motors R&D, Warren

Model-Based Development

- To develop complex software systems
  - Model → Validate → Refine → Auto-generate code
- Employs high-level modeling languages
  - Formal syntax → less ambiguous than natural language
  - Formal semantics → enables automated analyses
- Highly tool intensive
  - Syntax checking, Simulation, Analysis, Test generation, Code generation (Collectively called model processors)
- Advantages
  - Less development time, ease of re-design
  - Early verification and debugging
  - Model-based test-case generation
  - Automatic code generation

Code Generator

- Code generators are tools that take as input “models” in a modelling language and output various artifacts:
  - Code
  - Other models (one man’s model is another man’s code)

- Examples of code-generators
  - Rhapsody code-generator
  - Matlab/Stateflow simulator
  - Lex/Yacc
  - Query optimizers
  - …

Approaches to Verify Code Generators

- Formally verifying the code generator
  - White-box, one-time, interactive, strong guarantee
- Testing the code generator
  - Black-box, one-time, automated, weak guarantee
  - Manual / automated test generation
    - Special ATG methods to handle syntactic and semantic structure of inputs and outputs
- Model based testing (most common in practice)
  - Black-box, every-run, automated, weak guarantee
- Translation validation
  - Black-box, every-run, automated, strong guarantee

Different Approaches

- Proving a code generator

  $\forall$ m:models, $\forall$ i:inputs:

  ModelExec(m, i) $\approx$ CodeExec(CodeGen(m),i)

- Testing a code generator

  Formany m:models, Formany i:inputs:

  ModelExec(m, i) $\approx$ CodeExec(CodeGen(m),i)

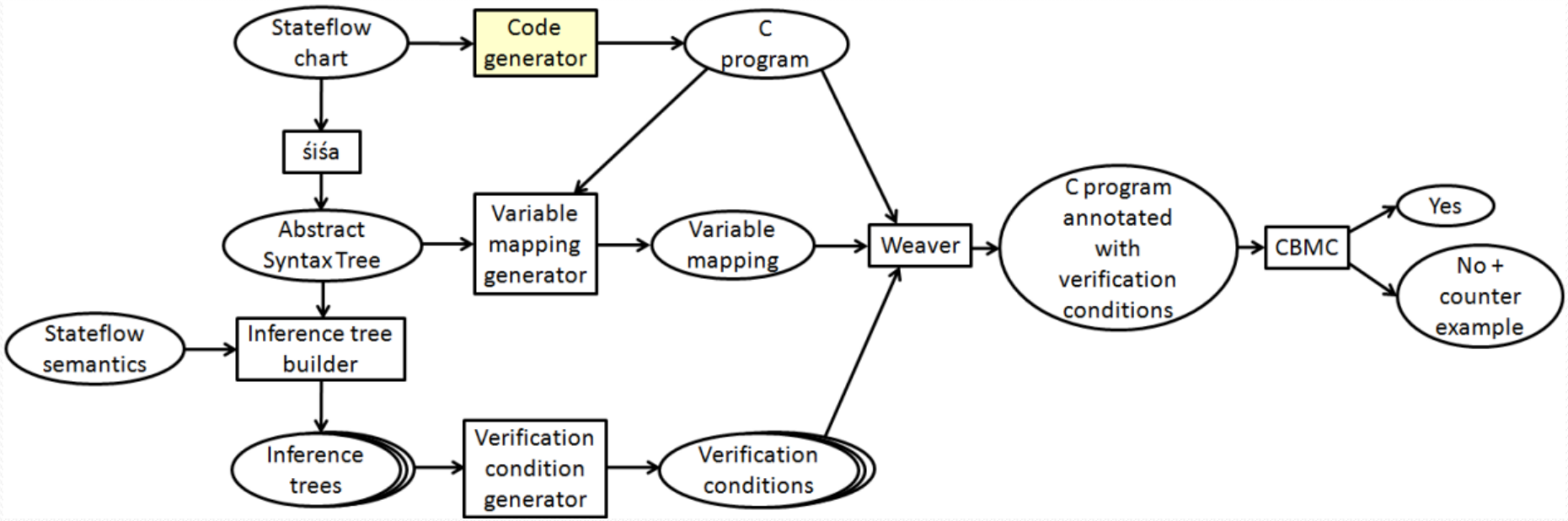
- Translation validation : fix a model m

  $\forall$ i:inputs: ModelExec(m, i) $\approx$ CodeExec(CodeGen(m),i)

Translation Validation

- Mathematical proof of equivalence between model and program
  - Every translation is followed by validation
- Strengths
  - Strong guarantee
  - Does not require source code of translator
  - Automated
- Weaknesses
  - Validation has to be done after every run of the translator
  - Computation intensive
  - Based on the following assumptions
    - Formal semantics of the modeling and programming languages are available
    - Behaviours of the model and program are finite in number
    - A mapping can be identified between model elements and program elements
    - Verification conditions can be proved

Tool Architecture



Step-1

- Obtain *all behaviours* of the given Stateflow model
  - Using a formal semantics for Stateflow
  - Generate all possible inference trees corresponding to the given model
    - Using inference rules in semantics
  - Iterate over all “proofs” using a Hoare logic style semantics
    - Assumes “bounded” behaviour – no loops!

Step-2

- Generate verification conditions from inference trees
  - As Hoare tuples: {Pre-condition} Ch {Post-condition}
    - Active states before and after execution
      - Identify from the structure of the inference tree
    - Variable values before and after execution
      - Extract the sequence S of guards and actions from the inference tree
        - Guards: boolean conditions over variables, presence/absence of events
        - Actions: variable assignments, event broadcasts
      - Compute wp(S, true)
        - wp($x \leftarrow exp$, P) = P[x/exp], wp($event^{+}(e)$, P) = P && e^{+}, wp($event^{-}(e)$, P) = P && e^{-}
      - Symbolically execute S with respect to wp(S, true)
        - *Assuming* wp(true, S) = $P(x_1, ..., x_n)$, *we compute symsim(P, S)* = $Q(x_1, ..., x_n, x'_1, ..., x'_n)$

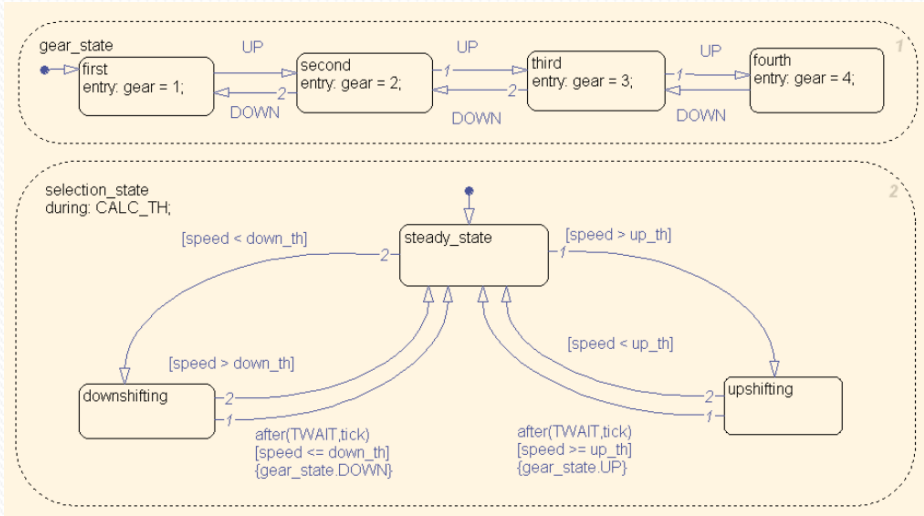
Example: Shift_logic

- 37 inference trees = 37 unique behaviours
- An inference tree:

$$\dfrac{\dfrac{\dfrac{\textbf{Trans-S}(\text{t10}) \quad \textbf{Atom-dE}(\text{first})}{\textbf{OR-dE-dE}(\text{gear\_state})} \quad \dfrac{\textbf{Trans-S}(\text{t11}) \quad \textbf{Atom-dE}(\text{steady\_state})}{\textbf{OR-dE-dE}(\text{selection\_state})}}{\textbf{AND-dEs-dE}(\text{root})}$$

- Pre-condition: all states are inactive (WP calculation)
- Post-condition: gear_state, first, selection_state and steady_state are active, gear == 1 (Symbolic simulation)

{true}
gear = 1
{true}



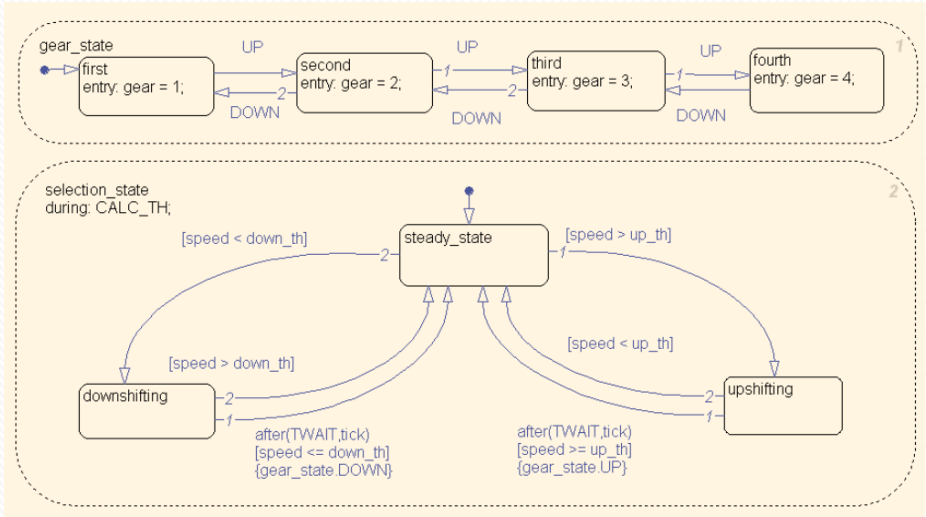
Example: Shift_logic

- 37 inference trees = 37 unique behaviours
- An inference tree:

$$\dfrac{\dfrac{\dfrac{\textbf{Trans-S}(\text{t10}) \quad \textbf{Atom-dE}(\text{first})}{\textbf{OR-dE-dE}(\text{gear\_state})} \quad \dfrac{\textbf{Trans-S}(\text{t11}) \quad \textbf{Atom-dE}(\text{steady\_state})}{\textbf{OR-dE-dE}(\text{selection\_state})}}{\textbf{AND-dEs-dE}(\text{root})}$$

- Pre-condition: all states are inactive (WP calculation)
- Post-condition: gear_state, first, selection_state and steady_state are active, gear == 1 (Symbolic simulation)

{true}
gear = 1
{gear == 1}



Step-3

- Identify the mapping between model elements and code elements
  - Files: *md.c, md_data.c, md.h, md_private.h*
  - Chart ch: function *void md_ch(void)*
  - Events: integer variable *_sfEvent_md_* with values from *{md_event_e1, …, md_event_en, CALL_EVENT}*
  - State s: field *is_active_s* (*boolean*) and field *is_s* (*{md_IN_s1, …, md_IN_sn, md_IN_NO_ACTIVE_CHILD}*) in structure variable *md_DWork*
    - History junction in s: field *was_s* (*boolean*) in *md_DWork*
  - Local variables: fields in structure variable *md_B*
  - Inputs: fields in structure variable *md_U*

Step-4

- Prove the verification conditions on C code
  - Annotate the generated C code with {Pre-condition} and {Post-condition}
    - Use the mapping between model elements and code elements
  - Prove using C model-checker CBMC
    - Failed proof can provide a test-case showing the difference between the behaviours of model and code

Annotated Code

- C file: atc.c

```
void ForTV(){
  _sfEvent_atc_ = nondet_uint8_T();
  atc_DWork.is_active_c1_atc = nondet_uint8_T();
  atc_DWork.is_active_gear_state = nondet_uint8_T();
  atc_DWork.is_gear_state = nondet_uint8_T();
  atc_DWork.is_active_selection_state = nondet_uint8_T();
  atc_DWork.is_selection_state = nondet_uint8_T();
  atc_DWork.temporalCounter_i1 = nondet_uint8_T();
  atc_U.in_speed = nondet_real_T();
  atc_U.in_up_th = nondet_real_T();
  atc_U.in_down_th = nondet_real_T();
  atc_B.gear = nondet_real_T();

  __CPROVER_assume(atc_DWork.is_active_c1_atc == 0 &&
                   atc_DWork.is_active_gear_state == 0 &&
                   atc_DWork.is_gear_state == atc_IN_NO_ACTIVE_CHILD &&
                   atc_DWork.is_active_selection_state == 0 &&
                   atc_DWork.is_selection_state == atc_IN_NO_ACTIVE_CHILD);

  D_Work_atc atc_DWork_1 = atc_DWork;
  BlockIO_atc atc_B_1 = atc_B;
  ExternalInputs_atc atc_U_1 = atc_U;

  atc_Shift_logic();

  assert(atc_DWork.is_active_c1_atc == 1 &&
         atc_DWork.is_active_gear_state == 1 &&
         atc_DWork.is_gear_state == atc_IN_first &&
         atc_DWork.is_active_selection_state == 1 &&
         atc_DWork.is_selection_state == atc_IN_steady_state &&
         atc_B.gear == 1);
}
```

Some Case-studies

- Shift_logic in ATC demo model: 37 verification conditions
- A number of models with history junctions, event broadcasts, graphical functions, multi-level transitions, etc.
- HVAC controller models

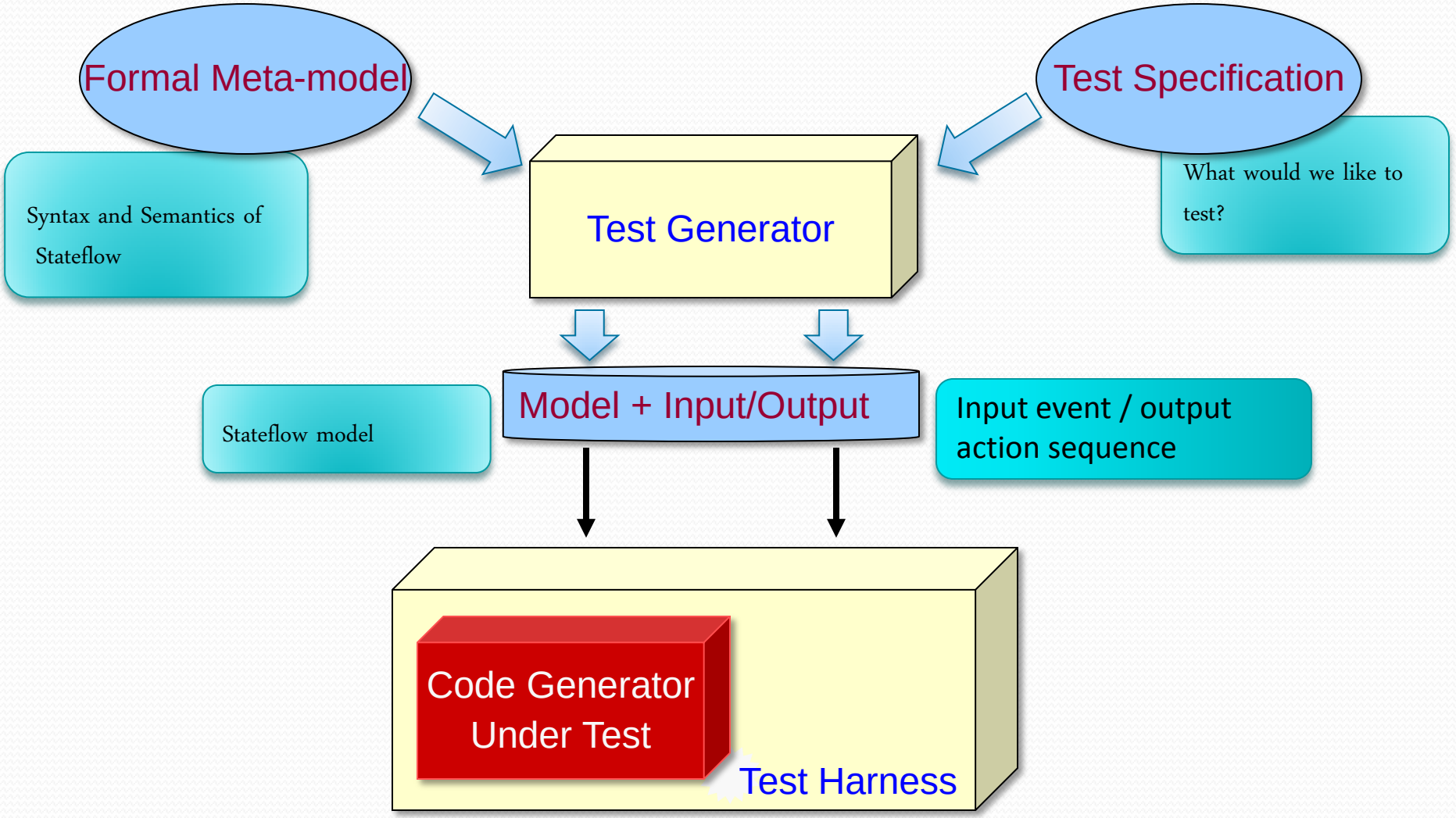
Challenges

- Semantics of modelling language
  - Is our formalization correct?
- Binary Yes/No answer is not great
  - Can we do better?

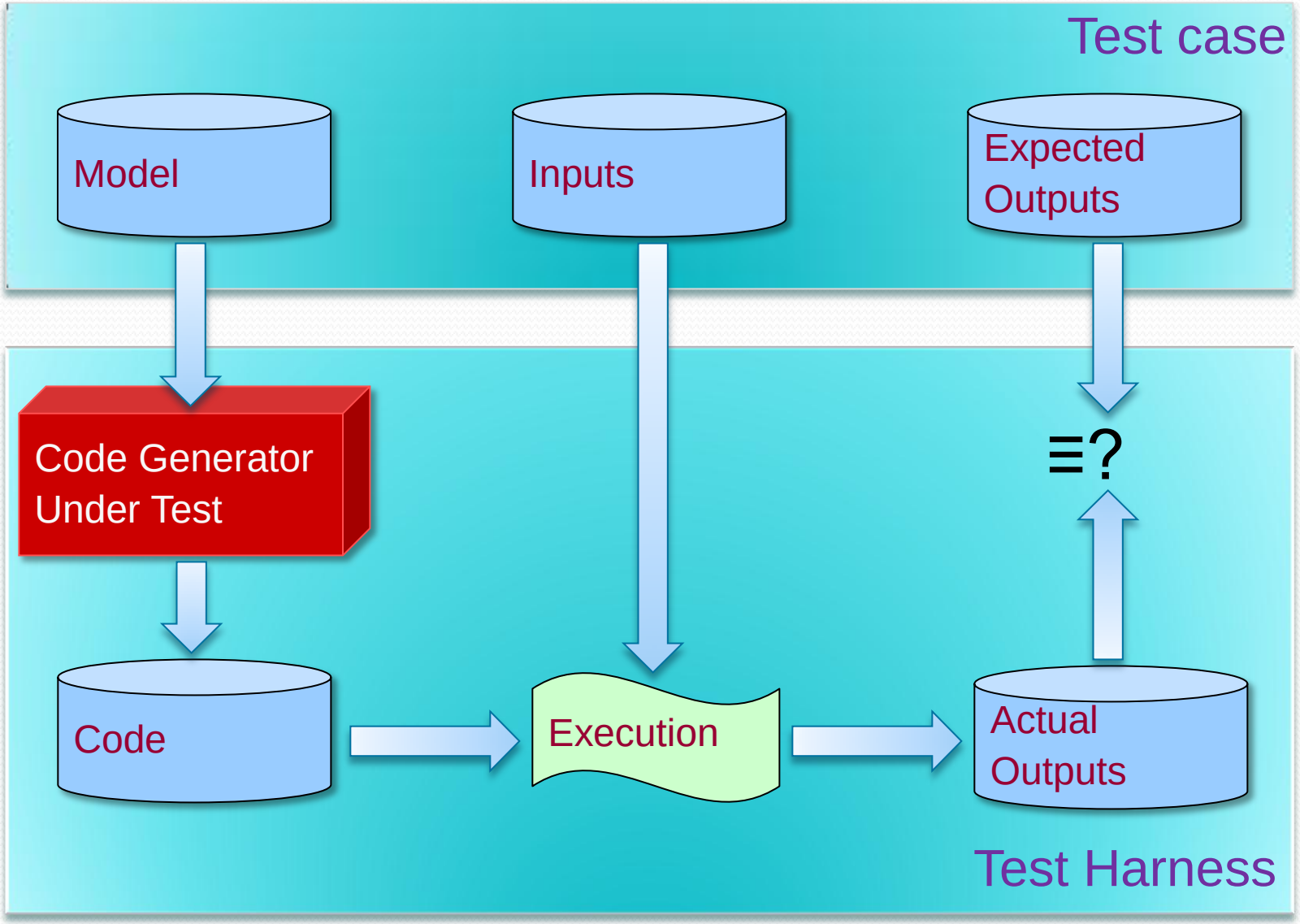
User Feedback

- Generate test-cases from proofs
- Any proof visualization techniques?
- Tabulation of all cases and reporting?
- …

Testing the Semantics



Testing the Semantics



Examples of Semantic Rules

- Semantics for a lexical analyzer

$$\frac{a \in \Sigma}{a \in \mathrm{a}}\ (\textsc{Ax-Char}) \qquad \frac{s_1 \in r_1 \quad s_2 \in r_2}{s_1.s_2 \in r_1.r_2}\ (\textsc{Dot}) \qquad \frac{}{\epsilon \in r?}\ (\textsc{Ax-Opt}) \qquad \frac{s \in r}{s \in r?}\ (\textsc{Opt})$$

$$\frac{}{\epsilon \in r^*}\ (\textsc{Ax-Star}) \qquad \frac{s \in r}{s \in r^*}\ (\textsc{Star1}) \qquad \frac{s_1 \in r \quad s_2 \in r^*}{s_1.s_2 \in r^*}\ (\textsc{Star2})$$

$$\frac{s \in r}{s \in r^+}\ (\textsc{Plus1}) \qquad \frac{s_1 \in r \quad s_2 \in r^+}{s_1.s_2 \in r^+}\ (\textsc{Plus2}) \qquad \frac{s \in r_1}{s \in r_1 \,[\!]\, r_2}\ (\textsc{Choice1}) \qquad \frac{s \in r_2}{s \in r_1 \,[\!]\, r_2}\ (\textsc{Choice2})$$

Examples of Semantic Rules

- Semantics for a simple while-language

$$\{P\}\ \mathsf{skip}\ \{P\}\ \text{[Skip]}$$

$$\{P[e/x]\}\ \mathsf{x} := \mathsf{e}\ \{P\}\ \text{[Assgn]}$$

$$\frac{\{P\}\ C_1\ \{Q\} \qquad \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1; C_2\ \{R\}}\ \text{[Seq]}$$

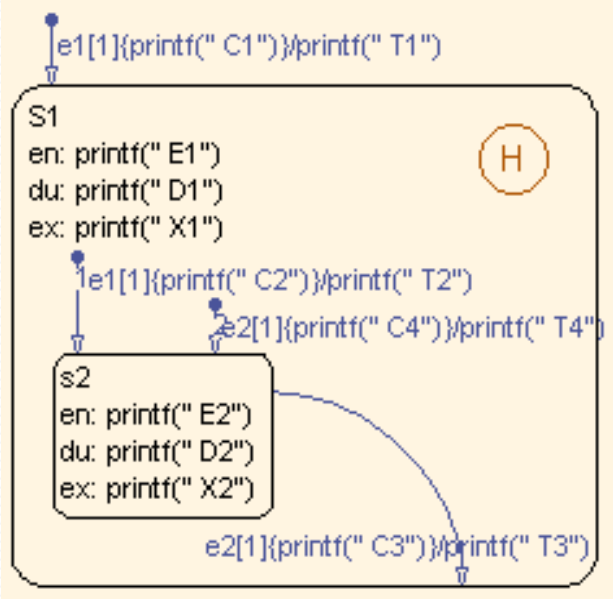
$$\frac{P \Rightarrow [\![b]\!] \qquad \{P\}\ C_1\ \{Q\}}{\{P\}\ \mathsf{if}\ b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \{Q\}}\ \text{[If-1]}$$

$$\frac{P \Rightarrow \neg[\![b]\!] \qquad \{P\}\ C_2\ \{Q\}}{\{P\}\ \mathsf{if}\ b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \{Q\}}\ \text{[If-2]}$$

$$\frac{\{P \wedge b\}\ C\ \{P\}}{\{P\}\ \mathsf{while}\ b\ \mathsf{do}\ C\ \{P \wedge \neg b\}}\ \text{[While]}$$

Examples of Semantic Rules

- Inference rules for Stateflow:



- Entering an atomic state s by a transition

$$\frac{\{P\}\ \mathsf{entryAct}(s)\ \{Q \rhd \Psi'\}}{(\!|\Psi \lhd P|\!)\ \tau \Rightarrow\!\Box\ s\ (\!|Q \rhd \Psi'|\!)}\ (\text{Atom-E})$$

- Entering an OR state by a transition, and its child state by default transition

$$\frac{\{P\}\ \mathsf{entryAct}(s)\ \{P_0 \rhd \Psi_0\} \qquad (\!|\Psi \lhd P_0|\!)\ \vDash_s \mathcal{T}_d\ (\!|P_1 \rhd \Psi_1|\!) \qquad (\!|\Psi \lhd P_1|\!)\ \Rightarrow\!\boxdot\ s_1\ (\!|Q \rhd \Psi_2|\!)}{(\!|\Psi \lhd P|\!)\ \tau \Rightarrow\!\Box\ s\ (\!|Q \rhd \cup_{k=0}^{2} \Psi_k|\!)}\ (\text{OR-dE-E})$$

Generating Test-Cases

- Generate a set of “proof-trees” based on coverage criteria
- Given a particular behaviour as a generated “proof-tree”
  - Compute possible models, inputs and outputs that give rise to the given behaviour
  - Invert semantics!

$$\dfrac{\dfrac{}{\textsc{If-1}}\textsc{Assgn} \qquad \textsc{Assgn}}{\textsc{Seq}}$$

{b}

If b then

x := e1

else

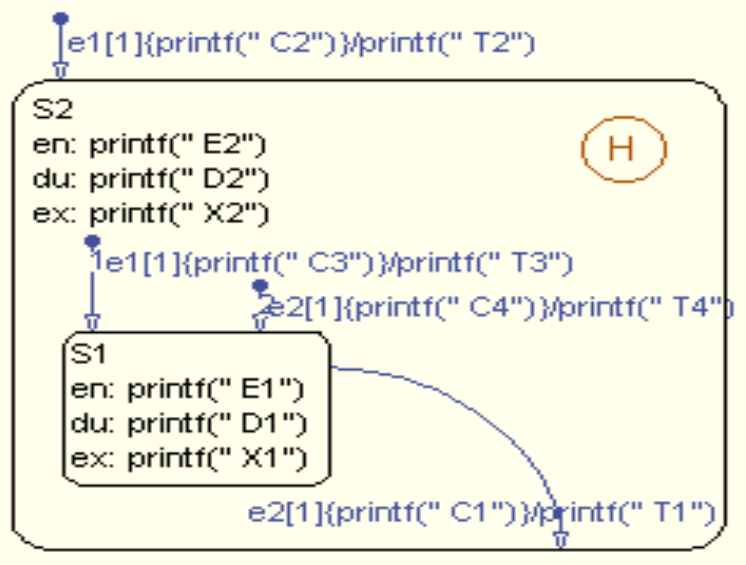
skip

x := e2

{x = e2[e1[x’/x]/x]}

Reveals Subtle Bugs/Issues

History junction bug:
Inputs: e1 e2
Expected: D2 C1 X1 T1 E1
Actual: D2 C1 X1 T1 **C4 T4** E1
Above bug in V6.2.1, fixed in V7.0



Reveals Subtle Bugs/Issues

(R, s, T)	LA by Flex	LA by JFlex
$(\{\mathtt{b}^*/(\mathtt{a}\vert\mathtt{b}), \mathtt{a}\vert\mathtt{b}^*\}, a, \{(a,2)\})$	diverge	abort
$(\{\mathtt{b}^*/(\mathtt{a}\vert\mathtt{b}), \mathtt{a}\vert\mathtt{b}^*\}, b, \{(b,2)\})$	$\{(b,1)\}$	$\{(b,1)\}$
$(\{\mathtt{b}^*/(\mathtt{a}\vert\mathtt{b}), \mathtt{a}\vert\mathtt{b}^*\}, aa, \{(a,2),(a,2)\})$	diverge	abort
$(\{\mathtt{b}^*/(\mathtt{a}\vert\mathtt{b}), \mathtt{a}\vert\mathtt{b}^*\}, ab, \{(a,2),(b,2)\})$	diverge	abort
$(\{\mathtt{b}^*/(\mathtt{a}\vert\mathtt{b}), \mathtt{a}\vert\mathtt{b}^*\}, ba, \{(b,1),(a,2)\})$	diverge	diverge
$(\{\mathtt{b}^*/(\mathtt{a}\vert\mathtt{b}), \mathtt{a}\vert\mathtt{b}^*\}, bb, \{(b,1),(b,2)\})$	$\{(bb,1)\}$	$\{(bb,1)\}$

Questions

References

- ***An Axiomatic Semantics for Stateflow.*** In preparation
- ***Translation Validation for Stateflow to C.*** Under submission
- ***CoGenTe: A Tool for Code Generator Testing.*** IEEE/ACM International Conference on Automated Software Engineering (ASE'10), Antwerp, Belgium, 2010.
- ***Behaviour Directed Testing of Auto-code Generators.*** IEEE International Conference on Software Engineering and Formal Methods (SEFM'08), Cape Town, SA, 2008.
- ***Verification of Model Processing Tools.*** Safety-Critical Systems Session, SAE World Congress & Exhibition (**SAE'08**), Detroit, USA, 2008.
- ***How to Test Program Generators? A Case Study using flex.*** IEEE International Conference on Software Engineering and Formal Methods (SEFM'07), London, UK, 2007.
- ***Testing Model-Processing Tools for Embedded Systems.*** IEEE International Real-Time and Embedded Technology and Applications Symposium (**RTAS'07**), Bellevue, WA, USA, April 2007.