GeneAuto for Ada and SPARK

A verifying model compiler

Matteo Bordin
bordin@adacore.com

Franco Gasperoni
gasperoni@adacore.com

GeneAuto2 meeting (Toulouse)
September 2009
Model Compilers: State-of-the-Art

- **Traditional model compiler**
  - Rarely added value (model = graphical coding)

- **Properties verifiable at model level**
  - Show they hold at source level: qualified code generator
  - Show they hold in the executable: compiler qualification kit
  - Assume they are respected during execution

- **Properties *non*-verifiable at model level**
  - Platform-dependent properties (overflows, stack size, wcet, ...)
  - No feedback in the model
Verifying Model Compilers: High-level Vision

- **Properties for an intermediate representation**
  - Formal semantics
  - Executable semantics
  - Show properties are formally preserved
  - Complement V&V activities

- **Our first experiment: Simulink**
  - Need open technology to rely on: GeneAuto
  - Emphasis on the intermediate representation for:
    - Correctness (property preservation + complement V&V)
    - Efficiency
    - Traceability (model→source & source→object)
SPARK: target language of GeneAuto/Ada

package My_Block

--# own Input_Ports, Output_Ports, Is_initialized;

is

procedure Initialize;

--# global out Is_initialized;
--# post Is_initialized;

procedure Pass_Data (D : Data);

--# global out Input_Ports;
--# pre Is_initialized;
--# derives Input_Ports from D;

procedure Compute;

--# global in Input_Ports; out Output_Ports;
--# pre Is_initialized;
--# derives Output_Ports from Input_Ports;
GeneAuto/Ada: a Verifying Model Compiler (I)

• What we can *formally prove* with SPARK
  - Absence of run-time errors:
    - Overflows, underflows
    - Array-out-of-bounds
    - Data/Information flow errors (uninitialized variables, ineffective statements, ...)
  - Partial correctness
    - Hoare-like pre/post conditions

• Major advantages:
  - Model translation is *formally* void of errors
  - The *formal representation (in SPARK)* is *executable* and *efficient*
Use SPARK as an intermediate representation

**ADDED VALUE:** Complement model-level V&V by automated source code analysis

Round-trip of analysis results
A typical day in GeneAuto/Ada...

This is the C code we generate: how would the SPARK version look like?

This is the corresponding SPARK code. I've found:
- Possible source of overflow
- Unused parameters
- Access to uninitialized variables
- ...

Yep, you’re right. We discovered:
- A problem in GeneAuto requirements
- A problem in GeneAuto implementation
- A limit in model-level verification
Qualifying GeneAuto/Ada

• We considered OpenOffice... for almost 3 full minutes...

• We needed:
  – Agile framework supporting distributed artifacts manipulation
  – Easy revision control (at least 4 developers)
  – Tracking of:
    – Actions!
    – Approved requirements
    – Implemented requirements
    – Open questions
  – Traceability evidence
FitNesse: a Prototypal Qualifying Machine

- **A qualifying machine:**
  - Infrastructure keeping track of each atomic action
  - Analyze artifacts deployment/history to discover:
    - Workflow was (not) followed
    - Traceability problems
  - Agile user interface (possibly web-based)

- **Our current choice: FitNesse**
  - A tool for test-driven development (agile paradigm)
  - Slightly modified to better fit our needs
FitNesse: a Prototypal Qualifying Machine (II)

Artifact Editing & Action tracking

Managing Open Questions
Future Directions

- Increase the number of supported blocks
- Pursuing the verifying model compiler vision
- Investigate integration with UML, SysML, MARTE/AADL
- Improve qualifying machine
- Opening the project: Open-DO
- Business model (OPEES)