ABV – A Verifier for the Architecture Analysis and Design Language (AADL)

Stefan Björnander, Cristina Seceleanu
Kristina Lundqvist, Paul Pettersson

Mälardalen University
Sweden
Outline

• Motivation
• Background
• Our Formal Analysis Framework
  • The Denotational Semantics for AADL Elements
  • The Implementation in Standard ML
  • The ABV Model Checker
  • Illustrative Examples
• Conclusions and Future Work
Embedded Systems

• Microprocessor-based systems embedded into larger systems.
• 99% of all software.
• Everywhere around us, from mp3-players to nuclear plants.
• Often expected to run for years without failure.
What Can Go Wrong?

• The Mercury Space Shuttle
  • The Famous Fortran Bug: “DO 10 I=1.10” instead of “DO 10 I=1,10”.

• The Mariner 1 Flight
  • Its mission was to carry a probe to Venus.
  • Due to a spelling error in the algorithm specification, the mission was aborted and the shuttle destroyed after six minutes.
Motivation

Software Design Issues

• Abstraction and Refinement
• Algorithms and Data Structures
• Modularity and Information Hiding
• Software Architecture
Motivation

Software Design Issues

• Abstraction and Refinement
• Algorithms and Data Structures
• Modularity and Information Hiding
• Software Architecture
Motivation

Software Architecture

• A system is the set of structures needed to reason about the system, both its hardware and software.
• Model-Driven Architecture (MDA).
• Architecture Description Languages (ADLs).
  • Formal Verification
Motivation

Software Architecture

- A system is the set of structures needed to reason about the system, both its hardware and software.
- Model-Driven Architecture (MDA).
- Architecture Description Languages (ADLs).
  - Formal Verification
Motivation

AADL

- A SAE (Society of Automotive Engineers) standard.
- Popular in the automobile and avionics industry.
- Models both the hardware and software of the system. Supports encapsulation and inheritance.

However:
- Has not yet, in total, been formally defined.
- Does not support formal verification.
Motivation

AADL

• A SAE (Society of Automotive Engineers) standard.
• Popular in the automobile and avionics industry.
• Models both the hardware and software of the system. Supports encapsulation and inheritance.
• However:
  • Has not yet, in total, been formally defined.
  • Does not support formal verification.
Formal Verification

- An act of proving or disproving suitable to guarantee the correctness of the system.
- Using rigorous mathematical models, most often with assistance of a computer.
  - True/False answers.
  - Number answers.
- Formal Verification Methods
  - Theorem Proving
  - Model Checking
Motivation

Formal Verification

• An act of proving or disproving suitable to guarantee the correctness of the system.
• Using rigorous mathematical models, most often with assistance of a computer.
  • True/False answers.
  • Number answers.
• Formal Verification Methods
  • Theorem Proving
  • Model Checking
Outline

- Motivation
- **Background**
- Our Formal Analysis Framework
  - The Denotational Semantics for AADL Elements
  - The Implementation in Standard ML
  - The ABV Model Checker
  - Illustrative Examples
- Conclusions and Future Work
The AADL Behavior Annex

• States
• Transitions
• State Variables with Initializations.
Computation Tree Logic (CTL)

- Branching-time temporal logic.
- Models time as a tree structure with a non-determined future.

- Properties
  - Safety (all global)
  - Liveness (all eventually)
  - Reachability (exists eventually)
  - Deadlock
  - Mutual Exclusion
Outline

• Motivation
• Background
• Our Formal Analysis Framework
  • The Denotational Semantics for AADL Elements
  • The Implementation in Standard ML
  • The ABV Model Checker
  • Illustrative Examples
• Conclusions and Future Work
The Formal Analysis Framework

- Denotational Semantics
  - Support Model Checking with CTL
- Implementation in Standard ML
  - Line-to-line translation
- The ABV Tool
  - Performs model checking on CTL properties on AADL models.
  - User-friendly graphical tool.
The Denotational Semantics

Denotational Semantics for AADL and its Behavior Annex

- Formally defines a subset of AADL and its Behavior Annex.
- Supports Model Checking.
- Implemented in Standard ML.
The Denotational Semantics

The AADL subset

- System
- System implementation
- Subcomponent
- Connection
The AADL Syntax

Model ::= System SystemImpl
System ::= System System
      | system Identifier SystemBody end ;
SystemBody ::= OptionalFeatures OptionalAnnex
OptionalFeatures ::= features Feature
      | ε
Feature ::= Feature Feature
      | Identifier : in event port ;
      | Identifier : out event port ;

SystemImpl ::= system implementation Identifier . Identifier SystemImplBody end ;
SystemImplBody ::= OptionalSubcomponents OptionalConnections
OptionalSubcomponents ::= subcomponents Subcomponent
      | ε
Subcomponent ::= Subcomponent Subcomponent
      | Identifier : system Identifier ;
OptionalConnections ::= connections Connection
      | ε
Connection ::= Connection Connection
      | event port Identifier . Identifier -> Identifier . Identifier ;
The Denotational Semantics

The Behavior Annex Syntax

- Formalization of the whole annex

```
Annex ::= annex Identifier {** OptionalStateVariables
    OptionalInitializations OptionalStates
    OptionalTransitions **} ;
OptionalStateVariables ::= state variables StateVariables
    | ε
StateVariables ::= StateVariable StateVariable
    | Identifier : integer ;
OptionalStates ::= states State
    | ε
State ::= State State
    | Identifier : initial state ;
    | Identifier : state ;
OptionalInitializations ::= initializations Action
    | ε
OptionalTransitions ::= transitions Transition
    | ε
Transition ::= Transition Transition
    | Identifier -[ Expression ]-> Identifier
    | OptionalActions
OptionalActions ::= { Action }
    | ;
Action ::= Action Action
    | Identifier := Expression ;
    | Identifier ! ;
Expression ::= Identifier
    | Expression ArithmeticOperator Expression
ArithmeticOperator ::= + | - | * | / |
```
The Standard ML Implementation

• **Purpose**
  • Automated model checking on CTL Properties.

• **Motivation for Standard ML**
  • Small gap between Denotational Semantics and Standard ML.
  • They are both based on the lambda-calculus.
  • Constructs:
    • if-then-else-statement
    • let-in-blocks.
  • Both supports recursively defined data types.
The **Feature** Semantic Function

\[
\begin{align*}
\text{feature} : \text{Feature} \rightarrow \text{Table} \\
\text{feature} \ [F_1 \ F_2] = \\
\quad \text{let} \ \text{port_table}_1 = \text{feature} \ F_1 \ \text{in} \\
\quad \text{let} \ \text{port_table}_2 = \text{feature} \ F_2 \ \text{in} \\
\quad \text{table_merge} \ \text{port_table}_1 \ \text{port_table}_2 \\
\text{feature} \ [I : \text{in event port}] = \\
\quad \text{table_set} \ I \ (\text{boolean} \ \text{false}) \ \text{table_empty} \\
\text{feature} \ [I : \text{out event port}] = \\
\quad \text{table_set} \ I \ (\text{boolean} \ \text{false}) \ \text{table_empty}
\end{align*}
\]

(*) \ val \ \text{feature} = \text{fn} : \text{Feature} \rightarrow \text{Value} \ \text{Table} *)

\[
\begin{align*}
\text{fun} \ \text{feature} \ (\text{features} \ (F_1, \ F_2)) = \\
\quad \text{let} \ \text{val} \ \text{port_table}_1 = \text{feature} \ F_1 \ \text{in} \\
\quad \text{let} \ \text{val} \ \text{port_table}_2 = \text{feature} \ F_2 \ \text{in} \\
\quad \text{table_merge} \ \text{port_table}_1 \ \text{port_table}_2 \ \text{end} \ \text{end} \\
\quad | \ \text{feature} \ (\text{inport} \ I) = \\
\quad \quad \text{table_set} \ I \ (\text{boolean_value} \ \text{false}) \ \text{table_empty} \\
| \ \text{feature} \ (\text{outport} \ I) = \\
\quad \quad \text{table_set} \ I \ (\text{boolean_value} \ \text{false}) \ \text{table_empty};
\end{align*}
\]
The Standard ML Implementation

- **The AADL-to-ML Parser**
  - Translates the AADL source code and CTL property specification to Standard ML format.

- **Modules**
  - Symbol Table and Type Checking
  - State Space Tree Generator
  - CTL Property Evaluator
The ABV Tool

The AADL and its Behavior Annex Verifier (ABV)

- A tool for model checking of CTL properties.
- Implemented in Standard ML, based on the Denotational semantics.
- Encapsulated in an Eclipse plug-in.
Example 1: Mutual Exclusion

Safety Property

system SubSystem1
features
  CriticalEnter: in event port;
  CriticalLeave: out event port;
annex SubSystem1 {**
initializations
  CriticalLeave!;
states
  Waiting: initial state;
  Critical: state;
transitions
  Waiting -[CriticalEnter?] -> Critical;
  Critical -[true] -> Waiting {CriticalLeave!;}
**};
end SubSystem1;
...
system implementation MainSystem.impl
subcomponents
  subSystem1: system SubSystem1;
  subSystem2: system SubSystem2;
connections
  event port subSystem1.CriticalLeave -> subSystem2.Critical
  event port subSystem2.CriticalLeave -> subSystem1.Critical
end MainSystem.impl;
Example 1: Mutual Exclusion

Safety Property

- We want to prove that the subsystem1 and subsystem2 subcomponents never reach their critical sections at the same time.

- CTL Safety Property:
  all global not (subSystem1.Critical and subSystem2.Critical)
Example 2: The Production Cell System

Behavior Property

- Based on an automated manufacturing system (first described by Lewerentz and Lindner in 1995).

- Functionality:
  - Moves a block through the system.
  - Presses the block.
  - Deposits blocks on belt.

As depicted by Martin Ouimet, 2007.
Example 2: The Production Cell System

Behavior Property

- *storer* subcomponent
  - *StoredBlocks*: counts the number of processed blocks.

- We want to prove that a block added at the beginning reaches the end.
- CTL Liveness Property: all eventually storer.StoredBlocks = 1
Example 2: The Production Cell System

Architectural Property

- We want to prove that a signal does not become overwritten before it is read.

- CTL Safety Property:
  \[ \text{all global feedBelt.InBlockReady\_count \leq 1} \]
Example 3: The Wolf, Goat, and Cabbage

• Initial State: \texttt{wgc.BWGC_{}}

• CTL Reachability Property: exists eventually \texttt{wgc._BWGC} and (\texttt{wgc.WAteG = 0}) and (\texttt{wgc.GAteC = 0})
Example 3: The Wolf, Goat, and Cabbage

- Scalability
- Trace Generation

Log File

0:
Transition: wgc.BWGC_ -> wgc.WC_BG
State: wgc = WC_BG, WAtG = 0, GAteC = 0

1:
Transition: wgc.WC_BG -> wgc.BWC_G
State: wgc = BWC_G, WAtG = 0, GAteC = 0

2:
Transition: wgc.BWC_G -> wgc.C_BWG
State: wgc = C_BWG, WAtG = 0, GAteC = 0

3:
Transition: wgc.C_BWG -> wgc.BGC_W
State: wgc = BGC_W, WAtG = 0, GAteC = 0

4:
Transition: wgc.BGC_W -> wgc.G_BWC
State: wgc = G_BWC, WAtG = 0, GAteC = 0

5:
Transition: wgc.G_BWC -> wgc.BG_WC
State: wgc = BG_WC, WAtG = 0, GAteC = 0

6:
Transition: wgc.BG_WC -> wgc._BWGC
State: wgc = _BWGC, WAtG = 0, GAteC = 0
Outline

• Motivation
• Background
• Our Formal Analysis Framework
  • The Denotational Semantics for AADL Elements
  • The Implementation in Standard ML
  • The ABV Model Checker
  • Illustrative Examples
• Conclusions and Future Work
Conclusions and Future Work

Conclusions

- The ABV Tool for Formal Verification of AADL Models with CTL Properties
- Exemplified on Three Illustrative Systems
- Promising Scalability
- Provides Insight on Architecture and Related Behavior
Future Work

• At present: the state space tree becomes completely generated before evaluation.
• Future: should be possibly to generate and evaluate the state space tree “on-the-fly”.
• Add time annotation to the transitions in order to perform real-time analysis.
• Other architecture description languages, such as MARTE or EAST-ADL as source language.
Questions and Suggestions?

www.idt.mdh.se/~sbr02
stefan.bjornander@mdh.se

Thank You!