MÄLARDALEN UNIVERSITY
SWEDEN

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

Stefan Björnander, Cristina Seceleanu
Kristina Lundqvist, Paul Pettersson

Mälardalen University
Sweden

SWEDISH FOUNDATION for STRATEGIC RESEARCH

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

PROGRESS
The **PROGRESS** Centre for Predictable Embedded Software Systems

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

MÄLARDALEN UNIVERSITY
SWEDEN

Motivation

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.

Space Shuttle Atlantis

Computerized Toaster

Motivation

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

MÄLARDALEN UNIVERSITY
SWEDEN

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.

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

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

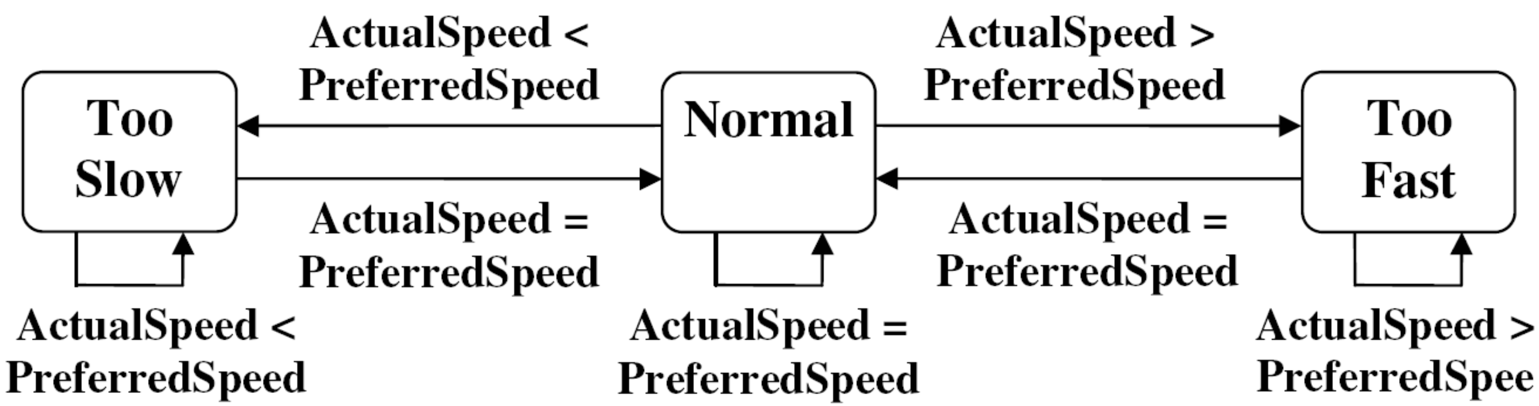
Outline

- 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

Background

The AADL Behavior Annex

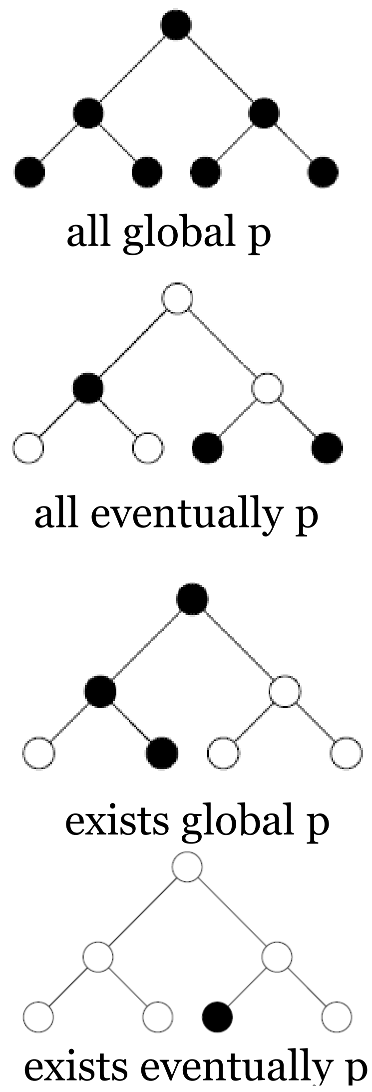
- States
- Transitions
- State Variables with Initializations.



Background

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



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

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 Denotational Semantics

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 ::= + | − | ∗ | / |
```

MÄLARDALEN UNIVERSITY SWEDEN

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 Standard ML Implementation

The *Feature* Semantic Function

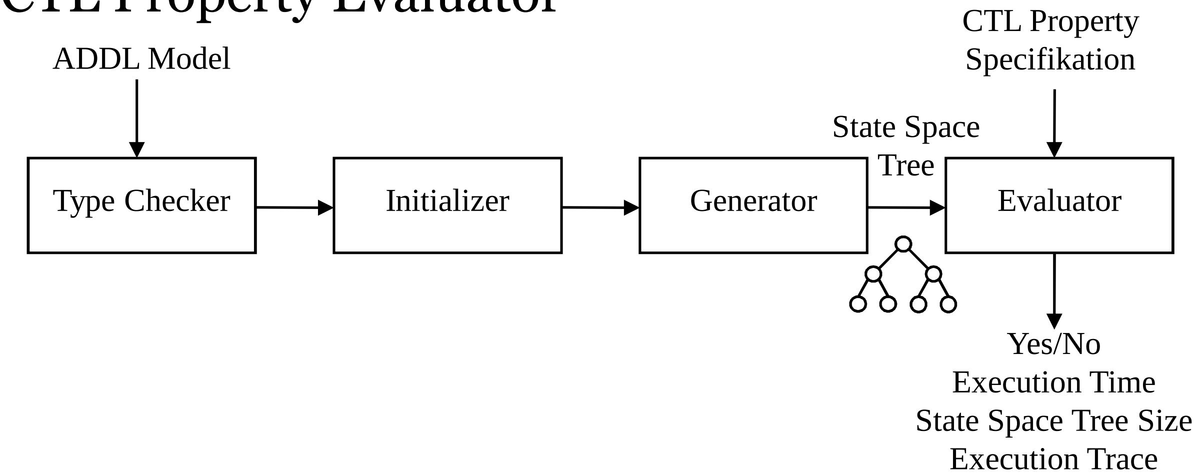
$$
\begin{aligned}
&\mathit{feature} : \text{Feature} \rightarrow \text{Table}\\
&\mathit{feature}\ [\![F_1\ F_2]\!] =\\
&\quad \textbf{let}\ \mathit{port\_table}_1 = \mathit{feature}\ F_1\ \textbf{in}\\
&\quad \textbf{let}\ \mathit{port\_table}_2 = \mathit{feature}\ F_2\ \textbf{in}\\
&\quad\ \ \mathit{table\_merge}\ \mathit{port\_table}_1\ \mathit{port\_table}_2\\
&\mathit{feature}\ [\![I : \textbf{in event port}]\!] =\\
&\quad \mathit{table\_set}\ I\ (\mathit{boolean}\ \textbf{false})\ \mathit{table\_empty}\\
&\mathit{feature}\ [\![I : \textbf{out event port}]\!] =\\
&\quad \mathit{table\_set}\ I\ (\mathit{boolean}\ \textbf{false})\ \mathit{table\_empty}
\end{aligned}
$$

```
(* val feature = fn : Feature -> Value Table *)
fun feature (features (F1, F2)) =
     let val port_table1 = feature F1 in
     let val port_table2 = feature F2 in
         table_merge port_table1 port_table2 end end
  | feature (inport I) =
     table_set I (boolean_value false) table_empty
  | feature (outport I) =
     table_set I (boolean_value false) table_empty;
```

MÄLARDALEN UNIVERSITY SWEDEN

The Standard ML Implementation

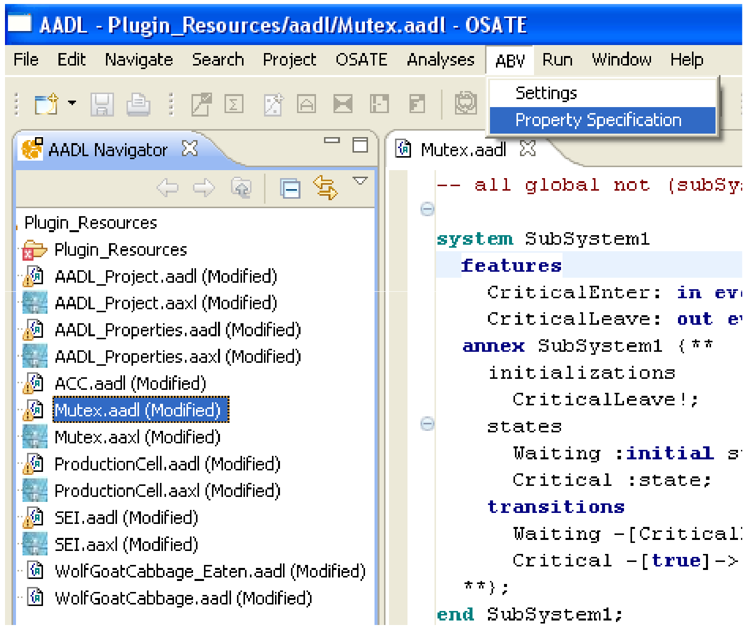
- The AADL-to-ML Parser
  - Translates the AADL source code and CTL property specification to Standard ML format.
- Modules
  - Symbol Table and Type Checking
  - State Space Tree Generator
  - CTL Property Evaluator



The ABV Tool

The AADL and its Behavior Annex Verifier (ABV)

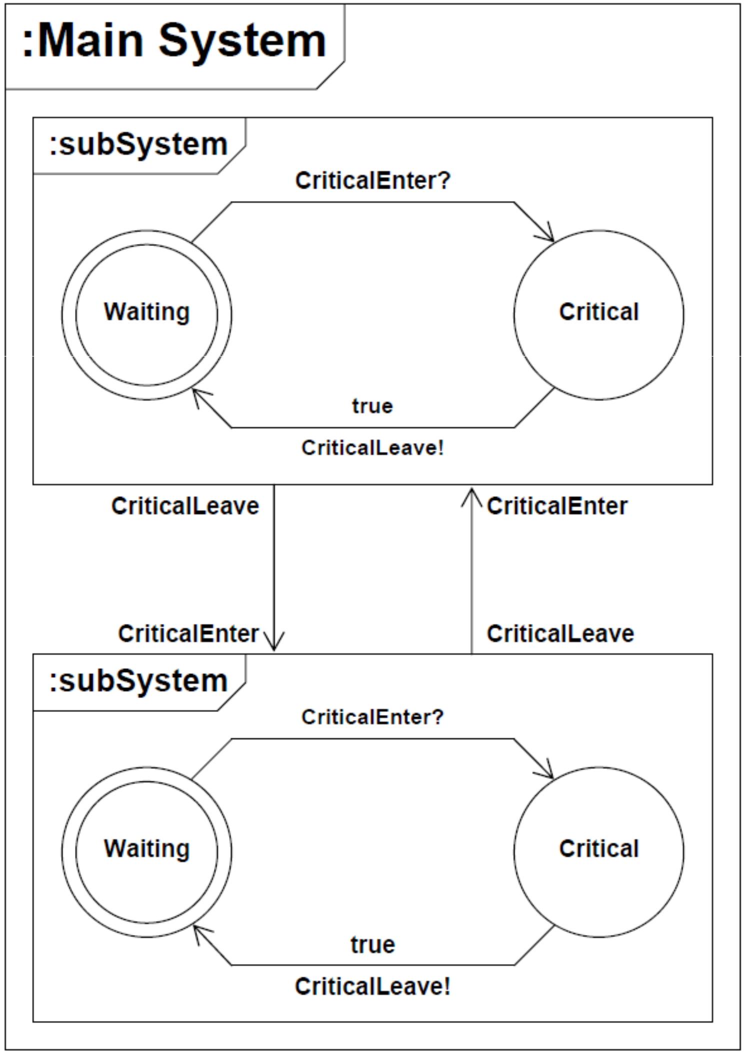
- A tool for model checking of CTL properties.
- Implemented in Standard ML, based on the Denotational semantics.
- Encapsulated in an Eclipse plug-in.



Example 1: Mutual Exlusion

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;
```



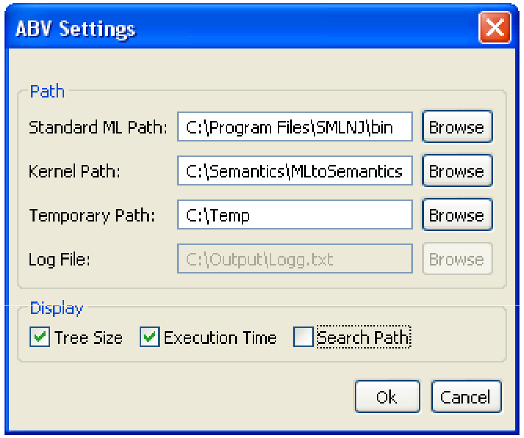
Example 1: Mutual Exlusion

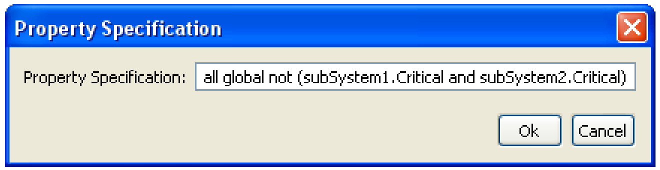
Safety Property

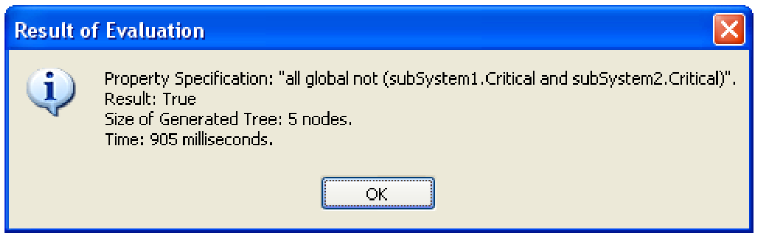
- We want to prove that the *subsystem1* and *subsystem2* subcomponents never reach their critical sections at the same time.

- CTL Safety Property:

all global not (subSystem1.Critical and subSystem2.Critical)





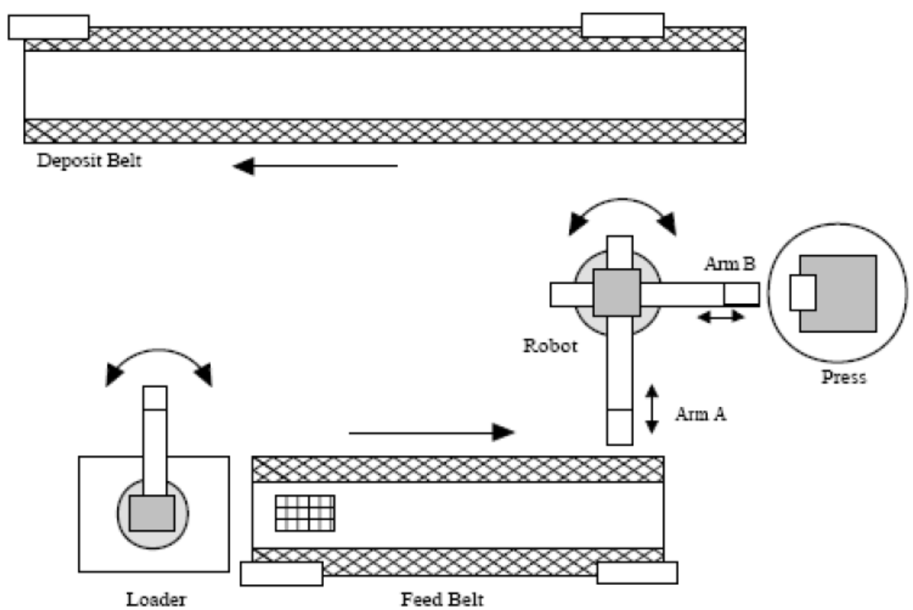


Example 2: The Production Cell System

Behavior Property

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

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



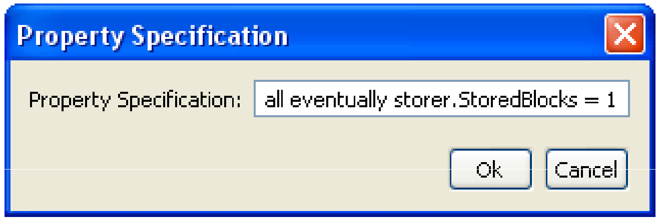
As depicted by Martin Ouimet, 2007.

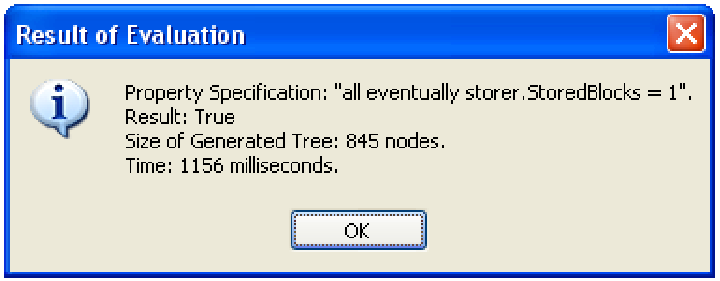
MÄLARDALEN UNIVERSITY SWEDEN

Example 2: The Production Cell System

Behavior Property

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





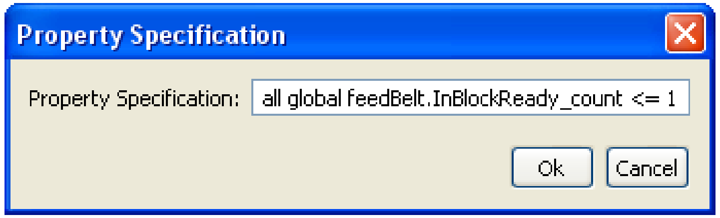
- We want to prove that a block added at the beginning reaches the end.
- CTL Liveness Property: all eventually storer.StoredBlocks = 1

MÄLARDALEN UNIVERSITY SWEDEN

Example 2: The Production Cell System

Architectural Property

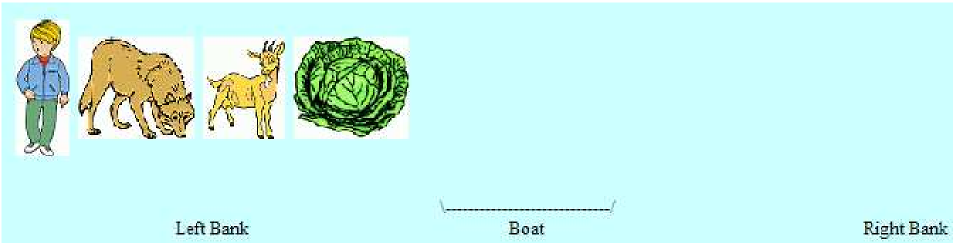
- We want to prove that a signal does not become overwritten before it is read.



- CTL Safety Property:

  all global feedBelt.InBlockReady_count <= 1

Example 3: The Wolf, Goat, and Cabbage

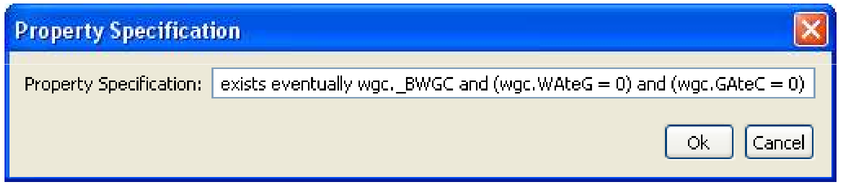


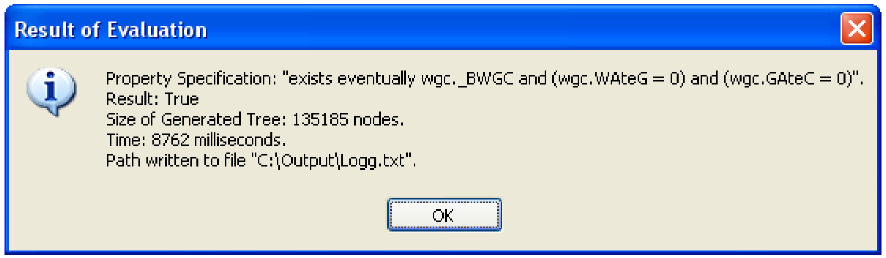
- Initial State: wgc.BWGC_
- CTL Reachability Property:
  exists eventually wgc._BWGC
  and (wgc.WAteG = 0)
  and (wgc.GAteC = 0)

```
system WolfGoatCabbage
 annex WolfGoatCabbage_Annex
 {**
  state variables
   WAteG, GAteC : integer;

  initializations
   WAteG := 0; GAteC := 0;

  states
   BWGC_ : initial state;
   BWG_C, BWC_G, BGC_W, BW_GC, BG_WC, BC_WG,
   B_WGC, WGC_B, WG_BC, WC_BG, GC_BW, W_BGC,
   G_BWC, C_BWG, _BWGC : state;

  transitions
   BWGC_ -[true]-> WGC_B;
   BWGC_ -[true]-> GC_BW {GAteC := 1;}
   …
 **};
end WolfGoatCabbage;

system Main
end Main;

system implementation Main.impl
 subcomponents
  wgc : system WolfGoatCabbage;
end Main.impl;
```

Example 3: The Wolf, Goat, and Cabbage

- Scalability
- Trace Generation





Log File

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

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

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

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

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

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

6:
Transition: wgc.BG_WC -> wgc._BWGC
State: wgc = _BWGC, WAteG = 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

Conclusions and Future Work

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.

Thank You!

Questions
and
Suggestions?

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