Specification and Verification with SCR

Part II: Toolset and Verification

Building Software Requirements: Apply Practical Formal Methods

- Specify the system precisely
- Apply “consistency checking”
- Simulate the system behavior
- Verify the specifications (model checking)
- Verify the specifications (theorem proving)

Increasing assurance cost and expertise
Tools for Specifying and Analyzing Requirements

The SPECIFICATION EDITOR supports the creation of formal requirements specs.

The DEPENDENCY GRAPH BROWSER displays the relationship between variables in spec.

The CONSISTENCY CHECKER ensures that the specs satisfy application-independent properties.

The SIMULATOR checks that the specs satisfy their intent.

The VERIFIER allows users to check application properties.
Spec Editor Organizes the Requirements into Dictionaries and Tables

Dictionaries

Tables

A Variable Dictionary

Lists each monitored variable, controlled variable, and term along with its type, initial value, and required precision.
A Mode Class Dictionary

- Lists each mode class along with its:
  - member modes
  - initial mode

Constant and Type Dictionaries
Dependency Graph Browser

Dependency Graph for Safety Injection System

Consistency Checking

- Tests *application-independent* properties -- consistency of spec with our requirements model
- Most individual checks require a single table and the type definitions

- Proper syntax
- Type correctness
- Completeness of entity definitions
- Disjointness
- Coverage
- Initial values
- Reachability
- Lack of circularity
Detection of a Cycle

A Circular Definition

"Overridden" is defined in terms of "Pressure" and "Pressure" is defined in terms of "Overridden"

Simulator Display with Four Pending Input Events

System
State
Pending
Events
Simulator Log and Table Defining a Variable of Interest

Clicking on a variable in the log displays the defining table with the rule that caused the variable to change highlighted.

---

Writing Assertions

- Assertions are properties that the specification is supposed to satisfy
  - In SCR: invariant properties involving a single state or two consecutive states.

Examples:

- “If the water pressure is too low while the system is not being blocked, safety-injection will be on”
  \[
  \text{Block=OFF AND WaterPres < Low} \Rightarrow \text{Safety\_Injection=ON}
  \]

- “If the water pressure becomes too low, the system will transition into mode TooLow”
  \[
  (\text{WaterPres} \geq \text{Low} \text{ AND WaterPres'} < \text{Low}) \Rightarrow \text{Pressure'}=\text{TooLow}
  \]
Assertions (Cont’d)

Assertions and Assumptions:
- environmental assumptions (what the system assumes to be true about the environment) (NAT)
- specification assertions (what the system should establish to be true) (ASSERT)

Want to ensure that if the system satisfies NAT, then our requirements are sufficient to prove ASSERT.

Can do so either during simulation (for each step, verify the assertions) or during verification. What is the difference?

Assertion Checking during Simulation

At step 3, the simulator has detected a violation of an assertion. Clicking on the warning msg. in the log highlights the failed assertion.
Verification of SCR Specifications

Expensive failures, simple properties
- **F-16 Landing gear**: In system test, pilot raises landing gear with aircraft on the runway
- **Rogue HARM**: In live fire trials, HARM missile arms and locks on releasing aircraft radar and must be destroyed
- **Patriot Missile**: System fails to react to SCUD attack

To reduce such problems, one can apply verification to the requirements specification

Two different approaches
- Model checking
- Theorem proving

Approaches to Verification

<table>
<thead>
<tr>
<th>Theorem Proving: Establish properties by supplying a proof</th>
<th>Model Checking: Check finite state model for properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>General, powerful</td>
<td>Restricted, Limited to finite domains</td>
</tr>
<tr>
<td>No restrictions on size of domain</td>
<td>Provides deep insight</td>
</tr>
<tr>
<td>Provides deep insight</td>
<td>Provides little insight</td>
</tr>
<tr>
<td>Requires mathematical training and theorem proving skills</td>
<td>Automatic</td>
</tr>
<tr>
<td>Minimal feedback when proof fails</td>
<td>Requires little training but STATE EXPLOSION!!!</td>
</tr>
<tr>
<td>Provides counterexamples</td>
<td></td>
</tr>
</tbody>
</table>

17

18
Verification Using Model-Checking

**Sample Properties**

1. \( \text{Reset} = \text{On} \land \text{Pressure} \neq \text{High} \Rightarrow \neg \text{Overridden} \)
2. \( \text{Reset} = \text{On} \land \text{Pressure} = \text{TooLow} \Rightarrow \text{SafetyInjection} = \text{On} \)
3. \( \text{Block} = \text{Off} \land \text{Pressure} = \text{TooLow} \Rightarrow \text{SafetyInjection} = \text{On} \)
4. “The altitude engage mode will be armed only when the flight path angle select mode is engaged.”

---

**Verification of SCR Specifications (History)**

- J. Atlee, J. Gannon: 1992-present. Use first MSB and then SMV to check correctness of SCR mode transition tables. Correctness criteria expressed in CTL.
- R. Bharadwaj, C. Heitmeyer: 1996-present. Use Spin and SMV to check state and transition invariants of complete SCR specifications (part of the SCR Toolset)
- M. Chechik: 1998-present. Use SMV to check correctness of complete SCR specifications. Correctness criteria expressed in CTL. Counter-examples represented in SCR.
Verification Via Model-Checking

Verifying SCR Specifications Using State Exploration

- Represent a system as the composition of a non-deterministic environment that generates monitored events and a deterministic system that generates externally visible changes in controlled quantities
- Sometimes need to abstract to enable effective verification.
  Types of abstractions:
  - eliminate irrelevant entities (automatic)
  - use abstract versions of the monitored variables (need a bit of user input)
- Violation results in the “counter-example” which can be validated through the simulator (validation is needed if the abstraction is not complete).
Abstraction Method 1 (Slicing): Eliminate Irrelevant Variables

1. Include in a set $O$ all variables that appear in the formula
2. Take the reflexive and transitive closure $O^*$ of the dependency sets of all variables in $O$

$$\text{Reset} = \text{On} \land \text{Pressure} \neq \text{High} \Rightarrow \neg \text{Overridden}$$

$O = \{\text{Reset}, \text{Pressure}, \text{Overridden}\}$

$O^* = \{\text{Block}, \text{Reset}, \text{WaterPres}, \text{Pressure}, \text{Overridden}\}$

At user request, the SCR toolset applies this method automatically.

Abstraction Method 2: Use Existing Abstractions of Monitored Variables

$$\text{Reset} = \text{On} \land \text{Pressure} \neq \text{High} \Rightarrow \neg \text{Overridden}$$

In this example, we can use the mode class $\text{Pressure}$ as an abstract model of $\text{WaterPres}$

The concrete variable $\text{WaterPres}$ can have infinitely many values.
Summary

SCR provides the basis for an industrial-strength formal method for requirements specification. The method is:

- **formal** - it has a solid mathematical foundation
- **easy to use** - detects many common errors automatically
- **scalable** - level of effort/complexity does not grow exponentially with the size of the specification
- **cost-effective** - evidence suggests that automated analysis provides early detection of errors people miss (at low cost)
- **extensible** - provides sound foundation for extending benefits to rest of life cycle.

Summary (Cont’d)

- SCR allows to specify and validate requirements for reactive high-assurance systems. These are model-based.
- Light-weight formalism, separation of concerns.
- Need to verify properties against the created models:
  - Model-Checking (Spin in SCRTool, SMV - later in this course)
  - Theorem-proving (prover for Hehner’s method is done later in this course)