Specification and Verification with SCR

Part I: Introduction, simple examples, formal model

Why write specifications?

“Real programmers don’t write specs -- users should consider themselves lucky to get any programs at all and take what they get...”

Common Wisdom
1983

The following lecture notes are adopted from an SCR tutorial given at COMPASS’97 by C. Heitmeyer and S. Faulk. The SCRTool refers to the tool distributed by the Naval Research Lab.
Outline

Part I.
◆ Introduction to specification languages and SCR
◆ Evolution and examples of industrial use
◆ Main concepts
  – Example: safety-injection system
◆ Formal semantics of SCR

Part II.
◆ SCRTool (also known as “SCR*”)
◆ Verification with SCRTool
◆ Summary

Formal Methods in Computer Systems

◆ Formal methods provide mathematically-based techniques used to specify, develop and verify properties of (software and hardware) systems.
◆ A formal method should possess a set of guidelines or a “style sheet”:
  – when can the method be applied
  – how can the method be applied most effectively
◆ Tangible output - formal requirements specification
Users of formal specs

what does this program do?

does this program satisfy this specification?

System Documentation

- specification is a description alternative to system implementation
- serves as a communication medium between
  - a client and specifier
  - an implementor and specifier
  - between implementors
- capture “what” rather than “how”
Programming vs Specification Languages

PL → SL
SL ↦ PL

◆ Specifications do not have to be executable
◆ are not restricted to expressing only computable functions
◆ All programs are formal objects (and thus can be formally manipulated - compiled and executed)
◆ Programmers can either work with informal requirements and formal programs or use additional formalism for requirements specification.

Assumptions

◆ Requirements Goal: Establish and Specify what the software must do (without describing how)
  – Establish precisely as possible what is to be built
  – Specify (document) results for communication
◆ Targeted to High-Assurance Systems
  – Large, complex embedded applications
    » Traffic control, avionics, medical
    » Expensive and difficult to build correctly
  – Real-Time constraints
    » Require temporal as well as logical correctness
    » Correct software delivers results within real time limits
◆ Software is safety/mission critical
  » small errors => BIG PROBLEMS!
  » High cost of failure => high cost of verification
Role of Correct Requirements

<table>
<thead>
<tr>
<th>DEVELOPMENT PHASE</th>
<th>HIGH-ASSURANCE GOAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUIREMENTS</td>
<td>1. Get the right requirements</td>
</tr>
<tr>
<td>REQUIREMENTS SPECIFICATION</td>
<td>2. Get the requirements right</td>
</tr>
</tbody>
</table>
| DESIGN             | a. Application-Independent Properties  
|                    |   • Consistency  
|                    |   • Completeness |
| CODING             | b. Application-Dependent Properties  
|                    |   • Safety properties  
|                    |   • Mission-critical properties |
| TEST PLAN          | 3. Verify the system against the requirements |
| SYSTEM INTEGRATION AND TESTING | a. Test coverage |
|                    | b. Traceability  
|                    | c. Statistically based testing |

Example: Temperature Control System

This system is to control a thermostat to maintain constant temperature. The system can be turned on or off. The user is also allowed to set the desired temperature. The system reads the current temperature and turns on the heater or the air conditioner to ensure that the desired temperature is maintained.

Insert picture here (thermostat.ps)
Specifying this system

◆ Property-oriented approach: specify which properties the system is supposed to have
  – If the system is ON and the temperature becomes too high, the system will turn the air conditioner on
  – If the system is ON and the temperature becomes too low, the system will turn the heater on
  – The system never runs the heater and the air conditioner at the same time
  – The system should always react to the OFF switch
  – The system should not continue running the heater or the air conditioner if the temperature is within the desired limits

Properties of the Thermostat

◆ Note that these are temporal properties. Can specify them using CTL.
◆ Example:
  – The system never runs the heater and the air conditioner at the same time

  – If the system is ON, it is either running an air conditioner, or a heater, or is being idle

  – If the temperature is too high, eventually the air conditioner will be turned on
Assessment of the Property-Oriented Method

- Advantages:
  - Describes WHAT not HOW
  - No implementation bias

- Disadvantages:
  -
  -

Model-Oriented Specifications

- Specification that defines a system’s behavior directly by constructing a model of the system in terms of some simple and well-understood mathematical structures.

- Examples:
  -
  -
  -
  -
Model of the Temperature-Control System

- Identify inputs:
- Identify outputs:
- Identify required relationship between input and output
- Does the system need to keep state? If so, what are the states? Draw the state machine

Assessment of the Model-Oriented Approach

- Advantages:
- Disadvantages:
The Formal Methods Dilemma

◆ Standard approaches (e.g., prose specs) lack sufficient rigor to meet high-assurance goals
◆ Formal requirements methods have desired technical virtues but industry slow to adopt
  – Capability for unambiguous specification, precision, testability, and analyzability
  – Industry concern for practicality
    » Concern for difficulty to write/read, required expertise, ability to scale
    » Concern for real-world issues of fuzzy or changing requirements
    » Concern for fit with industrial development context
    » Adds up to perceived cost/schedule risk
◆ Dilemma: Current methods don’t meet high-assurance needs, formal methods do but perceived as impractical

SCR Principles (1)

◆ Separation of Concerns (divide and conquer)
  – Provide distinct mechanisms for distinct concerns
  – Separate requirements concerns from design/implementation
    » Avoid premature design/implementation decisions
    » Know when to stop
  – Separate “customer view” from “software view”
    » Improved communication
    » Simpler specification
SCR Principles (2)

- Practical formalism (usability, communication)
  - Formal model for capturing requirements
    » Precise, unambiguous, testable specifications
    » Meaningful verification high-assurance properties
  - “User friendly” formats and notations
    » Use commonly understood math (sets, functions)
    » Use tabular representation for ease of writing, reading
- Effective automation
  - Essential to making formal methods practical
    » Increases speed and accuracy
    » Decreases cost

Evolution of SCR

  - Formal as possible
  - Notations for ease of use
  - Industrial strength: demonstrated scalability, effectiveness
- Parnas/Madey* (McMaster - 1991)
  - Standardized A-7E model (problem domain)
  - Mathematical basis for NRL formal model
- CoRE* (SPC 1987 - 1992)
  - Standardized industrial-strength process
  - Added modularization (OO) techniques, graphics
  - Applied to real high-assurance development
- SCR* (NRL - Ongoing)
  - Complete formal model
  - Basis for automated formal verification (consistency checking, property verification)
Some Industrial Applications of SCR

- A-7E
- Lockheed C-130J, 1993-1994 (using CoRE)
- Ontario Hydro
  - Integration with formal design/code verification
- Rockwell-Collins (CoRE using SCR tool)
- NASA Shuttle, 1997-1998 (SCR, Easterbrook + K°)

Case I: The Lockheed C-130J

- C-130J: Lockheed’s next-generation airlifter
  - New avionics, propulsion, and flight deck
  - Coordination, data transfer, and control by software
    - 100K SLOC Ada
    - All safety critical
- Advanced software development paradigm
  - Concurrent hardware/software development (IPT’s)
  - Asynchronous software development
- Key Goals of the C-130J Development
  - Safety: consistent with FAA DO-178A Level 1 certification
  - Flexibility with stability: Ability to meet market demands without re-engineering software
  - Low risk: Maintain safety while meeting cost and schedule
  - Chose CoRE/SCR as meeting these goals
Case II: Ontario Hydro

- Build/run nuclear power plants
  - 25 years experience with digital systems
  - Use of digital systems in safety systems led to regulatory problems
  - Led to development standardization based on SCR
- Development framework
  - SCR for requirements specification
  - Captures results of hazard analysis
  - Basis for formal verification of design
  - Basis for verification including reviews, inspections, and testing
- Currently being applied on 6th system

Case III:
Rockwell Collins Avionics*

- Experimental application of CoRE/SCR to generic autopilot (sanitized but real example)
  - Full specification of mode logic written in CoRE
  - Analysis of behavioral model using SCR tools
- In spite of extensive review, tool finds errors
  - Found 24 errors, “many of them significant”
  - One third each, constructing the model, running the completeness and consistency checks, and executing the model
- Example: Disjointness error leading to two possible flight modes (selection of Go Around leading to Pitch or Go Around mode, ambiguously)
- Example: Missing case for controlled variable (Lateral Armed Annunciation field not defined if Nav source not selected prior to Lateral Nav Mode selection)

The Ideal System

- Ideal system is a state machine whose outputs are a function of the history of the environment
  - All digital systems approximate the ideal
  - SCR approach: specify the ideal, then specify the tolerances

The Four-Variable Model
SCR and 4-Variable Model

- Model relevant environmental quantities as mathematical variables (called *environmental variables*).
  
  1. *Monitored variables* = quantities measured (e.g., temperature, pressure, altitude)
  
  2. *Controlled Variables* = quantities affected (e.g., displayed value, valve state, air conditioner on)

  Both are either continuous or discrete.

- Hardware allocation determines available resources
  
  3. *Inputs* = resources form which the values of monitored variables must be determined
  
  4. *Outputs* = resources available to affect controlled variables

  Both are always discrete.

The 4-Variable Viewpoint: Variables
4 Variables of the Thermostat System

- Monitored quantities: On/Off switch, desired temperature, current temperature (continuous)
- Controlled quantities: AC, Heater
- Input data items - values produced by input devices:
  - On/Off switch (Switch)
  - desired temperature, represented as float (DesiredTemp)
  - current temperature, represented as float, sampled every 3 sec (CurrentTemp)
- Output data items: values written to the output devices that turn the AC and the heater on and off.
- NAT: in a given time interval, the room temperature can only change by a certain amount.
Four Relations Define the Software Requirements (SOFTREQ)

- **REQ**: Required relation between the monitored and the controlled variables
- **NAT**: Constraints imposed on the system behavior by physical laws and the system environment
- **IN**: Required relation between the monitored variables and the system’s input data items
- **OUT**: Required relation between the controlled variables and the system’s output data items

Representing SCR Relations

- All behavioral requirements are defined as a set of functions over the states of SCR variables
- Representing state (and state changes)
  - A **state** is a mapping of variables to values
  - **Condition** - a predicate over the state variables
    - Altitude > 500 ft. or Pressure = 500 psi or ButtonPressed = true
  - Event - a predicate over successive states (denotes a change in state)
    - @T(Altitude > 500 ft.)  @T(Running=false)
    - @T(Running=true) WHEN [Altitude > 500 ft.]
Representing Functions of State

- Representing functions of state
  - A modeclass is a set of system states, called modes, which partition the monitored environment’s state space. Modeclasses are finite state machines used to capture history.
  - The system is exactly in one mode of each modeclass.
    For the Thermostat:
    one modeclass, four modes: Off, Idle, RunningHeat, RunningAC.
- Term - a function over two or more variables or states

---

Thermostat

Term tempStatus =

\[
\begin{align*}
\text{Above if CurrentTemp-DesiredTemp} & > 3 \\
\text{Below if DesiredTemp-CurrentTemp} & > 3 \\
\text{Normal otherwise}
\end{align*}
\]

Mode Transition Table

<table>
<thead>
<tr>
<th>Old Mode</th>
<th>Event</th>
<th>New Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off</td>
<td>@T(Switch=On)</td>
<td>Idle</td>
</tr>
<tr>
<td>Idle, RunningAC, RunningHeat</td>
<td>@T(Switch=Off)</td>
<td>Off</td>
</tr>
<tr>
<td>Idle</td>
<td>@T(tempStatus=Above) WHEN Switch=On</td>
<td>RunningAC</td>
</tr>
<tr>
<td>Idle</td>
<td>@T(tempStatus=Below) WHEN Switch=On</td>
<td>RunningHeat</td>
</tr>
<tr>
<td>RunningAC, RunningHeat</td>
<td>@T(tempStatus=Normal) WHEN Switch=On</td>
<td>Idle</td>
</tr>
</tbody>
</table>
Representing SCR Relations

- Controlled variable function - a function giving the (ideal) value of a controlled variable
- Controlled variable functions define visible behavior
  - Specified as a set of functions written in terms of monitored variables, terms and modes defining:
    - *Ideal value*: specifies the ideal value of the controlled variable over all possible states
    - *Accuracy*: defines the allowed tolerance in value
    - *Timing*: defines timing constraints, e.g., minimum delay and deadline
  - Together these define the set of acceptable values over all possible states of the monitored space

SCR Methodology: Overview

1. Identify the system outputs (*controlled quantities*)
2. Determine the inputs that the system monitors (*monitored quantities*) to produce the outputs
3. Define auxiliary variables (*mode classes and terms*)
   - functions of the inputs
   - help make the model concise
4. Specify the *ideal* system behavior
5. Specify the *actual* system behavior (i.e., the required timing and precision)
Example: Control System for Safety Injection

- Based on a control system in a real nuclear power plant*
- System is required to turn on safety injection when water pressure falls below a threshold ‘Low’.
- Operator can set a Block button to inhibit safety injection and a Reset button to reset the system after blockage.

Example: Control System for Safety Injection

- Mode Class Pressure - capture three states of WaterPres of interest and the possible transitions among them
- Term Overridden - denotes condition of the injection system being overridden by the operator
- Controlled variable SafetyInjecton - defined in terms of terms, modes, monitored variables
Table Functions and Dependencies among the Variables

To define the required behavior, a function is associated with every mode, term, and output. Most of these functions are expressed in a tabular form and hence are called Table Functions.

**Condition Table for Safety-Injection**

<table>
<thead>
<tr>
<th>Mode</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>High,Permitted</td>
<td>True</td>
</tr>
<tr>
<td>TooLow</td>
<td>Overridden</td>
</tr>
<tr>
<td>SafetyInjection</td>
<td>Off</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT Overridden</td>
<td></td>
</tr>
<tr>
<td>On</td>
<td></td>
</tr>
</tbody>
</table>
### Event Table for Safety-Injection

<table>
<thead>
<tr>
<th>Mode</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>False</td>
</tr>
<tr>
<td>TooLow, Permitted</td>
<td>@T(Block=On) OR @T(Reset=On)</td>
</tr>
<tr>
<td>Overridden</td>
<td>True</td>
</tr>
</tbody>
</table>

### Mode Transition Table for Safety-Injection

<table>
<thead>
<tr>
<th>Old Mode</th>
<th>Event</th>
<th>New Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>TooLow</td>
<td>@T(WaterPres ≥ Low)</td>
<td>Permitted</td>
</tr>
<tr>
<td>Permitted</td>
<td>@T(WaterPres ≥ Permit)</td>
<td>High</td>
</tr>
<tr>
<td>Permitted</td>
<td>@T(WaterPres &lt; Low)</td>
<td>TooLow</td>
</tr>
<tr>
<td>High</td>
<td>@T(WaterPres &lt; Permit)</td>
<td>Permitted</td>
</tr>
</tbody>
</table>
SCR Requirements Formal Modal

- What is the relationship between the Parnas-Madey Four Variable model and the implicit state machine model that underlies SCR specifications?
- How does one formally represent a condition? an event? a conditioned event?
- What are the semantics of SCR tables?
- How are the variables in SCR requirements specs (monitored and controlled variables, mode classes, terms) related?

Entities, Type and System State

- \(MS\) is the union of \(N\) pairwise disjoint sets, called mode classes. Each member of a mode class is called a mode.
- \(TS\) is a union of data types, where each type is a nonempty set of values
- \(VS = MS \cup TS\) is the set of entity values
- \(RF\) is a set of entity names \(r\). \(RF\) is partitioned into four subsets: \(MR\), the set of mode class names, \(IR\), the set of input variable names, \(GR\), the set of term names, and \(OR\), the set of output variable names. For all \(r \in RF\), \(TY( r ) \subseteq VS\) is the type of the entity named \(r\).
- A system state \(s\) is a function that maps each entity name \(r\) in \(RF\) to a value: \(\forall r \in RF : s(r) = v\)
  where \(v \in TY(r)\).
**Example: System State**

In the example control system are the following sets:

Set of input variables: $IR = \{ \text{Block, Reset, WaterPres} \}$
Set of output variables: $OR = \{ \text{SafetyInjection} \}$
Set of terms: $GR = \{ \text{Overridden} \}$
Set of mode classes: $MS = \{ \text{Pressure} \}$

Type definitions associated with these sets are:

- $TY(\text{WaterPres}) = \{14, 15, ..., 2000\}$
- $TY(\text{SafetyInjection}) = \{\text{On, Off}\}$
- $TY(\text{Block}) = TY(\text{Reset}) = \{\text{On, Off}\}$
- $TY(\text{Overridden}) = \{\text{true, false}\}$
- $TY(\text{Pressure}) = \{\text{TooLow, Permitted, High}\}$

<table>
<thead>
<tr>
<th>entity name</th>
<th>WaterPres</th>
<th>Block</th>
<th>Reset</th>
<th>Pressure</th>
<th>Overridden</th>
<th>SafetyInjection</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity value</td>
<td>880</td>
<td>Off</td>
<td>Cn</td>
<td>TooLow</td>
<td>false</td>
<td>Off</td>
</tr>
</tbody>
</table>

**Conditions**

- **Conditions** are defined on the values of entities in RF.
- A **simple condition** is true, false, or a logical statement $r \, \mathcal{O} \, v$, where $r \in RF$ is an entity name, $\mathcal{O} \in \{\approx, \neq, >, <, \geq, \leq\}$ is a relational operator, and $v \in TY(r)$ is a constant value.
- A condition is a logical statement composed of simple conditions connected in the standard way by logical connectives AND, OR and NOT.
## Events

- A **primitive event** is denoted \(@ T(r=v)\), where \(r\) is an entity in \(RF\), and \(v\) \(\in\ TY(r)\). An **input event** is a primitive event \(@ T(r=v)\) with \(r \in IR\) is an input variable.
- A **basic event** is denoted \(@ T(c)\), where \(c\) is any simple condition.
- A **simple conditioned event** is denoted \(@ T(c) \text{ WHEN } d\), where \(@ T(c)\) is a basic event and \(d\) is a simple condition or a conjunction of simple conditions.
- A **conditioned event** \(e\) is composed of simple conditioned events connected by AND and OR.

The logical statement represented by a simple conditioned event is:

\[
@ T(c) \text{ WHEN } d = \neg c \land c' \land d
\]

\((c\) denotes the condition in the old state, \(c'\) - in the new state)  

## Examples: Conditions and Events

<table>
<thead>
<tr>
<th>Concept</th>
<th>Notation</th>
<th>Semantics</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple condition</td>
<td>Reset=Off, WaterPres&lt;900</td>
<td>(\text{Res-Off; WaterPres&lt;900})</td>
<td>(\text{true in } s \text{ and } s')</td>
</tr>
<tr>
<td>condition</td>
<td>Reset=Off AND WaterPres&lt;900, Reset=Off OR WaterPres&lt;900</td>
<td>(\text{Res-Off AND WaterPres&lt;900} \text{ OR Res-Off OR WaterPres&lt;900})</td>
<td>(\text{true in } s \text{, false in } s')</td>
</tr>
<tr>
<td>primitive event</td>
<td>(#T(\text{WaterPres=900}))</td>
<td>(\text{WaterPres=900} \text{ AND WaterPres'=900})</td>
<td>(\text{true in } s, s')</td>
</tr>
<tr>
<td>conditioned event</td>
<td>(#T(\text{Reset=Off}))</td>
<td>(\text{Reset=Off} ) &amp; (\text{Reset'=Off})</td>
<td>(\text{false in } s, s')</td>
</tr>
</tbody>
</table>

\(\text{states} \\{\text{Reset=Off, WaterPres=25} \cdots \text{Reset=Off, WaterPres=900} \}\)
System

A system $\Sigma$ is a 4-tuple, $\Sigma = (E^m, S, s_0, T)$, where

- $E^m$ is a set of input events
- $S$ is the set of possible system states
- $s_0$ is the initial state
- $T$, the system transform, is a partial function from $E^m \times S$ into $S$

Condition Tables

Each condition table describes an output variable or term $r_i$ as a relation $\rho_i$ defined on modes, conditions, and values. The relation $\rho_i$ must satisfy the following four properties:

1. The $m_j$ and the $v_k$ are unique.
2. $\bigcup_{j=1}^{n} m_j = M_{	ext{all}}$ (All modes are included).
3. For all $j$: $\vee_{k=1}^{n} c_{j,k} = \text{true}$ (Coverage: The disjunction of the conditions in each row of the table is true).
4. For all $j, k, l$, $k \neq l$: $c_{j,k} \wedge c_{j,l} = \text{false}$ (Disjointness: The pairwise conjunction of conditions in a row is always false).

To make explicit entity $r_i$'s dependencies on other entities, we consider an alternate form $F_i$ of $\rho_i$. The four properties above ensure that $F_i$ is a total function.
Condition Table: Example

<table>
<thead>
<tr>
<th>Modes</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>High, Permitted</td>
<td>True</td>
</tr>
<tr>
<td>TooLow</td>
<td>Overridden</td>
</tr>
<tr>
<td>SafetyInjection</td>
<td>Off</td>
</tr>
</tbody>
</table>

Based on the new state dependencies set
\{Pressure, Overridden\}

and the above condition table, the function for Safety Injection, denoted $F_6$, is defined by

$F_6 = \text{Safety\_Injection} = \begin{cases} 
\text{Off} & \text{if } \text{Pressure} = \text{High} \lor \text{Pressure} = \text{Permitted} \lor \\
\text{On} & \text{if } \text{Pressure} = \text{TooLow} \land \text{Overridden} = \text{true} 
\end{cases}$

Event Tables

Each event table describes an output variable or term $r_i$ as a relation $\rho_i$ between modes, conditioned events, and values. The relation $\rho_i$ must satisfy the following two properties:

1. The $m_j$ and the $v_k$ are unique.
2. For all $j, k, l, k \neq l$: $e_{j,k} \land e_{j,l} = false$ (Disjointness: The pairwise conjunction of events in a row of the table is always false).

To make explicit entity $r_i$’s dependencies on other entities, we consider an alternate form $F'_i$ of $\rho_i$. The One Input Assumption (only one input event occurs at a time) and the two properties ensure that $F'_i$ is a function. The “no change” part of $F'_i$’s definition guarantees totality.
Event Table: Example

<table>
<thead>
<tr>
<th>Mode</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>True</td>
</tr>
<tr>
<td>Low, Permitted</td>
<td>GT(Block=On)</td>
</tr>
<tr>
<td></td>
<td>GT(Reset=Dif) OR GT(Reset=On)</td>
</tr>
<tr>
<td>Overridden</td>
<td>False</td>
</tr>
</tbody>
</table>

Based on the new state and old state dependencies sets,

{Block, Reset, Pressure, Overridden} and {Block, Reset, Pressure},

and the above event table, the function for Overridden, denoted \( \bar{E}_5 \), is defined by

\[
\bar{E}_5(\text{Block, Reset, Pressure, Overridden, Block', Reset', Pressure'}) =
\begin{cases}
\text{true} & \text{if } [(\text{Pressure} = \text{TooLow} \land \text{Block'} = \text{On}) \lor (\text{Block'} = \text{Off} \land \text{Reset'} = \text{Off})] \\
\text{false} & \text{if } [(\text{Pressure} = \text{TooLow} \land \text{Reset'} = \text{On}) \lor (\text{Block'} = \text{Off} \land \text{Reset'} = \text{Off})] \\
\text{false otherwise}
\end{cases}
\]

Mode Transition Tables

<table>
<thead>
<tr>
<th>Old Mode</th>
<th>Event</th>
<th>New Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m_1 )</td>
<td>( e_{11} )</td>
<td>( m_{1,1} )</td>
</tr>
<tr>
<td></td>
<td>( e_{12} )</td>
<td>( m_{1,2} )</td>
</tr>
<tr>
<td></td>
<td>( \ldots )</td>
<td>( m_{1n} )</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( m_k )</td>
<td>( e_{k1} )</td>
<td>( m_{k,1} )</td>
</tr>
<tr>
<td></td>
<td>( e_{k2} )</td>
<td>( m_{k,2} )</td>
</tr>
<tr>
<td></td>
<td>( \ldots )</td>
<td>( m_{kn} )</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
</tr>
</tbody>
</table>

Each mode transition table describes a mode \( r_i \) as a relation \( \rho_i \) defined on modes, events, and modes. The relation \( \rho_i \) must satisfy the following four properties:

1. The \( m_j \) are unique.
2. For all \( k \neq i, m_{ij} \neq m_{ij} \), and for all \( k \) and for all \( k, m_{ij} \neq m_{ij} \).
3. For all \( j, k, h, i \neq j \land i \neq j \) \( \rho_{ijk} \text{ false} \) (Disjointness: The pairwise conjunction of conditioned events in a row of the table is always false).
4. Let \( m_0 \) be the initial mode. Then, \( M_{ij} \subseteq \{ m \mid Q^*(m_0, m) \} \), where \( Q(m_0, m) \) is the reflexive and transitive closure of \( Q \) (Reachability). Each mode must be reachable from the initial mode.

Each mode transition table can be expressed in the format shown for event tables.
Mode Transition Table: Example

<table>
<thead>
<tr>
<th>Old Mode</th>
<th>Event</th>
<th>New Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>TooLow</td>
<td>$\text{GT(WaterPress \geq \text{Low})}$</td>
<td>Permitted</td>
</tr>
<tr>
<td>Permitted</td>
<td>$\text{GT(WaterPress \geq \text{Permit})}$</td>
<td>High</td>
</tr>
<tr>
<td>Permitted</td>
<td>$\text{GT(WaterPress &lt; \text{Low})}$</td>
<td>TooLow</td>
</tr>
<tr>
<td>High</td>
<td>$\text{GT(WaterPress &lt; \text{Permit})}$</td>
<td>Permitted</td>
</tr>
</tbody>
</table>

Based on the new state and old state dependency sets, $(\text{WaterPres}, \text{Pressure})$ and $(\text{WaterPres})$, and the above table, the function for Pressure, denoted $P_a$, is defined by

$$P_a = P_a(\text{Pressure}, \text{WaterPres}, \text{WaterPres}') =$$

- TooLow if $\text{Pressure = Permitted \land WaterPres' < \text{Low} \land WaterPres < \text{Low}}$
- High if $\text{Pressure = Permitted \land WaterPres' \geq \text{Permit} \land WaterPres \geq \text{Permit}}$
- Permitted if $(\text{Pressure = TooLow \land WaterPres' < \text{Low} \land WaterPres < \text{Low}}) \lor (\text{Pressure = High \land WaterPres' \geq \text{Permit} \land WaterPres \geq \text{Permit}})$
- Pressure otherwise.

Partial Ordering of the Variables

To reflect the dependencies, we order the entities in $R^r$ as a sequence $R$,

$$R = <r_1, r_2, \ldots, r_i, r_{i+1}, \ldots, r_k, r_{K+1}, \ldots, r_{P^r}>,$$

where

$$R_i = <r_1, r_2, \ldots, r_i>, r_i \in IR_i$$

is the subsequence of monitored state variables,

$$R_Y = <r_{i+1}, r_{i+2}, \ldots, r_k>, r_i \in GR \cup MR_i$$

is the subsequence containing terms and modes, and

$$R_O = <r_{K+1}, r_{K+2}, \ldots, r_P>, r_i \in OR_i$$

is the subsequence of controlled state variables.

The entities $r_i \in R$ are partially ordered such that for all $i, i', i > i', 1 \leq i' \leq K$, the value of entity $r_i$ can only depend on the value of entity $r_{i'}$ if $i' < i$. This definition reflects the fact that each monitored variable can only be changed by external events and cannot depend on the other entities in $R$. In contrast, each term can depend on the monitored variables, the terms, or other terms; each mode can depend on the monitored variables, the terms, or other modes; and each controlled variable can depend on the monitored variables, the terms, and the modes.
Four Different Notations with the Same Underlying Semantics

1. MATHEMATICAL NOTATION

2. GRAPHICAL NOTATION

3a. TABULAR NOTATION (van schouwen)

<table>
<thead>
<tr>
<th>Mode</th>
<th>Triggering Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALOK</td>
<td>X</td>
</tr>
<tr>
<td>BadLevDev</td>
<td>X</td>
</tr>
<tr>
<td>HardFail</td>
<td>X</td>
</tr>
<tr>
<td>Failure</td>
<td>ALOK, BadLevDev</td>
</tr>
</tbody>
</table>

*贴近于定义的在3A。

3b. TABULAR NOTATION (faultk)

<table>
<thead>
<tr>
<th>Current Mode</th>
<th>New Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALOK</td>
<td>BadLevDev</td>
</tr>
<tr>
<td>BadLevDev</td>
<td>HardFail</td>
</tr>
<tr>
<td>HardFail</td>
<td>HardFail</td>
</tr>
</tbody>
</table>