OMSE 531: Software Requirements Engineering

Spring 2012 Assignment 2

Due: June 9th, 2012
Instructor: Joe Maybee

jmaybee@pdx.edu

Overview

This assignment is a continuation of the work completed in assignment 1.

In this assignment you will build a framework for formal requirements that will analyze finite state automata requirements specifications and generate natural language specifications from a formal representation of finite state automata.

The assignment is due no later than Saturday, June 9th, 2012 at 10 PM. Assignments must be delivered via the D2L drop box.

Assignment: Requirements Checking

Consider a state machine defined by its requirements named RSM.[1] The RSM is denoted by the tuple: $(Q, q_0, E, R, \delta, \gamma)$ where:

$\boldsymbol{Q}$: The finite set of states

$\boldsymbol{q_0} \in \boldsymbol{Q}$: The initial state of the machine.

$\boldsymbol{E}$ **:** The set of event predicates (sometimes called *trigger predicates*).

$\boldsymbol{R}$ **:** The set of response predicates.

$\boldsymbol{\delta}$: The state transition ("next state") function mapping $Q \times E$ to Q. That is, $\delta(q, e)$ where $q \in Q$ and $e \in E$, defines the next state when the software is in state q and takes the transition having e as the event predicate.

$\boldsymbol{\gamma}$: The event to response function mapping $Q \times E$ to R. That is, $\gamma(q, e)$ where $q \in Q$ and $e \in E$, defines the response when the software is in state q and takes the transition having e as the event predicate.

Three additional fact sets will be provided in addition to the previous:

`initial(X)`: specifies state `X` as the initial state.

`safe(X)` : specifies state `X` as a safe state.

`hazard(X)` : specifies state `X` as a hazardous state.

`recurrent(X)`: specifies state `X` as a state that is part of desired recurrent behavior.

In the following rules:

$\hat{\delta}(q, s)$: The sequence of predicates (in our case, events) leading to state q.

$\phi(s_i)$: The conjunction of the predicates (events) in a sequence s_i.

[1] Based on: M. Jaffe, N. Leveson, M. Heimdahl and B. Melhart, *"Software Requirements Analysis for Real-Time Process Control Systems"*, IEEE Transactions on Software Engineering, Vol. 17, No. 3, pp. 241-258, March 1991.

Implement a clause in Prolog list_violations/0 that prints a list of requirements that violate the rules listed in the following section, along with the rule(s) or definitions violated. For example:

```
violation of definition:  requirement id_21 next state
'turn_off_pumpe' is not a member of the set of states.

violation of definition:  requirement id_22 response
'halt_andx_catch_fire' is not a valid defined response.

violation of rule 1.2:  state 'trace_off' not reachable
from the initial state.
```

General Rules:

- There is one and only one initial state definition which must be a valid defined state.
- All defined states must appear in at least one requirement.
- All defined events must appear in at least one requirement.
- All defined responses must appear in at least one requirement.
- All defined requirement identifiers must be unique.
- All requirement identifiers must be unique.
- For each requirement, each state, event, response and next state must be defined.

Rule 1.1: Behavior must be deterministic. There must be only one exit for an event clause for any given state. That is:

$$e_i, e_j \in \mathrm{E} :$$

$$\forall\, i\ \forall\, j : \Big((i \neq j) \Rightarrow \neg\big(e_i \wedge e_j\big)\Big)$$

Rule 1.2: All states must be reachable from the initial state:

$$\forall\, q\ \exists\, s\ :\ \left(\hat{\delta}(q_0, s) = q\right)\ \wedge\ \left(\phi(s_i)\right)$$

Rule 1.3: Every path from a hazardous state must lead to a safe state:

$q_h \in Q_h : Q_h$ Is the set of hazardous states, and
$s_i \in Q_s : Q_s$ Is the set of safe states.

$$\forall\, q_h, s\ :\ \left(\left(\hat{\delta}(q_h, s) = q\right)\ \wedge\ \left(\phi(s_i)\right)\ \Rightarrow\ (q \in Q_s)\right)$$

Rule 1.4: Recurrent behavior must be part of at least one cycle. State q is part of a cycle iff:

$$\exists\, s\ :\ \left(\hat{\delta}(q, s) = q\right)\ \wedge\ \left(\phi(s_i)\right)\ \wedge\ (s \neq \lambda)$$

The representation of the requirements will consist of two separate files as before.

Grading

The artifacts will be judged on the following merits:

Correct: Does the code perform correctly?

Clarity: Are the code and comments easy to understand?

Conciseness / Succinctness: Is the code straight to the point?

Coherence: Does the code reflect the process as it was defined?

Completeness: Are all elements of the code present and easy to identify?