Program Monitoring

Lecture 4 : State Machines and Regular Expressions

Wednesday May 13, 2009

This fourth lecture is an introduction to monitoring with state machines and regular expressions. We start with the propositional case, without data values. That is, for example, we can express properties about files being opened and closed, but we cannot yet speak about what files. The subsequent lecture 5 will introduce such parameterized state machines and regular expressions and show how they are supported by the JavaMOP system. Although this lecture does not yet fully open up for JavaMOP, you may start playing with it, specifically the FSM (Finite State Machines) and ERE (Extended Regular Expressions) parts.

JavaMOP

You can fully operate JavaMOP though the JavaMOP website, which is:

`http://fsl.cs.uiuc.edu/index.php/Special:JavaMOP2.0`

There is no need to download anything. Try to experiment with the FSM and ERE plugins (click on these names in the top row). In each of these you can type specifications (see the plugin *input syntax* - click on field in upper right corner when in the plugin) or look at examples (click on example names to the left when in the plugin). Note, some of the text explaining these examples is wrong. In these plugins events are just mentioned by name. There is no linking to any Java program being monitored yet.

An example regular expression with event definitions is:

```
event open
event close

ere: (open close)*
```

Note: a monitor of this specification will only consider declared events, in this case `open` and `close`. All other events in the system will be ignored.

When you press the "Run" button, the regular expression is translated to an automaton, which is visualized. This is specifically useful in order to understand how this translation works. The generated automaton is also shown in the input format for the FSM

module, in this case it becomes a state machine with two states `s0` and `s1`, where state `s0` is a final state, expressed with "`!`". State `s0` has a transition to state `s1` labelled `open`, etc.

```
event open
event close

fsm:

!s0[
  open -> s1
]

s1[
  close -> s0
]
```

In the FSM module one can enter such specifications, and the "Run" button will just visualize the automaton (no translation is performed).

Reading

The reading suggested here (available from the course website) covers lectures 4 and 5.

"MOP: An Efficient and Generic Runtime Verification Framework".
Feng Chen and Grigore Rosu.

This slightly theoretic paper introduces the JavaMOP system. Note: it's not a manual. The syntax presented in the paper is out of date. The correct syntax for the MOP logics are given on the web-site.

Optional Reading

"Adding Trace Matching with Free Variables to AspectJ".
Chris Allan, Pavel Avgustinov, Aske Simon Christensen, Laurie Hendren, Sascha Kuzins, Ondrej Lhotak, Oege de Moore, Damien Sereni, Ganesh Sittampalam and Julian Tibble.

This paper introduces an alternative system that supports regular expressions. It *extends* AspectJ, in contrast to JavaMOP which *includes* AspectJ language constructs (and generates AspectJ code).

Assignments

None.