Graphical models for QuickCheck

Thomas Arts, Kirill Bogdanov, Alex Gerdes, John Hughes

This project has received funding from the EU FP7 Collaborative project PROWESS, grant number 317820, http://www.prowessproject.eu
Testing with QuickCheck

- QuickCheck permits one to write generators for test data and pre/postconditions.

- The expectation is that user provides a model, based on which test data is randomly generated.

- Illustration of testing a `write` operation:

  ```
  write_args(_) -> [key(), int()].
  write(Key, Value) -> lock:write(Key, Value).
  write_post(_, [Key, Value], Res) -> eq(Res, ok).
  ```
Global state is a record-type of type `state` with element `started`, passed as an argument to all operations.

- `write_pre(S) -> S#state.started`
- `write_args(S) -> [ key(), int() ]`
Testing **write** using global state

Assuming *started* is a boolean component of the global state reflecting if the system was started,

\[
\text{write_args}(S) \rightarrow \text{[key(), int()]}.
\]

\[
\text{write}(\text{Key}, \text{Value}) \rightarrow \text{lock:write(}\text{Key}, \text{Value})\).
\]

\[
\text{write_pre}(S) \rightarrow S\#\text{state}.\text{started}
\]

\[
\text{write_post}(S, [\text{Key,Value}], \text{Res}) \rightarrow \text{eq(}\text{Res,ok})\).
\]

\[
\text{write_next}(S, \text{Res, [Key, Value]}) \rightarrow
S\#\text{state}\{
\text{kvs} = \{
\text{Key,Value} | 
\text{proplists:delete(}\text{Key},S\#\text{state.kvs})\}\}
\].
Locker example

- Can be started/stopped
- Can be locked/unlocked
- Does not include read/write
lock_pre(S) -> S#state.started andalso not S#state.locked.
lock_args(S) -> [].
lock_next(S,Res,[]) -> S#state{locked=true}.

unlock_pre(S) -> S#state.started andalso S#state.locked.
unlock_args(S) -> [].
unlock_next(S,Res,[]) -> S#state{locked=false}.

Very easy to make a mistake in one of the above expressions
Now if we are doing something more complex, a lot of effort will go into ‘state maintenance’.
What we did

• Developed a tool to edit graphical models.

• Names of operations are extracted from Erlang code.

• For the above example, the resulting model is half the size of the traditional model …

    … and much easier to maintain.

• Test failures and frequencies are automatically extracted from results of test execution.
Addition of a *read* transition around *unlocked*. 
Frequencies

Running tests produces a distribution of transitions
Weights can be updated

Changing weights makes operations of interest run more frequently.
If you would like to try it

- You have access to both QuickCheck tool and the graphical editor online at [http://quviq.de/euc2015](http://quviq.de/euc2015)

- The .zip file contains both eqc_graphedit (the graphical editing tool) and time-limited version of QuickCheck that you need to install first.

- `lock.erl` is the module we are testing

- `lock_eqc.erl` is the QuickCheck model for testing `lock.erl`

- I’ll do the demo how to use the editor.
How to model the graphical editor using itself

we need to start with a state machine, hence generate one at random

the first thing to do is click ‘edit’
Selecting states

nothing is selected

something is selected

can click on any state to select it
Example: adding the ‘lock’ state
Adding states

- The tool waits for us to pick where to add and we can enter details of the state.
- Decide no to add a state.
- Cancel adding a state.
Editing states

- State has to be selected.
- Edit has to be clicked.
- If ‘initial?’ is ticked it cannot be cleared.
Editing states

contains generators for both successful edits and invalid ones

similar dialog to add
Commands on the right-hand side

- Used to add transitions: choose a command, then drag a transition.
- Can be clicked at any time.
- Consequently, the corresponding transitions have to be added to each state.
Conclusions

• Existing QuickCheck models are hard to develop for complex state-transition diagrams.

• Developed interface to edit such diagrams.

• Part of the most recent version of QuickCheck.

• Tested using itself.