Compiling Statemate Statecharts into CSP and verifying them using FDR – abstract

Bill Roscoe
Oxford University Computing Laboratory
Bill.Roscoe@comlab.ox.ac.uk

January 2, 2003

I visited Motorola Australia Software Centre[1] (near Adelaide) in April and May 2002 and was asked to look at how methods developed for tackling the state exploration problem in the CSP/FDR framework would work in the setting of their Statechart tool Veristate. To attack this problem I needed to understand the semantics of Statecharts in some appropriately rigorous setting. My eventual solution to understanding the semantics was to write a translation from Statecharts into CSP. I developed the translation as a program written in CSP (making heavy use of the powerful functional programming language embedded within CSP) which takes a description of a Statechart as input data. Depending on how you look at it, the CSP program is either a semantic function from the input to CSP, or an interpreter in CSP for the input Statechart.

This automated translation makes it easy to check for faithfulness, since one can observe the behaviour of the resulting CSP. It also opens up many opportunities for exploitation such as the following.

- We can verify properties of Statecharts using FDR.
- CSP methods such as compression, abstraction, data independence and symmetry are immediately applicable to improve the performance and/or scope of FDR checks.

[1]I am very grateful for all the help the engineers there gave me in understanding Statecharts, and for supplying me with examples.

- There is the potential, subject to future analysis, of checking properties such as the above in the translated CSP and then exploiting them on other Statechart verification tools such as Veristate.

The present prototype compiler (entirely written by the author in about 2 man-months) supports many of the features of Statecharts, including the following:

1. two level timing model (big steps representing passage of time and interactions with the environment, small "instantaneous" steps representing internal progress),
2. assignments to variables which take effect only on the next step (this is done via suitable processes running in parallel with the system),
3. timers (ones attached to the entry of states and independently named ones that are initiated and cancelled as actions),
4. the `ch(x)` construction (true if the input `x` has just had its value altered externally),
5. AND and OR states, and an implementation of the priority rules governing the firing of transitions,
6. actions promoted from sub-states.

There appears to be variation in the precise interpretation of some language features between different users, as well as in the range of features actually used. In this initial work I have concentrated on the range from the set of examples supplied to me by Motorola, but this will be easily extensible. Also, the nature of the CSP will in many cases make it easy to produce output which respects different semantics of some language features.

In the present version all nondeterminism present in a Statechart is considered an error and is detected by a standard FDR check of the CSP output.

The current input to the compiler is primitive: the user writes a CSP script which contains descriptions of all the sequential charts, the hierarchical constructions which put them together, plus information on variables, constants, timers etc. These are all just members of appropriately designed types. The script also contains all the specifications to check (whether standard ones, ones in the form of CSP refinement tests, or Statecharts which raise error flags, to be run in parallel). It includes the compiler, which makes the various checks runnable on FDR (and the chart and its components runnable on ProBE).

As with all FDR verifications, the execution of a check comes in two parts. The compilation phase has the effect of producing an efficient representation of the CSP (and hence, in the present case, the Statechart) for running the check. The running phase then actually performs the check. Experience to date has been that these two phases typically take similar times, though this will inevitably vary depending on example and the sizes of various parameters. The compiler allows the user to trade one of these times against the other by controlling which variables are cached by sequential charts (increased compilation time but decreases running time).

At present some optimisation tricks are used, such as extensive use of the CSP function `chase` (a form of partial order reduction), but there is undoubtedly scope for others. The compiler has been able to handle every example the author has thrown at it so far (namely, trivial-sized test examples, commercial-in-confidence ones from Motorola and the burglar alarm example which is available with this abstract, see below) within a few minutes. No comparisons with other tools have yet been performed.

When a specification fails FDR can be used to analyse the counter-example in the usual way (inspecting traces, divergences etc of the overall system and subprocesses). However, just as the input format is not really suitable for a typical engineer to use, neither is this output format really appropriate for him or her to understand easily.

> *It follows that one of the most important pieces of work to make my work accessible will be to improve the interface: there should be no difficult in writing automatically generating the CSP form of input from any reasonable Statechart tool suite, nor any in driving an animator from the debug traces (which can be made very detailed).*

One of the reasons for this work was to be able to apply CSP techniques to prove more general results or to prove results more efficiently. I have not had time to try this extensively yet but have introduced a feature (illustrated in the burglar alarm example) to allow timers to become nondeterministic (in a sense which does not cause a nondeterminism error to be flagged by the check which detects Statechart problems). This can often be used to prove that a result is true independently of a specific timer limit (or at least for any limit over some chosen threshold). This not only proves a much more general result than one that apparently depends on a specific set of constants, but will, if the original limits were large, typically use many fewer states.

There is enormous scope for developing further methods.

The compiler, and the burglar alarm example (both as graphical charts and as CSP input) can be found in the Concurrency Group website at Oxford University Computing Laboratory:

`web.comlab.ox.ac.uk/oucl/research/areas/concurrency`

I am keen to develop links with commercial and other organisations who have an interest in this style of verifying systems described as Statecharts. Contact me using the email address at the start of this abstract.