Experience using Spin and Promela in the Design of a Storm Surge Barrier Control System (abstract)

Pim Kars
Formal Methods Group
Dept. of Tele-Informatics and Open Systems
Faculty of Computer Science, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands
kars@cs.utwente.nl

October 12, 1995

Abstract

The Spin tool set (Spin and Xspin) was used to validate parts of the design of a storm surge barrier control system, in particular the communication interfaces with the outside world. Promela combined with Z is used to specify crucial aspects of the design. In this talk we outline our experience with the use of Spin and Promela, and discuss some ideas for extensions/improvements.

1 Introduction

As the final phase of a 40-year project to protect the Netherlands against floods due to storm surges, a movable barrier is currently being built near the Hook of Holland. A fully automated system, called BOS, will operate the barrier. BOS takes the decision to open or close the barrier based on measurements collected through various sensors and networks, and it controls the movements of the barrier. BOS has to satisfy high reliability demands. Not closing in time could result in the flooding of cities like Rotterdam. If the barrier is not opened in time after a closure, it could be severely damaged. Moreover, when the barrier is closed, the important harbour of Rotterdam is cut off from the North Sea, leading to great economic losses.

Computer Management Group (CMG) was awarded the contract from the Ministery of Public Works to build BOS. Because of the high reliability requirements BOS has to meet, CMG established a close collaboration with the Formal Methods group at the University of Twente to transfer knowledge and provide consultancy on the application of formal methods to system design.

As part of an effort to integrate formal methods with CMG's design trajectory the decision was made to

- use Spin to validate the communication between subsystems and with the outside world, and
- use Promela combined with Z to formally specify crucial parts of the design.

2 Use of Spin and Promela

2.1 Validation

Spin was used to validate most of the communication interfaces with the outside world and parts of the internal process control layer. A typical validation model has three (or more) processes: one process to model the interface, one process to model the environment, and one or more "user processes". The environment process typically consists of a non-deterministic choice between normal actions and error actions (losing a message, spontaneous responses due to noise, etc.). The user processes encode the

requirements on the expected service provided by the combination of the interface and the environment.

We used an incremental approach for validation, which seems fairly standard.

1. Start with a design and nice environment.
2. Iterate validation cycle
   - IF violations found: fix the design.
   - ELSE add more error behaviour to the environment.

A validation cycle starts with checking simple properties, then on to the more expensive properties. Each new design is first simulated to check that it correctly models the intentions of the designer.

Timeouts on asynchronous channels are modelled "locally" using `else`:

```
if
:: channel?data -> ...
:: empty(channel) -> ... timeout behaviour ...
fi
```

(`else` cannot be used instead of `empty(channel)` since this appears to give problems with partial order reduction.) Modelling timeouts in this way gives smaller models than using `skip`. Premature timeouts are still covered in our models due to the interleaving of sender and receiver processes.

2.2 Specification

Besides the communication interfaces, also other parts of the design are specified using a combination of Promela and Z. Z schemas are used to model data (in particular the state of a subsystem), and also to specify operations on the state. The Z operations are embedded in a Promela program to add control. We adopt the convention that a Z operation is executable when its precondition holds. This embedding is ad hoc in that no formal semantics is given for the combination. This can only be done when Promela is provided with a formal semantics.

The advantage of this approach is that a powerful, expressive language is used to specify data and operations. A potential drawback is that Z is not constructive and no tools exist to validate the combination. However, the specification is primarily intended to give an unambiguous specification of what has to be built. Also the Z *style* that is used is largely constructive, so that e.g. test oracles can still be derived from the Z specifications.

3 Experience

On the whole our experience was definitely positive.

- Spin/Promela is very easy to learn and use, even for an engineer with no previous exposure to formal methods. This was important since most of the actual work was done by engineers of CMG. Most of the language elements are simple/intuitive, there is a fine tutorial and many of the validation requirements can be expressed in a simple way without having to learn temporal logic.
- Spin was useful to expose errors in the proposed communication interfaces and in getting confidence in the final design.
- Message Sequence Charts were a valuable aid in convincing people of problems in the protocols and the need to repair them.

On the negative side, the lack of a formal semantics makes it difficult to resolve fine issues such as interaction between (new) language features.

4 Suggestions for improvements/extensions

Some of our suggestions have already found their way into Spin: the use of breakpoints and skipping part of the output on long, guided simulation runs.

Other things on our wish list:

Tools

- A mixture of simulation and validation (Spin).

  Sometimes one wants to validate only part of a (large) design. Then it would be handy if one could interactively steer the model to a particular part and run a validation from that point onwards. Unfortunately, the design of Spin strictly distinguishes between simulation and validation. Gerard Holzmann has suggested to use `gotos` to guide the model to a specific part, but that is not feasible in general. One would also have to fill channels and set variables to their correct initial values. If not done with care, an unreachable state might be the result. Interactive simulation then gives a better solution.

  Another solution would be the ability to take the synchronous product of the model with another one that does the steering. A never claim operates in this way, but it precludes the checking of some properties.

- Enhancement of the user interface (Xspin).

  One is typically interested in finding the shortest path to a certain error. The usual strategy is to use a kind of binary search on the search depth. It would save a lot of user time if this search process could be done at the press of one button. Similar repetitious jobs are cycling through the range of properties to be validated, and validating several never claims in sequence.

  A possible solution is to use a meta-language in which validation strategies can be programmed, cf. ML which originated as the "interface language" for the theorem prover LCF. Maybe Tcl/Tk already plays the role of such an interface language?

  A management system around Xspin would also be useful: to register models, validation runs, trails and outputs.

- Display/output of the reachable state space as a Finite State Machine. This is useful to check your intuition of language features on small models or as an interface to other tools.

Language extensions

- More structuring facilities, e.g.
  - A CSP/LOTOS/OCCAM-like parallel operator.

    The interleaving operator is simple, could be done by a macro (fork and wait); the synchronized one is more powerful and will be harder to implement. Of course the effectiveness of Promela should not be compromised, so an option is to provide this kind of language extensions by translators.
  - Procedures/functions instead of *cpp* macros.
  - More refined name-space hierarchy.

    Consider for instance a model of a distributed system with 2 protocol stacks. Currently it is not possible to separate the global data of one stack from the other, leading to large global data areas.

    A solution might be to allow subprocesses running in the same space as their creator. Syntax: proctypes declared within proctypes (cf. Pascal). Problem: a never claim may only access global data.
- The ability to specify for each channel separately whether it blocks or not.
- More control over exiting of processes.

Processes do not disappear immediately after their last action. This can be a nuisance when modelling a dynamic network of terminating processes. For example, MSCs quickly become too wide to handle.

- A minor issue, but a synchronous channel used only for synchronization (i.e. without passing data) could be written as `chan c = [0] of { }` and input/output on such a channel by `c?` and `c!`. Instead, Promela now requires a dummy data type, e.g. `chan c = [0] of { bit }` and input/output on such a channel by `c?0` and `c!0`.

Other issues

- A formal semantics of Promela, e.g. a structured operational semantics.
- Comprehensive documentation, including a tutorial describing all Promela/Spin features with examples of use, and a cookbook with examples of modelling idioms.
- Support for (discrete) time.
- Description of the various output formats to facilitate extraction of relevant information.

5 Concluding remarks

We feel that the use of formal methods has resulted in a better design. Some of the things we might do in the future:

- Support for testing.

  Promela may be used as a test specification language. It has many features similar to TTCN, the standardized notation for protocol tests. Test derivation algorithms, in particular for (nondeterministic) finite-state machines or labelled transition systems, could be used to derive Promela tests for Promela models. Alternatively, use could be made of test derivation algorithms for other languages like LOTOS, by translating Promela models into LOTOS.

- Script validation.

  The decision algorithm for the control of the barrier is not fixed, but can be changed. A special language has been designed to encode the algorithm. This encoding of the algorithm, called a "script", is then interpreted by the BOS system. Translation of scripts to Promela seems feasible, making it amenable to validation. Problem: modelling the environment of the script interpreter, which is basically the whole BOS system.

- Support for getting from design to C++ code.

  Code generation from Promela might be useful here.

- Validation of the (complete) design.

  Now that we have a (mostly) formal design, we might try to validate it.

Finally, some research subjects not mentioned already:

- Combined model checking and symbolic verification.

  Especially where models are parameterized, e.g. by channel capacity or by configuration tables, symbolic verification could be used to show that it is sufficient to validate a particular finite model.

  A related issue: include algorithms for checking boundedness of channels, and lift algorithms for establishing deadlock-freedom based on the communication pattern of processes to Promela (such algorithms exist e.g. for CSP and occam).

- Cheapest way to express a property to be validated.

  Since there are several ways to express correctness criteria in Promela, a theory that relates these criteria and allows to select the "cheapest" could be useful.

- Translation of various languages to Promela, to make model checking available for those languages.

For example from LOTOS to Promela or from Occam to Promela. Occam is another descendant of CSP (78) and hence shares many features with Promela. Notable differences: occam has `PAR` instead of `run`, procedures/functions (even locally declared ones), timers, priorities on guards, guards containing both a Boolean expression and an I/O guard, and variable length arrays as channel data. Moreover, occam 3 has modules, union types.

The major problem will be to find translations that produce small models.

- Investigation of the relation between the π-calculus and Promela (note that channels are treated as just another data type).

Acknowledgements Thanks to the BOS project team at CMG, The Hague, in particular the chief designer Klaas Wijbrans and Eric Burgers, who did most of the work. Thanks to Gerard Holzmann for his instant reaction to our questions and solutions to our problems. Thanks to the other members of the "BOS team" at the University of Twente: Ed Brinksma, Wil Janssen and Job Zwiers, and to my colleague Rom Langerak for discussions on Promela/Spin.