Issues in State Space Reduction

J.-Ch. Grégoire
INRS-Télécommunications

October 16th, 1995

1 Reflection

Is it worth repeating one more time the fundamental limitation of reachability analysis as a verification technique? Most industrial problems have state spaces too large for the memory capacity of our machines, in spite of the astounding progress made over the past decades in memory density and ever decreasing prices.

Various techniques have been studied and proposed to alleviate the problem, in two different directions. One consists of reducing the number of states visited by controlling the interleavings (the so-called "partial orders") or by exploring only parts of the state space (symmetry). Another method consists of reducing the amount of space required to store state information. If one is satisfied with a partial search, other alternatives are possible. Indeed, one of the major contributions of SPIN has certainly been the so-called *supertrace* algorithm, where state exploration is reduced to a partial random search by the simple exploitation of collisions in a hash table and by keeping only one or two bits per state in memory.

On the other front, the only mechanism originally available for reducing the number of explored states was the ATOMIC construct, which basically reduced the number of interleavings. The integration of a "partial order" mechanism only came much later. It is also worth mentioning, however, that communication channels were introduced as a primitive with the effect of both reducing the number and the size of the states.

More recently, a new construct for deterministic operations and identification of different classes of variables have made their appearance. But where do we go from here? My claim is that SPIN is becoming baroque and a deeper reflection on the need and nature of state reduction features is required.

2 Statement

I describe here some of the results of experiments performed at INRS-Télécommunications on various extensions of SPIN. Our goal is to expose some possible avenues of evolution of the PROMELA language and its tool.

My vision of PROMELA is of a language with two distinctive components. A *reactive* component has the nondeterministic constructs (if, do), the processes as well as the communications. A *deterministic* component is responsible for the computations. Each component

has its own sequencing structure and constructs. We also consider an extended type system, with the exclusion of pointers. The ultimate goal, obviously, is to bring a PROMELA-like modeling language closer to a "real" programming language.

In the following sections, I describe the salient features of such a language.

3 The language

The first distinctive character of the language is the separation of model from annotations and properties. One goal is to avoid to clutter a specification with verification information, which makes it more difficult to read. More fundamentally, there should be a clear separation between any semantic definition of the modeling language, and constructs specific to a finite-state automation semantic interpretation.

Such a separation might also facilitate the use of different verification engines from the same, unique specification.

Variables may belong to different categories, depending on their contribution to the state space or the verification itself. Variables can hold state information or be useful only for local computations (e.g. initialization loops in deterministic constructs). There is always redundancy in the state information. Such redundancy can sometimes be captured by a simple functional expression of several state variables.

Computation of the scope and impact on reactivity of a variable can be done automatically in some (a lot of?) cases, with dataflow and other techniques.

It is also possible to imagine other variables, which would hold information about the exploration itself, for statistical or performance purposes.

Operations are reactive or computational, following the Harel/Pnueli model. Computations should be of arbitrary complexity. The functional abstraction, in various guises, is the widely acknowledged way to encapsulate a computation, and it should be included in the language in some form. A computation sequencing construct is also required.

Functions can implement an ADT, which itself may have been formally verified, say algebraically, separately. The definition of the function can be internal to the modeling language, or external, which then requires a bridging mechanism.

It is worth noting that the combination of these two features gives a user a "poor man's partial order mechanism", and can help to achieve quite significant reductions in the number of states generated. However, there is an obvious tradeoff to observe here. Having a richer language, and more complex data types and structures will contribute to increasing the number of states, and we may have a no-win situation.

4 Tool

Variables can be stored in four different types of locations, depending on their purpose:

- in the state space store;

- on the stack (to allow backtracking);
- locally in deterministic blocs;
- globally in the verification program.

They can also be replaced by a function call.

Functions are supported in both internal and external form. If an external function needs a persistent context, it must be saved either in the store or on the stack, possibly in compressed form.

Compression hasn't been widely pursued in SPIN. There hasn't been any report, to my knowledge, of the effect of simple polynomial domain-based encoding of whole or parts of the state information. We have however experimented with a graph encoding of the state vector, to exploit the redundancy, with a qualified success. There is obviously potential for further research. The goal, obviously, is to push forward the limits of complete exploration of the state space, as opposed to the partial search mode of "supertrace".

The computational model is an extended Mealy machine, with conditions and actions evaluated and/or executed in a single transition.

Acknowledgments Inspiration for parts of this work came from M. Ferguson, F. Gagnon and B. Johnston.