Semantic Embedding of Promela-Lite in PVS

Shamim H Ripon* Alice Miller
* Department of Computing Science
University of Glasgow
{shamim,alice}@dcs.gla.ac.uk

1 Introduction

Promela-Lite [3] is a specification language which captures the essential features of Promela [4]. Unlike Promela a full grammar and type system of the language, and Kripke structure semantics of Promela-Lite specification have been defined and used to prove the correctness of automatic symmetry detection techniques used in Promela.

Mechanical verification is widely used as a tool to verify the properties of a language. It allows one to identify potential flaws in the language and gives confidence in the language definition. Theorem provers are heavily used as a tool to mechanically verify language properties. The language is to be embedded into the theorem prover for this purpose. Here we outline work in progress to embed Promela-Lite syntax and semantics into the general purpose theorem prover PVS [5] and use these embeddings to interactively prove both consistency with the syntax/semantics definitions and language properties.

2 Promela-Lite

Promela-Lite includes the core features of Promela including parameterised processes, channels (first class) and global variables, but omits some features such as rendez-vous channels, enumerated and record types, and arrays. The syntax of Promela-Lite is specified by using the standard BNF form [1] summarised in Figure 1(a). The language has primitive datatypes and channel types of the form $chan\{\overline{T}\}$, where $\overline{T}$ is comma separated list of types (details in [3]). A Promela-Lite specification consists of a series of channel and global variable declarations, one or more proctypes and an `init` process. Part of Promela-Lite syntax is shown as a production rule in Figure 1(b), where only the syntax of `expr` is presented.

$$
\begin{aligned}
\langle type\rangle ::= {} & \texttt{int} \\
| {} & \texttt{pid} \\
| {} & \langle chantype\rangle \\
| {} & \langle typevar\rangle \\
\langle chantype\rangle ::= {} & \langle recursive\rangle^{?}\ \texttt{chan}\ \{\langle type-list, `,'\rangle\} \\
\langle recursive\rangle ::= {} & \texttt{rec}\ \langle typevar\rangle \\
\langle typevar\rangle ::= {} & \langle name\rangle
\end{aligned}
$$

(a) Promela-Lite type systax

$$
\begin{aligned}
\langle expr\rangle ::= {} & \langle name\rangle \\
| {} & \langle number\rangle \\
| {} & \texttt{\_pid} \\
| {} & \texttt{null} \\
| {} & \texttt{len}(\langle name\rangle) \\
| {} & (\langle expr\rangle) \\
| {} & \langle expr\rangle \circ \langle expr\rangle \quad (\text{where} \circ \in \{+,-,*\})
\end{aligned}
$$

(b) Syntax for `expr`

Figure 1: Promela-lite syntax

A Promela-Lite specification is considered to be well-typed if its statements and declarations are well-typed according to the typing rules. The typing rules of Promela-Lite are defined by following the notation used in [2]. For example, the typing rule for $\langle expr\rangle \circ \langle expr\rangle$ in Figure 1(b) is defined as follows:

$$\frac{\Gamma \vdash e_1 : int \qquad \Gamma \vdash e_2 : int \qquad \circ \in \{+,-,*\}}{\Gamma \vdash e_1 \circ e_2 : int}$$

The semantics of a Promela-Lite specification $\mathcal{P}$ is denoted as a Kripke structure $\mathcal{M}$. If $\mathcal{P}$ is well-typed according to the typing rules then the Kripke structure $\mathcal{M}$ is well-defined. A function $eval_{p,i}$ is defined that evaluates an expression e for a process i at state s, where $proctype(i) = p$. For the syntax of `expr` mentioned earlier, $eval_{p,i}$ is used to evaluate the expressions at a given state as follows (details in [3]):

$$eval_{p,i}(s, e_1 \circ e_2) \;=\; eval_{p,i}(s, e1) \circ eval_{p,i}(s, e_2) \qquad \text{where } \circ \in \{+,-,*\}$$

*This author is supported by EPSRC grant EP/E032354/1

3 PVS Embedding

An embedding is a semantic encoding of one specification language into another. There are two main variants for the embeddings: *shallow* and *deep* embeddings [6]. In a shallow embedding, a program or a specification is translated into a semantically equivalent representation of the host logic. In a deep embedding, the language and the semantics are fully formalised in the logic of the specification language. This allows reasoning about the language itself, not just concrete programs.

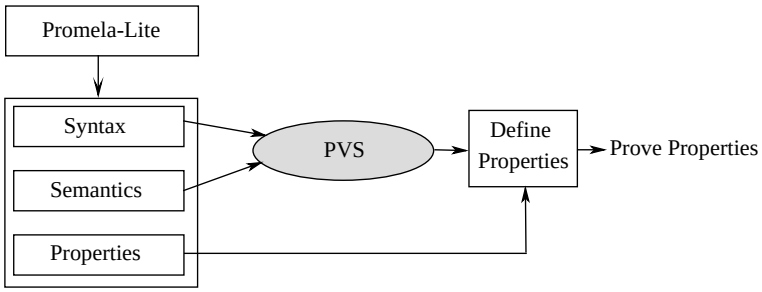

Figure 2: Mechanisation steps

Our mechanisation is based on deep embedding. Figure 2 briefly outlines the steps that we follow to mechanise Promela-Lite in PVS. The syntax of the language is defined using the abstract datatype mechanism of PVS which allows recursive definitions over the terms of the language. The definition is similar to the traditional use of BNF to define the syntax. The datatype definition enumerates constructors, lists their parameters, and provides recogniser predicates. Part of the datatype definition to define the syntax of `expr` ($\langle expr \rangle \circ \langle expr \rangle$) is shown here:

```
expr_syntax : DATATYPE
 BEGIN
  plus (e1:expr_syntax, e2:expr_syntax) : plus?
  minus(e1:expr_syntax, e2:expr_syntax) : minus?
  star (e1:expr_syntax, e2:expr_syntax) : star?
  ...
END expr_syntax
```

The semantics are defined recursively by following the original definitions. Care has to be taken to ensure that these definitions follow the typing rules. Following the definitions of how expressions are evaluated, a recursive definition is given to the semantics, part of which is shown as follows:

```
sem(e : expr_syntax, env: environment, s : STATE): RECURSIVE VALUE =
 CASES e OF
  plus(e1,e2) : integer_value(intvalue(sem(e1,env,s)) + intvalue(sem(e2,env,s))
  ...
 ENDCASES  MEASURE BY <<
```

After embedding the semantics, the properties to be proved will be defined in the form of theorems and supporting lemmas, and proved using the interactive theorem prover.

References

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers – Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[2] Luca Cardelli. *The Computer Science and Engineering Handbook*, chapter Type Systems, pages 2208 – 2236. CRC Press, Boca Raton, 1997.

[3] Alastair F. Donaldson and Alice Miller. Automatic symmetry detection for promela. *Journal of Automated Reasoning*, 41:251–293, 2008.

[4] Gerard J. Holzman. *The SPIN MODEL CHECKER: Primer and Reference Manual*. Addison-Wesley, 2003.

[5] Sam Owre, J.M. Rushby, and N Shankar. PVS: A Prototype Verification System. In Deepak Kapur, editor, *CADE'92*, volume 607 of *LNAI*, pages 748–752. Springer-Verlag, June 1992.

[6] R. Boulton, A. Gordon, M.J.C. Gordon, J. Herbert, and J. van Tassel. Experience with embedding hardware description languages in HOL. In *TPCD'93*, pages 129–156. North-Holland, 1993.