Natural language for hardware verification: semantic interpretation and model checking

Alexander Holt, Ewan Klein and Claire Grover
{alexander.holt,ewan.klein,claire.grover}@ed.ac.uk

HCRC Language Technology Group
Division of Informatics, University of Edinburgh
http://www.ltg.ed.ac.uk/

1 Overview

Our system allows the formal verification of digital circuits using specifications expressed in English. Verification is carried out by the SMV model checker program [McM92]. SMV requires specifications to be written in the temporal logic CTL (computation tree logic). SMV's model checking algorithm carries out inference over CTL formulas, with respect to a formal representation of a circuit.

The system can turn English sentences into CTL formulas, allowing natural language specifications to be used. A parser for English, returning general-purpose semantic representations, is allied with a convertor from these representations to CTL. The convertor is integrated with the SMV model checker, so that inferential information may be used during semantic interpretation.

The result exemplifies an interesting class of natural language understanding systems where the target is a specialised reasoning engine (as opposed, say, to a database). The semantic interpretation process exploits this engine, and the application as a whole delivers increased access to a significant industrial technique.

A web interface is available at http://www.ltg.ed.ac.uk/prosper/.

2 Background

2.1 Model checking for hardware verification

Mechanised formal specification and verification tools can significantly aid system design in both software and hardware. One established approach to verification is temporal model checking, which allows the designer to check that certain desired properties hold of the system. However, the normal

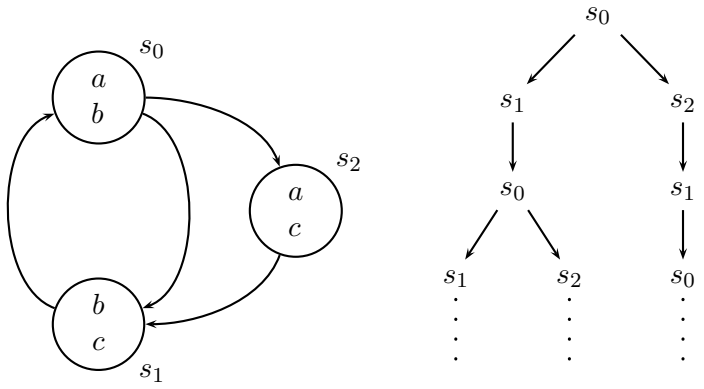

Figure 1: A CTL structure and corresponding computation tree

requirement that specifications be expressed in temporal logic has proved an obstacle to its adoption by circuit designers—hence the motivation for a natural language interface.

2.2 Computation Tree Logic

The SMV program implements a model checking algorithm where circuit properties are expressed in the temporal logic CTL [CES86]. In models of CTL, the temporal order $<$ defines a tree which branches towards the future.

CTL formulas that start with A express necessity. $\mathsf{AG}\, f$ is true at a time t just in case f is true along all paths that branch forward from the tree at t (true globally). $\mathsf{AF}\, f$ holds when, on all paths, f is true at some time in the future. $\mathsf{AX}\, f$ is true at t when f is true at the next time point, along all paths. Finally, $\mathsf{A}[f\ \mathsf{U}\ g]$ holds if, for each path, g is true at some time, and from now until that point f is true.

Figure 1, from [CES86], illustrates a CTL model structure, with the relation $<$ represented by arrows between circles (states), and atomic propositions being the letters contained in a circle. A CTL structure gives rise to an infinite computation tree, and the figure shows the initial part of such a tree when s_0 is the initial state. States correspond to points of time in the course of a computation, and branches represent non-determinism. Formulas of CTL are either true or false with respect to any given model; see Table 1 for three examples interpreted at s_0.

3 Example

Sentence (1), taken from our corpus of specification discourses, is ambiguous. The two CTL readings which our system assigns are shown in (2) and (2′).[1]

[1]In fact for this example the system currently chooses a 'weak' reading of until, expressible in terms of—but not equivalent to—CTL's 'strong' until ($\mathsf{A}[f\ \mathsf{U}\ g]$). The

formula	*sense*	*at* s_0
$\mathsf{AX}\, c$	for all paths, at the next state c is true	true
$\mathsf{AG}\, b$	for all paths, globally b is true	false
$\mathsf{AF}(\mathsf{AX}(a \wedge b))$	for all paths, eventually there is a state from which, for all paths, at the following state a and b are true	true

Table 1: Interpretation of CTL formulas

(1) After sig1 becomes active sig2 should not become active until sig3 becomes active.

$$\mathsf{A}[(\texttt{sig1} \rightarrow \mathsf{AX}(\neg\texttt{sig2}))\ \mathsf{U}\ \texttt{sig3}] \tag{2}$$

$$\mathsf{AG}(\texttt{sig1} \rightarrow \mathsf{AX}\,\mathsf{A}[\neg\texttt{sig2}\ \mathsf{U}\ \texttt{sig3}]) \tag{2'}$$

We can use SMV to test the truth of these readings for the circuit being specified, and rank false readings according to the length of the computation tree that SMV requires in order to generate a counter-example.

4 Architecture

4.1 System structure

The system consists of four components: (i) a parser, (ii) a convertor from semantic representations to CTL, (iii) the SMV model checker, and (iv) a module that mediates interaction between SMV and the semantic convertor. These are connected using Python (`http://www.python.org/`), and components (ii) and (iv) are also written in Python. The system currently runs under Solaris and Linux, and has a web interface, accessible from `http://www.ltg.ed.ac.uk/prosper/`.

4.2 Parser

We used the Alvey Natural Language Tools Grammar [GCB93] to implement a parser for a restricted subset of English. The Alvey parser is written in Common Lisp. The original broad-coverage grammar was modified by omitting large chunks of the lexicon and making numerous changes to specific grammar rules. The definition of an appropriate subset of English for this task raised interesting issues of its own, discussed in [HK99].

CTL translations given here use strong until instead to improve readability; this choice is independent of the ambiguity in question.

4.3 Conversion to CTL

This component converts the semantic representations produced by the Alvey system into CTL formulas. This process has some interesting semantic aspects. For example, temporal information must be converted from relations over Davidsonian event variables to modal operators. It is also necessary to deal with the unscoped nature of the Alvey semantics. This part of the system was initially written in Prolog, based on work by Danny Tidhar [Tid98].

In (3), (4) and (5) we show respectively a short sentence from our corpus, its Alvey semantics, and the CTL formula to which we convert it:

(3) After sig1 is active, sig2 is active for three cycles.

(4)
```
(DECL
 (AFTER (uqe (some (e1) e1))
  (BE (uqe (some (e2) (PRES e2)))
   (ACTIVE (name (the (x1) (and (sg x1) (named x1 sig1))))
    (degree unknown)))
  (and
   (BE #1=(uqe (some (e3) (PRES e3)))
    (ACTIVE (name (the (x2) (and (sg x2) (named x2 sig2))))
     (degree unknown)))
   (FOR #1# (uq ((NN \3) (x3) (and (pl x3) (CYCLE x3))))))
  (timespan unknown)))
```

(5) $$\mathsf{AG}(\mathtt{sig1} \rightarrow \mathsf{AX}(\mathtt{sig2} \wedge \mathsf{AX}\,\mathtt{sig2} \wedge \mathsf{AX}\,\mathsf{AX}\,\mathtt{sig2}))$$

4.4 SMV

The SMV model checker is a self-contained C program. As well as reading an input file and accepting command line arguments, it has a line-oriented interactive mode, allowing each SMV invocation to check an arbitrary number of CTL formulas against a given circuit description.

4.5 SMV/convertor interaction

This component uses UNIX pipes to communicate with SMV, exploiting its per-circuit interactive mode. Some work is necessary to interpret the output of SMV in the case of a model checking failure, when a counter-example of arbitrary length is returned in a plain text format.

5 Prospects

We are considering the adoption of a higher-order logic formalism for our semantic representations. This would permit new kinds of inference prior to

model checking, and would also allow a more natural conversion from the output of the parser.

The SMV program can only provide our semantic interpretation process with limited information about the success or failure of a model checking attempt. It would be interesting to experiment with model checkers that make a richer perspective available—game theoretic approaches, for example.

Acknowledgements

This work has been carried out within PROSPER (Proof and Specification Assisted Design Environments), ESPRIT Framework IV LTR 26241, `http://www.dcs.gla.ac.uk/prosper/`.

References

[CES86] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[GCB93] Claire Grover, John Carroll, and Ted Briscoe. The Alvey Natural Language Tools Grammar (4th release). Technical Report 284, Computer Laboratory, University of Cambridge, 1993. `ftp://ftp.cl.cam.ac.uk/nltools/reports/grammar.ps`.

[HK99] Alexander Holt and Ewan Klein. A semantically-derived subset of English for hardware verification. In *37th Annual Meeting of the Association for Computational Linguistics: Proceedings of the Conference: 20–26 June 1999, University of Maryland, College Park, Maryland, USA*, pages 451–456. Association for Computational Linguistics, 1999. `http://www.ltg.ed.ac.uk/prosper/papers/holt-1999-sds/`.

[McM92] K. L. McMillan. *The SMV system.* Carnegie-Mellon University, Pittsburgh, PA, 2 February 1992. `http://www.cs.cmu.edu/~modelcheck/smv/smvmanual.r2.2.ps`.

[Tid98] Dan Tidhar. ALVEY to CTL translation — A preparatory study for finite-state verification natural language interface. MSc dissertation, Department of Linguistics, University of Edinburgh, 1998. `http://www.ltg.ed.ac.uk/prosper/papers/tidhar-1998-act/`.