LCF-Style Bit-blasting in HOL4

Anthony Fox
University of Cambridge

Fixed width bit-vectors

- A wide range of bit-vector problems arise when verifying hardware and software.
- There are many useful bit-vector operations, including:
  - **Machine arithmetic** and **orderings** (signed and unsigned).
  - **Bitwise logic**.
  - Concatenation, extraction, repetition, shifts, rotations, extensions, conversions…

Bit-vectors in HOL4

- Wai Wong developed a theory of words in the 1990s. Based on **lists,** with **predicates** used to constrain word lengths.

- Early in 2002 a theory of 32-bit words was added to HOL. This was based on a **quotient type** construction.

- Later in 2002 this was generalised with an ML **functor**, which built a theory for any word length.

- In 2005 the current theory replaced these old developments. It uses **parametric polymorphism** (an idea by John Harrison).

Deciding bit-vector problems in HOL4

- This has evolved over time:
  - Started with very little support – a collection of theorems/ rewrites. (Proofs were often tedious and painful.)
  - Then more powerful simplification based tools were developed. (Some proofs were now trivial, but some still hard work.)
  - This rough diamonds paper shows how things have improved with the development of a bit-blasting based decision procedure.

Deciding bit-vector problems

- I received e-mails of the form: How do I solve this apparently simple 32-bit problem?

$$(\mathbf{0x3} \,\&\&\, x = \mathbf{0x0}) \wedge x \leq_+ y \implies x \leq_+ (\neg\mathbf{0x3} \,\&\&\, y)$$

  The current simplifications don't seem to simplify this goal?

- In this case we could add more (domain specific) simplification rules…

- However, combinations of arithmetic and bitwise operations can be problematic. Something more general would be nice.

Bit-blasting

- Bit-blasting is a more general approach for solving bit-vector problems.

- The basic idea is to convert a finite bit-vector problem into a propositional formula. Then call a SAT solver.

- This is trivial for bitwise operations. For example, to solve

$$\neg(x \,\&\&\, y) = \neg x \parallel \neg y$$

one simply needs to show

$$\neg(x(i) \wedge y(i)) = \neg x(i) \vee \neg y(i)$$

where *i* ranges over bit positions.

Bit-blasting

- Linear arithmetic can be covered by considering ripple carry addition.

- Propositions can become very large but fortunately Hasan Amjad's HolSat development works well here. (MiniSat is used as an efficient external SAT solver, with proofs reconstructed in HOL.)

- Still important to efficiently generate propositions from bit-vector goals.

- Some goals can be discharged during an initial simplification stage, avoiding the need for bit-blasting.

Performance and Limitations

- In practice, the tool has reasonable performance. For example, our 32-bit goal

$$(\mathbf{0x3} \,\&\&\, x = \mathbf{0x0}) \wedge x \leq_+ y \implies x \leq_+ (\neg\mathbf{0x3} \,\&\&\, y)$$

is solved in 0.15s. With 128-bit words this takes 1.8s.

- Obviously this brute force approach does suffer with certain goals (those generating very large propositions).

- Bit-blasting **non-linear arithmetic** is not supported (except multiplication for very small word sizes). Also bit extraction with *variable* bounds.

SMT?

- SMT provers, such as **Z3**, are very capable with respect to bit-vector problems. Tjark Weber has implemented proof reconstruction for SMT in HOL4.
- So why not rely on SMT alone?
  - ⋆ SMT solvers are not distributed with HOL and it can be a hassle installing them. (Z3 is developed for Windows.)
  - ⋆ Z3 is closed source and is being constantly developed. It's a real effort to maintain proof reconstruction tools.
  - ⋆ **At the moment many bit-vector proofs cannot be re-constructed** due to insufficient details, or syntax errors, in Z3's proof output.
  - ⋆ HOL's LCF approach offers a very high level of assurance, which is lost **when proofs are not re-constructed** (i.e. in Oracle mode). The approach is then vulnerable to bugs in Z3 or in the translation process.

Summary

- The new tool is already in use, offers good coverage, is fully automatic and in many cases has reasonable performance.
- It provides counterexamples, which can be very instructive.
- The current approach is very simple.
- More advances techniques could be employed to cope with hard problems, e.g. heuristics, abstractions (laziness), more simplification, …
- It may be beneficial to produce a standalone tool, adopting an integration approach similar to that of Metis.
- It is likely that any future improvements would be application/ demand driven.

Questions?