Improved Automation for SPARK Verification Conditions

Paul Jackson and Grant Olney Passmore

Paul.Jackson@ed.ac.uk,G.Passmore@ed.ac.uk

Background

SPARK is a subset of Ada which adds in assertions such as pre-conditions, post-conditions and loop invariants. It is used for high-integrity applications in the aerospace, defence, rail and security industries.

The current automatic prover for SPARK program verification conditions (VCs) typically proves fewer than 100% of VCs. Proving the rest by manual means or interactive prover takes specialist skills and is very tedious. Typically assertions are restricted to checks of exception freedom in order to keep the level of proof automation reasonable.

Goal

The goal is to broaden the range of automatically-checkable properties, thus speeding VC verification and supporting use of richer assertions.

SMT solvers

An initial line of investigation has been to see how well SMT (SAT modulo theories) solvers perform on SPARK VCs. SMT solvers are a new breed of automatic prover, well suited to the kinds of problems arising in VCs. For example they can handle linear arithmetic over the integers and reals, bit-vector operations, and datastructures such as records and arrays. They leverage off SAT technology for propositional reasoning and combine decision procedures for different theories using a generic Nelson-Oppen architecture.

VC Translator

A tool has been developed for translating SPARK VCs into a variety of different formats. The translation is 2 phase, firstly to a prover-independent standard form with full typing information, secondly to particular prover languages. The translator can work in an offline mode where translated VCs are written to files or an online mode where solvers are immediately invoked on the translated VCs. For the online mode, results are output in a database-friendly tabular format, and analysis is aided by command-line functions for merging and filtering tables.

Provers Considered

- **Z3**: SMT solver from Microsoft
- **Yices**: SMT solver from SRI
- **CVC3**: SMT solver from NYU and U. Iowa.
- **Simplify**: Legacy prover. Used in ESC/Java.
- **Simplifier**: Current VC prover from Praxis UK.

Case Studies

	Autopilot	**Simulator**	**Tokeneer**
Lines of code	1075	19259	30441
No. funcs & procs	17	330	286
No. annotations	17	37	114
No. VCs	133	1806	1880

Tokeneer is a just-released case study commissioned by NSA for evaluating Praxis's SPARK-based high-integrity software development methodology.

Experimental Results

SMT solvers can be 10× faster than Simplifier and can provide comparable coverage.

	Z3	Yices	CVC3	Simplify	Simplifier
Autopilot	96.2	95.5	96.2	96.2	97.0
Simulator	94.5	93.9	94.5	92.6	95.5
Tokeneer	94.6	94.8			94.1

Coverage of VCs (%):

Simplifier's better coverage is because of its better support for non-linear arithmetic.

	Z3	Yices	CVC3	Simplify	Simplifier
Autopilot	29	9	248	24	44
Simulator	32	12	452	26	163
Tokeneer	46	25			197

Timing (msec per VC)

VC Examples

- Proven only by CVC3 and Simplifier

```
H1:    s >= 0 .
H2:    s <= 971 .
        ->
C2:    43 + s * (37 + s * (19 + s)) <= 214783647 .
```

- Proven only by Simplifier

```
      ->
C1:   (e1 mod 971) * (e2 mod 971)  >= 0 .
C2:   (e1 mod 971) * (e2 mod 971)  <= 2147483647 .
```

- Unproven by any

```
H1:    f > 0 .
H2:    f <= 100 .
H3:    v >= 0 .
H4:    v <= 100 .
       ->
C1:    (100 * f) div (f + v) <= 100 .
```

Remarks on Experiments

- **Supplementary axioms**: It was useful to provide such axioms as

$$\forall x, y : \mathbb{Z}.\ 0 < y \ \Rightarrow\ 0 \leq x \bmod y$$

$$\forall x, y : \mathbb{Z}.\ 0 \leq x \ \wedge\ 0 < y \ \Rightarrow\ y \times (x \,\mathsf{div}\, y) \ \leq\ x$$

- **Soundness:** Simplify will 'prove'

```
(IMPLIES
    (EQ x 2000000000)
    (EQ (+ x x) (- 294967296)))
```

  because of use of machine arithmetic. A common workaround of making large constants symbolic was found to reduce but not eliminate instances of unsoundness.
- **Exceptions**: While Z3 and Yices were generally robust, CVC3 and Simplify sometimes aborted on run-time assertion failures, and CVC3 occasionally had a segmentation fault.

Current Activities

- Building bridges with SPARK users
- Deeper analysis of prover incompletenesses.
- Further case studies
- Release of VCs in SMT-LIB, HOL, PVS formats
- Release of VC translator

Possible Directions

- Invariant generation (following Ellis and Ireland?)
- Counter-example presentation
- Proof explanation
- Tool integration architectures

Research Priorities

- Developing a HOL-Light based VC prover, integrating SMT solvers and other automated provers
- Improving support for non-linear arithmetic

Non-Linear Arithmetic

Support is desired for features such as integer division and modulus, non linear multiplication, transcendental functions, axiomatically characterised functions (e.g. square root). While problems can be in general undecidable, often actual problems have structure that heuristics can exploit.

Currently, Passmore has a system RAHD (Real Algebra in Higher Dimensions) which heuristically combines Gröbner basis calculations, semi-definite programming and real quantifier elimination (QEPCAD). He is developing RAHD while visiting SRI: RAHD is already integrated into an internal version of SRI's theorem prover PVS and there are plans to integrate it into the Yices SMT solver.

Passmore is working through a corpus of problems from John Harrison and others, and is proving goals that up till now were unproven by automatic means.

School of informatics