Uniform Reduction to SMT

Predrag Janičić Filip Marić
www.matf.bg.ac.rs/~janicic www.matf.bg.ac.rs/~filip

Automated Reasoning GrOup (ARGO)
Faculty of Mathematics
University of Belgrade, Serbia

Synthesis, Verification, and Analysis of Rich Models (SVARM 2010)
Edinburgh, July 20/21, 2010.

Motivation

- SAT/SMT solvers are widely used, but encoding to SAT/SMT is typically made by special-purpose tools
- There are interchange formats for SAT/SMT (e.g., SMT-lib) but no high-level specification languages
- Goal: Build a new modelling and solving system (for CSP, verification problems, etc.) with:
  - simple but expressible, high-level specification language
  - efficient interface to powerful SAT/SMT solvers

The Basic Idea

- Consider problems of the form: find values that satisfy given conditions
- It is often hard to develop an efficient specialized procedure that finds required values
- It is often easy to specify an imperative test if given values satisfy the conditions
- Such test can be a problem specification itself
- Convert this imperative specification to a SAT/SMT formula and use solvers to search for its models

Toy example

- *Alice picked a number and added 3. Then she doubled what she got. If the sum of the two numbers that Alice got is 12, what is the number that she picked?*
- A simple test cthat A is indeed Alice's number:

```
B:=A+3;
C:=2*B;
assert(B+C==12);
```

- This test is a specification of the problem
- Unknowns are exactly the variables that were accessed before they were assigned a value

Expressiveness

- The C-like specification language supports:
  - integer and Boolean data types; arrays
  - implicit casting
  - arithmetical, logical, relational and bit-wise operators
  - flow-control statements (if, for, while)
  - defined and undefined functions
- Restriction: conditions in the `if`, `for`, `while` statements and array indices must be ground (cannot contain unknowns)

Interpretation

- Specifications are symbolically executed
- The semantics is different from the standard semantics of imperative languages (e.g., undefined variables can be accessed)
- The result of the interpretation is a quantifier free FOL formula
- This formula is passed to a SAT/SMT solver
- If it is satisfiable, its models give solutions of the problem

Toy Example

- Consider the code:

```
nB=nA+3;
nC=2*nB;
assert(nB+nC==12);
```

- If A corresponds to the unknown `nA`, then the asserted expression is evaluated to $A + 3 + 2 * (A + 3) == 12$
- An SMT solver (e.g., for BVA or LIA) can confirm that the formula is satisfiable (and is true for A equals 1)

Overall Architecture

URSA MAJOR problem specification

↓ interpreter

Quantifier free FOL formula

↓ bitblasting

Propositional formula

↓ SAT solver ↓ SMT (BVA,LA,...) solver

Values of unknowns/Solutions

Implementation

- The tool URSA Major (Uniform Reduction to SAtisfiability Modulo Theory)
- Implemented in C++
- Employs a subsystem for bitblasting and reduction to SAT
- Currently: SAT solvers – ArgoSAT and Clasp, SMT (BVA, LIA, EUF, ...) solvers – MathSAT, Yices, Boolector
- Under constant development (support for new underlying theories and solvers being added)

CSP Example: The Eight Queens Puzzle

```
nDim=8;
bDomain = true;
bNoCapture = true;
for(ni=0; ni<nDim; ni++) {
   bDomain &&= (n[ni]<nDim);
   for(nj=0; nj<nDim; nj++)
      if(ni!=nj) {
         bNoCapture &&= (n[ni]!=n[nj]);
         bNoCapture &&= (ni+n[nj]!=nj+ n[ni]) && (ni+n[ni] != nj+n[nj]);
      }
}
assert(bDomain && bNoCapture);
```

Verification Example: Bit-counters

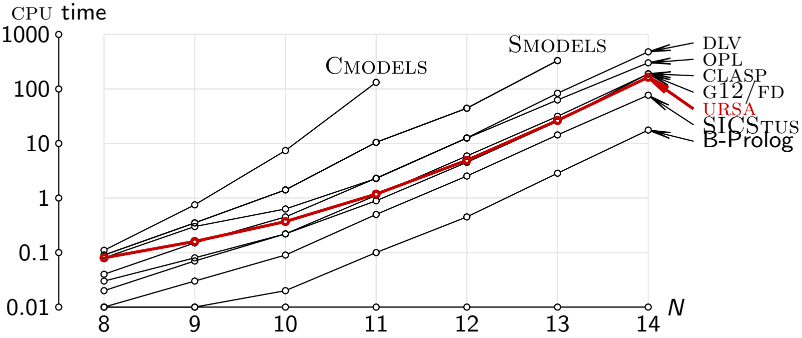
```
function nBC1(nX) {
   nBC1 = 0;
   for (nI = 0; nI < 16; nI++)
      nBC1 += nX & (1 << nI) ? 1 :  0;
}
function nBC2(nX) {
   nBC2 = nX;
   nBC2 = (nc2 & 0x5555) + (nc2>>1 & 0x5555);
   nBC2 = (nc2 & 0x3333) + (nc2>>2 & 0x3333);
   nBC2 = (nc2 & 0x0077) + (nc2>>4 & 0x0077);
   nBC2 = (nc2 & 0x000F) + (nc2>>8 & 0x000F);
}
assert(nBC1(nX)!=nBC2(nX));
```

Best Underlying Solver?

- There is no best underlying solver
- Each of the used solvers was most efficient for some problem
- This shows that different solvers should be used within the system
- For instance, for the magic square problem and the queens problem SAT solver Clasp was the most efficient

Sample Experimental Data

Problem: N queens problem (all solutions)



Conclusions

- A novel (imperative-declarative) programming paradigm
- The user controls the encoding employed
- Applicable to a wide range of problems (e.g., for all NP problems there is a simple witness test)
- Competitive to other modelling systems
- A high level interface to SMT
- Can be used for producing benchmarks

Current and Further Work

- Support for more theories and SAT/SMT solvers
- Providing APIs for standard programming languages
- Real-world applications
- Link to Rich Model Language?