SMT Solvers for Termination Provers

Johannes Waldmann
HTWK Leipzig, Germany

Constraint Solving

Programming actually is constraint solving:
the algorithm produces an output that fulfils the specification ($=$ the constraints).

- most programs act as application-specific constraint solvers,
- clever programs transform the constraint system and hand it to a general-purpose constraint solver
- pro: benefit from the built-in knowledge
- con: overhead due to the transformation (forth and back)

Constraints from Termination

applications:

- precedences $\Rightarrow$ finite domain constraints
- DP graph approximation $\Rightarrow$ reachability constraints
- coefficients for (polynomial, matrix) interpretations $\Rightarrow$ arithmetical constraints

approaches:

- direct encoding in suitable logic
- transform to bitvector logic
- transform to propositional logic (bit-wise)

SMT-lib and -comp

- SMT (satisfiability modulo theory)
- quantor-free first-order logic over numbers, bitvectors, arrays, terms, . . .
- library of benchmarks
- competition for solvers
- typical participants: Barcelogic (UPC), Boolector (Linz), Yices (Stanford), Z3 (Microsoft)

Preprocessing (I)

from term (rule) to its symbolic interpretation:
apply compression:

- replace repeated context $f_k(\dots, g_l(\dots), \dots)$
- by fresh function symbol h_{k-1+l}
- and constrain its interpretation

```
D (* (x, y))
    -> + (* (y, D (x)), * (x, D (y)))
a (x, y) = * (x, D (y))
D (* (x, y)) -> + (a (y, x), a (x, y))
```

nice for DP-transformed systems (lots of identical

Preprocessing (II)

The (SMT/SAT) solver should be able to recognize identical subexpressions?
Yes, but it doesn't know that multiplication is associcative.

```
a a b b -> b b b a a a
    a2 = a a ; b2 = b b
a2 b2 -> b b2 a a2
```

Remark (not implemented in Matchbox): should take into account the cost of matrix multiplication: top symbols are row vectors, instead of $A(bc)$ compute $(Ab)c$.

Polynomials

```
(benchmark termination   :logic QF_NIA
   :extrafuns ((n0 Int)) :assumption (>= n0 0)
   :extrafuns ((n1 Int)) :assumption (>= n1 1)
   :extrafuns ((n2 Int)) :assumption (>= n2 0)
   :extrafuns ((n3 Int)) :assumption (>= n3 1)
   :extrafuns ((n4 Int)) :assumption (= n4 (* n1 n2))
   :extrafuns ((n5 Int)) :assumption (= n5 (+ n0 n4))
   :extrafuns ((n6 Int)) :assumption (= n6 (* n1 n3))
   :extrafuns ((n7 Int)) :assumption (= n7 (* n3 n0))
   :extrafuns ((n8 Int)) :assumption (= n8 (+ n2 n7))
   :extrafuns ((n9 Int)) :assumption (= n9 (* n3 n1))
   :assumption (>= n5 n8)  :assumption (>= n6 n9)
   :assumption (> n5 n8)   :formula true)
```

Polynomials: Performance

SMT logic: `QF_NIA`, not in recent competition, not (reliably) supported by solvers

- yices rejects
- Z3 (2008): (many) false positives
- Z3 (2009): (some) false negatives, slow

submitted several NIA benchmarks

- sat, by bitblasting
- unsat, by derivational complexity:
  $\{bc \to cbb, ba \to acb\}$

Arctic Arithmetic (LIA)

from arctic matrix interpretations, domain $= \mathbb{N} \cup \{-\infty\}$,

```
   :extrapreds ((f3))   :extrafuns ((c4 Int)
```

max

```
:ep ((f79))   :as (iff f79 (or f75 f77))
:ef ((c80 Int)) :as (= c80
  (ite f75 (ite f77 (ite (> c76 c78) c76 c78
```

plus

```
   :ep ((f55)) :as (iff f55 (and f3 f19))
   :ef ((c56 Int))
   :as (= c56 (ite f55 (+ c4 c20) 0))
```

Arctic Performance

Barcelogic, Yices, Z3: fast and powerful for small systems (matrix dim 2),
for dim ≥ 3 need several minutes.

Note: get “below zero” for free:
in RTA08 paper, replace “$> -\infty$” by “≥ 0”,
“somewhere finite” $\Rightarrow$ “somewhere positive”
domain still is $\mathbb{N} \times (-\infty \cup \mathbb{Z})^*$

Arctic Trickery

intention: drop the extra boolean (for $-\infty$):

- encode $-\infty$ by some very low value
  - introduce bounds $S \ll L$,
  - constrain unknowns x by $x < -L \vee -S < x < S$,
  - standard operations max,plus
  - $x \geq' y \iff x \geq y \vee x < -L + kS$
- omit $-\infty$ (all values are $> \infty$)

  seems to be good enough in practice

Difference Logic (IDL)

from matchbounds
special treatment of $\pm\infty$ in final comparison (not in intermediate operations)

Z001 is match-bounded

a match-bound certificate for $\{a^2b^2 \to b^3a^3\}$

```
/5 5 + 4 + + 5 5 +\ /1 - + + + + + + +\
|+ + + + + + + 2 +| |+ + + + + + + + +|
|5 5 + 5 + + 5 5 +| |5 5 + 5 + + 5 - +|
|5 - + 4 + + 4 5 +| |5 - - + 5 + + + +|
|- 5 - - + - 5 - +| |1 4 + 2 + + 5 1 +|
|5 - + 3 + + - 5 +| |- - - - - - - - -|
|- 4 + 1 + + 4 - +| |5 - 3 + 5 + + + +|
|+ - + + + + + + +| |4 - 1 + - + + + +|
\- - - - + - - - +/ \- - - - - - - - -/
```

Z001 is match-bounded (II)

```
aabb                           bbbaaa
/5 5 + 5 + + 5 4 +\  /4 4 + 4 + + 4 2 +\
|+ + + + + + + + +|  |+ + + + + + + + +|
|5 5 + 5 + + 5 5 +|  |4 2 + 4 + + 4 4 +|
|2 4 + 2 + + 5 2 +|  |- 3 + 1 + + 4 - +|
|5 5 + 5 + + 5 3 +|  |4 2 + 4 + + 4 2 +|
|1 1 + 2 + + 5 1 +|  |- - + 1 + + 4 - +|
|4 4 + 4 + + 5 4 +|  |3 3 + 3 + + 4 3 +|
|2 4 + 2 + + 5 2 +|  |1 3 + 1 + + 4 1 +|
\1 1 + 2 + + 5 1 +/  \- - + 1 + + 4 - +/
```

Bitvectors

arithmetic operations (plus, times) and comparisons for bitvectors of fixed length

problem: these are silently overflowing.

solution: additional constraints to detect overflow

multiplication:

naive: use double bit width, restrict leading bits

clever: $[x_0, \ldots, x_{n-1}] \times [x_0, \ldots, x_{n-1}]$ iff $\exists i, j : i + j \geq n \wedge x_i x_j = 1$ or the MSB of the computation with $(n + 1)$ bits is set.

better: add non-overflowing operations to the BV specification

Bitblasting

simulate the usual circuits for bitwise addition and multiplication
then use SAT solver (minisat, picosat)
this can be better than SMT solver for BV

- built-in handling of overflow
- specific optimizations for small bit widths

Optimized Bitblasting

(Matchbox since 2008, with help by Peter Lietz)
idea: for a given operation, e.g.,
$[p1, p2]_2 \times [p3, p4]_2 = [p5, p6]_2$

- generate canoncical CNF (from truth table)
- find minimal equivalent CNF
- use linear integer programming solver to find the best set of clauses

$(p3 \vee \neg p5) \wedge (p3 \vee p4 \vee \neg p6) \wedge (\neg p2 \vee \neg p4) \wedge (\neg p2 \vee \neg p3 \vee p6) \wedge (p2 \vee p4 \vee \neg p6) \wedge (\neg p1 \vee \neg p4 \vee p6) \wedge (\neg p1 \vee \neg p3 \vee p5) \wedge (p1 \vee \neg p5) \wedge (p1 \vee \neg p4 \vee \neg p6)$

minisat preprocessor removes $< 1\%$

Optimized Bitblasting (II)

This approach is really limited to bit widths ≤ 4.
Above that, need auxiliary variables.
Then CNFs are harder to optimize
(if no semantics for additional variables is prescribed)

Applied Bitblasting

choose any binary encoding for arctic numbers,
e.g., $[-\infty, 0, 1, \ldots, 6] \leftrightarrow [0, 1, \ldots, 7]$
or below zero, e.g.,
$[-\infty, -2, -1, \ldots, 4] \leftrightarrow [0, 1, \ldots, 7]$
then compute optimized CNFs from truth tables of operations, as described. does not need any special treatment for $-\infty$.

Rational Bitblasting

can even handle rationals. Again, choose some bijection
$[0, 1/4, 1/3, 1/2, 1, 3/2, 2, 5/2] \leftrightarrow [0 \dots 7]$
and then compute optimized CNFs for best lower and upper approximation

can even use $\top$ (“very large number”),
makes functions total,
but $\top \neq \top$

Finite Domains for Matrices?

nice “theoretical” problem:
given arbitrary (partial) functions $\oplus$ and $\otimes$ on some finite domain, can they be used for matrix termination proofs?
I.e., what are the necessary axioms? (ring axioms $+$ strictness surely, but need to discuss “missing” values)

Conclusion

recommendation for termination provers:

- LIA (arctic) for dimension 1 and 2
- NIA (standard) via BV (larger width) for dimension 1 and 2
- NIA+LIA via BV via SAT (widht 3) for larger dimension

Implementation Remarks

how to call external solvers (SAT, SMT)?

- textual (stdin, stdout)
- API

Formatting for SMT solvers

- input is well-defined (SMT)
- output is not

(components of) termination provers should be usable as components for other software, too.