# Complete integer decision procedures as derived rules in HOL

Michael Norrish

Michael.Norrish@nicta.com.au

NATIONAL ICT AUSTRALIA

# Introduction

- Historically, theorem-provers have provided incomplete methods for universal Presburger arithmetic over $\mathbb{N}$ and $\mathbb{Z}$
- Alternating quantifiers not handled at all
- Performance of complete methods can be acceptable:
  - Omega Test's performance on goals proved by Fourier-Motzkin variable elimination (used in HOL, Isabelle/HOL and Coq), should be identical.
- Provide illustration of implementation techniques for derived rules in LCF-like setting
- Will cover Omega Test (paper also describes Cooper's algorithm)

# Presburger formulas

$$
\begin{array}{lcl}
\textit{formula} & ::= & \textit{formula} \wedge \textit{formula} \;\mid\; \textit{formula} \vee \textit{formula} \;\mid \\
& & \neg \textit{formula} \;\mid\; \exists \textit{var}.\, \textit{formula} \;\mid\; \forall \textit{var}.\, \textit{formula} \;\mid \\
& & \textit{numeral} \,|\, \textit{term} \;\mid\; \textit{term relop term} \\
\textit{term} & ::= & \textit{numeral} \;\mid\; \textit{term} + \textit{term} \;\mid\; -\, \textit{term} \;\mid \\
& & \textit{numeral} * \textit{term} \;\mid\; \textit{var} \\
\textit{relop} & ::= & < \;\mid\; \leq \;\mid\; = \;\mid\; \geq \;\mid\; > \\
\textit{var} & ::= & x \;\mid\; y \;\mid\; z \ldots \\
\textit{numeral} & ::= & 0 \;\mid\; 1 \;\mid\; 2 \ldots
\end{array}
$$

# Presburger formulas

*formula* ::= *formula* $\wedge$ *formula* | *formula* $\vee$ *formula* |
$\neg$*formula* | $\exists$*var. formula* | $\forall$*var. formula* |
*numeral* | *term* | *term relop term*

*term* "is divisible by" *numeral*

*term* ::= *numeral* | *term* $+$ *term* | $-$ *term* |
*numeral* $*$ *term* | *var*

*relop* ::= $<$ | $\leq$ | $=$ | $\geq$ | $>$

*var* ::= $x$ | $y$ | $z \ldots$

*numeral* ::= $0$ | $1$ | $2 \ldots$

# FMVE Basics in a Slide

Over $\mathbb{R}$ (or $\mathbb{Q}$), with $c, d > 0$:

$$(\exists x : \mathbb{R}.\ a \leq cx \land dx \leq b) \equiv ad \leq bc$$

($\Rightarrow$: from transitivity of $\leq$. $\Leftarrow$: pick $x$ to be $\frac{b}{d}$.)

Provides a quantifier elimination procedure for $\mathbb{R}$

- extends to multiple inequalities

  $$\text{\# of constraints on RHS} = (\text{\# of upper bounds})(\text{\# of lower bounds})$$

- extends to handle $<$

# FMVE for $\mathbb{Z}$?

- Central theorem is false:

$$(\exists x : \mathbb{Z}.\ 3 \leq 2x \leq 3) \not\equiv 6 \leq 6$$

- But one direction still works:

$$(\exists x.\ a \leq cx \land dx \leq b) \Rightarrow ad \leq bc$$

- Thus an incomplete semi-procedure for universal formulas over $\mathbb{Z}$:
  1. Compute negation: $(\forall x.\ P(x)) \equiv \neg(\exists x.\ \neg P(x))$
  2. Compute consequences: if $(\exists x.\ \neg P(x)) \Rightarrow \bot$ then $(\exists x.\ \neg P(x)) \equiv \bot$ and $(\forall x.\ P(x)) \equiv \top$
- This is Phase 1 of the Omega Test (when there are no alternating quantifiers)

# Some Shadows

Given $\exists x. (\bigwedge_i a_i \leq c_i x) \wedge (\bigwedge_j d_j x \leq b_j)$

- The formula

  $$\bigwedge_{i,j} a_i d_j \leq b_j c_i$$

  is known as the *real shadow*.
- If all of the $c_i$ or all of the $d_j$ are equal to 1, then the real shadow is *exact*
- If the shadow is exact, then the formula can be used as an equivalence.

# Exact Shadows

- When $c = 1$ or $d = 1$, the core theorem

$$(\exists x : \mathbb{Z}.\ a \leq cx \wedge dx \leq b) \equiv ad \leq bc$$

  is valid because
  - $\Rightarrow$: transitivity still holds
  - $\Leftarrow$: take $x = b$ if $d = 1$, $x = a$ if $c = 1$
- Pugh claims many problems in his domain have exact shadows. Experience suggests the same is true in interactive theorem-proving.

# Dark Shadows

- The formula

$$\bigwedge_{i,j} (c_i - 1)(d_j - 1) \leq b_j c_i - a_i d_j$$

  is known as the *dark shadow*. (NB: if all $c_i$ or all $d_j$ are one, then this is the same as the real shadow.)
- The real shadow provides a test for unsatisfiability
- The dark shadow tests for satisfiability, because

$$(c - 1)(d - 1) \leq bc - ad \Rightarrow (\exists x.\ a \leq cx \wedge dx \leq b)$$

  (proof in paper)
- This is the Phase 2 of the Omega Test

# Splinters—I

- Purely existential formulas are “often”
  - proved false by their real shadow; or
  - proved true by their dark shadow
- But in “rare” cases, the main theorem is needed. Let $m$ be the maximum of all the $d_j$s. Then

$$(\exists x. (\bigwedge_i a_i \leq c_i x) \wedge (\bigwedge_j d_j x \leq b_j)) \;\equiv$$

$$(\bigwedge_{i,j} (c_i - 1)(d_j - 1) \leq b_j c_i - a_i d_j)$$

$$\vee$$

$$\bigvee_i \bigvee_{k=0}^{\lfloor \frac{mc_i - c_i - m}{m} \rfloor} \left( \exists x. \begin{array}{c} (\bigwedge_i a_i \leq c_i x) \wedge (\bigwedge_j d_j x \leq b_j) \wedge \\ (c_i x = a_i + k) \end{array} \right)$$

# Splinters—I

- Purely existential formulas are "often"
  - proved false by their real shadow; or
  - proved true by their dark shadow
- But in "rare" cases, the main theorem is needed. Let $m$ be the maximum of all the $d_j$s. Then

$$(\exists x. (\bigwedge_i a_i \leq c_i x) \wedge (\bigwedge_j d_j x \leq b_j)) \equiv$$

$$(\bigwedge_{i,j} (c_i - 1)(d_j - 1) \leq b_j c_i - a_i d_j)$$

$$\vee$$

$$\bigvee_i \bigvee_{k=0}^{\lfloor \frac{m c_i - c_i - m}{m} \rfloor} \left( \exists x. \begin{array}{c} (\bigwedge_i a_i \leq c_i x) \wedge (\bigwedge_j d_j x \leq b_j) \wedge \\ (c_i x = a_i + k) \end{array} \right)$$

dark shadow

a splinter

# Splinters—II

- A splinter

$$\exists x.\ (\bigwedge_i a_i \leq c_i x) \wedge (\bigwedge_j d_j x \leq b_j)\ \wedge (c_i x = a_i + k)$$

*does* represent a smaller problem than the original because the extra equality allows $x$ to be eliminated.

- When quantifiers alternate, and there is no exact shadow, the main theorem is used as an equivalence, and splinters can't be avoided.
- Splinters must also be checked if neither real nor dark shadows decide a goal.

# Implementations in HOL

**Theorem instance re-proof:** The proof of the technique's "main theorem" is played out for each problem instance. (Used to implement Cooper's algorithm; see paper.)

***Pro forma* theorems:** The "main theorem" is proved once and for all, and is instantiated with each problem.

**External proof discovery:** An external tool finds a proof that can then be replayed in HOL. If proof search dominates this can be very effective.

# External proof discovery

- External proof discovery works best when proofs are short, but finding a proof is slow
- Manipulating logical formulas in the HOL kernel is always “slow” if it can be done elsewhere (in a C program?) instead
- Proofs *are* short in our domain:
  - Prove an existential formula valid by providing witnesses
  - Prove an existential formula invalid by specifying the chain of $\leq$-transitivity inferences that leads to $\perp$
- External proofs only for formulas with no alternation of quantifiers

# Shadow computation in ML

Provide an ML function that takes a vector of constraints and returns a result:

```
datatype 'a result =
    CONTR of 'a deriv
  | SATISFIABLE of Arbint.int PIntMap.t
  | NO_CONCL
```

A derivation is a proof of $0 \leq c_1 x_1 + \ldots + c_n x_n + c$

```
datatype 'a deriv =
    ASM of 'a
  | REAL_COMBIN of int * 'a deriv * 'a deriv
  | GCD_CHECK of 'a deriv
  | DIRECT_CONTR of 'a deriv * 'a deriv
```

Code can be completely decoupled from HOL.

# Replaying proofs

- With witnesses: instantiate input formula and peform ground reduction to check
- With proof tree for refutation: small piece of ML code plays out corresponding proof in HOL kernel
- If ML code returns `NO_CONCL` or if check fails, resort to *pro forma* approach
- Errors in ML code masked by use of alternative method

# Using *pro forma* theorems

- The “main theorem” and its supporting lemmas are results about formulas of a particular form
- HOL users work with arithmetic formulas that are existentially or universally quantified predicates over $\mathbb{Z}$, with type $\mathbb{Z} \to \mathbb{B}$
- Can’t prove results by induction over $\mathbb{Z} \to \mathbb{B}$
- But *can* prove results over lists of constraints, interpreted by special constants
- Using the theorem will involve at least $O(n)$ translation work: into constraint lists with interpreters; and then back out again.

# Example: *pro forma* for exact shadows

```
EVERY fst_nzero uppers ∧ EVERY fst_nzero lowers ⇒
EVERY fst1 uppers ∨ EVERY fst1 lowers ⇒
((∃x. evalupper x uppers ∧ evallower x lowers) ≡
 real_shadow uppers lowers)
```

- `uppers` and `lowers` are lists of pairs of numbers ($x$'s coefficient and its upper/lower bound)
- `fst1`$(c, b) \equiv (c = 1)$
- ```
  evallower x [] = ⊤
  evallower x ((c,lb)::cs) =
    lb <= c * x ∧ evallower x cs
  ```
- ```
  real_shadow uppers lowers =
    ∀c d lb ub.
      MEM (c,ub) uppers ∧ MEM (d,lb) lowers ⇒
      c * lb <= d * ub
  ```

# Using the *pro forma* theorem

$$(\exists x.\ 3x + y \leq 10 \wedge 20 \leq x - y)$$

# Using the *pro forma* theorem

$$(\exists x.\ 3x + y \leq 10 \wedge 20 \leq x - y)$$

*(re-arrange)*

$$\equiv \quad (\exists x.\ 3x \leq 10 - y \wedge 20 + y \leq x)$$

# Using the *pro forma* theorem

$$(\exists x.\ 3x + y \leq 10 \wedge 20 \leq x - y)$$

*(re-arrange)*

$$\equiv \quad (\exists x.\ 3x \leq 10 - y \wedge 20 + y \leq x)$$

*(re-express with `evalupper` & `evallower`)*

$$\equiv \quad (\exists x.\ \texttt{evalupper}\ x\ [(3, 10 - y)] \wedge \texttt{evallower}\ x\ [(1, 20 + y)])$$

# Using the *pro forma* theorem

$$(\exists x.\ 3x + y \leq 10 \wedge 20 \leq x - y)$$

*(re-arrange)*

$$\equiv \quad (\exists x.\ 3x \leq 10 - y \wedge 20 + y \leq x)$$

*(re-express with* `evalupper` *&* `evallower`*)*

$$\equiv \quad (\exists x.\ \texttt{evalupper}\ x\ [(3, 10 - y)] \wedge \texttt{evallower}\ x\ [(1, 20 + y)])$$

*(apply theorem)*

$$\equiv \quad \texttt{real\_shadow}\ [(3, 10 - y)]\ [(1, 20 + y)]$$

# Using the *pro forma* theorem

$$(\exists x.\ 3x + y \leq 10 \wedge 20 \leq x - y)$$

*(re-arrange)*

$$\equiv \quad (\exists x.\ 3x \leq 10 - y \wedge 20 + y \leq x)$$

*(re-express with* `evalupper` *&* `evallower`*)*

$$\equiv \quad (\exists x.\ \texttt{evalupper}\ x\ [(3, 10 - y)] \wedge \texttt{evallower}\ x\ [(1, 20 + y)])$$

*(apply theorem)*

$$\equiv \quad \texttt{real\_shadow}\ [(3, 10 - y)]\ [(1, 20 + y)]$$

*(unfold def'n of* `real_shadow`*)*

$$\equiv \quad 3(20 + y) \leq (10 - y)$$

$$\equiv \quad 4y \leq -50$$

$$\equiv \quad y \leq -13$$

# Pre-processing for efficiency

- The Omega Test's *big* disadvantage is that it requires formula under quantifier to be eliminated to be in DNF
- Consider

$$\forall x.\ x \neq 10 \land x \neq 11 \land 9 < x \leq 12 \Rightarrow x = 12$$

- Negate, remove $\neq$, $<$:

$$\exists x.\ (x \leq 9 \ \lor \ 11 \leq x) \land (x \leq 10 \ \lor \ 12 \leq x) \land$$
$$10 \leq x \land x \leq 12 \land (x \leq 11 \ \lor \ 13 \leq x)$$

- Evaluate 8 ($= 2^3$) clauses.
- Clever preparation of input formulas can make orders of magnitude difference

# Pre-processing for scope

- Procedure for $\mathbb{Z}$ trivially extends to be one for $\mathbb{N}$ (or any mixture of $\mathbb{N}$ and $\mathbb{Z}$) too
- Unfold definitions of constants like MAX and $\exists!$
- Ignore non-Presburger sub-terms by trying to prove more general goals. E.g., $\forall x, y.\ xy > 6 \Rightarrow 2xy > 13$ becomes $\forall z.\ z > 6 \Rightarrow 2z > 13$
- Handle (integer) division by constants:

$$
\begin{aligned}
P(x/d) \equiv & \\
& \exists q\, r.\, (x = qd + r) \wedge (0 \leq r < d \vee d < r \leq 0) \wedge P(q)
\end{aligned}
$$

- (Pre-processing code shared with Cooper's algorithm)

# Comparisons?

Comparisons are odious, but. . .

- Omega Test looks quicker than Cooper's algorithm on small sample

On the other hand

- Omega Test can be destroyed by examples that need work converting to DNF
- I wrote the implementation of Cooper's algorithm before that of the Omega Test; despite some sharing, code is probably better in Omega Test implementation

# Conclusions

- Used well-understood techniques to implement complete methods for $\mathbb{Z}$
- Demonstrated that complete methods need not be infeasible

- Made HOL slightly more usable