Deciding Presburger Arithmetic

Michael Norrish

`Michael.Norrish@nicta.com.au`

National ICT Australia

Outline

Linear Arithmetic D.P.s—Introduction

- If the language is rich enough (has multiplication, has quantifiers), deciding the validity of arbitrary mathmatical formulas (over $\mathbb{Z}$ or $\mathbb{N}$) is impossible.
- With a more impoverished language, a theory may be decidable.
- Historically, this research was part of the attempt to determine the limits of decidability.
- In the present, techniques similar to these are used to solve real-world problems, in a huge variety of systems.

Presburger formulas

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

$\mathit{numeral} * \mathit{term}$ isn't really multiplication; it's short-hand for $\mathit{term} + \mathit{term} + \cdots + \mathit{term}$.

Decision Procedures

- The aim is to produce an algorithm for determining whether or not a Presburger formula is valid with respect to the standard interpretation in arithmetic.
- Such an algorithm is a decision procedure if it is sure to correctly say “true” or “false” for all **closed** formulas.
- Will discuss algorithms for determining truth of formulas of Presburger arithmetic:
  - **Fourier-Motzkin** variable elimination (FMVE), when variables are from $\mathbb{R}$ (or $\mathbb{Q}$)
  - **Omega Test** when variables are from $\mathbb{Z}$ (or $\mathbb{N}$)
  - **Cooper’s algorithm** for $\mathbb{Z}$ (or $\mathbb{N}$)

Quantifier Elimination

- All the methods we'll look at are **quantifier elimination** procedures.
- If a formula with no free variables has no quantifiers, then it is easy to determine its truth value, e.g., $10 > 11 \lor 3 + 4 < 5 \times 3 - 6$.
- Quantifier elimination works by taking input P with n quantifiers and turning it into equvalent formula P' with m quantifiers, and where $m < n$.
- So, eventually

  $$P \equiv P' \equiv ... \equiv Q$$

  and Q has no quantifiers.
- Q will be trivially true or false, and that's the decision

Normalisation

- Methods require input formulas to be normalised (e.g., collect coefficients, use only $<$ and $\leq$)
- Methods eliminate innermost **existential** quantifiers. Universal quantifiers are normalised with
  $(\forall x.\, P(x)) \equiv \neg(\exists x.\, \neg P(x))$
- In FMVE, the sub-formula under the innermost existential quantifier must be a conjunction of relations.
- This means the inner formula must be converted to **disjunctive normal form** (DNF):

$$(c_{11} \wedge c_{12} \wedge \cdots \wedge c_{1n_1}) \vee \cdots \vee (c_{m1} \wedge c_{m2} \wedge \cdots \wedge c_{mn_m})$$

Disjunctive Normal Form

Transform with equivalences

$$
\begin{array}{rcl}
p \wedge (q \vee r) & \equiv & (p \wedge q) \vee (p \wedge r) \\
(p \vee q) \wedge r & \equiv & (p \wedge r) \vee (q \wedge r)
\end{array}
$$

Possibly exponential cost.

Must have also moved negations inwards, achieving **Negation Normal Form**, using

$$
\begin{array}{rcl}
\neg(p \wedge q) & \equiv & \neg p \vee \neg q \\
\neg(p \vee q) & \equiv & \neg p \wedge \neg q \\
\neg\neg p & \equiv & p
\end{array}
$$

Normalisation (cont.)

The formula under $\exists$ is in DNF.
Next, the $\exists$ must be moved inwards

- First over disjuncts, using

$$(\exists x. P \vee Q) \equiv (\exists x.\ P) \vee (\exists x.\ Q)$$

- Must then ensure every conjunct under the quantifier mentions the bound variable.
  Use

$$(\exists x.\ \ P(x)\ \wedge\ Q) \equiv (\exists x.\ \ P(x))\ \wedge\ Q$$

  For example

$$\begin{array}{l}(\exists x.\ \ 3 < x\ \wedge\ x + 2y \leq 6\ \wedge\ y < 0) \longrightarrow \\ \qquad (\exists x.\ \ 3 < x\ \wedge\ x + 2y \leq 6)\ \wedge\ y < 0\end{array}$$

Outline

Fourier-Motzkin theorems

The following simple facts are the basis for a very simple-minded quantifier elimination procedure.
Over $\mathbb{R}$ (or $\mathbb{Q}$), with $a, b > 0$:

$$
\begin{array}{rcl}
(\exists x.\ c \leq ax \wedge bx \leq d) & \equiv & bc \leq ad \\
(\exists x.\ c < ax \wedge bx \leq d) & \equiv & bc < ad \\
(\exists x.\ c \leq ax \wedge bx < d) & \equiv & bc < ad \\
(\exists x.\ c < ax \wedge bx < d) & \equiv & bc < ad
\end{array}
$$

In all four, the right hand side is implied by the left because of transitivity (e.g., $x < y \wedge y \leq z \Rightarrow x < z$).

Fourier-Motzkin theorems (cont.)

In the other direction:

$$bc < ad \Rightarrow (\exists x.\, c < ax \wedge bx \leq d)$$

take x to be $\frac{d}{b}$: $c < a(\frac{d}{b})$, and $b(\frac{d}{b}) \leq d$.

Fourier-Motzkin theorems (cont.)

In the other direction:

$$bc < ad \Rightarrow (\exists x.\ c < ax \wedge bx \leq d)$$

take x to be $\frac{d}{b}$: $c < a(\frac{d}{b})$, and $b(\frac{d}{b}) \leq d$.

For

$$bc < ad \Rightarrow (\exists x.\ c < ax \wedge bx < d)$$

take x to be $\frac{bc+ad}{2ab}$:

$$c < a\left(\frac{bc + ad}{2ab}\right) \equiv 2bc < bc + ad \equiv bc < ad$$

(and similarly for the other bound)

Extending to a full procedure

- So far: a quantifier elimination procedure for formulas where quantifiers only ever have scope over 1 upper bound, and 1 lower bound.
- The method needs to extend to cover cases with multiple constraints.
- No lower bound, many upper bounds:

$$(\exists x.\ b_1 x < d_1 \wedge b_2 x < d_2 \cdots \wedge b_n x < d_n)$$

  Verdict: **True!** (take $\min(\frac{d_i}{b_i}) - 1$ as witness for x)
- No upper bound, many lower bounds: obviously analogous.

Combining many constraints—I

Example:

$$(\exists x.\ c \leq ax \wedge b_1 x \leq d_1 \wedge b_2 x \leq d_2) \equiv b_1 c \leq ad_1 \wedge b_2 c \leq ad_2$$

- From left to right, result just depends on transitivity.
- From right to left, take x to be $\min(\frac{d_1}{b_1}, \frac{d_2}{b_2})$.

In general, with many constraints, combine all possible lower-upper bound pairs.

(Proof that this is possible is by induction on number of constraints.)

Combining many constraints—II

The core elimination formula is

$$\exists x.\ (\bigwedge_h c_h \leq a_h x) \land (\bigwedge_i c_i < a_i x) \land (\bigwedge_j b_j x \leq d_j) \land (\bigwedge_k b_k x < d_k)$$

$$\equiv$$

$$(\bigwedge_{h,j} b_j c_h \leq a_h d_j) \land (\bigwedge_{h,k} b_k c_h < a_h d_k) \land$$
$$(\bigwedge_{i,j} b_j c_i < a_i d_j) \land (\bigwedge_{i,k} b_k c_i < a_i d_k)$$

With n constraints initially, evenly divided between upper and lower bounds, this formula generates $\frac{n^2}{4}$ new constraints.

FMVE example

$$
\begin{array}{rl}
& \forall x.\, 20 + x \leq 0 \;\Rightarrow\; \exists y.\, 3y + x \leq 10 \,\wedge\, 20 \leq y - x \\
& \quad \textit{(re-arrange)} \\
\equiv & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; \exists y.\, 20 + x \leq y \,\wedge\, 3y \leq 10 - x
\end{array}
$$

FMVE example

$$
\begin{array}{ll}
 & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; \exists y.\, 3y + x \leq 10 \,\wedge\, 20 \leq y - x \\
 & \quad \textit{(re-arrange)} \\
\equiv & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; \exists y.\, 20 + x \leq y \,\wedge\, 3y \leq 10 - x \\
 & \quad \textit{(eliminate } y\textit{)} \\
\equiv & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; 60 + 3x \leq 10 - x
\end{array}
$$

FMVE example

$$
\begin{array}{rl}
& \forall x.\, 20 + x \leq 0 \;\Rightarrow\; \exists y.\, 3y + x \leq 10 \;\wedge\; 20 \leq y - x \\
& \quad \textit{(re-arrange)} \\
\equiv & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; \exists y.\, 20 + x \leq y \;\wedge\; 3y \leq 10 - x \\
& \quad \textit{(eliminate } y\textit{)} \\
\equiv & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; 60 + 3x \leq 10 - x \\
& \quad \textit{(re-arrange)} \\
\equiv & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; 4x + 50 \leq 0
\end{array}
$$

FMVE example

$$
\begin{array}{ll}
& \forall x.\, 20 + x \leq 0 \;\Rightarrow\; \exists y.\, 3y + x \leq 10 \wedge 20 \leq y - x \\
& \quad \textit{(re-arrange)} \\
\equiv & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; \exists y.\, 20 + x \leq y \wedge 3y \leq 10 - x \\
& \quad \textit{(eliminate } y\textit{)} \\
\equiv & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; 60 + 3x \leq 10 - x \\
& \quad \textit{(re-arrange)} \\
\equiv & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; 4x + 50 \leq 0 \\
& \quad \textit{(normalise universal)}
\end{array}
$$

FMVE example

$$
\begin{aligned}
& \forall x.\, 20 + x \le 0 \;\Rightarrow\; \exists y.\, 3y + x \le 10 \,\wedge\, 20 \le y - x \\
& \quad \textit{(re-arrange)} \\
\equiv\; & \forall x.\, 20 + x \le 0 \;\Rightarrow\; \exists y.\, 20 + x \le y \,\wedge\, 3y \le 10 - x \\
& \quad \textit{(eliminate } y\textit{)} \\
\equiv\; & \forall x.\, 20 + x \le 0 \;\Rightarrow\; 60 + 3x \le 10 - x \\
& \quad \textit{(re-arrange)} \\
\equiv\; & \forall x.\, 20 + x \le 0 \;\Rightarrow\; 4x + 50 \le 0 \\
& \quad \textit{(normalise universal)} \\
\equiv\; & \neg\exists x.\, 20 + x \le 0 \,\wedge\, 0 < 4x + 50
\end{aligned}
$$

FMVE example

$$\forall x.\, 20 + x \leq 0 \;\Rightarrow\; \exists y.\, 3y + x \leq 10 \,\wedge\, 20 \leq y - x$$

(re-arrange)

$$\equiv\; \forall x.\, 20 + x \leq 0 \;\Rightarrow\; \exists y.\, 20 + x \leq y \,\wedge\, 3y \leq 10 - x$$

(eliminate y)

$$\equiv\; \forall x.\, 20 + x \leq 0 \;\Rightarrow\; 60 + 3x \leq 10 - x$$

(re-arrange)

$$\equiv\; \forall x.\, 20 + x \leq 0 \;\Rightarrow\; 4x + 50 \leq 0$$

(normalise universal)

$$\equiv\; \neg\exists x.\, 20 + x \leq 0 \,\wedge\, 0 < 4x + 50$$

(re-arrange)

FMVE example

$$
\begin{array}{rl}
 & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; \exists y.\, 3y + x \leq 10 \,\wedge\, 20 \leq y - x \\
 & \quad \textit{(re-arrange)} \\
\equiv & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; \exists y.\, 20 + x \leq y \,\wedge\, 3y \leq 10 - x \\
 & \quad \textit{(eliminate } y\textit{)} \\
\equiv & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; 60 + 3x \leq 10 - x \\
 & \quad \textit{(re-arrange)} \\
\equiv & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; 4x + 50 \leq 0 \\
 & \quad \textit{(normalise universal)} \\
\equiv & \neg\exists x.\, 20 + x \leq 0 \,\wedge\, 0 < 4x + 50 \\
 & \quad \textit{(re-arrange)} \\
\equiv & \neg\exists x.\, -50 < 4x \,\wedge\, x \leq -20
\end{array}
$$

FMVE example

$$
\begin{array}{ll}
 & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; \exists y.\, 3y + x \leq 10 \,\wedge\, 20 \leq y - x \\
 & \quad \textit{(re-arrange)} \\
\equiv & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; \exists y.\, 20 + x \leq y \,\wedge\, 3y \leq 10 - x \\
 & \quad \textit{(eliminate } y\textit{)} \\
\equiv & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; 60 + 3x \leq 10 - x \\
 & \quad \textit{(re-arrange)} \\
\equiv & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; 4x + 50 \leq 0 \\
 & \quad \textit{(normalise universal)} \\
\equiv & \neg\exists x.\, 20 + x \leq 0 \,\wedge\, 0 < 4x + 50 \\
 & \quad \textit{(re-arrange)} \\
\equiv & \neg\exists x.\, -50 < 4x \,\wedge\, x \leq -20 \\
 & \quad \textit{(eliminate } x\textit{)}
\end{array}
$$

FMVE example

$$
\begin{array}{ll}
& \forall x.\, 20 + x \leq 0 \;\Rightarrow\; \exists y.\, 3y + x \leq 10 \,\wedge\, 20 \leq y - x \\
& \quad \textit{(re-arrange)} \\
\equiv & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; \exists y.\, 20 + x \leq y \,\wedge\, 3y \leq 10 - x \\
& \quad \textit{(eliminate } y\textit{)} \\
\equiv & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; 60 + 3x \leq 10 - x \\
& \quad \textit{(re-arrange)} \\
\equiv & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; 4x + 50 \leq 0 \\
& \quad \textit{(normalise universal)} \\
\equiv & \neg\exists x.\, 20 + x \leq 0 \,\wedge\, 0 < 4x + 50 \\
& \quad \textit{(re-arrange)} \\
\equiv & \neg\exists x.\, -50 < 4x \,\wedge\, x \leq -20 \\
& \quad \textit{(eliminate } x\textit{)} \\
\equiv & \neg(-50 < -80)
\end{array}
$$

FMVE example

$$
\begin{array}{ll}
& \forall x.\, 20 + x \leq 0 \;\Rightarrow\; \exists y.\, 3y + x \leq 10 \,\wedge\, 20 \leq y - x \\
& \quad \textit{(re-arrange)} \\
\equiv & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; \exists y.\, 20 + x \leq y \,\wedge\, 3y \leq 10 - x \\
& \quad \textit{(eliminate } y\textit{)} \\
\equiv & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; 60 + 3x \leq 10 - x \\
& \quad \textit{(re-arrange)} \\
\equiv & \forall x.\, 20 + x \leq 0 \;\Rightarrow\; 4x + 50 \leq 0 \\
& \quad \textit{(normalise universal)} \\
\equiv & \neg\exists x.\, 20 + x \leq 0 \,\wedge\, 0 < 4x + 50 \\
& \quad \textit{(re-arrange)} \\
\equiv & \neg\exists x.\, -50 < 4x \,\wedge\, x \leq -20 \\
& \quad \textit{(eliminate } x\textit{)} \\
\equiv & \neg(-50 < -80) \qquad \equiv \top
\end{array}
$$

Efficiency

- As before, when eliminating an existential over n constraints we may introduce $\frac{n^2}{4}$ new constraints.
- With k quantifiers to eliminate, we might end with

$$\frac{n^{2^k}}{4^k}$$

  constraints.
- If dealing with alternating quantifiers, repeated conversions to DNF may really hurt.

Expressivity

- Unique existence:
  $$(\exists! x.\ P(x)) \equiv (\exists x.\ P(x) \wedge \forall y.\ P(y) \Rightarrow (y = x))$$
- Conditional expressions:
  - **if** *formula*$_1$ **then** *formula*$_2$ **else** *formula*$_3$ is the same as $(\mathit{formula}_1 \wedge \mathit{formula}_2) \vee (\neg\mathit{formula}_1 \wedge \mathit{formula}_3)$
  - **if-then-else** expressions over *term*, can be moved up and out to be over formulas:
    $$(\textbf{if } x < y \textbf{ then } x \textbf{ else } y) < z$$
    $$\equiv$$
    $$\textbf{if } x < y \textbf{ then } x < z \textbf{ else } y < z$$
- Minimum, maximum, absolute value...

Constraint satisfaction, optimisation

- It's possible to make the algorithm return witnesses to purely existential problems.
- E.g.,

$$\exists x\, y.\, 3x + 4y = 18 \wedge 5x - y \leq 7$$

  might return $\{(x, 2), (y, 3)\}$ (or $\{(x, \frac{2}{3}), (y, 4)\}$, or ...).

- Can also maximise (minimise) z in system $\exists \vec{x}\, z.\ P(\vec{x}, z)$:
  - First check $\exists \vec{x}\, z.\ P(\vec{x}, z)$
  - If it has a solution, check

$$\exists z.\ (\exists \vec{x}.\ P(\vec{x}, z)) \wedge (\forall \vec{x}\, z'.\ P(\vec{x}, z') \Rightarrow z' \leq z)$$

  - If there is a maximum solution for z, this will find it
    Note alternation of quantifiers!

Outline

Expressivity over Integers—I

- Can't do primality

$$prime(x) \equiv \exists y\, z.\; x = yz \wedge 1 < y < x$$

  because of restriction on multiplication
- Can do divisibility by specific numerals:

$$2|e \;\; \equiv \;\; \exists x.\, 2x = e$$

  and so (for example):

$$\forall x.\;\; 0 < x < 30 \;\Rightarrow\; \neg(2|x \wedge 3|x \wedge 5|x)$$

Expressivity over Integers—II

- Can do integer division and modulus, as long as divisor is constant
- Use one of the following results (similar for division)

$$
\begin{aligned}
P(x \bmod d) \;\equiv\; & \\
& \exists q\, r.\ (x = qd + r) \land (0 \le r < d \lor d < r \le 0) \land P(r)
\end{aligned}
$$

$$
\begin{aligned}
P(x \bmod d) \;\equiv\; & \\
& \forall q\, r.\ (x = qd + r) \land (0 \le r < d \lor d < r \le 0) \Rightarrow P(r)
\end{aligned}
$$

Any formula involving modulus or integer division by a constant can be translated to one without.

When d is known, one of the disjuncts will immediately simplify away to false.

Expressivity over Integers—III

- Any procedure for $\mathbb{Z}$ trivially extends to be one for $\mathbb{N}$ (or any mixture of $\mathbb{N}$ and $\mathbb{Z}$) too: add extra constraints stating that variables are ≥ 0
- Ignore non-Presburger sub-terms by trying to prove more general goals.

  For example,

  $$\forall x\, y.\; xy > 6 \Rightarrow 2xy > 13$$

  becomes

  $$\forall z.\; z > 6 \Rightarrow 2z > 13$$

One Nice Thing About the Integers

The relations $<$ and $\leq$ are inter-convertible:

$$\begin{aligned} x \leq y &\equiv x < y+1 \\ x < y &\equiv x+1 \leq y \end{aligned}$$

Decision procedures can normalise one relation into the other.

Fourier-Motzkin for Integers?

- Central theorem is false:

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

- But one direction still works (thanks to transitivity):

$$(\exists x.\ c \leq ax \wedge bx \leq d) \;\Rightarrow\; bc \leq ad$$

- We can compute consequences of existentially quantified formulas

Fourier-Motzkin for Integers?

Have

$$(\exists x.\ c \leq ax \wedge bx \leq d) \;\Rightarrow\; bc \leq ad$$

Thus an incomplete 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
   $(\forall x.\ P(x)) \equiv \top$

   (Repeat for all quantified variables.)

This is Phase 1 of the Omega Test (when there are no alternating quantifiers)

Omega Phase 1—Example

$$\forall x\, y : \mathbb{Z}.\ 0 < x \land y < x \ \Rightarrow\ y + 1 < 2x$$

Omega Phase 1—Example

$$
\begin{array}{rl}
 & \forall x\, y : \mathbb{Z}.\ 0 < x \land y < x \ \Rightarrow\ y + 1 < 2x \\
 & \quad \textit{(normalise)} \\
\equiv & \neg\exists x\, y.\ 1 \leq x\ \land\ y + 1 \leq x\ \land\ 2x \leq y + 1
\end{array}
$$

Omega Phase 1—Example

$$\forall x\, y : \mathbb{Z}.\ 0 < x \land y < x \ \Rightarrow\ y + 1 < 2x$$

(normalise)

$$\equiv \quad \neg\exists x\, y.\ 1 \leq x \ \land\ y + 1 \leq x \ \land\ 2x \leq y + 1$$

$$\exists x\, y.\ 1 \leq x \ \land\ y + 1 \leq x \ \land\ 2x \leq y + 1$$

(eliminate y)

$$\Rightarrow \quad \exists x.\ 1 \leq x \ \land\ 2x \leq x$$

Omega Phase 1—Example

$$
\begin{array}{ll}
 & \forall x\, y : \mathbb{Z}.\ 0 < x \wedge y < x \Rightarrow y + 1 < 2x \\
 & \quad \textit{(normalise)} \\
\equiv & \neg\exists x\, y.\ 1 \le x \wedge y + 1 \le x \wedge 2x \le y + 1
\end{array}
$$

$$
\begin{array}{ll}
 & \exists x\, y.\ 1 \le x \wedge y + 1 \le x \wedge 2x \le y + 1 \\
 & \quad \textit{(eliminate } y\textit{)} \\
\Rightarrow & \exists x.\ 1 \le x \wedge 2x \le x \\
 & \quad \textit{(normalise)} \\
\Rightarrow & \exists x.\ 1 \le x \wedge x \le 0
\end{array}
$$

Omega Phase 1—Example

$$
\begin{array}{ll}
& \forall x\, y : \mathbb{Z}.\ 0 < x \wedge y < x \ \Rightarrow\ y + 1 < 2x \\
& \quad \textit{(normalise)} \\
\equiv & \neg \exists x\, y.\ 1 \leq x \ \wedge\ y + 1 \leq x \ \wedge\ 2x \leq y + 1
\end{array}
$$

$$
\begin{array}{ll}
& \exists x\, y.\ 1 \leq x \ \wedge\ y + 1 \leq x \ \wedge\ 2x \leq y + 1 \\
& \quad \textit{(eliminate } y\textit{)} \\
\Rightarrow & \exists x.\ 1 \leq x \ \wedge\ 2x \leq x \\
& \quad \textit{(normalise)} \\
\Rightarrow & \exists x.\ 1 \leq x \ \wedge\ x \leq 0 \\
& \quad \textit{(eliminate } x\textit{)} \\
\Rightarrow & 1 \leq 0 \quad (\equiv \bot)
\end{array}
$$

Omega Phase 1 and the Interactive Theorem-Provers

The Omega Test's Phase 1 is used by systems like Coq, HOL4, HOL Light and Isabelle to decide arithmetic problems.

Against:

- it's incomplete
- it's inefficient
  - conversion to DNF
  - quadratic increase in numbers of constraints

For:

- it's easy to implement
- it's easy to adapt the procedures to create proofs that can be checked by other tools

Outline

Some Shadows

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

- The formula

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

  is known as the **real shadow**.
- If all of the a_i or all of the b_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 $a = 1$ or $b = 1$, the core theorem

$$(\exists x : \mathbb{Z}.\ c \leq ax \wedge bx \leq d) \equiv bc \leq ad$$

  is valid because
  - $\Rightarrow$: transitivity still holds
  - $\Leftarrow$: take $x = d$ if $b = 1$; $x = c$ if $a = 1$
- Omega Test's inventor, Bill Pugh claims many problems in his domain (compiler optimisations) have exact shadows.
- Experience suggests the same is true in other domains too, such as interactive theorem-proving.
- When shadows are exact, can pretend problem is over $\mathbb{R}$ rather than $\mathbb{Z}$ and life is easy.

Dark Shadows

- The formula

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

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

$$(a - 1)(b - 1) \leq ad - bc \Rightarrow (\exists x.\, c \leq ax \wedge bx \leq d)$$

  (proof to come)
- This is the Phase 2 of the Omega Test

Omega Test phases 1 & 2

Problem is $\exists\vec{x}.\ P(\vec{x})$

- If input is **exact** for one of $\vec{x}$, then eliminate this variable

$$(\exists\vec{x}.\ P(\vec{x})) \equiv (\exists\vec{x}'.\ P'(\vec{x}'))$$

- Otherwise, calculate real shadow R:

$$(\exists\vec{x}.\ P(\vec{x})) \Rightarrow R$$

  so, if $R = \bot$, then input formula is not valid.
- Otherwise, calculate dark shadow D:

$$D \Rightarrow (\exists\vec{x}.\ P(\vec{x}))$$

  so, if $D = \top$, then input formula is valid.

Omega Phase 2—Example

$$\boxed{(a-1)(b-1) \leq ad - bc \Rightarrow (\exists x.\, c \leq ax \wedge bx \leq d)}$$

$$\exists x\, y.\ 3x + 2y \leq 18 \ \wedge\ 3y \leq 4x \ \wedge\ 3x \leq 2y + 1$$

$3y \leq 4x \quad 3x \leq 2y + 1$

$3y \leq 4x \quad 3x \leq 18 - 2y$

Omega Phase 2—Example

$$\boxed{(a-1)(b-1) \leq ad - bc \Rightarrow (\exists x.\, c \leq ax \wedge bx \leq d)}$$

$$\exists x\, y.\; 3x + 2y \leq 18 \;\wedge\; 3y \leq 4x \;\wedge\; 3x \leq 2y + 1$$

$3y \leq 4x \quad 3x \leq 2y + 1$

$6 \leq 8y + 4 - 9y$

$3y \leq 4x \quad 3x \leq 18 - 2y$

Omega Phase 2—Example

$$\boxed{(a-1)(b-1) \leq ad - bc \Rightarrow (\exists x.\, c \leq ax \wedge bx \leq d)}$$

$$\exists x\, y.\ 3x + 2y \leq 18 \ \wedge\ 3y \leq 4x \ \wedge\ 3x \leq 2y + 1$$

$$3y \leq 4x \quad 3x \leq 2y + 1$$
$$6 \leq 8y + 4 - 9y$$

$$3y \leq 4x \quad 3x \leq 18 - 2y$$
$$6 \leq 72 - 8y - 9y$$

Omega Phase 2—Example

$$\boxed{(a-1)(b-1) \leq ad - bc \Rightarrow (\exists x.\ c \leq ax \wedge bx \leq d)}$$

$$\exists x\, y.\ 3x + 2y \leq 18 \ \wedge\ 3y \leq 4x \ \wedge\ 3x \leq 2y + 1$$

$$3y \leq 4x \quad 3x \leq 2y + 1$$
$$6 \leq 8y + 4 - 9y$$
$$y \leq -2$$

$$3y \leq 4x \quad 3x \leq 18 - 2y$$
$$6 \leq 72 - 8y - 9y$$

Omega Phase 2—Example

$$\boxed{(a-1)(b-1) \leq ad - bc \Rightarrow (\exists x.\ c \leq ax \wedge bx \leq d)}$$

$$\exists x\, y.\ 3x + 2y \leq 18 \ \wedge\ 3y \leq 4x \ \wedge\ 3x \leq 2y + 1$$

$$3y \leq 4x \quad 3x \leq 2y + 1$$
$$6 \leq 8y + 4 - 9y$$
$$y \leq -2$$

$$3y \leq 4x \quad 3x \leq 18 - 2y$$
$$6 \leq 72 - 8y - 9y$$
$$17y \leq 66$$

Omega Phase 2—Example

$$\boxed{(a-1)(b-1) \leq ad - bc \Rightarrow (\exists x.\ c \leq ax \wedge bx \leq d)}$$

$$\exists x\, y.\ 3x + 2y \leq 18 \ \wedge\ 3y \leq 4x \ \wedge\ 3x \leq 2y + 1$$

$$3y \leq 4x \quad 3x \leq 2y + 1$$
$$6 \leq 8y + 4 - 9y$$
$$y \leq -2$$

$$3y \leq 4x \quad 3x \leq 18 - 2y$$
$$6 \leq 72 - 8y - 9y$$
$$17y \leq 66$$
$$y \leq 3$$

Omega Phase 2—Example

$$\boxed{(a-1)(b-1) \leq ad - bc \Rightarrow (\exists x.\, c \leq ax \wedge bx \leq d)}$$

$$\exists x\, y.\ 3x + 2y \leq 18 \ \wedge\ 3y \leq 4x \ \wedge\ 3x \leq 2y + 1$$

$$3y \leq 4x \quad 3x \leq 2y + 1$$
$$6 \leq 8y + 4 - 9y$$
$$y \leq -2$$

$$3y \leq 4x \quad 3x \leq 18 - 2y$$
$$6 \leq 72 - 8y - 9y$$
$$17y \leq 66$$
$$y \leq 3$$

redundant

Omega Phase 2—Example

$$\boxed{(a-1)(b-1) \leq ad - bc \Rightarrow (\exists x.\ c \leq ax \wedge bx \leq d)}$$

$$\exists x\, y.\ 3x + 2y \leq 18 \ \wedge\ 3y \leq 4x \ \wedge\ 3x \leq 2y + 1$$

$$\begin{array}{c} 3y \leq 4x \quad 3x \leq 2y+1 \\ 6 \leq 8y + 4 - 9y \\ y \leq -2 \end{array}$$

$$\begin{array}{c} 3y \leq 4x \quad 3x \leq 18 - 2y \\ 6 \leq 72 - 8y - 9y \\ 17y \leq 66 \\ y \leq 3 \\ \text{redundant} \end{array}$$

This gives a suitable value for y, and by back-substitution, finds $x = -1, y = -2$ as a possible solution.

Correctness of Phase 2

Want to show that

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

(extends to multiple constraints by induction)

Proof by contradiction. Assume

$$\begin{array}{l} (a-1)(b-1) \leq ad - bc \\ \forall x.\ ax < c \vee d < bx \end{array}$$

Multiply inequalities in last constraint to get

$$\forall x.\ abx < bc \vee ad < abx$$

$\equiv$ "there are no multiples of ab between bc and ad"

Correctness of Phase 2

Have

$$(a-1)(b-1) \leq ad - bc$$

$$\forall x.\ abx < bc \vee ad < abx$$

As a and b positive, $bc \leq ad$.

Let j be the greatest number such that $abj < bc$.

Then, $ad < ab(j+1)$, and

$$abj < bc \leq ad < ab(j+1)$$

j is the point where the multiples of ab "step over" the $bc \ldots ad$ interval.

Correctness of Phase 2

Have

$$(a-1)(b-1) \leq ad - bc$$
$$\forall x.\ abx < bc \vee ad < abx$$
$$abj < bc \leq ad < ab(j+1)$$

The “gap” between abj and bc must be at least b.

Similarly, the gap between ad and $ab(j+1)$ must be at least a.

I.e., also have

$$b \leq bc - abj$$
$$a \leq ab(j+1) - ad$$

Correctness of Phase 2

Have

$$
\begin{aligned}
(a-1)(b-1) &\leq ad - bc \\
b &\leq bc - abj \\
a &\leq ab(j+1) - ad
\end{aligned}
$$

Add last two constraints:

$$a + b \leq bc + ab - ad$$

Correctness of Phase 2

Have

$$
\begin{aligned}
&(a-1)(b-1) \leq ad - bc \\
&b \leq bc - abj \\
&a \leq ab(j+1) - ad
\end{aligned}
$$

Add last two constraints:

$$
\begin{aligned}
& a+b \leq bc+ab-ad \\
\equiv \quad & ad-bc \leq ab-a-b
\end{aligned}
$$

Correctness of Phase 2

Have

$$
\begin{aligned}
&(a-1)(b-1) \leq ad - bc \\
&b \leq bc - abj \\
&a \leq ab(j+1) - ad
\end{aligned}
$$

Add last two constraints:

$$
\begin{aligned}
& a+b \leq bc + ab - ad \\
\equiv \quad & ad - bc \leq ab - a - b \\
\equiv \quad & ad - bc < ab - a - b + 1
\end{aligned}
$$

Correctness of Phase 2

Have

$$(a-1)(b-1) \leq ad - bc$$
$$b \leq bc - abj$$
$$a \leq ab(j+1) - ad$$

Add last two constraints:

$$\begin{aligned} & a+b \leq bc+ab-ad \\ \equiv\quad & ad-bc \leq ab-a-b \\ \equiv\quad & ad-bc < ab-a-b+1 \\ \equiv\quad & ad-bc < (a-1)(b-1) \end{aligned}$$

Contradiction!

Splinters

- 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_js. Then

$$
\begin{array}{c}
(\exists x. (\bigwedge_i c_i \leq a_i x) \wedge (\bigwedge_j b_j x \leq d_j)) \;\equiv \\[1em]
(\bigwedge_{i,j} (a_i - 1)(b_j - 1) \leq a_i d_j - b_j c_i) \\
\vee \\
\bigvee_i \bigvee_{k=0}^{\left\lfloor \frac{m c_i - c_i - m}{m} \right\rfloor} \left(\exists x. \begin{array}{c} (\bigwedge_i c_i \leq a_i x) \wedge (\bigwedge_j b_j x \leq d_j) \wedge \\ (a_i x = c_i + k) \end{array} \right)
\end{array}
$$

- (Proof in notes.)

Splinters

- 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_js. Then

a splinter

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

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

$$\vee$$

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

dark shadow

- (Proof in notes.)

Splinters

- A splinter

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

  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 an input formula.

Eliminating Equalities

In an expression

$$\exists x.\ \cdots \wedge cx = e \wedge \cdots$$

the existential can be eliminated.

First, multiply all leaves involving x so that they have a common coefficient. Formula becomes

$$\exists x.\ \ \cdots c'x \cdots \wedge\ c'x = e'\ \wedge \cdots c'x \cdots$$

This is equivalent to

$$\cdots e' \cdots \wedge\ c' \,|\, e'\ \wedge \cdots e' \cdots$$

Eliminating Equalities

In an expression

$$\exists x.\ \cdots \wedge cx = e \wedge \cdots$$

the existential can be eliminated.

First, multiply all leaves involving x so that they have a common coefficient. Formula becomes

$$\exists x.\ \cdots c'x \cdots \wedge\ c'x = e'\ \wedge \cdots c'x \cdots$$

This is equivalent to

$$\cdots e' \cdots \wedge\ c \,|\, e\ \wedge \cdots e' \cdots$$

(But what to do with divisibility leaves?)

Eliminating Divisibilities

All leaves under an existential must be inequalities.
What to do with a “divides-term”?

$$\exists x.\ \cdots \wedge\ c \,|\, dx + e\ \wedge \cdots$$

Note: $d < c$ (take modulus if not).

Introduce temporary new existential variable:

$$\exists x\ y.\ \cdots \wedge\ cy = dx + e\ \wedge \cdots$$

Re-arrange:

$$\exists x\ y.\ \cdots \wedge\ dx = cy - e\ \wedge \cdots$$

Eliminating Divisibilities

Started with: $\exists x.\ \cdots \wedge\ c \mid dx + e\ \wedge \cdots$ and knowing $d < c$

Now have: $\exists x\ y.\ \cdots \wedge\ dx = cy - e\ \wedge \cdots$

Use equality elimination to derive

$$\exists y.\ \cdots \wedge\ d \mid cy - e\ \wedge \cdots$$

Because $d < c$, this process must terminate with elimination of divisibility term.

Eliminating Divisibilities

Can eliminate “divides-term” from

$$\exists x.\ \cdots \wedge\ c \,|\, dx + e\ \wedge \cdots$$

by converting to an equality and eliminating that.

But what if a divides-term comes to be negated, and we have to eliminate

$$\exists x.\ \cdots \wedge\ \neg(c \,|\, dx + e)\ \wedge \cdots$$

Eliminating Divisibilities

Can eliminate "divides-term" from

$$\exists x.\ \cdots \wedge\ c \,|\, dx + e\ \wedge \cdots$$

by converting to an equality and eliminating that.

But what if a divides-term comes to be negated, and we have to eliminate

$$\exists x.\ \cdots \wedge\ \neg(c \,|\, dx + e)\ \wedge \cdots$$

Answer:

$$\neg(c \,|\, e) \equiv \bigvee_{i \in 1 \ldots c-1} c \,|\, e + i$$

Introduces lots of disjuncts amongst conjoined leaves (conversion to DNF will be ugly).

Implementation—Constraint tracking

Keep all constraints in canonical form:

$$0 \leq c_1 v_1 + c_2 v_2 + \cdots + c_n$$

and store constraints in a data structure (hash table, say) where keys are coefficients of variables.

So,

$$0 \leq 3x - 4y + 6$$

goes into the $(3, {}^{-}4)$ bucket, and so does

$$0 \leq 3x - 4y + 10$$

But one of these can be dropped!

Implementation—Redundant Constraints

In general, if $p \Rightarrow q$, then $p \wedge q \equiv p$.

All our constraints are implicitly conjoined together, so if we see that one implies another, then the implied one can be dropped.

If two constraints have same set of coefficients, then one is redundant

$$x \leq y \ \wedge \ 0 \leq \Sigma_i c_i v_i + x \ \ \Rightarrow \ \ 0 \leq \Sigma_i c_i v_i + y$$

We can drop $0 \leq 3x - 4y + 10$ if we also have $0 \leq 3x - 4y + 6$

Eliminating constraints makes the problem smaller, and the procedure more efficient.

Implementation—Contradictory Constraints

Use buckets to store potentially "opposite" constraints.

Require bucket keys to have first component positive, so there is a $(3, {}^{-}4)$ bucket, but no $({}^{-}3, {}^{-}4)$ bucket.

If a constraint has a negative first coefficient, put it into the "opposite" bucket.

Constraint	**Bucket**
$0 \le 3x - 4y + 6$	$(3, {}^{-}4)$
$0 \le -3x + 4y + 6$	$(3, {}^{-}4)$
$0 \le -2x - 3y - 10$	$(2, 3)$

This allows easy, early detection of contradictions.

Implementation—Contradictory Constraints

If two constraints have "opposite" constraints, then it's possible that there is an early contradiction

$$x + y < 0 \;\Rightarrow\; \neg(0 \leq \Sigma_i c_i v_i + x \,\wedge\, 0 \leq -\Sigma_i c_i v_i + y)$$

Alternatively, if you have

$$0 \leq \Sigma_i c_i v_i + x \qquad 0 \leq -\Sigma_i c_i v_i + y$$

then by addition, you'd better also have

$$0 \leq x + y$$

By storing opposite constraints together, this check is easy to perform.

Implementation—Normalisation

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

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

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

$$\begin{array}{ll}\exists x. & (x \leq 9 \vee 11 \leq x) \wedge (x \leq 10 \vee 12 \leq x) \wedge \\ & 10 \leq x \wedge x \leq 12 \wedge (x \leq 11 \vee 13 \leq x)\end{array}$$

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

Implementation—Normalisation

The propositional tautology $(p \Rightarrow (q \equiv q')) \Rightarrow (p \wedge q \equiv p \wedge q')$ justifies the following procedure:

- If P is an atomic formula, then when processing $P \wedge Q$, assume P is true while processing Q:
  - If a sub-formula Q_0 of Q is such that $P \Rightarrow Q_0$, then replace Q_0 in Q by $\top$.
  - If a sub-formula Q_0 of Q is such that $P \Rightarrow \neg Q_0$, then replace Q_0 in Q by $\bot$.

Similarly, $(\neg p \Rightarrow (q \equiv q')) \Rightarrow (p \vee q \equiv p \vee q')$ for disjunctions.
This optimisation can make a huge difference to usability.

(Unit propagation is a special case of this.)

Contextual Rewriting—example

Over $\wedge$:

$$0 \leq x + y + 4 \;\wedge\; (0 \leq x + y + 6 \;\vee\; 0 \leq 2x + 3y + 6)$$

is equivalent to

$$0 \leq x + y + 4$$

Contextual Rewriting—example

Over $\wedge$:

$$0 \leq x + y + 4 \;\wedge\; (0 \leq x + y + 6 \;\vee\; 0 \leq 2x + 3y + 6)$$

is equivalent to

$$0 \leq x + y + 4$$

And

$$0 \leq x + y + 4 \;\wedge\; 0 \leq -x - y - 6 \;\wedge\; 0 \leq 2x + 3y + 6$$

is equivalent to

$$\perp$$

Contextual Rewriting—example

Over $\vee$:

$$0 \leq x + y + 4 \ \vee \ 0 \leq x + y + 1 \ \vee \ 0 \leq 2x + 3y + 6$$

is equivalent to

$$0 \leq x + y + 4 \ \vee \ 0 \leq 2x + 3y + 6$$

Outline

Cooper's Algorithm

A non-Fourier-Motzkin alternative:

- Cooper's algorithm is a decision procedure for (integer) Presburger arithmetic.
- It is also a quantifier elimination procedure, which also works from the inside out, eliminating existentials.
- Its **big** advantage is that it doesn't need to normalise input formulas to DNF.

Description is of simplest possible implementation: many tweaks are possible.

Cooper's Algorithm: outline

To eliminate the quantifier in $\exists x.\ P(x)$:

1. Normalise so that only operators are $<$, and divisibility ($c|e$), and negations only occur around divisibility leaves.
2. Compute **least common multiple** of all coefficients of x, and multiply all leaves through by appropriate numbers so that every leaf features x multiplied by the same number c.
3. Now apply $(\exists x.\ P(cx)) \equiv (\exists x.\ P(x) \land c|x)$.

Cooper's Algorithm: normalisation

$$\forall x\, y : \mathbb{Z}.\ 0 < y \land x < y \Rightarrow x + 1 < 2y$$

Cooper's Algorithm: normalisation

$$
\begin{array}{rl}
 & \forall x\, y : \mathbb{Z}.\ 0 < y \wedge x < y \Rightarrow x + 1 < 2y \\
 & \quad \textit{(normalise)} \\
\equiv & \neg\exists x\, y.\ 0 < y \wedge x < y \wedge 2y < x + 2
\end{array}
$$

Cooper's Algorithm: normalisation

$$
\begin{array}{ll}
 & \forall x\, y : \mathbb{Z}.\; 0 < y \wedge x < y \Rightarrow x + 1 < 2y \\
 & \quad \textit{(normalise)} \\
\equiv & \neg\exists x\, y.\; 0 < y \wedge x < y \wedge 2y < x + 2 \\
 & \quad \textit{(transform } y \textit{ to } 2y \textit{ everywhere)} \\
\equiv & \neg\exists x\, y.\; 0 < 2y \wedge 2x < 2y \wedge 2y < x + 2
\end{array}
$$

Cooper's Algorithm: normalisation

$$
\begin{array}{rl}
& \forall x\, y : \mathbb{Z}.\, 0 < y \land x < y \Rightarrow x + 1 < 2y \\
& \quad \textit{(normalise)} \\
\equiv & \neg\exists x\, y.\, 0 < y \land x < y \land 2y < x + 2 \\
& \quad \textit{(transform } y \textit{ to } 2y \textit{ everywhere)} \\
\equiv & \neg\exists x\, y.\, 0 < 2y \land 2x < 2y \land 2y < x + 2 \\
& \quad \textit{(give } y \textit{ unit coefficient)} \\
\equiv & \neg\exists x\, y.\, 0 < y \land 2x < y \land y < x + 2 \land 2 \mid y
\end{array}
$$

Cooper's Algorithm: two cases

How might $\exists x.\ P(x)$ be true?

Either:

- there is a least x making P true; or
- there is no least x: however small you go, there will be a smaller x that still makes P true

Construct two formulas corresponding to both cases.

Cooper's Algorithm: infinitely many small solutions

The case when the values of x satisfying P "go all the way down".

Look at the leaf formulas in P, and think about their values when x has been made arbitrarily small:

- $x < e$: if x goes as small as we like, this will be **true**
- $e < x$: if x goes small, this will be **false**
- $c | x + e$: **unchanged**

This constructs $P_{-\infty}$, a formula where x only occurs in divisibility leaves.

Say δ is the **l.c.m.** of the constants involved in divisibility leaves. Need just test $P_{-\infty}$ on $1 \ldots \delta$.

Cooper's Algorithm: $P_{-\infty}$ example

For

$$\exists y.\, 0 < y \wedge 2x < y \wedge y < x + 2 \wedge 2 \mid y$$

- $0 < y$ will become false as y gets small
- $2x < y$ also becomes false as y gets small
- $y < x + 2$ will be true as y gets small
- $2 \mid y$ doesn't change (it tests if y is even or not)

So in this case, $P_{-\infty}(y) \equiv (\bot \wedge \bot \wedge \top \wedge 2 \mid y) \equiv \bot$.

Cooper's Algorithm: least solution

The case when there is a least x satisfying P.

For there to be a least x satisfying P, it must be the case that one of the leaves $e < x$ is true, and that if x was any smaller the formula would become false.

Let $B = \{e \,:\, e < x \text{ is a leaf of } P\}$

Need just consider $P(b+j)$, where $b \in B$ and $j \in 1 \ldots \delta$.

Final elimination formula is:

$$(\exists x.\, P(x)) \equiv \bigvee_{j=1..\delta} P_{-\infty}(j) \vee \bigvee_{j=1..\delta} \bigvee_{b \in B} P(b+j)$$

Cooper's Algorithm: example continued

For

$$\exists y.\, 0 < y \wedge 2x < y \wedge y < x + 2 \wedge 2 \mid y$$

least solutions, if they exist, will be at $y = 1$, $y = 2$, $y = 2x + 1$, or $y = 2x + 2$.

The divisibility constraint eliminates two of these.

Original formula is equivalent to:

$$(2x < 2 \wedge 0 < x) \;\vee\; (0 < 2x + 2 \wedge x < 0)$$

(Which is unsatisfiable for x.)

Conclusions

- This just scratches the surface of a very big area.
- Fourier-Motzkin methods are very simple techniques for solving problems in $\mathbb{R}$, $\mathbb{Q}$, $\mathbb{Z}$, and $\mathbb{N}$.
- The correctness of the Omega Test and of Cooper's algorithm are alternative proofs of Presburger's 1929 result that Presburger arithmetic is decidable.
- Many other methods exist (particularly for purely existential problems, which is the field of **linear programming**).
- Though most interesting maths remains undecidable, these methods are extremely useful in practical situations.