More Quantifier Elimination
Satisfiability Modulo Theories

Viktor Kunčak

Exposing the Variable to Eliminate: Example

$$\exists y.\ \ 0 < -w + \underline{\mathbf{3y}} + 1 \ \ \wedge \ \ 0 < -\underline{\mathbf{2y}} + z + 6 \ \wedge \ 4 \mid \underline{\mathbf{5y}} + 1$$

Least common multiple of coefficients next to y, $M = lcm(3,2,5) = 30$
Make all occurrences of y in the body have this coefficient:

$$\exists y.\ \ 0 < -10w + \underline{\mathbf{30y}} + 10 \wedge 0 < -\underline{\mathbf{30y}} + 15z + 90 \wedge 24 \mid \underline{\mathbf{30y}} + 6$$

Now we are quantifying over y and using $30y$ everywhere.
Let x denote $30y$.
It is **not an arbitrary** x. It is divisible by 30.

$$\exists x.\ \ 0 < -10w + x + 10 \wedge \ 0 < -x + 15z + 90 \ \wedge \ 24 \mid x + 6 \ \wedge \ 30 \mid x$$

Lower and upper bounds:

Consider the coefficient next to x in $0 < t$. If it is -1, move the term to left side. If it is 1, move the remaining terms to the left side. We obtain formula $F_1(x)$ of the form

$$\bigwedge_{i=1}^{L} a_i < x \wedge \bigwedge_{j=1}^{U} x < b_j \wedge \bigwedge_{i=1}^{D} K_i \mid (x + t_i)$$

Lower and upper bounds:

Consider the coefficient next to x in $0 < t$. If it is -1, move the term to left side. If it is 1, move the remaining terms to the left side. We obtain formula $F_1(x)$ of the form

$$\bigwedge_{i=1}^{L} a_i < x \wedge \bigwedge_{j=1}^{U} x < b_j \wedge \bigwedge_{i=1}^{D} K_i \mid (x + t_i)$$

If there are no divisibility constraints ($D = 0$), what is the formula equivalent to?

Lower and upper bounds:

Consider the coefficient next to x in $0 < t$. If it is -1, move the term to left side. If it is 1, move the remaining terms to the left side. We obtain formula $F_1(x)$ of the form

$$\bigwedge_{i=1}^{L} a_i < x \wedge \bigwedge_{j=1}^{U} x < b_j \wedge \bigwedge_{i=1}^{D} K_i \mid (x + t_i)$$

If there are no divisibility constraints ($D = 0$), what is the formula equivalent to?

$$\max_i a_i + 1 \leq \min_j b_j - 1 \text{ which is equivalent to } \bigwedge_{ij} a_i + 1 < b_j$$

Replacing variable by test terms

There is a an alternative way to express the above condition by replacing $F_1(x)$ with $\bigvee_k F_1(t_k)$ where t_k do not contain x. This is a common technique in quantifier elimination. Note that if $F_1(t_k)$ holds then certainly $\exists x.F_1(x)$.
What are example terms t_i when $D = 0$ and $L > 0$? Hint: ensure that at least one of them evaluates to $\max a_i + 1$.

$$\bigvee_{k=1}^{L} F_1(a_k + 1)$$

What if $D > 0$ i.e. we have additional divisibility constraints?

$$\bigvee_{k=1}^{L} \bigvee_{i=1}^{N} F_1(a_k + i)$$

What is N?

Replacing variable by test terms

There is a an alternative way to express the above condition by replacing $F_1(x)$ with $\bigvee_k F_1(t_k)$ where t_k do not contain x. This is a common technique in quantifier elimination. Note that if $F_1(t_k)$ holds then certainly $\exists x.F_1(x)$.
What are example terms t_i when $D = 0$ and $L > 0$? Hint: ensure that at least one of them evaluates to $\max a_i + 1$.

$$\bigvee_{k=1}^{L} F_1(a_k + 1)$$

What if $D > 0$ i.e. we have additional divisibility constraints?

$$\bigvee_{k=1}^{L} \bigvee_{i=1}^{N} F_1(a_k + i)$$

What is N? least common multiple of $K_1, \ldots, K_D$

Replacing variable by test terms

There is a an alternative way to express the above condition by replacing $F_1(x)$ with $\bigvee_k F_1(t_k)$ where t_k do not contain x. This is a common technique in quantifier elimination. Note that if $F_1(t_k)$ holds then certainly $\exists x.F_1(x)$.
What are example terms t_i when $D=0$ and $L>0$? Hint: ensure that at least one of them evaluates to $\max a_i + 1$.

$$\bigvee_{k=1}^{L} F_1(a_k+1)$$

What if $D>0$ i.e. we have additional divisibility constraints?

$$\bigvee_{k=1}^{L} \bigvee_{i=1}^{N} F_1(a_k+i)$$

What is N? least common multiple of $K_1,\ldots,K_D$
Correctness: note that if $F_1(u)$ holds then also $F_1(u-N)$ holds.

Back to Example

$$\exists x. -10 + 10w < x \wedge x < 90 + 15z \wedge 24 \mid x + 6 \wedge 30 \mid x$$

Back to Example

$$\exists x. -10 + 10w < x \wedge x < 90 + 15z \wedge 24 \mid x + 6 \wedge 30 \mid x$$

$$\bigvee_{i=1}^{120} 10w + i < 100 + 15z \wedge 0 < i \wedge 24 \mid 10w - 4 + i \wedge 30 \mid 10w - 10 + i$$

No lower bounds

Now consider the case $L = 0$. We cannot try all lower bounds, as there are none:

$$\bigwedge_{j=1}^{U} x < b_j \wedge \bigwedge_{i=1}^{D} K_i \mid (x + t_i)$$

No lower bounds

Now consider the case $L = 0$. We cannot try all lower bounds, as there are none:

$$\bigwedge_{j=1}^{U} x < b_j \wedge \bigwedge_{i=1}^{D} K_i \mid (x + t_i)$$

We first drop all constraints except divisibility, obtaining $F_2(x)$:

$$\bigwedge_{i=1}^{D} K_i \mid (x + t_i)$$

and then eliminate quantifier as

$$\bigvee_{i=1}^{N} F_2(i)$$

Exercise: prove equivalence of this result with $\exists x.F_1(x)$ in this case.

An Example with No Lower Bounds

Eliminate quantifier from this formula:

$$\exists x.\ x < 5 + 7z \ \wedge\ 2 \mid x \ \wedge\ 3 \mid x + 2$$

Another Example with No Lower Bounds

Eliminate quantifier from this formula:

$$\exists x.\ x < 5 + 7z \ \wedge\ 2 \mid x \ \wedge\ 3 \mid x+2 \ \wedge\ 6 \mid x+1$$

Wrap Up

This completes the description of a quantifier elimination algorithm for Presburger Arithmetic (PA).

Wrap Up

This completes the description of a quantifier elimination algorithm for Presburger Arithmetic (PA).
This algorithm and its correctness prove that:

- PA admits quantifier elimination
- Satisfiability, validity, entailment, equivalence of PA formulas is decidable
  We can use the algorithm to prove verification conditions.
  Even if not the most efficient way, it gives us insights on which we can later build to come up with better algorithms.
- Quantified and quantifier-free formulas have the same expressive power

Also implies that Presburger arithmetic has quantifier-free interpolants.

Eliminate Quantifiers: Example

$$\exists y.\exists x.\ \ x < -2 \wedge 1 - 5y < x \wedge 1 + y < 13x$$

Check whether the formula is satisfiable

$$x < y + 2 \ \wedge \ y < x + 1 \wedge x = 3k \wedge (y = 6p + 1 \vee y = 6p - 1)$$

Apply quantifier elimination

$$\exists x.\ (3x+1<10 \vee 7x-6<7) \wedge 2 \mid x$$

Approaches to Making QE for PA More Efficient in Practice

Avoid transforming to conjunctions of literals: work directly on negation-normal form. The technique is similar to what we described for conjunctive normal form.

+ no need for DNF
- we may end up trying irrelevant bounds

This is the Cooper's algorithm:

- Reddy, Loveland: Presburger Arithmetic with Bounded Quantifier Alternation. (Gives a slight improvement of the original Cooper's algorithm.)
- Section 7.2 of the Calculus of Computation Textbook

Another Direction for Improvement

Handle a system of equalities more efficiently, without introducing divisibility constraints too eagerly.

Hermite normal form of an integer matrix.

Eliminate variables x and y

$$5x + 7y = a \wedge x \leq y \wedge 0 \leq x$$

Complexity of Deciding Quantified PA Formulas

Regardless how we proceed, we can aobut what the inherent computational complexity of deciding if quantified PA formula is true.

Complexity of Deciding Quantified PA Formulas

Regardless how we proceed, we can aobut what the inherent computational complexity of deciding if quantified PA formula is true.

A set A is in $STA(s(n), t(n), a(n))$ iff there exists a single-tape *alternating* Turing machine M_i (making $\wedge$ as well as $\vee$ choices) that accepts A and runs within space $s(n)$, time $t(n)$ and performs at most $a(n)$ alternations between $\wedge$ and $\vee$ choices.

Deciding of a PA formula of size n with m quantifiers is in

$$STA(*, 2^{n^{O(m)}}, m)$$

Complexity is particularly sensitive to m.

Interpolation For Logical Theories

Interpolation can be useful in generalizing counterexamples to invariants.
Universal **Entailment**: we will write $F_1 \models F_2$ to denote that for all free variables of F_1 and F_2, if F_1 holds then F_2 holds.
Given two formulas such that

$$F_0(\bar{x}, \bar{y}) \models F_1(\bar{y}, \bar{z})$$

an interpolant for F_1, F_2 is a formula $I(\bar{y})$, which has only variables common to F_0 and F_1, such that

- $F_0(\bar{x}, \bar{y}) \models I(\bar{y})$, and
- $I(\bar{y}) \models F_1(\bar{y}, \bar{z})$

In other words, the entailment between F_0 and F_1 can be explained through $I(\bar{y})$.
Logic has **interpolation property** if, whenever $F_0 \models F_1$, then there exists an interpolant for F_0, F_1.
We often wish to have *simple* interpolants, for example ones that are quantifier free.

Quantifier Elimination Implies Interpolation

If logic has QE, it also has quantifier-free interpolants.

Consider the formula

$$\forall \bar{x}, \bar{y}, \bar{z}.\ F_0(\bar{x}, \bar{y}) \rightarrow F_1(\bar{y}, \bar{z})$$

pushing $\bar{x}$ into assumption we get

$$\forall \bar{y}, \bar{z}.\ (\exists \bar{x}. F_0(\bar{x}, \bar{y})) \rightarrow F_1(\bar{y}, \bar{z})$$

and pushing $\bar{z}$ into conclusion we get

$$\forall \bar{x}, \bar{y}.\ F_0(\bar{x}, \bar{y}) \rightarrow (\forall \bar{z}. F_1(\bar{y}, \bar{z}))$$

Given two formulas F_0 and F_1, each of the formulas satisfies properties of interpolation:

- $\exists \bar{x}. F_0(\bar{x}, \bar{y})$
- $\forall \bar{z}. F_1(\bar{y}, \bar{z})$

Applying QE to them, we obtain quantifier-free interpolants.

Quantifier Elimination for Linear Rational Arithmetic

Consider first-order formulas with equality and $<$ relation, interpreted over rationals.
This theory is called **dense linear order without endpoints**
For example:

$$\forall \varepsilon . \exists \delta . \big(|x_1 - x_2| < \delta \wedge |y_1 - y_2| < \delta \rightarrow |3x_1 + 4y_1 - 3x_2 - 4y_2| < \varepsilon \big)$$

(i) Show that absolute value can be defined in first-order logic in terms of other linear operations and comparison.

Quantifier Elimination for Linear Rational Arithmetic

Consider first-order formulas with equality and $<$ relation, interpreted over rationals.
This theory is called **dense linear order without endpoints**
For example:

$$\forall \varepsilon . \exists \delta . \big(|x_1 - x_2| < \delta \wedge |y_1 - y_2| < \delta \rightarrow |3x_1 + 4y_1 - 3x_2 - 4y_2| < \varepsilon \big)$$

(i) Show that absolute value can be defined in first-order logic in terms of other linear operations and comparison.
Answer: replace $F(|t|)$ with, for example

$$(t > 0 \wedge F(t)) \vee (\neg(t > 0) \wedge F(-t))$$

Is there a way to remove |...| while increasing formula size only linearly?

Quantifier Elimination for Linear Rational Arithmetic

Consider first-order formulas with equality and $<$ relation, interpreted over rationals.
This theory is called **dense linear order without endpoints**
For example:

$$\forall \varepsilon . \exists \delta . \big(|x_1 - x_2| < \delta \wedge |y_1 - y_2| < \delta \rightarrow |3x_1 + 4y_1 - 3x_2 - 4y_2| < \varepsilon \big)$$

(i) Show that absolute value can be defined in first-order logic in terms of other linear operations and comparison.
Answer: replace $F(|t|)$ with, for example

$$(t > 0 \wedge F(t)) \vee (\neg(t > 0) \wedge F(-t))$$

Is there a way to remove |...| while increasing formula size only linearly?
(ii) Give quantifier elimination algorithm for this theory.

Quantifier Elimination for Linear Rational Arithmetic

Consider first-order formulas with equality and $<$ relation, interpreted over rationals.
This theory is called **dense linear order without endpoints**
For example:

$$\forall \varepsilon . \exists \delta . \big(|x_1 - x_2| < \delta \wedge |y_1 - y_2| < \delta \rightarrow |3x_1 + 4y_1 - 3x_2 - 4y_2| < \varepsilon \big)$$

(i) Show that absolute value can be defined in first-order logic in terms of other linear operations and comparison.
Answer: replace $F(|t|)$ with, for example

$$(t > 0 \wedge F(t)) \vee (\neg(t > 0) \wedge F(-t))$$

Is there a way to remove |...| while increasing formula size only linearly?
(ii) Give quantifier elimination algorithm for this theory.
Solution is simpler than for Presburger arithmetic—no divisibility.

Satisfiability for Quantifier Free (QF) Formulas

Why Checking Satisfiability for QF Formulas Matters

Deciding classes of quantified formulas is often very expensive: beyond singly exponential, often undecidable.

If theory admits quantifier elimination, then assertions need not be written using quantifiers: ask users to write the corresponding quantifier-free formula.

For quantifier-free formulas we often have specialized algorithms that work better than applying generic algorithms for quantified cases.

Satisfiability for QF Formulas = SAT + satisfiability-of-literals

A quantifier free formula G connects atomic formulas using $\wedge, \vee, \neg$.

We can write such formula as the result of taking a propositional formula F and substituting propositional variables with atomic formulas:

$$G \equiv F[p_1 := A_1, \ldots, p_n := A_n]$$

For example, given QF formula G: $x < y + 1 \vee (y < x \wedge x < 0)$, let $F \equiv p \vee (q \wedge r)$. Then G is

$$F[p := (x<y+1), q := (y<x), r := (x<0)]$$

Formula G is satisfiable iff there exists propositional assignment e_F where $[\![F]\!]e_F = 1$ and the values e for variables inside $A_1, \ldots, A_n$ such that $[\![A_i]\!]e = [\![p_i]\!]e_F$.

In the example above, there exist such $e_F = \{(p,1),(q,0),(r,0)\}$ and $e = \{(x,0),(y,0)\}$, so formula is satisfiable.

QF Formulas: Second Example

Formula

$$x < y \wedge (x = y + 1 \vee x = y + 2)$$

is

QF Formulas: Second Example

Formula

$$x < y \wedge (x = y + 1 \vee x = y + 2)$$

is unsatisfiable because. Indeed, for $F \equiv a \wedge (b \vee c)$, satisfying assignments are:

$$(a,1),(b,1),(c,1)$$
$$(a,1),(b,1),(c,0)$$
$$(a,1),(b,0),(c,1)$$

and they correspond to checking satisfiability of these three conjunctions:

$$x < y \wedge x = y + 1 \wedge x = y + 2$$
$$x < y \wedge x = y + 1 \wedge \neg(x = y + 2)$$
$$x < y \wedge \neg(x = y + 1) \wedge x = y + 2$$

Each satisfying assignment for propositional formula requires us to check corresponding conjunction of literals. In this case, each of these conjunctions of literals is unsatisfiable.

We have checked whether any disjunct in disjunctive normal form is satisfiable.

QF Formulas: Third Example

Formula

$$(x < y \vee y + 1 < x) \wedge \neg(y < x)$$

is

QF Formulas: Third Example

Formula

$$(x < y \vee y + 1 < x) \wedge \neg(y < x)$$

is satisfiable.
Let the formula be $F[p := (x < y), q := (y + 1 < x), r := (y < x)]$ where

$$F \equiv (p \vee q) \wedge \neg r$$

A satisfying assignment for F is $\{(p, 1), (q, 0), (r, 0)\}$ and the *conjunction of literals*

$$(x < y) \wedge \neg(y + 1 < x) \wedge \neg(y < x)$$

is satisfiable with, e.g., $x = 0, y = 0$.

Algorithms for Solving QF Formulas in Linear Rational Arithmetic

LRA: Atomic formulas are of the form $t_1 < t_2$ or $t_1 \leq t_2$ where t_1, t_2 are linear functions of variables: $\sum_{i=1}^{n} c_i x_i$ with c_i rational constants.
Observation: negation of such formulas are again such formulas:
$\neg(t_1 < t_2) \leftrightarrow (t_2 \leq t_1) \quad \neg(t_1 \leq t_2) \leftrightarrow (t_2 < t_1)$
Conjunction of such formulas is a *linear programming problem* (without objective)
Algorithm:

1. SAT solver lazily enumerates satisfying assignments for propositional structure
2. use Simplex algorithm (or e.g. ellipsoid method) to solve *feasibility* of the resulting conjunction of literals

Because ellipsoid is polynomial, the complexity of solving quantifier-free rational arithmetic formula is NP (like for SAT itself).
Special case: if we only have linear equalities $t = 0$, instead of solving linear program we can use, e.g., Gaussian elimination in step 2 to solve a system of linear equations.

Algorithms for Solving QF Formulas in Linear **Integer** Arithmetic

1. SAT solver lazily enumerates satisfying assignments for propositional structure
2. use integer linear programming solver to solve *feasibility* of the resulting conjunction of linear integer arithmetic literals

Observation: conjunction of literals are constraints of an integer linear programming problem:

$\neg(t_1 < t_2) \leftrightarrow (t_2 < t_1 + 1) \quad \neg(t_1 \leq t_2) \leftrightarrow (t_2 < t_1)$

Convert divisibility and its negation into linear constraint with fresh variables:

$c \mid t \iff t = ck$ where k is a fresh integer variable

$c \mid t \iff t = ck + c_R \ \wedge \ 0 < c_R \wedge c_R < c$, k is fresh

Special case: if we only have linear equalities $t = 0$, instead of solving integer linear program it suffices to solve *linear Diophantine equations* which can be done using variable elimination based on extended Euclid algorithm for Greatest Common Divisor computation.

Algorithms for Polynomials

Tarski has shown that there exists a quantifier elimination algorithm for conjunction of polynomials over reals and over complex numbers.
Later method: Cylindrical Algebraic Decomposition (CUD), used in computer algebra systems
Z3 has implemented complete procedure for reals.

Over integers, the problem is undecidable 10th Hilbert's problem
Over rationals, the decidability of the problem is still open

Satisfiability Modulo Theory Solvers and DPLL(T)

Satisfiability Modulo Theories

SAT = Satisfiability for Propositional Logic

- formula: $p \wedge (\neg q \vee r) \wedge s$
- Do there exist truth values p, q, r, s that make formula true?

Satisfiability Modulo Theories

SAT = Satisfiability for Propositional Logic

- formula: $p \wedge (\neg q \vee r) \wedge s$
- Do there exist truth values p, q, r, s that make formula true? yes, e.g., $p \mapsto 1, q \mapsto 0, s \mapsto 1$ (r is any)

Satisfiability Modulo Theories

SAT = Satisfiability for Propositional Logic

- formula: $p \wedge (\neg q \vee r) \wedge s$
- Do there exist truth values p, q, r, s that make formula true? yes, e.g., $p \mapsto 1, q \mapsto 0, s \mapsto 1$ (r is any)

SMT = Satisfiability Modulo Theories (e.g. Z3 solver)

- formula: $a = b \wedge (\neg(f(a) = f(b)) \vee b = c) \wedge \neg(f(a) = f(c))$
- Do there exist values of a, b, c that makes formula true?

Satisfiability Modulo Theories

SAT = Satisfiability for Propositional Logic

- formula: $p \wedge (\neg q \vee r) \wedge s$
- Do there exist truth values p, q, r, s that make formula true? yes, e.g., $p \mapsto 1, q \mapsto 0, s \mapsto 1$ (r is any)

SMT = Satisfiability Modulo Theories (e.g. Z3 solver)

- formula: $a = b \wedge (\neg(f(a) = f(b)) \vee b = c) \wedge \neg(f(a) = f(c))$
- Do there exist values of a, b, c that makes formula true? No. We have: $a = b$, $f(a) = f(b)$, $b = c$, $a = c$, $f(a) = f(c)$, $\neg(f(a) = f(c))$ - unsat

Satisfiability Modulo Theories

SAT = Satisfiability for Propositional Logic

- formula: $p \wedge (\neg q \vee r) \wedge s$
- Do there exist truth values p, q, r, s that make formula true? yes, e.g., $p \mapsto 1, q \mapsto 0, s \mapsto 1$ (r is any)

SMT = Satisfiability Modulo Theories (e.g. Z3 solver)

- formula: $a = b \wedge (\neg(f(a) = f(b)) \vee b = c) \wedge \neg(f(a) = f(c))$
- Do there exist values of a, b, c that makes formula true? No. We have: $a = b$, $f(a) = f(b)$, $b = c$, $a = c$, $f(a) = f(c)$, $\neg(f(a) = f(c))$ - unsat
- Another example: $x = y \wedge f(x) < f(y)$

Satisfiability Modulo Theories

SAT = Satisfiability for Propositional Logic

- formula: $p \wedge (\neg q \vee r) \wedge s$
- Do there exist truth values p, q, r, s that make formula true? yes, e.g., $p \mapsto 1, q \mapsto 0, s \mapsto 1$ (r is any)

SMT = Satisfiability Modulo Theories (e.g. Z3 solver)

- formula: $a = b \wedge (\neg(f(a) = f(b)) \vee b = c) \wedge \neg(f(a) = f(c))$
- Do there exist values of a, b, c that makes formula true? No. We have: $a = b$, $f(a) = f(b)$, $b = c$, $a = c$, $f(a) = f(c)$, $\neg(f(a) = f(c))$ - unsat
- Another example: $x = y \wedge f(x) < f(y)$ - unsat

Satisfiability Modulo Theories

SAT = Satisfiability for Propositional Logic

- formula: $p \wedge (\neg q \vee r) \wedge s$
- Do there exist truth values p, q, r, s that make formula true? yes, e.g., $p \mapsto 1, q \mapsto 0, s \mapsto 1$ (r is any)

SMT = Satisfiability Modulo Theories (e.g. Z3 solver)

- formula: $a = b \wedge (\neg(f(a) = f(b)) \vee b = c) \wedge \neg(f(a) = f(c))$
- Do there exist values of a, b, c that makes formula true? No. We have: $a = b$, $f(a) = f(b)$, $b = c$, $a = c$, $f(a) = f(c)$, $\neg(f(a) = f(c))$ - unsat
- Another example: $x = y \wedge f(x) < f(y)$ - unsat
- Another example: $1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1)$

Satisfiability Modulo Theories

SAT = Satisfiability for Propositional Logic

- formula: $p \wedge (\neg q \vee r) \wedge s$
- Do there exist truth values p, q, r, s that make formula true? yes, e.g., $p \mapsto 1, q \mapsto 0, s \mapsto 1$ (r is any)

SMT = Satisfiability Modulo Theories (e.g. Z3 solver)

- formula: $a = b \wedge (\neg(f(a) = f(b)) \vee b = c) \wedge \neg(f(a) = f(c))$
- Do there exist values of a, b, c that makes formula true? No. We have: $a = b$, $f(a) = f(b)$, $b = c$, $a = c$, $f(a) = f(c)$, $\neg(f(a) = f(c))$ - unsat
- Another example: $x = y \wedge f(x) < f(y)$ - unsat
- Another example: $1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1)$: $x \mapsto 2$

Satisfiability Modulo Theories

SAT = Satisfiability for Propositional Logic

- formula: $p \wedge (\neg q \vee r) \wedge s$
- Do there exist truth values p, q, r, s that make formula true? yes, e.g., $p \mapsto 1, q \mapsto 0, s \mapsto 1$ (r is any)

SMT = Satisfiability Modulo Theories (e.g. Z3 solver)

- formula: $a = b \wedge (\neg(f(a) = f(b)) \vee b = c) \wedge \neg(f(a) = f(c))$
- Do there exist values of a, b, c that makes formula true? No. We have: $a = b$, $f(a) = f(b)$, $b = c$, $a = c$, $f(a) = f(c)$, $\neg(f(a) = f(c))$ - unsat
- Another example: $x = y \wedge f(x) < f(y)$ - unsat
- Another example: $1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1)$: $x \mapsto 2$

Large formulas with few no or few quantifiers (unlike pure FOL provers)

- propositional structure explored using SAT solver
- function and relation symbols come from decidable theories (quantifier-free linear arithmetic, algebraic data types)
- atomic formulas solved using decision procedures (theory solvers)
- quantifiers handled mostly by instantiation

Flattening and Extracting Propositional Structure

$$a = b \wedge (f(a) \neq f(b) \vee b = c) \wedge f(a) \neq f(c)$$

for each atomic formula introduce propositional variable:

Flattening and Extracting Propositional Structure

$$a = b \wedge (f(a) \neq f(b) \vee b = c) \wedge f(a) \neq f(c)$$

for each atomic formula introduce propositional variable:

$$\begin{aligned} & p \wedge (\neg q \vee r) \wedge \neg s \\ & p \Leftrightarrow a = b \\ & q \Leftrightarrow f(a) = f(b) \\ & r \Leftrightarrow b = c \\ & s \Leftrightarrow f(a) = f(c) \end{aligned}$$

flatten: give name to each subterm, e.g. f_a denotes $f(a)$:

Flattening and Extracting Propositional Structure

$$a = b \wedge (f(a) \neq f(b) \vee b = c) \wedge f(a) \neq f(c)$$

for each atomic formula introduce propositional variable:

$$p \wedge (\neg q \vee r) \wedge \neg s$$
$$p \Leftrightarrow a = b$$
$$q \Leftrightarrow f(a) = f(b)$$
$$r \Leftrightarrow b = c$$
$$s \Leftrightarrow f(a) = f(c)$$

flatten: give name to each subterm, e.g. f_a denotes $f(a)$:

$\{p \wedge (\neg q \vee r) \wedge \neg s\}$ give to SAT solver, who returns e.g. $p \wedge \neg q \wedge s$

Flattening and Extracting Propositional Structure

$$a = b \wedge (f(a) \neq f(b) \vee b = c) \wedge f(a) \neq f(c)$$

for each atomic formula introduce propositional variable:

$$\begin{aligned} & p \wedge (\neg q \vee r) \wedge \neg s \\ & p \Leftrightarrow a = b \\ & q \Leftrightarrow f(a) = f(b) \\ & r \Leftrightarrow b = c \\ & s \Leftrightarrow f(a) = f(c) \end{aligned}$$

flatten: give name to each subterm, e.g. f_a denotes $f(a)$:

$\{p \wedge (\neg q \vee r) \wedge \neg s\}$ give to SAT solver, who returns e.g. $p \wedge \neg q \wedge s$

$$\left.\begin{aligned} & p \Leftrightarrow a = b \\ & q \Leftrightarrow f_a = f_b \\ & r \Leftrightarrow b = c \\ & s \Leftrightarrow f_a = f_c \end{aligned}\right\}$$

maps prop. assignment to conjunction of literals

$a = b \wedge f_a \neq f_b \wedge f_a \neq f_c$

Flattening and Extracting Propositional Structure

$$a = b \wedge (f(a) \neq f(b) \vee b = c) \wedge f(a) \neq f(c)$$

for each atomic formula introduce propositional variable:

$$\begin{array}{l} p \wedge (\neg q \vee r) \wedge \neg s \\ p \Leftrightarrow a = b \\ q \Leftrightarrow f(a) = f(b) \\ r \Leftrightarrow b = c \\ s \Leftrightarrow f(a) = f(c) \end{array}$$

flatten: give name to each subterm, e.g. f_a denotes $f(a)$:

$p \wedge (\neg q \vee r) \wedge \neg s\}$ give to SAT solver, who returns e.g. $p \wedge \neg q \wedge s$

$\left.\begin{array}{l} p \Leftrightarrow a = b \\ q \Leftrightarrow f_a = f_b \\ r \Leftrightarrow b = c \\ s \Leftrightarrow f_a = f_c \end{array}\right\}$ maps prop. assignment to conjunction of literals

$a = b \wedge f_a \neq f_b \wedge f_a \neq f_c$

$\left.\begin{array}{l} f_a = f(a) \\ f_b = f(b) \\ f_c = f(c) \end{array}\right\}$ each theory uses its conjuncts and definitions

$\ldots \wedge a = b \wedge f_a \neq f_b \wedge f_a \neq f_c$

UNSAT, give to SAT solver : $\neg(p \wedge \neg q \wedge s)$

Formula containing function symbols and arithmetic

- f is **uninterpreted symbol** (as in FOL)
- $+, <, \leq, 1, 3, 5$ are as in linear integer arithmetic; x is of type integer

$$\underbrace{1 \leq x}_{p} \wedge \underbrace{x < 3}_{q} \wedge \big(\underbrace{(f(1) + 1 \leq f(x))}_{r} \wedge \underbrace{f(x) < f(2)}_{s} \big) \vee \underbrace{4 = 2x}_{t}$$

Formula containing function symbols and arithmetic

- f is **uninterpreted symbol** (as in FOL)
- $+, <, \leq, 1, 3, 5$ are as in linear integer arithmetic; x is of type integer

$$\underbrace{1 \leq x}_{p} \wedge \underbrace{x < 3}_{q} \wedge \big(\underbrace{(f(1) + 1 \leq f(x))}_{r} \wedge \underbrace{f(x) < f(2)}_{s} \big) \vee \underbrace{4 = 2x}_{t}$$

$$\begin{aligned}
& p \wedge q \wedge ((r \wedge s) \vee t) \wedge \\
& \quad p \Leftrightarrow 1 \leq x \ \wedge \\
& \quad q \Leftrightarrow x < 3 \ \wedge \\
& \quad r \Leftrightarrow u_1 \leq u_2 \ \wedge \quad u_1 = u_3 + 1 \wedge u_3 = f(u_4) \wedge u_2 = f(x) \wedge u_4 = 1 \\
& \quad s \Leftrightarrow u_2 < u_5 \ \wedge \quad u_5 = f(u_6) \wedge u_6 = 2 \\
& \quad t \Leftrightarrow u_7 = u_8 \ \wedge \quad u_7 = 4 \wedge u_8 = 2x
\end{aligned}$$

Formula containing function symbols and arithmetic

- f is **uninterpreted symbol** (as in FOL)
- $+, <, \leq, 1, 3, 5$ are as in linear integer arithmetic; x is of type integer

$$\underbrace{1 \leq x}_{p} \wedge \underbrace{x < 3}_{q} \wedge \big(\underbrace{f(1) + 1 \leq f(x)}_{r} \wedge \underbrace{f(x) < f(2)}_{s} \big) \vee \underbrace{4 = 2x}_{t}$$

$$\begin{aligned}
& p \wedge q \wedge ((r \wedge s) \vee t) \wedge \\
& \quad p \Leftrightarrow 1 \leq x \ \wedge \\
& \quad q \Leftrightarrow x < 3 \ \wedge \\
& \quad r \Leftrightarrow u_1 \leq u_2 \ \wedge \quad u_1 = u_3 + 1 \wedge u_3 = f(u_4) \wedge u_2 = f(x) \wedge u_4 = 1 \\
& \quad s \Leftrightarrow u_2 < u_5 \ \wedge \quad u_5 = f(u_6) \wedge u_6 = 2 \\
& \quad t \Leftrightarrow u_7 = u_8 \ \wedge \quad u_7 = 4 \wedge u_8 = 2x
\end{aligned}$$

Who handles which part in this example:

propositional formula	SAT solver
pure equalities ($u_7 = u_8$)	both theory solvers
highlighted formulas	solver for theory of uninterpreted functions
remaining ones	solver for theory of integer linear arithmetic

Theory of Uninterpreted Function Symbols

Quantifier-free first-order logic with equality
Assume it is interpreted over an infinite domain
Assume no relation symbols: replace $R(t_1, \ldots, t_n)$ with $f_R(t_1, \ldots, t_n) = T$ for some fresh constant T
SAT solver handles disjunctions: assume conjunction of equalities and disequalities
Key inference rule, for each function symbol f of n arguments:

$$\frac{t_1 = t_1' \quad \ldots \quad t_n = t_n'}{f(t_1, \ldots, t_n) = f(t_1', \ldots, t_n')}$$

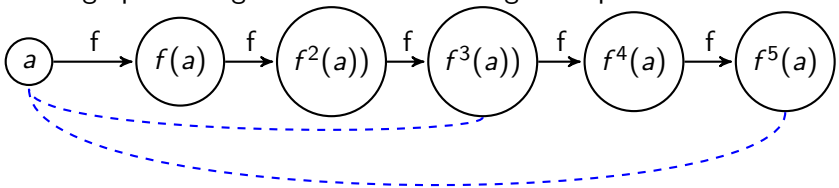
Also: "=" is equivalence relation and $t \neq t$ is contradictory
Apply these rules only to those terms that occur in the formula
Implementation: E-graph stores congruence relation computed so far.
Applying rules: merging nodes in this graph

Example of Running the Algorithm

Let $f^k(a)$ denote $f(\ldots f(a)\ldots)$ with k-fold application of f. Consider

$$f^3(a) = a \wedge f^5(a) = a \wedge f^2(a) \neq a$$

Apply the congruence closure algorithm to check its satisfiability.
Initial graph of all ground terms and the given equalities:



Congruence rule

in this case: $x = y f(x) = f(y)$
Equivalence maintained using union-find algorithm
Conjunction is satisfiable $\Leftrightarrow$ there is literal $t_1 \neq t_2$ where t_1, t_2 are merged
$\Leftarrow$): by properties of equality, conclusions are sound
): computed congruence extends to congruence on the Herbrand model

Quantifier Instantiation During SMT Solving Process

$$G \wedge \forall x.F(x) \quad \rightsquigarrow \quad G \wedge F(t) \wedge \forall x.F(x)$$

where t is a term occurring in G

- this can go on forever
- in general this is incomplete: may need to invent terms that do not occur
- even in the limit it is not complete with respect to the ideal semantics of e.g. integers (theory of quantified integers is not even enumerable)

Controlling the instantiation process using **triggers**

- for each quantified formula $\forall \bar{x}.F(\bar{x})$ require a pattern $P(\bar{x})$ that contains all free variables in $F(\bar{x})$
- instantiate $F(\bar{x})$ only if the the pattern $P(x)$ occurs in the ground formula so far
- introduced in Simplify: a theorem prover for program checking

More information in these papers

- Solving Quantified Verification Conditions using Satisfiability Modulo Theories
- Efficient E-matching for SMT solvers