Semantics and Verification of Loops and Recursion
Quantifier Elimination

Viktor Kunčak

Semantics of a Program with a Loop

Compute and simplify relation for this program:

```
x = 0
while (y > 0) {
  x = x + y
  y = y − 1
}
```

$$\rho(x=0)\circ$$
$$(\Delta_{y \tilde{>} 0} \circ \rho(x=x+y; y=y-1))^{*}\circ$$
$$\Delta_{y \tilde{\leq} 0}$$

$R(x=0)$	$x'=0 \land y'=y$
$R([y>0])$	$y'>0 \land x'=x \land y'=y$
$R([y \leq 0])$	$y' \leq 0 \land x'=x \land y'=y$
$R(\ [y>0];$ $x=x+y;$ $y=y-1)$	$y>0 \land x'=x+y \land y'=y-1$
$R((\ [y>0];$ $x=x+y;$ $y=y-1)^k), k>0$	$y-(k-1)>0 \land$ $x'=x+(y+(y–1)+\cdots+y-(k–1)) \land y'=y-k$ *i.e.* $y \geq k \land x'=x+k(y+y-(k–1))/2 \land y'=y-k$
$R((\ [y>0];$ $x=x+y;$ $y=y-1)^{*})$	$(x'=x \land y'=y) \lor$ $\exists k>0.$ $y \geq k \land x'=x+k(2y-k+1))/2 \land y'=y-k$ *i.e.* $(k=y-y')$ $(x'=x \land y'=y) \lor \big(y-y'>0 \land y' \geq 0 \land x'=x+(y-y')(y+y'+1)/2\big)$ *i.e.*
$R(\text{program})$	$(x'=0 \land y'=y \land y' \leq 0) \lor \big(y>0 \land y'=0 \land x'=y(y+1)/2\big)$

Remarks on Previous Solution

Intermediate components can be more complex than final result

- they must account for all possible initial states, even those never reached in actual executions

Be careful with handling base case. This solution is "almost correct" but incorrectly describes behavior when the initial state has, for example, $y = -2$:

$$y' = 0 \land x' = y(y+1)/2$$

Approximate Semantics of Loops

Instead of computing exact semantics, it can be sufficient to compute approximate semantics. Observation: $r_1 \subseteq r_2 \rightarrow r_1^* \subseteq r_2^*$ (monotonicity still holds).
Suppose we only wish to show that the semantics is included in $s = \{(x, y, x', y') \mid y' \leq y\}$. Note $s \circ s \subseteq s$, $s^* \subseteq s$. Then

```
x = 0
while (y > 0) {
  x = x + y
  y = y - 1
}
```

$$\rho(x = 0) \circ$$
$$(\Delta_{y \tilde{>} 0} \circ \rho(x = x + y; y = y - 1))^* \circ \Delta_{y \tilde{\leq} 0}$$

Approximate Semantics of Loops

Instead of computing exact semantics, it can be sufficient to compute approximate semantics. Observation: $r_1 \subseteq r_2 \to r_1^* \subseteq r_2^*$ (monotonicity still holds).
Suppose we only wish to show that the semantics is included in
$s = \{(x, y, x', y') \mid y' \leq y\}$. Note $s \circ s \subseteq s$, $s^* \subseteq s$. Then

```
x = 0
while (y > 0) {
  x = x + y
  y = y − 1
}
```

$$\rho(x = 0) \circ$$
$$(\Delta_{y \tilde{>} 0} \circ \rho(x = x + y; y = y - 1))^* \circ \Delta_{y \tilde{\leq} 0}$$

$\rotatebox{-90}{$\sqsubseteq$}$ $\rotatebox{-90}{$\subseteq$}$

Approximate Semantics of Loops

Instead of computing exact semantics, it can be sufficient to compute approximate semantics. Observation: $r_1 \subseteq r_2 \rightarrow r_1^* \subseteq r_2^*$ (monotonicity still holds).
Suppose we only wish to show that the semantics is included in
$s = \{(x, y, x', y') \mid y' \leq y\}$. Note $s \circ s \subseteq s$, $s^* \subseteq s$. Then

```
x = 0
while (y > 0) {
  x = x + y
  y = y − 1
}
```

$$\rho(x = 0) \circ$$
$$(\Delta_{y \tilde{>} 0} \circ \rho(x = x + y; y = y - 1))^* \circ \Delta_{y \tilde{\leq} 0}$$

$\sqsupseteq$ $\qquad$ $\supseteq$

```
x = 0
while (y > 0) {
  val y0 = y
  havoc(x,y); assume(y ≤ y0)
}
```

$$s \circ$$
$$(s \circ s)^* \circ s$$

$$\supseteq$$

$$s$$

Recursion

Example of Recursion

For simplicity assume no parameters
(we can simulate them using global variables)

```
def f =
 if (x > 0) {
   if (x % 2 == 0) {
     x = x / 2;
     f;
     y = y * 2
   } else {
     x = x − 1;
     y = y + x;
     f
   }
 }
```

$$
\begin{aligned}
&E(r_f) = \\
&\Delta_{x \tilde{>} 0} \circ \big(\\
&\quad (\Delta_{x\%2=0} \circ \\
&\quad \rho(x = x/2) \circ \\
&\quad r_f \circ \\
&\quad \rho(y = y*2)) \\
&\cup \\
&\quad (\Delta_{x\%2\neq 0} \circ \\
&\quad \rho(x = x-1) \circ \\
&\quad \rho(y = y+x) \circ \\
&\quad r_f) \\
&\big) \cup \Delta_{x \tilde{\leq} 0}
\end{aligned}
$$

Assume recursive function call denotes some relation r_f
Need to find relation r_f such that $r_f = E(r_f)$

Simpler Example of Recursion

def f =
  if (x > 0) {
    x = x − 1
    f
    y = y + 2
  }

$$
\begin{aligned}
E(r_f) = \ & \Big(\Delta_{x \tilde{>} 0} \circ \Big(\\
& \quad \rho(x = x - 1) \circ \\
& \quad r_f \circ \\
& \quad \rho(y = y + 2)\Big) \\
& \Big) \cup \Delta_{x \tilde{\leq} 0}
\end{aligned}
$$

Simpler Example of Recursion

def f =
  if (x > 0) {
    x = x − 1
    f
    y = y + 2
  }

What is $E(\emptyset)$?

$$
\begin{aligned}
E(r_f) = \ & \Big(\Delta_{x \tilde{>} 0} \circ \big(\\
& \quad \rho(x = x-1) \circ \\
& \quad r_f \circ \\
& \quad \rho(y = y+2)) \\
& \big) \cup \Delta_{x \tilde{\leq} 0}
\end{aligned}
$$

Simpler Example of Recursion

def f =
  if (x > 0) {
    x = x − 1
    f
    y = y + 2
  }

$$E(r_f) = \Big(\Delta_{x \tilde{>} 0} \circ \Big(\rho(x = x-1) \circ r_f \circ \rho(y = y+2)\Big) \Big) \cup \Delta_{x \tilde{\leq} 0}$$

What is $E(\emptyset)$?
What is $E(E(\emptyset))$?

Simpler Example of Recursion

```
def f =
  if (x > 0) {
    x = x − 1
    f
    y = y + 2
  }
```

$$E(r_f) = \Big(\Delta_{x\tilde{>}0} \circ \Big(\rho(x = x-1)\circ r_f \circ \rho(y = y+2)\Big)\Big) \cup \Delta_{x\tilde{\leq}0}$$

What is $E(\emptyset)$?
What is $E(E(\emptyset))$?
$E^k(\emptyset)$?

Review from Before: Expressions E on Relations

The law

$$E(\bigcup_{i \in I} r_i) = \bigcup_{i \in I} E(r_i)$$

holds, for each of these cases

1. If $E(r)$ is given by an expression containing r at most once.
2. $\Rightarrow$ If $E(r)$ contains r any number of times, but I is a set of natural numbers and r_i is an increasing sequence: $r_1 \subseteq r_2 \subseteq r_3 \subseteq \ldots$
3. If $E(r)$ contains r any number of times, but $r_i, i \in I$ is a **directed family** of relations: for each i, j there exists k such that $r_i \cup r_j \subseteq r_k$, and I is possibly uncountably infinite.

Sequence of Bounded Recursions

Consider the sequence of relations $r_0 = \emptyset$, $r_k = E^k(\emptyset)$.
What is the relationship between r_k and r_{k+1}?

Sequence of Bounded Recursions

Consider the sequence of relations $r_0 = \emptyset$, $r_k = E^k(\emptyset)$.
What is the relationship between r_k and r_{k+1}?

- $r_0 \subseteq r_1$ because $\emptyset \subseteq \ldots$. Moreover, we showed several lectures earlier that E is monotonic
- from here it follows $r_1 \subseteq r_2$ and, by induction, $r_k \subseteq r_{k+1}$

Sequence of Bounded Recursions

Consider the sequence of relations $r_0 = \emptyset$, $r_k = E^k(\emptyset)$.
What is the relationship between r_k and r_{k+1}?

- $r_0 \subseteq r_1$ because $\emptyset \subseteq \ldots$. Moreover, we showed several lectures earlier that E is monotonic
- from here it follows $r_1 \subseteq r_2$ and, by induction, $r_k \subseteq r_{k+1}$

Define

$$s = \bigcup_{k \geq 0} r_k$$

Then

$$E(s) = E(\bigcup_{k \geq 0} r_k) \stackrel{?}{=} \bigcup_{k \geq 0} E(r_k) = \bigcup_{k \geq 0} r_{k+1} = \bigcup_{k \geq 1} r_k = \emptyset \cup \bigcup_{k \geq 1} r_k = s$$

Sequence of Bounded Recursions

Consider the sequence of relations $r_0 = \emptyset$, $r_k = E^k(\emptyset)$.
What is the relationship between r_k and r_{k+1}?

- $r_0 \subseteq r_1$ because $\emptyset \subseteq \ldots$. Moreover, we showed several lectures earlier that E is monotonic
- from here it follows $r_1 \subseteq r_2$ and, by induction, $r_k \subseteq r_{k+1}$

Define

$$s = \bigcup_{k \geq 0} r_k$$

Then

$$E(s) = E(\bigcup_{k \geq 0} r_k) \stackrel{?}{=} \bigcup_{k \geq 0} E(r_k) = \bigcup_{k \geq 0} r_{k+1} = \bigcup_{k \geq 1} r_k = \emptyset \cup \bigcup_{k \geq 1} r_k = s$$

If $E(s) = s$ we say s is a **fixed point (fixpoint)** of function E

Sequence of Bounded Recursions

Consider the sequence of relations $r_0 = \emptyset$, $r_k = E^k(\emptyset)$.
What is the relationship between r_k and r_{k+1}?

- $r_0 \subseteq r_1$ because $\emptyset \subseteq \ldots$. Moreover, we showed several lectures earlier that E is monotonic
- from here it follows $r_1 \subseteq r_2$ and, by induction, $r_k \subseteq r_{k+1}$

Define

$$s = \bigcup_{k \geq 0} r_k$$

Then

$$E(s) = E(\bigcup_{k \geq 0} r_k) \stackrel{?}{=} \bigcup_{k \geq 0} E(r_k) = \bigcup_{k \geq 0} r_{k+1} = \bigcup_{k \geq 1} r_k = \emptyset \cup \bigcup_{k \geq 1} r_k = s$$

If $E(s) = s$ we say s is a **fixed point (fixpoint)** of function E

We will define meaning of a recursive program as a fixpoint of the corresponding E

Exercise with Fixpoints of Real Functions

1. Find all fixpoints of function $f : \mathbb{R} \to \mathbb{R}$ defined as

$$f(x) = x^2 - x - 3$$

Exercise with Fixpoints of Real Functions

1. Find all fixpoints of function $f : \mathbb{R} \to \mathbb{R}$ defined as

$$f(x) = x^2 - x - 3$$

Solution of $x^2 - x - 3 = x$, that is, $(x-1)^2 = 4$, i.e., $|x-1| = 2$, is $x_1 = -1$ and $x_2 = 3$

Exercise with Fixpoints of Real Functions

1. Find all fixpoints of function $f : \mathbb{R} \to \mathbb{R}$ defined as

$$f(x) = x^2 - x - 3$$

Solution of $x^2 - x - 3 = x$, that is, $(x-1)^2 = 4$, i.e., $|x-1| = 2$, is $x_1 = -1$ and $x_2 = 3$

2. Compute the fixpoint that is smaller than all other fixpoints

Exercise with Fixpoints of Real Functions

1. Find all fixpoints of function $f : \mathbb{R} \to \mathbb{R}$ defined as

$$f(x) = x^2 - x - 3$$

Solution of $x^2 - x - 3 = x$, that is, $(x-1)^2 = 4$, i.e., $|x-1| = 2$, is $x_1 = -1$ and $x_2 = 3$

2. Compute the fixpoint that is smaller than all other fixpoints $x_1 = -1$ is the smallest.

Union of Finite Unfoldings is the **Least** Fixpoint

C - a collection (set) of sets (e.g. sets of pairs, i.e. relations)
$E : C \longrightarrow C$ such that for $r_0 \subseteq r_1 \subseteq r_2 \ldots$
we have

$$E(\bigcup_i r_i) = \bigcup_i E(r_i)$$

(This holds when E is given in terms of $\circ$ and $\cup$.) Then $s = \bigcup_i E^i(\emptyset)$ is such that

1. $E(s) = s$ (we have shown this)
2. if r is arbitrary such that $E(r) \subseteq r$ (special case: if $E(r) = r$), then $s \subseteq r$ (we will show this fact in next slide)

Showing that the Fixpoint is Least

$$s = \bigcup_i E^i(\emptyset)$$

Now take any r such that $E(r) \subseteq r$.
We will show $s \subseteq r$, that is

$$\bigcup_i E^i(\emptyset) \subseteq r \tag{$*$}$$

This means showing $E^i(\emptyset) \subseteq r$, for every i. For $i = 0$ this is just $\emptyset \subseteq r$. We proceed by induction. If $E^i(\emptyset) \subseteq r$, then by monotonicity of E

$$E(E^i(\emptyset)) \subseteq E(r) \subseteq r$$

This completes the proof of $(*)$

Summary: Least Fixpoint as Meaning of Recursion

A recursive program is a recursive definition of a relation $E(r) = r$

We define the intended meaning as $s = \bigcup_{i \geq 0} E(\emptyset)$, which satisfies $E(s) = s$ and also is the least among all relations r such that $E(r) \subseteq r$ (therefore, also the least among r for which $E(r) = r$)

We picked **least** fixpoint, so if the execution cannot terminate on a state x, then there is no x' such that $(x, x') \in s$.
This model is simple (just relations on states) though it has some limitations: let q be a program that never terminates, then

- $\rho(q) = \emptyset$ and $\rho(c \,\square\, q) = \rho(c) \cup \emptyset = \rho(c)$
  (we cannot observe optional non-termination in this model)
- also, $\rho(q) = \rho(\Delta_{\emptyset})$ (assume(false)), so the absence of results due to path conditions and infinite loop are represented in the same way

Summary: Least Fixpoint as Meaning of Recursion

A recursive program is a recursive definition of a relation $E(r) = r$

We define the intended meaning as $s = \bigcup_{i \geq 0} E(\emptyset)$, which satisfies $E(s) = s$ and also is the least among all relations r such that $E(r) \subseteq r$ (therefore, also the least among r for which $E(r) = r$)

We picked **least** fixpoint, so if the execution cannot terminate on a state x, then there is no x' such that $(x, x') \in s$.
This model is simple (just relations on states) though it has some limitations: let q be a program that never terminates, then

- $\rho(q) = \emptyset$ and $\rho(c \,\square\, q) = \rho(c) \cup \emptyset = \rho(c)$
  (we cannot observe optional non-termination in this model)
- also, $\rho(q) = \rho(\Delta_{\emptyset})$ (assume(false)), so the absence of results due to path conditions and infinite loop are represented in the same way

Alternative: error states for non-termination (we will not pursue this approach)

Procedure Meaning is the Least Relation

```
def f =
  if (x > 0) {
    x = x − 1
    f
    y = y + 2
  }
```

$$
\begin{aligned}
E(r_f) = \; & (\Delta_{x \tilde{>} 0} \circ (\\
& \quad \rho(x = x - 1) \circ \\
& \quad r_f \circ \\
& \quad \rho(y = y + 2)) \\
&) \cup \Delta_{x \tilde{\leq} 0}
\end{aligned}
$$

What does it mean that $E(r) \subseteq r$?

Procedure Meaning is the Least Relation

```
def f =
  if (x > 0) {
    x = x − 1
    f
    y = y + 2
  }
```

$$E(r_f) = \Big(\Delta_{x\tilde{>}0} \circ \big(\rho(x = x-1) \circ r_f \circ \rho(y = y+2)\big)\Big) \cup \Delta_{x\tilde{\leq}0}$$

What does it mean that $E(r) \subseteq r$?

Plugging r instead of the recursive call results in something that conforms to r

Justifies modular reasoning for recursive functions

To prove that recursive procedure with body E satisfies specification r, show

- $E(r) \subseteq r$
- Because procedure meaning s is least, conclude $s \subseteq r$

Proving that recursive function meets specification

Prove that if s is the relation denoting the recursive function below, then

$$((x,y),(x',y')) \in s \rightarrow y' \geq y$$

```
def f =
  if (x > 0) {
    x = x − 1
    f
    y = y + 2
  }
```

$$E(r_f) = \Big(\Delta_{x \tilde{>} 0} \circ \Big(\rho(x = x-1) \circ r_f \circ \rho(y = y+2)\Big) \Big) \cup \Delta_{x \tilde{\leq} 0}$$

Proving that recursive function meets specification

Prove that if s is the relation denoting the recursive function below, then

$$((x,y),(x',y')) \in s \rightarrow y' \geq y$$

```
def f =
  if (x > 0) {
    x = x − 1
    f
    y = y + 2
  }
```

$$E(r_f) = \Big(\Delta_{x \tilde{>} 0} \circ \big(\rho(x = x-1) \circ r_f \circ \rho(y = y+2)\big)\Big) \cup \Delta_{x \tilde{\leq} 0}$$

Solution: let specification relation be $q = \{((x,y),(x',y')) \mid y' \geq y\}$

Proving that recursive function meets specification

Prove that if s is the relation denoting the recursive function below, then

$$((x,y),(x',y')) \in s \rightarrow y' \geq y$$

def f =
  if (x > 0) {
    x = x − 1
    f
    y = y + 2
  }

$$E(r_f) = \Big(\Delta_{x \tilde{>} 0} \circ \big(\rho(x = x-1) \circ r_f \circ \rho(y = y+2)\big)\Big) \cup \Delta_{x \tilde{\leq} 0}$$

Solution: let specification relation be $q = \{((x,y),(x',y')) \mid y' \geq y\}$
Prove $E(q) \subseteq q$ - given by a quantifier-free formula

Formula for Checking Specification

```
def f =
  if (x > 0) {
    x = x − 1
    f
    y = y + 2
  }
```

Specification: $q = \{((x,y),(x',y')) \mid y' \geq y\}$

Formula to prove, generated by representing $E(q) \subseteq q$:

$$\big((x > 0 \land x_1 = x - 1 \land y_1 = y \land y_2 \geq y_1 \land y' = y_2 + 2) \lor (\neg(x > 0) \land x' = x \land y' = y)\big) \rightarrow y' \geq y$$

- Because q appears as $E(q)$ and q, the condition appears twice.
- Proving $f \subseteq q$ by $E(q) \subseteq q$ is always sound, whether or not function f terminates; the meaning of f talks only about properties of terminating executions (relations can be partial)

Multiple Procedures: Functions on Pairs of Relations

Two mutually recursive procedures $r_1 = E_1(r_1, r_2), \quad r_2 = E_2(r_1, r_2)$
We extend the approach to work on pairs of relations:

$$(r_1, r_2) = (E_1(r_1, r_2), E_2(r_1, r_2))$$

Define $\bar{E}(r_1, r_2) = (E_1(r_1, r_2), E_2(r_1, r_2))$, let $\bar{r} = (r_1, r_2)$. We define semantics of procedures as the least solution of

$$\bar{E}(\bar{r}) = \bar{r}$$

where $(r_1, r_2) \sqsubseteq (r_1', r_2')$ means $r_1 \subseteq r_1'$ and $r_2 \subseteq r_2'$
Even though pairs of relations are not sets but pairs of sets, we can define set-like operations on them, e.g.

$$(r_1, r_2) \sqcup (r_1', r_2') = (r_1 \cup r_1', \ r_2 \cup r_2')$$

The entire theory works when we have a partial order $\sqsubseteq$ with some "good properties". (**Lattice** elements are a generalization of sets.)

Multiple Procedures: Least Fixedpoint and Consequences

Two mutually recursive procedures $r_1 = E_1(r_1, r_2), \quad r_2 = E_2(r_1, r_2)$
For $E(r_1, r_2) = (E_1(r_1, r_2), E_2(r_1, r_2))$, semantics is

$$(s_1, s_2) = \bigsqcup_{i \geq 0} \bar{E}^i(\emptyset, \emptyset)$$

It follows that for any c_1, c_2 if

$$E_1(c_1, c_2) \subseteq c_1 \quad \text{and} \quad E_2(c_1, c_2) \subseteq c_2$$

then $s_1 \subseteq c_1$ and $s_2 \subseteq c_2$.

Induction-like principle: To prove that mutually recursive relations satisfy two contracts, prove those contracts for the relation body definitions in which recursive calls are replaced by those contracts.

Replacing Calls by Contracts: Example

```
def r1 = {
  if (x % 2 == 1) {
    x = x − 1
  }
  y = y + 2
  r2
} ensuring(y > old(y))
```

```
def r2 = {
  if (x != 0) {
    x = x / 2
    r1
  }
} ensuring(y >= old(y))
```

Replacing Calls by Contracts: Example

```
def r1 = {
  if (x % 2 == 1) {
    x = x − 1
  }
  y = y + 2
  r2
} ensuring(y > old(y))
```

```
def r2 = {
  if (x != 0) {
    x = x / 2
    r1
  }
} ensuring(y >= old(y))
```

Reduces to checking these two non-recursive procedures:

```
def r1 = {
  if (x % 2 == 1) {
    x = x − 1
  }
  y = y + 2
  { val x0 = x; y0 = y
    havoc(x,y)
    assume(y >= y0) }
} ensuring(y > old(y))
```

```
def r2 = {
  if (x != 0) {
    x = x / 2
    val x0 = x; y0 = y
    havoc(x,y)
    assume(y > y0)
  }
} ensuring(y >= old(y))
```

Deductive Verification

Three-step approach:

1. Compile program meaning to logical formulas (verification-condition generator, symbolic execution)
2. Express properties in logic or code (assertions, preconditions, post-conditions, invariants, run-time error conditions)
3. Develop and use an automated theorem prover for generated conditions (SAT solving, SMT solving, resolution-based theorem proving, rewriting, interactive provers)

Which logic to use? Today: integer linear arithmetic

Presburger arithmetic

Integer arithmetic with logical operations (and, or, not), quantifiers, only addition as an arithmetic operation, and $<$ and $=$ as a relation.

- minimalistically one can define a variant of it over non-negative natural numbers as having $\wedge, \neg, \forall, +, =$ as the only symbols

One of the earliest theories shown decidable. Mojesz Presburger gave an algorithm for quantifier elimination in 1929.

- a student of a famous logician Alfred Tarski
- Tarski gave him this question for his MSc thesis

The result at this time was of interest to mathematical logic and foundations of mathematics

- only much later it found applications in automated reasoning (Cooper 1972, Derek C. Oppen - STOC 1973)

Presburger Arithmetic for Verification

```
res = 0
i = x
while // invariant I(res,i): res + 2*i == 2*x && 0 <= i
(i > 0) {
    i = i - 1
    res = res + 2
}
```

Verification condition (VC) for preservation of loop invariant:

$$\left[I(res,i) \land i' = i-1 \land res' = res+2 \land 0 < i\right] \rightarrow I(res',i')$$

To prove that this VC is valid, we check whether its *negation*

$$I(res,i) \land i' = i-1 \land res' = res+2 \land 0 < i \land \neg I(res',i')$$

is *satisfiable*, i.e. whether this PA formula is true:

$$\begin{aligned}\exists x, res, i, res', i'.\ \big[&res+2i = 2x \ \land\ 0 \leq i \ \land\ 0 < i \ \land \\ &i' = i-1 \ \land\ res' = res+2 \ \land\ \neg(res'+2i' = 2x \ \land\ 0 \leq i')\big]\end{aligned}$$

Introducing: One-Point Rule

If $\bar{y}$ is a tuple of variables not containing x, then

$$\exists x.(x = t(\bar{y}) \wedge F(x,\bar{y})) \iff F(t(\bar{y}),\bar{y})$$

Proof:

- $\rightarrow$: Consider the values of $\bar{y}$ such that there exists x, say x_1, for which $x_1 = t(\bar{y}) \wedge F(x_1,\bar{y})$. Because $F(x_1,\bar{y})$ evaluates to true and the values of x_1 and $t(\bar{y})$ are the same, $F(t,\bar{y})$ also evaluates to true.
- $\leftarrow$: Let $\bar{y}$ be such that $F(t,\bar{y})$ holds. Let x be the value of $t(\bar{y})$. Then of course $x = t(\bar{y})$ evaluates to true and so does $F(x,\bar{y})$. So there exists x for which $x = t(\bar{y}) \wedge F(x,\bar{y})$ holds.

One point rule:
replaces left side (LHS) of equivalence by the right side (RHS).
Flattening, used when t is complex, replaces RHS by LHS.

Dual One-Point Rule for $\forall$

$$\forall x.(x = t(\bar{y}) \rightarrow F(x, \bar{y})) \iff F(t(\bar{y}), \bar{y})$$

To prove it, negate both sides:

$$\exists x.(x = t(\bar{y}) \land \neg F(x, \bar{y})) \iff \neg F(t(\bar{y}), \bar{y})$$

so it reduces to the rule for $\exists$.

Using One-Point Rule on Negated Verification Condition

$$\exists x, res, i, res', \underline{\mathbf{i'}}.\ \Big[res + 2i = 2x \ \wedge\ 0 \leq i \ \wedge\ 0 < i \ \wedge \\ \underline{\mathbf{i' = i\ \text{-}\ 1}} \ \wedge\ res' = res + 2 \ \wedge \\ \neg\big(res' + 2i' = 2x \ \wedge\ 0 \leq i'\big)\Big]$$

Using One-Point Rule on Negated Verification Condition

$$\exists x, res, i, res', \underline{\mathbf{i'}}.\ \Big[res + 2i = 2x\ \wedge\ 0 \leq i\ \wedge\ 0 < i\ \wedge \\ \underline{\mathbf{i' = i - 1}}\ \wedge\ res' = res + 2\ \wedge \\ \neg\big(res' + 2i' = 2x\ \wedge\ 0 \leq i'\big)\Big]$$

$$\exists x, res, i, \underline{\mathbf{res'}}.\ \Big[res + 2i = 2x\ \wedge\ 0 \leq i\ \wedge\ 0 < i\ \wedge \\ \underline{\mathbf{res' = res + 2}}\ \wedge \\ \neg\big(res' + 2(i-1) = 2x\ \wedge\ 0 \leq i-1\big)\Big]$$

Using One-Point Rule on Negated Verification Condition

$$\exists x, res, i, res', \underline{\mathbf{i'}}.\ \Big[res + 2i = 2x\ \wedge\ 0 \le i\ \wedge\ 0 < i\ \wedge \\ \underline{\mathbf{i' = i - 1}}\ \wedge\ res' = res + 2\ \wedge \\ \neg\big(res' + 2i' = 2x\ \wedge\ 0 \le i'\big)\Big]$$

$$\exists x, res, i, \underline{\mathbf{res'}}.\ \Big[res + 2i = 2x\ \wedge\ 0 \le i\ \wedge\ 0 < i\ \wedge \\ \underline{\mathbf{res' = res + 2}}\ \wedge \\ \neg\big(res' + 2(i-1) = 2x\ \wedge\ 0 \le i-1\big)\Big]$$

$$\exists x, res, i.\ \Big[\underline{res + 2i = 2x}\ \wedge\ 0 \le i\ \wedge\ 0 < i\ \wedge \\ \neg\big(res + 2 + 2(i-1) = 2x\ \wedge\ 0 \le i-1\big)\Big]$$

Using One-Point Rule on Negated Verification Condition

$$\exists x, res, i, res', \underline{\mathbf{i'}}.\ \Big[res + 2i = 2x \ \wedge\ 0 \le i \ \wedge\ 0 < i \ \wedge \\ \underline{\mathbf{i' = i - 1}} \ \wedge\ res' = res + 2 \ \wedge \\ \neg\big(res' + 2i' = 2x \ \wedge\ 0 \le i'\big)\Big]$$

$$\exists x, res, i, \underline{\mathbf{res'}}.\ \Big[res + 2i = 2x \ \wedge\ 0 \le i \ \wedge\ 0 < i \ \wedge \\ \underline{\mathbf{res' = res + 2}} \ \wedge \\ \neg\big(res' + 2(i-1) = 2x \ \wedge\ 0 \le i-1\big)\Big]$$

$$\exists x, res, i.\ \Big[\underline{res + 2i = 2x} \ \wedge\ 0 \le i \ \wedge\ 0 < i \ \wedge \\ \neg\big(res + 2 + 2(i-1) = 2x \ \wedge\ 0 \le i-1\big)\Big]$$

$$\exists x, \underline{\mathbf{res}}, i.\ \Big[\underline{\mathbf{res = 2x - 2i}} \ \wedge\ 0 \le i \ \wedge\ 0 < i \ \wedge \\ \neg\big(res + 2 + 2(i-1) = 2x \ \wedge\ 0 \le i-1\big)\Big]$$

Using One-Point Rule on Negated Verification Condition

$$\exists x, res, i, res', \underline{\mathbf{i'}}.\ \begin{bmatrix} res+2i=2x \ \wedge\ 0\le i \ \wedge\ 0<i \ \wedge \\ \underline{\mathbf{i' = i - 1}} \ \wedge\ res'=res+2 \ \wedge \\ \neg(res'+2i'=2x \ \wedge\ 0\le i') \end{bmatrix}$$

$$\exists x, res, i, \underline{\mathbf{res'}}.\ \begin{bmatrix} res+2i=2x \ \wedge\ 0\le i \ \wedge\ 0<i \ \wedge \\ \underline{\mathbf{res' = res + 2}} \ \wedge \\ \neg(res'+2(i-1)=2x \ \wedge\ 0\le i-1) \end{bmatrix}$$

$$\exists x, res, i.\ \begin{bmatrix} \underline{res+2i=2x} \ \wedge\ 0\le i \ \wedge\ 0<i \ \wedge \\ \neg(res+2+2(i-1)=2x \ \wedge\ 0\le i-1) \end{bmatrix}$$

$$\exists x, \underline{\mathbf{res}}, i.\ \begin{bmatrix} \underline{\mathbf{res = 2x - 2i}} \ \wedge\ 0\le i \ \wedge\ 0<i \ \wedge \\ \neg(res+2+2(i-1)=2x \ \wedge\ 0\le i-1) \end{bmatrix}$$

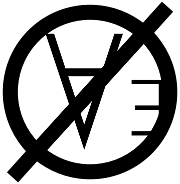
$$\exists x, i.\ \begin{bmatrix} 0\le i \ \wedge\ 0<i \ \wedge \\ \neg(2x-2i+2+2(i-1)=2x \ \wedge\ 0\le i-1) \end{bmatrix}$$

Simplifies to $\exists x, i.\ 0<i \wedge \neg(0\le i-1)$ and then to false.

But there is more

One-point rule is one of the many steps used in **quantifier elimination** procedures.

Quantifier Elimination (QE)



Given a formula $F(\bar{y})$ containing quantifiers find a formula $G(\bar{y})$

- **equivalent** to $F(\bar{y})$
- that has **no quantifiers**
- and has a **subset (or equal set) of free variables** of F

Note

- Equivalence: For all $\bar{y}$, $F(\bar{y})$ and $G(\bar{y})$ have same truth value
  $\rightsquigarrow$ we can use $G(\bar{y})$ instead of $F(\bar{y})$
- No quantifiers: easier to check satisfiability of $G(\bar{y})$

$\bar{y}$ is a possibly empty tuple of variables

We are lucky when a theory has (“admits”) QE

Suppose F has no free variables (all variables are quantified).
What is the result of applying QE to F?

We are lucky when a theory has (“admits”) QE

Suppose F has no free variables (all variables are quantified).
What is the result of applying QE to F?
Are there any variables in the resulting formula?

We are lucky when a theory has ("admits") QE

Suppose F has no free variables (all variables are quantified).
What is the result of applying QE to F?
Are there any variables in the resulting formula?

- No free variables: they are a subset of the original, empty set
- No quantified variables: because it has no quantifiers ☺

Formula without any variables! Example:

$$(2+4=7) \vee (1+1=2)$$

We check the truth value of such formula by simply evaluating it!

Using QE for Deciding Satisfiability/Validity

- To check satisfiability of $H(\bar{y})$: eliminate the quantifiers from $\exists \bar{y}.H(\bar{y})$ and evaluate.
- Validity: eliminate quantifiers from $\forall \bar{y}.H(\bar{y})$ and evaluate

We can even check formulas like this:

$$\forall x, y, r.\ \exists z.\ (5 \leq r \wedge x + r \leq y) \rightarrow (x < z \wedge z < y \wedge 3|z)$$

Here $3|z$ denotes that z is divisible by 3.

Does Presburger Arithmetic admit QE?

Does Presburger Arithmetic admit QE?

Depends on the particular set of symbols!
(Recall objective: given $F(\bar{y})$ containing quantifiers find a formula $G(\bar{y})$

- **equivalent** to $F(\bar{y})$
- that has **no quantifiers**
- and has a **subset (or equal set) of free variables** of F)

If we lack some operations that can be expressed using quantifiers, there may be no equivalent formula without quantifiers.

- $\exists y. x = y + y + y$, so we better have divisibility

Quantifier elimination says: if you can define some relationship between variables using an arbitrary, possibly quantified, formula F,

$$r \stackrel{def}{=} \{(x, y) \mid F(x, y)\}$$

then you can also define same r using another quantifier-free formula G.

Presburger Arithmetic (PA)

We look at the theory of integers with addition.

- introduce constant for each integer constant
- to be able to restrict values to natural numbers when needed, and to compare them, we introduce $<$
- introduce not only addition but also subtraction
- to conveniently express certain expressions, introduce function m_K for each $K \in \mathscr{Z}$, to be interpreted as multiplication by a constant, $m_K(x) = K \cdot x$. We write m_K as $K \cdot x$.
  Note: there is *no multiplication between variables* in PA
- to enable quantifier elimination from $\exists x. y = K \cdot x$ introduce for each K predicate $K|y$ (divisibility, $y\%K = 0$)

The resulting language has these function and relation symbols:
$\{+, -, =, <\} \cup \{K \mid K \in \mathbb{Z}\} \cup \{(K \cdot \_) \mid K \in \mathbb{Z}\} \cup \{(K|\_) \mid K \in \mathbb{Z}\}$ We also have, as usual:
$\wedge, \vee, \neg, \rightarrow$ and also: $\exists, \forall$

Example

Eliminate y from this formula:

$$\exists y.\ \ 3y - 2w + 1 > -w \land 2y - 6 < z \land 4 \mid 5y + 1$$

What should we do first?

Example

Eliminate y from this formula:

$$\exists y.\ \ 3y - 2w + 1 > -w \wedge 2y - 6 < z \wedge 4 \,|\, 5y + 1$$

What should we do first?
Simplify/normalize what we can using properties of integer operations:

$$\exists y.\ \ 0 < -w + 3y + 1 \ \ \wedge\ \ 0 < -2y + z + 6 \ \wedge\ 4 \,|\, 5y + 1$$

Example

Eliminate y from this formula:

$$\exists y.\ \ 3y - 2w + 1 > -w \wedge 2y - 6 < z \wedge 4 \mid 5y + 1$$

What should we do first?
Simplify/normalize what we can using properties of integer operations:

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

First we will consider only eliminating existential from a **conjunction of literals**.

Conjunctions of Literals

Atomic formula: a relation applied to argument.
Here, relations are: $=$, $<$, $K|\_$. So, atomic formulas are:
$t_1 = t_2, \quad t_1 < t_2, \quad K \mid t$

Conjunctions of Literals

Atomic formula: a relation applied to argument.
Here, relations are: $=$, $<$, $K|\_$. So, atomic formulas are:
$t_1 = t_2, \quad t_1 < t_2, \quad K \mid t$
Literal: Atomic formula or its negation. Example: $\neg(x = y + 1)$
Conjunction of literals: $L_1 \wedge \ldots \wedge L_n$

- no disjunctions, no implications
- negation only applies to atomic formulas

We first consider the quantifier elimination problem of the form:

$$\exists y.\ L_1 \wedge \ldots \wedge L_n$$

This will prove to be sufficient to eliminate all quantifiers!

Eliminating ∃ from conjunction of literals suffices

Can we eliminate ∃ from any **quantifier-free formula**?

$$\exists x. F(x, \bar{y})$$

where F is quantifier-free?

Eliminating ∃ from conjunction of literals suffices

Can we eliminate ∃ from any **quantifier-free formula**?

$$\exists x. F(x, \bar{y})$$

where F is quantifier-free?
Formula without quantifiers has $\wedge$,$\vee$,$\neg$ applied to atomic formulas.

Eliminating ∃ from conjunction of literals suffices

Can we eliminate ∃ from any **quantifier-free formula**?

$$\exists x. F(x, \bar{y})$$

where F is quantifier-free?
Formula without quantifiers has ∧,∨,¬ applied to atomic formulas.
Convert F to **disjunctive normal form**:

$$F \iff \bigvee_{i=1}^{m} C_i$$

each C_i is a **conjunction of literals**.

Eliminating ∃ from conjunction of literals suffices

Can we eliminate ∃ from any **quantifier-free formula**?

$$\exists x. F(x, \bar{y})$$

where F is quantifier-free?
Formula without quantifiers has $\wedge, \vee, \neg$ applied to atomic formulas.
Convert F to **disjunctive normal form**:

$$F \iff \bigvee_{i=1}^{m} C_i$$

each C_i is a **conjunction of literals**.

$$\left[\exists x. \bigvee_{i=1}^{m} C_i\right] \iff \bigvee_{i=1}^{m} (\exists x. C_i)$$

How does disjunctive normal form (DNF) transformation work?

Which steps should we use?

How does disjunctive normal form (DNF) transformation work?

Which steps should we use?

Negation propagation:

$$\neg(p \wedge q) \rightsquigarrow (\neg p) \vee (\neg q)$$

$$\neg(p \vee q) \rightsquigarrow (\neg p) \wedge (\neg q)$$

$$\neg\neg p \rightsquigarrow p$$

Result is **negation-normal form**, NNF

NNF transformation is polynomial (exercise!)

How does disjunctive normal form (DNF) transformation work?

Which steps should we use?
Negation propagation:

$$\neg(p \wedge q) \rightsquigarrow (\neg p) \vee (\neg q)$$

$$\neg(p \vee q) \rightsquigarrow (\neg p) \wedge (\neg q)$$

$$\neg\neg p \rightsquigarrow p$$

Result is **negation-normal form**, NNF
NNF transformation is polynomial (exercise!)
Distributivity

$$a \wedge (b_1 \vee b_2) \rightsquigarrow (a \wedge b_1) \vee (a \wedge b_2)$$

This can lead to exponential explosion.
Can we obtain *equivalent* DNF formula without explosion?

How does disjunctive normal form (DNF) transformation work?

Which steps should we use?
Negation propagation:

$$\neg(p \wedge q) \rightsquigarrow (\neg p) \vee (\neg q)$$

$$\neg(p \vee q) \rightsquigarrow (\neg p) \wedge (\neg q)$$

$$\neg\neg p \rightsquigarrow p$$

Result is **negation-normal form**, NNF
NNF transformation is polynomial (exercise!)
Distributivity

$$a \wedge (b_1 \vee b_2) \rightsquigarrow (a \wedge b_1) \vee (a \wedge b_2)$$

This can lead to exponential explosion.
Can we obtain *equivalent* DNF formula without explosion?
No! We can prove this (no equivalent DNF formula exists), unrelated to NP vs P

Eliminating from quantifier free formulas

$$\exists x.F \iff \left[\exists x.\bigvee_{i=1}^{m} C_i\right] \iff \bigvee_{i=1}^{m}(\exists x.C_i)$$

Nested Existential Quantifiers

$$\exists x_1 . \exists x_2 . \underline{\exists \mathbf{x_3}} . F_0(x_1, x_2, x_3, \bar{y})$$

Nested Existential Quantifiers

$$\exists x_1.\exists x_2.\underline{\exists \mathbf{x_3}}.F_0(x_1, x_2, x_3, \bar{y})$$

$$\exists x_1.\underline{\exists \mathbf{x_2}}.F_1(x_1, x_2, \bar{y})$$

Nested Existential Quantifiers

$$\exists x_1 . \exists x_2 . \underline{\exists \mathbf{x_3}} . F_0(x_1, x_2, x_3, \bar{y})$$

$$\exists x_1 . \underline{\exists \mathbf{x_2}} . F_1(x_1, x_2, \bar{y})$$

$$\underline{\exists \mathbf{x_1}} . F_2(x_1, \bar{y})$$

Nested Existential Quantifiers

$$\exists x_1 . \exists x_2 . \underline{\exists \mathbf{x_3}} . F_0(x_1, x_2, x_3, \bar{y})$$

$$\exists x_1 . \underline{\exists \mathbf{x_2}} . F_1(x_1, x_2, \bar{y})$$

$$\underline{\exists \mathbf{x_1}} . F_2(x_1, \bar{y})$$

$$F_3(\bar{y})$$

Nested Existential Quantifiers

$$\exists x_1.\exists x_2.\underline{\exists \mathbf{x_3}.}F_0(x_1, x_2, x_3, \bar{y})$$

$$\exists x_1.\underline{\exists \mathbf{x_2}.}F_1(x_1, x_2, \bar{y})$$

$$\underline{\exists \mathbf{x_1}.}F_2(x_1, \bar{y})$$

$$F_3(\bar{y})$$

☺

Universal Quantifiers

If $F_0(x, \bar{y})$ is quantifier-free, how to eliminate

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

Universal Quantifiers

If $F_0(x, \bar{y})$ is quantifier-free, how to eliminate

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

Equivalence (property always holds if there is no counterexample):

$$\forall y. F_0(x, \bar{y}) \iff \neg\left[\exists y. \neg F_0(x, \bar{y})\right]$$

It thus suffices to process:

$$\neg\left[\exists y. \neg F_0(x, \bar{y})\right]$$

Note that $\neg F_0(x, \bar{y})$ is quantifier-free, so we know how to handle it:

$$\exists y. \neg F_0(x, \bar{y}) \quad \rightsquigarrow \quad F_1(\bar{y})$$

Therefore, we obtain

$$\neg F_1(\bar{y})$$

Removing any **alternation** of quantifiers: illustration

Alternation: switch between existentials and universals

$$\exists x_1.\forall x_2.\forall x_3.\exists x_4.F_0(x_1,x_2,x_3,x_4,\bar{y})$$

$$\exists x_1.\neg\exists x_2.\exists x_3.\neg\underline{\exists x_4.F_0(x_1,x_2,x_3,x_4,\bar{y})}$$

$$\exists x_1.\neg\exists x_2.\underline{\exists x_3.\neg F_1(x_1,x_2,x_3,\bar{y})}$$

$$\exists x_1.\neg\underline{\exists x_2.F_2(x_1,x_2,\bar{y})}$$

$$\exists x_1.\neg F_3(x_1,\bar{y})$$

$$F_4(\bar{y})$$

Each quantifier alternation involves a disjunctive normal form transformation.
In practice, we do not have many alternations.

Back to Presburger Arithmetic

Consider the quantifier elimination problem of the form:

$$\exists y.\ L_1 \wedge \ldots \wedge L_n$$

where L_i are literals from PA.

Note that, for integers:

- $\neg(x < y) \iff y \leq x$
- $x < y \iff x + 1 \leq y$
- $x \leq y \iff x < y + 1$

We use these observations below.

Instead of $\leq$ we choose to use $<$ only.

We do not write $x > y$ but only $y < x$.

Normalizing Literals for PA

Normal Form of Terms: All *terms* are built from $K, +, -, K \cdot \_$, so using standard transformations they can be represented as: $K_0 + \sum_{i=1}^{n} K_i x_i$ We call such term a linear term.
Normal Form for Literals in PA:

$$\begin{aligned}
\neg(t_1 < t_2) &\text{ becomes } t_2 < t_1 + 1 \\
\neg(t_1 = t_2) &\text{ becomes } t_1 < t_2 \vee t_2 < t_1 \\
t_1 = t_2 &\text{ becomes } t_1 < t_2 + 1 \wedge t_2 < t_1 + 1 \quad (*) \\
\neg(K \mid t) &\text{ becomes } \bigvee_{i=1}^{K-1} K \mid t + i \\
t_1 < t_2 &\text{ becomes } 0 < t_2 - t_1
\end{aligned}$$

To remove disjunctions we generated, compute DNF again.
$(*)$ We transformed equalities just for simplicity. Usually we handle them directly.

Why one-point rule will not be enough

Note that we must handle inequalities, not merely equalities

If we have integers, we cannot always divide perfectly.
Variable to eliminate can occur not as y but as, e.g. $3y$

Exposing the Variable to Eliminate: Example

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

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:

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

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.

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

Exposing the Variable to Eliminate in General

Eliminating y from conjunction $F(y)$ of literals:

- $0 < t$
- $K \mid t$

where t is a linear term. To eliminate $\exists y$ from such conjunction, we wish to ensure that the coefficient next to y is one or minus one.
Observation:

- $0 < t$ is equivalent to $0 < c\,t$
- $K \mid t$ is equivalent to $c\,K \mid c\,t$

for c a positive integer.
Let $K_1, \ldots, K_n$ be all coefficients next to y in the formula.
Let M be a positive integer such that $K_i \mid M$ for all i, $1 \le i \le n$

- for example, let M be the **least common multiple**

$$M = lcm(K_1, \ldots, K_n)$$

Ensuring Coefficient One

Multiply each literal where y occurs in subterm $K_i y$ by constant $M/|K_i|$

- the point is, M is divisible by $|K_i|$ by construction

What is the coefficient next to y in the resulting formula?

Ensuring Coefficient One

Multiply each literal where y occurs in subterm $K_i y$ by constant $M/|K_i|$

- the point is, M is divisible by $|K_i|$ by construction

What is the coefficient next to y in the resulting formula?

$$M \text{ or } -M$$

Ensuring Coefficient One

Multiply each literal where y occurs in subterm $K_i y$ by constant $M/|K_i|$

- the point is, M is divisible by $|K_i|$ by construction

What is the coefficient next to y in the resulting formula?

$$M \text{ or } -M$$

We obtain a formula of the form $\exists y.F(M \cdot y)$.
Letting $x = My$, we conclude the formula is equivalent to

$$\exists x.\ F(x) \wedge (M \mid x)$$

What is the coefficient next to y in the resulting formula?

Ensuring Coefficient One

Multiply each literal where y occurs in subterm $K_i y$ by constant $M/|K_i|$

- the point is, M is divisible by $|K_i|$ by construction

What is the coefficient next to y in the resulting formula?

$$M \text{ or } -M$$

We obtain a formula of the form $\exists y.F(M \cdot y)$.
Letting $x = My$, we conclude the formula is equivalent to

$$\exists x.\ F(x) \wedge (M \mid x)$$

What is the coefficient next to y in the resulting formula?

$$1 \text{ or } -1$$

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

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

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? least common multiple of $K_1,\ldots,K_D$
Note that if $F_1(u)$ holds then also $F_1(u-N)$ holds.

Back to Example

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

Back to Example

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

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

Special cases

What if $L = 0$? 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

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

It works

We finished describing a complete quantifier elimination algorithm for Presburger Arithmetic!

It works

We finished describing a complete quantifier elimination algorithm for Presburger Arithmetic!

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

Many other properties follow (e.g. interpolation).

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.

More on QE: One Direction to Make it More Efficient

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

Eliminate Quantifiers: Example

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

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

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

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

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

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:

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

(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:

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

(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

$$(t > 0 \land F(t)) \lor (\neg (t > 0) \land F(-t))$$

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.