

THE CALCULUS OF COMPUTATION:  
Decision Procedures with  
Applications to Verification

by  
Aaron Bradley  
Zohar Manna

Springer 2007

PART I: FOUNDATION

1. Propositional Logic (1)
2. First-Order Logic (2)
3. First-Order Theories (1)
4. Induction (2)
5. Program Correctness: Mechanics (2)
6. Program Correctness: Strategies (1)



PART II: ALGORITHMIC REASONING

7. Quantified Linear Arithmetic (1)
8. Quantifier-Free Linear Arithmetic (2)
9. Quantifier-Free Equality and Data Structures (2)
10. Combining Decision Procedures (1)
11. Arrays (2)
12. Invariant Generation (1)



Part I: FOUNDATIONS

1. Propositional Logic(PL)

## Propositional Logic(PL)

### PL Syntax

<u>Atom</u>	truth symbols $\top$ ("true") and $\perp$ ("false") propositional variables $P, Q, R, P_1, Q_1, R_1, \dots$
<u>Literal</u>	atom $\alpha$ or its negation $\neg\alpha$
<u>Formula</u>	literal or application of a logical connective to formulae $F, F_1, F_2$
	$\neg F$ "not" (negation)
	$F_1 \wedge F_2$ "and" (conjunction)
	$F_1 \vee F_2$ "or" (disjunction)
	$F_1 \rightarrow F_2$ "implies" (implication)
	$F_1 \leftrightarrow F_2$ "if and only if" (iff)

### Example:

formula  $F : (P \wedge Q) \rightarrow (T \vee \neg Q)$

atoms:  $P, Q, T$

literal:  $\neg Q$

subformulas:  $P \wedge Q, T \vee \neg Q$

abbreviation

$$F : P \wedge Q \rightarrow T \vee \neg Q$$



## PL Semantics (meaning)

Sentence  $F$  + Interpretation  $I$  = Truth value  
(true, false)

Interpretation

$$I : \{P \mapsto \text{true}, Q \mapsto \text{false}, \dots\}$$

Evaluation of  $F$  under  $I$ :

$F$	$\neg F$	
0	1	where 0 corresponds to value false
1	0	1 true

$F_1$	$F_2$	$F_1 \wedge F_2$	$F_1 \vee F_2$	$F_1 \rightarrow F_2$	$F_1 \leftrightarrow F_2$
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	0
1	1	1	1	1	1



### Example:

$F : P \wedge Q \rightarrow P \vee \neg Q$

$I : \{P \mapsto \text{true}, Q \mapsto \text{false}\}$

$P$	$Q$	$\neg Q$	$P \wedge Q$	$P \vee \neg Q$	$F$
1	0	1	0	1	1

1 = true

0 = false

$F$  evaluates to true under  $I$

## Inductive Definition of PL's Semantics

$I \models F$  if  $F$  evaluates to true under  $I$   
 $I \not\models F$  if  $F$  evaluates to false under  $I$

### Base Case:

$I \models \top$   
 $I \not\models \perp$   
 $I \models P$  iff  $I[P] = \text{true}$   
 $I \not\models P$  iff  $I[P] = \text{false}$

### Inductive Case:

$I \models \neg F$  iff  $I \not\models F$   
 $I \models F_1 \wedge F_2$  iff  $I \models F_1$  and  $I \models F_2$   
 $I \models F_1 \vee F_2$  iff  $I \models F_1$  or  $I \models F_2$   
 $I \models F_1 \rightarrow F_2$  iff, if  $I \models F_1$  then  $I \models F_2$   
 $I \models F_1 \leftrightarrow F_2$  iff,  $I \models F_1$  and  $I \models F_2$ ,  
 or  $I \not\models F_1$  and  $I \not\models F_2$

### Note:

$I \not\models F_1 \rightarrow F_2$  iff  $I \models F_1$  and  $I \not\models F_2$

### Example:

$$F : P \wedge Q \rightarrow P \vee \neg Q$$

$$I : \{P \mapsto \text{true}, Q \mapsto \text{false}\}$$

- $I \models P$  since  $I[P] = \text{true}$
- $I \not\models Q$  since  $I[Q] = \text{false}$
- $I \models \neg Q$  by 2 and  $\neg$
- $I \not\models P \wedge Q$  by 2 and  $\wedge$
- $I \models P \vee \neg Q$  by 1 and  $\vee$
- $I \models F$  by 4 and  $\rightarrow$  Why?

Thus,  $F$  is true under  $I$ .

## Satisfiability and Validity

$F$  **satisfiable** iff there exists an interpretation  $I$  such that  $I \models F$ .

$F$  **valid** iff for all interpretations  $I$ ,  $I \models F$ .

$F$  is valid iff  $\neg F$  is unsatisfiable

### Method 1: Truth Tables

Example  $F : P \wedge Q \rightarrow P \vee \neg Q$

$P$	$Q$	$P \wedge Q$	$\neg Q$	$P \vee \neg Q$	$F$
0	0	0	1	1	1
0	1	0	0	0	1
1	0	0	1	1	1
1	1	1	0	1	1

Thus  $F$  is valid.

Example  $F : P \vee Q \rightarrow P \wedge Q$

$P$	$Q$	$P \vee Q$	$P \wedge Q$	$F$
0	0	0	0	1
0	1	1	0	0
1	0	1	0	0
1	1	1	1	1

← satisfying  $I$

← falsifying  $I$

Thus  $F$  is satisfiable, but invalid.

## Method 2: Semantic Argument

### Proof rules

$$\frac{I \models \neg F}{I \not\models F}$$

$$\frac{I \not\models \neg F}{I \models F}$$

$$\frac{I \models F \wedge G}{I \models F \quad \text{and} \quad I \models G}$$

$$\frac{I \not\models F \wedge G}{I \not\models F \quad \text{or} \quad I \not\models G}$$

$$\frac{I \models F \vee G}{I \models F \quad | \quad I \models G}$$

$$\frac{I \not\models F \vee G}{I \not\models F \quad | \quad I \not\models G}$$

$$\frac{I \models F \rightarrow G}{I \not\models F \quad | \quad I \models G}$$

$$\frac{I \not\models F \rightarrow G}{I \models F \quad | \quad I \not\models G}$$

$$\frac{I \models F \rightarrow G}{I \models F \wedge G \quad | \quad I \not\models F \vee G}$$

$$\frac{I \not\models F \rightarrow G}{I \models F \wedge \neg G \quad | \quad I \models \neg F \wedge G}$$

$$\frac{I \models F}{I \not\models F}$$

$$\frac{I \not\models F}{I \models \perp}$$

◀ ▶ ↻ 🔍 ↺ ↻ 🔍

### Example 2: Prove

$F: (P \rightarrow Q) \wedge (Q \rightarrow R) \rightarrow (P \rightarrow R)$  is valid.

Let's assume that  $F$  is not valid.

- |   |                     |
|---|---------------------|
| 1. $I \not\models F$                                      | assumption          |
| 2. $I \models (P \rightarrow Q) \wedge (Q \rightarrow R)$ | 1 and $\rightarrow$ |
| 3. $I \not\models P \rightarrow R$                        | 1 and $\rightarrow$ |
| 4. $I \models P$  | 3 and $\rightarrow$ |
| 5. $I \not\models R$                                      | 3 and $\rightarrow$ |
| 6. $I \models P \rightarrow Q$                            | 2 and of $\wedge$   |
| 7. $I \models Q \rightarrow R$                            | 2 and of $\wedge$   |

◀ ▶ ↻ 🔍 ↺ ↻ 🔍

### Example 1: Prove

$F: P \wedge Q \rightarrow P \vee \neg Q$  is valid.

Let's assume that  $F$  is not valid and that  $I$  is a falsifying interpretation.

- |   |                           |
|---|---------------------------|
| 1. $I \not\models P \wedge Q \rightarrow P \vee \neg Q$ | assumption                |
| 2. $I \models P \wedge Q$                               | 1 and $\rightarrow$       |
| 3. $I \not\models P \vee \neg Q$                        | 1 and $\rightarrow$       |
| 4. $I \models P$  | 2 and $\wedge$            |
| 5. $I \not\models P$                                    | 3 and $\vee$              |
| 6. $I \models \perp$                                    | 4 and 5 are contradictory |

Thus  $F$  is valid.

◀ ▶ ↻ 🔍 ↺ ↻ 🔍

### Two cases from 6

- |                       |                            |
|-----------------------|----------------------------|
| 8a. $I \not\models P$ | 6 and $\rightarrow$        |
| 9a. $I \models \perp$ | 4 and 8a are contradictory |

and

- |                   |                     |
|-------------------|---------------------|
| 8b. $I \models Q$ | 6 and $\rightarrow$ |
|-------------------|---------------------|

### Two cases from 7

- |                         |                              |
|-------------------------|------------------------------|
| 9ba. $I \not\models Q$  | 7 and $\rightarrow$          |
| 10ba. $I \models \perp$ | 8b and 9ba are contradictory |

and

- |                         |                             |
|-------------------------|-----------------------------|
| 9bb. $I \models R$      | 7 and $\rightarrow$         |
| 10bb. $I \models \perp$ | 5 and 9bb are contradictory |

Our assumption is incorrect in all cases —  $F$  is valid.

◀ ▶ ↻ 🔍 ↺ ↻ 🔍

### Example 3: Is

$$F: P \vee Q \rightarrow P \wedge Q \text{ valid?}$$

Let's assume that  $F$  is not valid.

1.  $I \not\models P \vee Q \rightarrow P \wedge Q$  assumption
2.  $I \models P \vee Q$  1 and  $\rightarrow$
3.  $I \not\models P \wedge Q$  1 and  $\rightarrow$

Two options

- 4a.  $I \models P$  2 and  $\vee$     4b.  $I \models Q$  2 and  $\vee$   
 5a.  $I \not\models Q$  3 and  $\wedge$     5b.  $I \not\models P$  3 and  $\wedge$

We cannot derive a contradiction.  $F$  is not valid.

Falsifying interpretation:

$$I_1: \{P \mapsto \text{true}, Q \mapsto \text{false}\} \quad I_2: \{Q \mapsto \text{true}, P \mapsto \text{false}\}$$

We have to derive a contradiction in both cases for  $F$  to be valid.

## Equivalence

$F_1$  and  $F_2$  are equivalent ( $F_1 \Leftrightarrow F_2$ )

iff for all interpretations  $I, I \models F_1 \leftrightarrow F_2$

To prove  $F_1 \Leftrightarrow F_2$  show  $F_1 \leftrightarrow F_2$  is valid.

$F_1$  implies  $F_2$  ( $F_1 \Rightarrow F_2$ )

iff for all interpretations  $I, I \models F_1 \rightarrow F_2$

$F_1 \Leftrightarrow F_2$  and  $F_1 \Rightarrow F_2$  are not formulae!

## Normal Forms

### 1. Negation Normal Form (NNF)

Negations appear only in literals. (only  $\neg, \wedge, \vee$ )

To transform  $F$  to equivalent  $F'$  in NNF we recursively use the following template equivalences (left-to-right):

$$\begin{aligned} \neg\neg F_1 &\Leftrightarrow F_1 & \neg\top &\Leftrightarrow \perp & \neg\perp &\Leftrightarrow \top \\ \neg(F_1 \wedge F_2) &\Leftrightarrow \neg F_1 \vee \neg F_2 \\ \neg(F_1 \vee F_2) &\Leftrightarrow \neg F_1 \wedge \neg F_2 \end{aligned} \left. \vphantom{\begin{aligned} \neg\neg F_1 &\Leftrightarrow F_1 \\ \neg(F_1 \wedge F_2) &\Leftrightarrow \neg F_1 \vee \neg F_2 \\ \neg(F_1 \vee F_2) &\Leftrightarrow \neg F_1 \wedge \neg F_2 \end{aligned}} \right\} \text{De Morgan's Law}$$

$$F_1 \rightarrow F_2 \Leftrightarrow \neg F_1 \vee F_2$$

$$F_1 \leftrightarrow F_2 \Leftrightarrow (F_1 \rightarrow F_2) \wedge (F_2 \rightarrow F_1)$$

Example: Convert  $F: \neg(P \rightarrow \neg(P \wedge Q))$  to NNF

$$\begin{aligned} F' &: \neg(\neg P \vee \neg(P \wedge Q)) && \rightarrow \text{to } \vee \\ F'' &: \neg\neg P \wedge \neg\neg(P \wedge Q) && \text{De Morgan's Law} \\ F''' &: P \wedge P \wedge Q && \neg\neg \end{aligned}$$

$F'''$  is equivalent to  $F$  ( $F''' \Leftrightarrow F$ ) and is in NNF.

### 2. Disjunctive Normal Form (DNF)

Disjunction of conjunctions of literals

$$\bigvee_i \bigwedge_j \ell_{i,j} \text{ for literals } \ell_{i,j}$$

To convert  $F$  into equivalent  $F'$  in DNF, transform  $F$  into NNF and then use the following template equivalences (left-to-right):

$$\begin{aligned} (F_1 \vee F_2) \wedge F_3 &\Leftrightarrow (F_1 \wedge F_3) \vee (F_2 \wedge F_3) \\ F_1 \wedge (F_2 \vee F_3) &\Leftrightarrow (F_1 \wedge F_2) \vee (F_1 \wedge F_3) \end{aligned} \left. \vphantom{\begin{aligned} (F_1 \vee F_2) \wedge F_3 &\Leftrightarrow (F_1 \wedge F_3) \vee (F_2 \wedge F_3) \\ F_1 \wedge (F_2 \vee F_3) &\Leftrightarrow (F_1 \wedge F_2) \vee (F_1 \wedge F_3) \end{aligned}} \right\} \text{dist}$$

Example: Convert

$$F: (Q_1 \vee \neg\neg Q_2) \wedge (\neg R_1 \rightarrow R_2) \text{ into DNF}$$

$$\begin{aligned} F' &: (Q_1 \vee Q_2) \wedge (R_1 \vee R_2) && \text{in NNF} \\ F'' &: (Q_1 \wedge (R_1 \vee R_2)) \vee (Q_2 \wedge (R_1 \vee R_2)) && \text{dist} \\ F''' &: (Q_1 \wedge R_1) \vee (Q_1 \wedge R_2) \vee (Q_2 \wedge R_1) \vee (Q_2 \wedge R_2) && \text{dist} \end{aligned}$$

$F'''$  is equivalent to  $F$  ( $F''' \Leftrightarrow F$ ) and is in DNF.

### 3. Conjunctive Normal Form (CNF)

Conjunction of disjunctions of literals

$$\bigwedge_i \bigvee_j \ell_{i,j} \text{ for literals } \ell_{i,j}$$

To convert  $F$  into equivalent  $F'$  in CNF, transform  $F$  into NNF and then use the following template equivalences (left-to-right):

$$\begin{aligned} (F_1 \wedge F_2) \vee F_3 &\Leftrightarrow (F_1 \vee F_3) \wedge (F_2 \vee F_3) \\ F_1 \vee (F_2 \wedge F_3) &\Leftrightarrow (F_1 \vee F_2) \wedge (F_1 \vee F_3) \end{aligned}$$

Navigation icons

### Boolean Constraint Propagation (BCP)

Based on unit resolution

$$\frac{\ell \quad C[-\ell]}{C[\perp]} \leftarrow \text{clause} \quad \text{where } \ell = P \text{ or } \ell = \neg P$$

throughout

Example:

$$F: (\neg P \vee Q \vee R) \wedge (\neg Q \vee R) \wedge (\neg Q \vee \neg R) \wedge (P \vee \neg Q \vee \neg R)$$

Branching on  $Q$

$$F\{Q \mapsto \top\}: (R) \wedge (\neg R) \wedge (P \vee \neg R)$$

By unit resolution

$$\frac{R \quad (\neg R)}{\perp}$$

$$F\{Q \mapsto \top\} = \perp \Rightarrow \text{false}$$

Navigation icons

### Davis-Putnam-Logemann-Loveland (DPLL) Algorithm

Decides the satisfiability of PL formulae in CNF

In book, efficient conversion of  $F$  to  $F'$  where

$F'$  is in CNF and

$F'$  and  $F$  are equisatisfiable ( $F$  is satisfiable iff  $F'$  is satisfiable)

Decision Procedure DPLL: Given  $F$  in CNF

DPLL( $F$ ) =

```

if  $F = \top$  then true           ...  $F$  is satisfiable
else if  $F = \perp$  then false     ...  $F$  is unsatisfiable
else if some  $P$  appears only positively then DPLL( $F\{P \mapsto \top\}$ )
else if some  $P$  appears only negatively then DPLL( $F\{P \mapsto \perp\}$ )
else
  let  $F' = \text{BCP}(F)$  in
  if  $F' \neq F$  then DPLL( $F'$ )
  else
    let  $P = \text{choose}(\text{vars}(F))$  in
    DPLL( $F\{P \mapsto \top\}$ )  $\vee$  DPLL( $F\{P \mapsto \perp\}$ )
  
```

Navigation icons

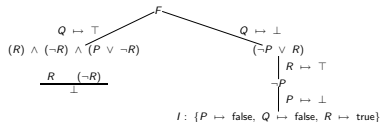
On the other branch

$$F\{Q \mapsto \perp\}: (\neg P \vee R)$$

$$F\{Q \mapsto \perp, R \mapsto \top, P \mapsto \perp\} = \top \Rightarrow \text{true}$$

$F$  is satisfiable with satisfying interpretation

$$I: \{P \mapsto \text{false}, Q \mapsto \text{false}, R \mapsto \text{true}\}$$



Navigation icons



## Translations of English Sentences into FOL

- ▶ The length of one side of a triangle is less than the sum of the lengths of the other two sides

$$\forall x, y, z. \text{triangle}(x, y, z) \rightarrow \text{length}(x) < \text{length}(y) + \text{length}(z)$$

- ▶ Fermat's Last Theorem.

$$\begin{aligned} \forall n. \text{integer}(n) \wedge n > 2 \\ \rightarrow \forall x, y, z. \\ \text{integer}(x) \wedge \text{integer}(y) \wedge \text{integer}(z) \\ \wedge x > 0 \wedge y > 0 \wedge z > 0 \\ \rightarrow x^n + y^n \neq z^n \end{aligned}$$



### Example:

$$F: p(f(x, y), z) \rightarrow p(y, g(z, x))$$

Interpretation  $I: (D_I, \alpha_I)$

$$D_I = \mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\} \quad \text{integers}$$

$$\alpha_I: \{f \mapsto +, g \mapsto -, p \mapsto >\}$$

Therefore, we can write

$$F_I: x + y > z \rightarrow y > z - x$$

(This is the way we'll write it in the future!)

Also

$$\alpha_I: \{x \mapsto 13, y \mapsto 42, z \mapsto 1\}$$

Thus

$$F_I: 13 + 42 > 1 \rightarrow 42 > 1 - 13$$

Compute the truth value of  $F$  under  $I$

1.  $I \models x + y > z$  since  $13 + 42 > 1$
2.  $I \models y > z - x$  since  $42 > 1 - 13$
3.  $I \models F$  by 1, 2, and  $\rightarrow$

$F$  is true under  $I$



## FOL Semantics

An interpretation  $I: (D_I, \alpha_I)$  consists of:

- ▶ Domain  $D_I$   
non-empty set of values or objects  
cardinality  $|D_I|$  finite (eg, 52 cards),  
countably infinite (eg, integers), or  
uncountably infinite (eg, reals)

- ▶ Assignment  $\alpha_I$

- ▶ each variable  $x$  assigned value  $x_I \in D_I$
- ▶ each  $n$ -ary function  $f$  assigned

$$f_I: D_I^n \rightarrow D_I$$

In particular, each constant  $a$  (0-ary function) assigned value  $a_I \in D_I$

- ▶ each  $n$ -ary predicate  $p$  assigned

$$p_I: D_I^n \rightarrow \{\text{true}, \text{false}\}$$

In particular, each propositional variable  $P$  (0-ary predicate) assigned truth value (true, false)



## Semantics: Quantifiers

$x$  variable.

$x$ -variant of interpretation  $I$  is an interpretation  $J: (D_J, \alpha_J)$  such that

- ▶  $D_I = D_J$
- ▶  $\alpha_J[y] = \alpha_I[y]$  for all symbols  $y$ , except possibly  $x$

That is,  $I$  and  $J$  agree on everything except possibly the value of  $x$

Denote  $J: I \triangleleft \{x \mapsto v\}$  the  $x$ -variant of  $I$  in which  $\alpha_J[x] = v$  for some  $v \in D_I$ . Then

- ▶  $I \models \forall x. F$  iff for all  $v \in D_I$ ,  $I \triangleleft \{x \mapsto v\} \models F$
- ▶  $I \models \exists x. F$  iff there exists  $v \in D_I$  s.t.  $I \triangleleft \{x \mapsto v\} \models F$



### Example

For  $\mathbb{Q}$ , the set of rational numbers, consider

$$F_I : \forall x. \exists y. 2 \times y = x$$

Compute the value of  $F_I$  ( $F$  under  $I$ ):

Let

$$J_1 : I \triangleleft \{x \mapsto v\} \quad J_2 : J_1 \triangleleft \{y \mapsto \frac{v}{2}\}$$

$x$ -variant of  $I$                        $y$ -variant of  $J_1$

for  $v \in \mathbb{Q}$ .

Then

1.  $J_2 \models 2 \times y = x$                       since  $2 \times \frac{v}{2} = v$
2.  $J_1 \models \exists y. 2 \times y = x$
3.  $I \models \forall x. \exists y. 2 \times y = x$               since  $v \in \mathbb{Q}$  is arbitrary



Second case

1.  $I \not\models \forall x. p(x)$                       assumption
2.  $I \models \neg \exists x. \neg p(x)$                   assumption
3.  $I \triangleleft \{x \mapsto v\} \not\models p(x)$                   1 and  $\forall$ , for some  $v \in D_I$
4.  $I \not\models \exists x. \neg p(x)$                       2 and  $\neg$
5.  $I \triangleleft \{x \mapsto v\} \not\models \neg p(x)$               4 and  $\exists$
6.  $I \triangleleft \{x \mapsto v\} \models p(x)$                   5 and  $\neg$

3 and 6 are contradictory.

Both cases end in contradictions for arbitrary  $I \Rightarrow F$  is valid.



## Satisfiability and Validity

$F$  is satisfiable iff there exists  $I$  s.t.  $I \models F$

$F$  is valid iff for all  $I$ ,  $I \models F$

$F$  is valid iff  $\neg F$  is unsatisfiable

Example:  $F : (\forall x. p(x)) \leftrightarrow (\neg \exists x. \neg p(x))$  valid?

Suppose not. Then there is  $I$  s.t.

$$0. \quad I \not\models (\forall x. p(x)) \leftrightarrow (\neg \exists x. \neg p(x))$$

First case

1.  $I \models \forall x. p(x)$                       assumption
2.  $I \not\models \neg \exists x. \neg p(x)$                       assumption
3.  $I \models \exists x. \neg p(x)$                       2 and  $\neg$
4.  $I \triangleleft \{x \mapsto v\} \models \neg p(x)$                   3 and  $\exists$ , for some  $v \in D_I$
5.  $I \triangleleft \{x \mapsto v\} \models p(x)$                       1 and  $\forall$

4 and 5 are contradictory.



Example: Prove

$$F : p(a) \rightarrow \exists x. p(x) \text{ is valid.}$$

Assume otherwise.

1.  $I \not\models F$                                   assumption
2.  $I \models p(a)$                               1 and  $\rightarrow$
3.  $I \not\models \exists x. p(x)$                           1 and  $\rightarrow$
4.  $I \triangleleft \{x \mapsto a_I[a]\} \not\models p(x)$               3 and  $\exists$

2 and 4 are contradictory. Thus,  $F$  is valid.



Example: Show

$F : (\forall x. p(x,x)) \rightarrow (\exists x. \forall y. p(x,y))$  is invalid.

Find interpretation  $I$  such that

$$I \models \neg[(\forall x. p(x,x)) \rightarrow (\exists x. \forall y. p(x,y))]$$

i.e.

$$I \models (\forall x. p(x,x)) \wedge \neg(\exists x. \forall y. p(x,y))$$

Choose  $D_I = \{0, 1\}$

$\rho_I = \{(0,0), (1,1)\}$  i.e.  $\rho_I(0,0)$  and  $\rho_I(1,1)$  are true  
 $\rho_I(1,0)$  and  $\rho_I(0,1)$  are false

$I$  falsifying interpretation  $\Rightarrow F$  is invalid.

Rename  $x$  by  $x'$ :

replace  $x$  in  $\forall x$  by  $x'$  and all free  $x$  in the scope of  $\forall x$  by  $x'$ .

$$\forall x. G[x] \Leftrightarrow \forall x'. G[x']$$

Same for  $\exists x$

$$\exists x. G[x] \Leftrightarrow \exists x'. G[x']$$

where  $x'$  is a fresh variable

Proposition (Substitution of Equivalent Formulae)

$$\sigma : \{F_1 \mapsto G_1, \dots, F_n \mapsto G_n\}$$

s.t. for each  $i$ ,  $F_i \Leftrightarrow G_i$

If  $F\sigma$  a safe substitution, then  $F \Leftrightarrow F\sigma$

## Safe Substitution $F\sigma$

Example:

$$F : (\forall x. \overbrace{p(x,y)}^{\text{scope of } \forall x}) \rightarrow q(f(y), x)$$

bound by  $\forall x$  / free      free / free

$$\text{free}(F) = \{x, y\}$$

substitution

$$\sigma : \{x \mapsto g(x), y \mapsto f(x), q(f(y), x) \mapsto \exists x. h(x, y)\}$$

$F\sigma$ ?

1. Rename

$$F' : \forall x'. p(x', y) \rightarrow q(f(y), x)$$

↑      ↑

where  $x'$  is a fresh variable

2.  $F'\sigma : \forall x'. p(x', f(x)) \rightarrow \exists x. h(x, y)$

## Formula Schema

Formula

$$(\forall x. p(x)) \leftrightarrow (\neg \exists x. \neg p(x))$$

Formula Schema

$$H_1 : (\forall x. F) \leftrightarrow (\neg \exists x. \neg F)$$

↑ place holder

Formula Schema (with side condition)

$$H_2 : (\forall x. F) \leftrightarrow F \text{ provided } x \notin \text{free}(F)$$

Valid Formula Schema

$H$  is valid iff valid for any FOL formula  $F_i$  obeying the side conditions

Example:  $H_1$  and  $H_2$  are valid.

## Substitution $\sigma$ of $H$

$$\sigma : \{F_1 \mapsto \dots, F_n \mapsto \}$$

mapping place holders  $F_i$  of  $H$  to FOL formulae,  
(obeying the side conditions of  $H$ )

### Proposition (Formula Schema)

If  $H$  is valid formula schema and  
 $\sigma$  is a substitution obeying  $H$ 's side conditions  
then  $H\sigma$  is also valid.

#### Example:

$H : (\forall x. F) \leftrightarrow F$  provided  $x \notin \text{free}(F)$  is valid

$\sigma : \{F \mapsto p(y)\}$  obeys the side condition

Therefore  $H\sigma : \forall x. p(y) \leftrightarrow p(y)$  is valid

◀ ▶ ↻ 🔍

## Proving Validity of Formula Schema

Example: Prove validity of

$$H : (\forall x. F) \leftrightarrow F \text{ provided } x \notin \text{free}(F)$$

Proof by contradiction. Consider the two directions of  $\leftrightarrow$ .

First case:

- |    |     |               |                |  |
|----|-----|---------------|----------------|--|
| 1. | $I$ | $\models$     | $\forall x. F$ | assumption                                     |
| 2. | $I$ | $\not\models$ | $F$            | assumption                                     |
| 3. | $I$ | $\models$     | $F$            | 1, $\forall$ , since $x \notin \text{free}(F)$ |
| 4. | $I$ | $\models$     | $\perp$        | 2, 3   |

Second Case:

- |    |     |               |                     |  |
|----|-----|---------------|---------------------|--|
| 1. | $I$ | $\not\models$ | $\forall x. F$      | assumption                                     |
| 2. | $I$ | $\models$     | $F$                 | assumption                                     |
| 3. | $I$ | $\models$     | $\exists x. \neg F$ | 1 and $\neg$                                   |
| 4. | $I$ | $\models$     | $\neg F$            | 3, $\exists$ , since $x \notin \text{free}(F)$ |
| 5. | $I$ | $\models$     | $\perp$             | 2, 4   |

Hence,  $H$  is a valid formula schema.

◀ ▶ ↻ 🔍

## Normal Forms

### 1. Negation Normal Forms (NNF)

Augment the equivalence with (left-to-right)

$$\neg \forall x. F[x] \Leftrightarrow \exists x. \neg F[x]$$

$$\neg \exists x. F[x] \Leftrightarrow \forall x. \neg F[x]$$

#### Example

$$G : \forall x. (\exists y. p(x, y) \wedge p(x, z)) \rightarrow \exists w. p(x, w).$$

- $\forall x. (\exists y. p(x, y) \wedge p(x, z)) \rightarrow \exists w. p(x, w)$
- $\forall x. \neg(\exists y. p(x, y) \wedge p(x, z)) \vee \exists w. p(x, w)$   
 $F_1 \rightarrow F_2 \Leftrightarrow \neg F_1 \vee F_2$
- $\forall x. (\forall y. \neg(p(x, y) \wedge p(x, z))) \vee \exists w. p(x, w)$   
 $\neg \exists x. F[x] \Leftrightarrow \forall x. \neg F[x]$
- $\forall x. (\forall y. \neg p(x, y) \vee \neg p(x, z)) \vee \exists w. p(x, w)$

◀ ▶ ↻ 🔍

### 2. Prenex Normal Form (PNF)

All quantifiers appear at the beginning of the formula

$$Q_1 x_1 \dots Q_n x_n. F[x_1, \dots, x_n]$$

where  $Q_i \in \{\forall, \exists\}$  and  $F$  is quantifier-free.

Every FOL formula  $F$  can be transformed to formula  $F'$  in PNF  
s.t.  $F' \Leftrightarrow F$ .

Example: Find equivalent PNF of

$$F : \forall x. \neg(\exists y. p(x, y) \wedge p(x, z)) \vee \exists y. p(x, y)$$

↑  
to the end of the formula

1. Write  $F$  in NNF

$$F_1 : \forall x. (\forall y. \neg p(x, y) \vee \neg p(x, z)) \vee \exists y. p(x, y)$$

◀ ▶ ↻ 🔍

2. Rename quantified variables to fresh names

$$F_2 : \forall x. (\forall y. \neg p(x, y) \vee \neg p(x, z)) \vee \exists w. p(x, w)$$

↑  
in the scope of  $\forall x$

3. Remove all quantifiers to produce quantifier-free formula

$$F_3 : \neg p(x, y) \vee \neg p(x, z) \vee p(x, w)$$

4. Add the quantifiers before  $F_3$

$$F_4 : \forall x. \forall y. \exists w. \neg p(x, y) \vee \neg p(x, z) \vee p(x, w)$$

Alternately,

$$F'_4 : \forall x. \exists w. \forall y. \neg p(x, y) \vee \neg p(x, z) \vee p(x, w)$$

Note: In  $F_2$ ,  $\forall y$  is in the scope of  $\forall x$ , therefore the order of quantifiers must be  $\dots \forall x \dots \forall y \dots$

$$F_4 \Leftrightarrow F \text{ and } F'_4 \Leftrightarrow F$$

Note: However  $G \not\Leftrightarrow F$

$$G : \forall y. \exists w. \forall x. \neg p(x, y) \vee \neg p(x, z) \vee p(x, w)$$

## Decidability of FOL

- ▶ FOL is undecidable (Turing & Church)

There does not exist an algorithm for deciding if a FOL formula  $F$  is valid, i.e. always halt and says "yes" if  $F$  is valid or say "no" if  $F$  is invalid.

- ▶ FOL is semi-decidable

There is a procedure that always halts and says "yes" if  $F$  is valid, but may not halt if  $F$  is invalid.

On the other hand,

- ▶ PL is decidable

There does exist an algorithm for deciding if a PL formula  $F$  is valid, e.g. the truth-table procedure.

Similarly for satisfiability

## Semantic Argument Proof

To show FOL formula  $F$  is valid, assume  $I \not\models F$  and derive a contradiction  $I \models \perp$  in all branches

- ▶ Soundness

If every branch of a semantic argument proof reach  $I \models \perp$ , then  $F$  is valid

- ▶ Completeness

Each valid formula  $F$  has a semantic argument proof in which every branch reach  $I \models \perp$

## 3. First-Order Theories

## First-Order Theories

First-order theory  $T$  defined by

- ▶ Signature  $\Sigma$  - set of constant, function, and predicate symbols
- ▶ Set of axioms  $A_T$  - set of closed (no free variables)  $\Sigma$ -formulae

$\Sigma$ -formula constructed of constants, functions, and predicate symbols from  $\Sigma$ , and variables, logical connectives, and quantifiers

The symbols of  $\Sigma$  are just symbols without prior meaning — the axioms of  $T$  provide their meaning

A  $\Sigma$ -formula  $F$  is valid in theory  $T$  ( $T$ -valid, also  $T \models F$ ), if every interpretation  $I$  that satisfies the axioms of  $T$ ,

i.e.  $I \models A$  for every  $A \in A_T$  ( $T$ -interpretation)

also satisfies  $F$ ,

i.e.  $I \models F$



## Theory of Equality $T_E$

Signature

$\Sigma = \{=, a, b, c, \dots, f, g, h, \dots, p, q, r, \dots\}$   
consists of

- ▶  $=$ , a binary predicate, interpreted by axioms.
- ▶ all constant, function, and predicate symbols.

Axioms of  $T_E$

- $\forall x. x = x$  (reflexivity)
- $\forall x, y. x = y \rightarrow y = x$  (symmetry)
- $\forall x, y, z. x = y \wedge y = z \rightarrow x = z$  (transitivity)
- for each positive integer  $n$  and  $n$ -ary function symbol  $f$ ,  
 $\forall x_1, \dots, x_n, y_1, \dots, y_n. \bigwedge_i x_i = y_i \rightarrow f(x_1, \dots, x_n) = f(y_1, \dots, y_n)$  (congruence)
- for each positive integer  $n$  and  $n$ -ary predicate symbol  $p$ ,  
 $\forall x_1, \dots, x_n, y_1, \dots, y_n. \bigwedge_i x_i = y_i \rightarrow (p(x_1, \dots, x_n) \leftrightarrow p(y_1, \dots, y_n))$  (equivalence)

Congruence and Equivalence are axiom schemata. For example,

Congruence for binary function  $f_2$  for  $n = 2$ :

$$\forall x_1, x_2, y_1, y_2. x_1 = y_1 \wedge x_2 = y_2 \rightarrow f_2(x_1, x_2) = f_2(y_1, y_2)$$



A  $\Sigma$ -formula  $F$  is satisfiable in  $T$  ( $T$ -satisfiable), if there is a  $T$ -interpretation (i.e. satisfies all the axioms of  $T$ ) that satisfies  $F$

Two formulae  $F_1$  and  $F_2$  are equivalent in  $T$  ( $T$ -equivalent), if  $T \models F_1 \leftrightarrow F_2$ .

i.e. if for every  $T$ -interpretation  $I$ ,  $I \models F_1$  iff  $I \models F_2$

A fragment of theory  $T$  is a syntactically-restricted subset of formulae of the theory.

Example: quantifier-free segment of theory  $T$  is the set of quantifier-free formulae in  $T$ .

A theory  $T$  is decidable if  $T \models F$  ( $T$ -validity) is decidable for every  $\Sigma$ -formula  $F$ ,

i.e., there is an algorithm that always terminate with "yes", if  $F$  is  $T$ -valid, and "no", if  $F$  is  $T$ -invalid.

A fragment of  $T$  is decidable if  $T \models F$  is decidable for every  $\Sigma$ -formula  $F$  in the fragment.



$T_E$  is undecidable.

The quantifier-free fragment of  $T_E$  is decidable. Very efficient algorithm.

Semantic argument method can be used for  $T_E$

Example: Prove

$$F: a = b \wedge b = c \rightarrow g(f(a), b) = g(f(c), a) \quad T_E\text{-valid.}$$

Suppose not; then there exists a  $T_E$ -interpretation  $I$  such that  $I \not\models F$ . Then,

- |  |                                |
|--|--------------------------------|
| 1. $I \not\models F$                       | assumption                     |
| 2. $I \models a = b \wedge b = c$          | 1, $\rightarrow$               |
| 3. $I \not\models g(f(a), b) = g(f(c), a)$ | 1, $\rightarrow$               |
| 4. $I \models a = b$                       | 2, $\wedge$                    |
| 5. $I \models b = c$                       | 2, $\wedge$                    |
| 6. $I \models a = c$                       | 4, 5, (transitivity)           |
| 7. $I \models f(a) = f(c)$                 | 6, (congruence)                |
| 8. $I \models g(f(a), b) = g(f(c), a)$     | 4, 7, (congruence), (symmetry) |

3 and 8 are contradictory  $\Rightarrow F$  is  $T_E$ -valid



## Natural Numbers and Integers

Natural numbers  $\mathbb{N} = \{0, 1, 2, \dots\}$   
 Integers  $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$

Three variations:

- ▶ Peano arithmetic  $T_{PA}$ : natural numbers with addition and multiplication
- ▶ Presburger arithmetic  $T_{\mathbb{N}}$ : natural numbers with addition
- ▶ Theory of integers  $T_{\mathbb{Z}}$ : integers with  $+$ ,  $-$ ,  $>$

### 1. Peano Arithmetic $T_{PA}$ (first-order arithmetic)

$\Sigma_{PA} : \{0, 1, +, \cdot, =\}$

The axioms:

- $\forall x. \neg(x + 1 = 0)$  (zero)
- $\forall x, y. x + 1 = y + 1 \rightarrow x = y$  (successor)
- $F[0] \wedge (\forall x. F[x] \rightarrow F[x + 1]) \rightarrow \forall x. F[x]$  (induction)
- $\forall x. x + 0 = x$  (plus zero)
- $\forall x, y. x + (y + 1) = (x + y) + 1$  (plus successor)
- $\forall x. x \cdot 0 = 0$  (times zero)
- $\forall x, y. x \cdot (y + 1) = x \cdot y + x$  (times successor)

Line 3 is an axiom schema.

Example:  $3x + 5 = 2y$  can be written using  $\Sigma_{PA}$  as

$$x + x + x + 1 + 1 + 1 + 1 + 1 = y + y$$

Navigation icons

Navigation icons

We have  $>$  and  $\geq$  since

$$\begin{aligned} 3x + 5 > 2y & \text{ write as } \exists z. z \neq 0 \wedge 3x + 5 = 2y + z \\ 3x + 5 \geq 2y & \text{ write as } \exists z. 3x + 5 = 2y + z \end{aligned}$$

Example:

- ▶ Pythagorean Theorem is  $T_{PA}$ -valid  
 $\exists x, y, z. x \neq 0 \wedge y \neq 0 \wedge z \neq 0 \wedge xx + yy = zz$
- ▶ Fermat's Last Theorem is  $T_{PA}$ -invalid (Andrew Wiles, 1994)  
 $\exists n. n > 2 \rightarrow \exists x, y, z. x \neq 0 \wedge y \neq 0 \wedge z \neq 0 \wedge x^n + y^n = z^n$

Remark (Gödel's first incompleteness theorem)

Peano arithmetic  $T_{PA}$  does not capture true arithmetic:

There exist closed  $\Sigma_{PA}$ -formulae representing valid propositions of number theory that are not  $T_{PA}$ -valid.

The reason:  $T_{PA}$  actually admits nonstandard interpretations

Satisfiability and validity in  $T_{PA}$  is undecidable.  
 Restricted theory – no multiplication

Navigation icons

Navigation icons

### 2. Presburger Arithmetic $T_{\mathbb{N}}$

$\Sigma_{\mathbb{N}} : \{0, 1, +, =\}$  no multiplication!

Axioms  $T_{\mathbb{N}}$ :

- $\forall x. \neg(x + 1 = 0)$  (zero)
- $\forall x, y. x + 1 = y + 1 \rightarrow x = y$  (successor)
- $F[0] \wedge (\forall x. F[x] \rightarrow F[x + 1]) \rightarrow \forall x. F[x]$  (induction)
- $\forall x. x + 0 = x$  (plus zero)
- $\forall x, y. x + (y + 1) = (x + y) + 1$  (plus successor)

3 is an axiom schema.

$T_{\mathbb{N}}$ -satisfiability and  $T_{\mathbb{N}}$ -validity are decidable  
 (Presburger, 1929)



## 1. Theory of Reals $T_{\mathbb{R}}$

$$\Sigma_{\mathbb{R}} : \{0, 1, +, -, \cdot, =, \geq\}$$

with multiplication.

Axioms in text.

Example:

$$\forall a, b, c. b^2 - 4ac \geq 0 \leftrightarrow \exists x. ax^2 + bx + c = 0$$

is  $T_{\mathbb{R}}$ -valid.

$T_{\mathbb{R}}$  is decidable (Tarski, 1930)  
High time complexity

## 2. Theory of Rationals $T_{\mathbb{Q}}$

$$\Sigma_{\mathbb{Q}} : \{0, 1, +, -, =, \geq\}$$

without multiplication.

Axioms in text.

Rational coefficients are simple to express in  $T_{\mathbb{Q}}$

Example: Rewrite

$$\frac{1}{2}x + \frac{2}{3}y \geq 4$$

as the  $\Sigma_{\mathbb{Q}}$ -formula

$$3x + 4y \geq 24$$

$T_{\mathbb{Q}}$  is decidable  
Quantifier-free fragment of  $T_{\mathbb{Q}}$  is efficiently decidable

## Recursive Data Structures (RDS)

### 1. RDS theory of LISP-like lists, $T_{\text{cons}}$

$$\Sigma_{\text{cons}} : \{\text{cons}, \text{car}, \text{cdr}, \text{atom}, =\}$$

where

$\text{cons}(a, b)$  – list constructed by concatenating  $a$  and  $b$

$\text{car}(x)$  – left projector of  $x$ :  $\text{car}(\text{cons}(a, b)) = a$

$\text{cdr}(x)$  – right projector of  $x$ :  $\text{cdr}(\text{cons}(a, b)) = b$

$\text{atom}(x)$  – true iff  $x$  is a single-element list

Axioms:

1. The axioms of reflexivity, symmetry, and transitivity of =
2. Congruence axioms

$$\forall x_1, x_2, y_1, y_2. x_1 = x_2 \wedge y_1 = y_2 \rightarrow \text{cons}(x_1, y_1) = \text{cons}(x_2, y_2)$$

$$\forall x, y. x = y \rightarrow \text{car}(x) = \text{car}(y)$$

$$\forall x, y. x = y \rightarrow \text{cdr}(x) = \text{cdr}(y)$$

### 3. Equivalence axiom

$$\forall x, y. x = y \rightarrow (\text{atom}(x) \leftrightarrow \text{atom}(y))$$

4.  $\forall x, y. \text{car}(\text{cons}(x, y)) = x$  (left projection)
5.  $\forall x, y. \text{cdr}(\text{cons}(x, y)) = y$  (right projection)
6.  $\forall x. \neg \text{atom}(x) \rightarrow \text{cons}(\text{car}(x), \text{cdr}(x)) = x$  (construction)
7.  $\forall x, y. \neg \text{atom}(\text{cons}(x, y))$  (atom)

$T_{\text{cons}}$  is undecidable  
Quantifier-free fragment of  $T_{\text{cons}}$  is efficiently decidable

## 2. Lists + equality

$$T_{\text{cons}}^{\text{=}} = T_{=} \cup T_{\text{cons}}$$

Signature:  $\Sigma_{=} \cup \Sigma_{\text{cons}}$

(this includes uninterpreted constants, functions, and predicates)

Axioms: union of the axioms of  $T_{=}$  and  $T_{\text{cons}}$

$T_{\text{cons}}^{\text{=}}$  is undecidable  
Quantifier-free fragment of  $T_{\text{cons}}^{\text{=}}$  is efficiently decidable

Example: We argue that the  $\Sigma_{\text{cons}}^{\text{=}}$ -formula

$$F: \begin{aligned} &\text{car}(a) = \text{car}(b) \wedge \text{cdr}(a) = \text{cdr}(b) \wedge \neg \text{atom}(a) \wedge \neg \text{atom}(b) \\ &\quad \rightarrow f(a) = f(b) \end{aligned}$$

is  $T_{\text{cons}}^{\text{=}}$ -valid.

Suppose not; then there exists a  $T_{\text{cons}}^{\text{=}}$ -interpretation  $I$  such that  $I \not\models F$ . Then,

- |  |  |                             |
|--|--|-----------------------------|
| 1. $I \not\models F$   |  | assumption                  |
| 2. $I \models \text{car}(a) = \text{car}(b)$   |  | 1, $\rightarrow$ , $\wedge$ |
| 3. $I \models \text{cdr}(a) = \text{cdr}(b)$   |  | 1, $\rightarrow$ , $\wedge$ |
| 4. $I \models \neg \text{atom}(a)$   |  | 1, $\rightarrow$ , $\wedge$ |
| 5. $I \models \neg \text{atom}(b)$   |  | 1, $\rightarrow$ , $\wedge$ |
| 6. $I \not\models f(a) = f(b)$   |  | 1, $\rightarrow$            |
| 7. $I \models \text{cons}(\text{car}(a), \text{cdr}(a)) = \text{cons}(\text{car}(b), \text{cdr}(b))$ |  | 2, 3, (congruence)          |
| 8. $I \models \text{cons}(\text{car}(a), \text{cdr}(a)) = a$   |  | 4, (construction)           |
| 9. $I \models \text{cons}(\text{car}(b), \text{cdr}(b)) = b$   |  | 5, (construction)           |
| 10. $I \models a = b$  |  | 7, 8, 9, (transitivity)     |
| 11. $I \models f(a) = f(b)$  |  | 10, (congruence)            |

Lines 6 and 11 are contradictory, so our assumption that  $I \not\models F$  must be wrong. Therefore,  $F$  is  $T_{\text{cons}}^{\text{=}}$ -valid.

## Theory of Arrays

### 1. Theory of Arrays $T_A$

Signature

$$\Sigma_A: \{ \cdot[\cdot], \cdot \langle \cdot \rangle, = \}$$

where

- ▶  $a[i]$  binary function – read array  $a$  at index  $i$  (“read( $a, i$ )”)
- ▶  $a \langle i \rangle v$  ternary function – write value  $v$  to index  $i$  of array  $a$  (“write( $a, i, v$ )”)

Axioms

1. the axioms of (reflexivity), (symmetry), and (transitivity) of  $T_{=}$
2.  $\forall a, i, j. i = j \rightarrow a[i] = a[j]$  (array congruence)
3.  $\forall a, v, i, j. i = j \rightarrow a \langle i \rangle v [j] = v$  (read-over-write 1)
4.  $\forall a, v, i, j. i \neq j \rightarrow a \langle i \rangle v [j] = a[j]$  (read-over-write 2)

Note:  $=$  is only defined for array elements

$$F: a[i] = e \rightarrow a \langle i \rangle e = a$$

not  $T_A$ -valid, but

$$F': a[i] = e \rightarrow \forall j. a \langle i \rangle e [j] = a[j],$$

is  $T_A$ -valid.

$T_A$  is undecidable  
Quantifier-free fragment of  $T_A$  is decidable

## 2. Theory of Arrays $T_A^=$ (with extensionality)

Signature and axioms of  $T_A^=$  are the same as  $T_A$ , with one additional axiom

$$\forall a, b. (\forall i. a[i] = b[i]) \leftrightarrow a = b \quad (\text{extensionality})$$

Example:

$$F : a[i] = e \rightarrow a(i \triangleleft e) = a$$

is  $T_A^=$ -valid.

$T_A^=$  is undecidable  
Quantifier-free fragment of  $T_A^=$  is decidable

◀ ▶ ↺ ↻ 🔍

qff = quantifier-free fragment

Nelson & Oppen showed that

if satisfiability of qff of  $T_1$  is decidable,  
satisfiability of qff of  $T_2$  is decidable, and  
certain technical simple requirements are met  
then satisfiability of qff of  $T_1 \cup T_2$  is decidable.

◀ ▶ ↺ ↻ 🔍

## Combination of Theories

How do we show that

$$1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1) \wedge f(x) \neq f(2)$$

is  $(T_1 \cup T_2)$ -unsatisfiable?

Or how do we prove properties about  
an array of integers, or  
a list of reals ... ?

Given theories  $T_1$  and  $T_2$  such that

$$\Sigma_1 \cap \Sigma_2 = \{=\}$$

The combined theory  $T_1 \cup T_2$  has

- ▶ signature  $\Sigma_1 \cup \Sigma_2$
- ▶ axioms  $A_1 \cup A_2$

◀ ▶ ↺ ↻ 🔍

## 4. Induction

◀ ▶ ↺ ↻ 🔍

## Induction

- ▶ Stepwise induction (for  $T_{PA}$ ,  $T_{cons}$ )
- ▶ Complete induction (for  $T_{PA}$ ,  $T_{cons}$ )  
Theoretically equivalent in power to stepwise induction, but sometimes produces more concise proof
- ▶ Well-founded induction  
Generalized complete induction
- ▶ Structural induction  
Over logical formulae

◀ ▶ ↻ 🔍

### Example:

Theory  $T_{PA}^+$  obtained from  $T_{PA}$  by adding the axioms:

- ▶  $\forall x. x^0 = 1$  (E0)
- ▶  $\forall x, y. x^{y+1} = x^y \cdot x$  (E1)
- ▶  $\forall x, z. \text{exp}_3(x, 0, z) = z$  (P0)
- ▶  $\forall x, y, z. \text{exp}_3(x, y + 1, z) = \text{exp}_3(x, y, x \cdot z)$  (P1)

Prove that

$$\boxed{\forall x, y. \text{exp}_3(x, y, 1) = x^y}$$

is  $T_{PA}^+$ -valid.

◀ ▶ ↻ 🔍

## Stepwise Induction (Peano Arithmetic $T_{PA}$ )

### Axiom schema (induction)

$F[0] \wedge$  ... base case  
 $(\forall n. F[n] \rightarrow F[n+1])$  ... inductive step  
 $\rightarrow \forall x. F[x]$  ... conclusion  
for  $\Sigma_{PA}$ -formulae  $F[x]$  with one free variable  $x$ .

To prove  $\forall x. F[x]$ , i.e.,

$F[x]$  is  $T_{PA}$ -valid for all  $x \in \mathbb{N}$ ,

it suffices to show

- ▶ base case: prove  $F[0]$  is  $T_{PA}$ -valid.
- ▶ inductive step: For arbitrary  $n \in \mathbb{N}$ ,  
assume inductive hypothesis, i.e.,  
 $F[n]$  is  $T_{PA}$ -valid,  
then prove the conclusion  
 $F[n+1]$  is  $T_{PA}$ -valid.

◀ ▶ ↻ 🔍

### First attempt:

$$\forall y \underbrace{[\forall x. \text{exp}_3(x, y, 1) = x^y]}_{F[y]}$$

We chose induction on  $y$ . Why?

### Base case:

$$F[0] : \forall x. \text{exp}_3(x, 0, 1) = x^0$$

OK since  $\text{exp}_3(x, 0, 1) = 1$  (P0) and  $x^0 = 1$  (E0).

### Inductive step: Failure.

For arbitrary  $n \in \mathbb{N}$ , we cannot deduce

$$F[n+1] : \forall x. \text{exp}_3(x, n+1, 1) = x^{n+1}$$

from the inductive hypothesis

$$F[n] : \forall x. \text{exp}_3(x, n, 1) = x^n$$

◀ ▶ ↻ 🔍

## Second attempt: Strengthening

### Strengthened property

$$\boxed{\forall x, y, z. \text{exp}_3(x, y, z) = x^y \cdot z}$$

Implies the desired property (choose  $z = 1$ )

$$\forall x, y. \text{exp}_3(x, y, 1) = x^y$$

Again, induction on  $y$

$$\forall y \underbrace{[\forall x, z. \text{exp}_3(x, y, z) = x^y \cdot z]}_{F[y]}$$

### Base case:

$$F[0] : \forall x, z. \text{exp}_3(x, 0, z) = x^0 \cdot z$$

OK since  $\text{exp}_3(x, 0, z) = z$  (P0) and  $x^0 = 1$  (E0).



## Inductive step: For arbitrary $n \in \mathbb{N}$

Assume inductive hypothesis

$$F[n] : \forall x, z. \text{exp}_3(x, n, z) = x^n \cdot z \quad (\text{IH})$$

prove

$$F[n+1] : \forall x, z'. \text{exp}_3(x, n+1, z') = x^{n+1} \cdot z'$$

$$\text{exp}_3(x, n+1, z') = \text{exp}_3(x, n, x \cdot z') \quad (\text{P1})$$

$$= x^n \cdot (x \cdot z') \quad \text{IH } F[n], z \mapsto x \cdot z'$$

$$= x^{n+1} \cdot z' \quad (\text{E1})$$

## Stepwise Induction (Lists $T_{\text{cons}}$ )

### Axiom schema (induction)

$$\begin{array}{ll} (\forall \text{atom } u. F[u] \wedge) & \dots \text{ base case} \\ (\forall u, v. F[v] \rightarrow F[\text{cons}(u, v)]) & \dots \text{ inductive step} \\ \rightarrow \forall x. F[x] & \dots \text{ conclusion} \end{array}$$

for  $\Sigma_{\text{cons}}$ -formulae  $F[x]$  with one free variable  $x$ .

To prove  $\forall x. F[x]$ , i.e.,

$F[x]$  is  $T_{\text{cons}}$ -valid for all lists  $x$ ,

it suffices to show

- ▶ **base case:** prove  $F[u]$  is  $T_{\text{cons}}$ -valid for arbitrary atom  $u$ .
- ▶ **inductive step:** For arbitrary list  $v$ , assume inductive hypothesis, i.e.,  $F[v]$  is  $T_{\text{cons}}$ -valid, then prove the **conclusion**  $F[\text{cons}(u, v)]$  is  $T_{\text{cons}}$ -valid for arbitrary atom  $u$ .



### Example

Theory  $T_{\text{cons}}^+$  obtained from  $T_{\text{cons}}$  by adding the axioms for concatenating two lists, reverse a list, and decide if a list is flat (i.e.,  $\text{flat}(x)$  is  $\top$  iff every element of list  $x$  is an atom).

- ▶  $\forall \text{atom } u. \forall v. \text{concat}(u, v) = \text{cons}(u, v)$  (C0)
- ▶  $\forall u, v, x. \text{concat}(\text{cons}(u, v), x) = \text{cons}(u, \text{concat}(v, x))$  (C1)
- ▶  $\forall \text{atom } u. \text{rvs}(u) = u$  (R0)
- ▶  $\forall x, y. \text{rvs}(\text{concat}(x, y)) = \text{concat}(\text{rvs}(y), \text{rvs}(x))$  (R1)
- ▶  $\forall \text{atom } u. \text{flat}(u)$  (F0)
- ▶  $\forall u, v. \text{flat}(\text{cons}(u, v)) \leftrightarrow \text{atom}(u) \wedge \text{flat}(v)$  (F1)

Prove

$$\boxed{\forall x. \text{flat}(x) \rightarrow \text{rvs}(\text{rvs}(x)) = x}$$

is  $T_{\text{cons}}^+$ -valid.

**Base case:** For arbitrary atom  $u$ ,

$$F[u] : \text{flat}(u) \rightarrow \text{rvs}(\text{rvs}(u)) = u$$

by R0.



Inductive step: For arbitrary lists  $u, v$ ,  
assume the inductive hypothesis

$$F[v] : \text{flat}(v) \rightarrow \text{rvs}(\text{rvs}(v)) = v \quad (\text{IH})$$

Prove

$$F[\text{cons}(u, v)] : \text{flat}(\text{cons}(u, v)) \rightarrow \text{rvs}(\text{rvs}(\text{cons}(u, v))) = \text{cons}(u, v) \quad (*)$$

Case  $\neg \text{atom}(u)$

$$\text{flat}(\text{cons}(u, v)) \Leftrightarrow \text{atom}(u) \wedge \text{flat}(v) \Leftrightarrow \perp$$

by (F1). (\*) holds since its antecedent is  $\perp$ .

Case  $\text{atom}(u)$

$$\text{flat}(\text{cons}(u, v)) \Leftrightarrow \text{atom}(u) \wedge \text{flat}(v) \Leftrightarrow \text{flat}(v)$$

by (F1).

$$\text{rvs}(\text{rvs}(\text{cons}(u, v))) = \dots = \text{cons}(u, v).$$



Is base case missing?

No. Base case is implicit in the structure of complete induction.

Note:

- ▶ Complete induction is theoretically equivalent in power to stepwise induction.
- ▶ Complete induction sometimes yields more concise proofs.

Example: Integer division  $\text{quot}(5, 3) = 1$  and  $\text{rem}(5, 3) = 2$

Theory  $T_{PA}^*$  obtained from  $T_{PA}$  by adding the axioms:

- ▶  $\forall x, y. x < y \rightarrow \text{quot}(x, y) = 0$  (Q0)
- ▶  $\forall x, y. y > 0 \rightarrow \text{quot}(x + y, y) = \text{quot}(x, y) + 1$  (Q1)
- ▶  $\forall x, y. x < y \rightarrow \text{rem}(x, y) = x$  (R0)
- ▶  $\forall x, y. y > 0 \rightarrow \text{rem}(x + y, y) = \text{rem}(x, y)$  (R1)

Prove

- (1)  $\forall x, y. y > 0 \rightarrow \text{rem}(x, y) < y$
- (2)  $\forall x, y. y > 0 \rightarrow x = y \cdot \text{quot}(x, y) + \text{rem}(x, y)$

Best proved by complete induction.



## Complete Induction (Peano Arithmetic $T_{PA}$ )

Axiom schema (complete induction)

$$(\forall n. (\forall n'. n' < n \rightarrow F[n']) \rightarrow F[n]) \quad \dots \text{ inductive step}$$

$$\rightarrow \forall x. F[x] \quad \dots \text{ conclusion}$$

for  $\Sigma_{PA}$ -formulae  $F[x]$  with one free variable  $x$ .

To prove  $\forall x. F[x]$ , i.e.,

$F[x]$  is  $T_{PA}$ -valid for all  $x \in \mathbb{N}$ ,  
it suffices to show

- ▶ inductive step: For arbitrary  $n \in \mathbb{N}$ ,  
assume inductive hypothesis, i.e.,  
 $F[n']$  is  $T_{PA}$ -valid for every  $n' \in \mathbb{N}$  such that  $n' < n$ ,  
then prove  
 $F[n]$  is  $T_{PA}$ -valid.

Proof of (1)

$$\forall x. \underbrace{\forall y. y > 0 \rightarrow \text{rem}(x, y) < y}_{F[x]}$$

Consider an arbitrary natural number  $x$ .

Assume the inductive hypothesis

$$\forall x'. x' < x \rightarrow \underbrace{\forall y'. y' > 0 \rightarrow \text{rem}(x', y') < y'}_{F[x']} \quad (\text{IH})$$

Prove  $F[x] : \forall y. y > 0 \rightarrow \text{rem}(x, y) < y$ .

Let  $y$  be an arbitrary positive integer

Case  $x < y$ :

$$\text{rem}(x, y) = x \quad \text{by (R0)}$$

$$< y \quad \text{case}$$

Case  $\neg(x < y)$ :

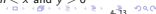
Then there is natural number  $n, n < x$  s.t.  $x = n + y$

$$\text{rem}(x, y) = \text{rem}(n + y, y) \quad x = n + y$$

$$= \text{rem}(n, y) \quad (\text{R1})$$

$$< y \quad \text{IH } (x' \mapsto n, y' \mapsto y)$$

since  $n < x$  and  $y > 0$



## Well-founded Induction

A binary predicate  $<$  over a set  $S$  is a well-founded relation iff there does not exist an infinite decreasing sequence

$$s_1 \succ s_2 \succ s_3 \succ \dots$$

Note: where  $s < t$  iff  $t \succ s$

Examples:

- $<$  is well-founded over the natural numbers.  
Any sequence of natural numbers decreasing according to  $<$  is finite:  
 $1023 > 39 > 30 > 29 > 8 > 3 > 0$ .
- $<$  is not well-founded over the rationals.  
 $1 > \frac{1}{2} > \frac{1}{3} > \frac{1}{4} > \dots$   
is an infinite decreasing sequence.
- The strict sublist relation  $<_c$  is well-founded on the set of all lists.



## Lexicographic Relation

Given pairs of sets and well-founded relations

$$(S_1, <_1), \dots, (S_m, <_m)$$

Construct

$$S = S_1 \times \dots \times S_m$$

Define lexicographic relation  $<$  over  $S$  as

$$\underbrace{(s_1, \dots, s_m)}_s < \underbrace{(t_1, \dots, t_m)}_t \Leftrightarrow \bigvee_{i=1}^m \left( s_i <_i t_i \wedge \bigwedge_{j=1}^{i-1} s_j = t_j \right)$$

for  $s_i, t_i \in S_i$ .

- If  $(S_1, <_1), \dots, (S_m, <_m)$  are well-founded relations, so is  $(S, <)$ .



## Well-founded Induction Principle

For theory  $T$  and well-founded relation  $<$ ,  
the axiom schema (well-founded induction)

$$(\forall n. (\forall n'. n' < n \rightarrow F[n']) \rightarrow F[n]) \rightarrow \forall x. F[x]$$

for  $\Sigma$ -formulae  $F[x]$  with one free variable  $x$ .

To prove  $\forall x. F[x]$ , i.e.,

$F[x]$  is  $T$ -valid for every  $x$ ,  
it suffices to show

- inductive step: For arbitrary  $n$ ,  
assume inductive hypothesis, i.e.,  
 $F[n']$  is  $T$ -valid for every  $n'$ , such that  $n' < n$   
then prove  
 $F[n]$  is  $T$ -valid.

Complete induction in  $T_{PA}$  is a specific instance of well-founded induction, where the well-founded relation  $<$  is  $<_s$ .



## Lexicographic well-founded induction principle

For theory  $T$  and well-founded lexicographic relation  $<$ ,

$$\left[ \begin{array}{l} \forall n_1, \dots, n_m. \\ \left[ (\forall n'_1, \dots, n'_m. (n'_1, \dots, n'_m) < (n_1, \dots, n_m) \rightarrow F[n'_1, \dots, n'_m]) \right] \right. \\ \left. \rightarrow F[n_1, \dots, n_m] \right] \\ \rightarrow \forall x_1, \dots, x_m. F[x_1, \dots, x_m] \end{array} \right]$$

for  $\Sigma$ -formula  $F[x_1, \dots, x_m]$  with free variables  $x_1, \dots, x_m$ , is  $T$ -valid.

Same as regular well-founded induction, just

$$n \Rightarrow \text{tuple } (n_1, \dots, n_m).$$



### Example: Puzzle

Bag of red, yellow, and blue chips

If one chip remains in the bag – remove it

Otherwise, remove two chips at random:

1. If one of the two is red – don't put any chips in the bag
2. If both are yellow – put one yellow and five blue chips
3. If one of the two is blue and the other not red – put ten red chips

Does this process terminate?

Proof: Consider

- ▶ Set  $S : \mathbb{N}^3$  of triples of natural numbers and
- ▶ Well-founded lexicographic relation  $<_3$  for such triples, e.g.

$$(11, 13, 3) \not<_3 (11, 9, 104) \quad (11, 9, 104) <_3 (11, 13, 3)$$



### Example: Ackermann function

Theory  $T_{\mathbb{N}}^{ack}$  is the theory of Presburger arithmetic  $T_{\mathbb{N}}$  (for natural numbers) augmented with

Ackermann axioms:

- ▶  $\forall y. ack(0, y) = y + 1$  (L0)
- ▶  $\forall x. ack(x + 1, 0) = ack(x, 1)$  (R0)
- ▶  $\forall x, y. ack(x + 1, y + 1) = ack(x, ack(x + 1, y))$  (S)

Ackermann function grows quickly:

$$\begin{aligned} ack(0, 0) &= 1 \\ ack(1, 1) &= 3 \\ ack(2, 2) &= 7 \\ ack(3, 3) &= 61 \end{aligned} \quad ack(4, 4) = 2^{2^{2^{2^{16}}}} - 3$$



Show

$$(y', b', r') <_3 (y, b, r)$$

for each possible case. Since  $<_3$  well-formed relation

$\Rightarrow$  only finite decreasing sequences  $\Rightarrow$  process must terminate

1. If one of the two removed chips is red – do not put any chips in the bag

$$\left. \begin{aligned} (y - 1, b, r - 1) \\ (y, b - 1, r - 1) \\ (y, b, r - 2) \end{aligned} \right\} <_3 (y, b, r)$$

2. If both are yellow – put one yellow and five blue

$$(y - 1, b + 5, r) <_3 (y, b, r)$$

3. If one is blue and the other not red – put ten red

$$\left. \begin{aligned} (y - 1, b - 1, r + 10) \\ (y, b - 2, r + 10) \end{aligned} \right\} <_3 (y, b, r)$$



Let  $<_2$  be the lexicographic extension of  $<$  to pairs of natural numbers.

- (L0)  $\forall y. ack(0, y) = y + 1$   
does not involve recursive call
- (R0)  $\forall x. ack(x + 1, 0) = ack(x, 1)$   
 $(x + 1, 0) >_2 (x, 1)$
- (S)  $\forall x, y. ack(x + 1, y + 1) = ack(x, ack(x + 1, y))$   
 $(x + 1, y + 1) >_2 (x + 1, y)$   
 $(x + 1, y + 1) >_2 (x, ack(x + 1, y))$

No infinite recursive calls  $\Rightarrow$  the recursive computation of  $ack(x, y)$  terminates for all pairs of natural numbers.



### Proof of property

Use well-founded induction over  $<_2$  to prove

$$\forall x, y. \text{ack}(x, y) > y$$

is  $T_{\mathbb{N}}^{\text{ack}}$  valid.

Consider arbitrary natural numbers  $x, y$ .

Assume the inductive hypothesis

$$\forall x', y'. (x', y') <_2 (x, y) \rightarrow \underbrace{\text{ack}(x', y') > y'}_{F[x', y']} \quad (\text{IH})$$

Show

$$F[x, y] : \text{ack}(x, y) > y.$$

Case  $x = 0$ :

$$\text{ack}(0, y) = y + 1 > y \quad \text{by (L0)}$$

Furthermore, since

$$\underbrace{(x, y-1)}_{<_2 (x, y)}$$

then

$$\text{ack}(x, y-1) > y-1 \quad (3)$$

By (1)–(3), we have

$$\text{ack}(x, y) \stackrel{(1)}{=} \text{ack}(x-1, \text{ack}(x, y-1)) \stackrel{(2)}{>} \text{ack}(x, y-1) \stackrel{(3)}{>} y-1$$

Hence

$$\text{ack}(x, y) > (y-1) + 1 = y$$

Case  $x > 0 \wedge y = 0$ :

$$\text{ack}(x, 0) = \text{ack}(x-1, 1) \quad \text{by (R0)}$$

Since

$$\underbrace{(x-1, 1)}_{<_2 (x, y)}$$

Then

$$\text{ack}(x-1, 1) > 1 \quad \text{by (IH) } (x' \mapsto x-1, y' \mapsto 1)$$

Thus

$$\text{ack}(x, 0) = \text{ack}(x-1, 1) > 1 > 0$$

Case  $x > 0 \wedge y > 0$ :

$$\text{ack}(x, y) = \text{ack}(x-1, \text{ack}(x, y-1)) \quad \text{by (S)} \quad (1)$$

Since

$$\underbrace{(x-1, \text{ack}(x, y-1))}_{<_2 (x, y)}$$

Then

$$\text{ack}(x-1, \text{ack}(x, y-1)) > \text{ack}(x, y-1) \quad (2)$$

$$\text{by (IH) } (x' \mapsto x-1, y' \mapsto \text{ack}(x, y-1)).$$

## Structural Induction

How do we prove properties about logical formulae themselves?

### Structural induction principle

To prove a desired property of FOL formulae,

inductive step: Assume the inductive hypothesis, that for arbitrary FOL formula  $F$ , the desired property holds for every strict subformula  $G$  of  $F$ .

Then prove that  $F$  has the property.

Since atoms do not have strict subformulae, they are treated as base cases.

Example: Prove that

Every propositional formula  $F$  is equivalent to a propositional formula  $F'$  constructed with only  $\top$ ,  $\vee$ ,  $\neg$  (and propositional variables)

Base cases:

$$F : \top \Rightarrow F' : \top$$

$$F : \perp \Rightarrow F' : \neg\top$$

$$F : P \Rightarrow F' : P \text{ for propositional variable } P$$

Inductive step:

Assume as the inductive hypothesis that  $G$ ,  $G_1$ ,  $G_2$  are equivalent to  $G'$ ,  $G'_1$ ,  $G'_2$  constructed only from  $\top$ ,  $\vee$ ,  $\neg$  (and propositional variables).

$$F : \neg G \Rightarrow F' : \neg G'$$

$$F : G_1 \wedge G_2 \Rightarrow F' : \neg(\neg G'_1 \vee \neg G'_2)$$

$$F : G_1 \rightarrow G_2 \Rightarrow F' : \neg G'_1 \vee G'_2$$

$$F : G_1 \leftrightarrow G_2 \Rightarrow F' : \dots$$

Each  $F'$  is equivalent to  $F$  and is constructed only by  $\top$ ,  $\vee$ ,  $\neg$  by the inductive hypothesis.

Program A: LinearSearch with function specification

---

```
@pre 0 ≤ ℓ ∧ u < |a|
@post rv ↔ ∃i. ℓ ≤ i ≤ u ∧ a[i] = e
bool LinearSearch(int[] a, int ℓ, int u, int e) {
  for @ T
    (int i := ℓ; i ≤ u; i := i + 1) {
      if (a[i] = e) return true;
    }
  return false;
}
```

---

## 5. Program Correctness: Mechanics

Function LinearSearch searches subarray of array  $a$  of integers for specified value  $e$ .

Function specifications

- ▶ Function postcondition (@post)  
It returns true iff  $a$  contains the value  $e$  in the range  $[\ell, u]$
- ▶ Function precondition (@pre)  
It behaves correctly only if  $0 \leq \ell$  and  $u < |a|$

for loop: initially set  $i$  to be  $\ell$ ,  
execute the body and increment  $i$  by 1  
as long as  $i \leq n$

@ - program annotation

Program B: BinarySearch with function specification

---

```
@pre  $0 \leq \ell \wedge u < |a| \wedge \text{sorted}(a, \ell, u)$ 
@post  $rv \leftrightarrow \exists i. \ell \leq i \leq u \wedge a[i] = e$ 
bool BinarySearch(int[] a, int  $\ell$ , int  $u$ , int  $e$ ) {
  if ( $\ell > u$ ) return false;
  else {
    int  $m := (\ell + u) \text{ div } 2$ ;
    if ( $a[m] = e$ ) return true;
    else if ( $a[m] < e$ ) return BinarySearch( $a, m + 1, u, e$ );
    else return BinarySearch( $a, \ell, m - 1, e$ );
  }
}
```

---

The recursive function BinarySearch searches subarray of sorted array  $a$  of integers for specified value  $e$ .

sorted: weakly increasing order, i.e.

$$\text{sorted}(a, \ell, u) \Leftrightarrow \forall i, j. \ell \leq i \leq j \leq u \rightarrow a[i] \leq a[j]$$

Defined in the combined theory of integers and arrays,  $T_{\mathbb{Z}, A}$

Function specifications

- ▶ Function postcondition (@post)  
It returns true iff  $a$  contains the value  $e$  in the range  $[\ell, u]$
- ▶ Function precondition (@pre)  
It behaves correctly only if  $0 \leq \ell$  and  $u < |a|$

Program C: BubbleSort with function specification

---

```
@pre  $\top$ 
@post  $\text{sorted}(rv, 0, |rv| - 1)$ 
int[] BubbleSort(int[]  $a_0$ ) {
  int[]  $a := a_0$ ;
  for @  $\top$ 
    (int  $i := |a| - 1; i > 0; i := i - 1$ ) {
      for @  $\top$ 
        (int  $j := 0; j < i; j := j + 1$ ) {
          if ( $a[j] > a[j + 1]$ ) {
            int  $t := a[j]$ ;
             $a[j] := a[j + 1]$ ;
             $a[j + 1] := t$ ;
          }
        }
      }
  return  $a$ ;
}
```

---

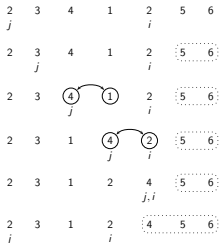
Function BubbleSort sorts integer array  $a$



by "bubbling" the largest element of the left unsorted region of  $a$  toward the sorted region on the right.

Each iteration of the outer loop expands the sorted region by one cell.

## Sample execution of BubbleSort



## Program A: LinearSearch with runtime assertions

```

@pre T
@post T
bool LinearSearch(int[] a, int ℓ, int u, int e) {
  for @ T
    (int i := ℓ; i ≤ u; i := i + 1) {
      @ 0 ≤ i < |a|;
      if (a[i] = e) return true;
    }
  return false;
}

```



## Program Annotation

### ▶ Function Specifications

function postcondition (@post)

function precondition (@pre)

### ▶ Runtime Assertions

e.g., @  $0 \leq j < |a| \wedge 0 \leq j + 1 < |a|$

$a[j] := a[j + 1]$

### ▶ Loop Invariants

e.g., @  $L : \ell \leq i \wedge \forall j. \ell \leq j < i \rightarrow a[j] \neq e$



## Program B: BinarySearch with runtime assertions

```

@pre T
@post T
bool BinarySearch(int[] a, int ℓ, int u, int e) {
  if (ℓ > u) return false;
  else {
    @ 2 ≠ 0;
    int m := (ℓ + u) div 2;
    @ 0 ≤ m < |a|;
    if (a[m] = e) return true;
    else {
      @ 0 ≤ m < |a|;
      if (a[m] < e) return BinarySearch(a, m + 1, u, e);
      else return BinarySearch(a, ℓ, m - 1, e);
    }
  }
}

```



### Program C: BubbleSort with runtime assertions

```
@pre T
@post T
int[] BubbleSort(int[] a0) {
  int[] a := a0;
  for @ T
    (int i := |a| - 1; i > 0; i := i - 1) {
      for @ T
        (int j := 0; j < i; j = j + 1) {
          @ 0 ≤ j < |a| ∧ 0 ≤ j + 1 < |a|;
          if (a[j] > a[j + 1]) {
            int t := a[j];
            a[j] := a[j + 1];
            a[j + 1] := t;
          }
        }
      }
  return a;
}
```

◀ ▶ ↺ ↻ 🔍 ↻

### Program A: LinearSearch with loop invariants

```
@pre 0 ≤ ℓ ∧ u < |a|
@post rv ↔ ∃i. ℓ ≤ i ≤ u ∧ a[i] = e
bool LinearSearch(int[] a, int ℓ, int u, int e) {
  for
    @L: ℓ ≤ i ∧ ∀j. ℓ ≤ j < i → a[j] ≠ e
    (int i := ℓ; i ≤ u; i := i + 1) {
      if (a[i] = e) return true;
    }
  return false;
}
```

◀ ▶ ↺ ↻ 🔍 ↻

### Loop Invariants

```
while
  @ F
  ⟨cond⟩ { ⟨body⟩ }
```

- ▶ apply  $\langle body \rangle$  as long as  $\langle cond \rangle$  holds
- ▶ assertion  $F$  holds at the beginning of every iteration evaluated before  $\langle cond \rangle$  is checked

```
for
  @ F
  ((init); ⟨cond⟩; ⟨incr⟩) { ⟨body⟩ }
```

⇒

```
⟨init⟩;
while
  @ F
  ⟨cond⟩ { ⟨body⟩ ⟨incr⟩ }
```

◀ ▶ ↺ ↻ 🔍 ↻

### Proving Partial Correctness

A function is partially correct if when the function's precondition is satisfied on entry, its postcondition is satisfied when the function halts.

- ▶ A function + annotation is reduced to finite set of verification conditions (VCs), FOL formulae
- ▶ If all VCs are valid, then the function obeys its specification (partially correct)

◀ ▶ ↺ ↻ 🔍 ↻

## Basic Paths: Loops

To handle loops, we break the function into basic paths

@ ← precondition or loop invariant

sequence of instructions  
(with no loop invariants)

@ ← loop invariant, assertion, or postcondition

## Program A: LinearSearch

### Basic Paths of LinearSearch

---

(1) \_\_\_\_\_  
@pre  $0 \leq \ell \wedge u < |a|$   
 $i := \ell;$   
@L:  $\ell \leq i \wedge \forall j. \ell \leq j < i \rightarrow a[j] \neq e$

---

(2) \_\_\_\_\_  
@L:  $\ell \leq i \wedge \forall j. \ell \leq j < i \rightarrow a[j] \neq e$   
assume  $i \leq u;$   
assume  $a[i] = e;$   
 $rv := \text{true};$   
@post  $rv \leftrightarrow \exists j. \ell \leq j \leq u \wedge a[j] = e$

---

◀ ▶ ⏪ ⏩ ↺ ↻ 🔍

◀ ▶ ⏪ ⏩ ↺ ↻ 🔍

---

(3) \_\_\_\_\_  
@L:  $\ell \leq i \wedge \forall j. \ell \leq j < i \rightarrow a[j] \neq e$   
assume  $i \leq u;$   
assume  $a[i] \neq e;$   
 $i := i + 1;$   
@L:  $\ell \leq i \wedge \forall j. \ell \leq j < i \rightarrow a[j] \neq e$

---

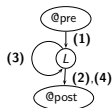
(4) \_\_\_\_\_  
@L:  $\ell \leq i \wedge \forall j. \ell \leq j < i \rightarrow a[j] \neq e$   
assume  $i > u;$   
 $rv := \text{false};$   
@post  $rv \leftrightarrow \exists j. \ell \leq j \leq u \wedge a[j] = e$

---

◀ ▶ ⏪ ⏩ ↺ ↻ 🔍

◀ ▶ ⏪ ⏩ ↺ ↻ 🔍

Visualization of basic paths of LinearSearch



```

@pre T
@post sorted(rv, 0, |rv| - 1)
int[] BubbleSort(int[] a0) {
  int[] a := a0;
  for
    @L1 : [ -1 ≤ i < |a|
           ∧ partitioned(a, 0, i, i + 1, |a| - 1) ]
    (int i := |a| - 1; i > 0; i := i - 1) {

```

```

for
  @L2 : [ 1 ≤ i < |a| ∧ 0 ≤ j ≤ i
         ∧ partitioned(a, 0, i, i + 1, |a| - 1)
         ∧ partitioned(a, 0, j - 1, j, j)
         ∧ sorted(a, i, |a| - 1) ]
    (int j := 0; j < i; j := j + 1) {
      if (a[j] > a[j + 1]) {
        int t := a[j];
        a[j] := a[j + 1];
        a[j + 1] := t;
      }
    }
  }
return a;
}

```

### Partition

$\text{partitioned}(a, \ell_1, u_1, \ell_2, u_2)$   
 $\Leftrightarrow \forall i, j. \ell_1 \leq i \leq u_1 < \ell_2 \leq j \leq u_2 \rightarrow a[i] \leq a[j]$

in  $T_{\mathbb{Z}} \cup T_{\mathcal{A}}$ .

That is, each element of  $a$  in the range  $[\ell_1, u_1]$  is  $\leq$  each element in the range  $[\ell_2, u_2]$ .

### Basic Paths of BubbleSort

(1)

```

@pre T;
a := a0;
i := |a| - 1;
@L1 : -1 ≤ i < |a| ∧ partitioned(a, 0, i, i + 1, |a| - 1)
      ∧ sorted(a, i, |a| - 1)

```

(2)

```

@L1 : -1 ≤ i < |a| ∧ partitioned(a, 0, i, i + 1, |a| - 1)
      ∧ sorted(a, i, |a| - 1)
assume i > 0;
j := 0;
@L2 : [ 1 ≤ i < |a| ∧ 0 ≤ j ≤ i ∧ partitioned(a, 0, i, i + 1, |a| - 1)
       ∧ partitioned(a, 0, j - 1, j, j) ∧ sorted(a, i, |a| - 1) ]

```

(3)

```

@L2 : [ 1 ≤ i < |a| ∧ 0 ≤ j ≤ i ∧ partitioned(a, 0, i, i + 1, |a| - 1)
       ∧ partitioned(a, 0, j - 1, j, j) ∧ sorted(a, i, |a| - 1) ]
assume j < i;
assume a[j] > a[j + 1];
t := a[j];
a[j] := a[j + 1];
a[j + 1] := t;
j := j + 1;
@L2 : [ 1 ≤ i < |a| ∧ 0 ≤ j ≤ i ∧ partitioned(a, 0, i, i + 1, |a| - 1)
       ∧ partitioned(a, 0, j - 1, j, j) ∧ sorted(a, i, |a| - 1) ]

```

---

**(4)**

---

```

@L2: [ 1 ≤ i < |a| ∧ 0 ≤ j ≤ i ∧ partitioned(a, 0, i, i + 1, |a| - 1)
      ∧ partitioned(a, 0, j - 1, j, j) ∧ sorted(a, i, |a| - 1) ]
assume j < i;
assume a[j] ≤ a[j + 1];
j := j + 1;
@L2: [ 1 ≤ i < |a| ∧ 0 ≤ j ≤ i ∧ partitioned(a, 0, i, i + 1, |a| - 1)
      ∧ partitioned(a, 0, j - 1, j, j) ∧ sorted(a, i, |a| - 1) ]

```

---



---

**(5)**

---

```

@L2: [ 1 ≤ i < |a| ∧ 0 ≤ j ≤ i ∧ partitioned(a, 0, i, i + 1, |a| - 1)
      ∧ partitioned(a, 0, j - 1, j, j) ∧ sorted(a, i, |a| - 1) ]
assume j ≥ i;
i := i - 1;
@L1: -1 ≤ i < |a| ∧ partitioned(a, 0, i, i + 1, |a| - 1)
      ∧ sorted(a, i, |a| - 1)

```

---

◀ ▶ ↻ 🔍

### Basic Paths: Function Calls

- ▶ Loops produce unbounded number of paths  
loop invariants cut loops to produce finite number of basic paths
- ▶ Reursive calls produce unbounded number of paths  
function specifications cut function calls

### In BinarySearch

```

@pre 0 ≤ ℓ ∧ u < |a| ∧ sorted(a, ℓ, u)           ... F[a, ℓ, u, e]
:
@R1: 0 ≤ m + 1 ∧ u < |a| ∧ sorted(a, m + 1, u) ... F[a, m + 1, u, e]
return BinarySearch(a, m + 1, u, e)
:
@R2: 0 ≤ ℓ ∧ m - 1 < |a| ∧ sorted(a, ℓ, m - 1) ... F[a, ℓ, m - 1, e]
return BinarySearch(a, ℓ, m - 1, e)

```

◀ ▶ ↻ 🔍

---

**(6)**

---

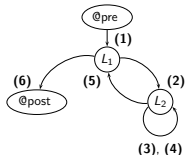
```

@L1: -1 ≤ i < |a| ∧ partitioned(a, 0, i, i + 1, |a| - 1) ∧
      sorted(a, i, |a| - 1)
assume i ≤ 0;
rv := a;
@post sorted(rv, 0, |rv| - 1)

```

---

### Visualization of basic paths of BubbleSort



◀ ▶ ↻ 🔍

### Program B: BinarySearch with function call assertions

---

```

@pre 0 ≤ ℓ ∧ u < |a| ∧ sorted(a, ℓ, u)
@post rv ↔ ∃ i. ℓ ≤ i ≤ u ∧ a[i] = e
bool BinarySearch(int[] a, int ℓ, int u, int e) {
  if (ℓ > u) return false;
  else {
    int m := (ℓ + u) div 2;
    if (a[m] = e) return true;
    else if (a[m] < e) {
      @R1: 0 ≤ m + 1 ∧ u < |a| ∧ sorted(a, m + 1, u);
      return BinarySearch(a, m + 1, u, e);
    } else {
      @R2: 0 ≤ ℓ ∧ m - 1 < |a| ∧ sorted(a, ℓ, m - 1);
      return BinarySearch(a, ℓ, m - 1, e);
    }
  }
}

```

---

◀ ▶ ↻ 🔍

## Verification Conditions

- ▶ **Program counter**  $pc$  — holds current location of control
- ▶ **State**  $s$  — assignment of values to all variables

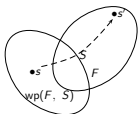
Example: Control resides at  $L_1$  of BubbleSort

$$s: \{pc \mapsto L_1, a \mapsto [2; 0; 1], i \mapsto 2, j \mapsto 0, \\ t \mapsto 2, rv \mapsto []\}$$

- ▶ **Weakest precondition**  $wp(F, S)$

For FOL formula  $F$ , program statement  $S$ ,

If  $s \models wp(F, S)$  and if statement  $S$  is executed on state  $s$  to produce state  $s'$ , then  $s' \models F$



◀ ▶ ↻ 🔍

Example: Basic path

(1)

$$\begin{aligned} @ F : x \geq 0 \\ S_1 : x := x + 1; \\ @ G : x \geq 1 \end{aligned}$$

The VC is

$$F \rightarrow wp(G, S_1)$$

That is,

$$\begin{aligned} wp(G, S_1) \\ \Leftrightarrow wp(x \geq 1, x := x + 1) \\ \Leftrightarrow (x \geq 1)\{x \mapsto x + 1\} \\ \Leftrightarrow x + 1 \geq 1 \\ \Leftrightarrow x \geq 0 \end{aligned}$$

Therefore the VC of path (1)

$$x \geq 0 \rightarrow x \geq 0,$$

which is  $T_Z$ -valid.

◀ ▶ ↻ 🔍

Weakest Precondition  $wp(F, S)$

- ▶  $wp(F, \text{assume } c) \Leftrightarrow c \rightarrow F$
- ▶  $wp(F[v, v := e]) \Leftrightarrow F[e]$
- ▶ For  $S_1; \dots; S_n$ ,  
 $wp(F, S_1; \dots; S_n) \Leftrightarrow wp(wp(F, S_n), S_1; \dots; S_{n-1})$

Verification Condition of basic path

@  $F$

$S_1$ ;

...

$S_n$ ;

@  $G$

is

$$F \rightarrow wp(G, S_1; \dots; S_n)$$

Also denoted by

$$\{F\} S_1; \dots; S_n \{G\}$$

◀ ▶ ↻ 🔍

Example: Basic path (2) of LinearSearch

(2)

$$\begin{aligned} @L : F : \ell \leq i \wedge \forall j. \ell \leq j < i \rightarrow a[j] \neq e \\ S_1 : \text{assume } i \leq u; \\ S_2 : \text{assume } a[i] = e; \\ S_3 : rv := \text{true}; \\ @\text{post } G : rv \rightarrow \exists j. \ell \leq j \leq u \wedge a[j] = e \end{aligned}$$

The VC is

$$F \rightarrow wp(G, S_1; S_2; S_3)$$

That is,

$$\begin{aligned} wp(G, S_1; S_2; S_3) \\ \Leftrightarrow wp(wp(rv \mapsto \exists j. \ell \leq j \leq u \wedge a[j] = e, rv := \text{true}), S_1; S_2) \\ \Leftrightarrow wp(\text{true} \mapsto \exists j. \ell \leq j \leq u \wedge a[j] = e, S_1; S_2) \\ \Leftrightarrow wp(\exists j. \ell \leq j \leq u \wedge a[j] = e, S_1; S_2) \\ \Leftrightarrow wp(wp(\exists j. \ell \leq j \leq u \wedge a[j] = e, \text{assume } a[i] = e), S_1) \\ \Leftrightarrow wp(a[i] = e \rightarrow \exists j. \ell \leq j \leq u \wedge a[j] = e, S_1) \\ \Leftrightarrow wp(a[i] = e \rightarrow \exists j. \ell \leq j \leq u \wedge a[j] = e, \text{assume } i \leq u) \\ \Leftrightarrow i \leq u \rightarrow (a[i] = e \rightarrow \exists j. \ell \leq j \leq u \wedge a[j] = e) \end{aligned}$$

◀ ▶ ↻ 🔍

Therefore the VC of path (2)

$$\begin{aligned} \ell \leq i \wedge (\forall j. \ell \leq j < i \rightarrow a[j] \neq e) \\ \rightarrow (i \leq u \rightarrow (a[i] = e \rightarrow \exists j. \ell \leq j \leq u \wedge a[j] = e)) \end{aligned} \quad (1)$$

or, equivalently,

$$\begin{aligned} \ell \leq i \wedge (\forall j. \ell \leq j < i \rightarrow a[j] \neq e) \wedge i \leq u \wedge a[i] = e \\ \rightarrow \exists j. \ell \leq j \leq u \wedge a[j] = e \end{aligned} \quad (2)$$

according to the equivalence

$$F_1 \wedge F_2 \rightarrow (F_3 \rightarrow (F_4 \rightarrow F_5)) \Leftrightarrow (F_1 \wedge F_2 \wedge F_3 \wedge F_4) \rightarrow F_5.$$

This formula (2) is  $(T_Z \cup T_A)$ -valid.

◀ ▶ ↻ 🔍

A formula  $F$  annotating location  $L$  of program  $P$  is  $P$ -invariant if for all  $P$ -computations  $s_0, s_1, s_2, \dots$  and for each index  $i$ ,

$$s_i[\rho c] = L \quad \Rightarrow \quad s_i \models F$$

Annotations of  $P$  are  $P$ -invariant (invariant) iff each annotation of  $P$  is  $P$ -invariant at its location.

Annotations of  $P$  are  $P$ -inductive (inductive) iff all VCs generated from program  $P$  are  $T$ -valid

$$P\text{-inductive} \quad \Rightarrow \quad P\text{-invariant}$$

◀ ▶ ↻ 🔍

## $P$ -invariant and $P$ -inductive

Consider program  $P$  with function  $f$  s.t.

function precondition  $F_0$  and

initial location  $L_0$ .

A  $P$ -computation is a sequence of states

$$s_0, s_1, s_2, \dots$$

such that

▶  $s_0[\rho c] = L_0$  and  $s_0 \models F_0$ , and

▶ for each  $i$ ,  $s_{i+1}$  is the result of executing the instruction at  $s_i[\rho c]$  on state  $s_i$ .

where  $s_i[\rho c]$  = value of  $\rho c$  given by state  $s_i$

◀ ▶ ↻ 🔍

## Total Correctness

Total Correctness = Partial Correctness + Termination

Given that the input satisfies the function precondition, the function eventually halts and produces output that satisfies the function postcondition.

Proving function termination:

▶ Choose set  $S$  with well-founded relation  $\prec$

Usually set of  $n$ -tuples of natural numbers with the lexicographic extension  $\prec_n$

▶ Find function  $\delta$  (ranking function)

mapping

$$\text{program states} \quad \rightarrow \quad S$$

such that  $\delta$  decreases according to  $\prec$  along every basic path.

Since  $\prec$  is well-founded, there cannot exist an infinite sequence of program states.

◀ ▶ ↻ 🔍

## Choosing well-founded relation and ranking function

Example: Ackermann function — recursive calls

Choose  $(\mathbb{N}^2, <_2)$  as well-founded set

---

```
@pre x ≥ 0 ∧ y ≥ 0
@post rv ≥ 0
↓ (x, y)      ... ranking function δ : (x, y)
int Ack(int x, int y) {
  if (x = 0) {
    return y + 1;
  }
  else if (y = 0) {
    return Ack(x - 1, 1);
  }
  else {
    int z := Ack(x, y - 1);
    return Ack(x - 1, z);
  }
}
```

---

(3)

---

```
@pre x ≥ 0 ∧ y ≥ 0
↓ (x, y)
assume x ≠ 0;
assume y ≠ 0;
assume v1 ≥ 0;
z := v1;
↓ (x - 1, z)
```

---

- ▶ Show  $\delta : (x, y)$  maps into  $\mathbb{N}^2$ , i.e.,  
 $x \geq 0$  and  $y \geq 0$  are invariants
- ▶ Show  $\delta : (x, y)$  decreases from function entry to each recursive call. We show this.

The basic paths are:

(1)

---

```
@pre x ≥ 0 ∧ y ≥ 0
↓ (x, y)
assume x ≠ 0;
assume y = 0;
↓ (x - 1, 1)
```

---

(2)

---

```
@pre x ≥ 0 ∧ y ≥ 0
↓ (x, y)
assume x ≠ 0;
assume y ≠ 0;
↓ (x, y - 1)
```

---

Showing decrease of ranking function

For basic path with ranking function

```
@ F
↓ δ[ $\bar{x}$ ]
S1;
⋮
Sk;
↓ κ[ $\bar{x}$ ]
```

We must prove that

the value of  $\kappa$  after executing  $S_1; \dots; S_n$

is less than

the value of  $\delta$  before executing the statements

Thus, we show the verification condition

$$F \rightarrow wp(\kappa < \delta[\bar{x}_0], S_1; \dots; S_k)\{\bar{x}_0 \mapsto \bar{x}\}.$$

Example: Ackermann function — recursive calls

Verification conditions for the three basic paths

- $x \geq 0 \wedge y \geq 0 \wedge x \neq 0 \wedge y = 0 \Rightarrow (x-1, 1) <_2 (x, y)$
- $x \geq 0 \wedge y \geq 0 \wedge x \neq 0 \wedge y \neq 0 \Rightarrow (x, y-1) <_2 (x, y)$
- $x \geq 0 \wedge y \geq 0 \wedge x \neq 0 \wedge y \neq 0 \wedge v_1 \geq 0 \Rightarrow (x-1, v_1) <_2 (x, y)$

Then compute

```
wp((x-1, z) <_2 (x_0, y_0),
  assume x ≠ 0; assume y ≠ 0; assume v_1 ≥ 0; z := v_1)
⇔ wp((x-1, v_1) <_2 (x_0, y_0),
  assume x ≠ 0; assume y ≠ 0; assume v_1 ≥ 0)
⇔ x ≠ 0 ∧ y ≠ 0 ∧ v_1 ≥ 0 → (x-1, v_1) <_2 (x_0, y_0)
```

Renaming  $x_0$  and  $y_0$  to  $x$  and  $y$ , respectively, gives

$$x \neq 0 \wedge y \neq 0 \wedge v_1 \geq 0 \rightarrow (x-1, v_1) <_2 (x, y).$$

Noting that path **(3)** begins by asserting  $x \geq 0 \wedge y \geq 0$ , we finally have

$$x \geq 0 \wedge y \geq 0 \wedge x \neq 0 \wedge y \neq 0 \wedge v_1 \geq 0 \Rightarrow (x-1, v_1) <_2 (x, y).$$

```
for
  @L_2 : i + 1 ≥ 0 ∧ i - j ≥ 0
  ↓ (i + 1, i - j) ... ranking function δ_2
  (int j := 0; j < i; j := j + 1) {
    if (a[j] > a[j + 1]) {
      int t := a[j];
      a[j] := a[j + 1];
      a[j + 1] := t;
    }
  }
}
return a;
```

Example: BubbleSort — loops

Choose  $(\mathbb{N}^2, <_2)$  as well-founded set

```
@pre T
@post T
int[] BubbleSort(int[] a_0) {
  int[] a := a_0;
  for
    @L_1 : i + 1 ≥ 0
    ↓ (i + 1, i + 1) ... ranking function δ_1
    (int i := |a| - 1; i > 0; i := i - 1) {
```

We have to prove

- loop invariants are inductive
- function decreases along each basic path.

The relevant basic paths

(1)  $\text{---}$

```
@L_1 : i + 1 ≥ 0
↓ L_1 : (i + 1, i + 1)
assume i > 0;
j := 0;
↓ L_2 : (i + 1, i - j)
```

(2),(3)  $\text{---}$

```
@L_2 : i + 1 ≥ 0 ∧ i - j ≥ 0
↓ L_2 : (i + 1, i - j)
assume j < i;
...
j := j + 1;
↓ L_2 : (i + 1, i - j)
```

(4)

 $@L_2 : i + 1 \geq 0 \wedge i - j \geq 0$  $\downarrow L_2 : (i + 1, i - j)$ assume  $j \geq i$ ; $i := i - 1$ ; $\downarrow L_1 : (i + 1, i + 1)$ Verification conditions

Path (1)

 $i + 1 \geq 0 \wedge i > 0 \Rightarrow (i + 1, i - 0) <_2 (i + 1, i + 1),$ 

Paths (2) and (3)

 $i + 1 \geq 0 \wedge i - j \geq 0 \wedge j < i \Rightarrow (i + 1, i - (j + 1)) <_2 (i + 1, i - j),$ 

Path (4)

 $i + 1 \geq 0 \wedge i - j \geq 0 \wedge j \geq i \Rightarrow ((i - 1) + 1, (i - 1) + 1) <_2 (i + 1, i - j),$ 

which are valid. Hence, BubbleSort always halts.

## 6. Program Correctness: Strategies

### Developing Inductive Assertions

Some structured techniques for developing inductive annotations for proving partial correctness. Just heuristics.

#### Basic Facts

Example: LinearSearch

```

for
  @L :  $\ell \leq i \leq u + 1$ 
  (int  $i := \ell; i \leq u; i := i + 1$ ) {
    if ( $a[i] = e$ ) return true;
  }

```

Example: BubbleSort

```

for
  @L1 :  $-1 \leq i < |a|$ 
  (int  $i := |a| - 1; i > 0; i := i - 1$ ) {
    for
      @L2 :  $0 < i < |a| \wedge 0 \leq j \leq i$ 
      (int  $j := 0; j < i; j := j + 1$ ) {
        if ( $a[j] > a[j + 1]$ ) {
          int  $t := a[j]$ ;
           $a[j] := a[j + 1]$ ;
           $a[j + 1] := t$ ;
        }
      }
  }

```

## The Precondition Method

- Given annotation  $@L : F$
- Compute the precondition of  $F$  backward
- Find new annotation  $@L' : F'$



### Example: BinarySearch

```

@pre H?
@post T
bool BinarySearch(int[] a, int ℓ, int u, int e) {
  if (ℓ > u) return false;
  else {
    @ 2 ≠ 0;          ... basic fact
    int m := (ℓ + u) div 2;
    @ 0 ≤ m < |a|;   ... basic fact
    if (a[m] = e) return true;
    else if (a[m] < e) return BinarySearch(a, m + 1, u, e);
    else return BinarySearch(a, ℓ, m - 1, e);
  }
}

```

$@pre H : 0 \leq \ell \wedge u < |a|$

guaranteed

$$0 \leq \ell \wedge u < |a| \rightarrow wp(F, S_1; S_2)$$

is  $T_{\mathbb{Z}}$ -valid. The runtime assertion

$$0 \leq m < |a|$$

holds in every execution of BinarySearch in which the precondition

$$@pre 0 \leq \ell \wedge u < |a|$$

is satisfied.

(·)

---

```

@pre H : ?
S1 : assume ℓ ≤ u;
S2 : m := (ℓ + u) div 2;
@ F : 0 ≤ m < |a|

```

---

Compute

```

wp(F, S1; S2)
⇔ wp(wp(F, m := (ℓ + u) div 2), S1)
⇔ wp(F {m ↦ (ℓ + u) div 2}, S1)
⇔ wp(F {m ↦ (ℓ + u) div 2}, assume ℓ ≤ u)
⇔ ℓ ≤ u → F {m ↦ (ℓ + u) div 2}
⇔ ℓ ≤ u → 0 ≤ (ℓ + u) div 2 < |a|
⇔ 0 ≤ ℓ ∧ u < |a|

```

## Part II: Algorithm Reasoning

### 7. Quantified Linear Arithmetic

Quantifier Elimination (QE) — algorithm for elimination of all quantifiers of formula  $F$  until quantifier-free formula  $G$  that is equivalent to  $F$  remains

Note: Could be enough  $F$  is equisatisfiable to  $F'$ , that is  $F$  is satisfiable iff  $F'$  is satisfiable

A theory  $T$  admits quantifier elimination if there is an algorithm that given  $\Sigma$ -formula returns a quantifier-free  $\Sigma$ -formula  $G$  that is  $T$ -equivalent

### Example

For  $\Sigma_{\mathbb{Q}}$ -formula

$$F : \exists x. 2x = y,$$

quantifier-free  $T_{\mathbb{Q}}$ -equivalent  $\Sigma_{\mathbb{Q}}$ -formula is

$$G : \top$$

For  $\Sigma_{\mathbb{Z}}$ -formula

$$F : \exists x. 2x = y,$$

there is no quantifier-free  $T_{\mathbb{Z}}$ -equivalent  $\Sigma_{\mathbb{Z}}$ -formula.

Let  $T_{\mathbb{Z}}$  be  $T_{\mathbb{Z}}$  with divisibility predicates.

For  $\Sigma_{\mathbb{Z}}$ -formula

$$F : \exists x. 2x = y,$$

a quantifier-free  $T_{\mathbb{Z}}$ -equivalent  $\Sigma_{\mathbb{Z}}$ -formula is

$$G : 2 \mid y.$$



In developing a QE algorithm for theory  $T$ , we need only consider formulae of the form

$$\exists x. F$$

for quantifier-free  $F$

Example: For  $\Sigma$ -formula

$$G_1 : \exists x. \forall y. \underbrace{\exists z. F_1[x, y, z]}_{F_2[x, y]}$$

$$G_2 : \exists x. \forall y. F_2[x, y]$$

$$G_3 : \exists x. \underbrace{\neg \exists y. \neg F_2[x, y]}_{F_3[x]}$$

$$G_4 : \underbrace{\exists x. \neg F_3[x]}_{F_4}$$

$$G_5 : F_4$$

$G_5$  is quantifier-free and  $T$ -equivalent to  $G_1$



## Quantifier Elimination for $T_{\mathbb{Z}}$

$$\Sigma_{\mathbb{Z}} : \{\dots, -2, -1, 0, 1, 2, \dots, -3, -2, 2, 3, \dots, +, -, =, <\}$$

### Lemma:

Given quantifier-free  $\Sigma_{\mathbb{Z}}$ -formula  $F$  s.t.  $\text{free}(F) = \{y\}$ .

$F$  represents the set of integers

$$S : \{n \in \mathbb{Z} : F\{y \mapsto n\} \text{ is } T_{\mathbb{Z}}\text{-valid}\}.$$

Either  $S \cap \mathbb{Z}^+$  or  $\mathbb{Z}^+ \setminus S$  is finite.

where  $\mathbb{Z}^+$  is the set of positive integers

Example:  $\Sigma_{\mathbb{Z}}$ -formula  $F : \exists x. 2x = y$

$S$ : even integers

$S \cap \mathbb{Z}^+$ : positive even integers — infinite

$\mathbb{Z}^+ \setminus S$ : positive odd integers — infinite

Therefore, by the lemma, there is no quantifier-free  $T_{\mathbb{Z}}$ -formula that is  $T_{\mathbb{Z}}$ -equivalent to  $F$ .

Thus,  $T_{\mathbb{Z}}$  does not admit QE.



### Augmented theory $\widehat{T}_{\mathbb{Z}}$

$\widehat{\Sigma}_{\mathbb{Z}}$ :  $\Sigma_{\mathbb{Z}}$  with countable number of unary divisibility predicates

$$k \mid \cdot \text{ for } k \in \mathbb{Z}^+$$

Intended interpretations:

$$k \mid x \text{ holds iff } k \text{ divides } x \text{ without any remainder}$$

Example:

$$x > 1 \wedge y > 1 \wedge 2 \mid x + y$$

is satisfiable (choose  $x = 2, y = 2$ ).

$$\neg(2 \mid x) \wedge 4 \mid x$$

is not satisfiable.

Axioms of  $\widehat{T}_{\mathbb{Z}}$ : axioms of  $T_{\mathbb{Z}}$  with additional countable set of axioms

$$\forall x. k \mid x \leftrightarrow \exists y. x = ky \text{ for } k \in \mathbb{Z}^+$$

Example:

$$\neg(x < y) \wedge \neg(x = y + 3)$$

$\Downarrow$

$$y < x + 1 \wedge (x < y + 3 \vee y + 3 < x)$$

Step 3

Collect terms containing  $x$  so that literals have the form

$$hx < t, \quad t < hx, \quad k \mid hx + t, \quad \text{or} \quad \neg(k \mid hx + t),$$

where  $t$  is a term and  $h, k \in \mathbb{Z}^+$ . The output is the formula  $\exists x. F_3[x]$ , which is  $\widehat{T}_{\mathbb{Z}}$ -equivalent to  $\exists x. F[x]$ .

Example:

$$x + x + y < z + 3z + 2y - 4x$$

$\Downarrow$

$$6x < 4z + y$$

### $\widehat{T}_{\mathbb{Z}}$ admits QE (Cooper's method)

Algorithm: Given  $\widehat{\Sigma}_{\mathbb{Z}}$ -formula  $\exists x. F[x]$ , where  $F$  is quantifier-free  
Construct quantifier-free  $\widehat{\Sigma}_{\mathbb{Z}}$ -formula that is equivalent to  $\exists x. F[x]$ .

Step 1

Put  $F[x]$  in NNF  $F_1[x]$ , that is,

$\exists x. F_1[x]$  has negations only in literals (only  $\wedge, \vee$ )  
and  $\widehat{T}_{\mathbb{Z}}$ -equivalent to  $\exists x. F[x]$

Step 2

Replace (left to right)

$$\begin{aligned} s = t &\Leftrightarrow s < t + 1 \wedge t < s + 1 \\ \neg(s = t) &\Leftrightarrow s < t \vee t < s \\ \neg(s < t) &\Leftrightarrow t < s + 1 \end{aligned}$$

The output  $\exists x. F_2[x]$  contains only literals of form

$$s < t, \quad k \mid t, \quad \text{or} \quad \neg(k \mid t),$$

where  $s, t$  are  $\widehat{T}_{\mathbb{Z}}$ -terms and  $k \in \mathbb{Z}^+$ .

Step 4

Let

$$\delta' = \text{lcm}\{h : h \text{ is a coefficient of } x \text{ in } F_3[x]\},$$

where lcm is the least common multiple. Multiply atoms in  $F_3[x]$  by constants so that  $\delta'$  is the coefficient of  $x$  everywhere:

$$\begin{aligned} hx < t &\Leftrightarrow \delta'x < h't && \text{where } h'h = \delta' \\ t < hx &\Leftrightarrow h't < \delta'x && \text{where } h'h = \delta' \\ k \mid hx + t &\Leftrightarrow h'k \mid \delta'x + h't && \text{where } h'h = \delta' \\ \neg(k \mid hx + t) &\Leftrightarrow \neg(h'k \mid \delta'x + h't) && \text{where } h'h = \delta' \end{aligned}$$

The result  $\exists x. F'_3[x]$ , in which all occurrences of  $x$  in  $F'_3[x]$  are in terms  $\delta'x$ .

Replace  $\delta'x$  terms in  $F'_3$  with a fresh variable  $x'$  to form

$$F''_3 : F_3\{\delta'x \mapsto x'\}$$

Finally, construct

$$\exists x'. \underbrace{F_3[x'] \wedge \delta' \mid x'}_{F_4[x']}$$

$\exists x'. F_4[x']$  is equivalent to  $\exists x. F[x]$  and each literal of  $F_4[x']$  has one of the forms:

- (A)  $x' < a$
- (B)  $b < x'$
- (C)  $h \mid x' + c$
- (D)  $\neg(k \mid x' + d)$

where  $a, b, c, d$  are terms that do not contain  $x$ , and  $h, k \in \mathbb{Z}^+$ .

◀ ◻ ▶ ↺ ↻ 🔍

Step 5 (trickiest part):

Construct

left infinite projection  $F_{-\infty}[x']$

of  $F_4[x']$  by

- (A) replacing literals  $x' < a$  by  $\perp$
- (B) replacing literals  $b < x'$  by  $\perp$

idea: very small numbers satisfy (A) literals but not (B) literals

Let

$$\delta = \text{lcm} \left\{ \begin{array}{l} h \text{ of (C) literals } h \mid x' + c \\ k \text{ of (D) literals } \neg(k \mid x' + d) \end{array} \right\}$$

and  $B$  be the set of  $b$  terms appearing in (B) literals. Construct

$$F_5 : \bigvee_{j=1}^{\delta} F_{-\infty}[j] \vee \bigvee_{j=1}^{\delta} \bigvee_{b \in B} F_4[b + j].$$

$F_5$  is quantifier-free and  $\widehat{T}_{\mathbb{Z}}$ -equivalent to  $F$ .

◀ ◻ ▶ ↺ ↻ 🔍

Example:  $\widehat{T}_{\mathbb{Z}}$ -formula

$$\exists x. \underbrace{3x + 1 > y \wedge 2x - 6 < z \wedge 4 \mid 5x + 1}_{F[x]}$$

after step 3

$$\exists x. \underbrace{2x < z + 6 \wedge y - 1 < 3x \wedge 4 \mid 5x + 1}_{F_3[x]}$$

Collecting coefficients of  $x$  (step 4),

$$\delta' = \text{lcm}(2, 3, 5) = 30$$

Multiply when necessary

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

Replacing  $30x$  with fresh  $x'$

$$\exists x'. \underbrace{x' < 15z + 90 \wedge 10y - 10 < x' \wedge 24 \mid x' + 6 \wedge 30 \mid x'}_{F_4[x']}$$

$\exists x'. F_4[x']$  is equivalent to  $\exists x. F[x]$

◀ ◻ ▶ ↺ ↻ 🔍

Intuition

Property (Periodicity)

if  $k \mid \delta$   
then  $k \mid n$  iff  $k \mid n + \lambda\delta$  for all  $\lambda \in \mathbb{Z}$

That is,  $k \mid \cdot$  cannot distinguish between  $k \mid n$  and  $k \mid n + \lambda\delta$ .

By the choice of  $\delta$  (lcm of the  $h$ 's and  $k$ 's) — no  $\mid$  literal in  $F_5$  can distinguish between  $n$  and  $n + \delta$ .

$$F_5 : \bigvee_{j=1}^{\delta} F_{-\infty}[j] \vee \bigvee_{j=1}^{\delta} \bigvee_{b \in B} F_4[b + j]$$

left disjunct  $\bigvee_{j=1}^{\delta} F_{-\infty}[j]$  :

Contains only  $\mid$  literals

Asserts: no least  $n \in \mathbb{Z}$  s.t.  $F[n]$ .

For if there exists  $n$  satisfying  $F_{-\infty}$ , then every  $n - \lambda\delta$ , for  $\lambda \in \mathbb{Z}^+$ , also satisfies  $F_{-\infty}$ .

◀ ◻ ▶ ↺ ↻ 🔍

right disjunct  $\bigvee_{j=1}^{\delta} \bigvee_{b \in B} F_4[b + j]$  :

Asserts: There is least  $n \in \mathbb{Z}$  s.t.  $F[n]$ .

For let  $b^*$  be the largest  $b$  in  $(B)$ .

If  $n \in \mathbb{Z}$  is s.t.  $F[n]$ ,

then

$$\exists j(1 \leq j \leq \delta). b^* + j \leq n \wedge F[b^* + j]$$

In other words,

if there is a solution,

then one must appear in  $\delta$  interval to the right of  $b^*$

Example (cont):

$$\begin{array}{c} \exists x. \underbrace{3x + 1 > y \wedge 2x - 6 < z \wedge 4 \mid 5x + 1}_{F[x]} \\ \quad \quad \quad \downarrow \\ \exists x'. \underbrace{x' < 15z + 90 \wedge 10y - 10 < x' \wedge 24 \mid x' + 6 \wedge 30 \mid x'}_{F_4[x']} \end{array}$$

Navigation icons

Example:

$$\exists x. \underbrace{(3x + 1 < 10 \vee 7x - 6 > 7) \wedge 2 \mid x}_{F[x]}$$

Isolate  $x$  terms

$$\exists x. (3x < 9 \vee 13 < 7x) \wedge 2 \mid x,$$

so

$$\delta' = \text{lcm}\{3, 7\} = 21.$$

After multiplying coefficients by proper constants,

$$\exists x. (21x < 63 \vee 39 < 21x) \wedge 42 \mid 21x,$$

we replace  $21x$  by  $x'$ :

$$\exists x'. \underbrace{(x' < 63 \vee 39 < x') \wedge 42 \mid x' \wedge 21 \mid x'}_{F_4[x']}.$$

Navigation icons

By step 5,

$$F_{-\infty}[x] : \top \wedge \perp \wedge 24 \mid x' + 6 \wedge 30 \mid x',$$

which simplifies to  $\perp$ . Compute

$$\delta = \text{lcm}\{24, 30\} = 120 \quad \text{and} \quad B = \{10y - 10\}.$$

Then replacing  $x'$  by  $10y - 10 + j$  in  $F_4[x']$  produces

$$F_5 : \bigvee_{j=1}^{120} \left[ \begin{array}{l} 10y - 10 + j < 15z + 90 \wedge 10y - 10 < 10y - 10 + j \\ \wedge 24 \mid 10y - 10 + j + 6 \wedge 30 \mid 10y - 10 + j \end{array} \right]$$

which simplifies to

$$F_5 : \bigvee_{j=1}^{120} \left[ \begin{array}{l} 10y + j < 15z + 100 \wedge 0 < j \\ \wedge 24 \mid 10y + j - 4 \wedge 30 \mid 10y - 10 + j \end{array} \right].$$

$F_5$  is quantifier-free and  $\widehat{T}_{\mathbb{Z}}$ -equivalent to  $F$ .

Navigation icons

Then

$$F_{-\infty}[x'] : (\top \vee \perp) \wedge 42 \mid x' \wedge 21 \mid x',$$

or, simplifying,

$$F_{-\infty}[x'] : 42 \mid x' \wedge 21 \mid x'.$$

Finally,

$$\delta = \text{lcm}\{21, 42\} = 42 \quad \text{and} \quad B = \{39\},$$

so

$$F_5 : \bigvee_{j=1}^{42} (42 \mid j \wedge 21 \mid j) \vee \bigvee_{j=1}^{42} ((39 + j < 63 \vee 39 < 39 + j) \wedge 42 \mid 39 + j \wedge 21 \mid 39 + j)$$

Since  $42 \mid 42$  and  $21 \mid 42$ , the left main disjunct simplifies to  $\top$ , so that  $F$  is  $\widehat{T}_{\mathbb{Z}}$ -equivalent to  $\top$ . Thus,  $F$  is  $\widehat{T}_{\mathbb{Z}}$ -valid.

Navigation icons

Example:

$$\exists x. \underbrace{2x = y}_{F[x]}$$

Rewriting

$$\exists x. \underbrace{y - 1 < 2x \wedge 2x < y + 1}_{F_3[x]}$$

Then

$$\delta' = \text{lcm}\{2, 2\} = 2,$$

so by Step 4

$$\exists x'. \underbrace{y - 1 < x' \wedge x' < y + 1 \wedge 2 \mid x'}_{F_4[x']}$$

$F_{-\infty}$  produces  $\perp$ .

◀ ◻ ▶ ↺ ↻ 🔍 ↻

Two Improvements:

#### A. Symmetric Elimination

In step 5, if there are fewer

(A) literals  $x' < a$

than

(B) literals  $b < x'$ .

Construct the right infinite projection  $F_{+\infty}[x']$  from  $F_4[x']$  by replacing

each (A) literal  $x' < a$  by  $\perp$

and

each (B) literal  $b < x'$  by  $\top$ .

Then right elimination.

$$F_5 : \bigvee_{j=1}^{\delta} F_{+\infty}[-j] \vee \bigvee_{j=1}^{\delta} \bigvee_{a \in A} F_4[a - j].$$

◀ ◻ ▶ ↺ ↻ 🔍 ↻

However,

$$\delta = \text{lcm}\{2\} = 2 \quad \text{and} \quad B = \{y - 1\},$$

so

$$F_5 : \bigvee_{j=1}^2 (y - 1 < y - 1 + j \wedge y - 1 + j < y + 1 \wedge 2 \mid y - 1 + j)$$

Simplifying,

$$F_5 : \bigvee_{j=1}^2 (0 < j \wedge j < 2 \wedge 2 \mid y - 1 + j)$$

and then

$$F_5 : 2 \mid y,$$

which is quantifier-free and  $\widehat{T}_{\mathbb{Z}}$ -equivalent to  $F$ .

◀ ◻ ▶ ↺ ↻ 🔍 ↻

#### B. Eliminating Blocks of Quantifiers

$$\exists x_1. \dots \exists x_n. F[x_1, \dots, x_n]$$

where  $F$  quantifier-free.

Eliminating  $x_n$  (left elimination) produces

$$G_1 : \exists x_1. \dots \exists x_{n-1}. \bigvee_{j=1}^{\delta} F_{-\infty}[x_1, \dots, x_{n-1}, j] \vee \bigvee_{j=1}^{\delta} \bigvee_{b \in B} F_4[x_1, \dots, x_{n-1}, b + j]$$

which is equivalent to

$$G_2 : \bigvee_{j=1}^{\delta} \exists x_1. \dots \exists x_{n-1}. F_{-\infty}[x_1, \dots, x_{n-1}, j] \vee \bigvee_{j=1}^{\delta} \bigvee_{b \in B} \exists x_1. \dots \exists x_{n-1}. F_4[x_1, \dots, x_{n-1}, b + j]$$

Treat  $j$  as a free variable and examine only  $1 + |B|$  formulae

▶  $\exists x_1. \dots \exists x_{n-1}. F_{-\infty}[x_1, \dots, x_{n-1}, j]$

▶  $\exists x_1. \dots \exists x_{n-1}. F_4[x_1, \dots, x_{n-1}, b + j]$  for each  $b \in B$

◀ ◻ ▶ ↺ ↻ 🔍 ↻

Example:

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

Since  $\delta' = \text{lcm}\{1, 13\} = 13$

$$\exists y. \exists x. 13x < -26 \wedge 13 - 65y < 13x \wedge 1 + y < 13x$$

Then

$$\exists y. \exists x'. x' < -26 \wedge 13 - 65y < x' \wedge 1 + y < x' \wedge 13 \mid x'$$

There is one (A) literal  $x' < \dots$  and two (B) literals  $\dots < x'$ , we use right elimination.

$$F_{+\infty} = \perp \quad \delta = \{13\} = 13 \quad A = \{-26\}$$

$$\exists y. \bigvee_{j=1}^{13} \left[ \begin{array}{l} -26 - j < -26 \wedge 13 - 65y < -26 - j \\ \wedge 1 + y < -26 - j \wedge 13 \mid -26 - j \end{array} \right]$$

Commute

$$G : \bigvee_{j=1}^{13} \exists y. j > 0 \wedge 39 + j < 65y \wedge y < -27 - j \wedge 13 \mid -26 - j$$

Apply QE (treating  $j$  as free variable)

$$H : \exists y. j > 0 \wedge 39 + j < 65y \wedge y < -27 - j \wedge 13 \mid -26 - j$$

Simplify

$$H' : \bigvee_{k=1}^{65} (k < -1794 - 66j \wedge 13 \mid -26 - j \wedge 65 \mid 39 + j + k)$$

Replace  $H$  with  $H'$  in  $G$

$$\bigvee_{j=1}^{13} \bigvee_{k=1}^{65} (k < -1794 - 66j \wedge 13 \mid -26 - j \wedge 65 \mid 39 + j + k)$$

This formula is  $\widehat{T}_{\mathbb{Z}}$ -equivalent to  $F$ .

## Quantifier Elimination over Rationals

$$\Sigma_{\mathbb{Q}} : \{0, 1, +, -, =, \geq\}$$

we use  $>$  instead of  $\geq$ , as

$$x \geq y \Leftrightarrow x > y \vee x = y \quad x > y \Leftrightarrow x \geq y \wedge \neg(x = y).$$

### Ferrante and Rackoff's Method

Given a  $\Sigma_{\mathbb{Q}}$ -formula  $\exists x. F[x]$ , where  $F[x]$  is quantifier-free

Generate quantifier-free formula  $F_4$  (four steps) s.t.

$$F_4 \text{ is } \Sigma_{\mathbb{Q}}\text{-equivalent to } \exists x. F[x].$$

Step 1: Put  $F[x]$  in NNF. The result is  $\exists x. F_1[x]$ .

Step 2: Replace literals (left to right)

$$\neg(s < t) \Leftrightarrow t < s \vee t = s$$

$$\neg(s = t) \Leftrightarrow t < s \vee t > s$$

The result  $\exists x. F_2[x]$  does not contain negations.

Step 3: Solve for  $x$  in each atom of  $F_2[x]$ , e.g.,

$$t < cx \quad \Rightarrow \quad \frac{t}{c} < x$$

where  $c \in \mathbb{Z} - \{0\}$ .

All atoms in the result  $\exists x. F_3[x]$  have form

(A)  $x < a$

(B)  $b < x$

(C)  $x = c$

where  $a, b, c$  are terms that do not contain  $x$ .

Step 4: Construct from  $F_3[x]$

- ▶ left infinite projection  $F_{-\infty}$  by replacing
  - (A) atoms  $x < a$  by  $\top$
  - (B) atoms  $b < x$  by  $\perp$
  - (C) atoms  $x = c$  by  $\perp$
- ▶ right infinite projection  $F_{+\infty}$  by replacing
  - (A) atoms  $x < a$  by  $\perp$
  - (B) atoms  $b < x$  by  $\top$
  - (C) atoms  $x = c$  by  $\perp$

Let  $S$  be the set of  $a, b, c$  terms from (A), (B), (C) atoms.  
Construct the final

$$F_4 : F_{-\infty} \vee F_{+\infty} \vee \bigvee_{s,t \in S} F_3 \left[ \frac{s+t}{2} \right],$$

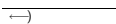
which is  $T_Q$ -equivalent to  $\exists x. F[x]$ .

- ▶  $F_{-\infty}$  captures the case when small  $n \in \mathbb{Q}$  satisfy  $F_3[n]$
- ▶  $F_{+\infty}$  captures the case when large  $n \in \mathbb{Q}$  satisfy  $F_3[n]$
- ▶ last disjunct: for  $s, t \in S$ 
  - if  $s = t$ , check whether  $s \in S$  satisfies  $F_4[s]$
  - if  $s \neq t$ ,  $\frac{s+t}{2}$  represents the whole interval  $(s, t)$ , so check  $F_4[\frac{s+t}{2}]$

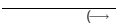
Intuition

Step 4 says that four cases are possible:

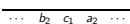
1. There is a left open interval s.t. all elements satisfy  $F(x)$ .



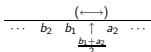
2. There is a right open interval s.t. all elements satisfy  $F(x)$ .



3. Some  $a_i, b_j$ , or  $c_j$  satisfies  $F(x)$ .



4. There is an open interval between two  $a_i, b_j$ , or  $c_j$  terms s.t. every element satisfies  $F(x)$ .



Example:  $\Sigma_Q$ -formula

$$\exists x. \underbrace{3x + 1 < 10 \wedge 7x - 6 > 7}_{F[x]}$$

Solving for  $x$

$$\exists x. \underbrace{x < 3 \wedge x > \frac{13}{7}}_{F_3[x]}$$

Step 4:  $x < 3$  in (A)  $\Rightarrow F_{-\infty} = \perp$   
 $x > \frac{13}{7}$  in (B)  $\Rightarrow F_{+\infty} = \perp$

$$F_4 : \bigvee_{s,t \in S} \underbrace{\left( \frac{s+t}{2} < 3 \wedge \frac{s+t}{2} > \frac{13}{7} \right)}_{F_3[\frac{s+t}{2}]}$$

$$S = \{3, \frac{13}{7}\} \Rightarrow$$

$$F_3 \left[ \frac{3+3}{2} \right] = \perp \quad F_3 \left[ \frac{\frac{13}{7} + \frac{13}{7}}{2} \right] = \perp$$

$$F_3 \left[ \frac{\frac{13}{7} + 3}{2} \right] : \frac{\frac{13}{7} + 3}{2} < 3 \wedge \frac{\frac{13}{7} + 3}{2} > \frac{13}{7}$$

simplifies to  $\top$ .

Thus,  $F_4 : \top$  is  $T_Q$ -equivalent to  $\exists x. F[x]$ ,  
so  $\exists x. F[x]$  is  $T_Q$ -valid.

## 8. Quantifier-Free Linear Arithmetic

### Decision Procedures for Quantifier-free Fragments

For theory  $T$  with signature  $\Sigma$  and axioms  $\Sigma$ -formulae of form  
 $\forall x_1, \dots, x_n. F[x_1, \dots, x_n]$

Decide if

$F[x_1, \dots, x_n]$  or  $\exists x_1, \dots, x_n. F[x_1, \dots, x_n]$  is  $T$ -satisfiable

Decide if  
 $F[x_1, \dots, x_n]$  or  $\forall x_1, \dots, x_n. F[x_1, \dots, x_n]$  is  $T$ -valid

where  $F$  is quantifier-free and  $\text{free}(F) = \{x_1, \dots, x_n\}$

Note: no quantifier alternations

We consider only conjunctive quantifier-free  $\Sigma$ -formulae, i.e.,  
 conjunctions of  $\Sigma$ -literals ( $\Sigma$ -atoms or negations of  $\Sigma$ -atoms).  
 For given arbitrary quantifier-free  $\Sigma$ -formula  $F$ , convert it into  
 DNF  $\Sigma$ -formula

$$F_1 \vee \dots \vee F_k$$

where each  $F_i$  conjunctive.

$F$  is  $T$ -satisfiable iff at least one  $F_i$  is  $T$ -satisfiable.

### Preliminary Concepts

#### Vector

variable  $n$ -vector  $n$ -vector  $\vec{a} \in \mathbb{Q}^n$  transpose

$$\vec{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad \vec{a} = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} \quad \vec{a}^T = [a_1 \ \dots \ a_n]$$

#### Matrix

$m \times n$ -matrix  $A \in \mathbb{Q}^{m \times n}$  transpose

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \quad A^T = \begin{bmatrix} a_{11} & \dots & a_{m1} \\ \vdots & & \vdots \\ a_{1n} & \dots & a_{mn} \end{bmatrix}$$

column  $a_{1j}$   
 $\vdots$   
 $a_{i1} \dots a_{ij} \dots a_{in}$   
 $\vdots$   
 $a_{mj}$

row

## Multiplication

vector-vector

$$\vec{a}^T \vec{b} = [a_1 \ \cdots \ a_n] \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} = \sum_{i=1}^n a_i b_i$$

matrix-vector

$$A\vec{x} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n a_{1i} x_i \\ \vdots \\ \sum_{i=1}^n a_{mi} x_i \end{bmatrix}$$

matrix-matrix

$$\begin{bmatrix} a_{ik} \\ \vdots \\ a_{mk} \end{bmatrix} \begin{bmatrix} b_{kj} \\ \vdots \\ b_{nj} \end{bmatrix} = \begin{bmatrix} p_{ij} \\ \vdots \\ p_{mj} \end{bmatrix}$$

$A \quad B \quad P$

where  $p_{ij} = \vec{a}_i \vec{b}_j = [a_{i1} \ \cdots \ a_{in}] \begin{bmatrix} b_{1j} \\ \vdots \\ b_{nj} \end{bmatrix} = \sum_{k=1}^n a_{ik} b_{kj}$

Vector Space - set  $S$  of vectors closed under addition and scaling of vectors. That is,

$$\text{if } \vec{v}_1, \dots, \vec{v}_k \in S \text{ then } \lambda_1 \vec{v}_1 + \cdots + \lambda_k \vec{v}_k \in S \\ \text{for } \lambda_1, \dots, \lambda_n \in \mathbb{R}.$$

## Linear Equation

$F : A\vec{x} = \vec{b}$

$m \times n$ -matrix      variable  $n$ -vector       $m$ -vector

represents the  $\Sigma_{\mathbb{Q}}$ -formula

$$F : (a_{11}x_1 + \cdots + a_{1n}x_n = b_1) \wedge \cdots \wedge (a_{m1}x_1 + \cdots + a_{mn}x_n = b_m)$$

## Gaussian Elimination

Find  $\vec{x}$  s.t.  $A\vec{x} = \vec{b}$  by elementary row operations

- ▶ Swap two rows.
- ▶ Multiply a row by a nonzero scalar.
- ▶ Add one row to another.

## Special Vectors and Matrices

$\vec{0}$  - vector (column) of 0s

$\vec{1}$  - vector of 1s

$$\text{Thus } \vec{1}^T \vec{x} = \sum_{i=1}^n x_i$$

$$I = \begin{bmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{bmatrix} \text{ identity matrix } (n \times n)$$

Thus  $IA = AI = A$

$$\text{unit vector } e_i = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \leftarrow i$$

## Example:

Solve

$$\begin{bmatrix} 3 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 1 \\ 2 \end{bmatrix}$$

Construct the augmented matrix

$$\left[ \begin{array}{ccc|c} 3 & 1 & 2 & 6 \\ 1 & 0 & 1 & 1 \\ 2 & 2 & 1 & 2 \end{array} \right]$$

Apply the row operations as follows:

1. Add  $-2\vec{a}_1 + 4\vec{a}_2$  to  $\vec{a}_3$

$$\left[ \begin{array}{ccc|c} 3 & 1 & 2 & 6 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & -6 \end{array} \right]$$

2. Add  $-\bar{a}_1 + 2\bar{a}_2$  to  $\bar{a}_3$

$$\left[ \begin{array}{ccc|c} 3 & 1 & 2 & 6 \\ 0 & -1 & 1 & -3 \\ 0 & 0 & 1 & -6 \end{array} \right]$$

This augmented matrix is in triangular form.

Solving

$$x_3 = -6$$

$$-x_2 - x_3 = -3 \Rightarrow x_2 = -3$$

$$3x_1 + x_2 + 2x_3 = 6 \Rightarrow x_1 = 7$$

The solution is  $\bar{x} = [7 \ -3 \ -6]^T$



### Linear Inequality

$$G : A\bar{x} \leq b$$

represents the  $\Sigma_Q$ -formula

$$G : (a_{11}x_1 + \dots + a_{1n}x_n \leq b_1) \wedge \dots \wedge (a_{m1}x_1 + \dots + a_{mn}x_n \leq b_m)$$

The inequality describes a polyhedron in  $\mathbb{R}^n$ .

For  $m \times n$ -matrix  $A$ ,  $m$ -vector  $b$ , variable  $n$ -vector  $\bar{x}$  where  $m \geq n$ :

An  $n$ -vector  $\bar{v}$  is a vertex of  $A\bar{x} \leq b$  if there is nonsingular  $n \times n$ -submatrix  $A_0$  and corresponding  $n$ -subvector  $b_0$  s.t.

$$A_0\bar{v} = b_0$$



### Inverse Matrix

$A^{-1}$  is the inverse matrix of square matrix  $A$  if

$$AA^{-1} = A^{-1}A = I$$

Square matrix  $A$  is nonsingular (invertible) if its inverse  $A^{-1}$  exists.

How to compute  $A^{-1}$  of  $A$ ?

$$\left[ A \mid I \right] \xrightarrow{\substack{\text{elementary} \\ \text{row operations}}} \left[ I \mid A^{-1} \right]$$

How to compute  $k$ th column of  $A^{-1}$ ?

Solve  $A\bar{y} = e_k$ , i.e.

$$\left[ \begin{array}{c|c} A & \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \end{array} \right] \xrightarrow{\substack{\text{elementary} \\ \text{row operations}}} \bar{y} = \dots \quad (\text{kth column of } A^{-1})$$



### Optimization Problem

$$\max \bar{c}^T \bar{x} \quad \dots \text{objective function}$$

**subject to**

$$A\bar{x} \leq \bar{b} \quad \dots \text{constraints}$$

Solution: vertex  $\bar{v}^*$  satisfying  $A\bar{x} \leq \bar{b}$  and maximize  $\bar{c}^T \bar{x}$ . That is,

$$A\bar{v}^* \leq \bar{b} \text{ and}$$

$$\bar{c}^T \bar{v}^* \text{ is maximal: } \bar{c}^T \bar{v}^* \geq \bar{c}^T \bar{u} \text{ for all } \bar{u} \text{ satisfying } A\bar{u} \leq \bar{b}$$

► If  $A\bar{x} \leq \bar{b}$  is unsatisfiable  $\Rightarrow$  maximum is  $-\infty$

► It's possible that the maximum is unbounded

$\Rightarrow$  maximum is  $\infty$



Example: Consider optimization problem:

$$\max \underbrace{\begin{bmatrix} 1 & 1 & -1 & -1 \end{bmatrix}}_{\bar{c}^T} \underbrace{\begin{bmatrix} x \\ y \\ z_1 \\ z_2 \end{bmatrix}}_{\bar{x}}$$

subject to

$$\underbrace{\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}}_A \underbrace{\begin{bmatrix} x \\ y \\ z_1 \\ z_2 \end{bmatrix}}_{\bar{x}} \leq \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 3 \\ 2 \\ 2 \end{bmatrix}}_{\bar{b}}$$

$A$  is a  $7 \times 4$ -matrix,  $\bar{b}$  is a 7-vector, and  $\bar{x}$  is a variable 4-vector representing the four variables  $\{x, y, z_1, z_2\}$ .

### Duality Theorem

For  $A \in \mathbb{Z}^{m \times n}$ ,  $\bar{b} \in \mathbb{Z}^m$ ,  $\bar{c} \in \mathbb{Z}^n$ ,

$$\max\{\bar{c}^T \bar{x} \mid A\bar{x} \leq \bar{b}\} = \min\{\bar{y}^T \bar{b} \mid \bar{y} \geq \bar{0} \wedge \bar{y}^T A = \bar{c}^T\}$$

if the constraints are satisfiable.

That is,

maximizing the function  $\bar{c}^T \bar{x}$  over  $A\bar{x} \leq \bar{b}$   
(the primal form of the optimization problem)

is equivalent to

minimizing the function  $\bar{y}^T \bar{b}$  over all the nonnegative  $\bar{y}$   
s.t.  $\bar{y}^T A = \bar{c}^T$   
(the dual form of the optimization problem)

### Example (cont):

The objective function is

$$(x - z_1) + (y - z_2).$$

The constraints are equivalent to the  $\Sigma_Q$ -formula

$$\begin{aligned} x \geq 0 \wedge y \geq 0 \wedge z_1 \geq 0 \wedge z_2 \geq 0 \\ \wedge x + y \leq 3 \wedge x - z_1 \leq 2 \wedge y - z_2 \leq 2 \end{aligned}$$

$\bar{v} = [2 \ 1 \ 0 \ 0]^T$  is a vertex of the constraints. For the nonsingular submatrix  $A_0$  (rows 3, 4, 5, 6 of  $A$ ), we have

$$\underbrace{\begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & -1 & 0 \end{bmatrix}}_{A_0} \underbrace{\begin{bmatrix} 2 \\ 1 \\ 0 \\ 0 \end{bmatrix}}_{\bar{v}} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ 3 \\ 2 \end{bmatrix}}_{\bar{b}_0}$$

### Outline of Algorithm

Given  $\Sigma_Q$ -formula

$$F: a_{11}x_1 + \dots + a_{1n}x_n \leq b_1 \wedge \dots \wedge a_{m1}x_1 + \dots + a_{mn}x_n \leq b_m \quad (7.1)$$

or in matrix notation

$$F: A\bar{x} \leq \bar{b}$$

Note: • equations

$$a_{j1}x_1 + \dots + a_{jn}x_n = b_j$$

are allowed — break into two inequalities

$$a_{j1}x_1 + \dots + a_{jn}x_n \leq b_j \wedge -a_{j1}x_1 - \dots - a_{jn}x_n \leq -b_j.$$

• Strict inequalities

$$a_{j1}x_1 + \dots + a_{jn}x_n < b_j.$$

excluded from our discussion — but can be added.

### Outline of Algorithm (cont)

To determine the satisfiability of  $F$ ,

Step 0: reformulate the satisfiability of  $F$  as an optimization problem

$$M_F : \max\{\bar{c}^T \bar{x} \mid A' \bar{x} \leq \bar{b}'\}$$

s.t.  $F$  is  $T_Q$ -satisfiable iff the optimal value of  $M_F$  is a particular value  $v_F$  (derived from the structure of  $F$ )

Step 1, Step 2, ... (until termination) execute the simplex method

◁ ○ ▷ ↻ 🔍 ↺ ↻

### Step 0: From Satisfiability to Optimization

Given  $\Sigma_Q$ -formula

$$F : A\bar{x} \leq \bar{b} \tag{7.1}$$

reformulate to new constraint system (new  $A, \bar{x}, \bar{b}$ )

$$F' : \bar{x} \geq 0, A\bar{x} \leq \bar{b}$$

$F'$  is  $T_Q$ -equisatisfiable to  $F$

The trick: replace each variable  $x$  in  $F$  by  $x_1 - x_2$

Separate the new constraints  $A\bar{x} \leq \bar{b}$  into

$$D_1 \bar{x} \leq \bar{e}_1 \quad \text{and} \quad D_2 \bar{x} \geq \bar{e}_2 \quad \text{for} \quad \bar{e}_1 \geq \bar{0}, \bar{e}_2 > \bar{0}$$

◁ ○ ▷ ↻ 🔍 ↺ ↻

### Outline of Algorithm (cont)

The simplex method traverses the vertices of  $A'\bar{x} \leq \bar{b}'$  searching for the maximum of the objective function  $\bar{c}^T \bar{x}$ :

if  $\bar{v}_1, \bar{v}_2, \dots$  are the traversed vertices in Step 1, Step 2, ..., then

$$\bar{c}^T \bar{v}_1 < \bar{c}^T \bar{v}_2 < \dots$$

The simplex method terminates at some vertex  $\bar{v}_{i^*}$  where  $\bar{c}^T \bar{v}_{i^*}$  is the global optimum

Final step: Compare the discovered optimal value  $\bar{c}^T \bar{v}_{i^*}$  to the desired value  $v_F$ .

- ▶ if equal, then  $F$  is  $T_Q$ -satisfiable
- ▶ otherwise,  $F$  is  $T_Q$ -unsatisfiable

◁ ○ ▷ ↻ 🔍 ↺ ↻

### Example: $\Sigma_Q$ -formula

$$F : x + y \geq 1 \wedge x - y \geq -1.$$

To convert it to the form  $\bar{x} \geq \bar{0} \wedge A\bar{x} \leq \bar{b}$ , introduce nonnegative  $x_1, x_2$  for  $x$  and  $y_1, y_2$  for  $y$ :

$$F' : (x_1 - x_2) + (y_1 - y_2) \geq 1 \wedge (x_1 - x_2) - (y_1 - y_2) \geq -1 \\ \wedge x_1, x_2, y_1, y_2 \geq 0$$

$F$  is  $T_Q$ -equisatisfiable to  $F'$ . In matrix form,

$$F' : \underbrace{\begin{bmatrix} -1 & 1 & -1 & 1 \\ -1 & 1 & 1 & -1 \end{bmatrix}}_A \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix} \leq \underbrace{\begin{bmatrix} -1 \\ 1 \end{bmatrix}}_{\bar{b}}$$

Since  $b_1 < 0$  and  $b_2 > 0$ , separating constraints yields

$$\underbrace{\begin{bmatrix} -1 & 1 & 1 & -1 \end{bmatrix}}_{D_1} \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix} \leq \underbrace{\begin{bmatrix} 1 \end{bmatrix}}_{\bar{e}_1} \quad \text{and} \quad \underbrace{\begin{bmatrix} 1 & -1 & 1 & -1 \end{bmatrix}}_{D_2} \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix} \geq \underbrace{\begin{bmatrix} 1 \end{bmatrix}}_{\bar{e}_2}$$

◁ ○ ▷ ↻ 🔍 ↺ ↻

Generate the optimization problem:

$$M_F : \max \bar{\mathbf{1}}^T (D_2 \bar{\mathbf{x}} - \bar{\mathbf{z}}) \quad (7.2)$$

**subject to**

$$\begin{aligned} \bar{\mathbf{x}}, \bar{\mathbf{z}} &\geq \bar{\mathbf{0}} & (1) \\ D_1 \bar{\mathbf{x}} &\leq \bar{\mathbf{e}}_1 & (2) \\ D_2 \bar{\mathbf{x}} - \bar{\mathbf{z}} &\leq \bar{\mathbf{e}}_2 & (3) \end{aligned}$$

length of variable vector  $\bar{\mathbf{z}}$  = # of rows of  $D_2$

- ▶ The point  $\bar{\mathbf{x}} = \bar{\mathbf{0}}, \bar{\mathbf{z}} = \bar{\mathbf{0}}$  satisfies constraints (1) – (3). It's a vertex.
- ▶ The optimum  $v_F$  equals  $\bar{\mathbf{1}}^T \bar{\mathbf{e}}_2$  iff  $F$  is  $T_Q$ -satisfiable.

$M_F$  can be written in standard form as

$$M_F : \max \underbrace{\bar{\mathbf{1}}^T}_{\bar{\mathbf{c}}^T} \underbrace{\begin{bmatrix} D_2 & -I \end{bmatrix}}_{\bar{\mathbf{A}}} \underbrace{\begin{bmatrix} \bar{\mathbf{x}} \\ \bar{\mathbf{z}} \end{bmatrix}}_{\bar{\mathbf{y}}} \quad (7.3)$$

**subject to**

$$\underbrace{\begin{bmatrix} -I & -I \\ D_1 & \\ D_2 & -I \end{bmatrix}}_{\bar{\mathbf{A}}} \underbrace{\begin{bmatrix} \bar{\mathbf{x}} \\ \bar{\mathbf{z}} \end{bmatrix}}_{\bar{\mathbf{y}}} \leq \underbrace{\begin{bmatrix} \bar{\mathbf{0}} \\ \bar{\mathbf{e}}_1 \\ \bar{\mathbf{e}}_2 \end{bmatrix}}_{\bar{\mathbf{b}}}$$



Example (cont):

$$\underbrace{\begin{bmatrix} -1 & 1 & 1 & -1 \end{bmatrix}}_{D_1} \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix} \leq \underbrace{[1]}_{\bar{\mathbf{e}}_1} \quad \text{and} \quad \underbrace{\begin{bmatrix} 1 & -1 & 1 & -1 \end{bmatrix}}_{D_2} \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix} \geq \underbrace{[1]}_{\bar{\mathbf{e}}_2}$$

$D_2$  has only one row, so  $\bar{\mathbf{z}} = [z]$ . Pose the following optimization problem:

$$\begin{aligned} \max \quad & [1 \quad -1 \quad 1 \quad -1] \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix} - [z] \\ \text{subject to} \quad & x_1, x_2, y_1, y_2, z \geq 0 \\ & [-1 \quad 1 \quad 1 \quad -1] \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix} \leq [1] \\ & [1 \quad -1 \quad 1 \quad -1] \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix} - [z] \leq [1] \end{aligned}$$

$F$  is  $T_Q$ -satisfiable iff the optimum is  $\bar{\mathbf{1}}^T \bar{\mathbf{e}}_2 = 1$ .

$[x_1 \ x_2 \ y_1 \ y_2 \ z] = [0 \ 0 \ 0 \ 0 \ 0]$  is a vertex.



Example (cont):

Rewriting the optimization problem

$$\begin{aligned} \max \quad & \underbrace{[1 \quad -1 \quad 1 \quad -1 \quad -1]}_{\bar{\mathbf{c}}^T} \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \\ z \end{bmatrix} \\ \text{subject to} \quad & \underbrace{\begin{bmatrix} -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \\ -1 & 1 & 1 & -1 & 0 \\ 1 & -1 & 1 & -1 & -1 \end{bmatrix}}_{\bar{\mathbf{A}}} \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \\ z \end{bmatrix} \leq \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}}_{\bar{\mathbf{b}}} \end{aligned}$$



### Step $i$ ( $i \geq 1$ ): Find a Better Vertex

Optimization problem of form

$$\begin{aligned} \max \quad & \bar{c}^T \bar{x} \\ \text{subject to} \quad & A\bar{x} \leq \bar{b} \end{aligned}$$

we are given satisfying vertex  $\bar{v}_i$ .

- ▶ The simplex method traverses vertices of the space defined by  $A\bar{x} \leq \bar{b}$  to find the vertex  $\bar{v}^*$  that maximizes  $\bar{c}^T \bar{x}$ .
- ▶ One iteration seeks vertex  $\bar{v}_{i+1}$  "adjacent" to  $\bar{v}_i$  s.t.  $\bar{c}^T \bar{v}_{i+1} > \bar{c}^T \bar{v}_i$ .
- ▶ For  $i = 1$ , the initial vertex  $\bar{v}_1$  of  $M_F$  is  $\bar{x} = \bar{0}$ ,  $\bar{z} = \bar{0}$

Example (cont):

$$\bar{v}_1 = [x_1 \ x_2 \ y_1 \ y_2 \ z]^T = [0 \ 0 \ 0 \ 0 \ 0]^T$$

Navigation icons

Example (cont)

Choose the first five rows of  $A$  and  $\bar{b}$  ( $R = [1; 2; 3; 4; 5]$ ) since

$$\underbrace{\begin{bmatrix} -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix}}_{A_i} \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\bar{v}_i} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\bar{b}_i}$$

Solving (Gaussian elimination)

$$\underbrace{\begin{bmatrix} -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix}}_{A_i^{-1}} \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\bar{v}_i} = \underbrace{\begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \\ -1 \end{bmatrix}}_{\bar{z}}$$

yields

$$\bar{v}_i^T = [-1 \ 1 \ -1 \ 1 \ 1]$$

Then

$$\bar{w} = [-1 \ 1 \ -1 \ 1 \ 0 \ 0]^T$$

Navigation icons

Find  $\bar{u}$

Construct vector  $\bar{u}$  s.t.

$$\bar{u}^T A = \bar{c}^T \tag{7.4}$$

- ▶ Given  $\bar{v}_i$
- ▶ Construct  $n \times n$  nonsingular submatrix  $A_i$  with corresponding rows  $\bar{b}_i$  s.t.

$$A_i \bar{v}_i = \bar{b}_i$$

- ▶ Let  $R =$  rows of  $A$  in  $A_i$
- ▶ Solve

$$A_i^T \bar{u}_i = \bar{c}$$

- ▶ Let  $\bar{u}$  be  $\bar{u}_i$  for indices in  $R$  and 0's for indices not in  $R$ .

Navigation icons

Case 1:  $\bar{u} \geq \bar{0}$

In this case,  $\bar{v}_i$  is actually the optimal point with optimal value  $\bar{c}^T \bar{v}_i$ .

Case 2:  $\bar{u} \not\geq \bar{0}$ , i.e. there exists some  $u_k < 0$

In this case,  $\bar{v}_i$  is not the optimal point. We need to move along an edge to an adjacent vertex to increase the value of the objective function.

- ▶ Let  $k$  be the lowest index of  $\bar{u}$  s.t.  $u_k < 0$  (must be  $k \in R$ )
- ▶ Let  $k'$  be the corresponding row of  $\bar{w}_i$  and  $A_i$  and the corresponding column of  $-A_i^{-1}$

Find  $\bar{y}$

- ▶ Let  $\bar{y}$  be the  $k'$ th column of  $-A_i^{-1}$ . Solve  $A_i \bar{y} = -e_{k'}$

Then

$$\begin{aligned} \bar{a}_{\ell} \bar{y} &= 0 & \text{for every row } \bar{a}_{\ell} \text{ of } A_i, \ell \neq k' \\ \bar{a}_{k'} \bar{y} &= -1 & \text{for the } k' \text{th row } \bar{a}_{k'} \text{ of } A_i \end{aligned}$$

The vector  $\bar{y}$  provides the direction along which to move to the next vertex.

Navigation icons

### Example:

We found so far

$$\bar{v}_1 = [-1 \ 1 \ -1 \ 1 \ 1]^T \text{ and } \bar{w} = [-1 \ 1 \ -1 \ 1 \ 1 \ 0 \ 0]^T$$

$k = 1$  since the first row of  $\bar{w}$  is  $-1$ .  $k' = 1$  since it is also the first row of  $\bar{v}_i$ .

Thus, solve

$$\underbrace{\begin{bmatrix} -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix}}_{A_1} \bar{y} = \underbrace{\begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{-e_1}$$

for  $\bar{y}$ , yielding  $\bar{y} = [1 \ 0 \ 0 \ 0 \ 0]^T$



### Final Step: Satisfiability

The  $T_Q$ -formula  $F$  (7.1) is satisfiable

iff

the returned optimal value of the optimization problem  $M_F$  (7.2) is  $\mathbf{1}^T \bar{e}_2$ .

### Example (cont): We found in Step 1

$$\bar{y} = [1 \ 0 \ 0 \ 0 \ 0]^T$$

where

$$\underbrace{\begin{bmatrix} -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix}}_{A_1} \underbrace{\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\bar{y}} = \underbrace{\begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{-e_1}$$



### Find $\lambda$ and $v_{i+1}$

We move along edge  $\bar{y}$  to better vertex  $\bar{v}_{i+1}$ .

► Let  $S =$  indices  $\ell$  s.t.  $\bar{a}_\ell \bar{y} > 0$

► Find greatest  $\lambda \geq 0$  s.t.

$$A(\bar{v}_i + \lambda \bar{y}) \leq \bar{b}$$

Choose  $\lambda_j$  s.t.

$$\bar{a}_\ell(\bar{v}_i + \lambda_j \bar{y}) = b_\ell \quad \text{for some } \ell \in S$$

$$\bar{a}_m(\bar{v}_i + \lambda_j \bar{y}) \leq b_m \quad \text{for other } m \neq \ell$$

► Set

$$\boxed{\bar{v}_{i+1} = \bar{v}_i + \lambda_j \bar{y}} \quad (7.7)$$

Vertex  $\bar{v}_{i+1}$  is discovered by moving along ray  $\bar{y}$  as far as possible without violating the constraints. Moreover

$$\bar{c}^T \bar{v}_{i+1} > \bar{c}^T \bar{v}_i$$

► Construct  $A_{i+1}$  from  $A_i$  for next iteration by substituting row  $\bar{a}_\ell$  of  $A$  for row  $\bar{a}_{k'}$  of  $A_i$

Since there are only finite number of vertices to examine, Case 1 eventually occurs.



### Example (cont)

Compute  $A\bar{y}$

$$\underbrace{\begin{bmatrix} -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \\ -1 & 1 & 1 & -1 & 0 \\ 1 & -1 & 1 & -1 & -1 \end{bmatrix}}_A \underbrace{\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\bar{y}} = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ 1 \end{bmatrix}$$

$S = [7]$  since  $\bar{a}_7 \bar{y} = 1 > 0$ . Examining the 7th row of the constraints, choose the greatest  $\lambda_1$  such that (7.7b)

$$\underbrace{[1 \ -1 \ 1 \ -1 \ -1]}_{\bar{a}_7} (\bar{v}_1 + \lambda_1 \bar{y}) = [1 \ -1 \ 1 \ -1 \ -1] \left( \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \lambda_1 \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right) = 1$$

that is, choose  $\lambda_1 = 1$ . Therefore, (7.7c)

$$\boxed{\bar{v}_2 = \bar{v}_1 + \lambda_1 \bar{y} = [1 \ 0 \ 0 \ 0 \ 0]^T}$$



### Example (cont):

Form  $A_2$  from  $A_1$  replacing the 1st row ( $k' = 1$ ) of  $A_1$  by the 7th row ( $\ell = 7$ ) of  $A$ .

$$A_2 = \begin{bmatrix} 1 & -1 & 1 & -1 & -1 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

This move to vertex  $\bar{v}_2$  makes progress:

$$\underbrace{\begin{bmatrix} 1 & -1 & 1 & -1 & -1 \end{bmatrix}}_{\bar{c}^T} \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\bar{v}_1} = 0 < \underbrace{\begin{bmatrix} 1 & -1 & 1 & -1 & -1 \end{bmatrix}}_{\bar{c}^T} \underbrace{\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\bar{v}_2} = 1.$$

Now  $R = [7; 2; 3; 4; 5]$  (rows of  $A$  in  $A_2$ ).

### Example: $\Sigma_Q$ -formula (7.1)

$$F: x \geq 0 \wedge y \geq 0 \wedge x \geq 2 \wedge y \geq 2 \wedge x + y \leq 3,$$

or, in matrix form,

$$F: \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ -1 & 0 \\ 0 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \\ -2 \\ -2 \\ 3 \end{bmatrix}$$

Is  $F$   $T_Q$ -satisfiable?

#### Step 0

Because  $x$  and  $y$  are already constrained to be nonnegative, we do not need to introduce new  $x_1, x_2, y_1, y_2$ . Rewrite:

$$\underbrace{\begin{bmatrix} 1 & 1 \end{bmatrix}}_{D_1} \begin{bmatrix} x \\ y \end{bmatrix} \leq \underbrace{\begin{bmatrix} 3 \end{bmatrix}}_{\bar{e}_1} \quad \text{and} \quad \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{D_2} \begin{bmatrix} x \\ y \end{bmatrix} \geq \underbrace{\begin{bmatrix} 2 \\ 2 \end{bmatrix}}_{\bar{e}_2}$$

so that  $\bar{e}_1 \geq 0$  and  $\bar{e}_2 > 0$ .

### Example (cont):

Solve

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 0 & -1 \end{bmatrix}}_{A_2^T} \bar{u}_2 = \underbrace{\begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \\ -1 \end{bmatrix}}_{\bar{c}}$$

for  $\bar{u}_2$  yielding  $\bar{u}_2 = [1 \ 0 \ 0 \ 0]^T$ . Since  $\bar{u}_2 \geq 0$ , we are in **Case 1**: we have found an optimum point,  $\bar{v}_2$ , with optimal value 1.

Since we have that  $v_f = \bar{1}^T \bar{e}_2 = 1$ , the equality of the optimal point and  $v_f$  implies that

$$F: x + y \geq 1 \wedge x - y \geq -1$$

is  $T_Q$ -satisfiable. In particular, extract from

$$\begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \\ z \end{bmatrix} = \bar{v}_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

the assignment

$$x = x_1 - x_2 = 1 - 0 = 1 \quad \text{and} \quad y = y_1 - y_2 = 0 - 0 = 0$$

### Example (cont)

Then (7.2):

$$\begin{aligned} \max \quad & \bar{1}^T (D_2 \bar{x} - \bar{z}) \\ \text{subject to} \quad & \bar{x}, \bar{z} \geq \bar{0} \\ & D_1 \bar{x} \leq \bar{e}_1 \\ & D_2 \bar{x} - \bar{z} \leq \bar{e}_2 \end{aligned}$$

Expanding, we have

$$\begin{aligned} \bar{c}^T \bar{x} &= \bar{1}^T [D_2 \quad -I] \begin{bmatrix} x \\ y \\ z_1 \\ z_2 \end{bmatrix} = [1 \ 1] \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z_1 \\ z_2 \end{bmatrix} \\ &= \underbrace{\begin{bmatrix} 1 & 1 & -1 & -1 \end{bmatrix}}_{\bar{c}^T} \begin{bmatrix} x \\ y \\ z_1 \\ z_2 \end{bmatrix} \end{aligned}$$

Example (cont):

obtaining the optimization problem (7.3)

$$\max \underbrace{[1 \ 1 \ -1 \ -1]}_{\bar{c}^T} \begin{bmatrix} x \\ y \\ z_1 \\ z_2 \end{bmatrix}$$

subject to

$$\underbrace{\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}}_A \begin{bmatrix} x \\ y \\ z_1 \\ z_2 \end{bmatrix} \leq \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 3 \\ 2 \\ 2 \end{bmatrix}}_{\bar{b}}$$



Example (cont):

**Step 1**

Choose rows  $R = [1; 2; 3; 4]$  of  $A$  and  $\bar{b}$ , giving

$$\underbrace{\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}}_{A_1} \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\bar{v}_1} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\bar{b}_1}$$

Solving

$$\underbrace{\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}}_{A_1^T} \bar{u}_1 = \underbrace{\begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}}_{\bar{c}}$$

yields  $\bar{u}_1 = [-1 \ -1 \ 1 \ 1]^T$ . Adding 0s for the rows not in  $R$  produces  $\bar{u}$ :

$$\bar{u} = [-1 \ -1 \ 1 \ 1 \ 0 \ 0 \ 0]^T$$



Example (cont):

Use the initial vertex

$$\bar{v}_1 = \begin{bmatrix} x \\ y \\ z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

in **Step 1**.

$F$  is satisfiable iff the optimal value  $v_F$  is equal to

$$\bar{1}^T \bar{e}_2 = [1 \ 1] \begin{bmatrix} 2 \\ 2 \end{bmatrix} = 4.$$

We use the simplex algorithm to find the optimum.

Example (cont):

Since  $u_1, u_2 < 0$ , we are in Case 2 with  $k = k' = 1$ . Let  $\bar{v}$  be the first column of  $-A_1^{-1}$ : solve

$$\underbrace{\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}}_{A_1} \bar{v} = \underbrace{\begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{-\bar{e}_1}$$

to yield  $\bar{v} = [1 \ 0 \ 0 \ 0]^T$ . Then  $S = [5; 6]$ ; i.e., the 5th and 6th rows  $\bar{a}$  of  $A$  are such that  $\bar{a}\bar{v} > 0$ . Choose the largest  $\lambda_1$  such that

$$A(\bar{v}_1 + \lambda_1 \bar{v}) \leq \bar{b}$$



Example (cont):

Focusing on the 5th and 6th rows of  $A$  (since  $S' = [5; 6]$ ), choose the largest  $\lambda_1$  such that

$$\underbrace{\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & -1 & 0 \end{bmatrix}}_{\text{rows 5,6 of } A} \left( \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\bar{v}_1} + \lambda_1 \underbrace{\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\bar{y}} \right) \leq \underbrace{\begin{bmatrix} 3 \\ 2 \end{bmatrix}}_{\text{rows 5,6 of } \bar{b}}$$

Namely, choose  $\lambda_1 = 2$  (and  $\ell = 6$ ). Then

$$\bar{v}_2 = \bar{v}_1 + \lambda_1 \bar{y} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + 2 \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$



Example (cont):

**Step 2**

Now  $R = [6; 2; 3; 4]$  (the indices of rows of  $A$  in  $A_2$ ). Solve

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}}_{A_2^T} \bar{w}_2 = \underbrace{\begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}}_{\bar{c}}$$

to yield  $\bar{w}_2 = \begin{bmatrix} 1/6 & -1/2 & 0 & 1/3 & 1/4 \end{bmatrix}^T$ . Then filling in 0s for the other rows of  $A$  produces:

$$\bar{w} = \begin{bmatrix} 0 & -1/2 & 0 & 1/3 & 0 & 1/4 & 0 \end{bmatrix}^T$$

$u_2 < 0$ , so  $k = 2$ , which corresponds to row  $k' = 2$  of  $\bar{w}_2$ .

According to Case 2, let  $\bar{y}$  be the 2nd column of  $-A_2^{-1}$ : solve

$A_2 \bar{y} = -e_2$  to yield  $\bar{y} = [0 \ 1 \ 0 \ 0]^T$ . Then the 5th and 7th rows  $\bar{w}$  of  $A$  are such that  $\bar{w} > 0$  so that  $S = [5; 7]$ .



Example (cont):

Replace the 1st row of  $A_1$  (since  $k' = 1$ ) by the 6th row of  $A$  (since  $\ell = 6$ ) to produce

$$A_2 = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \quad \text{and} \quad \bar{b}_2 = \begin{bmatrix} 2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Have we made progress? Yes, for

$$\bar{c}^T \bar{v}_1 = 0 < 2 = \bar{c}^T \bar{v}_2.$$

The objective function has increased from 0 to 2.



Example (cont):

Focusing on the 5th and 7th rows of  $A$ , choose the largest  $\lambda_2$  such that

$$\underbrace{\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}}_{\text{rows 5,7 of } A} \left( \underbrace{\begin{bmatrix} 2 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\bar{v}_2} + \lambda_2 \underbrace{\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}}_{\bar{y}} \right) \leq \underbrace{\begin{bmatrix} 3 \\ 2 \end{bmatrix}}_{\text{rows 5,7 of } \bar{b}}$$

Choose  $\lambda_2 = 1$  (and  $\ell = 5$ ). Then

$$\bar{v}_3 = \bar{v}_2 + \lambda_2 \bar{y} = \begin{bmatrix} 2 \\ 0 \\ 0 \\ 0 \end{bmatrix} + 1 \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$



### Example (cont):

Replace the 2nd row of  $A_2$  (since  $k' = 2$ ) by the 5th row of  $A$  (since  $\ell = 5$ ) to produce

$$A_3 = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \quad \text{and} \quad \bar{b}_3 = \begin{bmatrix} 2 \\ 3 \\ 0 \\ 0 \end{bmatrix}$$

Have we made progress? Yes, for

$$\bar{c}^T \bar{v}_1 = 0 < \bar{c}^T \bar{v}_2 = 2 < \bar{c}^T \bar{v}_3 = 3.$$

The objective function has increased from 2 to 3.

### Example (cont):

#### Step 3

Now  $R = [6; 5; 3; 4]$ . Solve  $A_3^T \bar{v}_3 = \bar{c}$ , yielding  $\bar{v}_3 = [0 \ 1 \ 1 \ 1]^T$ .  
Now  $\bar{v}_3 \geq \bar{0}$ , so we are in Case 1:  $\bar{v}_3$  is the optimum with objective value

$$\underbrace{[1 \ 1 \ -1 \ -1]}_{\bar{c}^T} \underbrace{\begin{bmatrix} 2 \\ 1 \\ 0 \\ 0 \end{bmatrix}}_{\bar{v}_3} = 3.$$

#### Final Step: Satisfiability

The optimal value of the constructed optimization problem is 3, which is less than the required  $v_F = 4$  of **Step 0**. Hence,  $F$  is  $T_Q$ -unsatisfiable.

## 9. Quantifier-free Equality and Data Structures

### The Theory of Equality $T_E$

$$\Sigma_E: \{=, a, b, c, \dots, f, g, h, \dots, p, q, r, \dots\}$$

uninterpreted symbols:

- constants  $a, b, c, \dots$
- functions  $f, g, h, \dots$
- predicates  $p, q, r, \dots$

#### Example:

$$x = y \wedge f(x) \neq f(y) \quad T_E\text{-unsatisfiable}$$

$$f(x) = f(y) \wedge x \neq y \quad T_E\text{-unsatisfiable}$$

$$f(f(f(a))) = a \wedge f(f(f(f(a)))) = a \wedge f(a) \neq a \quad T_E\text{-unsatisfiable}$$

### Axioms of $T_E$

- $\forall x. x = x$  (reflexivity)
- $\forall x, y. x = y \rightarrow y = x$  (symmetry)
- $\forall x, y, z. x = y \wedge y = z \rightarrow x = z$  (transitivity)

define = to be an equivalence relation.

Axiom schema

- for each positive integer  $n$  and  $n$ -ary function symbol  $f$ ,
 
$$\forall x_1, \dots, x_n, y_1, \dots, y_n. \bigwedge_i x_i = y_i \rightarrow f(x_1, \dots, x_n) = f(y_1, \dots, y_n)$$
 (congruence)

For example,

$$\forall x, y. x = y \rightarrow f(x) = f(y)$$

Then

$$x = g(y, z) \rightarrow f(x) = f(g(y, z))$$

is  $T_E$ -valid.



### We discuss $T_E$ -formulae without predicates

For example, for  $\Sigma_E$ -formula

$$F : p(x) \wedge q(x, y) \wedge q(y, z) \rightarrow \neg q(x, z)$$

introduce fresh constant  $\bullet$  and fresh functions  $f_p$  and  $f_g$ , and transform  $F$  to

$$G : f_p(x) = \bullet \wedge f_q(x, y) = \bullet \wedge f_q(y, z) = \bullet \rightarrow f_q(x, z) \neq \bullet.$$



Axiom schema

- for each positive integer  $n$  and  $n$ -ary predicate symbol  $p$ ,

$$\forall x_1, \dots, x_n, y_1, \dots, y_n. \bigwedge_i x_i = y_i \rightarrow (p(x_1, \dots, x_n) \leftrightarrow p(y_1, \dots, y_n))$$
 (equivalence)

Thus,

$$x = y \rightarrow (p(x) \leftrightarrow p(y))$$

is  $T_E$ -valid.

## Equivalence and Congruence Relations: Basics

Binary relation  $R$  over set  $S$

- is an equivalence relation if
  - ▶ reflexive:  $\forall s \in S. sRs$ ;
  - ▶ symmetric:  $\forall s_1, s_2 \in S. s_1Rs_2 \rightarrow s_2Rs_1$ ;
  - ▶ transitive:  $\forall s_1, s_2, s_3 \in S. s_1Rs_2 \wedge s_2Rs_3 \rightarrow s_1Rs_3$ .

Example:

Define the binary relation  $\equiv_2$  over the set  $\mathbb{Z}$  of integers

$$m \equiv_2 n \text{ iff } (m \bmod 2) = (n \bmod 2)$$

That is,  $m, n \in \mathbb{Z}$  are related iff they are both even or both odd.

$\equiv_2$  is an equivalence relation

- is a congruence relation if in addition

$$\forall \vec{s}, \vec{t}. \bigwedge_{i=1}^n s_i R t_i \rightarrow f(\vec{s}) R f(\vec{t}).$$



## Classes

For  $\left\{ \begin{array}{l} \text{equivalence} \\ \text{congruence} \end{array} \right\}$  relation  $R$  over set  $S$ ,  
 The  $\left\{ \begin{array}{l} \text{equivalence} \\ \text{congruence} \end{array} \right\}$  class of  $s \in S$  under  $R$  is

$$[s]_R \stackrel{\text{def}}{=} \{s' \in S : sRs'\}.$$

### Example:

The equivalence class of 3 under  $\equiv_2$  over  $\mathbb{Z}$  is

$$[3]_{\equiv_2} = \{n \in \mathbb{Z} : n \text{ is odd}\}.$$

## Partitions

A partition  $P$  of  $S$  is a set of subsets of  $S$  that is

- ▶ total  $\left( \bigcup_{S' \in P} S' \right) = S$
- ▶ disjoint  $\forall S_1, S_2 \in P. S_1 \cap S_2 = \emptyset$

## Refinements

Two binary relations  $R_1$  and  $R_2$  over set  $S$ .  
 $R_1$  is refinement of  $R_2$ ,  $R_1 \prec R_2$ , if

$$\forall s_1, s_2 \in S. s_1 R_1 s_2 \rightarrow s_1 R_2 s_2.$$

$R_1$  refines  $R_2$ .

### Examples:

- ▶ For  $S = \{a, b\}$ ,  
 $R_1 : \{aR_1b\}$      $R_2 : \{aR_2b, bR_2a\}$   
 Then  $R_1 \prec R_2$
- ▶ For set  $S$ ,  
 $R_1$  induced by the partition  $P_1 : \{\{s\} : s \in S\}$   
 $R_2$  induced by the partition  $P_2 : \{S\}$   
 Then  $R_1 \prec R_2$ .
- ▶ For set  $\mathbb{Z}$   
 $R_1 : \{xR_1y : x \bmod 2 = y \bmod 2\}$   
 $R_2 : \{xR_2y : x \bmod 4 = y \bmod 4\}$   
 Then  $R_2 \prec R_1$ .

## Quotient

The quotient  $S/R$  of  $S$  by  $\left\{ \begin{array}{l} \text{equivalence} \\ \text{congruence} \end{array} \right\}$  relation  $R$  is the set of  
 $\left\{ \begin{array}{l} \text{equivalence} \\ \text{congruence} \end{array} \right\}$  classes

$$S/R = \{[s]_R : s \in S\}.$$

It is a partition

Example: The quotient  $\mathbb{Z}/\equiv_2$  is a partition of  $\mathbb{Z}$ . The set of equivalence classes

$$\{\{n \in \mathbb{Z} : n \text{ is odd}\}, \{n \in \mathbb{Z} : n \text{ is even}\}\}$$

Note duality between relations and classes

## Closures

Given binary relation  $R$  over  $S$ .

The equivalence closure  $R^E$  of  $R$  is the equivalence relation s.t.

- ▶  $R$  refines  $R^E$ , i.e.  $R \prec R^E$ ;
- ▶ for all other equivalence relations  $R'$  s.t.  $R \prec R'$ ,  
 either  $R' = R^E$  or  $R^E \prec R'$

That is,  $R^E$  is the "smallest" equivalence relation that "covers"  $R$ .

Example: If  $S = \{a, b, c, d\}$  and  $R = \{aRb, bRc, dRd\}$ , then

- $aRb, bRc, dRd \in R^E$  since  $R \subseteq R^E$ ;
- $aRa, bRb, cRc \in R^E$  by reflexivity;
- $bRa, cRb \in R^E$  by symmetry;
- $aRc \in R^E$  by transitivity;
- $cRa \in R^E$  by symmetry.

Hence,

$$R^E = \{aRb, bRa, aRa, bRb, bRc, cRb, cRc, aRc, cRa, dRd\}.$$

Similarly, the congruence closure  $R^C$  of  $R$  is the "smallest" congruence relation that "covers"  $R$ .

## Congruence Closure Algorithm

Given  $\Sigma_E$ -formula

$$F : s_1 = t_1 \wedge \dots \wedge s_m = t_m \wedge s_{m+1} \neq t_{m+1} \wedge \dots \wedge s_n \neq t_n$$

decide if  $F$  is  $\Sigma_E$ -satisfiable.

**Definition:** For  $\Sigma_E$ -formula  $F$ , the subterm set  $S_F$  of  $F$  is the set that contains precisely the subterms of  $F$ .

**Example:** The subterm set of

$$F : f(a, b) = a \wedge f(f(a, b), b) \neq a$$

is

$$S_F = \{a, b, f(a, b), f(f(a, b), b)\}.$$

$$F : \underbrace{s_1 = t_1 \wedge \dots \wedge s_m = t_m}_{\text{generate congruence closure}} \wedge \underbrace{s_{m+1} \neq t_{m+1} \wedge \dots \wedge s_n \neq t_n}_{\text{search for contradiction}}$$

The algorithm performs the following steps:

1. Construct the congruence closure  $\sim$  of

$$\{s_1 = t_1, \dots, s_m = t_m\}$$

over the subterm set  $S_F$ . Then

$$\sim \models s_1 = t_1 \wedge \dots \wedge s_m = t_m.$$

2. If for any  $i \in \{m+1, \dots, n\}$ ,  $s_i \sim t_i$ , return unsatisfiable.
3. Otherwise,  $\sim \models F$ , so return satisfiable.

How do we actually construct the congruence closure in Step 1?

## The Algorithm

Given  $\Sigma_E$ -formula  $F$

$$F : s_1 = t_1 \wedge \dots \wedge s_m = t_m \wedge s_{m+1} \neq t_{m+1} \wedge \dots \wedge s_n \neq t_n$$

with subterm set  $S_F$ ,  $F$  is  $T_E$ -satisfiable iff there exists a congruence relation  $\sim$  over  $S_F$  such that

- ▶ for each  $i \in \{1, \dots, m\}$ ,  $s_i \sim t_i$ ;
- ▶ for each  $i \in \{m+1, \dots, n\}$ ,  $s_i \not\sim t_i$ .

Such congruence relation  $\sim$  defines  $T_E$ -interpretation  $I : (D_i, \alpha_i)$  of  $F$ .  $D_i$  consists of  $|S_F / \sim|$  elements, one for each congruence class of  $S_F$  under  $\sim$ .

Instead of writing  $I \models F$  for this  $T_E$ -interpretation, we abbreviate  $\sim \models F$

The goal of the algorithm is to construct the congruence relation of  $S_F$ , or to prove that no congruence relation exists.

Initially, begin with the finest congruence relation  $\sim_0$  given by the partition

$$\{\{s\} : s \in S_F\}.$$

That is, let each term of  $S_F$  be its own congruence class.

Then, for each  $i \in \{1, \dots, m\}$ , impose  $s_i = t_i$  by merging the congruence classes

$$[s_i]_{\sim_{i-1}} \quad \text{and} \quad [t_i]_{\sim_{i-1}}$$

to form a new congruence relation  $\sim_i$ . To accomplish this merging,

- ▶ form the union of  $[s_i]_{\sim_{i-1}}$  and  $[t_i]_{\sim_{i-1}}$
- ▶ propagate any new congruences that arise within this union.

The new relation  $\sim_i$  is a congruence relation in which  $s_i \sim t_i$ .

Example: Given  $\Sigma_E$ -formula

$$F : f(a, b) = a \wedge f(f(a, b), b) \neq a$$

Construct initial partition by letting each member of the subterm set  $S_F$  be its own class:

1.  $\{\{a\}, \{b\}, \{f(a, b)\}, \{f(f(a, b), b)\}\}$

According to the first literal  $f(a, b) = a$ , merge

$$\{f(a, b)\} \text{ and } \{a\}$$

to form partition

2.  $\{\{a, f(a, b)\}, \{b\}, \{f(f(a, b), b)\}\}$

According to the (congruence) axiom,

$$f(a, b) \sim a, b \sim b \text{ implies } f(f(a, b), b) \sim f(a, b),$$

resulting in the new partition

3.  $\{\{a, f(a, b), f(f(a, b), b)\}, \{b\}\}$

This partition represents the congruence closure of  $S_F$ . Now, is it the case that

4.  $\{\{a, f(a, b), f(f(a, b), b)\}, \{b\}\} \models F?$

No, as  $f(f(a, b), b) \sim a$  but  $F$  asserts that  $f(f(a, b), b) \neq a$ .

Hence,  $F$  is  $T_E$ -unsatisfiable.

3.  $\{\{a, f^3(a)\}, \{f(a), f^4(a)\}, \{f^2(a), f^5(a)\}\}$ .

From the second literal,  $f^5(a) = a$ , merge

$$\{f^2(a), f^5(a)\} \text{ and } \{a, f^3(a)\}$$

to form the partition

4.  $\{\{a, f^2(a), f^3(a), f^5(a)\}, \{f(a), f^4(a)\}\}$ .

Propagating the congruence

$$f^3(a) \sim f^2(a) \Rightarrow f(f^3(a)) \sim f(f^2(a)) \text{ i.e. } f^4(a) \sim f^3(a)$$

yields the partition

5.  $\{\{a, f(a), f^2(a), f^3(a), f^4(a), f^5(a)\}\}$ ,

which represents the congruence closure in which all of  $S_F$  are equal. Now,

6.  $\{\{a, f(a), f^2(a), f^3(a), f^4(a), f^5(a)\}\} \models F?$

No, as  $f(a) \sim a$ , but  $F$  asserts that  $f(a) \neq a$ . Hence,  $F$  is  $T_E$ -unsatisfiable.

Example: Given  $\Sigma_E$ -formula

$$F : f(f(f(a))) = a \wedge f(f(f(f(a)))) = a \wedge f(a) \neq a$$

From the subterm set  $S_F$ , the initial partition is

1.  $\{\{a\}, \{f(a)\}, \{f^2(a)\}, \{f^3(a)\}, \{f^4(a)\}, \{f^5(a)\}\}$

where, for example,  $f^3(a)$  abbreviates  $f(f(f(a)))$ .

According to the literal  $f^3(a) = a$ , merge

$$\{f^3(a)\} \text{ and } \{a\}.$$

From the union,

2.  $\{\{a, f^3(a)\}, \{f(a)\}, \{f^2(a)\}, \{f^4(a)\}, \{f^5(a)\}\}$

deduce the following congruence propagations:

$$f^3(a) \sim a \Rightarrow f(f^3(a)) \sim f(a) \text{ i.e. } f^4(a) \sim f(a)$$

and

$$f^4(a) \sim f(a) \Rightarrow f(f^4(a)) \sim f(f(a)) \text{ i.e. } f^5(a) \sim f^2(a)$$

Thus, the final partition for this iteration is the following:

3.  $\{\{a, f^3(a)\}, \{f(a), f^4(a)\}, \{f^2(a), f^5(a)\}\}$ .

Example: Given  $\Sigma_E$ -formula

$$F : f(x) = f(y) \wedge x \neq y.$$

The subterm set  $S_F$  induces the following initial partition:

1.  $\{\{x\}, \{y\}, \{f(x)\}, \{f(y)\}\}$ .

Then  $f(x) = f(y)$  indicates to merge

$$\{f(x)\} \text{ and } \{f(y)\}.$$

The union  $\{f(x), f(y)\}$  does not yield any new congruences, so the final partition is

2.  $\{\{x\}, \{y\}, \{f(x), f(y)\}\}$ .

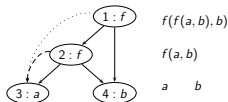
Does

3.  $\{\{x\}, \{y\}, \{f(x), f(y)\}\} \models F?$

Yes, as  $x \neq y$ , agreeing with  $x \neq y$ . Hence,  $F$  is  $T_E$ -satisfiable.

## Directed Acyclic Graph (DAG)

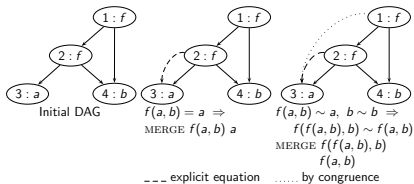
For  $\Sigma_E$ -formula  $F$ , graph-based data structure for representing the subterms of  $S_F$  (and congruence relation between them).



Efficient way for computing the congruence closure algorithm.

## $T_E$ -Satisfiability (Summary of idea)

$$f(a, b) = a \wedge f(f(a, b), b) \neq a$$



$$\text{FIND } f(f(a, b), b) = a = \text{FIND } a \left. \vphantom{\text{FIND}} \right\} \Rightarrow \text{Unsatisfiable}$$
  

$$f(f(a, b), b) \neq a$$



## DAG representation

```

type node = {
  id      : id      node's unique identification number
  fn      : string  constant or function name
  args    : id list list of function arguments
  mutable find : id  the representative of the congruence class
  mutable ccpair : id set if the node is the representative for its congruence class, then its ccpair (congruence closure parents) are all parents of nodes in its congruence class
}

```

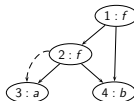


## DAG Representation of node 2

```

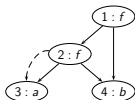
type node = {
  id      : id      ... 2
  fn      : string  ... f
  args    : id list ... [3,4]
  mutable find : id  ... 3
  mutable ccpair : id set ... {}
}

```



## DAG Representation of node 3

```
type node = {  
  id       : id    ... 3  
  fn       : string ... a  
  args     : idlist ... []  
  mutable find : id ... 3  
  mutable ccpair : idset ... {1,2}  
}
```



◀ ▶ 🔍 ↺ ↻

## UNION function

```
let UNION  $i_1$   $i_2$  =  
  let  $n_1$  = NODE (FIND  $i_1$ ) in  
  let  $n_2$  = NODE (FIND  $i_2$ ) in  
   $n_1$ .find ←  $n_2$ .find;  
   $n_2$ .ccpair ←  $n_1$ .ccpair  $\cup$   $n_2$ .ccpair;  
   $n_1$ .ccpair ←  $\emptyset$ 
```

$n_2$  is the representative of the union class

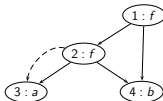
◀ ▶ 🔍 ↺ ↻

## The Implementation

### FIND function

returns the representative of node's congruence class

```
let rec FIND  $i$  =  
  let  $n$  = NODE  $i$  in  
  if  $n$ .find =  $i$  then  $i$  else FIND  $n$ .find
```



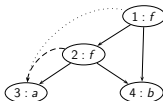
Example: FIND 2 = 3

FIND 3 = 3

3 is the representative of 2.

◀ ▶ 🔍 ↺ ↻

### Example



UNION 1 2     $n_1 = 1$      $n_2 = 3$

1.find ← 3

3.ccpair ← {1,2}

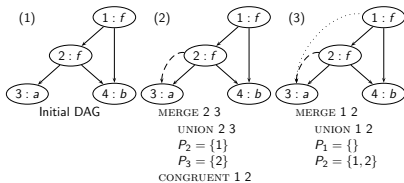
1.ccpair ←  $\emptyset$

◀ ▶ 🔍 ↺ ↻



### Example 1: $T_E$ -Satisfiability

$$f(a, b) = a \wedge f(f(a, b), b) \neq a$$



FIND  $f(f(a, b), b) = a =$  FIND  $a \Rightarrow$  **Unsatisfiable**

Given  $\Sigma_E$ -formula

$$F : f(a, b) = a \wedge f(f(a, b), b) \neq a .$$

The subterm set is

$$S_F = \{a, b, f(a, b), f(f(a, b), b)\} ,$$

resulting in the initial partition

$$(1) \{\{a\}, \{b\}, \{f(a, b)\}, \{f(f(a, b), b)\}\}$$

in which each term is its own congruence class. Fig (1).

Final partition

$$(2) \{\{a, f(a, b), f(f(a, b), b)\}, \{b\}\}$$

**Note:** dash edge ----- merge dictated by equalities in  $F$   
dotted edge ..... deduced merge

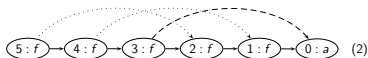
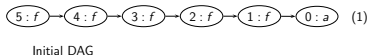
Does

$$(3) \{\{a, f(a, b), f(f(a, b), b)\}, \{b\}\} \models F ?$$

No, as  $f(f(a, b), b) \sim a$ , but  $F$  asserts that  $f(f(a, b), b) \neq a$ .  
Hence,  $F$  is  $T_E$ -unsatisfiable.

### Example 2: $T_E$ -Satisfiability

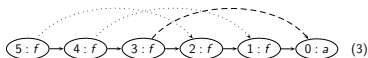
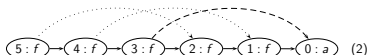
$$f(f(f(a))) = a \wedge f(f(f(f(f(a)))))) = a \wedge f(a) \neq a$$



$f(f(f(a))) = a \Rightarrow$  MERGE 3 0  $P_3 = \{4\}$   $P_0 = \{1\}$   
 $\Rightarrow$  MERGE 4 1  $P_4 = \{5\}$   $P_1 = \{2\}$   
 $\Rightarrow$  MERGE 5 2  $P_5 = \{1\}$   $P_2 = \{3\}$

### Example 2: $T_E$ -Satisfiability

$$f(f(f(a))) = a \wedge f(f(f(f(f(a)))))) = a \wedge f(a) \neq a$$



$f(f(f(f(f(a)))))) = a \Rightarrow$  MERGE 5 0  $P_5 = \{3\}$   $P_0 = \{1, 4\}$   
 $\Rightarrow$  MERGE 3 1 STOP. Why?

FIND  $f(a) = f(a) =$  FIND  $a \Rightarrow$  **Unsatisfiable**

Given  $\Sigma_E$ -formula

$$F : f(f(f(a))) = a \wedge f(f(f(f(f(a)))))) = a \wedge f(a) \neq a,$$

which induces the initial partition

1.  $\{\{a\}, \{f(a)\}, \{f^2(a)\}, \{f^3(a)\}, \{f^4(a)\}, \{f^5(a)\}\}$  .

The equality  $f^3(a) = a$  induces the partition

2.  $\{\{a, f^3(a)\}, \{f(a), f^4(a)\}, \{f^2(a), f^5(a)\}\}$  .

The equality  $f^5(a) = a$  induces the partition

3.  $\{\{a, f(a), f^2(a), f^3(a), f^4(a), f^5(a)\}\}$  .

Now, does

$$\{\{a, f(a), f^2(a), f^3(a), f^4(a), f^5(a)\}\} \models F ?$$

No, as  $f(a) \sim a$ , but  $F$  asserts that  $f(a) \neq a$ . Hence,  $F$  is  $T_E$ -unsatisfiable.

### Theorem (Sound and Complete)

Quantifier-free conjunctive  $\Sigma_E$ -formula  $F$  is  $T_E$ -satisfiable iff the congruence closure algorithm returns satisfiable.

## Recursive Data Structures

### Quantifier-free Theory of Lists $T_{\text{cons}}$

$$\Sigma_{\text{cons}} : \{\text{cons}, \text{car}, \text{cdr}, \text{atom}, =\}$$

- constructor  $\text{cons}$  :  $\text{cons}(a, b)$  list constructed by prepending  $a$  to  $b$
- left projector  $\text{car}$  :  $\text{car}(\text{cons}(a, b)) = a$
- right projector  $\text{cdr}$  :  $\text{cdr}(\text{cons}(a, b)) = b$
- atom : unary predicate

### Axioms of $T_{\text{cons}}$

- ▶ reflexivity, symmetry, transitivity
- ▶ congruence axioms:

$$\begin{aligned} \forall x_1, x_2, y_1, y_2. x_1 = x_2 \wedge y_1 = y_2 &\rightarrow \text{cons}(x_1, y_1) = \text{cons}(x_2, y_2) \\ \forall x, y. x = y &\rightarrow \text{car}(x) = \text{car}(y) \\ \forall x, y. x = y &\rightarrow \text{cdr}(x) = \text{cdr}(y) \end{aligned}$$

- ▶ equivalence axiom:

$$\forall x, y. x = y \rightarrow (\text{atom}(x) \leftrightarrow \text{atom}(y))$$



- (A1)  $\forall x, y. \text{car}(\text{cons}(x, y)) = x$  (left projection)
- (A2)  $\forall x, y. \text{cdr}(\text{cons}(x, y)) = y$  (right projection)
- (A3)  $\forall x. \neg \text{atom}(x) \rightarrow \text{cons}(\text{car}(x), \text{cdr}(x)) = x$  (construction)
- (A4)  $\forall x, y. \neg \text{atom}(\text{cons}(x, y))$  (atom)

## Simplifications

- Consider only quantifier-free conjunctive  $\Sigma_{\text{cons}}$ -formulae.  
Convert non-conjunctive formula to DNF and check each disjunct.
- $\neg \text{atom}(u_i)$  literals are removed:  

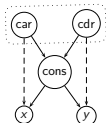
replace  $\neg \text{atom}(u_i)$  with  $u_i = \text{cons}(u_i^1, u_i^2)$

 by the (construction) axiom.
- Because of similarity to  $\Sigma_{=}$ , we sometimes combine  $\Sigma_{\text{cons}} \cup \Sigma_{=}$ .



## Algorithm: $T_{\text{cons}}$ -Satisfiability

- Construct the initial DAG for  $S_F$
- for each node  $n$  with  $n.\text{fn} = \text{cons}$ 
  - add  $\text{car}(n)$  and MERGE  $\text{car}(n)$   $n.\text{args}[1]$
  - add  $\text{cdr}(n)$  and MERGE  $\text{cdr}(n)$   $n.\text{args}[2]$
 by axioms (A1), (A2)
- for  $1 \leq i \leq m$ , MERGE  $s_i$   $t_i$
- for  $m+1 \leq i \leq n$ , if FIND  $s_i = \text{FIND } t_i$ , return **unsatisfiable**
- for  $1 \leq i \leq \ell$ , if  $\exists v$ . FIND  $v = \text{FIND } u_i \wedge v.\text{fn} = \text{cons}$ , return **unsatisfiable**
- Otherwise, return **satisfiable**



## Algorithm: $T_{\text{cons}}$ -Satisfiability (the idea)

$$\begin{aligned}
 F : & \quad \underbrace{s_1 = t_1 \wedge \cdots \wedge s_m = t_m}_{\text{generate congruence closure}} \\
 & \quad \wedge \underbrace{s_{m+1} \neq t_{m+1} \wedge \cdots \wedge s_n \neq t_n}_{\text{search for contradiction}} \\
 & \quad \wedge \underbrace{\text{atom}(u_1) \wedge \cdots \wedge \text{atom}(u_\ell)}_{\text{search for contradiction}}
 \end{aligned}$$

where  $s_i$ ,  $t_i$ , and  $u_j$  are  $T_{\text{cons}}$ -terms



## Example:

Given  $(\Sigma_{\text{cons}} \cup \Sigma_{=})$ -formula

$$\begin{aligned}
 F : & \quad \text{car}(x) = \text{car}(y) \wedge \text{cdr}(x) = \text{cdr}(y) \\
 & \quad \wedge \neg \text{atom}(x) \wedge \neg \text{atom}(y) \wedge f(x) \neq f(y)
 \end{aligned}$$

where the function symbol  $f$  is in  $\Sigma_{=}$

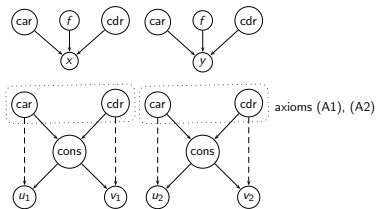
$$\begin{aligned}
 \text{car}(x) &= \text{car}(y) & \wedge & & (1) \\
 \text{cdr}(x) &= \text{cdr}(y) & \wedge & & (2) \\
 F' : \quad x &= \text{cons}(u_1, v_1) & \wedge & & (3) \\
 y &= \text{cons}(u_2, v_2) & \wedge & & (4) \\
 f(x) &\neq f(y) & & & (5)
 \end{aligned}$$

Recall the projection axioms:

$$\begin{aligned}
 (A1) \quad & \forall x, y. \text{car}(\text{cons}(x, y)) = x \\
 (A2) \quad & \forall x, y. \text{cdr}(\text{cons}(x, y)) = y
 \end{aligned}$$

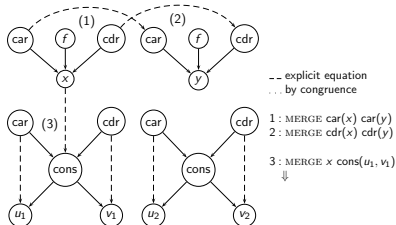


### Example(cont): Initial DAG



◁ ▷ ↻ 🔍 🔄

### Example(cont): MERGE

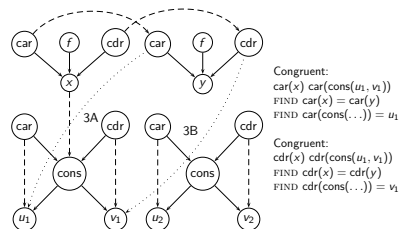


-- explicit equation  
... by congruence

- 1: MERGE car(x) car(y)
  - 2: MERGE cdr(x) cdr(y)
  - 3: MERGE x cons(u<sub>1</sub>, v<sub>1</sub>)
- ⇓

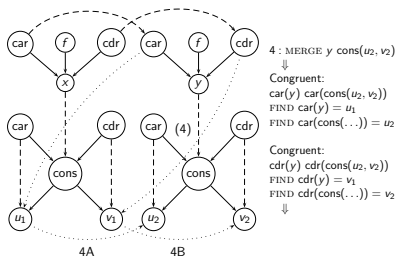
◁ ▷ ↻ 🔍 🔄

### Example(cont): Propagation



◁ ▷ ↻ 🔍 🔄

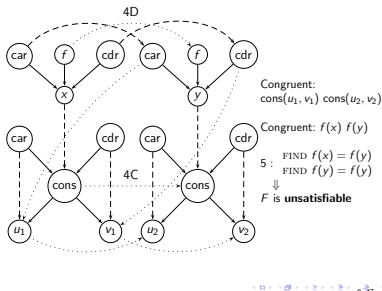
### Example(cont): MERGE



- 4: MERGE y cons(u<sub>2</sub>, v<sub>2</sub>)
- ⇓
- Congruent:  
car(y) car(cons(u<sub>2</sub>, v<sub>2</sub>))  
FIND car(y) = u<sub>1</sub>  
FIND car(cons(...)) = u<sub>2</sub>
- Congruent:  
cdr(y) cdr(cons(u<sub>2</sub>, v<sub>2</sub>))  
FIND cdr(y) = v<sub>1</sub>  
FIND cdr(cons(...)) = v<sub>2</sub>
- ⇓

◁ ▷ ↻ 🔍 🔄

## Example(cont): CONGRUENCE



## The Decision Procedure

Given quantifier-free conjunctive  $\Sigma_A$ -formula  $F$ .  
 To decide the  $T_A$ -satisfiability of  $F$ :

### Step 1

If  $F$  does not contain any write terms  $a(i \triangleleft v)$ , then

1. associate array variables  $a$  with fresh function symbol  $f_a$ , and replace read terms  $a[i]$  with  $f_a(i)$ ;
2. decide the  $T_{=}$ -satisfiability of the resulting formula.

## Arrays

### (1) Quantifier-free Fragment of $T_A$

$$\Sigma_A : \{ \cdot[\cdot], \cdot(\cdot \triangleleft \cdot), = \},$$

where

- ▶  $a[i]$  is a binary function representing read of array  $a$  at index  $i$ ;
- ▶  $a(i \triangleleft v)$  is a ternary function representing write of value  $v$  to index  $i$  of array  $a$ ;
- ▶  $=$  is a binary predicate.

Axioms of  $T_A$ :

1. axioms of (reflexivity), (symmetry), and (transitivity) of  $T_{=}$
2.  $\forall a, i, j. i = j \rightarrow a[i] = a[j]$  (array congruence)
3.  $\forall a, v, i, j. i = j \rightarrow a(i \triangleleft v)[j] = v$  (read-over-write 1)
4.  $\forall a, v, i, j. i \neq j \rightarrow a(i \triangleleft v)[j] = a[j]$  (read-over-write 2)

**Note:**  $a$  may itself be a write term, e.g.,  $a(i' \triangleleft v')$ . Then  $(a(i' \triangleleft v'))(i \triangleleft v)$

means: first write the value  $v'$  to index  $i'$  of  $a$   
 then write the value  $v$  to index  $i$  of  $a$

### Step 2

Select some read-over-write term  $a(i \triangleleft v)[j]$  (note that  $a$  may itself be a write term) and split on two cases:

1. According to (read-over-write 1), replace

$$F[a(i \triangleleft v)[j]] \quad \text{with} \quad F_1 : F[v] \wedge i = j,$$

and recurse on  $F_1$ . If  $F_1$  is found to be  $T_A$ -satisfiable, return satisfiable.

2. According to (read-over-write 2), replace

$$F[a(i \triangleleft v)[j]] \quad \text{with} \quad F_2 : F[a[j]] \wedge i \neq j,$$

and recurse on  $F_2$ . If  $F_2$  is found to be  $T_A$ -satisfiable, return satisfiable.

If both  $F_1$  and  $F_2$  are found to be  $T_A$ -unsatisfiable, return unsatisfiable.

Example: Consider  $\Sigma_A$ -formula

$$F : i_1 = j \wedge i_1 \neq i_2 \wedge a[j] = v_1 \wedge a(i_1 \triangleleft v_1)(i_2 \triangleleft v_2)[j] \neq a[j] .$$

$F$  contains a write term,

$$a(i_1 \triangleleft v_1)(i_2 \triangleleft v_2)[j] \neq a[j] .$$

According to (read-over-write 1), assume  $i_2 = j$  and recurse on

$$F_1 : i_2 = j \wedge i_1 = j \wedge i_1 \neq i_2 \wedge a[j] = v_1 \wedge v_2 \neq a[j] .$$

$F_1$  does not contain any write terms, so rewrite it to

$$F'_1 : i_2 = j \wedge i_1 = j \wedge i_1 \neq i_2 \wedge f_a(j) = v_1 \wedge v_2 \neq f_a(j) .$$

The first two literals imply that  $i_1 = i_2$ , contradicting the third literal, so  $F'_1$  is  $T_{=}$ -unsatisfiable.

## 10. Combining Decision Procedures

Returning, we try the second case:

according to (read-over-write 2), assume  $i_2 \neq j$  and recurse on

$$F_2 : i_2 \neq j \wedge i_1 = j \wedge i_1 \neq i_2 \wedge a[j] = v_1 \wedge a(i_1 \triangleleft v_1)[j] \neq a[j] .$$

$F_2$  contains a write term. According to (read-over-write 1), assume  $i_1 = j$  and recurse on

$$F_3 : i_1 = j \wedge i_2 \neq j \wedge i_1 = j \wedge i_1 \neq i_2 \wedge a[j] = v_1 \wedge v_1 \neq a[j] .$$

Contradiction because of the final two terms. Thus, according to (read-over-write 2), assume  $i_1 \neq j$  and recurse on

$$F_4 : i_1 \neq j \wedge i_2 \neq j \wedge i_1 = j \wedge i_1 \neq i_2 \wedge a[j] = v_1 \wedge a[j] \neq a[j] .$$

Two contradictions: the first and third literals contradict each other, and the final literal is contradictory. As all branches have been tried,  $F$  is  $T_A$ -unsatisfiable.

Suppose instead that  $F$  does not contain the literal  $i_1 \neq i_2$ . Is this new formula  $T_A$ -satisfiable?

### Combining Decision Procedures: Nelson-Oppen Method

#### Given

Theories  $T_i$  over signatures  $\Sigma_i$

(constants, functions, predicates)

with corresponding decision procedures  $P_i$  for  $T_i$ -satisfiability.

#### Goal

Decide satisfiability of a sentence in theory  $\cup_j T_j$ .

**Example:** How do we show that

$$F : 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1) \wedge f(x) \neq f(2)$$

is  $(T_E \cup T_Z)$ -unsatisfiable?



Example: Theory of partial orders

$\Sigma_{\preceq}$ -theory  $T_{\preceq}$

$$\Sigma_{\preceq} : \{\preceq, =\}$$

where  $\preceq$  is a binary predicate.

Axioms

- $\forall x. x \preceq x$  ( $\preceq$  reflexivity)
- $\forall x, y. x \preceq y \wedge y \preceq x \rightarrow x = y$  ( $\preceq$  antisymmetry)
- $\forall x, y, z. x \preceq y \wedge y \preceq z \rightarrow x \preceq z$  ( $\preceq$  transitivity)

We prove  $T_{\preceq}$  is stably infinite.

Consider  $T_{\preceq}$ -satisfiable quantifier-free  $\Sigma_{\preceq}$ -formula  $F$ .

Consider arbitrary satisfying  $T_{\preceq}$ -interpretation  $I : (D_I, \alpha_I)$ ,

where  $\alpha_I$  maps  $\preceq$  to  $\leq_I$ .

- ▶ Let  $A$  be any infinite set disjoint from  $D_I$
- ▶ Construct new interpretation  $J : (D_J, \alpha_J)$ 
  - ▶  $D_J = D_I \cup A$
  - ▶  $\alpha_J = \{\preceq \mapsto \leq_J\}$ , where for  $a, b \in D_J$ ,  
$$a \leq_J b \stackrel{\text{def}}{=} \begin{cases} a \leq_I b & \text{if } a, b \in D_I \\ a = b & \text{otherwise} \end{cases}$$

$J$  is  $T_{\preceq}$ -interpretation satisfying  $F$  with infinite domain.

Hence,  $T_{\preceq}$  is stably infinite.

Example: Consider quantifier-free conjunctive ( $\Sigma_E \cup \Sigma_Z$ )-formula

$$F : 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1) \wedge f(x) \neq f(2).$$

The signatures of  $T_E$  and  $T_Z$  only share  $=$ . Also, both theories are stably infinite. Hence, the NO combination of the decision procedures for  $T_E$  and  $T_Z$  decides the  $(T_E \cup T_Z)$ -satisfiability of  $F$ .

Intuitively,  $F$  is  $(T_E \cup T_Z)$ -unsatisfiable.

For the first two literals imply  $x = 1 \vee x = 2$  so that

$$f(x) = f(1) \vee f(x) = f(2).$$

Contradict last two literals.

Hence,  $F$  is  $(T_E \cup T_Z)$ -unsatisfiable.

## N-O Overview

Phase 1: Variable Abstraction

- ▶ Given conjunction  $\Gamma$  in theory  $T_1 \cup T_2$ .
- ▶ Convert to conjunction  $\Gamma_1 \cup \Gamma_2$  s.t.
  - ▶  $\Gamma_i$  in theory  $T_i$
  - ▶  $\Gamma_1 \cup \Gamma_2$  satisfiable iff  $\Gamma$  satisfiable.

Phase 2: Check

- ▶ If there is some set  $S$  of equalities and disequalities between the shared variables of  $\Gamma_1$  and  $\Gamma_2$   
shared( $\Gamma_1, \Gamma_2$ ) = free( $\Gamma_1$ )  $\cap$  free( $\Gamma_2$ )  
s.t.  $S \cup \Gamma_i$  are  $T_i$ -satisfiable for all  $i$ ,  
then  $\Gamma$  is **satisfiable**.
- ▶ Otherwise, **unsatisfiable**.

## Nelson-Oppen Method: Overview

Consider quantifier-free conjunctive  $(\Sigma_1 \cup \Sigma_2)$ -formula  $F$ .

Two versions:

- ▶ **nondeterministic** — simple to present, but high complexity
- ▶ **deterministic** — efficient

Nelson-Oppen (N-O) method proceeds in two steps:

- ▶ **Phase 1** (variable abstraction)
  - same for both versions
- ▶ **Phase 2**
  - nondeterministic: guess equalities/disequalities and check
  - deterministic: generate equalities/disequalities by equality propagation

## Phase 1: Variable abstraction

Given quantifier-free conjunctive  $(\Sigma_1 \cup \Sigma_2)$ -formula  $F$ .

Transform  $F$  into two quantifier-free conjunctive formulae

$$\Sigma_1\text{-formula } F_1 \quad \text{and} \quad \Sigma_2\text{-formula } F_2$$

s.t.  $F$  is  $(T_1 \cup T_2)$ -satisfiable iff  $F_1 \wedge F_2$  is  $(T_1 \cup T_2)$ -satisfiable  
 $F_1$  and  $F_2$  are linked via a set of shared variables.

For term  $t$ , let  $\text{hd}(t)$  be the root symbol, e.g.  $\text{hd}(f(x)) = f$ .

## Generation of $F_1$ and $F_2$

For  $i, j \in \{1, 2\}$  and  $i \neq j$ , repeat the transformations

- (1) if function  $f \in \Sigma_i$  and  $\text{hd}(t) \in \Sigma_j$ ,

$$F[f(t_1, \dots, t_n)] \Rightarrow F[f(t_1, \dots, w_1, \dots, t_n)] \wedge w = t$$

- (2) if predicate  $p \in \Sigma_i$  and  $\text{hd}(t) \in \Sigma_j$ ,

$$F[p(t_1, \dots, t_n)] \Rightarrow F[p(t_1, \dots, w_1, \dots, t_n)] \wedge w = t$$

- (3) if  $\text{hd}(s) \in \Sigma_i$  and  $\text{hd}(t) \in \Sigma_j$ ,

$$F[s = t] \Rightarrow F[\top] \wedge w = s \wedge w = t$$

- (4) if  $\text{hd}(s) \in \Sigma_i$  and  $\text{hd}(t) \in \Sigma_j$ ,

$$F[s \neq t] \Rightarrow F[w_1 \neq w_2] \wedge w_1 = s \wedge w_2 = t$$

where  $w$ ,  $w_1$ , and  $w_2$  are fresh variables.

Example: Consider  $(\Sigma_E \cup \Sigma_Z)$ -formula

$$F: 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1) \wedge f(x) \neq f(2).$$

According to transformation 1, since  $f \in \Sigma_E$  and  $1 \in \Sigma_Z$ , replace  $f(1)$  by  $f(w_1)$  and add  $w_1 = 1$ . Similarly, replace  $f(2)$  by  $f(w_2)$  and add  $w_2 = 2$ .

Now, the literals

$$\Gamma_Z: \{1 \leq x, x \leq 2, w_1 = 1, w_2 = 2\}$$

are  $T_Z$ -literals, while the literals

$$\Gamma_E: \{f(x) \neq f(w_1), f(x) \neq f(w_2)\}$$

are  $T_E$ -literals. Hence, construct the  $\Sigma_Z$ -formula

$$F_1: 1 \leq x \wedge x \leq 2 \wedge w_1 = 1 \wedge w_2 = 2$$

and the  $\Sigma_E$ -formula

$$F_2: f(x) \neq f(w_1) \wedge f(x) \neq f(w_2).$$

$F_1$  and  $F_2$  share the variables  $\{x, w_1, w_2\}$ .

$F_1 \wedge F_2$  is  $(T_E \cup T_Z)$ -equisatisfiable to  $F$ .

Example: Consider  $(\Sigma_E \cup \Sigma_Z)$ -formula

$$F: f(x) = x+y \wedge x \leq y+z \wedge x+z \leq y \wedge y = 1 \wedge f(x) \neq f(2).$$

In the first literal,  $\text{hd}(f(x)) = f \in \Sigma_{=}$  and  $\text{hd}(x+y) = + \in \Sigma_Z$ ; thus, by (3), replace the literal with

$$w_1 = f(x) \wedge w_1 = x+y.$$

In the final literal,  $f \in \Sigma_E$  but  $2 \in \Sigma_Z$ , so by (1), replace it with

$$f(x) \neq f(w_2) \wedge w_2 = 2.$$

Now, separating the literals results in two formulae:

$$F_1: w_1 = x+y \wedge x \leq y+z \wedge x+z \leq y \wedge y = 1 \wedge w_2 = 2$$

is a  $\Sigma_Z$ -formula, and

$$F_2: w_1 = f(x) \wedge f(x) \neq f(w_2)$$

is a  $\Sigma_E$ -formula.

The conjunction  $F_1 \wedge F_2$  is  $(T_E \cup T_Z)$ -equisatisfiable to  $F$ .



Example: Consider  $(\Sigma_E \cup \Sigma_Z)$ -formula

$$F: 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1) \wedge f(x) \neq f(2)$$

Phase 1 separates this formula into the  $\Sigma_Z$ -formula

$$F_1: 1 \leq x \wedge x \leq 2 \wedge w_1 = 1 \wedge w_2 = 2$$

and the  $\Sigma_E$ -formula

$$F_2: f(x) \neq f(w_1) \wedge f(x) \neq f(w_2)$$

with

$$V = \text{shared}(F_1, F_2) = \{x, w_1, w_2\}$$

There are 5 equivalence relations to consider, which we list by stating the partitions:



## Nondeterministic Version

### Phase 2: Guess and Check

- ▶ Phase 1 separated  $(\Sigma_1 \cup \Sigma_2)$ -formula  $F$  into two formulae:  $\Sigma_1$ -formula  $F_1$  and  $\Sigma_2$ -formula  $F_2$
- ▶  $F_1$  and  $F_2$  are linked by a set of shared variables:  
 $V = \text{shared}(F_1, F_2) = \text{free}(F_1) \cap \text{free}(F_2)$
- ▶ Let  $E$  be an equivalence relation over  $V$ .
- ▶ The arrangement  $\alpha(V, E)$  of  $V$  induced by  $E$  is:

$$\alpha(V, E): \bigwedge_{u,v \in V. uEv} u = v \wedge \bigwedge_{u,v \in V. \neg(uEv)} u \neq v$$

### Then,

the original formula  $F$  is  $(T_1 \cup T_2)$ -satisfiable iff

there exists an equivalence relation  $E$  of  $V$  s.t.

- (1)  $F_1 \wedge \alpha(V, E)$  is  $T_1$ -satisfiable, and
- (2)  $F_2 \wedge \alpha(V, E)$  is  $T_2$ -satisfiable.

Otherwise,  $F$  is  $(T_1 \cup T_2)$ -unsatisfiable.



- $\{\{x, w_1, w_2\}\}$ , i.e.,  $x = w_1 = w_2$ :  
 $x = w_1$  and  $f(x) \neq f(w_1) \Rightarrow F_2 \wedge \alpha(V, E)$  is  $T_E$ -unsatisfiable.
- $\{\{x, w_1\}, \{w_2\}\}$ , i.e.,  $x = w_1, x \neq w_2$ :  
 $x = w_1$  and  $f(x) \neq f(w_1) \Rightarrow F_2 \wedge \alpha(V, E)$  is  $T_E$ -unsatisfiable.
- $\{\{x, w_2\}, \{w_1\}\}$ , i.e.,  $x = w_2, x \neq w_1$ :  
 $x = w_2$  and  $f(x) \neq f(w_2) \Rightarrow F_2 \wedge \alpha(V, E)$  is  $T_E$ -unsatisfiable.
- $\{\{x\}, \{w_1, w_2\}\}$ , i.e.,  $x \neq w_1, w_1 = w_2$ :  
 $w_1 = w_2$  and  $w_1 = 1 \wedge w_2 = 2 \Rightarrow F_1 \wedge \alpha(V, E)$  is  $T_Z$ -unsatisfiable.
- $\{\{x\}, \{w_1\}, \{w_2\}\}$ , i.e.,  $x \neq w_1, x \neq w_2, w_1 \neq w_2$ :  
 $x \neq w_1 \wedge x \neq w_2$  and  $x = w_1 = 1 \vee x = w_2 = 2$   
(since  $1 \leq x \leq 2$  implies that  $x = 1 \vee x = 2$  in  $T_Z$ )  
 $\Rightarrow F_1 \wedge \alpha(V, E)$  is  $T_Z$ -unsatisfiable.

Hence,  $F$  is  $(T_E \cup T_Z)$ -unsatisfiable.



Example: Consider the  $(\Sigma_{\text{cons}} \cup \Sigma_Z)$ -formula

$$F : \text{car}(x) + \text{car}(y) = z \wedge \text{cons}(x, z) \neq \text{cons}(y, z) .$$

After two applications of (1), Phase 1 separates  $F$  into the  $\Sigma_{\text{cons}}$ -formula

$$F_1 : w_1 = \text{car}(x) \wedge w_2 = \text{car}(y) \wedge \text{cons}(x, z) \neq \text{cons}(y, z)$$

and the  $\Sigma_Z$ -formula

$$F_2 : w_1 + w_2 = z ,$$

with

$$V = \text{shared}(F_1, F_2) = \{z, w_1, w_2\} .$$

Consider the equivalence relation  $E$  given by the partition

$$\{\{z\}, \{w_1\}, \{w_2\}\} .$$

The arrangement

$$\alpha(V, E) : z \neq w_1 \wedge z \neq w_2 \wedge w_1 \neq w_2$$

satisfies both  $F_1$  and  $F_2$ :  $F_1 \wedge \alpha(V, E)$  is  $T_{\text{cons}}$ -satisfiable, and

$F_2 \wedge \alpha(V, E)$  is  $T_Z$ -satisfiable.

Hence,  $F$  is  $(T_{\text{cons}} \cup T_Z)$ -satisfiable.



## Deterministic Version

Phase 1 as before

Phase 2 asks the decision procedures  $P_1$  and  $P_2$  to propagate new equalities.

Example 1:

Real linear arithmetic  $T_{\mathbb{R}}$

$$\boxed{P_{\mathbb{R}}}$$

Theory of equality  $T_E$

$$\boxed{P_E}$$

$$F : f(f(x)-f(y)) \neq f(z) \wedge x \leq y \wedge y+z \leq x \wedge 0 \leq z$$

$(T_{\mathbb{R}} \cup T_E)$ -unsatisfiable

Intuitively,

last 3 conjuncts  $\Rightarrow x = y \wedge z = 0$

contradicts 1st conjunct



## Practical Efficiency

Phase 2 was formulated as "guess and check":

First, guess an equivalence relation  $E$ ,  
then check the induced arrangement.

The number of equivalence relations grows super-exponentially with the # of shared variables. It is given by Bell numbers, e.g., 12 shared variables  $\Rightarrow$  over four million equivalence relations.

Solution: Deterministic Version

## Phase 1: Variable Abstraction

$$F : f(f(x) - f(y)) \neq f(z) \wedge x \leq y \wedge y + z \leq x \wedge 0 \leq z$$

$$f(x) \Rightarrow u \quad f(y) \Rightarrow v \quad u - v \Rightarrow w$$

$$\Gamma_E : \{f(w) \neq f(z), u = f(x), v = f(y)\} \quad \dots T_E\text{-formula}$$

$$\Gamma_{\mathbb{R}} : \{x \leq y, y + z \leq x, 0 \leq z, w = u - v\} \quad \dots T_{\mathbb{R}}\text{-formula}$$

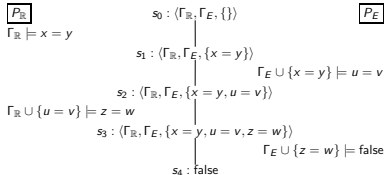
$$\text{shared}(\Gamma_{\mathbb{R}}, \Gamma_E) = \{x, y, z, u, v, w\}$$

Nondeterministic version — over 200 Es!

Let's try the deterministic version.



## Phase 2: Equality Propagation



Contradiction. Thus,  $F$  is  $(T_R \cup T_E)$ -unsatisfiable.  
 If there were no contradiction,  $F$  would be  $(T_R \cup T_E)$ -satisfiable.



## Convex Theories

- ▶  $T_E, T_R, T_Q, T_{\text{cons}}$  are convex
- ▶  $T_Z, T_A$  are not convex

Example:  $T_Z$  is not convex

Consider quantifier-free conjunctive

$$F : 1 \leq z \wedge z \leq 2 \wedge u = 1 \wedge v = 2$$

Then

$$F \models z = u \vee z = v$$

but

$$F \not\models z = u$$

$$F \not\models z = v$$



## Convex Theories

**Claim:**

Equality propagation is a decision procedure for convex theories.

**Def.** A  $\Sigma$ -theory  $T$  is *convex* iff  
 for every quantifier-free conjunction  $\Sigma$ -formula  $F$   
 and for every disjunction  $\bigvee_{i=1}^n (u_i = v_i)$

$$\text{if } F \models \bigvee_{i=1}^n (u_i = v_i)$$

$$\text{then } F \models u_i = v_i, \text{ for some } i \in \{1, \dots, n\}$$

## Example:

The theory of arrays  $T_A$  is not convex.  
 Consider the quantifier-free conjunctive  $\Sigma_A$ -formula

$$F : a(i < v)[j] = v .$$

Then

$$F \Rightarrow i = j \vee a[j] = v ,$$

but

$$F \not\models i = j$$

$$F \not\models a[j] = v .$$

### What if $T$ is Not Convex?

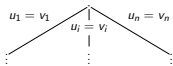
Case split when:

$$\Gamma \models \bigvee_{i=1}^n (u_i = v_i)$$

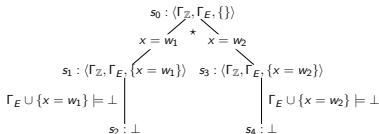
but

$$\Gamma \not\models u_i = v_i \quad \text{for all } i = 1, \dots, n$$

- ▶ For each  $i = 1, \dots, n$ , construct a branch on which  $u_i = v_i$  is assumed.
- ▶ If **all** branches are contradictory, then **unsatisfiable**. Otherwise, **satisfiable**.



### Example 2: Non-Convex Theory



$$* : \Gamma_Z \models x = w_1 \vee x = w_2$$

All leaves are labeled with  $\perp \Rightarrow \Gamma$  is  $(T_Z \cup T_E)$ -unsatisfiable.

### Example 2: Non-Convex Theory

$T_Z$  not convex!

$$\boxed{P_Z}$$

$T_E$  convex

$$\boxed{P_E}$$

$$\Gamma : \left\{ \begin{array}{l} 1 \leq x, \\ f(x) \neq f(1), \end{array} \quad \begin{array}{l} x \leq 2, \\ f(x) \neq f(2) \end{array} \right\} \quad \text{in } T_Z \cup T_E$$

- ▶ Replace  $f(1)$  by  $f(w_1)$ , and add  $w_1 = 1$ .
- ▶ Replace  $f(2)$  by  $f(w_2)$ , and add  $w_2 = 2$ .

Result:

$$\Gamma_Z = \left\{ \begin{array}{l} 1 \leq x, \\ x \leq 2, \\ w_1 = 1, \\ w_2 = 2 \end{array} \right\} \quad \text{and} \quad \Gamma_E = \left\{ \begin{array}{l} f(x) \neq f(w_1), \\ f(x) \neq f(w_2) \end{array} \right\}$$

$$\text{shared}(\Gamma_Z, \Gamma_E) = \{x, w_1, w_2\}$$

### Example 3: Non-Convex Theory

$$\Gamma : \left\{ \begin{array}{l} 1 \leq x, \\ f(x) \neq f(1), \end{array} \quad \begin{array}{l} x \leq 3, \\ f(x) \neq f(3), \\ f(1) \neq f(2) \end{array} \right\} \quad \text{in } T_Z \cup T_E$$

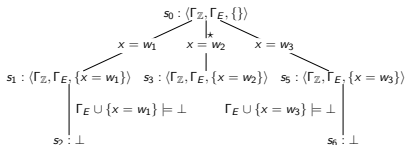
- ▶ Replace  $f(1)$  by  $f(w_1)$ , and add  $w_1 = 1$ .
- ▶ Replace  $f(2)$  by  $f(w_2)$ , and add  $w_2 = 2$ .
- ▶ Replace  $f(3)$  by  $f(w_3)$ , and add  $w_3 = 3$ .

Result:

$$\Gamma_Z = \left\{ \begin{array}{l} 1 \leq x, \\ x \leq 3, \\ w_1 = 1, \\ w_2 = 2, \\ w_3 = 3 \end{array} \right\} \quad \text{and} \quad \Gamma_E = \left\{ \begin{array}{l} f(x) \neq f(w_1), \\ f(x) \neq f(w_3), \\ f(w_1) \neq f(w_2) \end{array} \right\}$$

$$\text{shared}(\Gamma_Z, \Gamma_E) = \{x, w_1, w_2, w_3\}$$

### Example 3: Non-Convex Theory



$$\star : \Gamma_Z \models x = w_1 \vee x = w_2 \vee x = w_3$$

No more equations on middle leaf  $\Rightarrow \Gamma$  is  $(T_Z \cup T_E)$ -satisfiable.

## 11. Arrays

### (2) Array Property Fragment of $T_A$

Decidable fragment of  $T_A$  that includes  $\forall$  quantifiers

#### Array property

$\Sigma_A$ -formula of form

$$\forall \vec{i}. F[\vec{i}] \rightarrow G[\vec{i}],$$

where  $\vec{i}$  is a list of variables.

#### index guard $F[\vec{i}]$ :

$$\begin{array}{l}
 \text{iguard} \rightarrow \text{iguard} \wedge \text{iguard} \mid \text{iguard} \vee \text{iguard} \mid \text{atom} \\
 \text{atom} \rightarrow \text{var} = \text{var} \mid \text{evar} \neq \text{var} \mid \text{var} \neq \text{evar} \mid \top \\
 \text{var} \rightarrow \text{evar} \mid \text{uvar}
 \end{array}$$

where  $uvar$  is any universally quantified index variable, and  $evar$  is any constant or unquantified variable.

- value constraint  $G[\vec{i}]$ :** a universally quantified index can occur in a value constraint  $G[\vec{i}]$  only in a read  $a[i]$ , where  $a$  is an array term. The read cannot be nested; for example,  $a[b[i]]$  is not allowed.

### Array Property Fragment of $T_A$

Boolean combinations of quantifier-free  $T_A$ -formulae and array properties

Example:  $\Sigma_A$ -formulae

$$F : \forall i. i \neq a[k] \rightarrow a[i] = a[k]$$

The antecedent is not a legal index guard since  $a[k]$  is not a variable (neither a  $uvar$  nor an  $evar$ ); however, by simple manipulation

$$F' : v = a[k] \wedge \forall i. i \neq v \rightarrow a[i] = a[k]$$

Here,  $i \neq v$  is a legal index guard, and  $a[i] = a[k]$  is a legal value constraint.  $F$  and  $F'$  are equisatisfiable.

However, no manipulation works for:

$$G : \forall i. i \neq a[i] \rightarrow a[i] = a[k].$$

Thus,  $G$  is not in the array property fragment.

**Remark:** Array property fragment allows expressing equality between arrays (extensionality): two arrays are equal precisely when their corresponding elements are equal.

For given formula

$$F : \dots \wedge a = b \wedge \dots$$

with array terms  $a$  and  $b$ , rewrite  $F$  as

$$F' : \dots \wedge (\forall i. \top \rightarrow a[i] = b[i]) \wedge \dots$$

$F$  and  $F'$  are equisatisfiable.

### Decision Procedure for Array Property Fragment

The idea of the decision procedure for the array property fragment is to reduce universal quantification to finite conjunction. That is, it constructs a finite set of index terms s.t. examining only these positions of the arrays is sufficient.

**Example:** Consider

$$F : a(i \triangleleft v) = a \wedge a[i] \neq v,$$

which expands to

$$F' : \forall j. a(i \triangleleft v)[j] = a[j] \wedge a[i] \neq v.$$

Intuitively, to determine that  $F'$  is  $T_A$ -unsatisfiable requires merely examining index  $i$ :

$$F'' : \left( \bigwedge_{j \in \{i\}} a(i \triangleleft v)[j] = a[j] \right) \wedge a[i] \neq v,$$

or simply

$$a(i \triangleleft v)[i] = a[i] \wedge a[i] \neq v.$$

Simplifying,

$$v = a[i] \wedge a[i] \neq v,$$

it is clear that this formula, and thus  $F$ , is  $T_A$ -unsatisfiable.

### The Algorithm

Given array property formula  $F$ , decide its  $T_A$ -satisfiability by the following steps:

#### Step 1

Put  $F$  in NNF.

#### Step 2

Apply the following rule exhaustively to remove writes:

$$\frac{F[a(i \triangleleft v)]}{F[a] \wedge a[i] = v \wedge (\forall j. j \neq i \rightarrow a[j] = a'[j])} \text{ for fresh } a' \text{ (write)}$$

After an application of the rule, the resulting formula contains at least one fewer write terms than the given formula.

#### Step 3

Apply the following rule exhaustively to remove existential quantification:

$$\frac{F[\exists \vec{t}. G[\vec{t}]]}{F[G[\vec{t}]]} \text{ for fresh } \vec{t} \text{ (exists)}$$

Existential quantification can arise during Step 1 if the given formula has a negated array property.

Steps 4-6 accomplish the reduction of universal quantification to finite conjunction.

Main idea: select a set of symbolic index terms on which to instantiate all universal quantifiers. The set is sufficient for correctness.

#### Step 4

From the output  $F_3$  of Step 3, construct the **index set**  $\mathcal{I}$ :

$$\mathcal{I} = \cup \{ \lambda \} \cup \{ t : [t : \cdot][t] \in F_3 \text{ such that } t \text{ is not a universally quantified variable} \} \cup \{ t : t \text{ occurs as an } \textit{evar} \text{ in the parsing of index guards} \}$$

This index set is the finite set of indices that need to be examined. It includes

- ▶ all terms  $t$  that occur in some read  $a[t]$  anywhere in  $F$  (unless it is a universally quantified variable)
- ▶ all terms  $t$  (constant or unquantified variable) that are compared to a universally quantified variable in some index guard.
- ▶  $\lambda$  is a fresh constant that represents all other index positions that are not explicitly in  $\mathcal{I}$ .

### Step 5 (Key step)

Apply the following rule exhaustively to remove universal quantification:

$$\frac{H[\vec{v}. F[\vec{v}]] \rightarrow G[\vec{v}]}{H\left[\bigwedge_{\vec{i} \in \mathcal{I}^n} (F[\vec{i}] \rightarrow G[\vec{i}])\right]} \quad (\text{forall})$$

where  $n$  is the size of the list of quantified variables  $\vec{i}$ .

### Step 6

From the output  $F_5$  of Step 5, construct

$$F_6 : F_5 \wedge \bigwedge_{i \in \mathcal{I}\{\lambda\}} \lambda \neq i.$$

The new conjuncts assert that the variable  $\lambda$  introduced in Step 4 is indeed unique.

### Step 7

Decide the  $T_A$ -satisfiability of  $F_6$  using the decision procedure for the quantifier-free fragment.

⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ Ⓜ Ⓝ

$$F_5 : \begin{aligned} & a'[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge \bigwedge_{i \in \mathcal{I}} (i \neq \ell \rightarrow a[i] = b[i]) \\ & \wedge a'[\ell] = v \wedge \bigwedge_{j \in \mathcal{I}} (j \neq \ell \rightarrow a[j] = a'[j]) \end{aligned}$$

Expanding produces

$$F_5' : \begin{aligned} & a'[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\lambda \neq \ell \rightarrow a[\lambda] = b[\lambda]) \\ & \wedge (k \neq \ell \rightarrow a[k] = b[k]) \wedge (\ell \neq \ell \rightarrow a[\ell] = b[\ell]) \\ & \wedge a'[\ell] = v \wedge (\lambda \neq \ell \rightarrow a[\lambda] = a'[\lambda]) \\ & \wedge (k \neq \ell \rightarrow a[k] = a'[k]) \wedge (\ell \neq \ell \rightarrow a[\ell] = a'[\ell]) \end{aligned}$$

Simplifying produces

$$F_5'' : \begin{aligned} & a'[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\lambda \neq \ell \rightarrow a[\lambda] = b[\lambda]) \\ & \wedge (k \neq \ell \rightarrow a[k] = b[k]) \\ & \wedge a'[\ell] = v \wedge (\lambda \neq \ell \rightarrow a[\lambda] = a'[\lambda]) \\ & \wedge (k \neq \ell \rightarrow a[k] = a'[k]) \end{aligned}$$

⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ Ⓜ Ⓝ

Example: Consider array property formula

$$F : a[\ell < v][k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge \underbrace{(\forall i. i \neq \ell \rightarrow a[i] = b[i])}_{\text{array property}}$$

Index guard is  $i \neq \ell$  and the value constraint is  $a[i] = b[i]$ . It is already in NNF. By Step 2, rewrite  $F$  as

$$F_2 : \begin{aligned} & a'[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\forall i. i \neq \ell \rightarrow a[i] = b[i]) \\ & \wedge a'[\ell] = v \wedge (\forall j. j \neq \ell \rightarrow a[j] = a'[j]) \end{aligned}$$

$F_2$  does not contain any existential quantifiers. Its index set is

$$\begin{aligned} \mathcal{I} &= \{\lambda\} \cup \{k\} \cup \{\ell\} \\ &= \{\lambda, k, \ell\}. \end{aligned}$$

Thus, by Step 5, replace universal quantification:

$$F_5 : \begin{aligned} & a'[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge \bigwedge_{i \in \mathcal{I}} (i \neq \ell \rightarrow a[i] = b[i]) \\ & \wedge a'[\ell] = v \wedge \bigwedge_{j \in \mathcal{I}} (j \neq \ell \rightarrow a[j] = a'[j]) \end{aligned}$$

⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ Ⓜ Ⓝ

Step 6 distinguishes  $\lambda$  from other members of  $\mathcal{I}$ :

$$F_6 : \begin{aligned} & a'[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\lambda \neq \ell \rightarrow a[\lambda] = b[\lambda]) \\ & \wedge (k \neq \ell \rightarrow a[k] = b[k]) \\ & \wedge a'[\ell] = v \wedge (\lambda \neq \ell \rightarrow a[\lambda] = a'[\lambda]) \\ & \wedge (k \neq \ell \rightarrow a[k] = a'[k]) \\ & \wedge \lambda \neq k \wedge \lambda \neq \ell \end{aligned}$$

Simplifying,

$$F_6' : \begin{aligned} & a'[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \\ & \wedge a[\lambda] = b[\lambda] \wedge (k \neq \ell \rightarrow a[k] = b[k]) \\ & \wedge a'[\ell] = v \wedge a[\lambda] = a'[\lambda] \wedge (k \neq \ell \rightarrow a[k] = a'[k]) \\ & \wedge \lambda \neq k \wedge \lambda \neq \ell \end{aligned}$$

There are two cases to consider.

- ▶ If  $k = \ell$ , then  $a'[\ell] = v$  and  $a'[k] = b[k]$  imply  $b[k] = v$ , yet  $b[k] \neq v$ .
- ▶ If  $k \neq \ell$ , then  $a[k] = v$  and  $a[k] = b[k]$  imply  $b[k] = v$ , but again  $b[k] \neq v$ .

Hence,  $F_6'$  is  $T_A$ -unsatisfiable, indicating that  $F_5$  is  $T_A$ -unsatisfiable.

⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ Ⓜ Ⓝ

### (3) Theory of Integer-Indexed Arrays $T_A^Z$

$\leq$  enables reasoning about subarrays and properties such as subarray is sorted or partitioned.

signature of  $T_A^Z$ :  $\Sigma_A^Z = \Sigma_A \cup \Sigma_Z$

axioms of  $T_A^Z$ : both axioms of  $T_A$  and  $T_Z$

Array property:  $\Sigma_A^Z$ -formula of the form

$$\forall \vec{i}. F[\vec{i}] \rightarrow G[\vec{i}],$$

where  $\vec{i}$  is a list of integer variables.

►  $F[\vec{i}]$  index guard:

iguard  $\rightarrow$  iguard  $\wedge$  iguard | iguard  $\vee$  iguard | atom

atom  $\rightarrow$  expr  $\leq$  expr | expr = expr

expr  $\rightarrow$  *uvar* | pexpr

pexpr  $\rightarrow$  pexpr'

pexpr'  $\rightarrow$   $\mathbb{Z}$  |  $\mathbb{Z} \cdot$  *evar* | pexpr' + pexpr'

where *uvar* is any universally quantified integer variable, and *evar* is any existentially quantified or free integer variable.

►  $G[\vec{i}]$  value constraint:

Any occurrence of a quantified index variable  $i$  must be as a read into an array,  $a[i]$ , for array term  $a$ . Array reads may not be nested; e.g.,  $a[b[i]]$  is not allowed.

Array property fragment of  $T_A^Z$  consists of formulae that are

Boolean combinations of quantifier-free  $\Sigma_A^Z$ -formulae and array properties.

### Step 3

Apply the following rule exhaustively to remove existential quantification:

$$\frac{F[\exists \vec{i}. G[\vec{i}]]}{F[G[\vec{i}]]} \text{ for fresh } \vec{i} \quad (\text{exists})$$

Existential quantification can arise during Step 1 if the given formula has a negated array property.

### Step 4

From the output of Step 3,  $F_3$ , construct the index set  $\mathcal{I}$ :

$$\mathcal{I} = \{t : \cdot [t] \in F_3 \text{ such that } t \text{ is not a universally quantified variable}\} \cup \{t : t \text{ occurs as a pexpr in the parsing of index guards}\}$$

If  $\mathcal{I} = \emptyset$ , then let  $\mathcal{I} = \{0\}$ . The index set contains all relevant symbolic indices that occur in  $F_3$ .

### A Decision Procedure

The idea again is to reduce universal quantification to finite conjunction.

Given  $F$  from the array property fragment of  $T_A^Z$ , decide its  $T_A^Z$ -satisfiability as follows:

#### Step 1

Put  $F$  in NNF.

#### Step 2

Apply the following rule exhaustively to remove writes:

$$\frac{F[a(i \triangleleft e)]}{F[a'] \wedge a'[i] = e \wedge (\forall j. j \neq i \rightarrow a[j] = a'[j])} \text{ for fresh } a' \quad (\text{write})$$

To meet the syntactic requirements on an index guard, rewrite the third conjunct as

$$\forall j. j \leq i - 1 \vee i + 1 \leq j \rightarrow a[j] = a'[j].$$

### Step 5

Apply the following rule exhaustively to remove universal quantification:

$$\frac{H[\forall \vec{i}. F[\vec{i}] \rightarrow G[\vec{i}]]}{H \left[ \bigwedge_{\vec{i} \in \mathcal{I}^n} (F[\vec{i}] \rightarrow G[\vec{i}]) \right]} \quad (\text{forall})$$

$n$  is the size of the block of universal quantifiers over  $\vec{i}$ .

### Step 6

$F_5$  is quantifier-free in the combination theory  $T_A \cup T_Z$ . Decide the  $(T_A \cup T_Z)$ -satisfiability of the resulting formula.

◀ ▶ ↻ 🔍

Step 3 removes the existential quantifier by introducing a fresh constant  $k$ :

$$F_3: \begin{aligned} & (\forall i. \ell \leq i \leq u \rightarrow a[i] = b[i]) \\ & \wedge \ell \leq k \leq u+1 \wedge a'[k] \neq b[k] \\ & \wedge a'[u+1] = b[u+1] \\ & \wedge (\forall j. j \leq u+1-1 \vee u+1+1 \leq j \rightarrow a[j] = a'[j]) \end{aligned}$$

Simplifying,

$$F'_3: \begin{aligned} & (\forall i. \ell \leq i \leq u \rightarrow a[i] = b[i]) \\ & \wedge \ell \leq k \leq u+1 \wedge a'[k] \neq b[k] \\ & \wedge a'[u+1] = b[u+1] \\ & \wedge (\forall j. j \leq u \vee u+2 \leq j \rightarrow a[j] = a'[j]) \end{aligned}$$

The index set is

$$\mathcal{I} = \{k, u+1\} \cup \{\ell, u, u+2\},$$

which includes the read terms  $k$  and  $u+1$  and the terms  $\ell, u$ , and  $u+2$  that occur as pexprs in the index guards.

◀ ▶ ↻ 🔍

Example:  $\Sigma_A^Z$ -formula:

$$F: \begin{aligned} & (\forall i. \ell \leq i \leq u \rightarrow a[i] = b[i]) \\ & \wedge \neg(\forall i. \ell \leq i \leq u+1 \rightarrow a(u+1 < b[u+1])[i] = b[i]) \end{aligned}$$

In NNF, we have

$$F_1: \begin{aligned} & (\forall i. \ell \leq i \leq u \rightarrow a[i] = b[i]) \\ & \wedge (\exists i. \ell \leq i \leq u+1 \wedge a[u+1 < b[u+1]][i] \neq b[i]) \end{aligned}$$

Step 2 produces

$$F_2: \begin{aligned} & (\forall i. \ell \leq i \leq u \rightarrow a[i] = b[i]) \\ & \wedge (\exists i. \ell \leq i \leq u+1 \wedge a'[i] \neq b[i]) \\ & \wedge a'[u+1] = b[u+1] \\ & \wedge (\forall j. j \leq u+1-1 \vee u+1+1 \leq j \rightarrow a[j] = a'[j]) \end{aligned}$$

◀ ▶ ↻ 🔍

Step 5 rewrites universal quantification to finite conjunction over this set:

$$F_5: \begin{aligned} & \bigwedge_{i \in \mathcal{I}} (\ell \leq i \leq u \rightarrow a[i] = b[i]) \\ & \wedge \ell \leq k \leq u+1 \wedge a'[k] \neq b[k] \\ & \wedge a'[u+1] = b[u+1] \\ & \wedge \bigwedge_{j \in \mathcal{I}} (j \leq u \vee u+2 \leq j \rightarrow a[j] = a'[j]) \end{aligned}$$

Expanding the conjunctions according to the index set  $\mathcal{I}$  and simplifying according to trivially true or false antecedents (e.g.,  $\ell \leq u+1 \leq u$  simplifies to  $\perp$ , while  $u \leq u \vee u+2 \leq u$  simplifies to  $\top$ ) produces:

◀ ▶ ↻ 🔍

$$\begin{aligned}
 F'_5 : \quad & (\ell \leq k \leq u \rightarrow a[k] = b[k]) & (1) \\
 & \wedge (\ell \leq u \rightarrow a[\ell] = b[\ell] \wedge a[u] = b[u]) & (2) \\
 & \wedge \ell \leq k \leq u + 1 & (3) \\
 & \wedge a'[k] \neq b[k] & (4) \\
 & \wedge a'[u + 1] = b[u + 1] & (5) \\
 & \wedge (k \leq u \vee u + 2 \leq k \rightarrow a[k] = a'[k]) & (6) \\
 & \wedge (\ell \leq u \vee u + 2 \leq \ell \rightarrow a[\ell] = a'[\ell]) & (7) \\
 & \wedge a[u] = a'[u] \wedge a[u + 2] = a'[u + 2] & (8)
 \end{aligned}$$

$(T_A \cup T_Z)$ -unsatisfiability of this quantifier-free  $(\Sigma_A \cup \Sigma_Z)$ -formula can be decided using the techniques of Combination of Theories. Informally,  $\ell \leq k \leq u + 1$  (3)

- ▶ If  $k \in [\ell, u]$  then  $a[k] = b[k]$  (1). Since  $k \leq u$  then  $a[k] = a'[k]$  (6), contradicting  $a'[k] \neq b[k]$  (4).
- ▶ if  $k = u + 1$ ,  $a'[k] \neq b[k] = b[u + 1] = a'[u + 1] = a'[k]$  by (4) and (5), a contradiction.

Hence,  $F$  is  $T_A^Z$ -unsatisfiable.



## 12. Invariant Generation



Application: array property fragments

- ▶ Array equality  $a = b$  in  $T_A$ :

$$\forall i. a[i] = b[i]$$

- ▶ Bounded array equality  $\text{beq}(a, b, \ell, u)$  in  $T_A^Z$ :

$$\forall i. \ell \leq i \leq u \rightarrow a[i] = b[i]$$

- ▶ Universal properties  $F[x]$  in  $T_A$ :

$$\forall i. F[a[i]]$$

- ▶ Bounded universal properties  $F[x]$  in  $T_A^Z$ :

$$\forall i. \ell \leq i \leq u \rightarrow F[a[i]]$$

- ▶ Bounded and unbounded sorted arrays  $\text{sorted}(a, \ell, u)$  in  $T_A^Z \cup T_Z$  or  $T_A^Z \cup T_Q$ :

$$\forall i, j. \ell \leq i \leq j \leq u \rightarrow a[i] \leq a[j]$$

- ▶ Partitioned arrays  $\text{partitioned}(a, \ell_1, u_1, \ell_2, u_2)$  in  $T_A^Z \cup T_Z$  or  $T_A^Z \cup T_Q$ :

$$\forall i, j. \ell_1 \leq i \leq u_1 < \ell_2 \leq j \leq u_2 \rightarrow a[i] \leq a[j]$$



### Invariant Generation

Discover inductive assertions of programs

- General procedure
- Concrete analysis

- ▶ interval analysis  
invariants of form

$$c \leq v \text{ or } v \leq c$$

for program variable  $v$  and constant  $c$

- ▶ Karr's analysis  
invariants of form

$$c_0 + c_1x_1 + \dots + c_nx_n = 0$$

for program variables  $x_i$  and constants  $c_i$

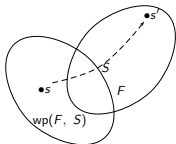
Other invariant generation algorithms in literature:

- ▶ linear inequalities  
 $c_0 + c_1x_1 + \dots + c_nx_n \leq 0$
- ▶ polynomial equalities and inequalities



## Background

### Weakest Precondition



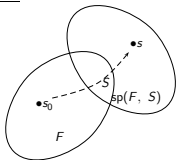
For FOL formula  $F$  and program statement  $S$ , the weakest precondition  $wp(F, S)$  is a FOL formula s.t. if for state  $s$

$$s \models wp(F, S)$$

and if statement  $S$  is executed on state  $s$  to produce state  $s'$ , then

$$s' \models F.$$

### Strongest Postcondition



For FOL formula  $F$  and program statement  $S$ , the strongest postcondition  $sp(F, S)$  is a FOL formula s.t. if  $s$  is the current state and

$$s \models sp(F, S)$$

then statement  $S$  was executed from a state  $s_0$  s.t.

$$s_0 \models F.$$

In other words, the weakest precondition moves a formula backwards over a series of statements:

for  $F$  to hold after executing  $S_1; \dots; S_n$ ,

$wp(F, S_1; \dots; S_n)$  must hold before executing the statements.

For assume and assignment statements

▶  $wp(F, \text{assume } c) \Leftrightarrow c \rightarrow F$ , and

▶  $wp(F[v], v := e) \Leftrightarrow F[e]$ ;

and on sequences of statements  $S_1; \dots; S_n$ :

$$wp(F, S_1; \dots; S_n) \Leftrightarrow wp(wp(F, S_n), S_1; \dots; S_{n-1}).$$

▶ On assume statements,

$$sp(F, \text{assume } c) \Leftrightarrow c \wedge F,$$

if program control makes it past the statement, then  $c$  must hold.

▶ Unlike in the case of  $wp$ , there is no simple definition of  $sp$  on assignments:

$$sp(F[v], v := e[v]) \Leftrightarrow \exists v^0. v = e[v^0] \wedge F[v^0].$$

▶ On a sequence of statements  $S_1; \dots; S_n$ :

$$sp(F, S_1; \dots; S_n) \Leftrightarrow sp(sp(F, S_1), S_2; \dots; S_n).$$

Example: Compute

$$\begin{aligned} \text{sp}(i \geq n, i := i + k) \\ \Leftrightarrow \exists i^0. i = i^0 + k \wedge i^0 \geq n \\ \Leftrightarrow i - k \geq n \end{aligned}$$

since  $i^0 = i - k$ .

Example: Compute

$$\begin{aligned} \text{sp}(i \geq n, \text{assume } k \geq 0; i := i + k) \\ \Leftrightarrow \text{sp}(\text{sp}(i \geq n, \text{assume } k \geq 0), i := i + k) \\ \Leftrightarrow \text{sp}(k \geq 0 \wedge i \geq n, i := i + k) \\ \Leftrightarrow \exists i^0. i = i^0 + k \wedge k \geq 0 \wedge i^0 \geq n \\ \Leftrightarrow k \geq 0 \wedge i - k \geq n \end{aligned}$$

◀ ▶ ↻ 🔍

◀ ▶ ↻ 🔍

Static Analysis: basic definition

- ▶ Program  $P$  with locations  $\mathcal{L}$  ( $L_0$  — initial location)
- ▶ Cutset of  $\mathcal{L}$   
each path from one cutpoint (location in the cutset) to the next cutpoint is basic path (does not cross loops)
- ▶ Assertion map

$$\mu: \mathcal{L} \rightarrow \text{FOL}$$

(map from  $\mathcal{L}$  to first-order assertions).

It is inductive (inductive map) if for each basic path

$$\begin{array}{c} \hline L_j : \textcircled{\mu(L_j)} \\ S_j; \\ \vdots \\ S_j; \\ L_j : \textcircled{\mu(L_j)} \\ \hline \end{array} \quad (\cdot)$$

for  $L_j, L_j \in \mathcal{L}$ , the verification condition

$$\{\mu(L_i)\} S_i; \dots; S_j \{\mu(L_j)\} \quad (\text{VC})$$

is valid.

◀ ▶ ↻ 🔍

◀ ▶ ↻ 🔍

Verification Condition

VCS in terms of wp:

$$\{F\} S_1; \dots; S_n \{G\} : F \Rightarrow \text{wp}(G, S_1; \dots; S_n).$$

VCS in terms of sp:

$$\{F\} S_1; \dots; S_n \{G\} : \text{sp}(F, S_1; \dots; S_n) \Rightarrow G.$$

Invariant Generation

Find inductive assertion maps  $\mu$  s.t. the  $\mu(L_i)$  satisfies (VC) for all basic paths.

Method: Symbolic execution (forward propagation)

- ▶ Initial map  $\mu_0$ :

$$\begin{aligned} \mu(L_0) &:= F_{\text{pre}}, \text{ and} \\ \mu(L) &:= \perp \quad \text{for } L \in \mathcal{L} \setminus \{L_0\}. \end{aligned}$$

- ▶ Maintain set  $S \subseteq \mathcal{L}$  of locations that still need processing. Initially, let  $S = \{L_0\}$ . Terminate when  $S = \emptyset$ .
- ▶ Iteration  $i$ : We have so far constructed  $\mu_i$ . Choose some  $L_j \in S$  to process and remove it from  $S$ .

◀ ▶ ↻ 🔍

◀ ▶ ↻ 🔍

For each basic path (starting at  $L_j$ )

```
(·)
Lj : @ μ(Lj)
Sj;
⋮
Sk;
Lk : @ μ(Lk)
```

compute and set

$$\mu(L_k) \Leftarrow \mu(L_k) \vee \text{sp}(\mu(L_j), S_j; \dots; S_k)$$

If

$$\text{sp}(\mu(L_j), S_j; \dots; S_k) \Rightarrow \mu_i(L_k)$$

that is, if  $\text{sp}$  does not represent new states not already represented in  $\mu_i(L_k)$ , then  $\mu_{i+1}(L_k) \Leftarrow \mu_i(L_k)$  (nothing new is learned)

Otherwise add  $L_k$  to  $S$ .

For all other locations  $L_\ell \in \mathcal{L}$ ,  $\mu_{i+1}(L_\ell) \Leftarrow \mu_i(L_\ell)$

When  $S = \emptyset$  (say iteration  $i^*$ ), then  $\mu_{i^*}$  is an inductive map.

⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿

**Problem:** algorithm may not terminate

**Example:** Consider loop with integer variables  $i$  and  $n$ :

```
@L0 : i = 0 ∧ n ≥ 0;
while
  @L1 : ?
  (i < n) {
    i := i + 1;
  }
```

There are two basic paths:

(1)

```
@L0 : i = 0 ∧ n ≥ 0;
@L1 : ?;
```

and

(2)

```
@L1 : ?;
assume i < n;
i := i + 1;
@L1 : ?;
```

⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿

The algorithm

let FORWARDPROPAGATE  $P F_{pre} \mathcal{L} =$

```
S := {L0};
μ(L0) := Fpre;
μ(L) := ⊥ for L ∈ ℒ \ {L0};
while S ≠ ∅ do
  let Lj = CHOOSE S in
  S := S \ {Lj};
```

```
foreach Lk ∈ succ(Lj) do [Lk ∈ succ(Lj) is a successor of Lj
                           if there is a basic path from Lj to Lk]
  let F = sp(μ(Lj), Sj; ...; Sk) in
```

```
  if F ≠ μ(Lk)
  then μ(Lk) := μ(Lk) ∨ F;
  S := S ∪ {Lk};
```

done;

done;

μ

⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿

► Initially,

$\mu(L_0) \Leftarrow i = 0 \wedge n \geq 0$
$\mu(L_1) \Leftarrow \perp$

► Following path (1) results in setting

$$\mu(L_1) := \mu(L_1) \vee (i = 0 \wedge n \geq 0)$$

$\mu(L_1)$  was  $\perp$ , so that it becomes

$\mu(L_1) \Leftarrow i = 0 \wedge n \geq 0$
---

► On the next iteration, following path (2) yields

$$\mu(L_1) := \mu(L_1) \vee \text{sp}(\mu(L_1), \text{assume } i < n; i := i + 1).$$

Currently  $\mu(L_1) \Leftarrow i = 0 \wedge n \geq 0$ , so

$$\begin{aligned} F &: \text{sp}(i = 0 \wedge n \geq 0, \text{assume } i < n; i := i + 1) \\ &\Leftarrow \text{sp}(i < n \wedge i = 0 \wedge n \geq 0, i := i + 1) \\ &\Leftarrow \exists i^0. i = i^0 + 1 \wedge i^0 < n \wedge i^0 = 0 \wedge n \geq 0 \\ &\Leftarrow i = 1 \wedge n > 0 \end{aligned}$$

⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿

Since the implication

$$\underbrace{i = 1 \wedge n > 0}_F \Rightarrow \underbrace{i = 0 \wedge n \geq 0}_{\mu(L_1)}$$

is invalid,

$$\mu(L_1) \Leftrightarrow \underbrace{(i = 0 \wedge n \geq 0)}_{\mu(L_1)} \vee \underbrace{(i = 1 \wedge n > 0)}_F$$

at the end of the iteration.

- ▶ At the end of the next iteration,

$$\mu(L_1) \Leftrightarrow \underbrace{(i = 0 \wedge n \geq 0) \vee (i = 1 \wedge n > 0)}_{\mu(L_1)} \vee \underbrace{(i = 2 \wedge n > 1)}_F$$

### Solution: Abstraction

A state  $s$  is reachable for program  $P$  if it appears in some computation of  $P$ .

The problem is that FORWARDPROPAGATE computes the exact set of reachable states.

Inductive annotations usually over-approximate the set of reachable states: every reachable state  $s$  satisfies the annotation, but other unreachable states can also satisfy the annotation.

Abstract interpretation cleverly over-approximate the reachable state set to guarantee termination.

Abstract interpretation is constructed in 6 steps.

- ▶ At the end of the  $k$ th iteration,

$$\mu(L_1) \Leftrightarrow (i = 0 \wedge n \geq 0) \vee (i = 1 \wedge n \geq 1) \vee \dots \vee (i = k \wedge n \geq k)$$

It is never the case that the implication

$$i = k \wedge n \geq k$$

↓

$$(i = 0 \wedge n \geq 0) \vee (i = 1 \wedge n \geq 1) \vee \dots \vee (i = k - 1 \wedge n \geq k - 1)$$

is valid, so the main loop of while never finishes.

- ▶ However, it is obvious that

$$0 \leq i \leq n$$

is an inductive annotation of the loop.

### Step 1: Choose an abstract domain $D$ .

The **abstract domain**  $D$  is a syntactic class of  $\Sigma$ -formulae of some theory  $T$ .

- ▶ **interval abstract domain**  $D_I$  consists of conjunctions of  $\Sigma_Q$ -literals of the forms

$$c \leq v \quad \text{and} \quad v \leq c,$$

for constant  $c$  and program variable  $v$ .

- ▶ **Karr's abstract domain**  $D_K$  consist of conjunctions of  $\Sigma_Q$ -literals of the form

$$c_0 + c_1 x_1 + \dots + c_n x_n = 0,$$

for constants  $c_0, c_1, \dots, c_n$  and variables  $x_1, \dots, x_n$ .

## Step 2: Construct a map from FOL formulae to $D$ .

Define

$$\nu_D : \text{FOL} \rightarrow D$$

to map a FOL formula  $F$  to element  $\nu_D(F)$  of  $D$ , with the property that for any  $F$ ,

$$F \Rightarrow \nu_D(F).$$

Example:

$$F : i = 0 \wedge n \geq 0$$

at  $L_0$  of the loop can be represented in the interval abstract domain by

$$\nu_{D_I}(F) : 0 \leq i \wedge i \leq 0 \wedge 0 \leq n$$

and in Karr's abstract domain by

$$\nu_{D_K}(F) : i = 0$$

with some loss of information.



### ► assignment statements:

More complex, for suppose that we use the standard definition

$$\text{sp}(F[v], v := e[v]) \Leftrightarrow \underbrace{\exists v^0. v = e[v^0] \wedge F[v^0]}_G,$$

which requires existential quantification. Then, later, when we compute the validity of

$$G \Rightarrow \mu(L), \quad \text{i.e., } \forall \vec{b}. G \rightarrow \mu(L),$$

$\mu(L)$  can contain existential quantification, resulting in a quantifier alternation. Most decision procedures, apply only to quantifier-free formulae. Therefore, introducing existential quantification in  $\text{sp}$  is undesirable.



## Step 3: Define an abstract sp.

Define an **abstract strongest postcondition**  $\overline{\text{sp}}_D$  for assumption and assignment statements such that

$$\text{sp}(F, S) \Rightarrow \overline{\text{sp}}_D(F, S) \quad \text{and} \quad \overline{\text{sp}}_D(F, S) \in D$$

for statement  $S$  and  $F \in D$ .

### ► statement assume $c$ :

$$\text{sp}(F, \text{assume } c) \Leftrightarrow c \wedge F.$$

Conjunction  $\wedge$  is used.

Define abstract conjunction  $\sqcap_D$ , such that

$$F_1 \wedge F_2 \Rightarrow F_1 \sqcap_D F_2 \quad \text{and} \quad F_1 \sqcap_D F_2 \in D$$

for  $F_1, F_2 \in D$ . Then if  $F \in D$ ,

$$\overline{\text{sp}}_D(F, \text{assume } c) \Leftrightarrow \nu_D(c) \sqcap_D F.$$

If the abstract domain  $D$  consists of conjunctions of literals,  $\sqcap_D$  is just  $\wedge$ . For example, in the interval domain,

$$\overline{\text{sp}}_{D_I}(F, \text{assume } c) \Leftrightarrow \nu_{D_I}(c) \wedge F.$$

## Step 4: Define abstract disjunction.

Disjunction is applied in FORWARDPROPAGATE

$$\mu(L_k) := F \vee \mu(L_k)$$

Define abstract disjunction  $\sqcup_D$  for this purpose, such that

$$F_1 \vee F_2 \Rightarrow F_1 \sqcup_D F_2 \quad \text{and} \quad F_1 \sqcup_D F_2 \in D$$

for  $F_1, F_2 \in D$ .

Unlike conjunction, exact disjunction is usually not represented in the domain  $D$ .

## Step 5: Define abstract implication checking.

On each iteration of the inner loop of FORWARDPROPAGATE, validity of the implication

$$F \Rightarrow \mu(L_k)$$

is checked to determine whether  $\mu(L_k)$  has changed. A proper selection of  $D$  ensures that this validity check is decidable.



### Step 6: Define widening.

Defining an abstraction is not sufficient to guarantee termination in general. Thus, abstractions that do not guarantee termination are equipped with a widening operator  $\nabla_D$ .

A **widening operator**  $\nabla_D$  is a binary function

$$\nabla_D : D \times D \rightarrow D$$

such that

$$F_1 \vee F_2 \Rightarrow F_1 \nabla_D F_2$$

for  $F_1, F_2 \in D$ . It obeys the following property. Let  $F_1, F_2, F_3, \dots$  be an infinite sequence of elements  $F_i \in D$  such that for each  $i$ ,

$$F_i \Rightarrow F_{i+1}.$$

Define the sequence

$$G_1 = F_1 \quad \text{and} \quad G_{i+1} = G_i \nabla_D F_{i+1}.$$

For some  $i^*$  and for all  $i \geq i^*$ ,

$$G_i \Leftrightarrow G_{i+1}.$$

That is, the sequence  $G_i$  converges even if the sequence  $F_i$  does not converge. A proper strategy of applying widening guarantees that the forward propagation procedure terminates.

```
let ABSTRACTFORWARDPROPAGATE  $P$   $F_{pre}$   $\mathcal{L} =$ 
```

```
   $S := \{L_0\};$ 
```

```
   $\mu(L_0) := \nu_D(F_{pre});$ 
```

```
   $\mu(L) := \perp$  for  $L \in \mathcal{L} \setminus \{L_0\};$ 
```

```
  while  $S \neq \emptyset$  do
```

```
    let  $L_j = \text{CHOOSE } S$  in
```

```
     $S := S \setminus \{L_j\};$ 
```

```
    foreach  $L_k \in \text{succ}(L_j)$  do
```

```
      let  $F = \overline{\text{sp}}_D(\mu(L_j), S_j; \dots; S_k)$  in
```

```
      if  $F \not\Rightarrow \mu(L_k)$ 
```

```
        then if WIDEN()
```

```
          then  $\mu(L_k) := \mu(L_k) \nabla_D (\mu(L_k) \sqcup_D F);$ 
```

```
          else  $\mu(L_k) := \mu(L_k) \sqcup_D F;$ 
```

```
           $S := S \cup \{L_k\};$ 
```

```
        done;
```

```
    done;
```

```
   $\mu$ 
```