The Calculus of Computation: Decision Procedures with Applications to Verification

by

Aaron Bradley
Zohar Manna

Springer 2007

Part I: Foundations

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)

THE CALCULUS OF COMPUTATION:
Decision Procedures with
Applications to Verification

by
Aaron Bradley
Zohar Manna

Springer 2007

Part I: FOUNDATIONS

1. Propositional Logic(PL)

Propositional Logic(PL)

PL Syntax

Atom	truth symbols $\top$("true") and $\bot$("false")
	propositional variables $P, Q, R, P_1, Q_1, R_1, \cdots$
Literal	atom α or its negation $\neg\alpha$
Formula	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 $F : (P \land Q) \rightarrow (\top \lor \neg Q)$
atoms: $P, Q, \top$
literal: $\neg Q$
subformulas: $P \land Q, \quad \top \lor \neg Q$
abbreviation

$$F : P \land Q \rightarrow \top \lor \neg Q$$

PL Semantics (meaning)

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

Interpretation

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

Evaluation of F under I:

F	$\neg F$
0	1
1	0

where 0 corresponds to value false
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

$1 = \text{true}$ $\qquad 0 = \text{false}$

F evaluates to true under I

Inductive Definition of PL's Semantics

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

Base Case:

$$I \models \top$$
$$I \not\models \bot$$

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

Inductive Case:

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

Note:

$$I \not\models F_1 \rightarrow F_2 \quad \text{iff} \quad I \models F_1 \text{ and } I \not\models F_2$$

Example:

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

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

1.	$I \models P$		since $I[P] = \text{true}$	
2.	$I \not\models Q$		since $I[Q] = \text{false}$	
3.	$I \models \neg Q$		by 2 and $\neg$	
4.	$I \not\models P \wedge Q$		by 2 and $\wedge$	
5.	$I \models P \vee \neg Q$		by 1 and $\vee$	
6.	$I \models F$		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, $I \models F$.

$$\boxed{F \text{ is valid iff } \neg F \text{ is unsatisfiable}}$$

Method 1: Truth Tables

Example $F : P \land Q \rightarrow P \lor \neg Q$

P	Q	$P \land Q$	$\neg Q$	$P \lor \neg Q$	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 $\quad F : P \lor Q \rightarrow P \land Q$

P	Q	$P \lor Q$	$P \land Q$	F	
0	0	0	0	1	$\leftarrow$ satisfying I
0	1	1	0	0	$\leftarrow$ falsifying I
1	0	1	0	0	
1	1	1	1	1	

Thus F is satisfiable, but invalid.

Method 2: Semantic Argument

Proof rules

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

$$\frac{I \models F \wedge G}{\begin{array}{l} I \models F \\ I \models G \end{array}} \ \leftarrow\text{and} \qquad\qquad \frac{I \not\models F \wedge G}{I \not\models F \quad \mid \quad I \not\models G} \ (\nwarrow \text{or})$$

$$\frac{I \models F \vee G}{I \models F \quad \mid \quad I \models G} \qquad\qquad \frac{I \not\models F \vee G}{\begin{array}{l} I \not\models F \\ I \not\models G \end{array}}$$

$$\frac{I \models F \rightarrow G}{I \not\models F \quad \mid \quad I \models G} \qquad\qquad \frac{I \not\models F \rightarrow G}{\begin{array}{l} I \models F \\ I \not\models G \end{array}}$$

$$\frac{I \models F \leftrightarrow G}{I \models F \wedge G \quad \mid \quad I \not\models F \vee G} \qquad \frac{I \not\models F \leftrightarrow G}{I \models F \wedge \neg G \quad \mid \quad I \models \neg F \wedge G}$$

$$\frac{\begin{array}{l} I \models F \\ I \not\models F \end{array}}{I \models \bot}$$

Example 1: Prove

$$F: \; P \wedge Q \rightarrow P \vee \neg Q \quad \text{is valid.}$$

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.

Example 2: Prove

$$F : (P \rightarrow Q) \wedge (Q \rightarrow R) \rightarrow (P \rightarrow R) \quad \text{is valid.}$$

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 $\wedge$

Two cases from 6

$$\begin{array}{lcccl} 8a. & I & \not\models & P & \text{6 and } \rightarrow \\ 9a. & I & \models & \bot & \text{4 and 8a are contradictory} \end{array}$$

and

$$\begin{array}{lcccl} 8b. & I & \models & Q & \text{6 and } \rightarrow \end{array}$$

Two cases from 7

$$\begin{array}{lcccl} 9ba. & I & \not\models & Q & \text{7 and } \rightarrow \\ 10ba. & I & \models & \bot & \text{8b and 9ba are contradictory} \end{array}$$

and

$$\begin{array}{lcccl} 9bb. & I & \models & R & \text{7 and } \rightarrow \\ 10bb. & I & \models & \bot & \text{5 and 9bb are contradictory} \end{array}$$

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

Example 3: Is

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

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:

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

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

Equivalence

F_1 and F_2 are <u>equivalent</u> ($F_1 \Leftrightarrow F_2$)
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 <u>implies</u> 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 use recursively the following template equivalences (left-to-right):

$$\neg\neg F_1 \Leftrightarrow F_1 \qquad \neg\top \Leftrightarrow \bot \qquad \neg\bot \Leftrightarrow \top$$

$$\left.\begin{array}{l} \neg(F_1 \wedge F_2) \Leftrightarrow \neg F_1 \vee \neg F_2 \\ \neg(F_1 \vee F_2) \Leftrightarrow \neg F_1 \wedge \neg F_2 \end{array}\right\} \text{De Morgan's Law}$$

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

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

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

$F' : \neg(\neg P \vee \neg(P \wedge Q))$	$\rightarrow$ to $\vee$
$F'' : \neg\neg P \wedge \neg\neg(P \wedge Q)$	De Morgan's Law
$F''' : P \wedge P \wedge Q$	$\neg\neg$

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

2. Disjunctive Normal Form (DNF)

Disjunction of conjunctions of literals

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

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

$$\left.\begin{array}{lcl} (F_1 \lor F_2) \land F_3 & \Leftrightarrow & (F_1 \land F_3) \lor (F_2 \land F_3) \\ F_1 \land (F_2 \lor F_3) & \Leftrightarrow & (F_1 \land F_2) \lor (F_1 \land F_3) \end{array}\right\} \textit{dist}$$

Example: Convert

$$F : (Q_1 \lor \neg\neg Q_2) \land (\neg R_1 \to R_2) \text{ into DNF}$$

$$\begin{array}{rll} F' & : (Q_1 \lor Q_2) \land (R_1 \lor R_2) & \text{in NNF} \\ F'' & : (Q_1 \land (R_1 \lor R_2)) \lor (Q_2 \land (R_1 \lor R_2)) & \text{dist} \\ F''' & : (Q_1 \land R_1) \lor (Q_1 \land R_2) \lor (Q_2 \land R_1) \lor (Q_2 \land R_2) & \text{dist} \end{array}$$

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

3. Conjunctive Normal Form (CNF)

Conjunction of disjunctions of literals

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

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

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

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

```
let rec DPLL F =
  let F' = BCP F in
  if F' = ⊤ then true
  else if F' = ⊥ then false
  else
     let P = CHOOSE vars(F') in
     (DPLL F'{P ↦ ⊤}) ∨ (DPLL F'{P ↦ ⊥})
```

Don't CHOOSE only-positive or only-negative variables for splitting.

Boolean Constraint Propagation (BCP)

Based on unit resolution

$$\frac{\ell \qquad C[\neg\ell]}{C[\bot]} \leftarrow \text{clause} \qquad \textit{where } \ell = P \textit{ or } \ell = \neg P$$

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

$$\frac{R \qquad (\neg R)}{\bot}$$

$F\{Q \mapsto \top\} = \bot \Rightarrow$ false

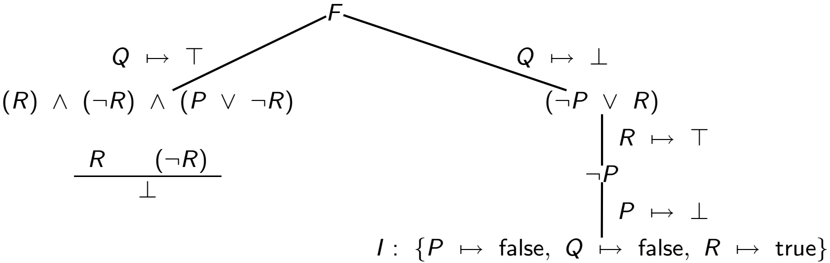
On the other branch

$F\{Q \mapsto \bot\} : (\neg P \lor R)$

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

F is satisfiable with satisfying interpretation

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



THE CALCULUS OF COMPUTATION:
Decision Procedures with
Applications to Verification

by
Aaron Bradley
Zohar Manna

Springer 2007

2. First-Order Logic (FOL)

First-Order Logic (FOL)

Also called Predicate Logic or Predicate Calculus

FOL Syntax

variables	$x, y, z, \cdots$
constants	$a, b, c, \cdots$
functions	$f, g, h, \cdots$
terms	variables, constants or n-ary function applied to n terms as arguments $a, x, f(a), g(x, b), f(g(x, g(b)))$
predicates	$p, q, r, \cdots$
atom	$\top$, $\bot$, or an n-ary predicate applied to n terms
literal	atom or its negation $p(f(x), g(x, f(x))), \quad \neg p(f(x), g(x, f(x)))$

Note: 0-ary functions: constant
0-ary predicates: $P, Q, R, \ldots$

quantifiers

existential quantifier $\exists x. F[x]$
"there exists an x such that $F[x]$"

universal quantifier $\forall x. F[x]$
"for all x, $F[x]$"

FOL formula literal, application of logical connectives ($\neg$, $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$) to formulae, or application of a quantifier to a formula

Example: FOL formula

$$\forall x.\ \underbrace{p(f(x),x)\ \rightarrow\ (\exists y.\ \underbrace{p(f(g(x,y)),g(x,y))}_{G})\ \wedge\ q(x,f(x))}_{F}$$

The scope of $\forall x$ is F.
The scope of $\exists y$ is G.
The formula reads:

"for all x,
if $p(f(x),x)$
then there exists a y such that
$p(f(g(x,y)),g(x,y))$ and $q(x,f(x))$"

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

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

- Fermat's Last Theorem.

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

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

    $$f_I : \ D_I^n \to D_I$$

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

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

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

Example:

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

Interpretation $I : (D_I, \alpha_I)$

$D_I = \mathbb{Z} = \{\cdots, -2, -1, 0, 1, 2, \cdots\}$ 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

Semantics: Quantifiers

x variable.
<u>x-variant</u> of interpretation I is an interpretation $J : (D_J, \alpha_J)$ such that

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

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

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

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

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

$J_1 : I \triangleleft \{x \mapsto \mathsf{v}\}$	$J_2 : J_1 \triangleleft \{y \mapsto \frac{\mathsf{v}}{2}\}$
x-variant of I	y-variant of J_1

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

Then

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

Satisfiability and Validity

F is <u>satisfiable</u> iff there exists I s.t. $I \models F$
F is <u>valid</u> iff for all I, $I \models F$

$$F \text{ is valid iff } \neg F \text{ is unsatisfiable}$$

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

Suppose not. Then there is I s.t.

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

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.	$I \triangleleft \{x \mapsto \mathsf{v}\}$	$\models$	$\neg p(x)$	3 and $\exists$, for some $\mathsf{v} \in D_I$
5.	$I \triangleleft \{x \mapsto \mathsf{v}\}$	$\models$	$p(x)$	1 and $\forall$

4 and 5 are contradictory.

Second case

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

3 and 6 are contradictory.

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

Example: Prove

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

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.	$I \triangleleft \{x \mapsto \alpha_I[a]\}$	$\not\models$	$p(x)$	3 and $\exists$

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

Example: Show

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

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 $D_I = \{0,1\}$

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

$p_I(1,0)$ and $p_I(1,0)$ are false

I falsifying interpretation $\Rightarrow$ F is invalid.

Safe Substitution $F\sigma$

Example:

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

bound by $\forall x$ ↗ ↖ free   free ↗ ↖ free

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

substitution

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

$F\sigma$?

1. Rename

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

$$\uparrow \qquad \uparrow$$

where x' is a fresh variable

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

Rename x by x':

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

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

Same for $\exists x$

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

where x' is a fresh variable

Proposition (Substitution of Equivalent Formulae)

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

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

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

Formula Schema

Formula

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

Formula Schema

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

$\uparrow$ place holder

Formula Schema (with side condition)

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

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 σ of H

$$\sigma : \{F_1 \mapsto \quad , \ldots , F_n \mapsto \quad \}$$

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
σ is a substitution obeying H's side conditions
then $H\sigma$ is also valid.

Example:

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

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

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 $x \notin$ free(F)
4.	I	$\models$	$\bot$	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 $x \notin$ free(F)
5.	I	$\models$	$\bot$	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

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

1. $\forall x.\ (\exists y.\ p(x,y) \;\land\; p(x,z)) \;\rightarrow\; \exists w.\ p(x,w)$
2. $\forall x.\ \neg(\exists y.\ p(x,y) \;\land\; p(x,z)) \;\lor\; \exists w.\ p(x,w)$
   $F_1 \;\rightarrow\; F_2 \;\Leftrightarrow\; \neg F_1 \;\lor\; F_2$
3. $\forall x.\ (\forall y.\ \neg(p(x,y) \;\land\; p(x,z))) \;\lor\; \exists w.\ p(x,w)$
   $\neg \exists x.\ F[x] \;\Leftrightarrow\; \forall x.\ \neg F[x]$
4. $\forall x.\ (\forall y.\ \neg p(x,y) \;\lor\; \neg p(x,z)) \;\lor\; \exists w.\ p(x,w)$

2. Prenex Normal Form (PNF)

All quantifiers appear at the beginning of the formula

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

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

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

$\uparrow$ 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 $\cdots \forall x \cdots \forall y \cdots$

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

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 $I \models \perp$

The Calculus of Computation: Decision Procedures with Applications to Verification

by

Aaron Bradley
Zohar Manna

Springer 2007

3. First-Order Theories

First-Order Theories

First-order theory T defined by

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

Σ-formula constructed of constants, functions, and predicate symbols from Σ, and variables, logical connectives, and quantifiers

The symbols of Σ are just symbols without prior meaning — the axioms of T provide their meaning

A Σ-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. $I \models F$

A Σ-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 Σ-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 Σ-formula F in the fragment.

Theory of Equality T_E

Signature

$$\Sigma_= : \{=, a, b, c, \cdots, f, g, h, \cdots, p, q, r, \cdots\}$$

consists of

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

Axioms of T_E

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

Congruence and Equivalence are axiom schemata. For example,
Congruence for binary function f_2 for $n = 2$:

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

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

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

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	$\mathbb{N} = \{0, 1, 2, \cdots\}$
Integers	$\mathbb{Z} = \{\cdots, -2, -1, 0, 1, 2, \cdots\}$

Three variations:

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

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

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

The axioms:

1. $\forall x.\ \neg(x + 1 = 0)$ (zero)
2. $\forall x, y.\ x + 1 = y + 1 \ \rightarrow\ x = y$ (successor)
3. $F[0]\ \wedge\ (\forall x.\ F[x]\ \rightarrow\ F[x + 1])\ \rightarrow\ \forall x.\ F[x]$ (induction)
4. $\forall x.\ x + 0 = x$ (plus zero)
5. $\forall x, y.\ x + (y + 1) = (x + y) + 1$ (plus successor)
6. $\forall x.\ x \cdot 0 = 0$ (times zero)
7. $\forall x, y.\ x \cdot (y + 1) = x \cdot y + x$ (times successor)

Line 3 is an axiom schema.

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

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

We have $>$ and $\geq$ since

$3x + 5 > 2y$ write as $\exists z.\ z \neq 0 \ \wedge\ 3x + 5 = 2y + z$

$3x + 5 \geq 2y$ write as $\exists z.\ 3x + 5 = 2y + z$

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 Σ_{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

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

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

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

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

3 is an axiom schema.

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

3. Theory of Integers $T_{\mathbb{Z}}$

$\Sigma_{\mathbb{Z}}: \{\ldots, -2, -1, 0, 1, 2, \ldots, -3\cdot, -2\cdot, 2\cdot, 3\cdot, \ldots, +, -, =, >\}$

where

- $\ldots, -2, -1, 0, 1, 2, \ldots$ are constants
- $\ldots, -3\cdot, -2\cdot, 2\cdot, 3\cdot, \ldots$ are unary functions
  (intended $2 \cdot x$ is $2x$)
- $+, -, =, >$

$T_{\mathbb{Z}}$ and $T_{\mathbb{N}}$ have the same expressiveness

- Every $T_{\mathbb{Z}}$-formula can be reduced to $\Sigma_{\mathbb{N}}$-formula.

Example: Consider the $T_{\mathbb{Z}}$-formula

$F_0: \forall w, x.\ \exists y, z.\ x + 2y - z - 13 > -3w + 5$

Introduce two variables, v_p and v_n (range over the nonnegative integers) for each variable v (range over the integers) of F_0

$$F_1 : \quad \begin{array}{l} \forall w_p, w_n, x_p, x_n.\ \exists y_p, y_n, z_p, z_n. \\ \quad (x_p - x_n) + 2(y_p - y_n) - (z_p - z_n) - 13 > -3(w_p - w_n) + 5 \end{array}$$

Eliminate $-$ by moving to the other side of $>$

$$F_2 : \quad \begin{array}{l} \forall w_p, w_n, x_p, x_n.\ \exists y_p, y_n, z_p, z_n. \\ \quad x_p + 2y_p + z_n + 3w_p > x_n + 2y_n + z_p + 13 + 3w_n + 5 \end{array}$$

Eliminate $>$

$$F_3 : \quad \begin{array}{l} \forall w_p, w_n, x_p, x_n.\ \exists y_p, y_n, z_p, z_n.\ \exists u. \\ \quad \neg(u = 0) \ \wedge \\ \quad x_p + y_p + y_p + z_n + w_p + w_p + w_p \\ \qquad = x_n + y_n + y_n + z_p + w_n + w_n + w_n + u \\ \qquad\quad +1+1+1+1+1+1+1+1+1 \\ \qquad\quad +1+1+1+1+1+1+1+1+1\ . \end{array}$$

which is a $T_{\mathbb{N}}$-formula equivalent to F_0.

- Every $T_{\mathbb{N}}$-formula can be reduced to $\Sigma_{\mathbb{Z}}$-formula.

Example: To decide the $T_{\mathbb{N}}$-validity of the $T_{\mathbb{N}}$-formula

$$\forall x.\ \exists y.\ x = y + 1$$

decide the $T_{\mathbb{Z}}$-validity of the $T_{\mathbb{Z}}$-formula

$$\forall x.\ x \geq 0 \ \rightarrow\ \exists y.\ y \geq 0 \ \wedge\ x = y + 1\ ,$$

where $t_1 \geq t_2$ expands to $t_1 = t_2 \ \vee\ t_1 > t_2$

$T_{\mathbb{Z}}$-satisfiability and $T_{\mathbb{N}}$-validity is decidable

Rationals and Reals

$$\Sigma = \{0,\ 1,\ +,\ -,\ =,\ \geq\}$$

- Theory of Reals $T_{\mathbb{R}}$ (with multiplication)

$$x^2 = 2 \quad \Rightarrow \quad x = \pm\sqrt{2}$$

- Theory of Rationals $T_{\mathbb{Q}}$ (no multiplication)

$$\underbrace{2x}_{x+x} = 7 \quad \Rightarrow \quad x = \frac{2}{7}$$

Note: Strict inequality OK

$$\forall x, y.\ \exists z.\ x + y > z$$

rewrite as

$$\forall x, y.\ \exists z.\ \neg(x + y = z)\ \wedge\ x + y \geq z$$

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_{cons}

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

where

cons(a, b) – list constructed by concatenating a and b
car(x) – left projector of x: $\text{car}(\text{cons}(a, b)) = a$
cdr(x) – right projector of x: $\text{cdr}(\text{cons}(a, b)) = b$
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_{cons} is undecidable
Quantifier-free fragment of T_{cons} is efficiently decidable

2. Lists + equality

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

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

(this includes uninterpreted constants, functions, and predicates)

Axioms: union of the axioms of T_{E} and T_{cons}

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

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

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

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

Suppose not; then there exists a $T_{\text{cons}}^{=}$-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 $T_{\text{cons}}^{=}$-valid.

Theory of Arrays

1. Theory of Arrays T_A

Signature

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

where

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

Axioms

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

Note: $=$ is only defined for array elements

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

not T_A-valid, but

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

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:

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

is $T_A^=$-valid.

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

Combination of Theories

How do we show that

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

is $(T_{\mathsf{E}} \ \cup \ T_{\mathbb{Z}})$-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

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

The combined theory $T_1 \ \cup \ T_2$ has

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

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.

The Calculus of Computation: Decision Procedures with Applications to Verification

by
Aaron Bradley
Zohar Manna

Springer 2007

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

Stepwise Induction (Peano Arithmetic T_{PA})

Axiom schema (induction)

$$
\begin{array}{ll}
F[0] \wedge & \ldots \text{ base case} \\
(\forall n.\ F[n] \rightarrow F[n+1]) & \ldots \text{ inductive step} \\
\rightarrow \forall x.\ F[x] & \ldots \text{ conclusion}
\end{array}
$$

for Σ_{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.

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)
- $\forall x, z.\ exp_3(x, 0, z) = z$ (P0)
- $\forall x, y, z.\ exp_3(x, y+1, z) = exp_3(x, y, x \cdot z)$ (P1)

Prove that

$$\boxed{\forall x, y.\ exp_3(x, y, 1) = x^y}$$

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

First attempt:

$$\forall y\ [\underbrace{\forall x.\ exp_3(x, y, 1) = x^y}_{F[y]}]$$

We chose induction on y. Why?

Base case:

$F[0]: \ \forall x.\ exp_3(x, 0, 1) = x^0$

OK since $exp_3(x, 0, 1) = 1$ (P0) and $x^0 = 1$ (E0).

Inductive step: Failure.

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

$F[n + 1] : \forall x.\ exp_3(x, n + 1, 1) = x^{n+1}$

from the inductive hypothesis

$F[n] : \forall x.\ exp_3(x, n, 1) = x^n$

Second attempt: Strengthening

Strengthened property

$$\boxed{\forall x, y, z.\ exp_3(x, y, z) = x^y \cdot z}$$

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

$$\forall x, y.\ exp_3(x, y, 1) = x^y$$

Again, induction on y

$$\forall y\ [\underbrace{\forall x, z.\ exp_3(x, y, z) = x^y \cdot z}_{F[y]}]$$

Base case:

$$F[0] :\ \forall x, z.\ exp_3(x, 0, z) = x^0 \cdot z$$

OK since $exp_3(x, 0, z) = z$ (P0) and $x^0 = 1$ (E0).

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

Assume inductive hypothesis

$$F[n] : \forall x, z.\ exp_3(x, n, z) = x^n \cdot z \qquad \text{(IH)}$$

prove

$$F[n+1] : \forall x, \underset{\uparrow}{z'}.\ exp_3(x, n+1, z') = x^{n+1} \cdot z'$$

$$\begin{aligned} exp_3(x, n+1, z') &= exp_3(x, n, x \cdot z') && \text{(P1)} \\ &= x^n \cdot (x \cdot z') && \text{IH } F[n], z \mapsto x \cdot z' \\ &= x^{n+1} \cdot z' && \text{(E1)} \end{aligned}$$

Stepwise Induction (Lists T_{cons})

Axiom schema (induction)

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

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

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

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

it suffices to show

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

Example

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

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

Prove

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

is T^+_{cons}-valid.

Base case: For arbitrary atom u,

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

by R0.

Inductive step: For arbitrary lists u, v,

assume the inductive hypothesis

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

Prove

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

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

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

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

Case $\text{atom}(u)$

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

by (F1).

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

Complete Induction (Peano Arithmetic T_{PA})

Axiom schema (complete induction)

$$(\forall n.\ (\forall n'.\ n' < n \ \rightarrow\ F[n']) \ \rightarrow\ F[n]) \qquad \ldots \text{ inductive step}$$
$$\rightarrow\ \forall x.\ F[x] \qquad \ldots \text{ conclusion}$$

for Σ_{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.

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 $quot(5, 3) = 1$ and $rem(5, 3) = 2$

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

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

Prove

(1) $\forall x, y.\ y > 0 \ \rightarrow\ rem(x, y) < y$

(2) $\forall x, y.\ y > 0 \ \rightarrow\ x = y \cdot quot(x, y) + rem(x, y)$

Best proved by complete induction.

Proof of (1)

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

Consider an arbitrary natural number x.
Assume the inductive hypothesis

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

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

Let y be an arbitrary positive integer

Case $x < y$:

$$\begin{array}{lcll} rem(x,y) & = & x & \text{by (R0)} \\ & < & y & \text{case} \end{array}$$

Case $\neg(x < y)$:

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

$$\begin{array}{lcll} rem(x,y) & = & rem(n+y,y) & x = n + y \\ & = & rem(n,y) & \text{(R1)} \\ & < & y & \text{IH } (x' \mapsto n, y' \mapsto y) \\ & & & \text{since } n < x \text{ and } y > 0 \end{array}$$

Well-founded Induction

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

$$s_1 \succ s_2 \succ s_3 \succ \cdots$$

Note: where $s \prec t$ 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.

  $$1 > \frac{1}{2} > \frac{1}{3} > \frac{1}{4} > \cdots$$

  is an infinite decreasing sequence.
- The strict sublist relation $\prec_c$ is well-founded on the set of all lists.

Well-founded Induction Principle

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

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

for Σ-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 $n' \prec n$
  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 $\prec$ is $<$.

Lexicographic Relation

Given pairs of sets and well-founded relations

$$(S_1, \prec_1), \ldots, (S_m, \prec_m)$$

Construct

$$S = S_1 \times \ldots, S_m$$

Define lexicographic relation $\prec$ over S as

$$\underbrace{(s_1, \ldots, s_m)}_{s} \prec \underbrace{(t_1, \ldots, t_m)}_{t} \Leftrightarrow \bigvee_{i=1}^{m} \left(s_i \prec_i t_i \ \wedge \ \bigwedge_{j=1}^{i-1} s_j = t_j \right)$$

for $s_i, t_i \in S_i$.

- If $(S_1, \prec_1), \ldots, (S_m, \prec_m)$ are well-founded relations, so is $(S, \prec)$.

Lexicographic well-founded induction principle

For theory T and well-founded lexicographic relation $\prec$,

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

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

Same as regular well-founded induction, just

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

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.

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

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

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

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

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

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:

$ack(0, 0) = 1$
$ack(1, 1) = 3$
$ack(2, 2) = 7$
$ack(3, 3) = 61$

$$ack(4, 4) = 2^{2^{2^{2^{16}}}} - 3$$

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

$$\forall x, y.\ ack(x, y) > y$$

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

Consider arbitrary natural numbers x, y.
Assume the inductive hypothesis

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

Show

$$F[x, y] : ack(x, y) > y.$$

Case $x = 0$:

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

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

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

Since

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

Then

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

Thus

$$ack(x,0) = ack(x-1,1) > 1 > 0$$

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

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

Since

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

Then

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

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

Furthermore, since

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

then

$$ack(x, y-1) > y-1 \qquad (3)$$

By (1)–(3), we have

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

Hence

$$ack(x, y) > (y-1)+1 = y$$

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:

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

$F : \bot \;\Rightarrow\; F' : \neg\top$

$F : P \;\Rightarrow\; F' : P$ 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).

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

$F : G_1 \wedge G_2 \quad \Rightarrow \quad F' : \neg(\neg G_1' \vee \neg G_2')$

$F : G_1 \rightarrow G_2 \quad \Rightarrow \quad F' : \neg G_1' \vee G_2'$

$F : G_1 \leftrightarrow G_2 \quad \Rightarrow \quad F' : \cdots$

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

The Calculus of Computation:
Decision Procedures with
Applications to Verification

by
Aaron Bradley
Zohar Manna

Springer 2007

5. Program Correctness: Mechanics

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 @ ⊤
    (int i := ℓ; i ≤ u; i := i + 1) {
    if (a[i] = e) return true;
  }
  return false;
}
```

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 ℓ,

execute the body and increment i by 1

as long as $i \leq n$

@ - program annotation

Program B: BinarySearch with function specification

```
@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) return BinarySearch(a, m + 1, u, e);
    else return BinarySearch(a, ℓ, 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, $T_{\mathbb{Z} \cup A}$

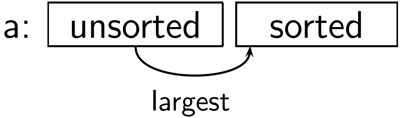
Function specifications

- 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 ⊤
@post sorted(rv, 0, |rv| − 1)
int[] BubbleSort(int[] a_0) {
  int[] a := a_0;
  for @ ⊤
    (int i := |a| − 1; i > 0; i := i − 1) {
    for @ ⊤
      (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;
}
```

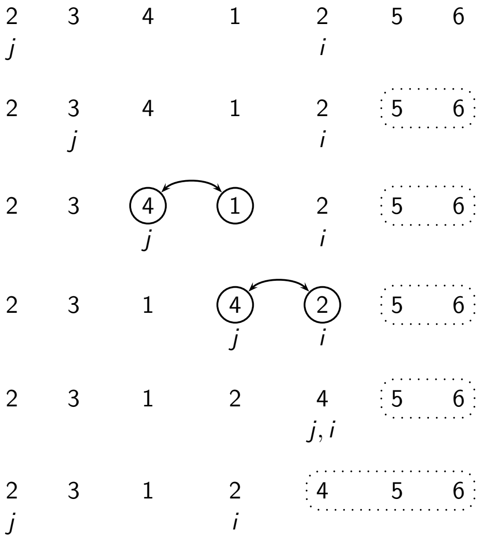
Function BubbleSort sorts integer array *a*



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 Annotation

- Function Specifications

  function postcondition (@*post*)

  function precondition (@*pre*)

- Runtime Assertions

  e.g., $@\ 0 \le j < |a| \ \land \ 0 \le j+1 < |a|$

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

- Loop Invariants

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

Program A: LinearSearch with runtime assertions

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

Program B: BinarySearch with runtime assertions

```
@pre ⊤
@post ⊤
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 ⊤
@post ⊤
int[] BubbleSort(int[] a₀) {
  int[] a := a₀;
  for @ ⊤
    (int i := |a| − 1; i > 0; i := i − 1) {
    for @ ⊤
      (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;
}
```

Loop Invariants

$\texttt{while}$
  $@\ F$
  $\langle cond \rangle$ { $\langle body \rangle$ }

- 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

$\texttt{for}$
  $@\ F$
  $(\langle init \rangle; \langle cond \rangle; \langle incr \rangle)$ { $\langle body \rangle$ }

$\Rightarrow$

$\langle init \rangle;$
$\texttt{while}$
  $@\ F$
  $\langle cond \rangle$ { $\langle body \rangle$ $\langle incr \rangle$ }

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;
}
```

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

@ $\leftarrow$ precondition or loop invariant

sequence of instructions
(with no loop invariants)

@ $\leftarrow$ loop invariant, assertion, or postcondition

Program A: LinearSearch

Basic Paths of LinearSearch

(1)

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

$i := \ell;$

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

(2)

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

$\texttt{assume } i \leq u;$

$\texttt{assume } a[i] = e;$

$rv := \texttt{true};$

$@\text{post } rv \ \leftrightarrow \ \exists j.\ \ell \leq j \leq u \ \wedge \ a[j] = e$

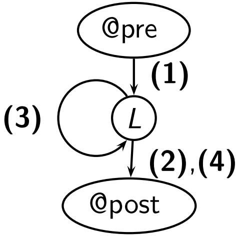
(3)

$$@L:\ \ell \le i\ \wedge\ \forall j.\ \ell \le j < i\ \rightarrow\ a[j] \ne e$$
`assume` $i \le u$;
`assume` $a[i] \ne e$;
$i := i + 1$;
$$@L:\ \ell \le i\ \wedge\ \forall j.\ \ell \le j < i\ \rightarrow\ a[j] \ne e$$

(4)

$$@L:\ \ell \le i\ \wedge\ \forall j.\ \ell \le j < i\ \rightarrow\ a[j] \ne e$$
`assume` $i > u$;
rv := `false`;
$$@\text{post}\ rv\ \leftrightarrow\ \exists j.\ \ell \le j \le u\ \wedge\ a[j] = e$$

Visualization of basic paths of LinearSearch



Program C: BubbleSort with loop invariants

```
@pre ⊤
@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)
            ∧ sorted(a, i, |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;
}
```

$$@L_2 : \begin{bmatrix} 1 \leq i < |a| \;\land\; 0 \leq j \leq i \\ \land\ \mathrm{partitioned}(a, 0, i, i+1, |a|-1) \\ \land\ \mathrm{partitioned}(a, 0, j-1, j, j) \\ \land\ \mathrm{sorted}(a, i, |a|-1) \end{bmatrix}$$

Partition

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

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

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 $\top$;
$a := a_0$;
$i := |a| - 1$;
@L_1 : $-1 \leq i < |a| \ \wedge \ \text{partitioned}(a, 0, i, i+1, |a|-1)$
$\quad \wedge \ \text{sorted}(a, i, |a|-1)$

(2)

$$@L_1: \ -1 \leq i < |a| \ \wedge \ \text{partitioned}(a, 0, i, i+1, |a|-1) \\ \wedge \ \text{sorted}(a, i, |a|-1)$$

`assume` $i > 0$;

$j := 0$;

$$@L_2: \begin{bmatrix} 1 \leq i < |a| \ \wedge \ 0 \leq j \leq i \ \wedge \ \text{partitioned}(a, 0, i, i+1, |a|-1) \\ \wedge \ \text{partitioned}(a, 0, j-1, j, j) \ \wedge \ \text{sorted}(a, i, |a|-1) \end{bmatrix}$$

(3)

$$@L_2: \begin{bmatrix} 1 \leq i < |a| \ \wedge \ 0 \leq j \leq i \ \wedge \ \text{partitioned}(a, 0, i, i+1, |a|-1) \\ \wedge \ \text{partitioned}(a, 0, j-1, j, j) \ \wedge \ \text{sorted}(a, i, |a|-1) \end{bmatrix}$$

`assume` $j < i$;

`assume` $a[j] > a[j+1]$;

$t := a[j]$;

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

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

$j := j+1$;

$$@L_2: \begin{bmatrix} 1 \leq i < |a| \ \wedge \ 0 \leq j \leq i \ \wedge \ \text{partitioned}(a, 0, i, i+1, |a|-1) \\ \wedge \ \text{partitioned}(a, 0, j-1, j, j) \ \wedge \ \text{sorted}(a, i, |a|-1) \end{bmatrix}$$

(4)

$$@L_2 : \begin{bmatrix} 1 \le i < |a| \;\wedge\; 0 \le j \le i \;\wedge\; \text{partitioned}(a, 0, i, i+1, |a|-1) \\ \wedge\; \text{partitioned}(a, 0, j-1, j, j) \;\wedge\; \text{sorted}(a, i, |a|-1) \end{bmatrix}$$

`assume` $j < i$;

`assume` $a[j] \le a[j+1]$;

$j := j + 1$;

$$@L_2 : \begin{bmatrix} 1 \le i < |a| \;\wedge\; 0 \le j \le i \;\wedge\; \text{partitioned}(a, 0, i, i+1, |a|-1) \\ \wedge\; \text{partitioned}(a, 0, j-1, j, j) \;\wedge\; \text{sorted}(a, i, |a|-1) \end{bmatrix}$$

---

(5)

$$@L_2 : \begin{bmatrix} 1 \le i < |a| \;\wedge\; 0 \le j \le i \;\wedge\; \text{partitioned}(a, 0, i, i+1, |a|-1) \\ \wedge\; \text{partitioned}(a, 0, j-1, j, j) \;\wedge\; \text{sorted}(a, i, |a|-1) \end{bmatrix}$$

`assume` $j \ge i$;

$i := i - 1$;

$$@L_1 : \; -1 \le i < |a| \;\wedge\; \text{partitioned}(a, 0, i, i+1, |a|-1) \;\wedge\; \text{sorted}(a, i, |a|-1)$$

---

(6)

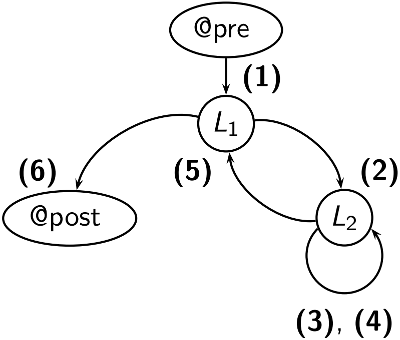
$$@L_1 : \ -1 \leq i < |a| \ \land \ \text{partitioned}(a, 0, i, i+1, |a|-1) \ \land$$
$$\text{sorted}(a, i, |a|-1)$$

$\texttt{assume}\ i \leq 0;$

$rv := a;$

$@\text{post sorted}(rv, 0, |rv|-1)$

Visualization of basic paths of BubbleSort



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

$$@\text{pre } 0 \le \ell \ \wedge \ u < |a| \ \wedge \ \text{sorted}(a, \ell, u) \qquad \ldots F[a, \ell, u, e]$$

$\vdots$

$$@R_1 : \ 0 \le m+1 \ \wedge \ u < |a| \ \wedge \ \text{sorted}(a, m+1, u) \qquad \ldots F[a, m+1, u, e]$$

`return` BinarySearch$(a, m+1, u, e)$

$\vdots$

$$@R_2 : \ 0 \le \ell \ \wedge \ m-1 < |a| \ \wedge \ \text{sorted}(a, \ell, m-1) \qquad \ldots F[a, \ell, m-1, e]$$

`return` BinarySearch$(a, \ell, m-1, e)$

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

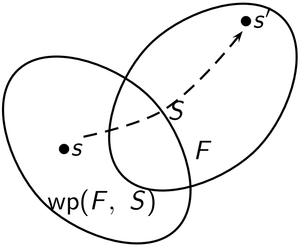
- Program counter pc — holds current location of control
- State s — assignment of values to all variables

  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 $\text{wp}(F, S)$

  For FOL formula F, program statement S,
  If $s \models \text{wp}(F, S)$ and if statement S is executed on state s to produce state s', then $s' \models F$



Weakest Precondition $\text{wp}(F,\ S)$

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

Verification Condition of basic path

$$
\begin{array}{l}
\texttt{@}\ F \\
S_1; \\
\ldots \\
S_n; \\
\texttt{@}\ G
\end{array}
$$

is

$$F \ \rightarrow\ \text{wp}(G,\ S_1; \ldots; S_n)$$

Also denoted by

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

Example: Basic path

(1)

@ $F: \; x \geq 0$
$S_1: \; x := x + 1;$
@ $G: \; x \geq 1$

The VC is

$$F \;\rightarrow\; \mathrm{wp}(G,\; S_1)$$

That is,

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

Therefore the VC of path (1)

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

which is $T_{\mathbb{Z}}$-valid.

Example: Basic path (2) of LinearSearch

(2)

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

The VC is

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

That is,

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

Therefore the VC of path (2)

$$\begin{array}{l} \ell \le i \;\wedge\; (\forall j.\, \ell \le j < i \;\rightarrow\; a[j] \ne e) \\ \;\rightarrow\; (i \le u \;\rightarrow\; (a[i] = e \;\rightarrow\; \exists j.\, \ell \le j \le u \;\wedge\; a[j] = e)) \end{array} \tag{1}$$

or, equivalently,

$$\begin{array}{l} \ell \le i \;\wedge\; (\forall j.\, \ell \le j < i \;\rightarrow\; a[j] \ne e) \;\wedge\; i \le u \;\wedge\; a[i] = e \\ \;\rightarrow\; \exists j.\, \ell \le j \le u \;\wedge\; a[j] = e \end{array} \tag{2}$$

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 $(T_{\mathbb{Z}} \cup T_{\mathsf{A}})$-valid.

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

$s_0, s_1, s_2, \ldots$

such that

- $s_0[pc] = L_0$ and $s_0 \models F_0$, and
- for each i, s_{i+1} is the result of executing the instruction at $s_i[pc]$ on state s_i.

where $s_i[pc]$ = value of pc given by state s_i

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

$$s_i[pc] = L \quad \Rightarrow \quad s_i \models F$$

Annotations of P are <u>P-invariant</u> (<u>invariant</u>) iff each annotation of P is P-invariant at its location.

Annotations of P are <u>P-inductive</u> (<u>inductive</u>) iff all VCs generated from program P are T-valid

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

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-tupules of natural numbers with the lexicographic extension $<_n$
- Find function δ (ranking function)
  mapping
  $$\text{program states} \quad \rightarrow \quad S$$
  such that δ 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);
  }
}
```

- 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 $x \geq 0 \;\wedge\; y \geq 0$
$\downarrow (x, y)$
`assume` $x \neq 0$;
`assume` $y = 0$;
$\downarrow (x - 1, 1)$

(2)

@pre $x \geq 0 \;\wedge\; y \geq 0$
$\downarrow (x, y)$
`assume` $x \neq 0$;
`assume` $y \neq 0$;
$\downarrow (x, y - 1)$

(3)

@pre $x \geq 0 \ \wedge \ y \geq 0$
$\downarrow (x, y)$
`assume` $x \neq 0$;
`assume` $y \neq 0$;
`assume` $v_1 \geq 0$;
$z := v_1$;
$\downarrow (x - 1, z)$

Showing decrease of ranking function

For basic path with ranking function

$$
\begin{array}{l}
@\ F \\
\downarrow\ \delta[\overline{x}] \\
S_1; \\
\vdots \\
S_k; \\
\downarrow\ \kappa[\overline{x}]
\end{array}
$$

We must prove that

the value of κ after executing $S_1; \cdots ; S_n$

is less than

the value of δ before executing the statements

Thus, we show the verification condition

$$F \;\rightarrow\; \mathsf{wp}(\kappa \prec \delta[\overline{x}_0],\; S_1; \cdots ; S_k)\{\overline{x}_0 \;\mapsto\; \overline{x}\}\ .$$

Example: Ackermann function — recursive calls

Verification conditions for the three basic paths

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

Then compute

$$
\begin{aligned}
& \mathrm{wp}((x-1, z) <_2 (x_0, y_0) \\
& \quad , \texttt{assume } x \neq 0;\ \texttt{assume } y \neq 0;\ \texttt{assume } v_1 \geq 0;\ z := v_1) \\
\Leftrightarrow\ & \mathrm{wp}((x-1, v_1) <_2 (x_0, y_0) \\
& \quad , \texttt{assume } x \neq 0;\ \texttt{assume } y \neq 0;\ \texttt{assume } v_1 \geq 0) \\
\Leftrightarrow\ & x \neq 0 \;\wedge\; y \neq 0 \;\wedge\; v_1 \geq 0 \;\rightarrow\; (x-1, v_1) <_2 (x_0, y_0)
\end{aligned}
$$

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

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

Example: BubbleSort — loops

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

```
@pre ⊤
@post ⊤
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) {
```

```
    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;
}
```

We have to prove

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

The relevant basic paths

(1)

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

(2),(3)

$@L_2 : \ i+1 \geq 0 \ \wedge \ i-j \geq 0$
$\downarrow L_2 : \ (i+1, i-j)$
`assume` $j < i$;
$\ldots$
$j := j+1$;
$\downarrow L_2 : \ (i+1, i-j)$

(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.

The Calculus of Computation:
Decision Procedures with
Applications to Verification

by
Aaron Bradley
Zohar Manna

Springer 2007

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 : ℓ ≤ i ≤ u + 1
  (int i := ℓ; i ≤ u; i := i + 1) {
  if (a[i] = e) return true;
}
```

Example: BubbleSort

```
for
  @L_1 :  −1 ≤ i < |a|
  (int i := |a| − 1;  i > 0;  i := i − 1) {
  for
    @L_2 :  0 < i < |a|  ∧  0 ≤ j ≤ 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

$@L: \; F'$

$s_1;$

$\vdots \quad \Uparrow$

$s_n;$

$@L': \; F$

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

Example: BinarySearch

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

$(\cdot)$

$$
\begin{array}{l}
@\text{pre } H : ? \\
S_1 : \ \texttt{assume}\ \ell \leq u; \\
S_2 : \ m := (\ell + u)\ \texttt{div}\ 2; \\
@\ F : \ 0 \leq m < |a|
\end{array}
$$

Compute

$$
\begin{array}{l}
\text{wp}(F,\ S_1; S_2) \\
\Leftrightarrow\ \text{wp}(\text{wp}(F,\ m := (\ell + u)\ \texttt{div}\ 2),\ S_1) \\
\Leftrightarrow\ \text{wp}(F\{m \ \mapsto\ (\ell + u)\ \texttt{div}\ 2\},\ S_1) \\
\Leftrightarrow\ \text{wp}(F\{m \ \mapsto\ (\ell + u)\ \texttt{div}\ 2\},\ \texttt{assume}\ \ell \leq u) \\
\Leftrightarrow\ \ell \leq u \ \rightarrow\ F\{m \ \mapsto\ (\ell + u)\ \texttt{div}\ 2\} \\
\Leftrightarrow\ \ell \leq u \ \rightarrow\ 0 \leq (\ell + u)\ \texttt{div}\ 2 < |a| \\
\Leftarrow\ 0 \leq \ell \ \wedge\ u < |a|
\end{array}
$$

$$\boxed{\text{@pre } H: \; 0 \leq \ell \;\wedge\; u < |a|}$$

guaranteed

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

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

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

holds in every execution of BinarySearch in which the precondition

$$\text{@pre } 0 \leq \ell \;\wedge\; u < |a|$$

is satisfied.

THE CALCULUS OF COMPUTATION:
Decision Procedures with
Applications to Verification

by
Aaron Bradley
Zohar Manna

Springer 2007

Part II: Algorithm Reasoning

7. Quantified Linear Arithmetic

Quantifier Elimination (QE) — algorithm for elminiation 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 Σ-formula returns a quantifier-free Σ-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 $T_{\widehat{\mathbb{Z}}}$ be $T_{\mathbb{Z}}$ with divisibility predicates.
For $\Sigma_{\widehat{\mathbb{Z}}}$-formula

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

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

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

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

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

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

$$G_5:\ F_4$$

G_5 is quantifier-free and T-equivalent to G_1

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

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

<u>Lemma</u>:

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

<u>Example</u>: $\Sigma_{\mathbb{Z}}$-formula $\quad F: \ \exists x.\ 2x = y$

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

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

Intended interpretations:

$k \mid x$ holds iff k divides x 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

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

$\widehat{T_{\mathbb{Z}}}$ 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)

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

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 $k \in \mathbb{Z}^+$.

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:

$$x + x + y < z + 3z + 2y - 4x$$
$$\Downarrow$$
$$6x < 4z + y$$

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 δ' is the coefficient of x everywhere:

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

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 $h, k \in \mathbb{Z}^+$.

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 $\exists x.\ F[x]$

Step 5 (trickiest part):

Construct

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

of $F_4[x']$ by

(A) replacing literals $x' < a$ by $\top$

(B) replacing literals $b < x'$ by $\bot$

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.

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 δ (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 δ interval to the right of b^*

Example (cont):

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

$$\Downarrow$$

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

By step 5,

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

which simplifies to $\bot$. 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

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

which simplifies to

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

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

Example:

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

Isolate x terms

$$\exists x.\ (3x < 9\ \lor\ 13 < 7x)\ \land\ 2 \mid x\ ,$$

so

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

After multiplying coefficients by proper constants,

$$\exists x.\ (21x < 63\ \lor\ 39 < 21x)\ \land\ 42 \mid 21x\ ,$$

we replace $21x$ by x':

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

Then

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

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

$$F_5 : \ \begin{array}{l} \displaystyle\bigvee_{j=1}^{42} (42 \mid j \ \wedge \ 21 \mid j) \quad \vee \\ \quad \displaystyle\bigvee_{j=1}^{42} ((39 + j < 63 \ \vee \ 39 < 39 + j) \ \wedge \ 42 \mid 39 + j \ \wedge \ 21 \mid 39 + j) \end{array}$$

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.

Example:

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

Rewriting

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

Then

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

so by Step 4

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

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

However,

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

so

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

Simplifying,

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

and then

$$F_5 : \ 2 \mid y \ ,$$

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

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.

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

B. Eliminating Blocks of Quantifiers

$$\exists x_1. \cdots \exists x_n. \ F[x_1, \ldots, x_n]$$

where F quantifier-free.
Eliminating x_n (left elimination) produces

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

which is equivalent to

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

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

- $\exists x_1. \cdots \exists x_{n-1}. \ F_{-\infty}[x_1, \ldots, x_{n-1}, j]$
- $\exists x_1. \cdots \exists x_{n-1}. \ F_4[x_1, \ldots, x_{n-1}, b + j]$ 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 $x' < \ldots$ and two (B) literals $\ldots < x'$, we use right elimination.

$$F_{+\infty} = \bot \qquad \delta = \{13\} = 13 \qquad A = \{-26\}$$

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

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

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

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 is $\Sigma_{\mathbb{Q}}$-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)

$$\begin{array}{lcl} \neg(s < t) & \Leftrightarrow & t < s \ \vee \ t = s \\ \neg(s = t) & \Leftrightarrow & t < s \ \vee \ t > s \end{array}$$

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.,

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

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 $\bot$
  (C) atoms $x = c$ by $\bot$

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

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 $T_{\mathbb{Q}}$-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 $s \equiv t$, check whether $s \in S$ satisfies $F_4[s]$
  if $s \not\equiv t$, $\frac{s+t}{2}$ represents the whole interval (s, t), so check $F_4[\frac{s+t}{2}]$

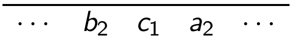
Intuition

Step 4 says that four cases are possible:

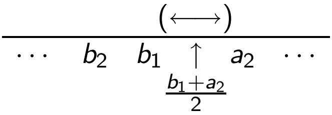
1. There is a left open interval s.t. all elements satisfy $F(x)$.

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

3. Some a_i, b_i, or c_i satisfies $F(x)$.



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



Example: $\Sigma_{\mathbb{Q}}$-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) $\Rightarrow$ $F_{-\infty} = \bot$
$x > \frac{13}{7}$ in (B) $\Rightarrow$ $F_{+\infty} = \bot$

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

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

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

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

simplifies to $\top$.

Thus, $F_4 : \top$ is $T_{\mathbb{Q}}$-equivalent to $\exists x.\ F[x]$,
so $\exists x.\ F[x]$ is $T_{\mathbb{Q}}$-valid.

THE CALCULUS OF COMPUTATION:
Decision Procedures with
Applications to Verification

by
Aaron Bradley
Zohar Manna

Springer 2007

8. Quantifier-Free Linear Arithmetic

Decision Procedures for Quantifier-free Fragments

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

Decide if
$F[x_1, \ldots, x_n]$ or $\exists x_1, \ldots, x_n.\ F[x_1, \ldots, x_n]$ is T-satisfiable

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

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

Note: no quantifier alternations

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

$F_1 \ \vee \ \ldots \ \vee \ F_k$

where each F_i conjunctive.

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

THE CALCULUS OF COMPUTATION: Decision Procedures with Applications to Verification

by
Aaron Bradley
Zohar Manna

Springer 2007

9. Quantifier-free Equality and Data Structures

The Theory of Equality T_E

$$\Sigma_E : \{=,\ a,\ b,\ c,\ \ldots,\ f,\ g,\ h,\ \ldots,\ p,\ q,\ r,\ \ldots\}$$

uninterpreted symbols:

- constants $a, b, c, \ldots$
- functions $f, g, h, \ldots$
- predicates $p, q, r, \ldots$

Example:

$x = y \ \wedge\ f(x) \neq f(y)$ T_E-unsatisfiable

$f(x) = f(y) \ \wedge\ x \neq y$ T_E-unsatisfiable

$f(f(f(a))) = a \ \wedge\ f(f(f(f(f(a))))) = a \ \wedge\ f(a) \neq a$

T_E-unsatisfiable

Axioms of T_E

1. $\forall x.\ x = x$ (reflexivity)
2. $\forall x, y.\ x = y \ \rightarrow\ y = x$ (symmetry)
3. $\forall x, y, z.\ x = y \ \wedge\ y = z \ \rightarrow\ x = z$ (transitivity)

define $=$ to be an equivalence relation.

Axiom schema

4. for each positive integer n and n-ary function symbol f,

$$\forall x_1, \ldots, x_n, y_1, \ldots, y_n.\ \bigwedge_i x_i = y_i \ \rightarrow\ f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \qquad \text{(congruence)}$$

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.

Axiom schema

5. for each positive integer n and n-ary predicate symbol p,

$$\forall x_1, \ldots, x_n, y_1, \ldots, y_n. \bigwedge_i x_i = y_i \rightarrow (p(x_1, \ldots, x_n) \leftrightarrow p(y_1, \ldots, y_n)) \qquad \text{(equivalence)}$$

Thus,

$$x = y \rightarrow (p(x) \leftrightarrow p(y))$$

is T_E-valid.

We discuss T_E-formulae without predicates

For example, for Σ_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

$$G: \; f_p(x) = \bullet \;\wedge\; f_q(x,y) = \bullet \;\wedge\; f_q(y,z) = \bullet \;\rightarrow\; f_q(x,z) \neq \bullet \; .$$

Equivalence and Congruence Relations: Basics

Binary relation R over set S

- is an equivalence relation if
  - reflexive: $\forall s \in S.\ sRs$;
  - symmetric: $\forall s_1, s_2 \in S.\ s_1 R s_2 \ \rightarrow\ s_2 R s_1$;
  - transitive: $\forall s_1, s_2, s_3 \in S.\ s_1 R s_2 \ \wedge\ s_2 R s_3 \ \rightarrow\ s_1 R s_3$.

Example:
Define the binary relation $\equiv_2$ over the set $\mathbb{Z}$ of integers

$$m \equiv_2 n \quad \text{iff} \quad (m \bmod 2) = (n \bmod 2)$$

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

$$\forall \overline{s}, \overline{t}.\ \bigwedge_{i=1}^{n} s_i R t_i \ \rightarrow\ f(\overline{s}) R f(\overline{t})\ .$$

Classes

For $\left\{ \begin{array}{c} \text{equivalence} \\ \text{congruence} \end{array} \right\}$ relation R over set S,

The $\left\{ \begin{array}{c} \underline{\text{equivalence}} \\ \underline{\text{congruence}} \end{array} \right\}$ 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 $\forall S_1, S_2 \in P.\ S_1 \cap S_2 = \emptyset$

Quotient

The quotient S/R of S by $\left\{ \begin{array}{c} \text{equivalence} \\ \text{congruence} \end{array} \right\}$relation R is the set of $\left\{ \begin{array}{c} \text{equivalence} \\ \text{congruence} \end{array} \right\}$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

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 $S = \{a, b\}$,
  $R_1 : \{aR_1 b\} \qquad R_2 : \{aR_2 b,\ bR_2 b\}$
  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 $\mathbb{Z}$
  $R_1 : \{xR_1 y \ : \ x \bmod 2 = y \bmod 2\}$
  $R_2 : \{xR_2 y \ : \ x \bmod 4 = y \bmod 4\}$
  Then $R_2 \prec R_1$.

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 Σ_E-formula

$$F: \; s_1 = t_1 \;\wedge\; \cdots \;\wedge\; s_m = t_m \;\wedge\; s_{m+1} \neq t_{m+1} \;\wedge\; \cdots \;\wedge\; s_n \neq t_n$$

decide if F is Σ_E-satisfiable.

Definition: For Σ_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

$$S_F = \{a, \; b, \; f(a,b), \; f(f(a,b),b)\} \; .$$

The Algorithm

Given Σ_E-formula F

$$F: \; s_1 = t_1 \; \wedge \; \cdots \; \wedge \; s_m = t_m \; \wedge \; s_{m+1} \neq t_{m+1} \; \wedge \; \cdots \; \wedge \; s_n \neq t_n$$

with subterm set S_F, F is T_E-satisfiable iff there exists a congruence relation $\sim$ over S_F such that

- for each $i \in \{1, \ldots, m\}$, $s_i \sim t_i$;
- for each $i \in \{m+1, \ldots, n\}$, $s_i \not\sim t_i$.

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.

$$F: \underbrace{s_1 = t_1 \ \wedge \ \cdots \ \wedge \ s_m = t_m}_{\text{generate congruence closure}} \wedge \underbrace{s_{m+1} \neq t_{m+1} \ \wedge \ \cdots \ \wedge \ s_n \neq t_n}_{\text{search for contradiction}}$$

The algorithm performs the following steps:

1. Construct the congruence closure $\sim$ of

   $$\{s_1 = t_1, \ldots, s_m = t_m\}$$

   over the subterm set S_F. Then

   $$\sim \models \ s_1 = t_1 \ \wedge \ \cdots \ \wedge \ s_m = t_m \ .$$

2. If for any $i \in \{m+1, \ldots, n\}$, $s_i \sim t_i$, return unsatisfiable.
3. Otherwise, $\sim \models F$, so return satisfiable.

How do we actually construct the congruence closure in Step 1?

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 $i \in \{1, \ldots, m\}$, 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 Σ_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

$\{f(a,b)\}$ and $\{a\}$

to form partition

2. $\{\{a, f(a,b)\}, \ \{b\}, \ \{f(f(a,b),b)\}\}$

According to the (congruence) axiom,

$f(a,b) \sim a, \ b \sim b$ 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.

Example: Given Σ_E-formula

$$F: \ f(f(f(a))) = a \ \wedge \ f(f(f(f(f(a))))) = a \ \wedge \ f(a) \neq a$$

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

$$\{f^3(a)\} \quad \text{and} \quad \{a\} .$$

From the union,

2. $\{\{a, f^3(a)\}, \{f(a)\}, \{f^2(a)\}, \{f^4(a)\}, \{f^5(a)\}\}$

deduce the following congruence propagations:

$$f^3(a) \sim a \ \Rightarrow \ f(f^3(a)) \sim f(a) \quad \text{i.e.} \quad f^4(a) \sim f(a)$$

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. $\{\{a, f^3(a)\}, \{f(a), f^4(a)\}, \{f^2(a), f^5(a)\}\}$.

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

$$\{f^2(a), f^5(a)\} \quad \text{and} \quad \{a, f^3(a)\}$$

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 Σ_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

$\{f(x)\}$ and $\{f(y)\}$.

The union $\{f(x), f(y)\}$ does not yield any new congruences, so the final partition is

2. $\{\{x\}, \; \{y\}, \; \{f(x), f(y)\}\}$.

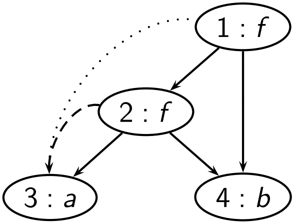
Does

3. $\{\{x\}, \; \{y\}, \; \{f(x), f(y)\}\} \models F$?

Yes, as $x \not\sim y$, agreeing with $x \neq y$. Hence, F is T_E-satisfiable.

Directed Acyclic Graph (DAG)

For Σ_E-formula F, graph-based data structure for representing the subterms of S_F (and congruence relation between them).



$f(f(a, b), b)$

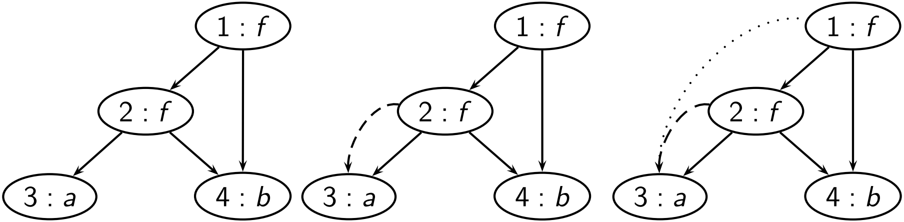
$f(a, b)$

$a \qquad b$

Efficient way for computing the congruence closure algorithm.

T_E-Satisfiability (Summary of idea)

$$f(a,b) = a \;\wedge\; f(f(a,b),b) \neq a$$



Initial DAG

$f(a,b) = a \;\Rightarrow$
MERGE $f(a,b)\ a$

$f(a,b) \sim a,\;\; b \sim b \;\Rightarrow$
$f(f(a,b),b) \sim f(a,b)$
MERGE $f(f(a,b),b)$
$f(a,b)$

- - - explicit equation ...... by congruence

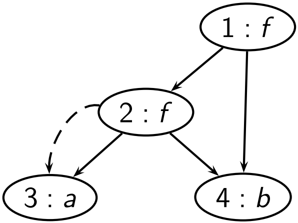
$$\left.\begin{array}{r} \text{FIND } f(f(a,b),b) = a = \text{FIND } a \\ f(f(a,b),b) \neq a \end{array}\right\} \Rightarrow \textbf{Unsatisfiable}$$

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 ccpar : id set
                    if the node is the representative for its
                    congruence class, then its ccpar
                    (congruence closure parents) are all
                    parents of nodes in its congruence class
}
```

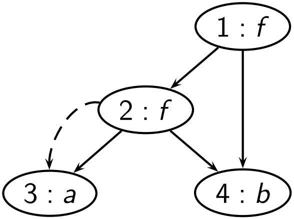
DAG Representation of node 2

type **node** = {			
`id`	:	**id**	... 2
`fn`	:	**string**	... f
`args`	:	**idlist**	... $[3, 4]$
`mutable find`	:	**id**	... 3
`mutable ccpar`	:	**idset**	... $\emptyset$
}			



DAG Representation of node 3

`type` **node** = {			
`id`	:	**id**	... 3
`fn`	:	**string**	... *a*
`args`	:	**idlist**	... []
`mutable find`	:	**id**	... 3
`mutable ccpar`	:	**idset**	... $\{1, 2\}$
}			

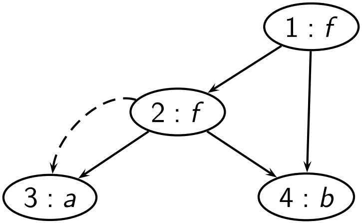


The Implementation

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

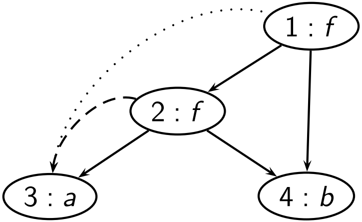
3 is the representative of 2.

UNION function

$$
\begin{array}{l}
\texttt{let}\ \text{UNION}\ i_1\ i_2 = \\
\quad \texttt{let}\ n_1 = \text{NODE}\ (\text{FIND}\ i_1)\ \texttt{in} \\
\quad \texttt{let}\ n_2 = \text{NODE}\ (\text{FIND}\ i_2)\ \texttt{in} \\
\quad n_1.\texttt{find} \leftarrow n_2.\texttt{find}; \\
\quad n_2.\texttt{ccpar} \leftarrow n_1.\texttt{ccpar} \cup n_2.\texttt{ccpar}; \\
\quad n_1.\texttt{ccpar} \leftarrow \emptyset
\end{array}
$$

n_2 is the representative of the union class

Example



UNION 1 2 $\quad n_1 = 1 \quad n_2 = 3$

$1.\texttt{find} \leftarrow 3$

$3.\texttt{ccpar} \leftarrow \{1, 2\}$

$1.\texttt{ccpar} \leftarrow \emptyset$

CCPAR function

Returns parents of all nodes in i's congruence class

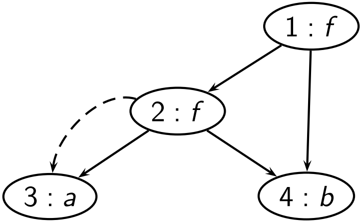
$$\texttt{let}\ \text{CCPAR}\ i = \\ (\text{NODE}\ (\text{FIND}\ i)).\texttt{ccpar}$$

CONGRUENT predicate

Test whether i_1 and i_2 are congruent

$$\begin{array}{l}
\texttt{let}\ \text{CONGRUENT}\ i_1\ i_2 = \\
\quad \texttt{let}\ n_1 \ = \ \text{NODE}\ i_1\ \texttt{in} \\
\quad \texttt{let}\ n_2 \ = \ \text{NODE}\ i_2\ \texttt{in} \\
\quad n_1.\texttt{fn} = n_2.\texttt{fn} \\
\qquad \wedge\ |n_1.\texttt{args}| = |n_2.\texttt{args}| \\
\qquad \wedge\ \forall i \in \{1, \ldots, |n_1.\texttt{args}|\}.\ \text{FIND}\ n_1.\texttt{args}[i] = \text{FIND}\ n_2.\texttt{args}[i]
\end{array}$$

Example:



Are 1 and 2 congruent?

`fn` fields	— both f
# of arguments	— same
left arguments $f(a, b)$ and a	— both congruent to 3
right arguments b and b	— both 4 (congruent)

Therefore 1 and 2 are congruent.

MERGE function

let rec MERGE i_1 i_2 =
  if FIND $i_1 \neq$ FIND i_2 then begin
    let P_{i_1} = CCPAR i_1 in
    let P_{i_2} = CCPAR i_2 in
    UNION i_1 i_2;
    foreach $t_1, t_2 \in P_{i_1} \times P_{i_2}$ do
      if FIND $t_1 \neq$ FIND t_2 $\wedge$ CONGRUENT t_1 t_2
      then MERGE t_1 t_2
    done
  end

P_{i_1} and P_{i_2} store the current values of CCPAR i_1 and CCPAR i_2.

Decision Procedure: T_E-satisfiability

Given Σ_E-formula

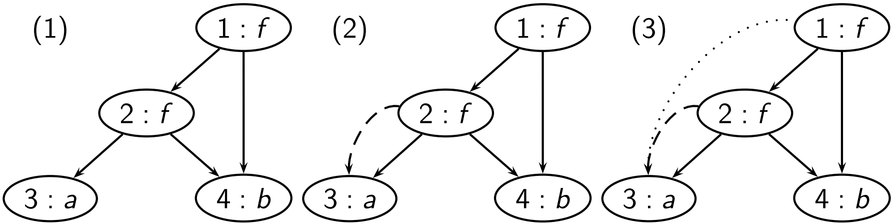
$$F: \; s_1 = t_1 \;\wedge\; \cdots \;\wedge\; s_m = t_m \;\wedge\; s_{m+1} \neq t_{m+1} \;\wedge\; \cdots \;\wedge\; s_n \neq t_n \;,$$

with subterm set S_F, perform the following steps:

1. Construct the initial DAG for the subterm set S_F.
2. For $i \in \{1, \ldots, m\}$, MERGE s_i t_i.
3. If FIND s_i = FIND t_i for some $i \in \{m+1, \ldots, n\}$, return unsatisfiable.
4. Otherwise (if FIND $s_i \neq$ FIND t_i for all $i \in \{m+1, \ldots, n\}$) return satisfiable.

Example 1: T_E-Satisfiability

$$f(a,b) = a \ \wedge \ f(f(a,b),b) \neq a$$



Initial DAG

MERGE 2 3

UNION 2 3

$P_2 = \{1\}$

$P_3 = \{2\}$

CONGRUENT 1 2

MERGE 1 2

UNION 1 2

$P_1 = \{\}$

$P_2 = \{1, 2\}$

FIND $f(f(a,b),b) = a =$ FIND $a \Rightarrow$ **Unsatisfiable**

Given Σ_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

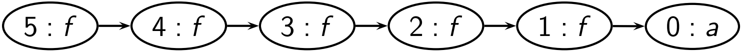
Does

$$(3) \; \{\{a, f(a,b), f(f(a,b),b)\},\; \{b\}\} \;\models\; F \; ?$$

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.

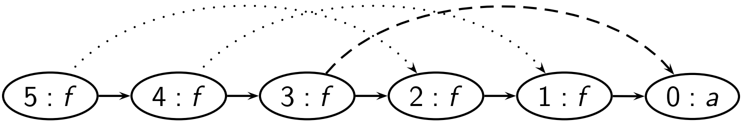
Example 2: T_E-Satisfiability

$$f(f(f(a))) = a \ \wedge \ f(f(f(f(f(a))))) = a \ \wedge \ f(a) \neq a$$



(1)

Initial DAG

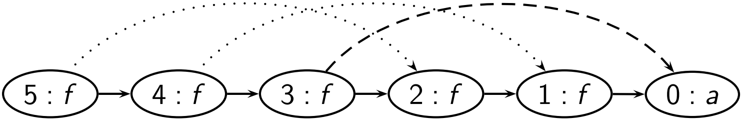


(2)

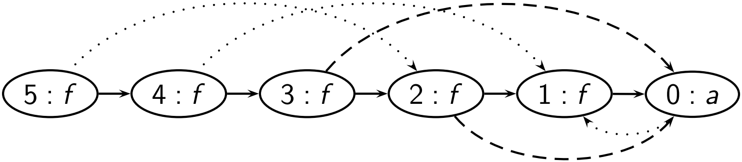
$$
\begin{aligned}
f(f(f(a))) = a \ \Rightarrow \ & \text{MERGE } 3\ 0 \qquad P_3 = \{4\} \quad P_0 = \{1\} \\
\Rightarrow \ & \text{MERGE } 4\ 1 \qquad P_4 = \{5\} \quad P_1 = \{2\} \\
\Rightarrow \ & \text{MERGE } 5\ 2 \qquad P_5 = \{\} \quad P_2 = \{3\}
\end{aligned}
$$

Example 2: T_E-Satisfiability

$$f(f(f(a))) = a \ \wedge \ f(f(f(f(f(a))))) = a \ \wedge \ f(a) \neq a$$



(2)



(3)

$f(f(f(f(f(a))))) = a \quad \Rightarrow$ MERGE 5 0 $\quad P_5 = \{3\} \quad P_0 = \{1, 4\}$

$\Rightarrow$ MERGE 3 1 $\quad$ STOP. Why?

FIND $f(a) = f(a) =$ FIND $a \ \Rightarrow$ **Unsatisfiable**

Given Σ_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 Σ_E-formula F is T_E-satisfiable iff the congruence closure algorithm returns satisfiable.

Recursive Data Structures

Quantifier-free Theory of Lists T_{cons}

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

- constructor cons : $\text{cons}(a, b)$ list constructed by prepending a to b
- left projector car : $\text{car}(\text{cons}(a, b)) = a$
- right projector cdr : $\text{cdr}(\text{cons}(a, b)) = b$
- atom : unary predicate

Axioms of T_{cons}

- reflexivity, symmetry, transitivity
- congruence axioms:

$$
\begin{array}{l}
\forall x_1, x_2, y_1, y_2.\ x_1 = x_2 \ \wedge\ y_1 = y_2 \ \rightarrow\ \text{cons}(x_1, y_1) = \text{cons}(x_2, y_2) \\
\forall x, y.\ x = y \ \rightarrow\ \text{car}(x) = \text{car}(y) \\
\forall x, y.\ x = y \ \rightarrow\ \text{cdr}(x) = \text{cdr}(y)
\end{array}
$$

- equivalence axiom:

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

-

$$
\begin{array}{ll}
(A1)\ \forall x, y.\ \text{car}(\text{cons}(x, y)) = x & \text{(left projection)} \\
(A2)\ \forall x, y.\ \text{cdr}(\text{cons}(x, y)) = y & \text{(right projection)} \\
(A3)\ \forall x.\ \neg\text{atom}(x) \rightarrow \text{cons}(\text{car}(x), \text{cdr}(x)) = x & \text{(construction)} \\
(A4)\ \forall x, y.\ \neg\text{atom}(\text{cons}(x, y)) & \text{(atom)}
\end{array}
$$

Simplifications

- Consider only quantifier-free conjunctive Σ_{cons}-formulae. Convert non-conjunctive formula to DNF and check each disjunct.
- $\neg\text{atom}(u_i)$ literals are removed:

$$\boxed{\text{replace} \quad \neg\text{atom}(u_i) \quad \text{with} \quad u_i = \text{cons}(u_i^1, u_i^2)}$$

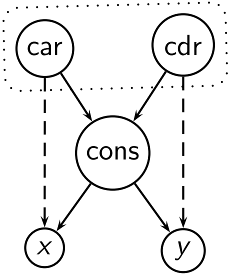
  by the (construnction) axiom.
- Because of similarity to Σ_{E}, we sometimes combine $\Sigma_{\text{cons}} \cup \Sigma_{\text{E}}$.

Algorithm: T_{cons}-Satisfiability (the idea)

$$
\begin{array}{rl}
F: & \underbrace{s_1 = t_1 \ \wedge\ \cdots\ \wedge\ s_m = t_m}_{\text{generate congruence closure}} \\
\wedge & \underbrace{s_{m+1} \neq t_{m+1} \ \wedge\ \cdots\ \wedge\ s_n \neq t_n}_{\text{search for contradiction}} \\
\wedge & \underbrace{\text{atom}(u_1) \ \wedge\ \cdots\ \wedge\ \text{atom}(u_\ell)}_{\text{search for contradiction}}
\end{array}
$$

where s_i, t_i, and u_i are T_{cons}-terms

Algorithm: T_{cons}-Satisfiability



1. Construct the initial DAG for S_F
2. for each node n with $n.\mathtt{fn} = \text{cons}$
   - add car(n) and MERGE car(n) $n.\mathtt{args}[1]$
   - add cdr(n) and MERGE cdr(n) $n.\mathtt{args}[2]$

   by axioms (A1), (A2)
3. for $1 \leq i \leq m$, MERGE s_i t_i
4. for $m + 1 \leq i \leq n$, if FIND s_i = FIND t_i, return **unsatisfiable**
5. for $1 \leq i \leq \ell$, if $\exists v$. FIND v = FIND u_i $\wedge$ $v.\mathtt{fn} = \text{cons}$, return **unsatisfiable**
6. Otherwise, return **satisfiable**

Example:

Given $(\Sigma_{\text{cons}} \cup \Sigma_{\text{E}})$-formula

$$F: \quad \begin{array}{l} \quad \text{car}(x) = \text{car}(y) \ \wedge \ \text{cdr}(x) = \text{cdr}(y) \\ \wedge \ \neg\text{atom}(x) \ \wedge \ \neg\text{atom}(y) \ \wedge \ f(x) \neq f(y) \end{array}$$

where the function symbol f is in Σ_{E}

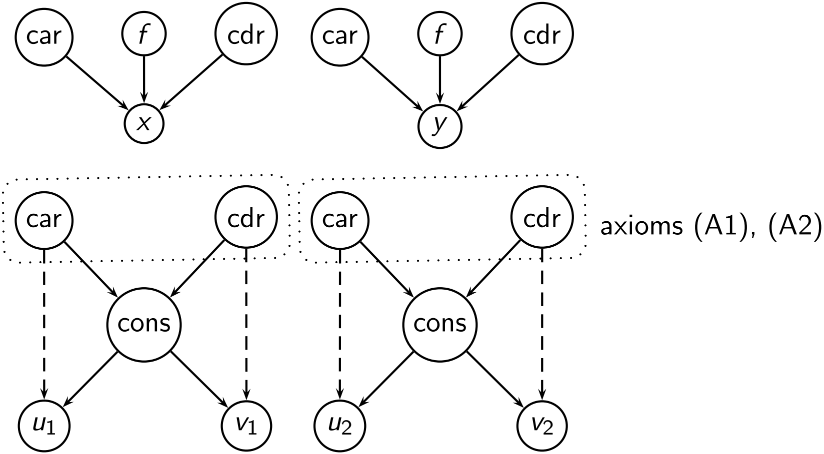
$$F': \quad \begin{array}{lll} \text{car}(x) = \text{car}(y) & \wedge & (1) \\ \text{cdr}(x) = \text{cdr}(y) & \wedge & (2) \\ x = \text{cons}(u_1, v_1) & \wedge & (3) \\ y = \text{cons}(u_2, v_2) & \wedge & (4) \\ f(x) \neq f(y) & & (5) \end{array}$$

Recall the projection axioms:

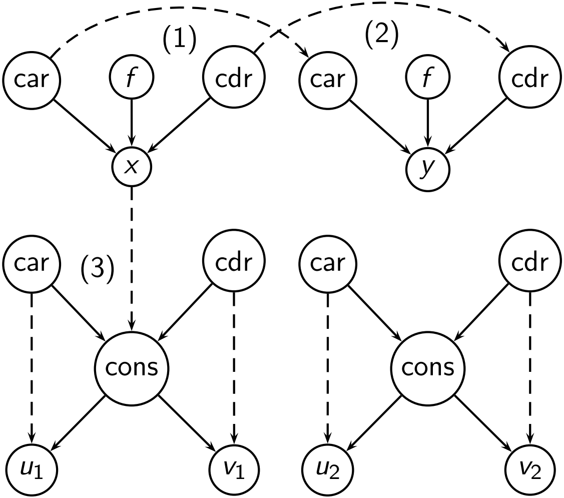
$$(A1) \quad \forall x, y.\ \text{car}(\text{cons}(x, y)) = x$$

$$(A2) \quad \forall x, y.\ \text{cdr}(\text{cons}(x, y)) = y$$

Example(cont): Initial DAG



Example(cont): MERGE



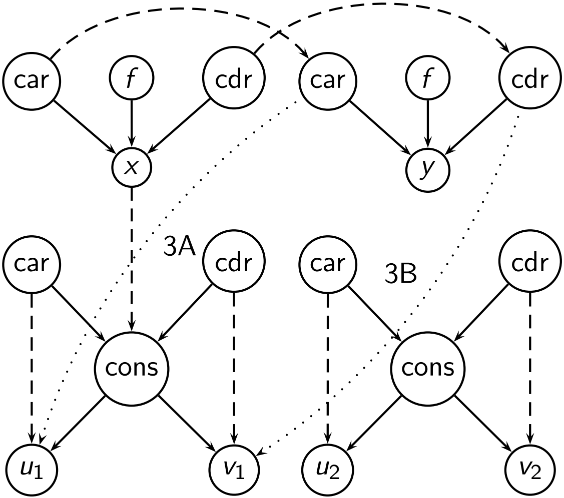
-- explicit equation

... by congruence

1 : MERGE car(x) car(y)
2 : MERGE cdr(x) cdr(y)

3 : MERGE x cons(u_1, v_1)
$\Downarrow$

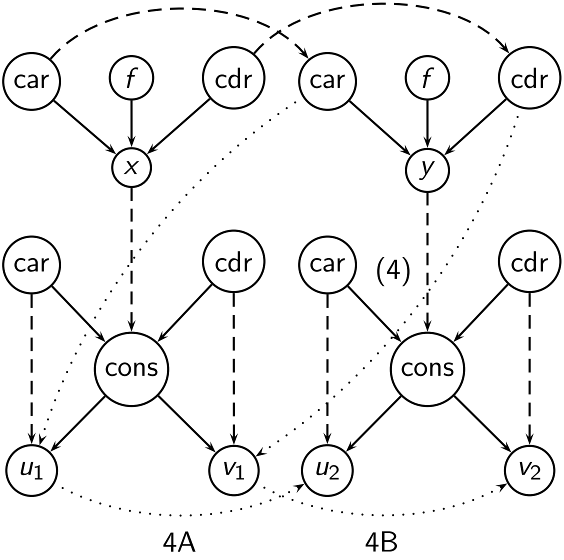
Example(cont): Propagation



Congruent:
car(x) car(cons(u_1, v_1))
FIND car(x) = car(y)
FIND car(cons(...)) = u_1

Congruent:
cdr(x) cdr(cons(u_1, v_1))
FIND cdr(x) = cdr(y)
FIND cdr(cons(...)) = v_1

Example(cont): MERGE



4 : MERGE y cons(u_2, v_2)

$\Downarrow$

Congruent:

car(y) car(cons(u_2, v_2))

FIND car(y) $= u_1$

FIND car(cons(...)) $= u_2$

Congruent:

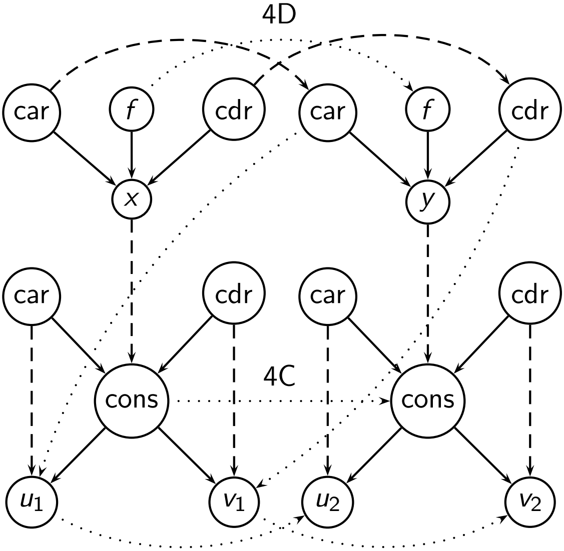
cdr(y) cdr(cons(u_2, v_2))

FIND cdr(y) $= v_1$

FIND cdr(cons(...)) $= v_2$

$\Downarrow$

Example(cont): CONGRUENCE



Congruent:
$cons(u_1, v_1)$ $cons(u_2, v_2)$

Congruent: $f(x)$ $f(y)$

5 : FIND $f(x) = f(y)$
FIND $f(y) = f(y)$

$\Downarrow$

F is **unsatisfiable**

Arrays

(1) Quantifier-free Fragment of T_A

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

where

- $a[i]$ is a binary function representing read of array a at index i;
- $a\langle i \triangleleft v\rangle$ 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_E
2. $\forall a, i, j.\ i = j \rightarrow a[i] = a[j]$ (array congruence)
3. $\forall a, v, i, j.\ i = j \rightarrow a\langle i \triangleleft v\rangle[j] = v$ (read-over-write 1)
4. $\forall a, v, i, j.\ i \neq j \rightarrow a\langle i \triangleleft v\rangle[j] = a[j]$ (read-over-write 2)

Note: a may itself be a write term, e.g., $a\langle i' \triangleleft v'\rangle$. Then

$$(a\langle i' \triangleleft v'\rangle)\langle i \triangleleft v\rangle$$

means: first write the value v' to index i' of a
then write the value v to index i of a

The Decision Procedure

Given quantifier-free conjunctive Σ_A-formula F.
To decide the T_A-satisfiability of F:

Step 1

If F does not contain any write terms $a\langle i \triangleleft v\rangle$, 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_E-satisfiability of the resulting formula.

Step 2

Select some read-over-write term $a\langle i \triangleleft v\rangle[j]$ (note that a may itself be a write term) and split on two cases:

1. According to (read-over-write 1), replace

$$F[a\langle i \triangleleft v\rangle[j]] \quad \text{with} \quad F_1 : \; F[v] \;\wedge\; i = j \;,$$

   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

$$F[a\langle i \triangleleft v\rangle[j]] \quad \text{with} \quad F_2 : \; F[a[j]] \;\wedge\; i \neq j \;,$$

   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 Σ_A-formula

$$F: \; i_1 = j \;\wedge\; i_1 \neq i_2 \;\wedge\; a[j] = v_1 \;\wedge\; a\langle i_1 \triangleleft v_1\rangle\langle i_2 \triangleleft v_2\rangle[j] \neq a[j] \; .$$

F contains a write term,

$$a\langle i_1 \triangleleft v_1\rangle\langle i_2 \triangleleft v_2\rangle[j] \neq a[j] \; .$$

According to (read-over-write 1), assume $\underline{i_2 = j}$ 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_E-unsatisfiable.

Returning, we try the second case:
according to (read-over-write 2), assume $\underline{i_2 \neq j}$ and recurse on

$$F_2 : \; i_2 \neq j \;\wedge\; i_1 = j \;\wedge\; i_1 \neq i_2 \;\wedge\; a[j] = v_1 \;\wedge\; a\langle i_1 \triangleleft v_1 \rangle[j] \neq a[j] \; .$$

F_2 contains a write term. According to (read-over-write 1), assume $\underline{i_1 = j}$ 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 $\underline{i_1 \neq j}$ 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?

The Calculus of Computation: Decision Procedures with Applications to Verification

by
Aaron Bradley
Zohar Manna

Springer 2007

10. Combining Decision Procedures

Given

Theories T_i over *signatures* Σ_i
(constants, functions, predicates)
with corresponding decision procedures P_i for T_i-satisfiability.

Goal

Decide satisfiability of a sentence in theory $\cup_i T_i$.

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 $(T_E \cup T_{\mathbb{Z}})$-unsatisfiable?

Combining Decision Procedures

Σ_1-theory T_1	Σ_2-theory T_2
$\boxed{P_1}$ for T_1-satisfiability	$\boxed{P_2}$ for T_2-satisfiability

?

$\boxed{P}$ for $(T_1 \cup T_2)$-satisfiability

Problem:
Decision procedures are domain specific.
How do we combine them?

Nelson-Oppen Combination Method (N-O Method)

$$\Sigma_1 \cap \Sigma_2 = \emptyset$$

Σ_1-theory T_1
stably infinite

$\boxed{P_1}$ for T_1-satisfiability
of quantifier-free Σ_1-formulae

Σ_2-theory T_2
stably infinite

$\boxed{P_2}$ for T_2-satisfiability
of quantifier-free Σ_2-formulae

$\boxed{P}$ for $(T_1 \cup T_2)$-satisfiability
of quantifier-free $(\Sigma_1 \cup \Sigma_2)$-formulae

Nelson-Oppen: Limitations

Given formula F in theory $T_1 \cup T_2$.

1. F must be quantifier-free.
2. Signatures Σ_i of the combined theory only share $=$, i.e.,

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

3. Theories must be stably infinite.

Note:

- Algorithm can be extended to combine arbitrary number of theories T_i — combine two, then combine with another, and so on.
- We restrict F to be conjunctive formula — otherwise convert to DNF and check each disjunct.

Stably Infinite Theories

A Σ-theory T is *stably infinite* iff
for every quantifier-free Σ-formula F:
if F is T-satisfiable
then there exists some T-interpretation that satisfies F.

Example: Σ-theory T

$$\Sigma : \{a, b, =\}$$

Axiom

$$\forall x.\ x = a \ \vee \ x = b$$

For every T-interpretation I, $|D_I| \leq 2$ (at most two elements).
Hence, T is *not* stably infinite.

All the other theories mentioned so far are stably infinite.

Example: Theory of partial orders

Σ-theory $T_{\preceq}$

$$\Sigma_{\preceq} : \{\preceq, =\}$$

where $\preceq$ is a binary predicate.

Axioms

1. $\forall x.\ x \preceq x$ ($\preceq$ reflexivity)
2. $\forall x, y.\ x \preceq y \ \land \ y \preceq x \ \rightarrow \ x = y$ ($\preceq$ antisymmetry)
3. $\forall x, y, z.\ x \preceq y \ \land \ y \preceq z \ \rightarrow \ x \preceq z$ ($\preceq$ 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 α_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 $a, b \in D_J$,

$$a \leq_J b \overset{\text{def}}{=} \begin{cases} a \leq_I b & \text{if } a, b \in D_I \\ a = b & \text{otherwise} \end{cases}$$

J is $T_{\preceq}$-interpretation satisfying F with infinite domain.
Hence, $T_{\preceq}$ is stably infinite.

Example: Consider quantifier-free conjunctive $(\Sigma_E \cup \Sigma_{\mathbb{Z}})$-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 $T_{\mathbb{Z}}$ only share $=$. Also, both theories are stably infinite. Hence, the NO combination of the decision procedures for T_E and $T_{\mathbb{Z}}$ decides the $(T_E \cup T_{\mathbb{Z}})$-satisfiability of F.

Intuitively, F is $(T_E \cup T_{\mathbb{Z}})$-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 $(T_E \cup T_{\mathbb{Z}})$-unsatisfiable.

N-O Overview

Phase 1: Variable Abstraction

- Given conjunction Γ in theory $T_1 \cup T_2$.
- Convert to conjunction $\Gamma_1 \cup \Gamma_2$ s.t.
  - Γ_i in theory T_i
  - $\Gamma_1 \cup \Gamma_2$ satisfiable iff Γ satisfiable.

Phase 2: Check

- If there is some set S of equalities and disequalities between the shared variables of Γ_1 and Γ_2
  $\text{shared}(\Gamma_1, \Gamma_2) = \text{free}(\Gamma_1) \cap \text{free}(\Gamma_2)$
  s.t. $S \cup \Gamma_i$ are T_i-satisfiable for all i,
  then Γ 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

$$\Sigma_1\text{-formula } F_1 \qquad \text{and} \qquad \Sigma_2\text{-formula } F_2$$

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 $\mathrm{hd}(t) \in \Sigma_j$,

$$F[f(t_1, \ldots, t, \ldots, t_n)] \quad \Rightarrow \quad F[f(t_1, \ldots, w, \ldots, t_n)] \wedge w = t$$

(2) if predicate $p \in \Sigma_i$ and $\mathrm{hd}(t) \in \Sigma_j$,

$$F[p(t_1, \ldots, t, \ldots, t_n)] \quad \Rightarrow \quad F[p(t_1, \ldots, w, \ldots, t_n)] \wedge w = t$$

(3) if $\mathrm{hd}(s) \in \Sigma_i$ and $\mathrm{hd}(t) \in \Sigma_j$,

$$F[s = t] \quad \Rightarrow \quad F[\top] \ \wedge \ w = s \ \wedge \ w = t$$

(4) if $\mathrm{hd}(s) \in \Sigma_i$ and $\mathrm{hd}(t) \in \Sigma_j$,

$$F[s \neq t] \quad \Rightarrow \quad F[w_1 \neq w_2] \ \wedge \ w_1 = s \ \wedge \ w_2 = t$$

where w, w_1, and w_2 are fresh variables.

Example: Consider $(\Sigma_E \cup \Sigma_{\mathbb{Z}})$-formula

$$F: \ 1 \leq x \ \land \ x \leq 2 \ \land \ f(x) \neq f(1) \ \land \ f(x) \neq f(2) \ .$$

According to transformation 1, since $f \in \Sigma_E$ and $1 \in \Sigma_{\mathbb{Z}}$, 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

$$\Gamma_{\mathbb{Z}}: \ \{1 \leq x, \ x \leq 2, \ w_1 = 1, \ w_2 = 2\}$$

are $T_{\mathbb{Z}}$-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 $\Sigma_{\mathbb{Z}}$-formula

$$F_1: \ 1 \leq x \ \land \ x \leq 2 \ \land \ w_1 = 1 \ \land \ w_2 = 2$$

and the Σ_E-formula

$$F_2: \ f(x) \neq f(w_1) \ \land \ f(x) \neq f(w_2) \ .$$

F_1 and F_2 share the variables $\{x, w_1, w_2\}$.

$F_1 \ \land \ F_2$ is $(T_E \cup T_{\mathbb{Z}})$-equisatisfiable to F.

Example: Consider $(\Sigma_E \cup \Sigma_{\mathbb{Z}})$-formula

$$F:\ f(x) = x+y \land x \leq y+z \land x+z \leq y \land y = 1 \land f(x) \neq f(2)\ .$$

In the first literal, $\text{hd}(f(x)) = f \in \Sigma_{\text{E}}$ and $\text{hd}(x+y) = + \in \Sigma_{\mathbb{Z}}$; thus, by (3), replace the literal with

$$w_1 = f(x)\ \land\ w_1 = x+y\ .$$

In the final literal, $f \in \Sigma_E$ but $2 \in \Sigma_{\mathbb{Z}}$, so by (1), replace it with

$$f(x) \neq f(w_2)\ \land\ w_2 = 2\ .$$

Now, separating the literals results in two formulae:

$$F_1:\ w_1 = x+y\ \land\ x \leq y+z\ \land\ x+z \leq y\ \land\ y = 1\ \land\ w_2 = 2$$

is a $\Sigma_{\mathbb{Z}}$-formula, and

$$F_2:\ w_1 = f(x)\ \land\ f(x) \neq f(w_2)$$

is a Σ_E-formula.

The conjunction $F_1\ \land\ F_2$ is $(T_E \cup T_{\mathbb{Z}})$-equisatisfiable to F.

Nondeterministic Version

Phase 2: Guess and Check

- Phase 1 separated $(\Sigma_1 \cup \Sigma_2)$-formula F into two formulae:
  Σ_1-formula F_1 and Σ_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.

Example: Consider $(\Sigma_E \cup \Sigma_{\mathbb{Z}})$-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 $\Sigma_{\mathbb{Z}}$-formula

$$F_1: \; 1 \leq x \;\wedge\; x \leq 2 \;\wedge\; w_1 = 1 \;\wedge\; w_2 = 2$$

and the Σ_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:

1. $\{\{x, w_1, w_2\}\}$, *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.
2. $\{\{x, w_1\}, \{w_2\}\}$, *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.
3. $\{\{x, w_2\}, \{w_1\}\}$, *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.
4. $\{\{x\}, \{w_1, w_2\}\}$, *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 $T_{\mathbb{Z}}$-unsatisfiable.
5. $\{\{x\}, \{w_1\}, \{w_2\}\}$, *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 $T_{\mathbb{Z}}$)
   $\Rightarrow\; F_1 \;\wedge\; \alpha(V, E)$ is $T_{\mathbb{Z}}$-unsatisfiable.

Hence, F is $(T_E \cup T_{\mathbb{Z}})$-unsatisfiable.

Example: Consider the $(\Sigma_{\text{cons}} \cup \Sigma_{\mathbb{Z}})$-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 Σ_{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 $\Sigma_{\mathbb{Z}}$-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_{cons}-satisfiable, and $F_2 \ \wedge \ \alpha(V, E)$ is $T_{\mathbb{Z}}$-satisfiable.
Hence, F is $(T_{\text{cons}} \cup T_{\mathbb{Z}})$-satisfiable.

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

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 arithmethic $T_{\mathbb{R}}$

$\boxed{P_{\mathbb{R}}}$

Theory of equality T_E

$\boxed{P_E}$

$$F: \quad f(f(x)-f(y)) \neq f(z) \ \wedge \ x \leq y \ \wedge \ y + z \leq x \ \wedge \ 0 \leq z$$

$(T_{\mathbb{R}} \cup T_E)$-unsatisfiable

Intuitively,

last 3 conjuncts $\Rightarrow \ x = y \ \wedge \ z = 0$

contradicts 1st conjunct

Phase 1: Variable Abstraction

$$F: \; f(f(x) - f(y)) \neq f(z) \;\wedge\; x \leq y \;\wedge\; y + z \leq x \;\wedge\; 0 \leq z$$

$$f(x) \;\Rightarrow\; u \qquad f(y) \;\Rightarrow\; v \qquad u - v \;\Rightarrow\; w$$

$$\Gamma_E: \quad \{f(w) \neq f(z),\; u = f(x),\; v = f(y)\} \qquad \ldots T_E\text{-formula}$$

$$\Gamma_{\mathbb{R}}: \quad \{x \leq y,\; y + z \leq x,\; 0 \leq z,\; w = u - v\} \quad \ldots T_{\mathbb{R}}\text{-formula}$$

$$\text{shared}(\Gamma_{\mathbb{R}}, \Gamma_E) = \{x, y, z, u, v, w\}$$

Nondeterministic version — over 200 Es!
Let's try the deterministic version.

Phase 2: Equality Propagation

$\boxed{P_{\mathbb{R}}}$ $\qquad\qquad$ $\boxed{P_E}$

$$s_0 : \langle \Gamma_{\mathbb{R}}, \Gamma_E, \{\} \rangle$$

$\Gamma_{\mathbb{R}} \models x = y$

$$s_1 : \langle \Gamma_{\mathbb{R}}, \Gamma_E, \{x = y\} \rangle$$

$\Gamma_E \cup \{x = y\} \models u = v$

$$s_2 : \langle \Gamma_{\mathbb{R}}, \Gamma_E, \{x = y, u = v\} \rangle$$

$\Gamma_{\mathbb{R}} \cup \{u = v\} \models z = w$

$$s_3 : \langle \Gamma_{\mathbb{R}}, \Gamma_E, \{x = y, u = v, z = w\} \rangle$$

$\Gamma_E \cup \{z = w\} \models \text{false}$

$$s_4 : \text{false}$$

Contradiction. Thus, F is $(T_{\mathbb{R}} \cup T_E)$-unsatisfiable.
If there were no contradiction, F would be $(T_{\mathbb{R}} \cup T_E)$-satisfiable.

Claim:
Equality propagation is a decision procedure for convex theories.

Def. A Σ-theory T is *convex* iff
for every quantifier-free conjunction Σ-formula F
and for every disjunction $\bigvee_{i=1}^{n} (u_i = v_i)$

if $F \models \bigvee_{i=1}^{n} (u_i = v_i)$
then $F \models u_i = v_i$, for some $i \in \{1, \ldots, n\}$

Convex Theories

- T_E, $T_{\mathbb{R}}$, $T_{\mathbb{Q}}$, T_{cons} are convex
- $T_{\mathbb{Z}}$, T_{A} are not convex

Example: $T_{\mathbb{Z}}$ is not convex

Consider quantifier-free conjunctive

$$F: \quad 1 \leq z \ \land \ z \leq 2 \ \land \ u = 1 \ \land \ v = 2$$

Then

$$F \ \models \ z = u \lor z = v$$

but

$$F \ \not\models \ z = u$$

$$F \ \not\models \ z = v$$

Example:

The theory of arrays T_{A} is not convex.
Consider the quantifier-free conjunctive Σ_{A}-formula

$$F : \; a\langle i \triangleleft v\rangle[j] = v \; .$$

Then

$$F \;\Rightarrow\; i = j \;\vee\; a[j] = v \; ,$$

but

$$F \not\Rightarrow i = j$$
$$F \not\Rightarrow a[j] = v \; .$$

What if T is Not Convex?

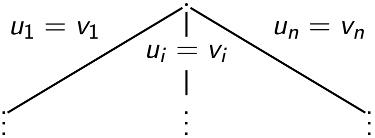
Case split when:

$$\Gamma \models \bigvee_{i=1}^{n} (u_i = v_i)$$

but

$$\Gamma \not\models u_i = v_i \qquad\qquad \text{for all } i = 1, \ldots, n$$

- For each $i = 1, \ldots, n$, 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

$T_{\mathbb{Z}}$ not convex! $\boxed{P_{\mathbb{Z}}}$

T_E convex $\boxed{P_E}$

$$\Gamma : \left\{ \begin{array}{ll} 1 \leq x, & x \leq 2, \\ f(x) \neq f(1), & f(x) \neq f(2) \end{array} \right\} \quad \text{in } T_{\mathbb{Z}} \cup T_E$$

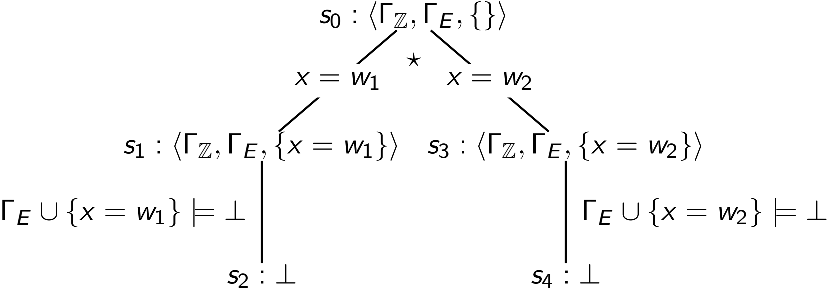
- Replace $f(1)$ by $f(w_1)$, and add $w_1 = 1$.
- Replace $f(2)$ by $f(w_2)$, and add $w_2 = 2$.

Result:

$$\Gamma_{\mathbb{Z}} = \left\{ \begin{array}{l} 1 \leq x, \\ x \leq 2, \\ w_1 = 1, \\ w_2 = 2 \end{array} \right\} \quad \text{and} \quad \Gamma_E = \left\{ \begin{array}{l} f(x) \neq f(w_1), \\ f(x) \neq f(w_2) \end{array} \right\}$$

$\text{shared}(\Gamma_{\mathbb{Z}}, \Gamma_E) = \{x, w_1, w_2\}$

Example 2: Non-Convex Theory



$\star : \;\; \Gamma_{\mathbb{Z}} \models x = w_1 \;\vee\; x = w_2$

All leaves are labeled with $\bot \Rightarrow \Gamma$ is $(T_{\mathbb{Z}} \cup T_E)$-unsatisfiable.

Example 3: Non-Convex Theory

$$\Gamma : \left\{ \begin{array}{c} 1 \leq x, \quad x \leq 3, \\ f(x) \neq f(1),\ f(x) \neq f(3),\ f(1) \neq f(2) \end{array} \right\} \quad \text{in } T_{\mathbb{Z}} \cup T_E$$

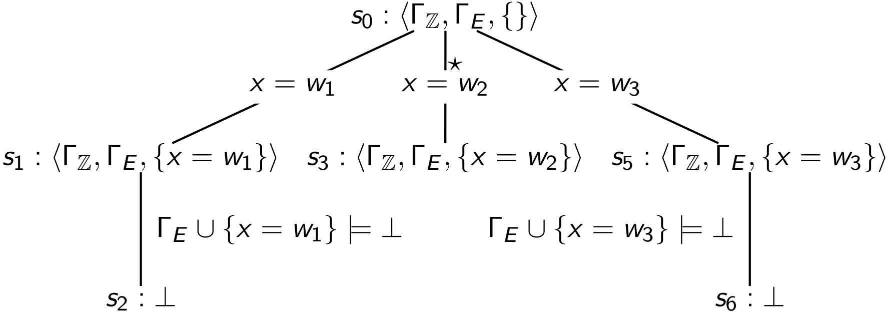
- Replace $f(1)$ by $f(w_1)$, and add $w_1 = 1$.
- Replace $f(2)$ by $f(w_2)$, and add $w_2 = 2$.
- Replace $f(3)$ by $f(w_3)$, and add $w_3 = 3$.

Result:

$$\Gamma_{\mathbb{Z}} = \left\{ \begin{array}{l} 1 \leq x, \\ x \leq 3, \\ w_1 = 1, \\ w_2 = 2, \\ w_3 = 3 \end{array} \right\} \quad \text{and} \quad \Gamma_E = \left\{ \begin{array}{l} f(x) \neq f(w_1), \\ f(x) \neq f(w_3), \\ f(w_1) \neq f(w_2) \end{array} \right\}$$

$\text{shared}(\Gamma_{\mathbb{Z}}, \Gamma_E) = \{x, w_1, w_2, w_3\}$

Example 3: Non-Convex Theory



$\star : \; \Gamma_{\mathbb{Z}} \models x = w_1 \;\vee\; x = w_2 \;\vee\; x = w_3$

No more equations on middle leaf $\Rightarrow$ Γ is $(T_{\mathbb{Z}} \cup T_E)$-satisfiable.

THE CALCULUS OF COMPUTATION:
Decision Procedures with
Applications to Verification

by
Aaron Bradley
Zohar Manna

Springer 2007

11. Arrays

(2) Array Property Fragment of T_A

Decidable fragment of T_A that includes $\forall$ quantifiers

Array property

Σ_A-formula of form

$$\forall \bar{i}.\ F[\bar{i}] \ \rightarrow \ G[\bar{i}]\ ,$$

where $\bar{i}$ is a list of variables.

- index guard $F[\bar{i}]$:

$$\begin{array}{rcl} \text{iguard} & \rightarrow & \text{iguard} \ \wedge \ \text{iguard} \mid \text{iguard} \ \vee \ \text{iguard} \mid \text{atom} \\ \text{atom} & \rightarrow & \text{var} = \text{var} \mid evar \neq \text{var} \mid \text{var} \neq evar \mid \top \\ \text{var} & \rightarrow & evar \mid uvar \end{array}$$

  where *uvar* is any universally quantified index variable, and *evar* is any constant or unquantified variable.

- value constraint $G[\bar{i}]$: a universally quantified index can occur in a value constraint $G[\bar{i}]$ 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: Σ_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

$$F : \ \cdots \ \wedge \ a = b \ \wedge \ \cdots$$

with array terms a and b, rewrite F as

$$F' : \ \cdots \ \wedge \ (\forall i.\ \top \ \rightarrow \ a[i] = b[i]) \ \wedge \ \cdots \ .$$

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

$$F : \ a\langle i \triangleleft v\rangle = a \ \land \ a[i] \neq v \ ,$$

which expands to

$$F' : \ \forall j.\ a\langle i \triangleleft v\rangle[j] = a[j] \ \land \ a[i] \neq v \ .$$

Intuitively, to determine that F' is T_{A}-unsatisfiable requires merely examining index i:

$$F'' : \ \left(\bigwedge_{j \in \{i\}} a\langle i \triangleleft v\rangle[j] = a[j] \right) \ \land \ a[i] \neq v \ ,$$

or simply

$$a\langle i \triangleleft v\rangle[i] = a[i] \ \land \ a[i] \neq v \ .$$

Simplifying,

$$v = a[i] \ \land \ a[i] \neq v \ ,$$

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:

$$\frac{F[a\langle i \triangleleft v\rangle]}{F[a'] \;\wedge\; a'[i] = v \;\wedge\; (\forall j.\ j \neq i \;\rightarrow\; a[j] = a'[j])} \text{ for fresh } a' \quad \text{(write)}$$

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:

$$\frac{F[\exists \bar{i}.\ G[\bar{i}]]}{F[G[\bar{j}]]} \text{ for fresh } \bar{j} \quad \text{(exists)}$$

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** $\mathcal{I}$:

$$\begin{aligned} \mathcal{I} = \; & \{\lambda\} \\ & \cup \; \{t \; : \; \cdot[t] \in F_3 \text{ such that } t \text{ is not a universally quantified variable}\} \\ & \cup \; \{t \; : \; t \text{ occurs as an } \textit{evar} \text{ in the parsing of index guards}\} \end{aligned}$$

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.
- λ 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:

$$\frac{H[\forall \bar{i}.\ F[\bar{i}] \ \rightarrow \ G[\bar{i}]]}{H\left[\bigwedge_{\bar{i} \in \mathcal{I}^n} \left(F[\bar{i}] \ \rightarrow \ G[\bar{i}]\right)\right]} \quad \text{(forall)}$$

where n is the size of the list of quantified variables $\bar{i}$.

Step 6

From the output F_5 of Step 5, construct

$$F_6 : \ F_5 \ \wedge \bigwedge_{i \ \in \ \mathcal{I} \setminus \{\lambda\}} \lambda \neq i \ .$$

The new conjuncts assert that the variable λ 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.

Example: Consider array property formula

$$F : \; a\langle \ell \triangleleft v\rangle[k] = b[k] \land b[k] \neq v \land a[k] = v \land \underbrace{(\forall i.\; i \neq \ell \;\rightarrow\; a[i] = b[i])}_{\text{array property}}$$

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

$$F_2 : \begin{array}{l} a'[k] = b[k] \;\land\; b[k] \neq v \;\land\; a[k] = v \;\land\; (\forall i.\; i \neq \ell \;\rightarrow\; a[i] = b[i]) \\ \qquad\qquad\quad \land\; a'[\ell] = v \;\land\; (\forall j.\; j \neq \ell \;\rightarrow\; a[j] = a'[j]) \end{array}$$

F_2 does not contain any existential quantifiers. Its index set is

$$\begin{array}{rcl} \mathcal{I} & = & \{\lambda\} \cup \{k\} \cup \{\ell\} \\ & = & \{\lambda, k, \ell\} \; . \end{array}$$

Thus, by Step 5, replace universal quantification:

$$F_5 : \begin{array}{l} a'[k] = b[k] \;\land\; b[k] \neq v \;\land\; a[k] = v \;\land\; \bigwedge_{i \,\in\, \mathcal{I}} (i \neq \ell \rightarrow a[i] = b[i]) \\ \;\land\; a'[\ell] = v \;\land\; \bigwedge_{j \,\in\, \mathcal{I}} \big(j \neq \ell \;\rightarrow\; a[j] = a'[j]\big) \end{array}$$

$$F_5 : \begin{array}{l} a'[k] = b[k] \ \wedge \ b[k] \neq v \ \wedge \ a[k] = v \ \wedge \ \bigwedge_{i \in \mathcal{I}} (i \neq \ell \rightarrow a[i] = b[i]) \\ \wedge \ a'[\ell] = v \ \wedge \ \bigwedge_{j \in \mathcal{I}} \big(j \neq \ell \ \rightarrow \ a[j] = a'[j]\big) \end{array}$$

Expanding produces

$$F_5' : \begin{array}{l} a'[k] = b[k] \ \wedge \ b[k] \neq v \ \wedge \ a[k] = v \ \wedge \ (\lambda \neq \ell \ \rightarrow \ a[\lambda] = b[\lambda]) \\ \wedge \ (k \neq \ell \ \rightarrow \ a[k] = b[k]) \ \wedge \ (\ell \neq \ell \ \rightarrow \ a[\ell] = b[\ell]) \\ \wedge \ a'[\ell] = v \ \wedge \ (\lambda \neq \ell \ \rightarrow \ a[\lambda] = a'[\lambda]) \\ \wedge \ (k \neq \ell \ \rightarrow \ a[k] = a'[k]) \ \wedge \ (\ell \neq \ell \ \rightarrow \ a[\ell] = a'[\ell]) \end{array}$$

Simplifying produces

$$F_5'' : \begin{array}{l} a'[k] = b[k] \ \wedge \ b[k] \neq v \ \wedge \ a[k] = v \ \wedge \ (\lambda \neq \ell \ \rightarrow \ a[\lambda] = b[\lambda]) \\ \wedge \ (k \neq \ell \ \rightarrow \ a[k] = b[k]) \\ \wedge \ a'[\ell] = v \ \wedge \ (\lambda \neq \ell \ \rightarrow \ a[\lambda] = a'[\lambda]) \\ \wedge \ (k \neq \ell \ \rightarrow \ a[k] = a'[k]) \end{array}$$

Step 6 distinguishes λ from other members of $\mathcal{I}$:

$$F_6: \quad \begin{array}{l} a'[k] = b[k] \ \wedge \ b[k] \neq v \ \wedge \ a[k] = v \ \wedge \ (\lambda \neq \ell \ \rightarrow \ a[\lambda] = b[\lambda]) \\ \ \wedge \ (k \neq \ell \ \rightarrow \ a[k] = b[k]) \\ \ \wedge \ a'[\ell] = v \ \wedge \ (\lambda \neq \ell \ \rightarrow \ a[\lambda] = a'[\lambda]) \\ \ \wedge \ (k \neq \ell \ \rightarrow \ a[k] = a'[k]) \\ \ \wedge \ \lambda \neq k \ \wedge \ \lambda \neq \ell \end{array}$$

Simplifying,

$$F_6': \quad \begin{array}{l} a'[k] = b[k] \ \wedge \ b[k] \neq v \ \wedge \ a[k] = v \\ \ \wedge \ a[\lambda] = b[\lambda] \ \wedge \ (k \neq \ell \ \rightarrow \ a[k] = b[k]) \\ \ \wedge \ a'[\ell] = v \ \wedge \ a[\lambda] = a'[\lambda] \ \wedge \ (k \neq \ell \ \rightarrow \ a[k] = a'[k]) \\ \ \wedge \ \lambda \neq k \ \wedge \ \lambda \neq \ell \end{array}$$

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 is T_A-unsatisfiable.

(3) Theory of Integer-Indexed Arrays $T_{\mathsf{A}}^{\mathbb{Z}}$

$\leq$ enables reasoning about subarrays and properties such as subarray is sorted or partitioned.

signature of $T_{\mathsf{A}}^{\mathbb{Z}}$: $\Sigma_{\mathsf{A}}^{\mathbb{Z}} = \Sigma_{\mathsf{A}} \cup \Sigma_{\mathbb{Z}}$

axioms of $T_{\mathsf{A}}^{\mathbb{Z}}$: both axioms of T_{A} and $T_{\mathbb{Z}}$

Array property: $\Sigma_A^{\mathbb{Z}}$-formula of the form

$$\forall \bar{i}.\ F[\bar{i}] \ \rightarrow\ G[\bar{i}]\ ,$$

where $\bar{i}$ is a list of integer variables.

- $F[\bar{i}]$ index guard:

$$\begin{array}{rcl} \text{iguard} & \rightarrow & \text{iguard} \wedge \text{iguard} \mid \text{iguard} \vee \text{iguard} \mid \text{atom} \\ \text{atom} & \rightarrow & \text{expr} \leq \text{expr} \mid \text{expr} = \text{expr} \\ \text{expr} & \rightarrow & uvar \mid \text{pexpr} \\ \text{pexpr} & \rightarrow & \text{pexpr}' \\ \text{pexpr}' & \rightarrow & \mathbb{Z} \mid \mathbb{Z} \cdot evar \mid \text{pexpr}' + \text{pexpr}' \end{array}$$

  where *uvar* is any universally quantified integer variable, and *evar* is any existentially quantified or free integer variable.
- $G[\bar{i}]$ 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 $T_A^{\mathbb{Z}}$ consists of formulae that are Boolean combinations of quantifier-free $\Sigma_A^{\mathbb{Z}}$-formulae and array properties.

A Decision Procedure

The idea again is to reduce universal quantification to finite conjunction.
Given F from the array property fragment of $T_{\mathsf{A}}^{\mathbb{Z}}$, decide its $T_{\mathsf{A}}^{\mathbb{Z}}$-satisfiability as follows:

Step 1
Put F in NNF.

Step 2
Apply the following rule exhaustively to remove writes:

$$\frac{F[a\langle i \triangleleft e\rangle]}{F[a'] \ \wedge \ a'[i] = e \ \wedge \ (\forall j.\ j \neq i \ \rightarrow \ a[j] = a'[j])} \text{ for fresh } a' \quad \text{(write)}$$

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 3
Apply the following rule exhaustively to remove existential quantification:

$$\frac{F[\exists \bar{i}.\ G[\bar{i}]]}{F[G[\bar{j}]]} \text{ for fresh } \bar{j} \qquad \text{(exists)}$$

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 $\mathcal{I}$:

$$\mathcal{I} = \begin{array}{l} \{t : \cdot[t] \in F_3 \text{ such that } t \text{ is not a universally quantified variable}\} \\ \cup \ \{t : t \text{ occurs as a pexpr in the parsing of index guards}\} \end{array}$$

If $\mathcal{I} = \emptyset$, then let $\mathcal{I} = \{0\}$. The index set contains all relevant symbolic indices that occur in F_3.

Step 5

Apply the following rule exhaustively to remove universal quantification:

$$\frac{H[\forall \bar{i}.\ F[\bar{i}] \ \rightarrow \ G[\bar{i}]]}{H\left[\bigwedge_{\bar{i} \in \mathcal{I}^n} \left(F[\bar{i}] \ \rightarrow \ G[\bar{i}]\right)\right]} \quad \text{(forall)}$$

n is the size of the block of universal quantifiers over $\bar{i}$.

Step 6

F_5 is quantifier-free in the combination theory $T_{\mathsf{A}} \cup T_{\mathbb{Z}}$. Decide the $(T_{\mathsf{A}} \cup T_{\mathbb{Z}})$-satisfiability of the resulting formula.

Example: $\Sigma_A^{\mathbb{Z}}$-formula:

$$F: \quad \begin{array}{l} (\forall i.\ \ell \le i \le u \ \rightarrow\ a[i] = b[i]) \\ \ \wedge\ \neg(\forall i.\ \ell \le i \le u+1 \ \rightarrow\ a\langle u+1 \triangleleft b[u+1]\rangle[i] = b[i]) \end{array}$$

In NNF, we have

$$F_1: \quad \begin{array}{l} (\forall i.\ \ell \le i \le u \ \rightarrow\ a[i] = b[i]) \\ \ \wedge\ (\exists i.\ \ell \le i \le u+1 \ \wedge\ a\langle u+1 \triangleleft b[u+1]\rangle[i] \ne b[i]) \end{array}$$

Step 2 produces

$$F_2: \quad \begin{array}{l} (\forall i.\ \ell \le i \le u \ \rightarrow\ a[i] = b[i]) \\ \ \wedge\ (\exists i.\ \ell \le i \le u+1 \ \wedge\ a'[i] \ne b[i]) \\ \ \wedge\ a'[u+1] = b[u+1] \\ \ \wedge\ (\forall j.\ j \le u+1-1 \ \vee\ u+1+1 \le j \ \rightarrow\ a[j] = a'[j]) \end{array}$$

Step 3 removes the existential quantifier by introducing a fresh constant k:

$$F_3 : \quad \begin{array}{l} (\forall i.\ \ell \le i \le u \ \rightarrow\ a[i] = b[i]) \\ \ \wedge\ \ell \le k \le u+1 \ \wedge\ a'[k] \ne b[k] \\ \ \wedge\ a'[u+1] = b[u+1] \\ \ \wedge\ (\forall j.\ j \le u+1-1 \ \vee\ u+1+1 \le j \ \rightarrow\ a[j] = a'[j]) \end{array}$$

Simplifying,

$$F_3' : \quad \begin{array}{l} (\forall i.\ \ell \le i \le u \ \rightarrow\ a[i] = b[i]) \\ \ \wedge\ \ell \le k \le u+1 \ \wedge\ a'[k] \ne b[k] \\ \ \wedge\ a'[u+1] = b[u+1] \\ \ \wedge\ (\forall j.\ j \le u \ \vee\ u+2 \le j \ \rightarrow\ a[j] = a'[j]) \end{array}$$

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 ℓ, u, and $u+2$ that occur as pexprs in the index guards.

Step 5 rewrites universal quantification to finite conjunction over this set:

$$F_5 : \quad \begin{array}{l} \bigwedge\limits_{i \,\in\, \mathcal{I}} (\ell \le i \le u \;\rightarrow\; a[i] = b[i]) \\ \wedge\;\; \ell \le k \le u+1 \;\wedge\; a'[k] \ne b[k] \\ \wedge\;\; a'[u+1] = b[u+1] \\ \wedge\;\; \bigwedge\limits_{j \,\in\, \mathcal{I}} (j \le u \;\vee\; u+2 \le j \;\rightarrow\; a[j] = a'[j]) \end{array}$$

Expanding the conjunctions according to the index set $\mathcal{I}$ and simplifying according to trivially true or false antecedents (*e.g.*, $\ell \le u+1 \le u$ simplifies to $\perp$, while $u \le u \;\vee\; u+2 \le u$ simplifies to $\top$) produces:

$$F'_5: \begin{array}{ll} (\ell \le k \le u \;\rightarrow\; a[k] = b[k]) & (1) \\ \wedge\; (\ell \le u \;\rightarrow\; a[\ell] = b[\ell] \;\wedge\; a[u] = b[u]) & (2) \\ \wedge\; \ell \le k \le u + 1 & (3) \\ \wedge\; a'[k] \ne b[k] & (4) \\ \wedge\; a'[u+1] = b[u+1] & (5) \\ \wedge\; (k \le u \;\vee\; u + 2 \le k \;\rightarrow\; a[k] = a'[k]) & (6) \\ \wedge\; (\ell \le u \;\vee\; u + 2 \le \ell \;\rightarrow\; a[\ell] = a'[\ell]) & (7) \\ \wedge\; a[u] = a'[u] \;\wedge\; a[u+2] = a'[u+2] & (8) \end{array}$$

$(T_A \cup T_{\mathbb{Z}})$-unsatisfiability of this quantifier-free $(\Sigma_A \cup \Sigma_{\mathbb{Z}})$-formula can be decided using the techniques of Combination of Theories.

Informally, $\ell \le k \le u + 1$ (3)

- If $k \in [\ell, u]$ then $a[k] = b[k]$ (1). Since $k \le u$ then $a[k] = a'[k]$ (6), contradicting $a'[k] \ne b[k]$ (4).
- if $k = u + 1$, $a'[k] \ne b[k] = b[u+1] = a'[u+1] = a'[k]$ by (4) and (5), a contradiction.

Hence, F is $T_A^{\mathbb{Z}}$-unsatisfiable.

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 $T_A^{\mathbb{Z}}$:

$$\forall i.\ \ell \leq i \leq u \ \rightarrow\ a[i] = b[i]$$

- Universal properties $F[x]$ in T_A:

$$\forall i.\ F[a[i]]$$

- Bounded universal properties $F[x]$ in $T_A^{\mathbb{Z}}$:

$$\forall i.\ \ell \leq i \leq u \ \rightarrow\ F[a[i]]$$

- Bounded and unbounded sorted arrays $\text{sorted}(a, \ell, u)$ in $T_A^{\mathbb{Z}} \cup T_{\mathbb{Z}}$ or $T_A^{\mathbb{Z}} \cup T_{\mathbb{Q}}$:

$$\forall i, j.\ \ell \leq i \leq j \leq u \ \rightarrow\ a[i] \leq a[j]$$

- Partitioned arrays $\text{partitioned}(a, \ell_1, u_1, \ell_2, u_2)$ in $T_A^{\mathbb{Z}} \cup T_{\mathbb{Z}}$ or $T_A^{\mathbb{Z}} \cup T_{\mathbb{Q}}$:

$$\forall i, j,\ \ell_1 \leq i \leq u_1 < \ell_2 \leq j \leq u_2 \ \rightarrow\ a[i] \leq a[j]$$

The Calculus of Computation: Decision Procedures with Applications to Verification

by
Aaron Bradley
Zohar Manna

Springer 2007

12. Invariant Generation

Invariant Generation

Discover inductive assertions of programs

- General procedure
- Concrete analysis

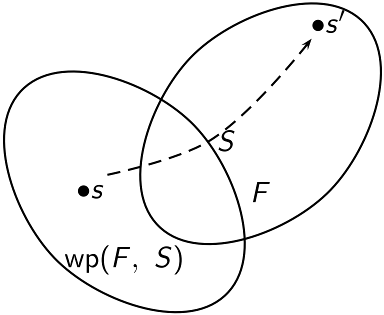
  - interval analysis
    invariants of form
    $$c \leq v \text{ or } v \leq c$$
    for program variable v and constant c
  - Karr's analysis
    invariants of form
    $$c_0 + c_1 x_1 + \cdots + c_n x_n = 0$$
    for program variables x_i and constants c_i

Other invariant generation algorithms in literature:

- linear inequalities
  $c_0 + c_1 x_1 + \cdots + c_n x_n \leq 0$
- 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

$$s \models \mathrm{wp}(F,\ S)$$

and if statement S is executed on state s to produce state s', then

$$s' \models F\ .$$

In other words, the weakest precondition moves a formula backwards over a series of statements:
for F to hold after executing $S_1; \ldots; S_n$,
$\text{wp}(F,\ S_1; \ldots; S_n)$ must hold before executing the statements.

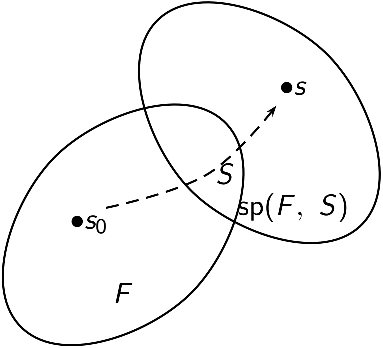
For assume and assignment statements

- $\text{wp}(F,\ \texttt{assume}\ c) \ \Leftrightarrow\ c \ \rightarrow\ F$, and
- $\text{wp}(F[v],\ v := e) \ \Leftrightarrow\ F[e]$;

and on sequences of statements $S_1; \ldots; S_n$:

$$\text{wp}(F,\ S_1; \ldots; S_n) \ \Leftrightarrow\ \text{wp}(\text{wp}(F,\ S_n),\ S_1; \ldots; S_{n-1})\ .$$

Strongest Postcondition



For FOL formula F and program statement S, the strongest postcondition $\text{sp}(F,\ S)$ is a FOL formula s.t. if s is the current state and

$$s \models \text{sp}(F,\ S)$$

then statement S was executed from a state s_0 s.t.

$$s_0 \models F\ .$$

- On assume statements,

$$\mathrm{sp}(F,\ \texttt{assume}\ c)\ \Leftrightarrow\ c\ \wedge\ F\ ,$$

for 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:

$$\mathrm{sp}(F[v],\ v := e[v])\ \Leftrightarrow\ \exists v^0.\ v = e[v^0]\ \wedge\ F[v^0]\ .$$

- On a sequence of statements $S_1; \ldots; S_n$:

$$\mathrm{sp}(F,\ S_1; \ldots; S_n)\ \Leftrightarrow\ \mathrm{sp}(\mathrm{sp}(F,\ S_1),\ S_2; \ldots; S_n)\ .$$

Example: Compute

$$\begin{aligned}
&\mathrm{sp}(i \geq n,\ i := i + k) \\
&\quad\Leftrightarrow\ \exists i^0.\ i = i^0 + k\ \wedge\ i^0 \geq n \\
&\quad\Leftrightarrow\ i - k \geq n
\end{aligned}$$

since $i^0 = i - k$.

Example: Compute

$$\begin{aligned}
&\mathrm{sp}(i \geq n,\ \texttt{assume}\ k \geq 0;\ i := i + k) \\
&\quad\Leftrightarrow\ \mathrm{sp}(\mathrm{sp}(i \geq n,\ \texttt{assume}\ k \geq 0),\ i := i + k) \\
&\quad\Leftrightarrow\ \mathrm{sp}(k \geq 0\ \wedge\ i \geq n,\ i := i + k) \\
&\quad\Leftrightarrow\ \exists i^0.\ i = i^0 + k\ \wedge\ k \geq 0\ \wedge\ i^0 \geq n \\
&\quad\Leftrightarrow\ k \geq 0\ \wedge\ i - k \geq n
\end{aligned}$$

Verification Condition

VCs in terms of wp:

$$\{F\}S_1; \ldots ; S_n\{G\} : \; F \;\Rightarrow\; \text{wp}(G,\; S_1; \ldots ; S_n) \; .$$

VCs in terms of sp:

$$\{F\}S_1; \ldots ; S_n\{G\} : \; \text{sp}(F,\; S_1; \ldots ; S_n) \;\Rightarrow\; G \; .$$

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
  $$\mu : \mathcal{L} \to \text{FOL}$$
  (map from $\mathcal{L}$ to first-order assertions).
  It is inductive (inductive map) if for each basic path

$$(\cdot)$$

$L_i : \ @\ \mu(L_i)$
$S_i;$
$\vdots$
$S_j;$
$L_j : \ @\ \mu(L_j)$

for $L_i, L_j \in \mathcal{L}$, the verification condition
$$\{\mu(L_i)\}S_i; \ldots ; S_j\{\mu(L_j)\} \qquad \text{(VC)}$$
is valid.

Invariant Generation

Find inductive assertion maps μ s.t. the $\mu(L_i)$ satisfies (VC) for all basic paths.

Method: Symbolic execution (formward propagation)

- Initial map μ_0:

$$\begin{array}{ll} \mu(L_0) := F_{\text{pre}} \ , & \text{and} \\ \mu(L) := \bot & \text{for} \quad L \in \mathcal{L} \setminus \{L_0\}. \end{array}$$

- 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 μ_i. Choose some $L_j \in S$ to process and remove it from S.

For each basic path (starting at L_j)

$$(\cdot)$$

$L_j: \ @\ \mu(L_j)$
S_j;
$\vdots$
S_k;
$L_k: \ @\ \mu(L_k)$

compute and set

$$\mu(L_k) \Leftrightarrow \mu(L_k) \vee \mathrm{sp}(\mu(L_j),\ S_j; \ldots ; S_k)$$

If

$$\mathrm{sp}(\mu(L_j),\ S_j; \ldots ; S_k) \Rightarrow \mu_i(L_k)$$

that is, if sp does not represent new states not already represented in $\mu_i(L_k)$, then $\mu_{i+1}(L_k) \Leftrightarrow \mu_i(L_k)$ (nothing new is learned)

Otherwise add L_k to S.

For all other locations $L_\ell \in \mathcal{L}, \mu_{i+1}(L_\ell) \Leftrightarrow \mu_i(L_\ell)$

When $S = \emptyset$ (say iteration i^*), then μ_{i^*} is an inductive map.

The algorithm

let $\textsc{ForwardPropagate}\ P\ F_{\text{pre}}\ \mathcal{L} =$

$S := \{L_0\};$

$\mu(L_0) := F_{\text{pre}};$

$\mu(L) := \bot$ for $L \in \mathcal{L} \setminus \{L_0\};$

while $S \neq \emptyset$ do

let $L_j = \textsc{choose}\ S$ in

$S := S \setminus \{L_j\};$

foreach $L_k \in \text{succ}(L_j)$ do [$L_k \in \text{succ}(L_j)$ is a **successor** of L_j if there is a basic path from L_j to L_k]

let $F = \text{sp}(\mu(L_j),\ S_j; \ldots ; S_k)$ in

if $F \not\Rightarrow \mu(L_k)$

then $\mu(L_k) := \mu(L_k) \lor F;$

$S := S \cup \{L_k\};$

done;

done;

μ

Problem: algorithm may not terminate

Example: Consider loop with integer variables i and n:

$@L_0 : i = 0 \ \land \ n \geq 0;$

```
while
```

$@L_1 : ?$

$(i < n) \ \{$

$i := i + 1;$

$\}$

There are two basic paths:

(1)

$@L_0 : \ i = 0 \ \land \ n \geq 0;$

$@L_1 : ?;$

and

(2)

$@L_1 : ?;$

- Initially,

$$\boxed{\begin{array}{lcl} \mu(L_0) & \Leftrightarrow & i = 0 \ \wedge \ n \geq 0 \\ \mu(L_1) & \Leftrightarrow & \bot \end{array}}$$

- Following path **(1)** results in setting

$$\mu(L_1) := \mu(L_1) \ \vee \ (i = 0 \ \wedge \ n \geq 0)$$

$\mu(L_1)$ was $\bot$, so that it becomes

$$\boxed{\mu(L_1) \ \Leftrightarrow \ i = 0 \ \wedge \ n \geq 0 \ .}$$

- On the next iteration, following path **(2)** yields

$$\mu(L_1) := \mu(L_1) \ \vee \ \mathrm{sp}(\mu(L_1), \ \texttt{assume}\ i < n; \ i := i + 1) \ .$$

Currently $\mu(L_1) \ \Leftrightarrow \ i = 0 \ \wedge \ n \geq 0$, so

$$\begin{aligned} F: \ \mathrm{sp}(i = 0 \ \wedge \ n \geq 0, \ \texttt{assume}\ i < n; \ i := i + 1) \\ \Leftrightarrow \ \mathrm{sp}(i < n \ \wedge \ i = 0 \ \wedge \ n \geq 0, \ i := i + 1) \\ \Leftrightarrow \ \exists i^0.\ i = i^0 + 1 \ \wedge \ i^0 < n \ \wedge \ i^0 = 0 \ \wedge \ n \geq 0 \\ \Leftrightarrow \ i = 1 \ \wedge \ n > 0 \end{aligned}$$

Since the implication

$$\underbrace{i = 1 \ \wedge \ n > 0}_{F} \ \Rightarrow \ \underbrace{i = 0 \ \wedge \ n \geq 0}_{\mu(L_1)}$$

is invalid,

$$\boxed{\mu(L_1) \ \Leftrightarrow \ \underbrace{(i = 0 \ \wedge \ n \geq 0)}_{\mu(L_1)} \vee \underbrace{(i = 1 \ \wedge \ n > 0)}_{F}}$$

at the end of the iteration.

- At the end of the next iteration,

$$\boxed{\mu(L_1) \ \Leftrightarrow \ \begin{array}{l} \underbrace{(i = 0 \ \wedge \ n \geq 0) \ \vee \ (i = 1 \ \wedge \ n > 0)}_{\mu(L_1)} \\ \vee \ \underbrace{(i = 2 \ \wedge \ n > 1)}_{F} \end{array}}$$

- At the end of the kth iteration,

$$\boxed{\mu(L_1) \;\Leftrightarrow\; \begin{array}{l}(i = 0 \;\wedge\; n \geq 0) \;\vee\; (i = 1 \;\wedge\; n \geq 1) \\ \quad\vee\; \cdots \;\vee\; (i = k \;\wedge\; n \geq k)\end{array}}$$

  It is never the case that the implication

$$i = k \;\wedge\; n \geq k$$
$$\Downarrow$$
$$(i = 0 \wedge n \geq 0) \vee (i = 1 \wedge n \geq 1) \vee \cdots \vee (i = k - 1 \wedge n \geq k - 1)$$

  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.

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.

Step 1: Choose an abstract domain D.

The **abstract domain** D is a syntactic class of Σ-formulae of some theory T.

- **interval abstract domain** D_{I} consists of conjunctions of $\Sigma_{\mathbb{Q}}$-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 $\Sigma_{\mathbb{Q}}$-literals of the form

$$c_0 + c_1 x_1 + \cdots + c_n x_n = 0 \ ,$$

  for constants $c_0, c_1, \ldots, c_n$ and variables $x_1, \ldots, x_n$.

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

$$\nu_{D_\text{I}}(F) : \ 0 \leq i \ \wedge \ i \leq 0 \ \wedge \ 0 \leq n$$

and in Karr's abstract domain by

$$\nu_{D_\text{K}}(F) : \ i = 0$$

with some loss of information.

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:

  $$\text{sp}(F,\ \texttt{assume}\ c) \ \Leftrightarrow\ c\ \wedge\ F\ .$$

  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 $F \in D$,

  $$\overline{\text{sp}}_D(F,\ \texttt{assume}\ c) \ \Leftrightarrow\ \nu_D(c)\ \sqcap_D\ F\ .$$

  If the abstract domain D consists of conjunctions of literals, $\sqcap_D$ is just $\wedge$. For example, in the interval domain,

  $$\overline{\text{sp}}_{D_I}(F,\ \texttt{assume}\ c) \ \Leftrightarrow\ \nu_{D_I}(c)\ \wedge\ F\ .$$

- assignment statements:
  More complex, for suppose that we use the standard definition

$$\mathrm{sp}(F[v],\ v := e[v]) \Leftrightarrow \underbrace{\exists v^0.\ v = e[v^0] \ \wedge\ F[v^0]}_{G}\ ,$$

  which requires existential quantification. Then, later, when we compute the validity of

$$G \ \Rightarrow\ \mu(L)\ , \quad \text{i.e.,} \quad \forall \overline{b}.\ G \ \rightarrow\ \mu(L)\ ,$$

  $\mu(L)$ can contain existential quantification, resulting in a quantifier alternation. Most decision procedures, apply only to quantifier-free formulae. Therefore, introducing existential quantification in $\overline{\mathrm{sp}}$ is undesirable.

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 $\triangledown_D$.
A **widening operator** $\triangledown_D$ is a binary function

$$\triangledown_D : D \times D \to D$$

such that

$$F_1 \vee F_2 \;\Rightarrow\; F_1 \triangledown_D F_2$$

for $F_1, F_2 \in D$. It obeys the following property. Let $F_1, F_2, F_3, \ldots$ be an infinite sequence of elements $F_i \in D$ such that for each i,

$$F_i \;\Rightarrow\; F_{i+1} \ .$$

Define the sequence

$$G_1 = F_1 \quad \text{and} \quad G_{i+1} = G_i \triangledown_D F_{i+1} \ .$$

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.

$\texttt{let}$ $\textsc{AbstractForwardPropagate}\ P\ F_{\mathsf{pre}}\ \mathcal{L} =$

$\quad S := \{L_0\};$

$\quad \mu(L_0) := \nu_D(F_{\mathsf{pre}});$

$\quad \mu(L) := \bot\ \texttt{for}\ L \in \mathcal{L} \setminus \{L_0\};$

$\quad \texttt{while}\ S \neq \emptyset\ \texttt{do}$

$\qquad \texttt{let}\ L_j\ =\ \textsc{choose}\ S\ \texttt{in}$

$\qquad S := S \setminus \{L_j\};$

$\qquad \texttt{foreach}\ L_k \in \mathsf{succ}(L_j)\ \texttt{do}$

$\qquad\quad \texttt{let}\ F\ =\ \overline{\mathsf{sp}}_D(\mu(L_j),\ S_j; \ldots; S_k)\ \texttt{in}$

$\qquad\quad \texttt{if}\ F \not\Rightarrow \mu(L_k)$

$\qquad\quad \texttt{then if}\ \textsc{Widen}()$

$\qquad\qquad\quad \texttt{then}\ \mu(L_k) := \mu(L_k) \bigtriangledown_D (\mu(L_k)\ \sqcup_D\ F);$

$\qquad\qquad\quad \texttt{else}\ \mu(L_k) := \mu(L_k)\ \sqcup_D\ F;$

$\qquad\qquad\quad S := S \cup \{L_k\};$

$\qquad \texttt{done};$

$\quad \texttt{done};$

$\quad \mu$