Decision Procedures

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

Winter Term 2015/16

Organisation

Organisation

Dates

- Lecture is Tuesday 14–16 (c.t) and Thursday 14–15 (c.t).
- Tutorials will be given on Thursday 15–16.
  Starting next week (this week is a two hour lecture).
- Exercise sheets are uploaded on Tuesday.
  They are due on Tuesday the week after.

To successfully participate, you must

- prepare the exercises (at least 50 %)
- actively participate in the tutorial
- pass an oral examination

THE CALCULUS OF COMPUTATION:
Decision Procedures with
Applications to Verification

by
Aaron Bradley
Zohar Manna

Springer 2007

Motivation

Motivation

Decision Procedures are algorithms to decide formulae.
These formulae can arise

- in Hoare-style software verification,
- in hardware verification,
- in synthesis,
- in scheduling,
- in planning,
- ...

Consider the following program:

for
$@\ \ell \leq i \leq u \wedge (rv \leftrightarrow \exists j.\ \ell \leq j < i \wedge a[j] = e)$
$(\texttt{int}\ i := \ell;\ i \leq u;\ i := i + 1)\ \{$
$\texttt{if}\ ((a[i] = e))\ \{$
$rv := \texttt{true};$
$\}$
$\}$

How can we prove that the formula is a loop invariant?

Motivation (3)

Prove the Hoare triples (one for if case, one for else case)

$\texttt{assume}\ \ell \leq i \leq u \wedge (rv \leftrightarrow \exists j.\ \ell \leq j < i \wedge a[j] = e)$
$\texttt{assume}\ i \leq u$
$\texttt{assume}\ a[i] = e$
$rv := \texttt{true};$
$i := i + 1$
$\texttt{@}\ \ell \leq i \leq u \wedge (rv \leftrightarrow \exists j.\ \ell \leq j < i \wedge a[j] = e)$

$\texttt{assume}\ \ell \leq i \leq u \wedge (rv \leftrightarrow \exists j.\ \ell \leq j < i \wedge a[j] = e)$
$\texttt{assume}\ i \leq u$
$\texttt{assume}\ a[i] \neq e$
$i := i + 1$
$\texttt{@}\ \ell \leq i \leq u \wedge (rv \leftrightarrow \exists j.\ \ell \leq j < i \wedge a[j] = e)$

Motivation (4)

A Hoare triple {P} S {Q} holds, iff

$$P \to wp(S, Q)$$

(wp denotes is weakest precondition)
For assignments wp is computed by substitution:

$\texttt{assume}\ \ell \leq i \leq u \wedge (rv \leftrightarrow \exists j.\ \ell \leq j < i \wedge a[j] = e)$
$\texttt{assume}\ i \leq u$
$\texttt{assume}\ a[i] = e$
$rv := \texttt{true};$
$i := i + 1$
$\texttt{@}\ \ell \leq i \leq u \wedge (rv \leftrightarrow \exists j.\ \ell \leq j < i \wedge a[j] = e)$

holds if and only if:

$$\begin{aligned} & \ell \leq i \leq u \wedge (rv \leftrightarrow \exists j.\ \ell \leq j < i \wedge a[j] = e) \wedge i \leq u \wedge a[i] = e \\ \to & \ell \leq i + 1 \leq u \wedge (\texttt{true} \leftrightarrow \exists j.\ \ell \leq j < i + 1 \wedge a[j] = e) \end{aligned}$$

We need an algorithm that decides whether a formula holds.

$$\begin{aligned}&\ell \leq i \leq u \land (rv \leftrightarrow \exists j.\, \ell \leq j < i \land a[j] = e) \land i \leq u \land a[i] = e\\ \rightarrow &\ell \leq i+1 \leq u \land (\texttt{true} \leftrightarrow \exists j.\, \ell \leq j < i+1 \land a[j] = e)\end{aligned}$$

If the formula does not hold it should give a counterexample, e.g.:

$$\ell = 0, i = 1, u = 1, rv = \mathit{false}, a[0] = 0, a[1] = 1, e = 1,$$

This counterexample shows that $i + 1 \leq u$ can be violated.

This lecture is about algorithms checking for validity and producing these counterexamples.

Contents of Lecture

Topics

- Propositional Logic
- First-Order Logic
- First-Order Theories
- Quantifier Elimination
- Decision Procedures for Linear Arithmetic
- Decision Procedures for Uninterpreted Functions
- Decision Procedures for Arrays
- Combination of Decision Procedures
- DPLL(T)
- Craig Interpolants

Foundations: Propositional Logic

Syntax of Propositional Logic

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: Syntax

formula F : $((P \land Q) \to (\top \lor \neg Q))$
atoms: $P, Q, \top$
literal: $\neg Q$
subformulas: $(P \land Q)$, $\quad (\top \lor \neg Q)$

Parentheses can be omitted: $\quad F : P \land Q \to \top \lor \neg Q$

- $\neg$ binds stronger than
- $\land$ binds stronger than
- $\lor$ binds stronger than
- $\to, \leftrightarrow$.

Semantics (meaning) of PL

Formula F and Interpretation I is evaluated to a truth value 0/1
where 0 corresponds to value false
1 true

Interpretation I : $\{P \mapsto 1, Q \mapsto 0, \cdots\}$

Evaluation of logical operators:

F_1	F_2	$\neg F_1$	$F_1 \land F_2$	$F_1 \lor F_2$	$F_1 \rightarrow F_2$	$F_1 \leftrightarrow F_2$
0	0	1	0	0	1	1
0	1		0	1	1	0
1	0	0	0	1	0	0
1	1		1	1	1	1

Example: Semantics

$F : P \wedge Q \rightarrow P \vee \neg Q$
$I : \{P \mapsto 1, Q \mapsto 0\}$

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 1 / true under I
$I \not\models F$ 0 / false

Base Case:

$$I \models \top$$
$$I \not\models \bot$$
$$I \models P \quad \text{iff} \quad I[P] = 1$$
$$I \not\models P \quad \text{iff} \quad I[P] = 0$$

Inductive Case:

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

Example: Inductive Reasoning

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

$$I : \{P \mapsto 1,\ Q \mapsto 0\}$$

1.	$I \models P$	since $I[P] = 1$
2.	$I \not\models Q$	since $I[Q] = 0$
3.	$I \models \neg Q$	by 2, $\neg$
4.	$I \not\models P \wedge Q$	by 2, $\wedge$
5.	$I \models P \vee \neg Q$	by 1, $\vee$
6.	$I \models F$	by 4, $\rightarrow$ Why?

Thus, F is true under I.

Satisfiability and Validity

Definition (Satisfiability)

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

Definition (Validity)

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

Note

F is valid iff $\neg F$ is unsatisfiable

Proof.

F is valid iff $\forall I : I \models F$ iff $\neg \exists I : I \not\models F$ iff $\neg F$ is unsatisfiable. $\square$

Decision Procedure: An algorithm for deciding validity or satisfiability.

Examples: Satisfiability and Validity

Now assume, you are a decision procedure.

Which of the following formulae is satisfiable, which is valid?

- $F_1 : P \wedge Q$
  satisfiable, not valid
- $F_2 : \neg(P \wedge Q)$
  satisfiable, not valid
- $F_3 : P \vee \neg P$
  satisfiable, valid
- $F_4 : \neg(P \vee \neg P)$
  unsatisfiable, not valid
- $F_5 : (P \rightarrow Q) \wedge (P \vee Q) \wedge \neg Q$
  unsatisfiable, not valid

Is there a formula that is unsatisfiable and valid?

Decision Procedure

We will present three Decision Procedures for propositional logic

- Truth Tables
- Semantic Tableaux
- DPLL/CDCL

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

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

Thus F is valid.

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

P	Q	$P \vee Q$	$P \wedge Q$	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 (Semantic Tableaux)

- Assume F is not valid and I a falsifying interpretation: $I \not\models F$
- Apply proof rules.
- If no contradiction reached and no more rules applicable, F is invalid.
- If in every branch of proof a contradiction reached, F is valid.

Semantic Argument: Proof rules

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

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

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

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

$$\frac{I \models F \leftrightarrow G}{I \models F \land G \quad \mid \quad I \not\models F \lor G} \qquad \frac{I \not\models F \leftrightarrow G}{I \models F \land \neg G \quad \mid \quad I \models \neg F \land G}$$

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

Example

Prove $\quad F : P \land Q \rightarrow P \lor \neg Q \quad$ is valid.

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

1.	$I \not\models P \land Q \rightarrow P \lor \neg Q$		assumption
2.	$I \models P \land Q$		1, Rule $\rightarrow$
3.	$I \not\models P \lor \neg Q$		1, Rule $\rightarrow$
4.	$I \models P$		2, Rule $\land$
5.	$I \not\models P$		3, Rule $\lor$
6.	$I \models \bot$		4 and 5 are contradictory

Thus F is valid.

Example 2

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

Let's assume that F is not valid.

$$
\begin{array}{lll}
1. & I \not\models F & \text{assumption} \\
2. & I \models (P \to Q) \wedge (Q \to R) & 1, \text{Rule } \to \\
3. & I \not\models P \to R & 1, \text{Rule } \to \\
4. & I \models P & 3, \text{Rule } \to \\
5. & I \not\models R & 3, \text{Rule } \to \\
6. & I \models P \to Q & 2, \text{Rule } \wedge \\
7. & I \models Q \to R & 2, \text{Rule } \wedge
\end{array}
$$

$$
\begin{array}{ll|ll}
8a. & I \not\models P & 8b. & I \models Q \quad 6 \to \\
9a. & I \models \bot & \begin{array}{ll} 9ba. & I \not\models Q \\ 10ba. & I \models \bot \end{array} & \begin{array}{|ll} 9bb. & I \models R \\ 10bb. & I \models \bot \end{array}
\end{array}
$$

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

Example 3

Is $\quad F : P \vee Q \rightarrow P \wedge Q \quad$ valid?

Let's assume that F is not valid.

$$
\begin{array}{llll}
1. & I \not\models P \vee Q \rightarrow P \wedge Q & & \text{assumption} \\
2. & I \models P \vee Q & & 1 \text{ and } \rightarrow \\
3. & I \not\models P \wedge Q & & 1 \text{ and } \rightarrow
\end{array}
$$

$$
\begin{array}{ll|ll}
4a. \; I \models P & 2 \text{ and } \vee & 4b. \; I \models Q & 2 \text{ and } \vee \\
\begin{array}{l|l} 5aa. \; I \not\models P & 5ab. \; I \not\models Q \\ 6aa. \; I \models \bot & \end{array} & & \begin{array}{l|l} 5ba. \; I \not\models P & 5bb. \; I \not\models Q \\ & 6bb. \; I \models \bot \end{array} &
\end{array}
$$

We cannot always 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 all cases for F to be valid.

DPLL/CDCL is a efficient decision procedure for propositional logic.
History:

- 1960s: Davis, Putnam, Logemann, and Loveland presented DPLL.
- 1990s: Conflict Driven Clause Learning (CDCL).
- Today, very efficient solvers using specialized data structures and improved heuristics.

DPLL/CDCL doesn't work on arbitrary formulas, but only on a certain normal form.

Normal Forms

Idea: Simplify decision procedure, by simplifying the formula first.
Convert it into a simpler normal form, e.g.:

- Negation Normal Form: No $\rightarrow$ and no $\leftrightarrow$; negation only before atoms.
- Conjunctive Normal Form: Negation normal form, where conjunction is outside, disjunction is inside.
- Disjunctive Normal Form: Negation normal form, where disjunction is outside, conjunction is inside.

The formula in normal form should be equivalent to the original input.

Equivalence

F_1 and F_2 are equivalent ($F_1 \Leftrightarrow F_2$)
iff for all interpretations I, $I \models F_1 \leftrightarrow F_2$

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

F_1 implies F_2 ($F_1 \Rightarrow F_2$)
iff for all interpretations I, $I \models F_1 \rightarrow F_2$

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

Equivalence is a Congruence relation

If $F_1 \Leftrightarrow F_1'$ and $F_2 \Leftrightarrow F_2'$, then

- $\neg F_1 \Leftrightarrow \neg F_1'$
- $F_1 \vee F_2 \Leftrightarrow F_1' \vee F_2'$
- $F_1 \wedge F_2 \Leftrightarrow F_1' \wedge F_2'$
- $F_1 \rightarrow F_2 \Leftrightarrow F_1' \rightarrow F_2'$
- $F_1 \leftrightarrow F_2 \Leftrightarrow F_1' \leftrightarrow F_2'$

- if we replace in a formula F a subformula F_1 by F_1' and obtain F', then $F \Leftrightarrow F'$.

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):

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

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

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

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

Example: Negation Normal Form

Convert $F : (Q_1 \vee \neg\neg R_1) \wedge (\neg Q_2 \rightarrow R_2)$ into NNF

$$\begin{array}{rl} & (Q_1 \vee \neg\neg R_1) \wedge (\neg Q_2 \rightarrow R_2) \\ \Leftrightarrow & (Q_1 \vee R_1) \wedge (\neg Q_2 \rightarrow R_2) \\ \Leftrightarrow & (Q_1 \vee R_1) \wedge (\neg\neg Q_2 \vee R_2) \\ \Leftrightarrow & (Q_1 \vee R_1) \wedge (Q_2 \vee R_2) \end{array}$$

The last formula is equivalent to F and is in NNF.

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):

$$\left.\begin{array}{l}(F_1 \vee F_2) \wedge F_3 \Leftrightarrow (F_1 \wedge F_3) \vee (F_2 \wedge F_3) \\ F_1 \wedge (F_2 \vee F_3) \Leftrightarrow (F_1 \wedge F_2) \vee (F_1 \wedge F_3)\end{array}\right\} \mathit{dist}$$

Example

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

$$\begin{array}{lll} & (Q_1 \lor \neg\neg R_1) \land (\neg Q_2 \to R_2) & \\ \Leftrightarrow & (Q_1 \lor R_1) \land (Q_2 \lor R_2) & \text{in NNF} \\ \Leftrightarrow & (Q_1 \land (Q_2 \lor R_2)) \lor (R_1 \land (Q_2 \lor R_2)) & \text{dist} \\ \Leftrightarrow & (Q_1 \land Q_2) \lor (Q_1 \land R_2) \lor (R_1 \land Q_2) \lor (R_1 \land R_2) & \text{dist} \end{array}$$

The last formula is equivalent to F and is in DNF. Note that formulas can grow exponentially.

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):

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

A disjunction of literals $P_1 \vee P_2 \vee \neg P_3$ is called a clause.
For brevity we write it as set: $\{P_1, P_2, \overline{P_3}\}$.
A formula in CNF is a set of clauses (a set of sets of literals).

Equisatisfiability

Definition (Equisatisfiability)

F and F' are equisatisfiable, iff

$$F \text{ is satisfiable if and only if } F' \text{ is satisfiable}$$

Every formula is equisatifiable to either $\top$ or $\bot$.

There is a efficient conversion of F to F' where

- F' is in CNF and
- F and F' are equisatisfiable

Note: efficient means polynomial in the size of F.

Conversion to equisatisfiable CNF

Basic Idea:

- Introduce a new variable P_G for every subformula G; unless G is already an atom.
- For each subformula $G : G_1 \circ G_2$ produce a small formula $P_G \leftrightarrow P_{G_1} \circ P_{G_2}$.
- encode each of these (small) formulae separately to CNF.

The formula

$$P_F \wedge \bigwedge_G CNF(P_G \leftrightarrow P_{G_1} \circ P_{G_2})$$

is equisatisfiable to F.

The number of subformulae is linear in the size of F.

The time to convert one small formula is constant!

Example: CNF

Convert $F : P \vee Q \rightarrow P \wedge \neg R$ to CNF.
Introduce new variables: P_F, $P_{P\vee Q}$, $P_{P\wedge\neg R}$, $P_{\neg R}$. Create new formulae and convert them to CNF separately:

- $P_F \leftrightarrow (P_{P\vee Q} \rightarrow P_{P\wedge\neg R})$ in CNF:

$$F_1 : \{\{\overline{P_F}, \overline{P_{P\vee Q}}, P_{P\wedge\neg R}\}, \{P_F, P_{P\vee Q}\}, \{P_F, \overline{P_{P\wedge\neg R}}\}\}$$

- $P_{P\vee Q} \leftrightarrow P \vee Q$ in CNF:

$$F_2 : \{\{\overline{P_{P\vee Q}}, P \vee Q\}, \{P_{P\vee Q}, \overline{P}\}, \{P_{P\vee Q}, \overline{Q}\}\}$$

- $P_{P\wedge\neg R} \leftrightarrow P \wedge P_{\neg R}$ in CNF:

$$F_3 : \{\{\overline{P_{P\wedge\neg R}} \vee P\}, \{\overline{P_{P\wedge\neg R}}, P_{\neg R}\}, \{P_{P\wedge\neg R}, \overline{P}, \overline{P_{\neg R}}\}\}$$

- $P_{\neg R} \leftrightarrow \neg R$ in CNF: $F_4 : \{\{\overline{P_{\neg R}}, \overline{R}\}, \{P_{\neg R}, R\}\}$

$\{\{P_F\}\} \cup F_1 \cup F_2 \cup F_3 \cup F_4$ is in CNF and equisatisfiable to F.

- Algorithm to decide PL formulae in CNF.
- Published by Davis, Logemann, Loveland (1962).
- Often miscited as Davis, Putnam (1960), which describes a different algorithm.

Davis-Putnam-Logemann-Loveland (DPLL) Algorithm

Decides the satisfiability of PL formulae in CNF

Decision Procedure DPLL: Given F in CNF

$$
\begin{array}{l}
\texttt{let rec } \textsc{dpll}\ F\ = \\
\quad \texttt{let } F' = \textsc{prop}\ F \texttt{ in} \\
\quad \texttt{let } F'' = \textsc{plp}\ F' \texttt{ in} \\
\quad \texttt{if } F'' = \top \texttt{ then true} \\
\quad \texttt{else if } F'' = \bot \texttt{ then false} \\
\quad \texttt{else} \\
\qquad \texttt{let } P = \textsc{choose}\ \mathsf{vars}(F'') \texttt{ in} \\
\qquad (\textsc{dpll}\ F''\{P \mapsto \top\}) \vee (\textsc{dpll}\ F''\{P \mapsto \bot\})
\end{array}
$$

Unit Propagation (PROP)

If a clause contains one literal ℓ,

- Set ℓ to $\top$.
- Remove all clauses containing ℓ.
- Remove $\neg\ell$ in all clauses.

Based on resolution

$$\frac{\ell \qquad \neg\ell \vee C}{C} \leftarrow \text{clause}$$

Pure Literal Propagagion

Pure Literal Propagation (PLP)

If P occurs only positive (without negation), set it to $\top$.
If P occurs only negative set it to $\bot$.

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

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

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

On the other branch

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

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

F is satisfiable with satisfying interpretation

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

Example

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

$$F$$

$Q \mapsto \top$

$$(R) \wedge (\neg R) \wedge (P \vee \neg R)$$

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

$Q \mapsto \bot$

$$(\neg P \vee R)$$

$R \mapsto \top$

$$\neg P$$

$P \mapsto \bot$

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

Knight and Knaves

A island is inhabited only by knights and knaves. Knights always tell the truth, and knaves always lie. You meet four inhabitants: Alice, Bob, Charles and Doris.

- Alice says that Doris is a knave.
- Bob tells you that Alice is a knave.
- Charles claims that Alice is a knave.
- Doris tells you, ‘Of Charles and Bob, exactly one is a knight.’

Knight and Knaves

Let A denote that Alice is a Knight, etc. Then:

- $A \leftrightarrow \neg D$
- $B \leftrightarrow \neg A$
- $C \leftrightarrow \neg A$
- $D \leftrightarrow \neg(C \leftrightarrow B)$

In CNF:

- $\{\overline{A}, \overline{D}\}, \{A, D\}$
- $\{\overline{B}, \overline{A}\}, \{B, A\}$
- $\{\overline{C}, \overline{A}\}, \{C, A\}$
- $\{\overline{D}, \overline{C}, \overline{B}\}, \{\overline{D}, C, B\}, \{D, \overline{C}, B\}, \{D, C, \overline{B}\}$

Solving Knights and Knaves

$$F : \{\{\overline{A}, \overline{D}\}, \{A, D\}, \{\overline{B}, \overline{A}\}, \{B, A\}, \{\overline{C}, \overline{A}\}, \{C, A\}, \{\overline{D}, \overline{C}, \overline{B}\}, \{\overline{D}, C, B\}, \{D, \overline{C}, B\}, \{D, C, \overline{B}\}\}$$

PROP and PLP are not applicable. Decide on A:

$F\{A \mapsto \bot\} : \{\{D\}, \{B\}, \{C\}, \{\overline{D}, \overline{C}, \overline{B}\}, \{\overline{D}, C, B\}, \{D, \overline{C}, B\}, \{D, C, \overline{B}\}\}$

By PROP we get:

$$F\{A \mapsto \bot, D \mapsto \top, B \mapsto \top, C \mapsto \top\} : \bot$$

Unsatisfiable! Now set A to $\top$:

$F\{A \mapsto \top\} : \{\{\overline{D}\}, \{\overline{B}\}, \{\overline{C}\}, \{\overline{D}, \overline{C}, \overline{B}\}, \{\overline{D}, C, B\}, \{D, \overline{C}, B\}, \{D, C, \overline{B}\}\}$

By PROP we get:

$$F\{A \mapsto \top, D \mapsto \bot, B \mapsto \bot, C \mapsto \bot\} : \top$$

Satisfying assignment!

Consider the following problem:

$$\{\{A_1, B_1\}, \{\overline{P_0}, \overline{A_1}, P_1\}, \{\overline{P_0}, \overline{B_1}, P_1\}, \{A_2, B_2\}, \{\overline{P_1}, \overline{A_2}, P_2\}, \{\overline{P_1}, \overline{B_2}, P_2\},$$
$$\dots, \{A_n, B_n\}, \{\overline{P_{n-1}}, \overline{A_n}, P_n\}, \{\overline{P_{n-1}}, \overline{B_n}, P_n\}, \{P_0\}, \{\overline{P_n}\}\}$$

For some literal orderings, we need exponentially many steps.
Note, that

$$\{\{A_i, B_i\}, \{\overline{P_{i-1}}, \overline{A_i}, P_i\}, \{\overline{P_{i-1}}, \overline{B_i}, P_i\}\} \Rightarrow \{\{\overline{P_{i-1}}, P_i\}\}$$

If we learn the right clauses, unit propagation will immediately give unsatisfiable.

Partial Assignments and Unit/Conflict Clauses

Do not change the clause set, but only assign literals (as global variables).
When you assign true to a literal ℓ,also assign false to $\bar{\ell}$.
For a partial assignment

- A clause is true if one of its literals is assigned true.
- A clause is a conflict clause if all its literals are assigned false.
- A clause is a unit clause if all but one literals are assigned false and the last literal is unassigned.

If the assignment of a literal from a conflict clause is removed we get a unit clause.
Explain unsatisfiability of partial assignment by conflict clause and learn it!

Conflict Driven Clause Learning (CDCL)

Idea: Explain unsatisfiability of partial assignment by conflict clause and learn it!

- If a conflict is found we return the conflict clause.
- If variable in conflict were derived by unit propagation use resolution rule to generate a new conflict clause.
- If variable in conflict was derived by decision, use learned conflict as unit clause

DPLL with CDCL

The functions DPLL and PROP return a conflict clause or satisfiable.

```
let rec DPLL =
  let PROP U =
    ...
  if conflictclauses ≠ ∅
    CHOOSE conflictclauses
  else if unitclauses ≠ ∅
    PROP (CHOOSE unitclauses)
  else if coreclauses ≠ ∅
     let ℓ = CHOOSE (⋃coreclauses) ∩ unassigned in
     val[ℓ] := ⊤
     let C = DPLL in
     if (C = satisfiable) satisfiable
     else
        val[ℓ] := undef
        if (ℓ̄ ∉ C) C
        else LEARN C; PROP C
  else satisfiable
```

Unit propagation

The function PROP takes a unit clause and does unit propagation. It calls DPLL recursively and returns a conflict clause or satisfiable. recursively:

$$
\begin{array}{l}
\texttt{let}\ \text{PROP}\ U = \\
\quad \texttt{let}\ \ell = \text{CHOOSE}\ U \cap \mathsf{unassigned}\ \texttt{in} \\
\quad \mathsf{val}[\ell] := \top \\
\quad \texttt{let}\ C = \text{DPLL}\ \texttt{in} \\
\quad \texttt{if}\ (C = \mathsf{satisfiable}) \\
\quad\quad \mathsf{satisfiable} \\
\quad \texttt{else} \\
\quad\quad \mathsf{val}[\ell] := \mathsf{undef} \\
\quad\quad \texttt{if}\ (\bar{\ell} \notin C)\ C \\
\quad\quad \texttt{else}\ \ U \setminus \{\ell\} \cup C \setminus \{\bar{\ell}\}
\end{array}
$$

The last line does resolution:

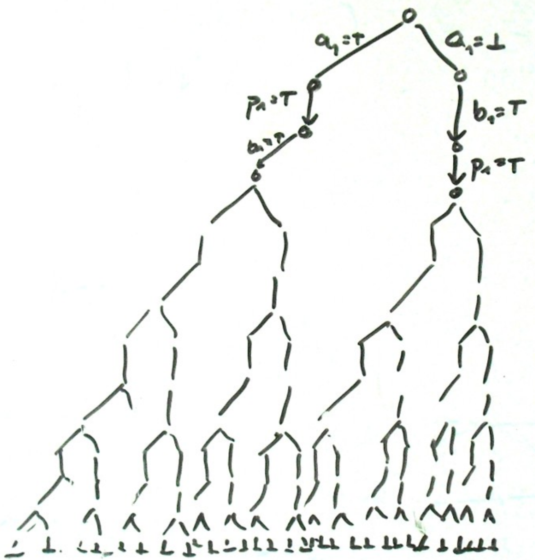
$$\frac{\ell \lor C_1 \qquad \neg\ell \lor C_2}{C_1 \lor C_2}$$

Example

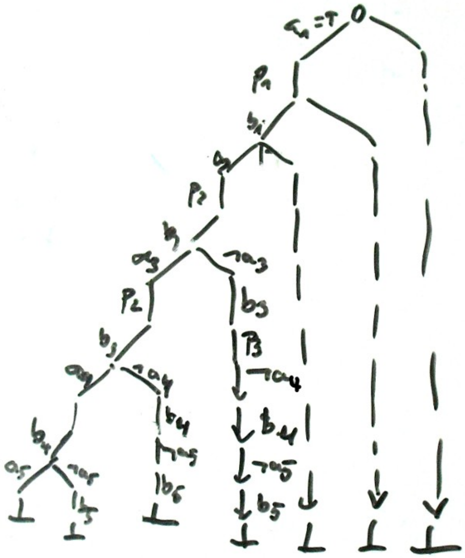
$$\{\{A_1, B_1\}, \{\overline{P_0}, \overline{A_1}, P_1\}, \{\overline{P_0}, \overline{B_1}, P_1\}, \{A_2, B_2\}, \{\overline{P_1}, \overline{A_2}, P_2\}, \{\overline{P_1}, \overline{B_2}, P_2\},$$
$$\dots, \{A_n, B_n\}, \{\overline{P_{n-1}}, \overline{A_n}, P_n\}, \{\overline{P_{n-1}}, \overline{B_n}, P_n\}, \{P_0\}, \{\overline{P_n}\}\}$$

- Unit propagation (PROP) sets P_0 and $\overline{P_n}$ to true.
- Decide, e.g. A_1, PROP sets $\overline{P_1}$
- Continue until A_{n-1}, PROP sets $\overline{P_{n-1}}, \overline{A_n}$ and $\overline{B_n}$
- Conflict clause computed: $\{\overline{A_{n-1}}, \overline{P_{n-2}}, P_n\}$.
- Conflict clause does not depend on $A_1, \dots, A_{n-2}$ and can be used again.

DPLL (without Learning)



DPLL with CDCL



Some Notes about DPLL with Learning

- Pure Literal Propagation is unnecessary:
  A pure literal is always chosen right and never causes a conflict.
- Modern SAT-solvers use this procedure but differ in
  - heuristics to choose literals/clauses.
  - efficient data structures to find unit clauses.
  - better conflict resolution to minimize learned clauses.
  - restarts (without forgetting learned clauses).
- Even with the optimal heuristics DPLL is still exponential:
  The Pidgeon-Hole problem requires exponential resolution proofs.

Summary

- Syntax and Semantics of Propositional Logic
- Methods to decide satisfiability/validity of formulae:
  - Truth table
  - Semantic Tableaux
  - DPLL
- Run-time of all presented algorithms is worst-case exponential in length of formula.
- Deciding satisfiability is NP-complete.

Further route of this lecture

- Syntax and Semantics of First Order Logic (FOL)
- Semantic Tableaux for FOL
- FOL is only semi-decidable

$\Longrightarrow$ Restrictions to decidable fragments of FOL

  - Quantifier Free Fragment (QFF)
  - QFF of Equality
  - Presburger arithmetic
  - (QFF of) Linear integer arithmetic
  - Real arithmetic
  - (QFF of) Linear real/rational arithmetic
  - QFF of Recursive Data Structures
  - QFF of Arrays
  - Putting it all together (Nelson-Oppen).

First-Order Logic

Also called Predicate Logic or Predicate Calculus

FOL Syntax

variables	$x, y, z, \cdots$
constants	$a, b, c, \cdots$
functions	$f, g, h, \cdots$ with arity $n > 0$
terms	variables, constants or
	n-ary function applied to n terms as arguments
	$a, x, f(a), g(x, b), f(g(x, f(b)))$
predicates	$p, q, r, \cdots$ with arity $n \geq 0$
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: $P, Q, R, \ldots$

Syntax of First-Order Logic (2)

<u>quantifiers</u>

existential quantifier $\exists x.F[x]$

"there exists an x such that $F[x]$"

universal quantifier $\forall x.F[x]$

"for all x, $F[x]$"

<u>FOL formula</u> 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 $q(x, f(x))$"

Famous theorems in FOL

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

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

- Fermat's Last Theorem.

$$\begin{aligned}
&\forall n.\ \mathit{integer}(n) \land n > 2 \\
&\quad \rightarrow \forall x, y, z. \\
&\qquad \mathit{integer}(x) \land \mathit{integer}(y) \land \mathit{integer}(z) \\
&\qquad \land x > 0 \land y > 0 \land z > 0 \\
&\qquad\quad \rightarrow x^n + y^n \neq z^n
\end{aligned}$$

Pumping Lemma

For every regular Language L there is some $n \geq 0$, such that for all words $z \in L$ with $|z| \geq n$ there is a decomposition $z = uvw$ with $|v| \geq 1$ and $|uv| \leq n$, such that for all $i \geq 0$: $uv^i w \in L$.

$$
\begin{array}{l}
\forall L.\ \mathit{regularlanguage}(L) \rightarrow \\
\quad \exists n.\ \mathit{integer}(n) \land n \geq 0 \land \\
\qquad \forall z.\ z \in L \land |z| \geq n \rightarrow \\
\qquad\quad \exists u, v, w.\ \mathit{word}(u) \land \mathit{word}(v) \land \mathit{word}(w) \land \\
\qquad\qquad z = uvw \land |v| \geq 1 \land |uv| \leq n \land \\
\qquad\qquad \forall i.\ \mathit{integer}(i) \land i \geq 0 \rightarrow uv^i w \in L
\end{array}
$$

Predicates: *regularlanguage*, *integer*, *word*, $\cdot \in \cdot$, $\cdot \leq \cdot$, $\cdot \geq \cdot$, $\cdot = \cdot$
Constants: 0, 1
Functions: $|\cdot|$ (word length), concatenation, iteration

FOL Semantics

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

- Domain D_I

  non-empty set of values or objects

  for example $D_I =$ playing cards (finite),
  integers (countable infinite), or
  reals (uncountable infinite)

- Assignment α_I
  - each variable x assigned value $\alpha_I[x] \in D_I$
  - each n-ary function f assigned

    $$\alpha_I[f] : D_I^n \rightarrow D_I$$

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

    $$\alpha_I[p] : D_I^n \rightarrow \{\top, \bot\}$$

    In particular, each propositional variable P (0-ary predicate) assigned truth value ($\top$, $\bot$)

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

$\alpha_I[f] : \; D_I^2 \rightarrow D_I$, $\quad (x,y) \mapsto x + y$

$\alpha_I[g] : \; D_I^2 \rightarrow D_I$, $\quad (x,y) \mapsto x - y$

$$\alpha_I[p] : \; D_I^2 \rightarrow \{\top, \bot\}$$

$$(x,y) \mapsto \begin{cases} \top & \text{if } x < y \\ \bot & \text{otherwise} \end{cases}$$

Also $\alpha_I[x] = 13, \alpha_I[y] = 42, \alpha_I[z] = 1$

Compute the truth value of F under I

1.	$I \not\models p(f(x,y),z)$	since $13 + 42 \geq 1$
2.	$I \not\models p(y,g(z,x))$	since $42 \geq 1 - 13$
3.	$I \models F$	by 1, 2, and $\rightarrow$

F is true under I

Semantics: Quantifiers

For a variable x:

Definition (x-variant)

An x-variant 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

Consider

$$F : \forall x.\ \exists y.\ 2 \cdot y = x$$

Here $2 \cdot y$ is the infix notatation of the term $\cdot(2, y)$,
and $2 \cdot y = x$ is the infix notatation of the atom $= (\cdot(2, y), x)$.

- 2 is a 0-ary function symbol (a constant).
- $\cdot$ is a 2-ary function symbol.
- $=$ is a 2-ary predicate symbol.
- x, y are variables.

What is the truth-value of F?

Example ($\mathbb{Z}$)

$$F : \forall x.\ \exists y.\ 2 \cdot y = x$$

Let I be the standard interpration for integers, $D_I = \mathbb{Z}$.
Compute the value of F under I:

$$I \models \forall x.\ \exists y.\ 2 \cdot y = x$$

iff

$$\text{for all v} \in D_I,\ I \triangleleft \{x \mapsto \text{v}\} \models \exists y.\ 2 \cdot y = x$$

iff

for all v $\in D_I$, there exists $\text{v}_1 \in D_I$, $I \triangleleft \{x \mapsto \text{v}\} \triangleleft \{y \mapsto \text{v}_1\} \models 2 \cdot y = x$

The latter is false since for $1 \in D_I$ there is no number v_1 with $2 \cdot v_1 = 1$.

Example ($\mathbb{Q}$)

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

Let I be the standard interpration for rational numbers, $D_I = \mathbb{Q}$.
Compute the value of F under I:

$$I \models \forall x.\, \exists y.\, 2 \cdot y = x$$

iff

$$\text{for all } \mathsf{v} \in D_I,\ I \triangleleft \{x \mapsto \mathsf{v}\} \models \exists y.\, 2 \cdot y = x$$

iff

for all $\mathsf{v} \in D_I$, there exists $\mathsf{v}_1 \in D_I$, $I \triangleleft \{x \mapsto \mathsf{v}\} \triangleleft \{y \mapsto \mathsf{v}_1\} \models 2 \cdot y = x$

The latter is true since for $\mathsf{v} \in D_I$ we can choose $\mathsf{v}_1 = \frac{\mathsf{v}}{2}$.

Satisfiability and Validity

Definition (Satisfiability)

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

Definition (Validity)

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

Note

F is valid iff $\neg F$ is unsatisfiable

Substitution

Suppose, we want to replace terms with other terms in formulas, e.g.

$$F \; : \; \forall y.\; (p(x, y) \rightarrow p(y, x))$$

should be transformed to

$$G \; : \; \forall y.\; (p(a, y) \rightarrow p(y, a))$$

We call the mapping from x to a a substituion denoted as $\sigma \; : \; \{x \mapsto a\}$.
We write $F\sigma$ for the formula G.
Another convenient notation is $F[x]$ for a formula containing the variable x and $F[a]$ for $F\sigma$.

Substitution

Definition (Substitution)

A substitution is a mapping from terms to terms, e.g.

$$\sigma : \{t_1 \mapsto s_1, \ldots, t_n \mapsto s_n\}$$

By $F\sigma$ we denote the application of σ to formula F, i.e., the formula F where all occurences of $t_1, \ldots, t_n$ are replaced by $s_1, \ldots, s_n$.

For a formula named $F[x]$ we write $F[t]$ as shorthand for $F[x]\{x \mapsto t\}$.

Safe Substitution

Care has to be taken in the presence of quantifiers:

$$F[x] : \exists y.\ y = \mathit{Succ}(x)$$

What is $F[y]$?
We need to rename bounded variables occuring in the substitution:

$$F[y] : \exists y'.\ y' = \mathit{Succ}(y)$$

Bounded renaming does not change the models of a formula:

$$(\exists y.\ y = \mathit{Succ}(x)) \Leftrightarrow (\exists y'.\ y' = \mathit{Succ}(x))$$

Recursive Definition of Substitution

$$t\sigma = \begin{cases} \sigma(t) & t \in \mathrm{dom}(\sigma) \\ f(t_1\sigma, \ldots, t_n\sigma) & t \notin \mathrm{dom}(\sigma) \wedge t = f(t_1, \ldots, t_n) \\ x & t \notin \mathrm{dom}(\sigma) \wedge t = x \end{cases}$$

$$p(t_1, \ldots, t_n)\sigma = p(t_1\sigma, \ldots, t_n\sigma)$$

$$(\neg F)\sigma = \neg(F\sigma)$$

$$(F \wedge G)\sigma = (F\sigma) \wedge (G\sigma)$$

$$\ldots$$

$$(\forall x.\ F)\sigma = \begin{cases} \forall x.\ F\sigma & x \notin \mathit{Vars}(\sigma) \\ \forall x'.\ ((F\{x \mapsto x'\})\sigma) & \text{otherwise and } x' \text{ is fresh} \end{cases}$$

$$(\exists x.\ F)\sigma = \begin{cases} \exists x.\ F\sigma & x \notin \mathit{Vars}(\sigma) \\ \exists x'.\ ((F\{x \mapsto x'\})\sigma) & \text{otherwise and } x' \text{ is fresh} \end{cases}$$

Example: Safe Substitution $F\sigma$

$$F \,:\, (\forall x.\ p(\underset{\nearrow}{x}, \underset{\nwarrow}{y})) \rightarrow q(f(\underset{\nearrow}{y}), \underset{\nwarrow}{x})$$

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

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

$F\sigma$?

1. Rename

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

where x' is a fresh variable

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

Recall rules from propositional logic:

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

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

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

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

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

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

The following additional rules are used for quantifiers:

$$\frac{I \models \forall x.F[x]}{I \models F[t]} \text{ for any term } t \qquad \frac{I \not\models \forall x.F[x]}{I \not\models F[a]} \text{ for a fresh constant } a$$

$$\frac{I \models \exists x.F[x]}{I \models F[a]} \text{ for a fresh constant } a \qquad \frac{I \not\models \exists x.F[x]}{I \not\models F[t]} \text{ for any term } t$$

(We assume that there are infinitely many constant symbols.)

The formula $F[t]$ is created from the formula $F[x]$ by the substitution $\{x \mapsto t\}$ (roughly, replace every x by t).

Example

Show that $(\exists x.\ \forall y.\ p(x,y)) \rightarrow (\forall x.\ \exists y.\ p(y,x))$ is valid.

Assume otherwise.

1.	$I \not\models (\exists x.\ \forall y.\ p(x,y)) \rightarrow (\forall x.\ \exists y.\ p(y,x))$		assumption
2.	$I \models \exists x.\ \forall y.\ p(x,y)$		1 and $\rightarrow$
3.	$I \not\models \forall x.\ \exists y.\ p(y,x)$		1 and $\rightarrow$
4.	$I \models \forall y.\ p(a,y)$		2, $\exists$ $(x \mapsto a$ fresh$)$
5.	$I \not\models \exists y.\ p(y,b)$		3, $\forall$ $(x \mapsto b$ fresh$)$
6.	$I \models p(a,b)$		4, $\forall$ $(y \mapsto b)$
7.	$I \not\models p(a,b)$		5, $\exists$ $(y \mapsto a)$
8.	$I \models \perp$		6,7 contradictory

Thus, the formula is valid.

Example

Is $F: (\forall x.\ p(x,x)) \rightarrow (\exists x.\ \forall y.\ p(x,y))$ valid?.

Assume I is a falsifying interpretation for F and apply semantic argument:

1.	$I \not\models (\forall x.\ p(x,x)) \rightarrow (\exists x.\ \forall y.\ p(x,y))$	
2.	$I \models \forall x.\ p(x,x)$	1 and $\rightarrow$
3.	$I \not\models \exists x.\ \forall y.\ p(x,y)$	1 and $\rightarrow$
4.	$I \models p(a_1,a_1)$	2, $\forall$
5.	$I \not\models \forall y. p(a_1,y)$	3, $\exists$
6.	$I \not\models p(a_1,a_2)$	5, $\forall$
7.	$I \models p(a_2,a_2)$	2, $\forall$
8.	$I \not\models \forall y. p(a_2,y)$	3, $\exists$
9.	$I \not\models p(a_2,a_3)$	8, $\forall$

$\vdots$

No contradiction. Falsifying interpretation I can be "read" from proof:

$$D_I = \mathbb{N}, \quad p_I(x,y) = \begin{cases} \text{true} & y = x, \\ \text{false} & y = x+1, \\ \text{arbitrary} & \text{otherwise.} \end{cases}$$

Semantic Argument Proof

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

- Soundness
  If every branch of a semantic argument proof reach $I \models \bot$, then F is valid
- Completeness
  Each valid formula F has a semantic argument proof in which every branch reaches $I \models \bot$
- Non-termination
  For an invalid formula F the method is not guaranteed to terminate. Thus, the semantic argument is not a decision procedure for validity.

Soundness (proof sketch)

If for interpretation I the assumption of the proof holds
then there is an interpretation I' and a branch
such that all statements on that branch hold.

I' differs from I in the values $\alpha_I[a_i]$ of fresh constants a_i.

If all branches of the proof end with $I \models \bot$, then the assumption was wrong. Thus, if the assumption was $I \not\models F$, then F must be valid.

Completeness (proof sketch)

Consider (finite or infinite) proof trees starting with $I \not\models F$. We assume that

- all possible proof rules were applied in all non-closed branches.
- the $\forall$ and $\exists$ rules were applied for all terms.
  This is possible since the terms are countable.

If every branch is closed, the tree is finite (Kőnig's Lemma) and we have a finite proof for F.

Completeness (proof sketch, continued)

Otherwise, the proof tree has at least one open branch P. We show that F is not valid.

1. The statements on that branch P form a Hintikka set:
   - $I \models F \wedge G \in P$ implies $I \models F \in P$ and $I \models G \in P$.
   - $I \not\models F \wedge G \in P$ implies $I \not\models F \in P$ or $I \not\models G \in P$.
   - $I \models \forall x.\ F[x] \in P$ implies for all terms t, $I \models F[t] \in P$.
   - $I \not\models \forall x.\ F[x] \in P$ implies for some term a, $I \not\models F[a] \in P$.
   - Similarly for $\vee, \rightarrow, \leftrightarrow, \exists$.
2. Choose $D_I := \{t \mid t \text{ is term}\}$, $\alpha_I[f](t_1, \dots, t_n) = f(t_1, \dots t_n)$, $\alpha_I[x] = x$ (every term is interpreted as itself)

$$\alpha_I[p](t_1, \dots, t_n) = \begin{cases} \text{true} & I \models p(t_1, \dots, t_n) \in P \\ \text{false} & \text{otherwise} \end{cases}$$

3. I satisfies all statements on the branch.
   In particular, I is a falsifying interpretation of F, thus F is not valid.

Normal Forms

Also in first-order logic normal forms can be used:

- Devise an algorithm to convert a formula to a normal form.
- Then devise an algorithm for satisfiability/validity that only works on the normal form.

Negation Normal Forms (NNF)

Negations appear only in literals. (only $\neg, \land, \lor, \exists, \forall$)

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

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

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

$$F_1 \rightarrow F_2 \Leftrightarrow \neg F_1 \lor F_2$$
$$F_1 \leftrightarrow F_2 \Leftrightarrow (F_1 \rightarrow F_2) \land (F_2 \rightarrow F_1)$$

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

Example: Conversion to NNF

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

1. $\forall x.\ (\exists y.\ p(x,y) \wedge p(x,z)) \rightarrow \exists w.\ p(x,w)$
2. $\forall x.\ \neg(\exists y.\ p(x,y) \wedge p(x,z)) \vee \exists w.\ p(x,w)$

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

   $$\neg \exists x.\ F[x] \Leftrightarrow \forall x.\ \neg F[x]$$
4. $\forall x.\ (\forall y.\ \neg p(x,y) \vee \neg p(x,z)) \vee \exists w.\ p(x,w)$

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. $F' \Leftrightarrow F$:

1. Write F in NNF
2. Rename quantified variables to fresh names
3. Move all quantifiers to the front

Example: PNF

Find equivalent PNF of

$$F \ : \ \forall x.\ ((\exists y.\ p(x,y) \land p(x,z)) \rightarrow \exists y.\ p(x,y))$$

- Write F in NNF

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

- Rename quantified variables to fresh names

$$F_2 \ : \ \forall x.\ (\underset{\uparrow}{\forall y}.\ \neg p(x,y) \lor \neg p(x,z)) \lor \exists w.\ p(x,w)$$

$\uparrow$ in the scope of $\forall x$

- Move all quantifiers to the front

$$F_3 \;:\; \forall x.\, \forall y.\, \exists w.\, \neg p(x,y) \lor \neg p(x,z) \lor p(x,w)$$

Alternately,

$$F_3' \;:\; \forall x.\, \exists w.\, \forall y.\, \neg p(x,y) \lor \neg p(x,z) \lor p(x,w)$$

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 $G \quad \not\Leftrightarrow \quad F$

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

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

Similarly for satisfiability

Theories

Theories

In first-order logic function symbols have no predefined meaning:

The formula $1 + 1 = 3$ is satisfiable.

We want to fix the meaning for some function symbols.
Examples:

- Equality theory
- Theory of natural numbers
- Theory of rational numbers
- Theory of arrays or lists

Definition (First-order theory)

A First-order theory T consists of

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

A Σ-formula is a 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

Theory of Equality T_E

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

- $=$, 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)

Axiom Schemata

Congruence and Equivalence are axiom schemata.

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)

For every function symbol there is an instance of the congruence axiom schemata.

Example: Congruence axiom for binary function f_2:
$\forall x_1, x_2, y_1, y_2.\ x_1 = y_1 \wedge x_2 = y_2 \rightarrow f_2(x_1, x_2) = f_2(y_1, y_2)$

$A_{T_{\mathrm{E}}}$ contains an infinite number of these axioms.

T-Validity and T-Satisfiability

Definition (T-interpretation)

An interpretation I is a T-interpretation, if it satisfies all the axioms of T.

Definition (T-valid)

A Σ-formula F is valid in theory T (T-valid, also $T \models F$),
if every T-interpretation satisfies F.

Definition (T-satisfiable)

A Σ-formula F is satisfiable in T (T-satisfiable),
if there is a T-interpretation that satisfies F

Definition (T-equivalent)

Two Σ-formulae F_1 and F_2 are equivalent in T (T-equivalent),
if $F_1 \leftrightarrow F_2$ is T-valid,

Example: T_E-validity

Semantic argument method can be used for T_E

Prove

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

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

Then,

1.	$I \not\models F$	assumption
2.	$I \models a = b \wedge b = c$	1, $\rightarrow$
3.	$I \not\models g(f(a), b) = g(f(c), a)$	1, $\rightarrow$
4.	$I \models \forall x, y, z.\ x = y \wedge y = z \rightarrow x = z$	transitivity
5.	$I \models a = b \wedge b = c \rightarrow a = c$	4, 3 $\times\ \forall\{x \mapsto a, y \mapsto b, z \mapsto c\}$
$6a$	$I \not\models a = b \wedge b = c$	5, $\rightarrow$
$7a$	$I \models \bot$	2 and 6a contradictory
$6b.$	$I \models a = c$	4, 5, (5, $\rightarrow$)
$7b.$	$I \models a = c \rightarrow f(a) = f(c)$	(congruence), 2 $\times\ \forall$
$8ba.$	$I \not\models a = c \quad \cdots I \models \bot$	
$8bb.$	$I \models f(a) = f(c)$	7b, $\rightarrow$
$9bb.$	$I \models a = b$	2, $\wedge$
$10bb.$	$I \models a = b \rightarrow b = a$	(symmetry), 2 $\times\ \forall$
$11bba.$	$I \not\models a = b \quad \cdots I \models \bot$	
$11bbb.$	$I \models b = a$	10bb, $\rightarrow$
$12bbb.$	$I \models f(a) = f(c) \wedge b = a \rightarrow g(f(a), b) = g(f(c), a)$	(congruence), 4 $\times\ \forall$
$\ldots 13$	$I \models g(f(a), b) = g(f(c), a)$	8bb, 11bbb, 12bbb

3 and 13 are contradictory. Thus, F is T_E-valid.

Decidability of T_E

Is it possible to decide T_E-validity?

T_E-validity is undecidable.

If we restrict ourself to quantifier-free formulae we get decidability:

For a quantifier-free formula T_E-validity is decidable.

Fragments of Theories

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

Example: quantifier-free fragment 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.

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 addition
- Theory of integers $T_{\mathbb{Z}}$: integers with $+, -, >$

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

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

Axioms of T_{PA}: axioms of T_E,

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.

Expressiveness of Peano Arithmetic

UNI FREIBURG

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

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

We can define $>$ and $\geq$: $3x + 5 > 2y$ write as $\exists z.\, z \neq 0 \land 3x + 5 = 2y + z$

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

Examples for valid formulae:

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

In Fermat's theorem we used x^n, which is not a valid term in Σ_{PA}.
However, there is the Σ_{PA}-formula $EXP[x, n, r]$ with

1. $EXP[x, 0, r] \leftrightarrow r = 1$
2. $EXP[x, i + 1, r] \leftrightarrow \exists r_1.\ EXP[x, i, r_1] \wedge r = r_1 \cdot x$

$$\begin{aligned}
EXP[x, n, r] :\ & \exists d, m.\ (\exists z.\ d = (m + 1)z + 1) \wedge \\
& (\forall i, r_1.\ i < n \wedge r_1 < m \wedge (\exists z.\ d = ((i + 1)m + 1)z + r_1) \rightarrow \\
& \qquad r_1 x < m \wedge (\exists z.\ d = ((i + 2)m + 1)z + r_1 \cdot x)) \wedge \\
& r < m \wedge (\exists z.\ d = ((n + 1)m + 1)z + r)
\end{aligned}$$

Fermat's theorem can be stated as:

$$\begin{aligned}
\forall n.\ n > 2 \rightarrow \neg\exists x, y, z, rx, ry.\ & x \neq 0 \wedge y \neq 0 \wedge z \neq 0 \wedge \\
& EXP[x, n, rx] \wedge EXP[y, n, ry] \wedge EXP[z, n, rx + ry]
\end{aligned}$$

Decidability of Peano Arithmetic

Gödel showed that for every recursive function $f : \mathbb{N}^n \to \mathbb{N}$ there is a Σ_{PA}-formula $F[x_1, \ldots, x_n, r]$ with

$$F[x_1, \ldots, x_n, r] \leftrightarrow r = f(x_1, \ldots, x_n)$$

T_{PA} is undecidable. (Gödel, Turing, Post, Church)

The quantifier-free fragment of T_{PA} is undecidable. (Matiyasevich, 1970)

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

For decidability: no multiplication

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

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

Axioms of $T_{\mathbb{N}}$: axioms of T_E,

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)

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

Signature:

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

where

- $\ldots, -2, -1, 0,\ 1,\ 2,\ \ldots$ are constants
- $\ldots, -3\cdot, -2\cdot, 2\cdot,\ 3\cdot,\ \ldots$ are unary functions
  (intended meaning: $2 \cdot x$ is $x + x$)
- $+, -, =, >$ have the usual meanings.

Relation between $T_{\mathbb{Z}}$ and $T_{\mathbb{N}}$

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

- For every $\Sigma_{\mathbb{Z}}$-formula there is an equisatisfiable $\Sigma_{\mathbb{N}}$-formula.
- For every $\Sigma_{\mathbb{N}}$-formula there is an equisatisfiable $\Sigma_{\mathbb{Z}}$-formula.

$\Sigma_{\mathbb{Z}}$-formula F and $\Sigma_{\mathbb{N}}$-formula G are equisatisfiable iff:

$$F \text{ is } T_{\mathbb{Z}}\text{-satisfiable} \quad \text{iff} \quad G \text{ is } T_{\mathbb{N}}\text{-satisfiable}$$

Example: The $\Sigma_{\mathbb{N}}$-formula

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

is equisatisfiable to the $\Sigma_{\mathbb{Z}}$-formula:

$$\forall x.\ x > -1 \rightarrow \exists y.\ y > -1 \wedge x = y + 1.$$

Example: $\Sigma_{\mathbb{Z}}$-formula to $\Sigma_{\mathbb{N}}$-formula

Consider the $\Sigma_{\mathbb{Z}}$-formula

$F_0 : \ \forall w, x.\ \exists y, z.\ x + 2y - z - 7 > -3w + 4$

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

$$F_1 : \quad \begin{array}{l} \forall w_p, w_n, x_p, x_n.\ \exists y_p, y_n, z_p, z_n. \\ \quad (x_p - x_n) + 2(y_p - y_n) - (z_p - z_n) - 7 > -3(w_p - w_n) + 4 \end{array}$$

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

$$F_2 : \quad \begin{array}{l} \forall w_p, w_n, x_p, x_n.\ \exists y_p, y_n, z_p, z_n. \\ \quad x_p + 2y_p + z_n + 3w_p > x_n + 2y_n + z_p + 7 + 3w_n + 4 \end{array}$$

Eliminate $>$ and numbers:

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

which is a $\Sigma_{\mathbb{N}}$-formula equisatisfiable to F_0.

Reducing $T_{\mathbb{Z}}$ to $T_{\mathbb{N}}$.

To decide $T_{\mathbb{Z}}$-validity for a $\Sigma_{\mathbb{Z}}$-formula F:

- transform $\neg F$ to an equisatisfiable $\Sigma_{\mathbb{N}}$-formula $\neg G$,
- decide $T_{\mathbb{N}}$-validity of G.

Rationals and Reals

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

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

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

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

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

Note: Strict inequality

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

can be expressed as

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

Theory of Reals $T_{\mathbb{R}}$

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

Axioms of $T_{\mathbb{R}}$: axioms of T_E,

1. $\forall x, y, z.\ (x + y) + z = x + (y + z)$ (+ associativity)
2. $\forall x, y.\ x + y = y + x$ (+ commutativity)
3. $\forall x.\ x + 0 = x$ (+ identity)
4. $\forall x.\ x + (-x) = 0$ (+ inverse)
5. $\forall x, y, z.\ (x \cdot y) \cdot z = x \cdot (y \cdot z)$ (· associativity)
6. $\forall x, y.\ x \cdot y = y \cdot x$ (· commutativity)
7. $\forall x.\ x \cdot 1 = x$ (· identity)
8. $\forall x.\ x \neq 0 \rightarrow \exists y.\ x \cdot y = 1$ (· inverse)
9. $\forall x, y, z.\ x \cdot (y + z) = x \cdot y + x \cdot z$ (distributivity)
10. $0 \neq 1$ (separate identies)
11. $\forall x, y.\ x \geq y \wedge y \geq x \rightarrow x = y$ (antisymmetry)
12. $\forall x, y, z.\ x \geq y \wedge y \geq z \rightarrow x \geq z$ (transitivity)
13. $\forall x, y.\ x \geq y \vee y \geq x$ (totality)
14. $\forall x, y, z.\ x \geq y \rightarrow x + z \geq y + z$ (+ ordered)
15. $\forall x, y.\ x \geq 0 \wedge y \geq 0 \rightarrow x \cdot y \geq 0$ (· ordered)
16. $\forall x.\ \exists y.\ x = y \cdot y \vee x = -y \cdot y$ (square root)
17. for each odd integer n,
    $\forall x_0, \dots, x_{n-1}.\ \exists y.\ y^n + x_{n-1} y^{n-1} \cdots + x_1 y + x_0 = 0$ (at least one root)

Example

F: $\forall a, b, c.\ b^2 - 4ac \geq 0 \leftrightarrow \exists x.\ ax^2 + bx + c = 0$ is $T_{\mathbb{R}}$-valid.
As usual: x^2 abbreviates $x \cdot x$, we omit $\cdot$, e.g. in $4ac$,
4 abbreviate $1 + 1 + 1 + 1$ and $a - b$ abbreviates $a + (-b)$.

1.	$I \not\models F$	assumption
2.	$I \models \exists y.\ bb - 4ac = y^2 \vee bb - 4ac = -y^2$	square root, $\forall$
3.	$I \models d^2 = bb - 4ac \vee d^2 = -(bb - 4ac)$	2, $\exists$
4.	$I \models d \geq 0 \vee 0 \geq d$	$\geq$ total
5.	$I \models d^2 \geq 0$	4, case distinction, $\cdot$ ordered
6.	$I \models 2a \cdot e = 1$	$\cdot$ inverse, $\forall$, $\exists$
7a.	$I \models bb - 4ac \geq 0$	1,$\leftrightarrow$
8a.	$I \not\models \exists x. axx + bx + c = 0$	1,$\leftrightarrow$
9a.	$I \not\models a((-b + d)e)^2 + b(-b + d)e + c = 0$	8a, $\exists$
10a.	$I \not\models ab^2e^2 - 2abde^2 + ad^2e^2$ $-b^2e + bde + c = 0$	distributivity
11a.	$I \models dd = bb - 4ac$	3, 5, 7a
12a.	$I \not\models ab^2e^2 - bde + a(b^2 - 4ac)e^2$ $-b^2e + bde + c = 0$	6, 11a, congruence
13a.	$I \not\models 0 = 0$	3, distributivity, inverse
14a.	$I \models \bot$	13a, reflexivity

Example

F: $\forall a, b, c.\ bb - 4ac \geq 0 \leftrightarrow \exists x.\ axx + bx + c = 0$ is $T_{\mathbb{R}}$-valid.
As usual: x^2 abbreviates $x \cdot x$, we omit $\cdot$, e.g., in $4ac$,
4 abbreviate $1 + 1 + 1 + 1$ and $a - b$ abbreviates $a + (-b)$.

1.	$I \not\models F$		assumption
2.	$I \models \exists y.\ bb - 4ac = y^2 \vee bb - 4ac = -y^2$		square root, $\forall$
3.	$I \models d^2 = bb - 4ac \vee d^2 = -(bb - 4ac)$		2, $\exists$
4.	$I \models d \geq 0 \vee 0 \geq d$		$\geq$ total
5.	$I \models d^2 \geq 0$		4, case distinction, $\cdot$ ordered
6.	$I \models 2a \cdot e = 1$		$\cdot$ inverse, $\forall$,$\exists$
$7b$.	$I \not\models bb - 4ac \geq 0$		1,$\leftrightarrow$
$8b$.	$I \models \exists x. axx + bx + c = 0$		1,$\leftrightarrow$
$9b$.	$I \models aff + bf + c = 0$		8b,$\exists$
$10b$.	$I \models (2af + b)^2 = bb - 4ac$		field axioms, T_E
$11b$.	$I \models (2af + b)^2 \geq 0$		analogous to 5
$12b$.	$I \models bb - 4ac \geq 0$		10b, 11b, equivalence
$13b$.	$I \models \bot$		12b, 7b

Decidability of $T_{\mathbb{R}}$

$T_{\mathbb{R}}$ is decidable (Tarski, 1930)
High time complexity: $O(2^{2^{kn}})$

Theory of Rationals $T_{\mathbb{Q}}$

Signature: $\Sigma_{\mathbb{Q}}$: $\{0,\ 1,\ +,\ -,\ =,\ \geq\}$ no multiplication!

Axioms of $T_{\mathbb{Q}}$: axioms of T_E,

1. $\forall x, y, z.\ (x + y) + z = x + (y + z)$ (+ associativity)
2. $\forall x, y.\ x + y = y + x$ (+ commutativity)
3. $\forall x.\ x + 0 = x$ (+ identity)
4. $\forall x.\ x + (-x) = 0$ (+ inverse)
5. $1 \geq 0 \wedge 1 \neq 0$ (one)
6. $\forall x, y.\ x \geq y \wedge y \geq x \rightarrow x = y$ (antisymmetry)
7. $\forall x, y, z.\ x \geq y \wedge y \geq z \rightarrow x \geq z$ (transitivity)
8. $\forall x, y.\ x \geq y \vee y \geq x$ (totality)
9. $\forall x, y, z.\ x \geq y \rightarrow x + z \geq y + z$ (+ ordered)
10. For every positive integer n:
    $\forall x.\ \exists y.\ x = \underbrace{y + \cdots + y}_{n}$ (divisible)

Expressiveness and Decidability of $T_{\mathbb{Q}}$

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

$$x + x + x + y + y + y + y \geq \underbrace{1 + 1 + \cdots + 1}_{24}$$

$T_{\mathbb{Q}}$ is decidable
Efficient algorithm for quantifier free fragment

Recursive Data Structures (RDS)

- Data Structures are tuples of variables.
  Like `struct` in C, `record` in Pascal.
- In Recursive Data Structures, one of the tuple elements can be the data structure again.
  Linked lists or trees.

RDS theory of LISP-like lists, T_{cons}

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

where

- $\text{cons}(a, b)$ – list constructed by adding a in front of list b
- $\text{car}(x)$ – left projector of x: $\text{car}(\text{cons}(a, b)) = a$
- $\text{cdr}(x)$ – right projector of x: $\text{cdr}(\text{cons}(a, b)) = b$
- $\text{atom}(x)$ – true iff x is a single-element list

Axioms: The axioms of A_{T_E} plus

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

Axioms of Theory of Lists T_{cons}

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

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

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)

Decidability of T_{cons}

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

Example: T_{cons}-Validity

We argue that the following Σ_{cons}-formula F is T_{cons}-valid:

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

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

Lines 6 and 10 are contradictory. Therefore, F is T_{cons}-valid.

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)

Equality in T_A

Note: $=$ is only defined for array elements

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

not T_A-valid, but

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

is T_A-valid.

Also

$$a = b \rightarrow a[i] = b[i]$$

is not T_A-valid: We only axiomatized a restricted congruence.

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

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

How do we show that

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

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

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

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 requirements are met
then satisfiability of qff of $T_1 \cup T_2$ is decidable.

Lists with equality $T_{\text{cons}}^{=}$

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

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

(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: $T_{\text{cons}}^{=}$-Validity

We argue that the following $\Sigma_{\text{cons}}^{=}$-formula F is $T_{\text{cons}}^{=}$-valid:

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

1.	$I \not\models F$	assumption
2.	$I \models \text{car}(a) = \text{car}(b)$	1, $\rightarrow$, $\land$
3.	$I \models \text{cdr}(a) = \text{cdr}(b)$	1, $\rightarrow$, $\land$
4.	$I \models \neg\text{atom}(a)$	1, $\rightarrow$, $\land$
5.	$I \models \neg\text{atom}(b)$	1, $\rightarrow$, $\land$
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. Therefore, F is $T_{\text{cons}}^{=}$-valid.

First-Order Theories

	Theory	Decidable	QFF Dec.
T_E	Equality	–	✓
T_{PA}	Peano Arithmetic	–	–
$T_{\mathbb{N}}$	Presburger Arithmetic	✓	✓
$T_{\mathbb{Z}}$	Linear Integer Arithmetic	✓	✓
$T_{\mathbb{R}}$	Real Arithmetic	✓	✓
$T_{\mathbb{Q}}$	Linear Rationals	✓	✓
T_{cons}	Lists	–	✓
$T_{\mathrm{cons}}^{=}$	Lists with Equality	–	✓
T_{A}	Arrays	–	✓
$T_{\mathrm{A}}^{=}$	Arrays with Extensionality	–	✓

Quantifier Elimination

Quantifier Elimination

Quantifier Elimination (QE) removes quantifiers from formulae:

- Given a formula with quantifiers, e.g., $\exists x.F[x, y, z]$.
- Goal: find an equivalent quantifier-free formula $G[y, z]$.
- The free variables of F and G are the same.

$$\exists x.F[x, y, z] \Leftrightarrow G[y, z]$$

QE as Decision Procedure

Decide satisfiabilty for a formula F, e.g. in $T_{\mathbb{Q}}$, using quantifier elimination:

- Given a formula F, with free variable $x_1, \ldots, x_n$.
- Build $\exists x_1 \ldots \exists x_n . F$.
- Build equivalent quantifier free formula G.
  G contains only constants, functions and predicates
  i.e. $0, 1, +, -, \geq, =$.
- Compute truth value of G.

QE algorithm

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

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

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

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

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

$$G_5\colon\ F_4$$

G_5 is quantifier-free and T-equivalent to G_1

Syntactic sugar for Rationals

Consider the Signature of Rationals: $\Sigma_{\mathbb{Q}} : \{0, 1, +, -, =, \geq\}$

We extend the signature with the predicate $>$, which is defined as

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

Additionally we allow predicates $<$ and $\leq$:

$$x < y :\Leftrightarrow y > x \qquad x \leq y :\Leftrightarrow y \geq x.$$

We extend the signature by fractions:

$$\frac{\cdot}{a} \in \Sigma_{\mathbb{Q}} \text{ for } a \in \mathbb{Z}^{+}$$

which are unary function symbols, with their usual meaning.

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 $\exists x.\ F[x]$.

1. Put $F[x]$ in NNF.
2. Eliminate negated literals.
3. Solve the literals s.t. x appears isolated on one side.
4. Finite disjunction $\bigvee_{t \in S_F} F[t]$.

$$\exists x. F[x] \Leftrightarrow \bigvee_{t \in S_F} F[t].$$

where S_F depends on the formula F.

Step 1 and 2

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

Step 2: Eliminate negated literals and $\geq$ (left to right)

$$\begin{array}{rcl}
s \geq t & \Leftrightarrow & s > t \lor s = t \\
\neg(s > t) & \Leftrightarrow & t > s \lor t = s \\
\neg(s \geq t) & \Leftrightarrow & t > s \\
\neg(s = t) & \Leftrightarrow & t < s \lor t > s
\end{array}$$

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.,

$$ax + t_2 < bx + t_1 \qquad \Rightarrow \qquad x < \frac{t_1 - t_2}{a - b}$$

where $a - b \in \mathbb{Z}^+$.

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

- (A) $x < t$
- (B) $t < x$
- (C) $x = t$

where t is a term that does not contain x.

Step 4 (Part 1)

Construct from $F_3[x]$

- left infinite projection $F_3[-\infty]$ by replacing
  - (A) atoms $x < t$ by $\top$
  - (B) atoms $t < x$ by $\bot$
  - (C) atoms $x = t$ by $\bot$
- right infinite projection $F_3[+\infty]$ by replacing
  - (A) atoms $x < t$ by $\bot$
  - (B) atoms $t < x$ by $\top$
  - (C) atoms $x = t$ by $\bot$

Step 4 (Part 2)

Let S be the set of terms t from (A), (B), (C) atoms.
Construct the formula

$$F_4 : \bigvee_{t \in S_F} F_3[t], \quad \text{where } S_F := \{-\infty, \infty\} \cup \left\{ \frac{s+t}{2} \;\middle|\; s, t \in S \right\}$$

which is $T_{\mathbb{Q}}$-equivalent to $\exists x.\ F[x]$.

- $F_3[-\infty]$ captures the case when small $x \in \mathbb{Q}$ satisfy $F_3[x]$
- $F_3[-\infty]$ captures the case when large $x \in \mathbb{Q}$ satisfy $F_3[x]$
- if $s \equiv t$, $\frac{s+t}{2} = s$ captures the case when $s \in S$ satisfies $F_3[s]$
  if $s < t$ are adjacent numbers, $\frac{s+t}{2}$ represents the whole interval (s, t).

Four cases are possible:

1. All numbers x smaller than the smallest term satisfy $F[x]$.

$$\longleftarrow)t_1\ \ t_2 \cdots t_n$$

2. All numbers x larger than the largest term satisfy $F[x]$.

$$t_1\ \ t_2 \cdots t_n(\longrightarrow$$

3. Some t_i,satisfies $F[x]$.

$$\begin{array}{c} t_1 \quad \cdots t_i \cdots \quad t_n \\ \uparrow \end{array}$$

4. On an open interval between two terms every element satisfies $F[x]$.

$$t_1 \cdots \quad t_i(\underset{\frac{t_i+t_{i+1}}{2}}{\longleftrightarrow})t_{i+1} \quad \cdots t_n$$

Correctness of Step 4

Theorem

Let S_F be the set of terms constructed from $F_3[x]$ as in Step 4. Then $\exists x.\ F_3[x] \Leftrightarrow \bigvee_{t \in S_F} F_3[t]$.

Proof of Theorem

$\Leftarrow$ If $\bigvee_{t \in S_F} F_3[t]$ is true, then $F_3[t]$ for some $t \in S_F$ is true.
If $F_3[\frac{s+t}{2}]$ is true, then obviously $\exists x.\ F_3[x]$ is true.
If $F_3[-\infty]$ is true, choose some $x < t$ for all $t \in S$. Then $F_3[x]$ is true.
If $F_3[\infty]$ is true, choose some $x > t$ for all $t \in S$. Then $F_3[x]$ is true.

Correctness of Step 4

$\Rightarrow$ If $I \models \exists x.\ F_3[x]$ then there is value v such that

$$I \lhd \{x \mapsto \mathrm{v}\} \models F_3\,.$$

If $\mathrm{v} < \alpha_I[t]$ for all $t \in S$, then $I \models F_3[-\infty]$.
If $\mathrm{v} > \alpha_I[t]$ for all $t \in S$, then $I \models F_3[\infty]$.
If $\mathrm{v} = \alpha_I[t]$ for some $t \in S$, then $I \models F[\frac{t+t}{2}]$.

Otherwise choose largest $s \in S$ with $\alpha_I[s] < \mathrm{v}$ and smallest $t \in S$ with $\alpha_I[t] > \mathrm{v}$.
Since no atom of F_3 can distinguish between values in interval (s, t), $F_3[\mathrm{v}] \Leftrightarrow F_3[\frac{s+t}{2}]$. Hence, $I \models F[\frac{s+t}{2}]$.

In all cases $I \models \bigvee_{t \in S_F} F_3[t]$.

Example

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

Solving for x

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

Step 4:

$$F_4 : \bigvee_{t \in S_F} \underbrace{\left(t < 3 \land t > \frac{13}{7}\right)}_{F_3[t]}$$

Example contd.

$$S_F = \{-\infty, +\infty, 3, \frac{13}{7}, \frac{3 + \frac{13}{7}}{2}\}.$$

$$F_3[x] = x < 3 \wedge x > 13/7$$

$$F_{-\infty} \Leftrightarrow \top \wedge \bot \Leftrightarrow \bot \qquad F_{+\infty} \Leftrightarrow \bot \wedge \top \Leftrightarrow \bot$$

$$F_3\,[3] \bot \wedge \top \Leftrightarrow \bot \qquad F_3\left[\frac{13}{7}\right] \Leftrightarrow \top \wedge \bot \Leftrightarrow \bot$$

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

Thus, $F_4 : \bigvee_{t \in S_F} F_3[t] \Leftrightarrow \top$ is $T_{\mathbb{Q}}$-equivalent to $\exists x.\ F[x]$, so $\exists x.\ F[x]$ is $T_{\mathbb{Q}}$-valid.

Example

$$\exists x.\ \underbrace{2x > y \land 3x < z}_{F[x]}$$

Solving for x

$$\exists x.\ \underbrace{x > \frac{y}{2} \land x < \frac{z}{3}}_{F_3[x]}$$

Step 4: $F_{-\infty} \Leftrightarrow \bot$, $F_{+\infty} \Leftrightarrow \bot$, $F_3[\frac{y}{2}] \Leftrightarrow \bot$ and $F_3[\frac{z}{3}] \Leftrightarrow \bot$.

$$F_4 \ :\ \frac{\frac{y}{2} + \frac{z}{3}}{2} > \frac{y}{2} \land \frac{\frac{y}{2} + \frac{z}{3}}{2} < \frac{z}{3}$$

which simplifies to:

$$F_4 \ :\ 2z > 3y$$

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

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

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

Which quantifier free formula $G[y]$ is equivalent to F?

There is no such formula!

No QE for $T_{\mathbb{Z}}$

Lemma

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

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

Either $\mathbb{Z}^+ \cap S_F$ or $\mathbb{Z}^+ \setminus S_F$ is finite.
where $\mathbb{Z}^+$ is the set of positive integers

Proof (Structural Induction over F)

Base case: F is an atomic formula:
$\top, \bot, t_1 = t_2, a \cdot y = t, t_1 < t_2, a \cdot y < t$.

- $\mathbb{Z}^+ \setminus S_\top = \mathbb{Z}^+ \cap S_\bot = \emptyset$ is finite
- $S_{t_1 = t_2}$ and $S_{t_1 < t_2}$ are either $S_\top$ or $S_\bot$.
- $\mathbb{Z}^+ \cap S_{a \cdot y = t}$, $(a \neq 0)$ has at most one element.
- $\mathbb{Z}^+ \cap S_{a \cdot y < t}$, $a > 0$ is finite.
- $\mathbb{Z}^+ \setminus S_{a \cdot y < t}$, $a < 0$ is finite.

No QE for $T_{\mathbb{Z}}$

Lemma

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

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

Either $\mathbb{Z}^+ \cap S_F$ or $\mathbb{Z}^+ \setminus S_F$ is finite.
where $\mathbb{Z}^+$ is the set of positive integers

Proof (Structural Induction over F)

Induction step: Assume property holds for F, G. Show it for $\neg F, F \wedge G, F \vee G, F \rightarrow G, F \leftrightarrow G$.

- $\neg F$: We have $\mathbb{Z}^+ \cap S_{\neg F} = \mathbb{Z}^+ \setminus S$ and $\mathbb{Z}^+ \setminus S_{\neg F} = \mathbb{Z}^+ \cap S$ and by ind.-hyp one of these sets is finite.
- $F \wedge G$: We have $\mathbb{Z}^+ \cap S_{F \wedge G} = (\mathbb{Z}^+ \cap S_F) \cap (\mathbb{Z}^+ \cap S_G)$ and $\mathbb{Z}^+ \setminus S_{F \wedge G} = (\mathbb{Z}^+ \setminus S_F) \cup (\mathbb{Z}^+ \setminus S_G)$.
  If the latter set is not finite then one of $\mathbb{Z}^+ \cap S_F$ or $\mathbb{Z}^+ \cap S_G$ is finite.
  In both cases $\mathbb{Z}^+ \cap S_{F \wedge G}$ is finite.

No QE for $T_{\mathbb{Z}}$

Lemma

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

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

Either $\mathbb{Z}^+ \cap S_F$ or $\mathbb{Z}^+ \setminus S_F$ is finite.
where $\mathbb{Z}^+$ is the set of positive integers

Proof (Structural Induction over F)

Induction step: Assume property holds for F, G. Show it for $\neg F, F \wedge G, F \vee G, F \rightarrow G, F \leftrightarrow G$.

- $F \vee G$ follows from previous, since $S_{F \vee G} = S_{\neg(\neg F \wedge \neg G)}$.
- $F \rightarrow G$ follows from $S_{F \rightarrow G} = S_{(\neg F \vee G)}$.
- $F \leftrightarrow G$ follows from $S_{F \leftrightarrow G} = S_{(F \rightarrow G) \wedge (G \rightarrow F)}$.

No QE for $T_{\mathbb{Z}}$

Lemma

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

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

Either $\mathbb{Z}^+ \cap S_F$ or $\mathbb{Z}^+ \setminus S_F$ is finite.
where $\mathbb{Z}^+$ is the set of positive integers

$\Sigma_{\mathbb{Z}}$-formula $\quad F \; : \; \exists x.\; 2x = y$ (with quantifier)

S_F: even integers

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

$$\Sigma_{\mathbb{Z}} \cup \{1|\cdot, 2|\cdot, 3|\cdot, \dots\}$$

Intended interpretations:

$k \mid x$ holds iff k divides x without any remainder

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

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

Example:

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

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

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

is not satisfiable.

$\widehat{T_{\mathbb{Z}}}$ 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 $\exists x.\ F[x]$.

1. Put F[x] into Negation Normal Form (NNF).
2. Normalize literals: $s < t$, $k|t$, or $\neg(k|t)$.
3. Put x in $s < t$ on one side: $hx < t$ or $s < hx$.
4. Replace hx with x' without a factor.
5. Replace $F[x']$ by $\bigvee F[j]$ for finitely many j.

Cooper's Method: 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 $\exists x.\ F[x]$

Example:

$$\exists x.\ \neg(x - 6 < z - x \wedge 4 \mid 5x + 1 \rightarrow 3x < y)$$

is equivalent to

$$\exists x.\ \neg(3x < y) \wedge x - 6 < z - x \wedge 4 \mid 5x + 1$$

Cooper's Method: 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:

$\exists x.\ \neg(3x < y) \wedge x - 6 < z - x \wedge 4 \mid 5x + 1$

is equivalent to

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

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:

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

is equivalent to

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

Cooper's Method: 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:

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

The result $\exists x.\ F_3'[x]$, in which all occurrences of x in $F_3'[x]$ are in terms δx.

Replace δx terms in F_3' with a fresh variable x' to form

$$F_3'' \;:\; F_3\{\delta x \mapsto x'\}$$

Finally, construct

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

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

(A) $x' < t$

(B) $t < x'$

(C) $k \mid x' + t$

(D) $\neg(k \mid x' + t)$

where t is a term that does not contain x, and $k \in \mathbb{Z}^+$.

Example: $\widehat{T_{\mathbb{Z}}}$-formula

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

Collecting coefficients of x:

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

Multiply when necessary

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

Replacing $30x$ with fresh x'

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

$\exists x'.\ F_4[x']$ is equivalent to $\exists x.\ F_3[x]$

Cooper's Method: Result of Step 4

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

(A) $x' < t$

(B) $t < x'$

(C) $k \mid x' + t$

(D) $\neg(k \mid x' + t)$

where t is a term that does not contain x, and $k \in \mathbb{Z}^+$.

Cooper's Method: Step 5

Construct

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

of $F_4[x']$ by

(A) replacing literals $x' < t$ by $\top$

(B) replacing literals $t < x'$ by $\bot$

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

Let

$$\delta = \text{lcm}\left\{ \begin{array}{l} k \text{ of (C) literals } k \mid x' + t \\ k \text{ of (D) literals } \neg(k \mid x' + t) \end{array} \right\}$$

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

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

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

Cooper's Method: Step 5 (Example)

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

Compute lcm: $\delta = lcm(24, 30) = 120$
Then

$$\begin{aligned} F_5 = & \bigvee_{j=1}^{120} \top \wedge \bot \wedge 24 \mid j + 6 \wedge 30 \mid j \\ & \vee \bigvee_{j=1}^{120} 10y - 10 + j < 15z + 90 \wedge 10y - 10 < 10y - 10 + j \\ & \qquad \wedge 24 \mid 10y - 10 + j + 6 \wedge 30 \mid 10y - 10 + j \end{aligned}$$

The formula can be simplified to:

$$F_5 = \bigvee_{j=1}^{120} 10y - 10 + j < 15z + 90 \wedge 24 \mid 10y - 10 + j + 6 \wedge 30 \mid 10y - 10 + j$$

Correctness of Step 5

Theorem

Let F_5 be the formula constructed from $\exists x'.\ F_4[x']$ as in Step 5. Then $\exists x'.\ F_4[x'] \Leftrightarrow F_5$.

Lemma[Periodicity]: For all atoms $k \mid x' + t$ in F_4, we have $k \mid \delta$.
Therefore, $k \mid x' + t$ iff $k \mid x' + \lambda\delta + t$ for all $\lambda \in \mathbb{Z}$.

Proof of Theorem

$\Leftarrow$ If F_5 is true, there are two cases: $F_{-\infty}[j]$ is true or $F_4[t+j]$ is true. If $F_4[t+j]$ is true, than obviously $\exists x'.\ F_4[x']$ is true. If $F_{-\infty}[j]$ is true, then (due to periodicity) $F_{-\infty}[j + \lambda \cdot \delta]$ is true.
If $\lambda < t - 1$ for all $t \in A \cup B$, then
$j + \lambda \cdot \delta < \delta + (t-1)\delta = \delta t \leq t$. Thus,

$$F_{-\infty}[j + \lambda \cdot \delta] \Leftrightarrow F_4[j + \lambda \cdot \delta] \Rightarrow \exists x'.\ F_4[x'] .$$

Correctness of Step 5

$\Rightarrow$ Assume for some x', $F_4[x']$ is true. If $\neg(t < x')$ for all $t \in B$, then choose $j_{x'} \in \{1, \dots, \delta\}$ such that $\delta \mid (j_{x'} - x')$. $j_{x'}$ will satisfy all (C) and (D) literals that x' satisfies. x' does not satisfy any (B) literal. Therefore if $F_4[x']$ is true, $F_{-\infty}[j]$ must be true. Therefore F_5 is true.
If $t < x'$ for some $t \in B$, then let

$$t_{x'} = \max\{t \in B | t < x'\}$$

and choose $j_{x'} \in \{1, \dots, \delta\}$ such that $\delta \,|(t_{x'} + j_{x'} - x')$. We claim that $F_4[t_{x'} + j_{x'}]$ is true.
Since $x' = t_{x'} + j_{x'} + \lambda\delta$, x' and $t_{x'} + j_{x'}$ satisfy the same (C) and (D) literals (due to periodicity).
Since $t_{x'} + j_{x'} > t_{x'} = \max\{t \in B | t < x'\}$, $t_{x'} + j_{x'}$ satisfies all (B) literals that are satisfied by x'.
Since $t_{x'} < x' = t_{x'} + j_{x'} + \lambda\delta \leq t_{x'} + (\lambda + 1)\delta$, we conclude that $\lambda \geq 0$. Hence, $x' \geq t_{x'} + j_{x'}$ and $t_{x'} + j_{x'}$ satisfies all (A) literals satisfied by x'.
Thus $F_4[t_x + j_x']$ is true. Therefore, F_5 is true.

Cooper's Method: Step 5

Construct

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

of $F_4[x']$ by

(A) replacing literals $x' < t$ by $\top$

(B) replacing literals $t < x'$ by $\bot$

Let

$$\delta = \text{lcm}\left\{ \begin{array}{l} k \text{ of (C) literals } k \mid x' + t \\ k \text{ of (D) literals } \neg(k \mid x' + t) \end{array} \right\}$$

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

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

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

Symmetric Elimination

In step 5, if there are fewer

(A) literals $x' < t$

than

(B) literals $t < x'$.

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

each (A) literal $x' < t$ by $\perp$

and

each (B) literal $t < x'$ by $\top$.

Then right elimination.

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

Symmetric Elimination (Example)

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

Compute lcm: $\delta = lcm(24, 30) = 120$
Then

$$\begin{aligned} F_5 = & \bigvee_{j=1}^{120} \bot \land \top \land 24 \mid -j + 6 \land 30 \mid -j \\ & \lor \bigvee_{j=1}^{120} 15z + 90 - j < 15z + 90 \land 10y - 10 < 15z + 90 - j \\ & \qquad \land 24 \mid 15z + 90 - j + 6 \land 30 \mid 15z + 90 - j \end{aligned}$$

The formula can be simplified to:

$$F_5 = \bigvee_{j=1}^{120} 10y - 10 < 15z + 90 - j \land 24 \mid 15z + 90 - j + 6 \land 30 \mid 15z + 90 - j$$

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

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

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,

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

so

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

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.

Quantifier elimination decides validity/satisfiable quantified formulae.

Can also be used for quantifier free formulae:
To decide satisfiability of $F[x_1, \ldots, x_n]$,
apply QE on $\exists x_1, \ldots, x_n.\ F[x_1, \ldots, x_n]$.

But high complexity (doubly exponential for $T_{\mathbb{Q}}$).

Therefore, we are looking for a fast procedure.

Quantifier-free Theory of Equality

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 $p, q, r, \ldots$

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,

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

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

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

Congruence Closure Algorithm

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

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

$$\sim \models s_1 = t_1 \land \cdots \land s_m = t_m \; .$$

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?

Begin with the finest congruence relation $\sim_0$:

$$\{\{s\} \;:\; s \in S_F\}\;.$$

Each term of S_F is only congruent to itself.

Then, for each $i \in \{1, \ldots, m\}$, impose $s_i = t_i$ by merging

$$[s_i]_{\sim_{i-1}} \quad \text{and} \quad [t_i]_{\sim_{i-1}}$$

to form a new congruence relation $\sim_i$. To accomplish this merging,

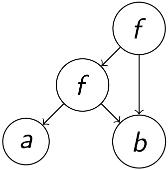
- form the union of $[s_i]_{\sim_{i-1}}$ and $[t_i]_{\sim_{i-1}}$
- propagate any new congruences that arise within this union.

The new relation $\sim_i$ is a congruence relation in which $s_i \sim t_i$.

Ingredients of Algorithm

Efficient data structure for computing the congruence closure.

- Directed Acyclic Graph (DAG) to represent terms.

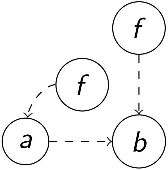


$f(f(a,b),b)$

$f(a,b)$

a b

- Union-Find data structure to represent equivalence classes:

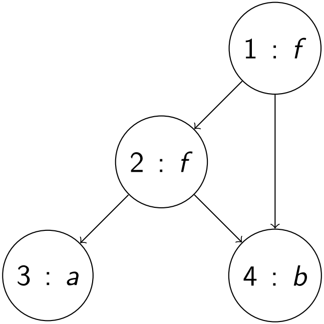


Directed Acyclic Graph (DAG)

For every subterm of the Σ_E-formula F, create

- a node labelled with the function symbols.
- and edges to the argument nodes.

If two subterms are equal, only one node is created.

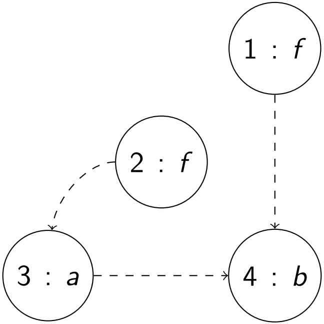


$f(f(a, b), b)$

$f(a, b)$

$a \qquad b$

Equivalence classes are connected by a tree structure, with arrows pointing to the root node.

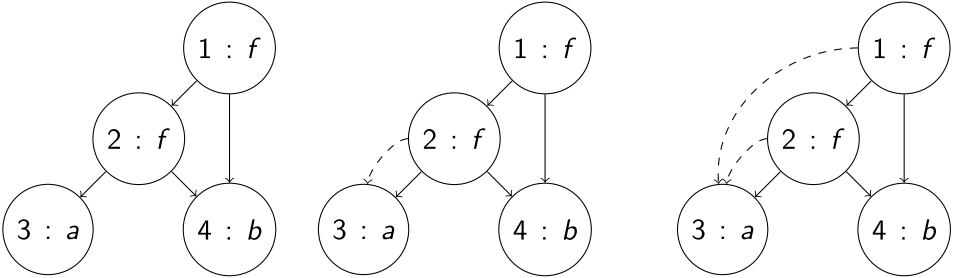


Two operations are defined:

- FIND: Find the representative of an equivalence class by following the edges. $O(\log n)$
- UNION: Merge two classes by connecting the representatives. $O(1)$

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

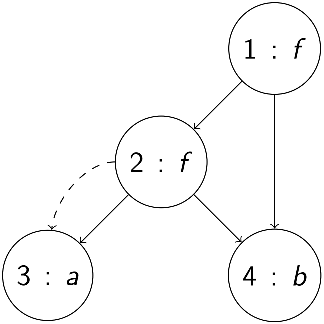
$f(a,b) \sim a, \ b \sim b \ \Rightarrow$
$f(f(a,b),b) \sim f(a,b)$
MERGE $f(f(a,b),b)$
$f(a,b)$

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

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 edge to the representative
    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
}
```

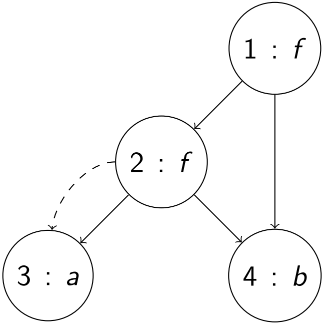
$$
\begin{array}{llll}
\texttt{type}\ \textbf{node} & = & \{ & \\
\quad \texttt{id} & : & \textbf{id} & \ldots 2 \\
\quad \texttt{fn} & : & \textbf{string} & \ldots f \\
\quad \texttt{args} & : & \textbf{idlist} & \ldots [3,4] \\
\quad \texttt{mutable find} & : & \textbf{id} & \ldots 3 \\
\quad \texttt{mutable ccpar} & : & \textbf{idset} & \ldots \emptyset \\
\quad \} & & &
\end{array}
$$



type **node** = {

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

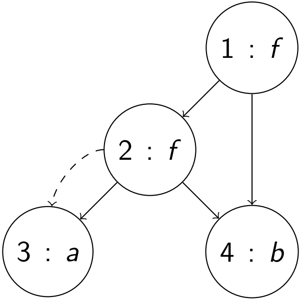
}



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: $\text{FIND}\ 2 = \text{FIND}\ 3 = 3$

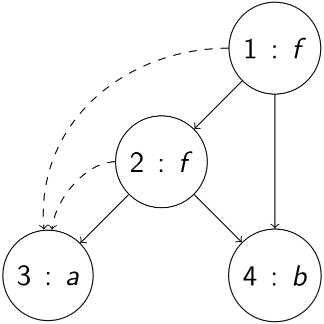
3 is the representative of 2.

UNION function

$$
\begin{array}{l}
\texttt{let}\ \textsc{union}\ i_1\ i_2\ = \\
\quad \texttt{let}\ n_1\ =\ \textsc{node}\ (\textsc{find}\ i_1)\ \texttt{in} \\
\quad \texttt{let}\ n_2\ =\ \textsc{node}\ (\textsc{find}\ i_2)\ \texttt{in} \\
\quad n_1.\texttt{find}\ \leftarrow\ n_2.\texttt{find}; \\
\quad n_2.\texttt{ccpar}\ \leftarrow\ n_1.\texttt{ccpar}\ \cup\ n_2.\texttt{ccpar}; \\
\quad n_1.\texttt{ccpar}\ \leftarrow\ \emptyset
\end{array}
$$

n_2 is the representative of the union class

Example



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

$\quad$ 1.find $\leftarrow 3$

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

$\quad$ 1.ccpar $\leftarrow \emptyset$

The Implementation: CONGRUENT

CCPAR function

Returns parents of all nodes in i's congruence class

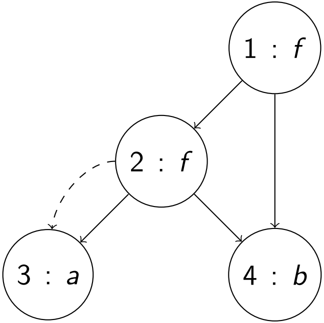
$$\begin{aligned}&\texttt{let}\ \text{CCPAR}\ i\ =\\ &\quad(\text{NODE}\ (\text{FIND}\ i)).\texttt{ccpar}\end{aligned}$$

CONGRUENT predicate

Test whether i_1 and i_2 are congruent

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

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.

UNI FREIBURG

MERGE function

$$
\begin{array}{l}
\texttt{let rec } \textsc{merge}\ i_1\ i_2 = \\
\quad \texttt{if } \textsc{find}\ i_1 \neq \textsc{find}\ i_2 \texttt{ then begin} \\
\quad\quad \texttt{let } P_{i_1} = \textsc{ccpar}\ i_1 \texttt{ in} \\
\quad\quad \texttt{let } P_{i_2} = \textsc{ccpar}\ i_2 \texttt{ in} \\
\quad\quad \textsc{union}\ i_1\ i_2; \\
\quad\quad \texttt{foreach } t_1, t_2 \in P_{i_1} \times P_{i_2} \texttt{ do} \\
\quad\quad\quad \texttt{if } \textsc{find}\ t_1 \neq \textsc{find}\ t_2 \wedge \textsc{congruent}\ t_1\ t_2 \\
\quad\quad\quad \texttt{then } \textsc{merge}\ t_1\ t_2 \\
\quad\quad \texttt{done} \\
\quad \texttt{end}
\end{array}
$$

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

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

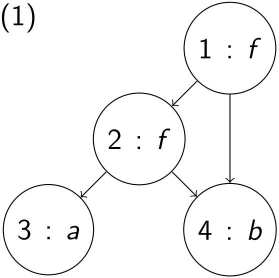
with subterm set S_F, perform the following steps:

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

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

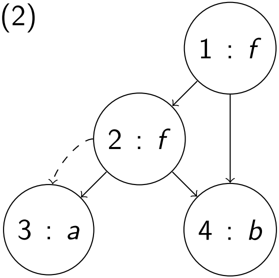
$$f(a,b) = a \;\wedge\; f(f(a,b),b) \neq a$$

(1)



Initial DAG

(2)



MERGE 2 3

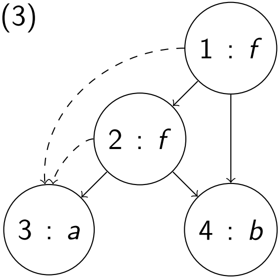
UNION 2 3

$P_2 = \{1\}$

$P_3 = \{2\}$

CONGRUENT 1 2

(3)



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

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

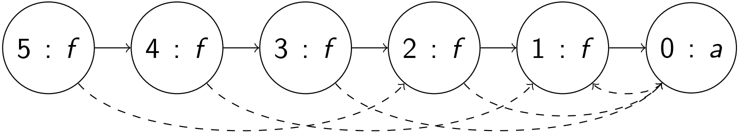
Does

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

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

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



Initial DAG

$f(f(f(a))) = a \;\Rightarrow\;$ MERGE 3 0 $\quad P_3 = \{4\} \;\; P_0 = \{1\}$
$\Rightarrow\;$ MERGE 4 1 $\quad P_4 = \{5\} \;\; P_1 = \{2\}$
$\Rightarrow\;$ MERGE 5 2 $\quad P_5 = \{\} \;\; P_2 = \{3\}$

$f(f(f(f(f(a))))) = a \;\Rightarrow\;$ MERGE 5 0 $\quad P_5 = \{3\} \;\; P_0 = \{1,4\}$
$\Rightarrow\;$ MERGE 3 1 $\quad P_3 = \{1,3,4\}, P_1 = \{2,5\}$

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.

Proof:

$\Rightarrow$ Let I be a satisfying interpretation.
By induction over the steps of the algorithm one can prove:
Whenever the algorithm merges nodes t_1 and t_2, $I \models t_1 = t_2$ holds.

Since $I \models s_i \neq t_i$ for $i \in \{m+1, \ldots, n\}$ they cannot be merged.

Hence the algorithm returns satisfiable.

Proof:

$\Leftarrow$ Let S denote the nodes of the graph and
Let $[t] := \{t' \mid t \sim t'\}$ denote the congruence class of t and
$S/\sim \;:= \{[t] \mid t \in S\}$ denote the set of congruence classes.
Show that there is an interpretation I:

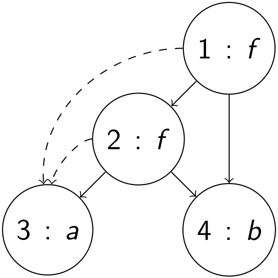
$$D_I = S/\sim \cup \{\Omega\}$$

$$\alpha_I[f](v_1, \dots, v_n) = \begin{cases} [f(t_1, \dots, t_n)] & v_1 = [t_1], \dots, v_n = [t_n], \\ & f(t_1, \dots, t_n) \in S \\ \Omega & \text{otherwise} \end{cases}$$

$$\alpha_I[=](v_1, v_2) = \top \text{ iff } v_1 = v_2$$

I is well-defined!
$\alpha_I[=]$ is a congruence relation,
$I \models F$.

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



$S = \{f(f(a,b),b), f(a,b), a, b\}$
$S/{\sim} = \{\{f(f(a,b),b), f(a,b), a\}, \{b\}\} = \{[a],[b]\}$
$D_I = \{[a],[b],\Omega\}$

$\alpha_I[f]$	$[a]$	$[b]$	Ω
$[a]$	Ω	$[a]$	Ω
$[b]$	Ω	Ω	Ω
Ω	Ω	Ω	Ω

$\alpha_I[=]$	$[a]$	$[b]$	Ω
$[a]$	$\top$	$\bot$	$\bot$
$[b]$	$\bot$	$\top$	$\bot$
Ω	$\bot$	$\bot$	$\top$

How to handle predicates?

We can get rid of predicates by

- Introduce fresh constant $\bullet$ corresponding to $\top$.
- Introduce a fresh function f_p for each predicate p.
- Replace $p(t_1, \ldots, t_n)$ with $f_p(t_1, \ldots, t_n) = \bullet$.

Compare the equivalence axiom for p
with the congruence axiom for f_p.

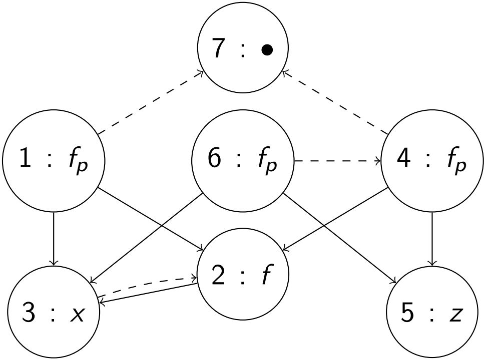
- $\forall x_1, x_2, y_1, y_2.\ x_1 = y_1 \land x_2 = y_2 \rightarrow p(x_1, x_2) \leftrightarrow p(y_1, y_2)$
- $\forall x_1, x_2, y_1, y_2.\ x_1 = y_1 \land x_2 = y_2 \rightarrow f_p(x_1, x_2) = f_p(y_1, y_2)$

Example

$$x = f(x) \wedge p(x, f(x)) \wedge p(f(x), z) \wedge \neg p(x, z)$$

is rewritten to

$$x = f(x) \wedge f_p(x, f(x)) = \bullet \wedge f_p(f(x), z) = \bullet \wedge f_p(x, z) \neq \bullet$$



FIND $f_p(x, z) = \bullet$

FIND $\bullet = \bullet$

$\Rightarrow$ **Unsatisfiable**

Theory of Lists

Theory of Lists T_{cons}

Σ_{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:

$$\begin{aligned}
&\forall x_1, x_2, y_1, y_2.\ x_1 = x_2 \ \wedge\ y_1 = y_2 \ \rightarrow\ \mathsf{cons}(x_1, y_1) = \mathsf{cons}(x_2, y_2)\\
&\forall x, y.\ x = y \ \rightarrow\ \mathsf{car}(x) = \mathsf{car}(y)\\
&\forall x, y.\ x = y \ \rightarrow\ \mathsf{cdr}(x) = \mathsf{cdr}(y)
\end{aligned}$$

- equivalence axiom:

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

- $\forall x, y.\ \mathsf{car}(\mathsf{cons}(x, y)) = x$ (left projection)
  $\forall x, y.\ \mathsf{cdr}(\mathsf{cons}(x, y)) = y$ (right projection)
  $\forall x.\ \neg\mathsf{atom}(x) \rightarrow \mathsf{cons}(\mathsf{car}(x), \mathsf{cdr}(x)) = x$ (construction)
  $\forall x, y.\ \neg\mathsf{atom}(\mathsf{cons}(x, y))$ (atom)

Satisfiabilty of Quantifier-free $\Sigma_{\text{cons}} \cup \Sigma_{\text{E}}$-formulae

First simplify the formula:

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

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

  by the (construction) axiom.

Result is a conjunctive $\Sigma_{\text{cons}} \cup \Sigma_{\text{E}}$-formula with the literals:

- $s = t$
- $s \neq t$
- $atom(u)$

where s, t, u are $T_{\text{cons}} \cup T_{\text{E}}$-terms.

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

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

where s_i, t_i, and u_i are $T_{\text{cons}} \cup T_{\text{E}}$-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 $\text{car}(n)$ and $\textsc{merge}\ \text{car}(n)\ n.\mathtt{args}[1]$
   - add $\text{cdr}(n)$ and $\textsc{merge}\ \text{cdr}(n)\ n.\mathtt{args}[2]$

   by axioms (left projection), (right projection)
3. for $1 \leq i \leq m$, $\textsc{merge}\ s_i\ t_i$
4. for $m + 1 \leq i \leq n$, if $\textsc{find}\ s_i = \textsc{find}\ t_i$, return **unsatisfiable**
5. for $1 \leq i \leq \ell$, if $\exists v.\ \textsc{find}\ v = \textsc{find}\ u_i \ \wedge\ v.\mathtt{fn} = \text{cons}$, return **unsatisfiable**
6. Otherwise, return **satisfiable**

Example

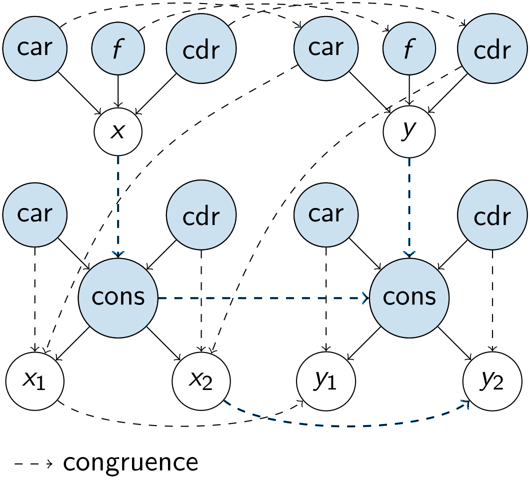
Given $(\Sigma_{\text{cons}} \cup \Sigma_{\text{E}})$-formula

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

where the function symbol f is in Σ_{E}

$$F' : \quad \begin{array}{lll} \text{car}(x) = \text{car}(y) & \wedge & (1) \\ \text{cdr}(x) = \text{cdr}(y) & \wedge & (2) \\ x = \text{cons}(x_1, x_2) & \wedge & (3) \\ y = \text{cons}(y_1, y_2) & \wedge & (4) \\ f(x) \neq f(y) & & (5) \end{array}$$

Example: $\mathrm{car}(x) = \mathrm{car}(y) \wedge \mathrm{cdr}(x) = \mathrm{cdr}(y) \wedge$ $x = cons(x_1, x_2) \wedge y = cons(y_1, y_2) \wedge f(x) \neq f(y)$



Step 1
Step 2
Step 3 :

MERGE car(x) car(y)
MERGE cdr(x) cdr(y)
MERGE x cons(x_1, x_2)
MERGE car(x) car(cons(x_1, x_2))
MERGE cdr(x) cdr(cons(x_1, x_2))
MERGE y cons(y_1, y_2)
MERGE car(y) car(cons(y_1, y_2))
MERGE cdr(y) cdr(cons(y_1, y_2))
MERGE cons(x_1, x_2) cons(y_1, y_2)
MERGE $f(x)$ $f(y)$

Step 4 :

FIND $f(x)$ = FIND $f(y)$
$\Rightarrow$ *unsatisfiable*

Theorem (Sound and Complete)

Quantifier-free conjunctive Σ_{cons}-formula F is T_{cons}-satisfiable iff the congruence closure algorithm for T_{cons} returns satisfiable*.*

Proof:

$\Rightarrow$ Let I be a satisfying interpretation.
By induction over the steps of the algorithm one can prove:
Whenever the algorithm merges nodes t_1 and t_2, $I \models t_1 = t_2$ holds.

Since $I \models s_i \neq t_i$ for $i \in \{m+1, \ldots, n\}$ they cannot be merged.
From $I \models \neg atom(\text{cons}(t_1, t_2))$ and $I \models atom(u_i)$
follows $I \models u_i \neq \text{cons}(t_1, t_2)$ by equivalence axiom.
Thus u_i for $i \in \{1, \ldots, \ell\}$ cannot be merged with a cons node.

Hence the algorithm returns satisfiable.

Proof:

$\Leftarrow$ Let S denote the nodes of the graph and
let $S/\!\sim$ denote the congruence classes computed by the algorithm.
Show that there is an interpretation I:

$$D_I = \{\text{binary trees with leaves labelled with } S/\!\sim\}$$
$$\setminus \left\{\text{trees with subtree } \begin{array}{c}\swarrow\ \searrow\\ [t_1]\qquad [t_2]\end{array} \text{ with } \mathrm{cons}(t_1, t_2) \in S\right\}$$

$$\mathrm{cons}_I(v_1, v_2) = \begin{cases} [cons(t_1, t_2)] & v_1 = [t_1], v_2 = [t_2], \mathrm{cons}(t_1, t_2) \in S \\ \begin{array}{c}\swarrow\ \searrow\\ v_1\qquad v_2\end{array} & \text{otherwise} \end{cases}$$

$$\mathrm{car}_I(v) = \begin{cases} [car(t)] & \text{if } v = [t], \mathrm{car}(t) \in S \\ v_1 & \text{if } v = \begin{array}{c}\swarrow\ \searrow\\ v_1\qquad v_2\end{array} \\ \text{arbitrary} & \text{otherwise} \end{cases}$$

Correctness of the Algorithm (3)

$$\mathrm{cdr}_I(v) = \begin{cases} [cdr(t)] & \text{if } v = [t], \mathrm{cdr}(t) \in S \\ v_2 & \text{if } v = \begin{array}{ccc} & \swarrow\;\searrow & \\ v_1 & & v_2 \end{array} \\ \text{arbitrary} & \text{otherwise} \end{cases}$$

$$atom_I(v) = \begin{cases} \text{false} & \text{if } v = [cons(t_1, t_2)] \\ \text{false} & \text{if } v = \begin{array}{ccc} & \swarrow\;\searrow & \\ v_1 & & v_2 \end{array} \\ \text{true} & \text{otherwise} \end{cases}$$

$$\alpha_I[=](v_1, v_2) = \text{true iff } v_1 = v_2$$

I is well-defined! $\alpha_I[=]$ is obviously a congruence relation.

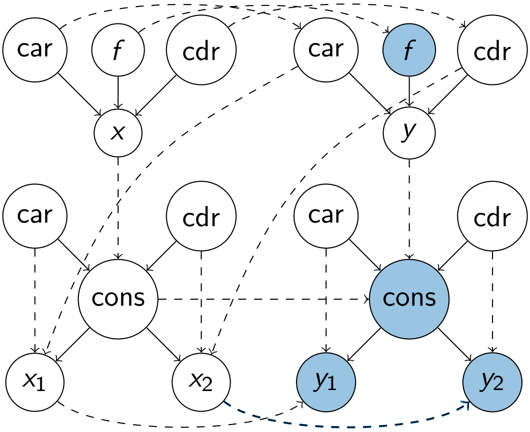
$\forall x, y.\ \mathrm{car}(\mathrm{cons}(x, y)) = x$ (left projection)

$\forall x, y.\ \mathrm{cdr}(\mathrm{cons}(x, y)) = y$ (right projection)

$\forall x.\ \neg\mathrm{atom}(x) \rightarrow \mathrm{cons}(\mathrm{car}(x), \mathrm{cdr}(x)) = x$ (construction)

$\forall x, y.\ \neg\mathrm{atom}(\mathrm{cons}(x, y))$ (atom)

Example: $\mathrm{car}(x) = \mathrm{car}(y) \wedge \mathrm{cdr}(x) = \mathrm{cdr}(y) \wedge$ $x = cons(x_1, x_2) \wedge y = cons(y_1, y_2)$



- - → congruence

Quantifier-free Rationals

Conjunctive Quantifier-free Fragment

In the next lectures, we consider conjunctive quantifier-free Σ-formulae, i.e., conjunctions of Σ-literals (Σ-atoms or negations of Σ-atoms).

Remark 1: From this an algorithm for arbitrary quantifier-free formulae can be built.
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.

Remark 2: One can also combine a decision procedure for conjunctive fragment with DPLL.

Conjunctive Quantifier-free Fragment of Rationals

For $T_{\mathbb{Q}}$ a formula in the conjunctive fragment looks like this:

$$\begin{array}{r}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1 \\
\wedge a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \leq b_2 \\
\vdots \quad \\
\wedge a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leq b_m
\end{array}$$

as vectors: $A \cdot \vec{x} \leq \vec{b}$.

Note: $x = b$ can be expressed as $x \leq b \wedge -x \leq -b$.
$\neg(x \leq b)$ can be expressed as $-x < -b$.
$x < b$ requires some additional handling (later).

Dutertre–de Moura Algorithm

- Presented 2006 by B. Dutertre and L. de Moura
- Based on Simplex algorithm
- Simpler; it doesn't optimize.

Nonbasic and Basic Variables

The set of variables in the formula is called $\mathcal{N}$ (set of non-basic variables).

Additionally we introduce basic variables $\mathcal{B}$, one variable for each linear term in the formula:

$$y_i := a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n$$

The basic variables depend on the non-basic variables.

Note: The naming is counter-intuitive. Unfortunately it is the standard naming for Simplex algorithm.

We need to find a solution for $y_1 \leq b_1, \ldots, y_m \leq b_m$

Computing Basic from Non-basic Variables

The basic variables can be computed by a simple Matrix computation:

$$\begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \dots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

One can also use tableaux notation:

	x_1	$\dots$	x_n
y_1	a_{11}	$\dots$	a_{1n}
$\vdots$	$\vdots$		$\vdots$
y_m	a_{m1}	$\dots$	a_{mn}

We start by setting all non-basic to 0 and computing the basic variables, denoted as $\beta_0(x) := 0$. The valuation β_s assigns values for the variables at step s.

Configuration

A configuration at step s of the algorithm consists of

- a partition of the variables into non-basic and basic variables

$$\mathcal{N}_s \cup \mathcal{B}_s = \{x_1, \ldots, x_n, y_1, \ldots y_m\},$$

- a tableaux A (a $m \times n$ matrix) where the columns correspond to non-basic and rows correspond to basic variables,
- and a valuation β_s, that assigns
  - $\beta_s(x_i) = 0$ for $x_i \in \mathcal{N}_s$,
  - $\beta_s(y_i) = b_i$ for $y_i \in \mathcal{N}_s$,
  - $\beta_s(z_i) = \sum_{z_j \in \mathcal{N}_s} a_{ij}\beta(z_j)$ for $z_i \in \mathcal{B}_s$.

  (Here z stands for either an x or a y variable.)

The initial configuration is:

$$\mathcal{N}_0 = \{x_1, \ldots, x_n\}, \mathcal{B}_0 = \{y_1, \ldots, y_m\}, A_0 = A, \beta_0(x_i) = 0$$

In later steps variables from $\mathcal{N}$ and $\mathcal{B}$ are swapped.

Suppose β_s is not a solution for $y_1 \leq b_1, \dots, y_m \leq b_m$.
Let y_i be a variable whose value $\beta_s(y_i) > b_i$.
Consider the row in the matrix:

$$y_i = a_{i1}z_1 + a_{i2}z_2 + \cdots + a_{in}z_n$$

Idea: Choose a z_j, then solve z_j in the above equation.
Thus, z_j becomes non-basic variable, y_i becomes basic.
Then decrease $\beta(y_i)$ to b_i.
This will either decrease z_j (if $a_{ij} > 0$)
or increase z_j (if $a_{ij} < 0$, z_j must be a x-variable).
Solving z_j in the above equation gives:

$$z_j = \frac{a_{i1}}{-a_{ij}}z_1 + \frac{a_{i2}}{-a_{ij}}z_2 + \cdots + \frac{a_{in}}{-a_{ij}}z_n + \frac{1}{a_{ij}}y_i$$

Result of Pivoting

After pivoting y_i and z_j the matrix looks as follows:

$$\begin{array}{rcrcccrcr}
y_1 & = & (a_{11} - \frac{a_{1j}a_{i1}}{a_{ij}})z_1 & + \cdots + & \frac{a_{1j}}{a_{ij}}y_i & + \cdots + & (a_{1n} - \frac{a_{1j}a_{in}}{a_{ij}})z_n \\
\vdots & & \vdots & & \vdots & & \vdots \\
z_j & = & -\frac{a_{i1}}{a_{ij}}z_1 & + \cdots + & \frac{1}{a_{ij}}y_i & + \cdots + & -\frac{a_{in}}{a_{ij}}z_n \\
\vdots & & \vdots & & \vdots & & \vdots \\
y_m & = & (a_{m1} - \frac{a_{mj}a_{i1}}{a_{ij}})z_1 & + \cdots + & \frac{a_{mj}}{a_{ij}}y_i & + \cdots + & (a_{mn} - \frac{a_{mj}a_{in}}{a_{ij}})z_n
\end{array}$$

Now, set $\beta_{s+1}(y_i)$ to b_i and recompute basic variables.

Detecting Conflicts

We may arrive at a configuration like:

$$y_i = 0 \cdot x_1 + \cdots + a_{ij_1} y_{j_1} + \cdots + a_{ij_k} y_{j_k} + 0 \cdot x_n$$

where the non-basic y variables are set to their bound:

$$\beta_s(y_{j_1}) = b_{j_1}, \ldots, \beta_s(y_{j_k}) = b_{j_k} ,$$

coefficients of x variables are zero, coefficients $a_{ij_1}, \ldots, a_{ij_k} \leq 0$, and $\beta_s(y_i) > b_i$.

Then, we have a conflict:

$$y_{j_1} \leq b_{j_1} \wedge \cdots \wedge y_{j_k} \leq b_{j_k} \rightarrow y_i > b_i .$$

The formula is not satisfiable.

Example

Consider the formula

$$F : x_1 + x_2 \geq 4 \wedge x_1 - x_2 \leq 1$$

We have two non-basic variables $\mathcal{N} = \{x_1, x_2\}$.
Define basic variables $\mathcal{B} = \{y_1, y_2\}$:

$$\begin{aligned} y_1 &= -x_1 - x_2, & y_1 &\leq -4 \\ y_2 &= x_1 - x_2, & y_2 &\leq 1 \end{aligned}$$

We write the equation as a tableaux:

	x_1	x_2
y_1	-1	-1
y_2	1	-1

Example (cont.)

Tableaux:

	x_1	x_2
y_1	-1	-1
y_2	1	-1

Values:

$x_1 = x_2 = 0$

$\rightarrow y_1 = 0 > -4$ (!)

$\rightarrow y_2 = 0 \leq 1$

Pivot y_1 against x_1: $x_1 = -y_1 - x_2$.

New Tableaux:

	y_1	x_2
x_1	-1	-1
y_2	-1	-2

Example (cont.)

Tableaux:

	y_1	x_2
x_1	-1	-1
y_2	-1	-2

Values:

$y_1 = -4, x_2 = 0$
$\rightarrow x_1 = 4$
$\rightarrow y_2 = 4 > 1$ (!)

y_2 cannot be pivoted with y_1, since -1 negative.
Pivot y_2 and x_2:

New Tableaux:

	y_1	y_2
x_1	-.5	.5
x_2	-.5	-.5

Example (cont.)

Tableaux:

	y_1	y_2
x_1	-.5	.5
x_2	-.5	-.5

Values:

$y_1 = -4, y_2 = 1$
$\rightarrow x_1 = 2.5$
$\rightarrow x_2 = 1.5$

We found a satisfying interpretation for:

$$F : x_1 + x_2 \geq 4 \land x_1 - x_2 \leq 1$$

Example

Now, consider the formula

$$F' : x_1 + x_2 \geq 4 \wedge x_1 - x_2 \leq 1 \wedge x_2 \leq 1$$

We have two non-basic variables $\mathcal{N} = \{x_1, x_2\}$.
Define basic variables $\mathcal{B} = \{y_1, y_2, y_3\}$:

$$\begin{aligned} y_1 &= -x_1 - x_2, & y_1 &\leq -4 \\ y_2 &= x_1 - x_2, & y_2 &\leq 1 \\ y_3 &= x_2, & y_3 &\leq 1 \end{aligned}$$

We write the equation as tableaux:

	x_1	x_2
y_1	-1	-1
y_2	1	-1
y_3	0	1

Example (cont.)

The first two steps are identical:
pivot y_1 resp. y_2 and x_1 resp. x_2.

	y_1	y_2
x_1	-.5	.5
x_2	-.5	-.5
y_3	-.5	-.5

Example (cont.)

Tableaux:

	y_1	y_2
x_1	-.5	.5
x_2	-.5	-.5
y_3	-.5	-.5

Values:

$y_1 = -4, y_2 = 1$

$\rightarrow x_1 = 2.5$

$\rightarrow x_2 = 1.5$

$\rightarrow y_3 = 1.5 > 1$!

Now, y_3 cannot pivot, since all coefficients in that row are negative.
Conflict is $-x_1 - x_2 \leq -4 \land x_1 - x_2 \leq 1 \rightarrow x_2 > 1$.
Formula F' is unsatisfiable

Termination

To guarantee termination we need a fixed pivot selection rule.

The following rule works:

When choosing the basic variable (row) to pivot:

- Choose the y-variable with the smallest index, whose value exceeds the bound.
- If there is no such variable, return satisfiable

When choosing the non-basic variable (column) to pivot with:

- if possible, take a x-variable.
- Otherwise, take the y-variable with the smallest index, such that the corresponding coefficient in the matrix is positive.
- If there is no such variable, return unsatisfiable

Termination Proof

Assume we have an infinite computation of the algorithm.

Let y_j be the variable with the largest index, that is infinitely often pivoted.

Look at the step where y_j is pivoted to a non-basic variable and where for $k > j$, y_k is not pivoted any more. The (ordered) tableaux at the point of pivoting looks like this:

	x	$\cdots$	x	y	$\cdots$	y	y_j	y	$\cdots$
$\vdots$									
y_i	0	$\cdots$	0	$-/0$	$\cdots$	$-/0$	$+$	$\pm/0$	$\cdots$
$\vdots$									

($+$ denotes a positive coefficient, $-$ a negative coefficient)

After pivoting the tableaux changes to:

	x	$\cdots$	x	y	$\cdots$	y	y_i	y	$\cdots$
$\vdots$									
y_j	0	$\cdots$	0	$+/0$	$\cdots$	$+/0$	$+$	$\mp/0$	$\cdots$
$\vdots$									

Termination Proof (cont.)

After pivoting the tableaux changes to:

	x	$\cdots$	x	y	$\cdots$	y	y_i	y	$\cdots$
$\vdots$									
y_j	0	$\cdots$	0	$+/0$	$\cdots$	$+/0$	$+$	$\mp/0$	$\cdots$
$\vdots$									

$$\sum_{k<j, y_k \in \mathcal{N}_s} a_k b_k + \sum_{k>j, y_k \in \mathcal{N}_s} a_k b_k = \beta_s(y_j) < b_j, \text{ where } a_k \geq 0 \text{ for } k < j.$$

Now look at the step s' where y_j is pivoted back.
By the pivoting rule: $\beta_{s'}(y_k) \leq b_k$ for all $k < j$.
For $k > j$, the non-basic/basic variables do not change.
Therefore, the value of y_j can only get smaller.

$$\beta_{s'}(y_j) = \sum_{k<j, y_k \in \mathcal{N}_s} a_k \cdot \beta_{s'}(y_k) + \sum_{k>j, y_k \in \mathcal{N}_s} a_k b_k < b_j$$

This contradicts $\beta_{s'}(y_j) > b_j$.

Therefore, assumption was wrong and algorithm terminates.

Strict Bounds

With strict bounds the formula looks like this:

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &\le b_1 \\
&\vdots \\
\wedge a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n &\le b_i \\
\wedge a_{(i+1)1}x_1 + a_{(i+1)2}x_2 + \cdots + a_{(i+1)n}x_n &< b_{i+1} \\
&\vdots \\
\wedge a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &< b_m
\end{aligned}
$$

If the formula is satisfiable, then there is an $\varepsilon > 0$ with:

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &\le b_1 \\
&\vdots \\
\wedge a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n &\le b_i \\
\wedge a_{(i+1)1}x_1 + a_{(i+1)2}x_2 + \cdots + a_{(i+1)n}x_n &\le b_{i+1} - \varepsilon \\
&\vdots \\
\wedge a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &\le b_m - \varepsilon
\end{aligned}
$$

Infinitesimal Numbers

We compute with ε symbolically. Our bounds are elements of

$$\mathbb{Q}_\varepsilon := \{a_1 + a_2\varepsilon \mid a_1, a_2 \in \mathbb{Q}\}$$

The arithmetical operators and the ordering are defined as:

$$\begin{aligned}(a_1 + a_2\varepsilon) + (b_1 + b_2\varepsilon) &= (a_1 + b_1) + (a_2 + b_2)\varepsilon \\ a \cdot (b_1 + b_2\varepsilon) &= ab_1 + ab_2\varepsilon \\ a_1 + a_2\varepsilon \leq b_1 + b_2\varepsilon &\text{ iff } a_1 < b_1 \lor (a_1 = b_1 \land a_2 \leq b_2)\end{aligned}$$

Note: $\mathbb{Q}_\varepsilon$ is a two-dimensional vector space over $\mathbb{Q}$.
Changes to the configuration:

- β gives values for variables in $\mathbb{Q}_\varepsilon$.
- The tableaux does not contain ε. It is still a $\mathbb{Q}^{m \times n}$ matrix.

Example

$$F_1 \;:\; 3x_1 + 2x_2 < 5 \wedge 2x_1 + 3x_2 < 1 \wedge x_1 + x_2 > 1$$

Example F_1

Step 1:

	x_1	x_2	β	b_i	
β	0	0			
y_1	3	2	0	$5 - \varepsilon$	
y_2	2	3	0	$1 - \varepsilon$	
y_3	-1	-1	0	$-1 - \varepsilon$	(!)

Step 2:

	y_3	x_2	β	b_i	
β	$-1 - \varepsilon$	0			
y_1	-3	-1	$3 + 3\varepsilon$	$5 - \varepsilon$	
y_2	-2	1	$2 + 2\varepsilon$	$1 - \varepsilon$	(!)
x_1	-1	-1	$1 + 1\varepsilon$		

Step 3:

	y_3	y_2	β	b_i
β	$-1 - \varepsilon$	$1 - \varepsilon$		
y_1	-5	-1	$4 + 6\varepsilon$	$5 - \varepsilon$
x_2	2	1	$-1 - 3\varepsilon$	
x_1	-3	-1	$2 + 4\varepsilon$	

$\beta(y_1) = 4 + 6\varepsilon \leq 5 - \varepsilon$ (for $0 < \varepsilon \leq 1/7$).

Solution ($\varepsilon = 0.1$): $x_1 = 2.4$, $x_2 = -1.3$.

Example

$$F_2 \,:\, 3x_1 + 2x_2 < 5 \land 2x_1 - x_2 > 1 \land x_1 + 3x_2 > 4$$

Example F_2

Step 1:

	x_1	x_2	β	b_i	
β	0	0			
y_1	3	2	0	$5-\varepsilon$	
y_2	-2	1	0	$-1-\varepsilon$	(!)
y_3	-1	-3	0	$-4-\varepsilon$	(!)

Step 2:

	x_1	y_2	β	b_i	
β	0	$-1-\varepsilon$			
y_1	7	2	$-2-2\varepsilon$	$5-\varepsilon$	
x_2	2	1	$-1-\varepsilon$		
y_3	-7	-3	$3+3\varepsilon$	$-4-\varepsilon$	(!)

Step 3:

	y_3	y_2	β	b_i	
β	$-4-\varepsilon$	$-1-\varepsilon$			
y_1	-1	-1	$5+2\varepsilon$	$5-\varepsilon$	(!)
x_2	-2/7	1/7	$1+1/7\varepsilon$		
x_1	-1/7	-3/7	$1+4/7\varepsilon$		

Now $5+2\varepsilon > 5-\varepsilon$ but all coefficients in first row negative.

Unsatisfiable.

Theorem (Sound and Complete)

Quantifier-free conjunctive $\Sigma_{\mathbb{Q}}$-formula F is $T_{\mathbb{Q}}$-satisfiable iff the Dutertre-de-Moura algorithm returns satisfiable*.*

Theory of Arrays

Arrays: 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. It is not used on arrays.

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)

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

Step 1
For every read-over-write term $a\langle i \triangleleft v\rangle[j]$ in F, replace F with the formula

$$\begin{array}{c}(i = j \wedge F\{a\langle i \triangleleft v\rangle[j] \mapsto v\}) \vee \\ (i \neq j \wedge F\{a\langle i \triangleleft v\rangle[j] \mapsto a[j]\})\end{array}$$

Repeat until there are no more read-over-write terms.

Step 2

Associate array variables a with fresh function symbol f_a.
Replace read terms $a[i]$ with $f_a(i)$.

Step 3

Now F is a T_E-Formula. Decide T_E-satisfiability using the congruence-closure algorithm for each of the disjuncts produced in Step 1.

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 read-over-write term,

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

Rewrite it to $F_1 \vee F_2$ with:

$$F_1 : i_2 = j \wedge i_1 = j \wedge i_1 \neq i_2 \wedge a[j] = v_1 \wedge v_2 \neq a[j] \; ,$$
$$F_2 : i_2 \neq j \wedge i_1 = j \wedge i_1 \neq i_2 \wedge a[j] = v_1 \wedge a\langle i_1 \triangleleft v_1\rangle[j] \neq a[j] \; .$$

F_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.

Now, we try the second case (F_2):
F_2 contains the read-over-write term $a\langle i_1 \triangleleft v_1\rangle[j]$. Rewrite it to $F_3 \vee F_4$ with

$$F_3 : i_1 = j \wedge i_2 \neq j \wedge i_1 = j \wedge i_1 \neq i_2 \wedge a[j] = v_1 \wedge v_1 \neq a[j] \,,$$
$$F_4 : i_1 \neq j \wedge i_2 \neq j \wedge i_1 = j \wedge i_1 \neq i_2 \wedge a[j] = v_1 \wedge a[j] \neq a[j] \,.$$

Rewrite the array reads to

$$F_3' : i_1 = j \wedge i_2 \neq j \wedge i_1 = j \wedge i_1 \neq i_2 \wedge f_a(j) = v_1 \wedge v_1 \neq f_a(j) \,,$$
$$F_4' : i_1 \neq j \wedge i_2 \neq j \wedge i_1 = j \wedge i_1 \neq i_2 \wedge f_a(j) = v_1 \wedge f_a(j) \neq f_a(j) \,.$$

In F_3' there is a contradiction because of the final two terms. In F_4', there are two contradictions: the first and third literals contradict each other, and the final literal is contradictory. Since F is equisatisfiable to $F_1' \vee F_3' \vee F_4'$, 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?

Complexity of Decision Procedure for T_A

Our algorithm has a big disadvantage. Step 1 doubles the size of the formula:

$$(i = j \wedge F\{a\langle i \triangleleft v\rangle[j] \mapsto v\}) \vee$$
$$(i \neq j \wedge F\{a\langle i \triangleleft v\rangle[j] \mapsto a[j]\})$$

This can be avoided by introducing fresh variables x_{aijv}:

$$F\{a\langle i \triangleleft v\rangle[j] \mapsto x_{aijv}\}\wedge$$
$$((i = j \wedge x_{aijv} = v) \vee (i \neq j \wedge x_{aijv} = a[j]))$$

However, this is not in the conjunctive fragment of T_E.

There is no way around:

The conjunctive fragment of T_A is NP-complete.

Arrays and Quantifiers

In programming languages, one often needs to express the following concepts:

- Containment *contains*(a, ℓ, u, e): the array a contains element e at some index between ℓ and u.

$$\exists i.\ell \leq i \leq u \land a[i] = e$$

- Sortedness *sorted*(a, ℓ, u): the array a is sorted between index ℓ and index u.

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

- Partitioning *partition*$(a, \ell_1, u_1, \ell_2, u_2)$: The array elements between ℓ_1 and u_1 are smaller than all elements between ℓ_2 and u_2.

$$\forall i, j.\ell_1 \leq i \leq u_1 \land \ell_2 \leq j \leq u_2 \implies a[i] \leq a[j]$$

These concepts can only be expressed as first-order formulae with quantifiers.

However: the general theory of arrays T_{A} with quantifier is not decidable.

Is there a decidable fragment of T_{A} that contains the above formulae?

Example

We want to prove validity for a formula, such as:

$$\neg contains(a, \ell, u, e) \wedge e \neq f \rightarrow \neg contains(a\langle j \lhd f\rangle, \ell, u, e)$$

$$\neg(\exists i. \ell \leq i \leq u \wedge a[i] = e) \wedge e \neq f$$
$$\rightarrow \neg(\exists i. \ell \leq i \leq u \wedge a\langle j \lhd f\rangle[i] \neq e).$$

Check satisfiability of negated formula:

$$\neg(\exists i. \ell \leq i \leq u \wedge a[i] = e) \wedge e \neq f \wedge (\exists i. \ell \leq i \leq u \wedge a\langle j \lhd f\rangle[i] \neq e).$$

Negation Normal Form:

$$(\forall i. \ell > i \vee i > u \vee a[i] \neq e) \wedge e \neq f \wedge (\exists i. \ell \leq i \wedge i \leq u \wedge a\langle j \lhd f\rangle[i] = e).$$

or the equisatisfiable formula

$$\forall i. \ell > i \vee i > u \vee a[i] \neq e \wedge e \neq f \wedge \ell \leq i_2 \wedge i_2 \leq u \wedge a\langle j \lhd f\rangle[i_2] = e.$$

We need to handle satisfiability for universal quantifiers.

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 $F[\bar{i}]$:

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

  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: Boolean combinations of quantifier-free T_A-formulae and array properties

Example: Array Property Fragment

Is this formula in the array property fragment?

$$F \;:\; \forall i.\; i \neq a[k] \rightarrow a[i] = a[k]$$

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.
This trick works for every term that does not contain a uvar.
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.

Example: Array Property Fragment (cont)

Is this formula in the array property fragment?

$$F' : \forall ij.\ i \neq j \rightarrow a[i] \neq a[j]$$

No, the term uvar $\neq$ uvar is not allowed in the index guard. There is no workaround.

Array property fragment and extensionality

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.
F' is in array property fragment of T_{A}.

Decision Procedure for Array Property Fragment

Basic Idea: Similar to quantifier elimination.

Replace universal quantification

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

by finite conjunction

$$F[t_1] \wedge \ldots \wedge F[t_n].$$

We call $t_1, \ldots, t_n$ the index terms and they depend on the formula.

Example

Consider

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

which expands to

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

Intuitively, only the index i is important:

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

or simply

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

Simplifying,

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

it is clear that this formula, and thus F, is T_{A}-unsatisfiable.

Given array property formula F, decide its T_A-satisfiability by the following steps:

Step 1
Put F in NNF, but do not rewrite inside a quantifier.

Step 2
Apply the following rule exhaustively to remove writes:

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

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 number 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

Is this $T_A^=$-formula valid?

$$F \; : \; (\forall i.\, i \neq k \rightarrow a[i] = b[i]) \wedge b[k] = v \rightarrow a\langle k \triangleleft v\rangle = b$$

Check satisfiability of:

$$\neg((\forall i.\, i \neq k \rightarrow a[i] = b[i]) \wedge b[k] = v \rightarrow (\forall i.\, a\langle k \triangleleft v\rangle[i] = b[i]))$$

Step 1: NNF

$$F_1 \; : \; (\forall i.\, i \neq k \rightarrow a[i] = b[i]) \wedge b[k] = v \wedge (\exists i.\, a\langle k \triangleleft v\rangle[i] \neq b[i])$$

Step 2: Remove array writes

$$\begin{aligned} F_2 \; : \; & (\forall i.\, i \neq k \rightarrow a[i] = b[i]) \wedge b[k] = v \wedge (\exists i.\, a'[i] \neq b[i]) \\ & \wedge a'[k] = v \wedge (\forall i.\, i \neq k \rightarrow a'[i] = a[i]) \end{aligned}$$

Step 3: Remove existential quantifier

$$\begin{aligned} F_3 \; : \; & (\forall i.\, i \neq k \rightarrow a[i] = b[i]) \wedge b[k] = v \wedge a'[j] \neq b[j] \\ & \wedge a'[k] = v \wedge (\forall i.\, i \neq k \rightarrow a'[i] = a[i]) \end{aligned}$$

Example (cont)

Step 4: Compute index set $\mathcal{I} = \{\lambda, k, j\}$
Step 5+6: Replace universal quantifier:

$$\begin{aligned} F_6 :\ & (\lambda \neq k \rightarrow a[\lambda] = b[\lambda]) \\ & \wedge (k \neq k \rightarrow a[k] = b[k]) \\ & \wedge (j \neq k \rightarrow a[j] = b[j]) \\ & \wedge b[k] = v \wedge a'[j] \neq b[j] \wedge a'[k] = v \\ & \wedge (\lambda \neq k \rightarrow a'[\lambda] = a[\lambda]) \\ & \wedge (k \neq k \rightarrow a'[k] = a[k]) \\ & \wedge (j \neq k \rightarrow a'[j] = a[j]) \\ & \wedge \lambda \neq k \wedge \lambda \neq j \end{aligned}$$

Case distinction on $j = k$ proves unsatisfiability of F_6.
Therefore F is valid

The importance of λ

Is this formula satisfiable?

$$F : (\forall i. i \neq j \rightarrow a[i] = b[i]) \wedge (\forall i. i \neq k \rightarrow a[i] \neq b[i])$$

The algorithm produces:

$$\begin{aligned} F_6 : \lambda \neq j \rightarrow a[\lambda] = b[\lambda] \\ \wedge\ j \neq j \rightarrow a[j] = b[j] \\ \wedge\ k \neq j \rightarrow a[k] = b[k] \\ \wedge\ \lambda \neq k \rightarrow a[\lambda] \neq b[\lambda] \\ \wedge\ j \neq k \rightarrow a[j] \neq b[j] \\ \wedge\ k \neq k \rightarrow a[k] \neq b[k] \\ \wedge\ \lambda \neq j \wedge \lambda \neq k \end{aligned}$$

The first, fourth and last line give a contradiction!

Without λ we had the formula:

$$\begin{aligned} F_6' : j \neq j &\rightarrow a[j] = b[j] \\ \land\, k \neq j &\rightarrow a[k] = b[k] \\ \land\, j \neq k &\rightarrow a[j] \neq b[j] \\ \land\, k \neq k &\rightarrow a[k] \neq b[k] \end{aligned}$$

which simplifies to:

$$j \neq k \rightarrow a[k] = b[k] \land a[j] \neq b[j]\,.$$

This formula is satisfiable!

Theorem

Consider a Σ_A-formula F from the array property fragment of T_A. The output F_6 of Step 6 of the algorithm is T_A-equisatisfiable to F.

This also works when extending the Logic with an arbitrary theory T with signature Σ for the elements:

Theorem

Consider a $\Sigma_A \cup \Sigma$-formula F from the array property fragment of $T_A \cup T$. The output F_6 of Step 6 of the algorithm is $T_A \cup T$-equisatisfiable to F.

Proof: It is easy to see that steps 1–3 do not change the satisfiability of formula.
For step 4–6 we need to show:

$$
\begin{array}{c}
(1) \quad H[\forall \bar{i}.\ (F[\bar{i}] \rightarrow G[\bar{i}])] \text{ is satisfiable} \\
\text{iff.} \\
(2) \quad H[\bigwedge_{\bar{i} \in \mathcal{I}^n} (F[\bar{i}] \rightarrow G[\bar{i}])] \wedge \bigwedge_{i \in \mathcal{I} \setminus \{\lambda\}} \lambda \neq i \text{ is satisfiable.}
\end{array}
$$

If the formula (1) is satisfied some Interpretation, then (2) holds in the same interpretation.

If the formula (2) holds in some interpretation I, we construct an interpretation J as follows:

$$\begin{aligned}
proj_{\mathcal{I}}(j) &= \begin{cases} i & \text{if } i \in \mathcal{I} \wedge \alpha_I[j] = \alpha_I[i] \\ \lambda & \text{otherwise} \end{cases} \\
\alpha_J[a[j]] &= \alpha_I[a[proj_{\mathcal{I}}(j)]] \\
\alpha_J[x] &= \alpha_I[x] \text{ for every non-array variable and constant}
\end{aligned}$$

J interprets the symbols occuring in formula (2) in the same way as I. Therefore, (2) holds in J.
To prove that formula (1) holds in J, it suffices to show:

$$J \models \bigwedge_{\bar{i} \in \mathcal{I}^n} (F[\bar{i}] \rightarrow G[\bar{i}]) \text{ implies } J \models \forall \bar{i}.\ (F[\bar{i}] \rightarrow G[\bar{i}])$$

Proof of Theorem (cont)

Assume $J \models \bigwedge_{\bar{i} \in \mathcal{I}^n}(F[\bar{i}] \to G[\bar{i}])$. Show:

$$F[\bar{i}] \to F[proj_{\mathcal{I}}(\bar{i})] \to G[proj_{\mathcal{I}}(\bar{i})] \to G[\bar{i}]$$

The first implication $F[\bar{i}] \to F[proj_{\mathcal{I}}(\bar{i})]$ can be shown by structural induction over F. Base cases:

- $var_1 = var_2 \to proj_{\mathcal{I}}(var_1) = proj_{\mathcal{I}}(var_2)$: trivial.
- $evar_1 \neq var_2 \to proj_{\mathcal{I}}(evar_1) \neq proj_{\mathcal{I}}(var_2)$:
  By definition of $\mathcal{I}$: $evar_1 \in \mathcal{I} \setminus \{\lambda\}$.
  If $evar_1 = proj_{\mathcal{I}}(evar_1) = proj_{\mathcal{I}}(var_2)$, then $var_2 \in \mathcal{I} \setminus \{\lambda\}$, hence
  $evar_1 = proj_{\mathcal{I}}(var_2) = var_2$
- $var_1 \neq evar_2$ analogously.

The induction step is trivial.
The second implication $F[proj_{\mathcal{I}}(\bar{i})] \to G[proj_{\mathcal{I}}(\bar{i})]$ holds by assumption.
The third implication $G[proj_{\mathcal{I}}(\bar{i})] \implies G[\bar{i}]$ holds because G contains variables i only in array reads $a[i]$. By definition of J:
$\alpha_J[a[i]] = \alpha_J[a[proj_{\mathcal{I}}(i)]]$.

Theory of Integer-Indexed Arrays

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

signature of $T_A^{\mathbb{Z}}$: $\Sigma_A^{\mathbb{Z}} = \Sigma_A \cup \Sigma_{\mathbb{Z}}$

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

Array Property Fragment of $T_A^{\mathbb{Z}}$

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:

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

  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.

- 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 $T_A^{\mathbb{Z}} \cup T_{\mathbb{Q}}$:

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

Step 1
Put F in NNF.

Step 2
Apply the following rule exhaustively to remove writes:

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

To meet the syntactic requirements on an index guard, rewrite the third conjunct as

$$\forall j.\, j \leq i - 1 \vee i + 1 \leq j \rightarrow a[j] = a'[j]\ .$$

The Decision Procedure (Step 3–4)

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:

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

In NNF, we have

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

Step 2 produces

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

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

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

Simplifying,

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

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:

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

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

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

$(T_{\mathsf{A}} \cup T_{\mathbb{Z}})$-unsatisfiability of this quantifier-free $(\Sigma_{\mathsf{A}} \cup \Sigma_{\mathbb{Z}})$-formula can be decided using the techniques of Combination of Theories.
Informally, $\ell \leq k \leq u + 1$ (3)

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

Hence, F is $T_{\mathsf{A}}^{\mathbb{Z}}$-unsatisfiable.

Correctness of Decision Procedure

Theorem

Consider a $\Sigma_{\mathsf{A}}^{\mathbb{Z}} \cup \Sigma$-formula F from the array property fragment of $T_{\mathsf{A}}^{\mathbb{Z}} \cup T$. The output F_5 of Step 5 of the algorithm is $T_{\mathsf{A}}^{\mathbb{Z}} \cup T$-equisatisfiable to F.

Proof: The proof proceeds using the same strategy as for T_A.
It is easy to see that steps 1–3 do not change the satisfiability of formula.
For step 4–5 we need to show:

$$\begin{array}{cl}(1) & H[\forall \bar{i}.\ (F[\bar{i}] \rightarrow G[\bar{i}])] \text{ is satisfiable} \\ & \text{iff.} \\ (2) & H[\bigwedge_{\bar{i} \in \mathcal{I}^n} (F[\bar{i}] \rightarrow G[\bar{i}])] \text{ is satisfiable.}\end{array}$$

$\Rightarrow$: Obviously formula (1) implies formula (2).

Proof of Theorem (cont)

If the formula (2) holds in some interpretation $I = (D_I, \alpha_I)$, we construct an interpretation $J = (D_J, \alpha_J)$ with $D_J := D_I$ and

$$proj_{\mathcal{I}}(j) = \begin{cases} max\{\alpha_I[i] \| i \in \mathcal{I} \wedge \alpha_I[i] \leq \alpha_I[j]\} & \text{if for some } i \in \mathcal{I}\text{:} \\ & \alpha_I[i] \leq \alpha_I[j] \\ min\{\alpha_I[i] \| i \in \mathcal{I} \wedge \alpha_I[i] \geq \alpha_I[j]\} & \text{otherwise} \end{cases}$$

$$\alpha_J[a[j]] = \alpha_I[a[proj_{\mathcal{I}}(j)]]$$

$$\alpha_J[x] = \alpha_I[x] \text{ for every non-array variable and constant}$$

J interprets the symbols occuring in formula (2) in the same way as I.
Therefore, (2) holds in J.
To prove that formula (1) holds in J, it suffices to show:

$$J \models \bigwedge_{\bar{i} \in \mathcal{I}^n} (F[\bar{i}] \rightarrow G[\bar{i}]) \text{ implies } J \models \forall \bar{i}.\ (F[\bar{i}] \rightarrow G[\bar{i}])$$

Proof of Theorem (cont)

Assume $J \models \bigwedge_{\bar{i} \in \mathcal{I}^n}(F[\bar{i}] \rightarrow G[\bar{i}])$. Show:

$$F[\bar{i}] \rightarrow F[proj_{\mathcal{I}}(\bar{i})] \rightarrow G[proj_{\mathcal{I}}(\bar{i})] \rightarrow G[\bar{i}]$$

The first implication $F[\bar{i}] \rightarrow F[proj_{\mathcal{I}}(\bar{i})]$ can be shown by structural induction over F. Base cases:

- $expr_1 \leq expr_2$: see exercise.
- $expr_1 = expr_2$: follows from first case since it is equivalent to

$$expr_1 \leq expr_2 \wedge expr_2 \leq expr_1 .$$

The induction step is trivial.
The second implication $F[proj_{\mathcal{I}}(\bar{i})] \rightarrow G[proj_{\mathcal{I}}(\bar{i})]$ holds by assumption.
The third implication $G[proj_{\mathcal{I}}(\bar{i})] \implies G[\bar{i}]$ holds because G contains variables i only in array reads $a[i]$. By definition of J:
$\alpha_J[a[i]] = \alpha_J[a[proj_{\mathcal{I}}(i)]]$.

Nelson-Oppen Theory Combination

Motivation: 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?

Given

Multiple 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 $\cup_i T_i$.

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

Σ_1-theory T_1

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

Σ_2-theory T_2

$\boxed{P_2}$ 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

We show how to get Procedure P from Procedures P_1 and P_2.

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

Problem: The T_1/T_2-interpretations must have the same data domain; it turns out same cardinality, e.g. infinite, is enough.

Definition (stably infinite)

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

Example: Stably Infinite

- $T_{\mathbb{Z}}$: stably infinite (all T-interpretations are infinite).
- $T_{\mathbb{Q}}$: stably infinite (all T-interpretations are infinite).
- T_{E}: stably infinite (one can add infinitely many fresh and distinct values).
- Σ-theory T with $\Sigma : \{a, b, =\}$ and axiom $\forall x.\ x = a \ \vee \ x = b$: not stable infinite,
  since every T-interpretation has at most two elements.

Example: Σ_E and $\Sigma_{\mathbb{Z}}$

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.

F is $(T_E \cup T_{\mathbb{Z}})$-unsatisfiable:
The first two literals imply $x = 1 \vee x = 2$ so that
$f(x) = f(1) \vee f(x) = f(2)$. This contradicts last two literals.

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 $\mathrm{hd}(t)$ be the root symbol, e.g. $\mathrm{hd}(f(x)) = f$.

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

$$F[f(t_1, \ldots, t, \ldots, t_n)] \quad \textit{eqsat.} \quad F[f(t_1, \ldots, w, \ldots, t_n)] \land w = t$$

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

$$F[p(t_1, \ldots, t, \ldots, t_n)] \quad \textit{eqsat.} \quad F[p(t_1, \ldots, w, \ldots, t_n)] \land w = t$$

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

$$F[s = t] \quad \textit{eqsat.} \quad F[\top] \land w = s \land w = t$$

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

$$F[s \neq t] \quad \textit{eqsat.} \quad F[w_1 \neq w_2] \land w_1 = s \land w_2 = t$$

where w, w_1, and w_2 are fresh variables.

Example: Phase 1

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: Phase 1

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

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

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

$$w_1 = f(x) \wedge w_1 = x + y \, .$$

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

$$f(x) \neq f(w_2) \wedge w_2 = 2 \, .$$

Now, separating the literals results in two formulae:

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

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

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

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

Phase 2: Guess and Check (Nondeterministic)

- 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:

$$\alpha(V, E) : \bigwedge_{u,v \in V.\ uEv} u = v \wedge \bigwedge_{u,v \in V.\ \neg(uEv)} u \neq v$$

Lemma

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.

Proof:

$\Rightarrow$ If F is $(T_1 \cup T_2)$-satisfiable, then $F_1 \wedge F_2$ is $(T_1 \cup T_2)$-satisfiable, hence there is a $T_1 \cup T_2$-Interpretation I with $I \models F_1 \wedge F_2$.

Define $E \subseteq V \times V$ with $u \; E \; v$ iff $I \models u = v$.
Then E is a equivalence relation.
By definition of E and $\alpha(V, E)$, $I \models \alpha(V, E)$.
Hence $I \models F_1 \wedge \alpha(V, E)$ and $I \models F_2 \wedge \alpha(V, E)$.
Thus, these formulae are T_1- and T_2-satisfiable, respectively.

$\Leftarrow$ Let I_1 and I_2 be T_1- and T_2-interpretations, respectively, with

$$I_1 \models F_1 \wedge \alpha(V,E) \text{ and } I_2 \models F_2 \wedge \alpha(V,E).$$

W.l.o.g. assume that $\alpha_{I_1}[=](v,w)$ iff $v = w$ iff $\alpha_{I_2}[=](v,w)$.
(Otherwise, replace D_{I_i} with $D_{I_i}/\alpha_{I_i}[=]$)

Since T_1 and T_2 are stably infinite, we can assume that D_{I_1} and D_{I_2} are of the same cardinality.

Since $I_1 \models \alpha(V,E)$ and $I_2 \models \alpha(V,E)$, for $x, y \in V$:

$$\alpha_{I_1}[x] = \alpha_{I_1}[y] \text{ iff } \alpha_{I_2}[x] = \alpha_{I_2}[y].$$

Construct bijective function $g : D_{I_1} \to D_{I_2}$ with $g(\alpha_{I_1}[x]) = \alpha_{I_2}[x]$ for all $x \in V$. Define I as follows: $D_I = D_{I_2}$,
$\alpha_I[x] = \alpha_{I_2}[x](= g(\alpha_{I_1}[x]))$ for $x \in V$,
$\alpha_I[=](v,w)$ iff $v = w$,
$\alpha_I[f_2] = \alpha_{I_2}[f_2]$ for $f_2 \in \Sigma_2$,
$\alpha_I[f_1](v_1,\dots,v_n) = g(\alpha_{I_1}[f_1](g^{-1}(v_1),\dots,g^{-1}(v_n)))$ for $f_1 \in \Sigma_1$.

Then I is a $T_1 \cup T_2$-interpretation, and satisfies $F_1 \wedge F_2$.
Hence F is $T_1 \cup T_2$-satisfiable.

Example: Phase 2

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:

Example: Phase 2 (cont)

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: Phase 2 (cont)

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 arithmetic $T_{\mathbb{R}}$

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

Theory of equality T_E

$\boxed{P_E}$

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

Phase 1: Variable Abstraction

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

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

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

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

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

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

Phase 2: Equality Propagation

$\boxed{P_{\mathbb{R}}}$ $\quad s_0 : \langle \Gamma_{\mathbb{R}}, \Gamma_E, \{\} \rangle$ $\quad \boxed{P_E}$

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

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

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

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

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

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

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

$$s_4 : \text{false}$$

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.

Convex Theories

Definition (convex theory)

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 $i \in \{1, \ldots, n\}$

Claim

Equality propagation is a decision procedure for 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

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

Then

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

but

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

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

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

but

$$\begin{array}{l} F \not\Rightarrow i = j \\ F \not\Rightarrow a[j] = v \ . \end{array}$$

What if T is Not Convex?

Case split when:

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

but

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

- 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

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

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

$x = w_1$

$x = w_2$

$s_1 : \langle \Gamma_{\mathbb{Z}}, \Gamma_E, \{x = w_1\} \rangle$

$\Gamma_E \cup \{x = w_1\} \models \bot$

$s_3 : \langle \Gamma_{\mathbb{Z}}, \Gamma_E, \{x = w_2\} \rangle$

$\Gamma_E \cup \{x = w_2\} \models \bot$

$s_2 : \bot$

$s_4 : \bot$

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

Example 3: Non-Convex Theory

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

- 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:

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

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

Example 3: Non-Convex Theory

$$s_0 : \langle \Gamma_{\mathbb{Z}}, \Gamma_E, \{\} \rangle$$
$$\Gamma_{\mathbb{Z}} \models x = w_1 \;\vee\; x = w_2 \;\vee\; x = w_3$$

$x = w_1$ | $x = w_2$ | $x = w_3$

$$s_1 : \langle \Gamma_{\mathbb{Z}}, \Gamma_E, \{x = w_1\} \rangle \qquad s_3 : \langle \Gamma_{\mathbb{Z}}, \Gamma_E, \{x = w_2\} \rangle \qquad s_4 : \langle \Gamma_{\mathbb{Z}}, \Gamma_E, \{x = w_3\} \rangle$$
$$\Gamma_E \cup \{x = w_1\} \models \bot \qquad\qquad \Gamma_E \cup \{x = w_3\} \models \bot$$

$$s_2 : \bot \qquad\qquad s_5 : \bot$$

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

DPLL(T)

Suppose we have a $T_{\mathbb{Q}}$-formulae that is not conjunctive:

$$(x \geq 0 \rightarrow y > z) \land (x + y \geq z \rightarrow y \leq z) \land (y \geq 0 \rightarrow x \geq 0) \land x + y \geq z$$

Our approach so far: Converting to DNF.
Yields in 8 conjuncts that have to be checked separately.

Is there a more efficient way to prove unsatisfiability?

CNF and Propositional Core

Suppose we have the following $T_{\mathbb{Q}}$-formulae:

$$(x \geq 0 \rightarrow y > z) \land (x + y \geq z \rightarrow y \leq z) \land (y \geq 0 \rightarrow x \geq 0) \land x + y \geq z$$

Converting to CNF and restricting to $\leq$:

$$\begin{aligned} &(\neg(0 \leq x) \lor \neg(y \leq z)) \land (\neg(z \leq x + y) \lor (y \leq z)) \\ &\qquad \land (\neg(0 \leq y) \lor (0 \leq x)) \land (z \leq x + y) \end{aligned}$$

Now, introduce boolean variables for each atom:

$$\begin{array}{llll} P_1 : 0 \leq x & & P_2 : y \leq z \\ P_3 : z \leq x + y & & P_4 : 0 \leq y \end{array}$$

Gives a propositional formula:

$$(\neg P_1 \lor \neg P_2) \land (\neg P_3 \lor P_2) \land (\neg P_4 \lor P_1) \land P_3$$

The core feature of the DPLL-algorithm is Unit Propagation.

$$(\neg P_1 \vee \neg P_2) \wedge (\neg P_3 \vee P_2) \wedge (\neg P_4 \vee P_1) \wedge P_3$$

The clause P_3 is a unit clause; set P_3 to $\top$.
Then $\neg P_3 \vee P_2$ is a unit clause; set P_2 to $\top$.
Then $\neg P_1 \vee \neg P_2$ is a unit clause; set P_1 to $\bot$.
Then $\neg P_4 \vee P_1$ is a unit clause; set P_4 to $\bot$.

Only solution is $P_3 \wedge P_2 \wedge \neg P_1 \wedge \neg P_4$.

DPLL-Algorithm

Only solution is $P_3 \wedge P_2 \wedge \neg P_1 \wedge \neg P_4$.

$$\begin{array}{llll} P_1 : 0 \leq x & & & P_2 : y \leq z \\ P_3 : z \leq x + y & & & P_4 : 0 \leq y \end{array}$$

This gives the conjunctive $T_{\mathbb{Q}}$-formula

$$z \leq x + y \wedge y \leq z \wedge x < 0 \wedge y < 0.$$

DPLL(T) with Learning (CDCL)

We describe DPLL(T) by a set of rules modifying a configuration.
A configuration is a triple

$$\langle M, F, C \rangle ,$$

where

- M (model) is a sequence of literals (that are currently set to true) interspersed with backtracking points denoted by $\square$.
- F (formula) is a formula in CNF,
  i. e., a set of clauses where each clause is a set of literals.
- C (conflict) is either $\top$ or a conflict clause (a set of literals).
  A conflict clause C is a clause with $F \Rightarrow C$ and $M \not\models C$.
  Thus, a conflict clause shows $M \not\models F$.

Rule Based Description

We describe the algorithm by a set of rules, which each describe a set of transitions between configurations, e. g.,

Explain $\dfrac{\langle M, F, C \cup \{\ell\}\rangle}{\langle M, F, C \cup \{\ell_1, \dots, \ell_k\}\rangle}$ where $\ell \notin C$, $\{\ell_1, \dots, \ell_k, \bar{\ell}\} \in F$, and $\bar{\ell_1}, \dots, \bar{\ell_k} \prec \bar{\ell}$ in M.

Here, $\bar{\ell_1}, \dots, \bar{\ell_k} \prec \ell$ in M means the literals $\bar{\ell_1}, \dots, \bar{\ell_k}$ occur in the sequence M before the literal ℓ (and all literals appear in M).

Example: for $M = P_1 \bar{P}_3 \bar{P}_2 \bar{P}_4$, $F = \{\{P_1\}, \{P_3, \bar{P}_4\}\}$, and $C = \{P_2\}$ the transition

$$\langle M, F, \{P_2, P_4\}\rangle \longrightarrow \langle M, F, \{P_2, P_3\}\rangle$$

is possible.

Rules for CDCL (Conflict Driven Clause Learning)

Decide	$\dfrac{\langle M, F, \top\rangle}{\langle M \cdot \Box \cdot \ell, F, \top\rangle}$	where $\ell \in lit(F)$, $\ell, \overline{\ell}$ $\not\text{in}$ M
Propagate	$\dfrac{\langle M, F, \top\rangle}{\langle M \cdot \ell, F, \top\rangle}$	where $\{\ell_1, \dots, \ell_k, \ell\} \in F$ and $\overline{\ell_1}, \dots, \overline{\ell_k}$ in M, $\ell, \overline{\ell}$ $\not\text{in}$ M.
Conflict	$\dfrac{\langle M, F, \top\rangle}{\langle M, F, \{\ell_1, \dots, \ell_k\}\rangle}$	where $\{\ell_1, \dots, \ell_k\} \in F$ and $\overline{\ell_1}, \dots, \overline{\ell_k}$ in M.
Explain	$\dfrac{\langle M, F, C \cup \{\ell\}\rangle}{\langle M, F, C \cup \{\ell_1, \dots, \ell_k\}\rangle}$	where $\ell \notin C$, $\{\ell_1, \dots, \ell_k, \overline{\ell}\} \in F$, and $\overline{\ell_1}, \dots, \overline{\ell_k} \prec \overline{\ell}$ in M.
Learn	$\dfrac{\langle M, F, C\rangle}{\langle M, F \cup \{C\}, C\rangle}$	where $C \neq \top$, $C \notin F$.
Back	$\dfrac{\langle M, F, \{\ell_1, \dots, \ell_k, \ell\}\rangle}{\langle M' \cdot \ell, F, \top\rangle}$	where $\{\ell_1, \dots, \ell_k, \ell\} \in F$, $M = M' \cdot \Box \cdots \overline{\ell} \cdots$, and $\overline{\ell_1}, \dots, \overline{\ell_k}$ in M'.

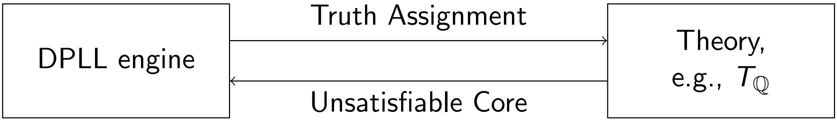
$P_1 \wedge (\neg P_2 \vee P_3) \wedge (\neg P_4 \vee P_3) \wedge (P_2 \vee P_4) \wedge (\neg P_1 \vee \neg P_4 \vee \neg P_3) \wedge (P_4 \vee \neg P_3)$

The algorithm starts with $M = \epsilon$, $C = \top$ and
$F = \{\{P_1\}, \{\bar{P}_2, P_3\}, \{\bar{P}_4, P_3\}, \{P_2, P_4\}, \{\bar{P}_1, \bar{P}_4, \bar{P}_3\}, \{P_4, \bar{P}_3\}\}$.

$\langle \epsilon, F, \top \rangle \xrightarrow{\text{Propagate}} \langle P_1, F, \top \rangle \xrightarrow{\text{Decide}} \langle P_1 \square \bar{P}_2, F, \top \rangle \xrightarrow{\text{Propagate}}$
$\langle P_1 \square \bar{P}_2 P_4, F, \top \rangle \xrightarrow{\text{Propagate}} \langle P_1 \square \bar{P}_2 P_4 P_3, F, \top \rangle \xrightarrow{\text{Conflict}}$
$\langle P_1 \square \bar{P}_2 P_4 P_3, F, \{\bar{P}_1, \bar{P}_4, \bar{P}_3\} \rangle \xrightarrow{\text{Explain}} \langle P_1 \square \bar{P}_2 P_4 P_3, F, \{\bar{P}_1, \bar{P}_4\} \rangle \xrightarrow{\text{Learn}}$
$\langle P_1 \square \bar{P}_2 P_4 P_3, F', \{\bar{P}_1, \bar{P}_4\} \rangle \xrightarrow{\text{Back}} \langle P_1 \bar{P}_4, F', \top \rangle \xrightarrow{\text{Propagate}}$
$\langle P_1 \bar{P}_4 P_2 P_3, F', \top \rangle \xrightarrow{\text{Conflict}} \langle P_1 \bar{P}_4 P_2 P_3, F', \{P_4, \bar{P}_3\} \rangle \xrightarrow{\text{Explain}}$
$\langle P_1 \bar{P}_4 P_2 P_3, F', \{P_4, \bar{P}_2\} \rangle \xrightarrow{\text{Explain}} \langle P_1 \bar{P}_4 P_2 P_3, F', \{P_4\} \rangle \xrightarrow{\text{Explain}}$
$\langle P_1 \bar{P}_4 P_2 P_3, F', \{\bar{P}_1\} \rangle \xrightarrow{\text{Explain}} \langle P_1 \bar{P}_4 P_2 P_3, F', \emptyset \rangle \xrightarrow{\text{Learn}}$
$\langle P_1 \bar{P}_4 P_2 P_3, F' \cup \{\emptyset\}, \emptyset \rangle$

where $F' = F \cup \{\{\bar{P}_1, \bar{P}_4\}\}$.

The DPLL/CDCL algorithm is combined with a Decision Procedures for a Theory



DPLL takes the propositional core of a formula,
assigns truth-values to atoms.

Theory takes a conjunctive formula (conjunction of literals),
returns a minimal unsatisfiable core.

Minimal Unsatisfiable Core

Suppose we have a decision procedure for a conjunctive theory, e.g., Simplex Algorithm for $T_{\mathbb{Q}}$.

Given an unsatisfiable conjunction of literals $\ell_1 \land \cdots \land \ell_n$.
Find a subset $\text{UnsatCore} = \{\ell_{i_1}, \ldots, \ell_{i_m}\}$, such that

- $\ell_{i_1} \land \ldots \land \ell_{i_m}$ is unsatisfiable.
- For each subset of UnsatCore the conjunction is satisfiable.

Possible approach: check for each literal whether it can be omitted.
$\longrightarrow$ n calls to decision procedure.

Most decision procedures can give small unsatisfiable cores for free.

Unsatisfiable Core and Conflict Clause

Theory returns an unsatisfiable core:

- a conjunction of literals from current truth assignment
- that is unsatisfible.

DPLL learns conflict clauses, a disjunction of literals

- that are implied by the formula
- and in conflict to current truth assignment.

Thus the negation of an unsatisfiable core is a conflict clause.

The DPLL part only needs one new rule:

TConflict $\dfrac{\langle M, F, \top\rangle}{\langle M, F, C\rangle}$ where M is unsatisfiable in the theory and $\neg C$ an unsatisfiable core of M.

Example: DPLL(T)

$$F : \; y \geq 1 \wedge (x \geq 0 \rightarrow y \leq 0) \wedge (x \leq 1 \rightarrow y \leq 0)$$

Atomic propositions:

$$P_1 : y \geq 1 \qquad P_2 : x \geq 0$$
$$P_3 : y \leq 0 \qquad P_4 : x \leq 1$$

Propositional core of F in CNF:

$$F_0 : (P_1) \wedge (\neg P_2 \vee P_3) \wedge (\neg P_4 \vee P_3)$$

Running DPLL(T)

$F_0 \ : \ \{\{P_1\}, \{\bar{P}_2, P_3\}, \{\bar{P}_4, P_3\}\}$
$P_1 \ : \ y \geq 1 \quad P_2 \ : \ x \geq 0 \quad P_3 \ : \ y \leq 0 \quad P_4 \ : \ x \leq 1$

$$\begin{aligned}
&\langle \epsilon, F_0, \top\rangle \xrightarrow{\text{Propagate}} \langle P_1, F_0, \top\rangle \xrightarrow{\text{Decide}} \langle P_1 \Box P_3, F_0, \top\rangle \xrightarrow{\text{TConflict}} \\
&\langle P_1 \Box P_3, F_0, \{\bar{P}_1, \bar{P}_3\}\rangle \xrightarrow{\text{Learn}} \langle P_1 \Box P_3, F_1, \{\bar{P}_1, \bar{P}_3\}\rangle \xrightarrow{\text{Back}} \\
&\langle P_1 \bar{P}_3, F_1, \top\rangle \xrightarrow{\text{Propagate}} \langle P_1 \bar{P}_3 \bar{P}_2, F_1, \top\rangle \xrightarrow{\text{Propagate}} \\
&\langle P_1 \bar{P}_3 \bar{P}_2 \bar{P}_4, F_1, \top\rangle \xrightarrow{\text{TConflict}} \langle P_1 \bar{P}_3 \bar{P}_2 \bar{P}_4, F_1, \{P_2, P_4\}\rangle \xrightarrow{\text{Explain}} \\
&\langle P_1 \bar{P}_3 \bar{P}_2 \bar{P}_4, F_1, \{P_2, P_3\}\rangle \xrightarrow{\text{Explain}} \langle P_1 \bar{P}_3 \bar{P}_2 \bar{P}_4, F_1, \{P_3\}\rangle \xrightarrow{\text{Explain}} \\
&\langle P_1 \bar{P}_3 \bar{P}_2 \bar{P}_4, F_1, \{\bar{P}_1\}\rangle \xrightarrow{\text{Explain}} \langle P_1 \bar{P}_3 \bar{P}_2 \bar{P}_4, F_1, \emptyset\rangle \xrightarrow{\text{Learn}} \\
&\langle P_1 \bar{P}_3 \bar{P}_2 \bar{P}_4, F_1 \cup \{\emptyset\}, \emptyset\rangle
\end{aligned}$$

where $F_1 := F_0 \cup \{\{\bar{P}_1, \bar{P}_3\}\}$

No further step is possible; the formula F is unsatisfiable.

Correctness of DPLL(T)

Theorem (Correctness of DPLL(T))

Let F be a Σ-formula and F' its propositional core. Let

$$\langle \epsilon, F', \top \rangle = \langle M_0, F_0, C_0 \rangle \longrightarrow \ldots \longrightarrow \langle M_n, F_n, C_n \rangle$$

be a maximal sequence of rule application of DPLL(T).

Then F is T-satisfiable iff C_n is $\top$.

Before proving the theorem, we note some important invariants:

- M_i never contains a literal more than once.
- M_i never contains ℓ and $\overline{\ell}$.
- Every $\square$ in M_i is followed immediately by a literal.
- If $C_i = \{\ell_1, \ldots, \ell_k\}$ then $\overline{\ell_1}, \ldots, \overline{\ell_k}$ in M.
- C_i is always implied by F_i (or the theory).
- F is equivalent to F_i for all steps i of the computation.
- If a literal ℓ in M is not immediately preceded by $\square$, then F contains a clause $\{\ell, \ell_1, \ldots, \ell_k\}$ and $\overline{\ell_1}, \ldots, \overline{\ell_k} \prec \ell$ in M.

Correctness proof

Proof: If the sequence ends with $\langle M_n, F_n, \top \rangle$ and there is no rule applicable, then:

- Since Decide is not applicable, all literals of F_n appear in M_n either positively or negatively.
- Since Conflict is not applicable, for each clause at least one literal appears in M_n positively.
- Since TConflict is not applicable, the conjunction of truth assignments of M_n is satisfiable by a model I.

Thus, I is a model for F_n, which is equivalent to F.

If the sequence ends with $\langle M_n, F_n, C_n \rangle$ with $C_n \neq \top$.
Assume $C_n = \{\ell_1, \dots, \ell_k, \ell\} \neq \emptyset$. W.l.o.g., $\bar{\ell}_1, \dots, \bar{\ell}_k \prec \bar{\ell}$. Then:

- Since Learn is not applicable, $C_n \in F_n$.
- Since Explain is not applicable $\bar{\ell}$ must be immediately preceded by $\Box$.
- However, then Back is applicable, contradiction!

Therefore, the assumption was wrong and $C_n = \emptyset (= \bot)$.
Since F implies C_n, F is not satisfiable.

Total Correctness of DPLL with Learning

Theorem (Termination of DPLL)

Let F be a propositional formula. Then every sequence

$$\langle \epsilon, F, \top \rangle = \langle M_0, F_0, C_0 \rangle \longrightarrow \langle M_1, F_1, C_1 \rangle \longrightarrow \dots$$

terminates.

Proof of Total Correctness

We define some well-ordering on the domains:

- We define $M \prec M'$ if $M\square\square$ comes lexicographically before $M'\square\square$, where every literal is considered to be smaller than $\square$.
  Example: $\ell_1\ell_2(\square\square) \prec \ell_1\square\bar{\ell}_2\ell_3(\square\square) \prec \ell_1\square\bar{\ell}_2(\square\square) \prec \ell_1(\square\square)$
- For a sequence $M = \bar{\ell}_1 \dots \bar{\ell}_n$, the conflict clauses are ordered by:
  $C \prec_M C'$, iff $C \neq \top, C' = \top$ or for some $k \leq n$:
  $C \cap \{\ell_{k+1}, \dots, \ell_n\} = C' \cap \{\ell_{k+1}, \dots \ell_n\}$ and $\ell_k \notin C, \ell_k \in C'$.
  Example: $\emptyset \prec_{\bar{\ell}_1\bar{\ell}_2\bar{\ell}_3} \{\ell_2\} \prec_{\bar{\ell}_1\bar{\ell}_2\bar{\ell}_3} \{\ell_1, \ell_3\} \prec_{\bar{\ell}_1\bar{\ell}_2\bar{\ell}_3} \{\ell_2, \ell_3\} \prec_{\bar{\ell}_1\bar{\ell}_2\bar{\ell}_3} \top$

These are well-orderings, because the domains are finite.

Termination Proof: Every rule application decreases the value of $\langle M_i, F_i, C_i\rangle$ according to the well-ordering:

$$\langle M, F, C\rangle \prec \langle M', F', C'\rangle, \text{ iff } \begin{cases} M \prec M', \\ \text{or } M = M', C \prec_M C', \\ \text{or } M = M', C = C', C \in F, C \notin F'. \end{cases}$$

Program Correctness

Road Map

- So far: decision procedures to decide validity in theories
- In the next lectures: the "practical" part
- Application of decision procedures to program verification

The programming language pi

- pi is an imperative programming language.
- built-in program annotations in first order logic
- annotation F at location L asserts that F is true whenever program control reaches L

Program 1: LinearSearch

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

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

```
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 f is partially correct if
when f's precondition is satisfied on entry and f terminates,
then f's postcondition is satisfied.

- 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)

Loop invariants

- Each loop needs an annotation @L called loop invariant
- while loop: L must hold
  - at the beginning of each iteration before the loop condition is evaluated
- for loop: L must hold
  - after the loop initialization, and
  - before the loop condition is evaluated

Basic Paths: Loops

To handle loops, we break the function into basic paths.

@ $\leftarrow$ precondition or loop invariant

finite sequence of instructions
(with no loop invariants)

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

Basic Paths: Loops

A basic path:

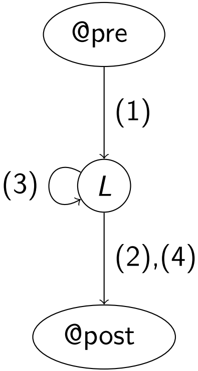
- begins at the function pre condition or a loop invariant,
- ends at an assertion, e.g., the loop invariant or the function post,
- does not contain the loop invariant inside the sequence,
- conditional branches are replaced by assume statements.

Assume statement c

- Remainder of basic path is executed only if c holds
- Guards with condition c split the path (assume(c) and assume($\neg c$))

Example: Basic Paths of LinearSearch

Visualization of basic paths of LinearSearch



Example: Basic Paths of LinearSearch

(1)

$@\mathrm{pre}\ 0 \leq \ell \land u < |a|$
$i := \ell;$
$@L:\ \ell \leq i \land \forall j.\ \ell \leq j < i \rightarrow a[j] \neq e$

(2)

$@L:\ \ell \leq i \land \forall j.\ \ell \leq j < i \rightarrow a[j] \neq e$
$\texttt{assume}\ i \leq u;$
$\texttt{assume}\ a[i] = e;$
$rv := \texttt{true};$
$@\mathrm{post}\ rv \leftrightarrow \exists j.\ \ell \leq j \leq u \land a[j] = e$

Example: Basic Paths of LinearSearch

(3)

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

(4)

$@L : \ell \leq i \wedge \forall j.\, \ell \leq j < i \rightarrow a[j] \neq e$
`assume` $i > u$;
$rv :=$ `false`;
@post $rv \leftrightarrow \exists j.\, \ell \leq j \leq u \wedge a[j] = e$

Proving Partial Correctness

Goal

- Prove that annotated function f agrees with annotations
- Therefore: Reduce f to finite set of verification conditions VC
- Validity of VC implies that function behaviour agrees with annotations

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

- Informally: What must hold before executing statement S to ensure that formula F holds afterwards?
- $\text{wp}(F,\ S) =$ weakest formula such that executing S results in formula that satisfies F
- For all states s such that $s \models \text{wp}(F,\ S)$: successor state $s' \models F$.

Computing weakest preconditions

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

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

Proving Partial Correctness

Proving partial correctness for programs with loops

- Input: Annotated program
- Produce all basic paths $P = \{p_1, \ldots, p_n\}$
- For all $p \in P$: generate verification condition $VC(p)$
- Check validity of $\bigwedge_{p \in P} VC(p)$

Theorem

If $\bigwedge_{p \in P} VC(p)$ is valid, then each function agrees with its annotation.

VC of 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.

(2)

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

The VC is: $F \to \mathrm{wp}(G,\ S_1; S_2; S_3)$

That is,

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

Therefore the VC of path (2)

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

or, equivalently,

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

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.

Tool Demo: PiVC

- Verifies pi programs
- Available at `http://cs.stanford.edu/people/jasonaue/pivc/`

Example 2: BinarySearch

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

sorted: weakly increasing order, i.e.

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

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 $0 \le \ell$ and $u < |a|$

```
@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);
  }
}
```

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

Program 3: BubbleSort

```
@pre ⊤
@post sorted(rv, 0, |rv| − 1)
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) {
      if (a[j] > a[j + 1]) {
        int t := a[j];
        a[j] := a[j + 1];
        a[j + 1] := t;
      }
    }
  }
  return a;
}
```

Example 3: BubbleSort

Function BubbleSort sorts integer array a

a:	unsorted	sorted

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.

$$
\begin{array}{ccccccc}
2 & 3 & 4 & 1 & 2 & \multicolumn{2}{c}{\boxed{5 \qquad 6}} \\
j & & & & i & & \\[1em]
2 & 3 & 4 & 1 & 2 & \multicolumn{2}{c}{\boxed{5 \qquad 6}} \\
 & j & & & i & & \\[1em]
2 & 3 & 4 & 1 & 2 & \multicolumn{2}{c}{\boxed{5 \qquad 6}} \\
 & & j & & i & & \\[1em]
2 & 3 & 1 & 4 & 2 & \multicolumn{2}{c}{\boxed{5 \qquad 6}} \\
 & & & j & i & & \\[1em]
2 & 3 & 1 & 2 & 4 & \multicolumn{2}{c}{\boxed{5 \qquad 6}} \\
 & & & & j, i & & \\[1em]
2 & 3 & 1 & 2 & \multicolumn{3}{c}{\boxed{4 \qquad 5 \qquad 6}} \\
j & & & i & & &
\end{array}
$$

BubbleSort with runtime assertions

```
@pre ⊤
@post ⊤
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) {
      @ 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;
}
```

BubbleSort with loop invariants

$$
\begin{array}{l}
@\mathrm{pre}\ \top \\
@\mathrm{post}\ \mathrm{sorted}(rv, 0, |rv| - 1) \\
\mathtt{int}[]\ \mathrm{BubbleSort}(\mathtt{int}[]\ a_0)\ \{ \\
\quad \mathtt{int}[]\ a := a_0; \\
\quad \mathtt{for} \\
\qquad @L_1 : \begin{bmatrix} -1 \le i < |a| \\ \land \mathrm{partitioned}(a, 0, i, i+1, |a|-1) \\ \land \mathrm{sorted}(a, i, |a|-1) \end{bmatrix} \\
\qquad (\mathtt{int}\ i := |a| - 1;\ i > 0;\ i := i - 1)\ \{
\end{array}
$$

```
    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

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

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

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

Basic Paths of BubbleSort

(1)

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

(2)

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

`assume` $i > 0$;

$j := 0$;

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

(3)

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

`assume` $j < i$;

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

$t := a[j]$;

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

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

$j := j+1$;

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

(4)

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

$\texttt{assume}\ j < i;$

$\texttt{assume}\ a[j] \le a[j+1];$

$j := j + 1;$

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

(5)

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

$\texttt{assume}\ j \ge i;$

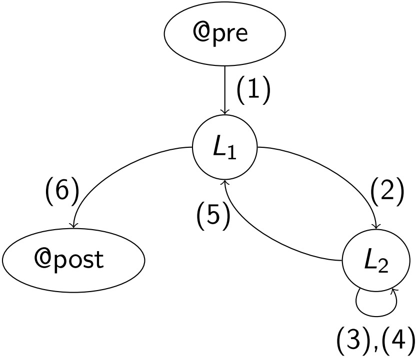
$i := i - 1;$

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

(6)

$@L_1 : \; -1 \le i < |a| \land \text{partitioned}(a, 0, i, i+1, |a|-1) \land$
$\quad \text{sorted}(a, i, |a|-1)$
`assume` $i \le 0$;
$rv := a$;
$@\text{post sorted}(rv, 0, |rv|-1)$

Visualization of basic paths of BubbleSort



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)

Total Correctness

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

Total Correctness = Partial Correctness + Termination

In the following, we focus on proving function termination. Therefore, we need the notion of well-founded relations and ranking functions.

Well-founded relation

Definition

For a set S, a binary relation $\prec$ is a well-founded relation iff there is no infinite sequence $s_1, s_2, s_3 \ldots$ of elements of S such that $s_1 \succ s_2 \succ s_3 \succ \cdots$, where $s \prec t$ iff $t \succ s$.

Example

$<$ is well-founded over $\mathbb{N}$. Decreasing sequences w.r.t. $<$ are always finite.
$123 > 98 > 42 > 11 > 7 > 2 > 0$

$<$ is not well-founded over $\mathbb{Q}$.
$1 > \frac{1}{2} > \frac{1}{3} > \frac{1}{4} > \cdots$

Proving function termination

- Choose set S with well-founded relation $\prec$
  Usually set of n-tuples of natural numbers with the lexicographic ordering.
- Find function δ such that
  - δ maps program states to S, and
  - δ decreases according to $\prec$ along every basic path.

  Such a function δ is called a ranking function.

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

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) ↦ (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);
  }
}
```

Proving function termination: Example

To prove function termination:

- 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.

The relevant basic paths are:

(1)

$\texttt{@}\mathsf{pre}\ x \geq 0 \wedge y \geq 0$
$\#\,(x, y)$
$\texttt{assume}\ x \neq 0;$
$\texttt{assume}\ y = 0;$
$\#\,(x - 1, 1)$

Proving function termination: Example

(2)

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

(3)

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

Proving function termination: Verification Condition

Showing decrease of ranking function

Basic path with ranking function:

$$\begin{array}{l} \texttt{@}\ F \\ \#\,\delta[\overline{x}] \\ S_1; \\ \vdots \\ S_n; \\ \#\,\kappa[\overline{x}] \end{array}$$

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

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

Example: Ackermann function — verification condition for basic path **(3)**

(3)

$@\text{pre } x \geq 0 \land y \geq 0$
$\#(x, y)$
$\texttt{assume } x \neq 0;$
$\texttt{assume } y \neq 0;$
$\texttt{assume } v_1 \geq 0;$
$z := v_1;$
$\#(x - 1, z)$

Verification condition:

$$x \geq 0 \land y \geq 0 \rightarrow$$
$$\text{wp}((x - 1, z) <_2 (x_0, y_0)$$
$$, \texttt{assume } x \neq 0; \texttt{assume } y \neq 0; \texttt{assume } v_1 \geq 0; z := v_1)$$

Computing the weakest precondition

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

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

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

We finally obtain the verification condition

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

Proving function termination: Verification Condition

Verification conditions for the three basic paths

1. $x \geq 0 \land y \geq 0 \land x \neq 0 \land y = 0 \rightarrow (x-1, 1) <_2 (x, y)$
2. $x \geq 0 \land y \geq 0 \land x \neq 0 \land y \neq 0 \rightarrow (x, y-1) <_2 (x, y)$
3. $x \geq 0 \land y \geq 0 \land x \neq 0 \land y \neq 0 \land v_1 \geq 0 \rightarrow (x-1, v_1) <_2 (x, y)$

Proving function termination: Example

BubbleSort — program with loops

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

$$
\begin{array}{l}
\texttt{@pre } \top \\
\texttt{@post } \top \\
\texttt{int[] BubbleSort(int[] } a_0\texttt{) \{} \\
\quad \texttt{int[] } a := a_0; \\
\quad \texttt{for} \\
\qquad @L_1 : \; i + 1 \geq 0 \\
\qquad \#(i+1,\; i+1) \qquad\qquad \ldots \text{ranking function } \delta_1 \\
\qquad (\texttt{int } i := |a| - 1;\; i > 0;\; i := i - 1)\ \{
\end{array}
$$

```
    for
      @L₂ :  i + 1 ≥ 0 ∧ i − j ≥ 0
      #(i + 1, i − j)               ... ranking function δ₂
      (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 that

- program is partially correct
- function decreases along each basic path.

The relevant basic paths

(1)

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

(2),**(3)**

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

(4)

$$
\begin{array}{l}
@L_2 : \ i+1 \geq 0 \wedge i-j \geq 0 \\
\#L_2 : \ (i+1, i-j) \\
\texttt{assume } j \geq i; \\
i := i-1; \\
\#L_1 : \ (i+1, i+1)
\end{array}
$$

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.

Summary

Specification and verification of sequential programs

- Programming language pi and the PiVC verifier
- Program specification
  - Program annotations as assertions
  - Including function preconditions, postconditions, loop invariants, ...
- Partial correctness
  - @pre + termination $\Rightarrow$ @post
  - Notion of weakest preconditions and verification conditions
- Total correctness
  - Additionally guarantees function termination
  - Notion of well-founded relations and ranking functions

Craig Interpolation

Given an unsatisfiable formula of the form:

$$F \land G$$

Can we find a "smaller" formula that explains the conflict?

I.e., a formula implied by F that is inconsistent with G?

Under certain conditions, there is an interpolant I with

- $F \Rightarrow I$.
- $I \land G$ is unsatisfiable.
- I contains only symbols common to F and G.

Craig Interpolation

A craig interpolant I for an unsatisfiable formula $F \land G$ is

- $F \Rightarrow I$.
- $I \land G$ is unsatisfiable.
- I contains only symbols common to F and G.

Craig interpolants exists in many theories and fragments:

- First-order logic.
- Quantifier-free FOL.
- Quantifier-free fragment of T_{E}.
- Quantifier-free fragment of $T_{\mathbb{Q}}$.
- Quantifier-free fragment of $\widehat{T_{\mathbb{Z}}}$ (augmented with divisibility).

However, QF fragment of $T_{\mathbb{Z}}$ does not allow Craig interpolation.

Program correctness

Consider this path through LINEARSEARCH:

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

$i := \ell$

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

$\texttt{assume}\ a[i] \neq e$

$i := i + 1$

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

$@\ 0 \leq i \wedge i < |a|$

Single Static Assignment (SSA) replaces assignments by assumes:

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

$\texttt{assume}\ i_1 = \ell$

$\texttt{assume}\ i_1 \leq u$

$\texttt{assume}\ a[i_1] \neq e$

$\texttt{assume}\ i_2 = i_1 + 1$

$\texttt{assume}\ i_2 \leq u$

$@\ 0 \leq i_2 \wedge i_2 < |a|$

Program correctness and Interpolants

If program contains only assumes, the VC looks like

$$VC \,:\, P \rightarrow (F_1 \rightarrow (F_2 \rightarrow (F_3 \rightarrow \ldots (F_n \rightarrow Q) \ldots)))$$

Using $\neg(F \rightarrow G) \Leftrightarrow F \wedge \neg G$ compute negation:

$$\neg VC \,:\, P \wedge F_1 \wedge F_2 \wedge F_3 \wedge \cdots \wedge F_n \wedge \neg Q$$

If verification condition is valid $\neg VC$ is unsatisfiable. We can compute interpolants for any program point, e.g. for

$$\color{red}{P \wedge F_1 \wedge F_2} \wedge \color{blue}{F_3 \wedge \cdots \wedge F_n \wedge \neg Q}$$

Verification Condition and Interpolants

Consider the path through LINEARSEARCH:

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

`assume` $i_1 = \ell$

`assume` $i_1 \leq u$

`assume` $a[i_1] \neq e$

`assume` $i_2 = i_1 + 1$

`assume` $i_2 \leq u$

$@\ 0 \leq i_2 \wedge i_2 < |a|$

The negated VC is unsatisfiable:

$$0 \leq \ell \wedge u < |a| \wedge i_1 = \ell$$
$$\wedge\ i_1 \leq u \wedge a[i_1] \neq e \wedge i_2 = i_1 + 1$$
$$\wedge\ i_2 \leq u \wedge (0 > i_2 \vee i_2 \geq |a|)$$

The interpolant I for the red and blue part is

$$i_1 \geq 0 \wedge u < |a|$$

This is actually the loop invariant needed to prove the assertion.

Computing Interpolants

Suppose $F_1 \wedge F_n \wedge G_1 \wedge G_n$
How can we compute an interpolant?

- The algorithm is dependent on the theory and the fragment.
- We will show an algorithm for
  - Quantifier-free conjunctive fragment of T_{E}.
  - Quantifier-free conjunctive fragment of $T_{\mathbb{Q}}$.

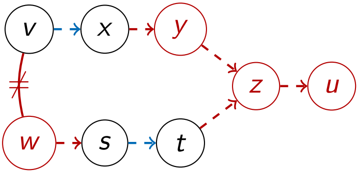
$$F_1 \wedge \cdots \wedge F_n \wedge G_1 \wedge \cdots \wedge G_n \text{ is unsat}$$

Let us first consider the case without function symbols.
The congruence closure algorithm returns unsat. Hence,

- there is a disequality $v \neq w$ and
- v,w have the same representative.

Example:

$v \neq w \wedge x = y \wedge y = z \wedge z = u \wedge w = s \wedge t = z \wedge s = t \wedge v = x$



The Interpolant "summarizes" the red edges: $I : v \neq s \wedge x = t$

Computing Interpolants for T_E

Given conjunctive formula:

$$F_1 \wedge \cdots \wedge F_n \wedge G_1 \wedge \cdots \wedge G_m$$

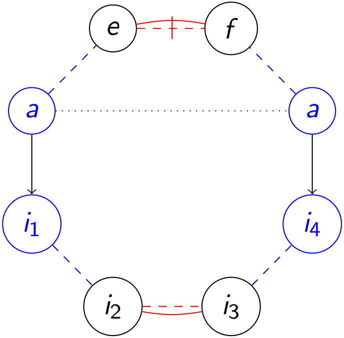
The following algorithm can be used unless there is a congruence edge:

- Build the congruence closure graph. Edges F_i are colored red, Edges G_j are colored blue.
- Add (colored) disequality edge. Find circle and remove all other edges.
- Combine maximal red paths, remove blue paths.
- The F paths start and end at shared symbols.
  Interpolant is the conjunction of the corresponding equalities.

Handling Congruence Edges (Case 1)

Both side of the congruence edge belong to G.

$$i_3 = i_2 \land e \neq f \land a(i_1) = e \land a(i_4) = f \land i_1 = i_2 \land i_3 = i_4$$

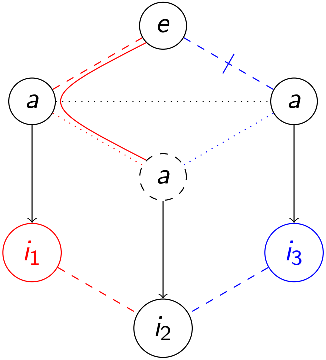


Interpolant:
$i_2 = i_3 \land e \neq f$

- Follow the path that connects the arguments.
- Also add summarized edges for that path.
- Treat the congruence edge as blue edge (ignore it).
- Interpolant is conjunction of all summarized paths.

Both side of the congruence edge belong to different formulas.

$$a(i_1) = e \land i_2 = i_1 \land i_3 = i_2 \land a(i_3) \neq e$$



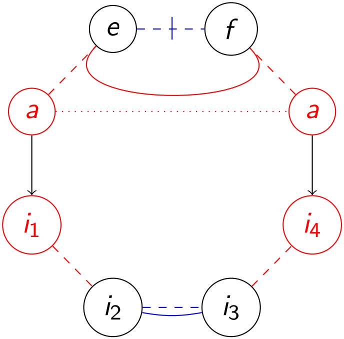
Interpolant: $e = a(i_2)$.

- Function symbol a must be shared.
- Follow the path that connects the arguments.
- Find first change from red to blue.
- Lift function application on that term.
- Summarize $e = a(i_1) \land i_1 = i_2$ by $e = a(i_2)$.
- Compute remaining interpolant as usual.

Handling Congruence Edges (Case 3)

Both side of the congruence edge belong to F.

$$a(i_1) = e \wedge a(i_4) = f \wedge i_1 = i_2 \wedge i_3 = i_4 \wedge i_3 = i_2 \wedge e \neq f$$



Interpolant:
$i_2 = i_3 \rightarrow e = f$

- Follow the path that connects the arguments.
- Find the first and last terms i_2, i_3 where color changes.
- Treat congruence edge as red edge and summarize path.
- The summary only holds under $i_2 = i_3$,i.e., add $i_2 = i_3 \rightarrow e = f$ to interpolants.
- Summarize remaining path segments as usual.

First apply Dutertre/de Moura algorithm.

- Non-basic variables $x_1, \ldots, x_n$.
- Basic variables $y_1, \ldots, y_m$.
- $y_i = \sum a_{ij} x_j$
- Conjunctive formula
  $y_1 \leq b_1 \ldots y_{m'} \leq b_{m'} \wedge y_{m'+1} \leq b_{m'+1} \ldots y_m \leq b_m$.

The algorithm returns unsatisfiable if and only if there is a line:

$$
\begin{array}{c|ccccccccc}
 & x & \cdots & x & y & \cdots & y & y & \cdots & y \\
\hline
\vdots & & & & & & & & & \\
y_i/y_i & 0 & \cdots & 0 & -/0 & \cdots & -/0 & -/0 & \cdots & -/0 \\
\vdots & & & & & & & & &
\end{array}
$$

$y_i = \sum -a'_k y_k$, $a'_k \geq 0$ and $\sum -a'_k b_k > b_i$
(the constraint $y_i \leq b_i$ is not satisfied)

Computing Interpolants for $T_{\mathbb{Q}}$

The conflict is:

$$b_i \geq y_i = \sum -a'_k y_k \geq \sum -a'_k b_k > b_i$$

or

$$0 = y_i + \sum a'_k y_k \leq b_i + \sum a'_k b_k < 0$$

We split the y variables into blue and red ones:

$$0 = \sum_{k=1}^{m'} a_{ik} y_k + \sum_{k=m'+1}^{m} a_{ik} y_k \leq \sum_{k=1}^{m'} a_{ik} b_k + \sum_{k=m'+1}^{m} a_{ik} b_k < 0$$

where $a'_k \geq 0, (a'_i = 1)$. The interpolant I is the red part:

$$\sum_{k=1}^{m'} a_{ik} y_k \leq \sum_{k=1}^{m'} a_{ik} b_k$$

where the basic variables y_k are replaced by their definition.

Example

$$x_1 + x_2 \leq 3 \wedge x_1 - x_2 \leq 1 \wedge x_3 - x_1 \leq 1 \wedge x_3 \geq 4$$

$$y_1 := x_1 + x_2 \qquad b_1 := 3 \qquad y_3 := -x_1 + x_3 \qquad b_3 := 1$$
$$y_2 := x_1 - x_2 \qquad b_1 := 1 \qquad y_4 := -x_3 \qquad b_4 := -4$$

Algorithm ends with the tableaux

	1	1	-4	
	y_2	y_3	y_4	β
y_1	-1	-2	-2	5
x_1	0	-1	-1	3
x_2	-1	-1	-1	2
x_3	0	0	-1	4

Conflict is $0 = y_1 + y_2 + 2y_3 + 2y_4 \leq 3 + 1 + 2 - 8 = -2$.
Interpolant is: $y_1 + y_2 \leq 3 + 1$
or (substituting non-basic vars): $2x_1 \leq 4$.

Correctness

$$F_k : y_k := \sum_{j=0}^{n} a_{kj} x_j \le b_k, {\scriptstyle (k=1,\dots,m)} \qquad G_k : y_k := \sum_{j=0}^{n} a_{kj} x_j \le b_k, {\scriptstyle (k=m',\dots,m)}$$

Conflict is $0 = \sum_{k=1}^{m'} a'_k y_k + \sum_{k=m'+1}^{m} a'_k y_k \le \sum_{k=1}^{m'} a'_k b_k + \sum_{k=m'+1}^{m} a'_k b_k < 0$

After substitution the red part $\sum_{k=1}^{m'} a'_k y_k \le \sum_{k=1}^{m'} a'_k b_k$ becomes

$$I : \sum_{j=1}^{n} \left(\sum_{k=1}^{m'} a'_k a_{kj} \right) x_j \le \sum_{k=1}^{m'} a'_k b_k.$$

- $F \Rightarrow I$ (sum up the inequalities in F with factors a'_k).
- $I \wedge G \Rightarrow \perp$ (sum up I and G with factors a'_k to get $0 \le \sum_{k=1}^{m} a'_k b_k < 0$).
- Only shared symbols in I: $0 = \sum_{k=1}^{m'} a_{kj} a'_k x_j + \sum_{k=m'+1}^{m} a_{kj} a'_k x_j$.
  If the left sum is not zero, the right sum is not zero and x_j appears in F and G.

A proof of unsatisfiability is a resolution tree:

$$\dfrac{\dfrac{\dfrac{r \lor p \qquad \dfrac{s \lor \overline{r} \lor q \quad \overline{s} \lor q}{\overline{r} \lor q}}{q \lor p} \qquad \overline{q} \lor p}{p} \qquad \dfrac{\dfrac{r \lor s \qquad \dfrac{q \lor \overline{r} \lor s \quad \overline{q} \lor \overline{r}}{\overline{r} \lor s}}{s} \qquad \overline{s} \lor \overline{p}}{\overline{p}}}{\bot}$$

where each node is generated by the rule

$$\frac{\ell \lor C_1 \qquad \overline{\ell} \lor C_2}{C_1 \lor C_2}$$

- The leaves are (trivial) consequences of $F \land G$.
- Therefore, every node is a consequence.
- Therefore, the root node $\bot$ is a consequence.

Interpolants for Conflict Clauses

Key Idea: Compute Interpolants for conflict clauses:
Split C into C_F and C_G (if literal appear in F and G put it in C_G).

The conflict clause follows from the original formula:

$$F \land G \Rightarrow C_F \lor C_G$$

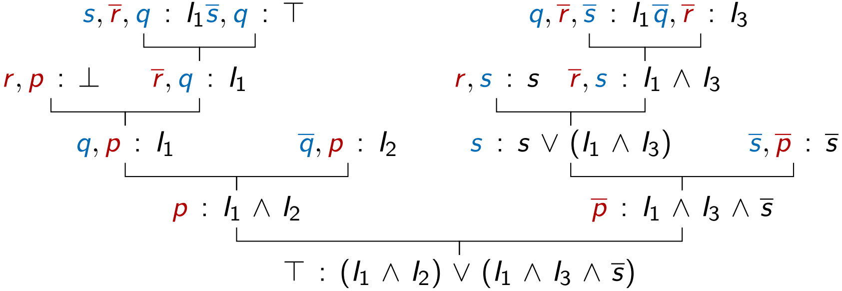
Hence, the following formula is unsatisfiable.

$$F \land \neg C_F \land G \land \neg C_G$$

An interpolant I_C for C is the interpolant of the above formula. I_C contains only symbols shared between F and G.

McMillan's algorithm

Assign all literals to either F or G.



Compute interpolants for the leaves.
Then, for every resolution step compute interpolant as

$$\frac{\overline{\ell}_F \wedge \overline{C_1} \,:\, I_1 \qquad \ell_F \wedge \overline{C_2} \,:\, I_2}{\overline{C_1} \wedge \overline{C_2} \,:\, I_1 \vee I_2} \qquad\qquad \frac{\overline{\ell}_G \wedge \overline{C_1} \,:\, I_1 \qquad \ell_G \wedge \overline{C_2} \,:\, I_2}{\overline{C_1} \wedge \overline{C_2} \,:\, I_1 \wedge I_2}$$

Computing Interpolants for Conflict Clauses

There are several points where conflict clauses are returned:

- Conflict clauses is returned by TCHECK.
  Then theory must give an interpolant.
- Conflict clauses comes from F.
  Then $F \Rightarrow C_F \vee C_G$.
  Hence, $(F \wedge \neg C_F) \Rightarrow C_G$. Also, $C_G \wedge G \wedge \neg C_G$ is unsatisfiable
  Interpolant is C_G.
- Conflict clauses comes from G.
  Then $C_G = C$, $G \Rightarrow C_G$.
  Hence, $(G \wedge \neg C_G)$ is unsatisfiable. Interpolant is $\top$.
- Conflict clause comes from resolution on ℓ.
  Then there is a unit clause $U = \ell \vee U'$ with interpolant I_U
  and conflict clause $C = \neg\ell \vee C'$ with interpolant I_C.

  If $\ell \in F$, set $I_{U' \vee C'} = I_U \vee I_C$
  If $\ell \in G$, set $I_{U' \vee C'} = I_U \wedge I_C$

The previous algorithm can compute interpolant for each conflict clause.
The final conflict clause returned is $\perp$.
$I_\perp$ is an interpolant of $F \land G$.

Computing Interpolants for Theory Combinations

Unfortunately, it is not that easy...
... because equalities shared by Nelson-Oppen can contain red and blue symbols simultaneously.

Example:

$$F\colon\ t \leq 2a \land 2a \leq s \land f(a) = q$$
$$G\colon\ s \leq 2b \land 2b \leq t \land f(b) \neq q$$

Nelson-Oppen proof

Purifying the example gives:

$$\Gamma_E : f(a) = q \wedge f(b) \neq q$$
$$\Gamma_{\mathbb{Q}} : t \leq 2a \wedge 2a \leq s \wedge s \leq 2b \wedge 2b \leq t$$

Shared variables $V = \{a, b\}$
Nelson-Oppen proceeds as follows

1. $\Gamma_{\mathbb{Q}}$ propagates $a = b$.
2. $\Gamma_E \cup a = b$ is unsatisfiable.

$$\Gamma_E : f(a) = q \land f(b) \neq q$$
$$\Gamma_{\mathbb{Q}} : t \leq 2a \land 2a \leq s \land s \leq 2b \land 2b \leq t$$

N-O introduces three literals: $a = b$, $a \leq b$, $a \geq b$.
Theory conflicts:

$$2b \leq t \land t \leq 2a \land \neg(b \leq a)$$
$$2a \leq s \land s \leq 2b \land \neg(a \leq b)$$
$$a \leq b \land b \leq a \land a \neq b$$
$$a = b \land f(a) = q \land f(b) \neq q$$

How can we compute interpolants for the conflicts?

Interpolant with $a = b$

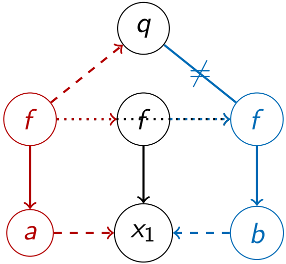
What is an interpolant of $a = b \wedge f(a) = q \wedge f(b) \neq q$?

Key Idea: Split

$$a = b$$

into

$$a = x_1 \wedge x_1 = b \text{ where } x_1 \text{ shared}$$



$a = x_1 \wedge f(a) = q \wedge$
$x_1 = b \wedge f(b) \neq q$

Interpolant: $f(x_1) = q$

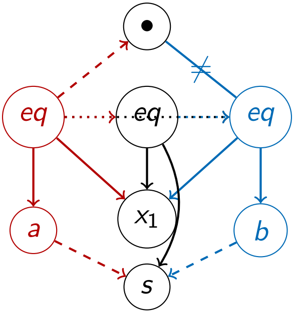
What is an interpolant of $a \neq b \wedge a = s \wedge b = s$?

Key Idea: Split

$$a \neq b$$

into

$eq(x_1, a) \wedge \neg eq(x_1, b)$ where x_1 shared, eq a predicate



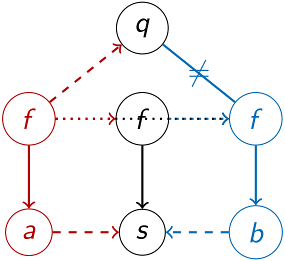
$eq(x_1, a) = \bullet \wedge a = s \wedge$
$eq(x_1, b) \neq \bullet \wedge b = s$

Interpolant: $eq(x_1, s)$

Consider the resolution step

$$\frac{a = b \lor a \neq s \lor b \neq s \qquad a \neq b \lor f(a) \neq q \lor f(b) = q}{f(a) \neq q \lor f(b) = q \lor a \neq s \lor b \neq s}$$

How to combine the interpolants $eq(x_1, s)$ and $f(x_1) = q$?



$f(a) = q \land a = s \land$
$f(b) \neq q \land s = b$

Interpolant: $f(s) = q$

$eq(x_1, s)$ indicates that x_1 should be replaced by s.

Resolution rule for $a = b$

The interpolation rule is

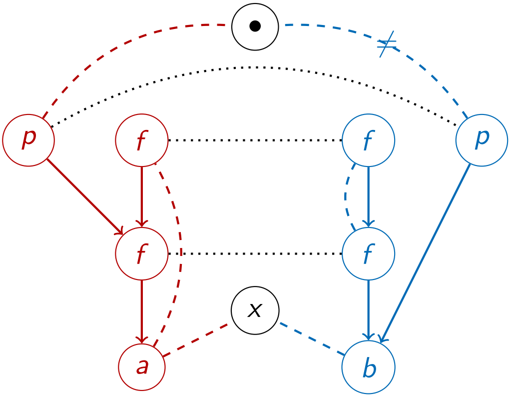
$$\frac{a = b \lor C_1 : I_1[eq(x, s_1)] \ldots [eq(x, s_n)] \qquad a \neq b \lor C_2 : I_2(x)}{C_1 \lor C_2 : I_1[I_2(s_1)] \ldots [I_2(s_n)]}$$

In our example

$$\frac{\begin{array}{c}\neg(a \neq b \land a = s \land b = s) : eq(x_1, s) \\ \neg(a = b \land f(a) = q \land f(b) \neq q) : q = f(x_1)\end{array}}{\neg(f(a) = q \land f(b) \neq q \land a = s \land b = s) : q = f(s)}$$

Example

$$a = f(f(a)) \land a = x \land p(f(a)) \land b = x \land f(b) = f(f(b)) \land \neg p(b)$$



Example: Proof Lemmas

$$a = f(f(a)) \land a = x \land p(f(a)) \land b = x \land f(b) = f(f(b)) \land \neg p(b)$$

Prove using the following lemmas:

$$
\begin{array}{rl}
F_1 : & a = x \land x = b \to f(a) =_{x_1} f(b) : eq(x_1, f(x)) \\
F_2 : & f(a) =_{x_1} f(b) \to f(f(a)) =_{x_2} f(f(b)) : eq(x_2, f(x_1)) \\
F_3 : & f(a) =_{x_1} f(b) = f(f(b)) =_{x_2} \\
 & f(f(a)) = a = x = b \to f(a) =_{x_3} b : eq(x_3, x_1) \land x_2 = x \\
F_4 : & f(a) =_{x_3} b \land p(f(a)) \to p(b) : p(x_3)
\end{array}
$$

Example: Annotating Proof with Interpolants

$$F_1 : \quad a = x \land x = b \to f(a) =_{x_1} f(b) \colon eq(x_1, f(x))$$

$$F_2 : \quad f(a) =_{x_1} f(b) \to f(f(a)) =_{x_2} f(f(b)) \colon eq(x_2, f(x_1))$$

$$F_3 : \quad f(a) =_{x_1} f(b) = f(f(b)) =_{x_2} f(f(a)) = a = x = b \to f(a) =_{x_3} b \colon eq(x_3, x_1) \land x_2 = x$$

$$F_4 : \quad f(a) =_{x_3} b \land p(f(a)) \to p(b) \colon p(x_3)$$

$$\dfrac{\dfrac{F_1 : eq(x_1, f(x)) \qquad \dfrac{F_2 : eq(x_2, f(x_1)) \qquad F_3 : eq(x_3, x_1) \land x_2 = x}{eq(x_3, x_1) \land f(x_1) = x}}{eq(x_3, f(x)) \land f(f(x)) = x} \qquad F_4 : p(x_3)}{p(f(x)) \land f(f(x)) = x}$$

Example: Checking Interpolants

$$a = f(f(a)) \land a = x \land p(f(a)) \land b = x \land f(b) = f(f(b)) \land \neg p(b)$$

Interpolant: $p(f(x)) \land f(f(x)) = x$

- $F \rightarrow I$: Substitute $a = x$ into other atoms.
- $I \land G \rightarrow \bot$: $b = x \land f(f(x)) = x \land \neg p(b)$ implies $\neg p(f(f(x)))$.
  With $b = x$, $f(b) = f(f(b))$ this implies $\neg p(f(x))$.
  This contradicts $p(f(x))$.
- Symbol condition: p, f, x are shared.

Back to the Nelson–Oppen Example

$$\Gamma_E : f(a) = q \land f(b) \neq q$$
$$\Gamma_{\mathbb{Q}} : t \leq 2a \land 2a \leq s \land s \leq 2b \land 2b \leq t$$

Theory conflicts:

$$2b \leq t \land t \leq 2a \land \neg(b \leq a)$$
$$2a \leq s \land s \leq 2b \land \neg(a \leq b)$$
$$a \leq b \land b \leq a \land a \neq b$$
$$a = b \land f(a) = q \land f(b) \neq q$$

How can we compute interpolants for the conflicts?

Interpolant with $a > b$

What is an interpolant of $2a \leq s \wedge s \leq 2b \wedge a > b$

Split

$$a > b$$

into

$$a \geq x_1 \wedge x_1 > a \text{ where } x_1 \text{ shared}$$

$$\begin{array}{rcl} 2a - s \leq 0 & & \cdot 1 \\ s - 2b \leq 0 & & \cdot 1 \\ x_1 - a \leq 0 & & \cdot 2 \\ b - x_1 < 0 & & \cdot 2 \\ \hline 0 < 0 & & \end{array} \qquad \begin{array}{rcl} 2a - s \leq 0 & & \cdot 1 \\ x_1 - a \leq 0 & & \cdot 2 \\ \hline 2x_1 - s \leq 0 & & \end{array}$$

Interpolant: $2x_1 - s \leq 0$.
We need the term $2x_1 - s$ later; we write interpolant as:

$$LA(2x_1 - s, 2x_1 - s \leq 0)$$

Interpolant with $a < b$

What is an interpolant of $t \leq 2a \wedge 2b \leq t \wedge a < b$

Split

$$a < b$$

into

$$a \leq x_2 \wedge x_2 < b \text{ where } x_2 \text{ shared}$$

$$\begin{array}{rcll} t - 2a & \leq & 0 & \cdot 1 \\ 2b - t & \leq & 0 & \cdot 1 \\ a - x_2 & \leq & 0 & \cdot 2 \\ x_2 - b & < & 0 & \cdot 2 \\ \hline 0 & < & 0 & \end{array} \qquad \begin{array}{rcll} t - 2a & \leq & 0 & \cdot 1 \\ a - x_2 & \leq & 0 & \cdot 2 \\ \hline t - 2x_2 & \leq & 0 & \end{array}$$

Interpolant: $t - 2x_2 \leq 0$.
We need the term $t - 2x_2$ later; we write interpolant as:

$$LA(t - 2x_2, t - 2x_2 \leq 0)$$

What is an interpolant of $a \leq b \wedge b \leq a \wedge a \neq b$

$$a \leq x_1 \wedge x_2 \leq a \wedge eq(x_3, a) \wedge x_1 \leq b \wedge b \leq x_2 \wedge \neg eq(x_3, b)$$

Manually we find the interpolant

$$x_2 - x_1 < 0 \vee (x_2 - x_1 \leq 0 \wedge eq(x_3, x_2))$$

Here $x_2 - x_1$ is the "critical term"; Interpolant:

$$LA(x_2 - x_1, x_2 - x_1 < 0 \vee (x_2 - x_1 \leq 0 \wedge eq(x_3, x_2)))$$

Magic rule:

$$\frac{a \leq b \vee C_1 : LA(s_1 + c_1 x_1, F_1(x_1)) \quad a > b \vee C_2 : LA(s_2 - c_2 x_1, F_2(x_2))}{C_1 \vee C_2 : LA(c_2 s_1 + c_1 s_2, c_2 s_1 + c_1 s_2 < 0 \vee (F_1(s_2/c_2) \wedge F_2(s_2/c_2)))}$$

Example:

$$\frac{\begin{array}{r} a \leq b \vee 2a > s \vee s > 2b : LA(2x_1 - s, 2x_1 - s \leq 0) \\ a > b \vee a < b \vee a = b : LA(x_2 - x_1, x_2 - x_1 < 0 \vee \\ (x_2 - x_1 \leq 0 \wedge eq(x_3, x_2))) \end{array}}{a < b \vee a = b \vee 2a > s \vee s > 2b : I_3}$$

$$I_3 : LA(2x_2 - s, 2x_2 - s < 0 \vee (2x_2 - s \leq 0 \wedge eq(x_3, x_2)))`$$

(simplifying $x_2 < x_2$ to $\perp$ and $x_2 \leq x_2$ to $\top$).

Example continued

Magic rule:

$$\frac{a \le b \vee C_1 : LA(s_1 + c_1 x_1, F_1(x_1)) \quad a > b \vee C_2 : LA(s_2 - c_2 x_1, F_2(x_2))}{C_1 \vee C_2 : LA(c_2 s_1 + c_1 s_2, c_2 s_1 + c_1 s_2 < 0 \vee (F_1(s_2/c_2) \wedge F_2(s_2/c_2)))}$$

$$\frac{\begin{array}{r} a < b \vee a = b \vee 2a > s \vee s > 2b : LA(2x_2 - s, 2x_2 - s < 0 \vee \\ (2x_2 - s \le 0 \wedge eq(x_3, x_2))) \\ a \ge b \vee t < 2a \vee 2b < s : LA(t - 2x_1, t - 2x_1 \le 0) \end{array}}{\begin{array}{l} a = b \vee 2a > s \vee s > 2b \\ \quad \vee\, t > 2a \vee t > 2b : I_4 \end{array}}$$

$$I_4 : LA(t - s, t - s < 0 \vee (t - s \le 0 \wedge eq(x_3, t/2)))$$

The critical term $t - s$ does not contain an auxiliary and can be removed.

$$I_4 : t - s < 0 \vee (t - s \le 0 \wedge eq(x_3, t/2))$$

Example continued (with equality)

$$\frac{\begin{array}{rcl} a = b \lor 2a > s \lor s > 2b & & t - s < 0 \lor \\ \lor\, t > 2a \lor t > 2b & : & (t - s \leq 0 \land eq(x_3, t/2)) \\ a \neq b \lor f(a) \neq q \lor f(b) = q & : & q = f(x_3) \end{array}}{\begin{array}{rcl} 2a > s \lor s > 2b & & t - s < 0 \lor \\ \lor\, t > 2a \lor t > 2b & : & (t - s \leq 0 \land q = f(t/2)) \\ \lor\, f(a) \neq q \lor f(b) = q & & \end{array}}$$

The interpolant of

$$2a \leq s \land t \leq 2a \land f(a) = q \land s \leq 2b \land 2b \leq t \land f(b) \neq q$$

is

$$t - s < 0 \lor (t - s \leq 0 \land q = f(t/2))$$

Conclusion

Topics

Topics
Propositional Logic
First-Order Logic
First-Order Theories
Quantifier Elimination for $T_{\mathbb{Z}}$ and $T_{\mathbb{Q}}$
Congruence Closure Algorithm (T_{E}, T_{cons}, T_{A})
Dutertre–de Moura Algorithm ($T_{\mathbb{Q}}$)
DP for Array Property Fragment
Nelson-Oppen
DPLL(T) with Learning
Program Correctness
Interpolation

Logics

PL	Propostional Logic
FOL	First-Order Logic
T_x	Theories

T_{E}

T_{PA}

$T_{\mathbb{N}}$

$T_{\mathbb{Z}}$

$T_{\mathbb{Q}}$

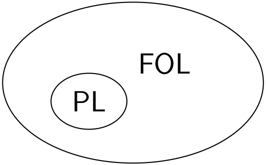
$T_{\mathbb{R}}$

T_{A}

$T_{\mathsf{A}}^{\mathbb{Z}}$

T_{RDS}

T_{cons}



Theories and their DPs

Theory	Full	Array Prop.	Quant. free	Conj. quant. free
T_{E}	✘	-	✔ (19–20)	✔ (13)
T_{PA}	✘	-	✘	✘
$T_{\mathbb{Z}}$	✔ (10)	-	✔	✔
$T_{\mathbb{Q}}$	✔ (9)	-	✔ (19–20)	✔ (11–12)
$T_{\mathbb{R}}$	✔ (-)	-	✔	✔
T_{A}	✘	✔ (15–16)	✔	✔ (15)
$T_{\mathsf{A}}^{\mathbb{Z}}$	✘	✔ (15–16)	✔	✔
T_{RDS}	✘	-	✔	✔
T_{cons}	✘	-	✔	✔ (14)
$T_1 \cup T_2$	-	-	✔ (-)	✔ (17–18)

- What is an atom, a literal, a formula.
- What is an interpretation?
- What does $I \models F$ mean, how do we compute it.
- What is satisfiability, validity.
- What is the duality between satisfiable and valid?
- What is the semantic argument?
- Write down the proof rules.
- How can we prove $P \wedge Q \rightarrow P \vee \neg Q$?
- What is $\Leftrightarrow$ (equivalent) and $\Rightarrow$ (implies).
- What Normal Forms do you know (NNF, DNF, CNF)?
- How to convert formulae into normal form.

DPLL for Propositional Logic

- What is a Decision Procedure?
- What is equisatisfiability; why is it useful?
- How to convert to CNF with polynomial time complexity?
- What is a clause?
- What does DPLL stand for?
- What is Boolean Constraint Propagation (BCP) (aka. Unit Propagation).
- What is Pure Literal Propagation (PL).
- Why is the DPLL algorithm correct?
- What is the worst case time complexity of DPLL?

First-Order Logic

- What is a variable, a constant, a function (symbol), a predicate (symbol), a term, an atom, a literal, a formula?
- How do first-order logic and predicate logic relate?
- What is an interpretation in FOL?
- Why is D_I non-empty?
- What does α_I assign?
- What is an x-variant of an interpretation?
- How do we compute whether $I \models F$?
- What is satisfiability, validity?
- What are the additional rules in the Semantic Argument (version of lecture 4)?
- Soundness and Completeness of semantic argument.
- What is a Hintikka set?
- Normal forms. What is PNF (prenex normal form)?
- Is validity for FOL decidable?

First-Order Theories

- What is a theory?
- What is a signature Σ?
- What do T-valid and emph T-satisfiable mean?
- What is T-equivalent?
- What is a decision procedure for a theory?
- What is a fragment of a theory?
- What are the most common fragments (quantifier-free, conjunctive)?
- What theories do you know?
- What are their axioms?
- What fragments of these theories are decidable?
- Bonus Question: Is there any closed formulae in T_{PA} that is satisfiable but not valid? What about $T_{\mathbb{Z}}$, $T_{\mathbb{Q}}$?

Quantifier Elimination

- What is Quantifier Elimination?
- Does $T_{\mathbb{Z}}$ admit quantifier elimination? What does it mean?
- Why is it enough to eliminate one existential quantifiers over a quantifier-free formula?
- How can we eliminate more than one quantifier?
- What is $\widehat{T_{\mathbb{Z}}}$?
- What is Cooper's method?
- What is Ferrante and Rackoff's method ($T_{\mathbb{Q}}$)?
- What is the Array Property Fragment?
- What do all quantifier elimination methods of the lecture have in common?
- What is the complexity of quantifier elimination?
- Why is quantifier elimination a decision procedure?

Simplex Based Algorithm

- Which theory does the Algorithm of Dutertre and de Moura decide?
- How does the algorithm work?
- How can we convert an arbitrary formula to the required format for the algorithm?
- What is the tableaux?
- What is a pivot step?
- Does the algorithm terminate?
- What is the complexity?

Congruence Closure Algorithm

- What is the congruence closure algorithm?
- How does it work for T_{E}?
- What are the data structures; what are the operations?
- What complexity does the algorithm have?
- What are the extensions for T_{cons}?
- What is the complexity?
- How did we prove correctness of the decision procedure?

- How does the DP for quantifier-free fragment of T_A work?
- What is the complexity?
- What is $T_\mathsf{A}^{=}$?

Array Property Fragment

- What is the Array Property Fragment of $T_{\mathsf{A}} / T_{\mathsf{A}}^{=}$?
- Why are there so many restrictions?
- What are the transformation steps?
- How are quantifier eliminated?
- What is λ and why is it necessary?
- Why is the decision procedure correct?
- What is the Array Property Fragment of $T_{\mathsf{A}}^{\mathbb{Z}}$?
- What are differences to T_{A}?
- Why do we not need λ for $T_{\mathsf{A}}^{\mathbb{Z}}$?
- Why is the decision procedure correct?
- How can we check this fragment?

- What is the Nelson-Oppen procedure?
- For what theories does it work? For which fragment of the theory?
- What is a stably infinite theory?
- Why is it important that theories are stably infinite?
- What are the two phases of Nelson-Oppen?
- What is the difference between the non-deterministic and deterministic variant of Nelson-Oppen?
- What is a convex theory?
- What is the emphcomplexity of the deterministic version for convex/non-convex theories?

DPLL(T)

- How can we extend the DPLL algorithm to decide T-satisfiability.
- What is a minimal unsatisfiable core?
- How can we compute it efficiently?
- What is the relation between min. unsat. core and conflict clause?
- Why is the algorithm correct, why does it terminate?
- How can we extend it two more than one theory?
- What is the relation to Nelson-Oppen?

Program Correctness

- What is a specification?
- What types of specification are in a typical program?
  (Precondition, postcondition, loop invariants, assertions)
- When is a procedure correct (partial/total correctness)?
- What is a basic path? Why is it useful?
- How do we prove correctness of a basic path?
- What is a verification condition?
- What is the weakest precondition?
- How do we compute weakest precondition?
- What is a P-invariant annotation, what is a P-inductive annotation?
- Why are we interested in P-inductive annotations?
- What is a ranking function? Why do we need it?
- What is a well-founded relation?
- How do we prove total correctness?

Interpolants

- What is an interpolant?
- What is the symbol condition?
- Why is an interpolant useful?
- How can we compute interpolants in T_E?
- How can we compute interpolants in $T_{\mathbb{Q}}$?
- How can we compute interpolants for DPLL proofs?
- What is the difficulty with theory combination?

General hints for exam

- You should learn definitions (formally).
  This includes the rules (semantic argument, DPLL with learning).
- You should understand them (informally).
- You should know important theorems.
- Knowing the proofs is a plus. Don't loose yourself in the details!
- You should be able to apply the decision procedures.
  Do the exercises! Invent some new exercises and solve them!
- You should know some examples/counter-examples,
  e.g., why is λ necessary?
- When you feel well prepared, check if you can answer the questions in this slide set.
- When learning, do not leave out a whole topic completely!
- Learn in a group. Ask question to each other and answer them as if you were in the exam.

Organisation

- There will be only oral exams for this lecture.
- You should have officially registered at the Prüfungsamt.
- The exams will be in March.