Overview of Automated Reasoning

Peter Baumgartner

Peter.Baumgartner@nicta.com.au

http://users.cecs.anu.edu.au/~baumgart/

What is Automated Reasoning?

Theme

Building push-button technology (software) for mathematical-logical reasoning on computer

Relevant fields

- Mathematical logic and philosophy: formal logics and calculi
- Theoretical computer science: computability theory, complexity theory
- Applied and practical computer science: artifical intelligence, data structures and algorithms

Applications: Software verification, hardware verification, analysing dynamic properties of reactive systems, databases, mathematical theorem proving, planning, diagnosis, knowledge representation (description logics), logic programming, constraint solving

Automated Reasoning systems parametrized in
logic and reasoning service

Logics and Reasoning Service: Theorem Proving

Mathematical structures, e.g. groups

$$\forall x\ 1 \cdot x = x \qquad \forall x\ x \cdot 1 = x \tag{N}$$

$$\forall x\ x^{-1} \cdot x = 1 \qquad \forall x\ x \cdot x^{-1} = 1 \tag{I}$$

$$\forall x, y, z\ (x \cdot y) \cdot z = x \cdot (y \cdot z) \tag{A}$$

Logic: First-order logic with equality

Reasoning Service: Theorem proving: prove that

$\forall x\ (x \cdot x) = 1 \rightarrow \forall x, y\ x \cdot y = y \cdot x$ follows

Meta-level: the word problem for groups is decidable

Logics and Reasoning Service: Constraint Solving

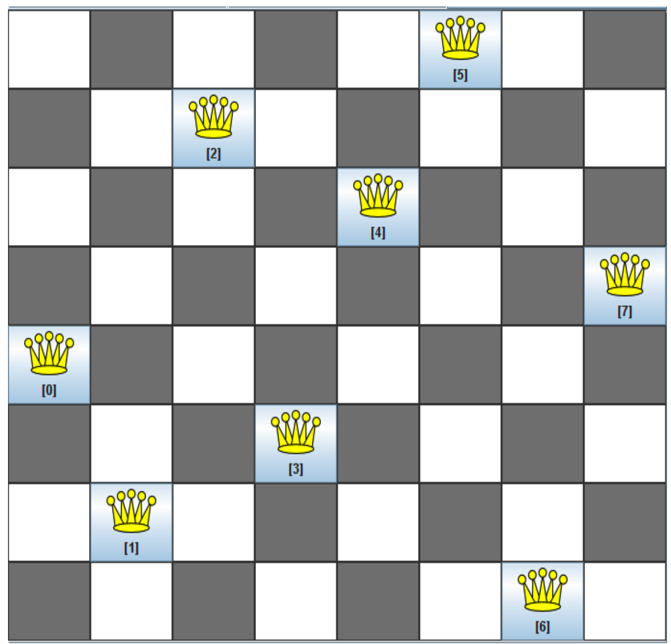
The n-queens problem:

Given: An $n \times n$ chessboard

Question: Is it possible to place n queens so that no queen attacks any other?

A solution for $n = 8$

$$p[1] = 6$$
$$p[2] = 3$$
$$p[3] = 5$$
$$p[4] = 8$$
$$p[5] = 1$$
$$p[6] = 4$$
$$p[7] = 2$$
$$p[8] = 7$$



Use a constraint solver to find a solution

Logics and Reasoning Service: Constraint Solving

A Zinc model, ready to be run by a constraint solver:

```
int: n = 8;
array [1..n] of var 1..n: p;
constraint
        forall (i in 1..n, j in i + 1..n) (
                p[i]     != p[j]
        /\      p[i] + i != p[j] + j
        /\      p[i] - i != p[j] - j
        );
solve satisfy; output ["Solution: ", show(p), "\n"];
```

Logic: Integer arithmetic, quantifiers, arrays

Reasoning Service: Constraint solving

Search assignments for all vars `p[1]` to `p[n]` such that `constraint` is satisfied

With `n` fixed, all variables and `i` and `j` range over finite domains.

Logics and Reasoning Service: Constraint Solving

The same problem, written in sorted first-order logic:

$$n : \mathbb{Z} \qquad \text{(Declaration of } n\text{)}$$

$$p : \mathbb{Z} \mapsto \mathbb{Z} \qquad \text{(Declaration of } p\text{)}$$

$$n = 8$$

$$\begin{aligned}&\forall i : \mathbb{Z}\ j : \mathbb{Z}\ (1 \leq i \wedge i \leq n \wedge i + 1 \leq j \wedge j < n \Rightarrow \\ &\qquad p(i) \neq p(j) \wedge p(i) + i \neq p(j) + j \wedge p(i) - i \neq p(j) - j)\end{aligned} \qquad \text{(Queens)}$$

$$p(1) = 1 \vee p(1) = 2 \vee \cdots \vee p(1) = 8 \qquad (p(1) \in \{1, \ldots, n\})$$

$$\vdots$$

$$p(8) = 1 \vee p(8) = 2 \vee \cdots \vee p(8) = 8 \qquad (p(n) \in \{1, \ldots, n\})$$

Logic: Integer arithmetic, quantifiers, “free” symbol p

Reasoning Service: Satisfiability: find a satisfying interpretation I (a model) and evaluate $I(p(1)), \ldots, I(p(n))$ to read off the answer

Logics and Reasoning Service: Constraint Solving

Summary so far

- Constraint solvers are applicable when all variables range over finite domains. They can exploit this fact when searching for a solution, in particular for "constraint propagation"
- Theorem provers are intended to work on infinite domains. In the N-queens example the variables are quantified over finite domains only coincidentially.
- On finite search problems constraint solvers perform usually much better

So, why theorem proving?

Answer: for analysing the problem for any board size n

Logical Analysis Example: N-Queens

$p[1] = 6$

$p[2] = 3$

$p[3] = 5$

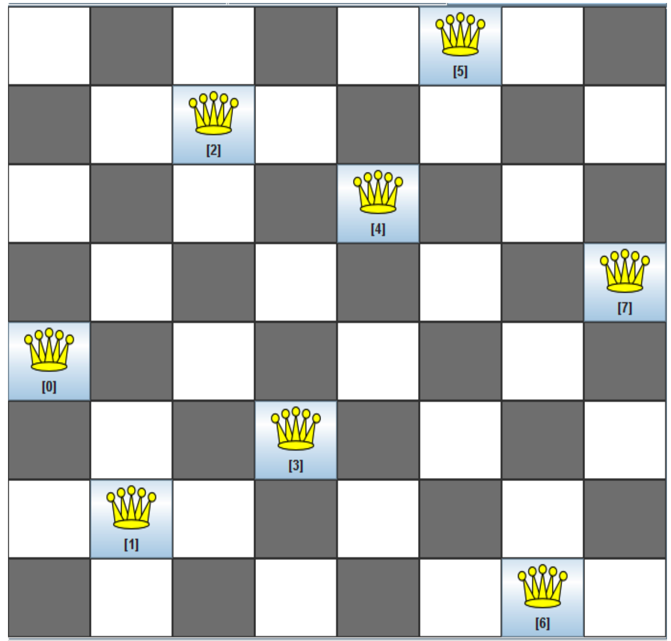
$p[4] = 8$

$p[5] = 1$

$p[6] = 4$

$p[7] = 2$

$p[8] = 7$



Number of solutions, depending on n:

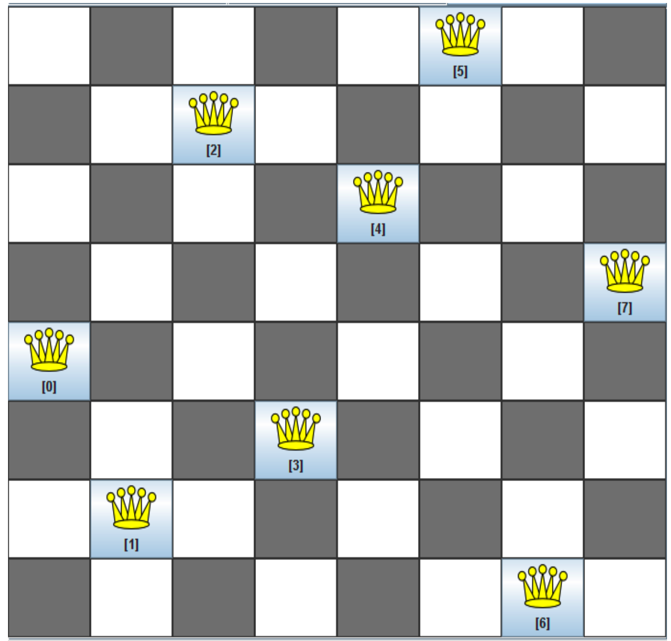
***n*:**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**..**	**24**	**25**
unique:	1	0	0	1	2	1	6	12	46	92	341	1,787	9,233	45,752	..	28,439,272,956,934	275,986,683,743,434
distinct:	1	0	0	2	10	4	40	92	352	724	2,680	14,200	73,712	365,596	..	227,514,171,973,736	2,207,893,435,808,352

"unique" is **"distinct"** modulo reflection/rotation symmetry

For efficiency reasons better avoid searching symmetric solutions

Logical Analysis Example: N-Queens

$p[1] = 6$
$p[2] = 3$
$p[3] = 5$
$p[4] = 8$
$p[5] = 1$
$p[6] = 4$
$p[7] = 2$
$p[8] = 7$



- The n-queens has variable symmetry: mapping $p[i] \mapsto p[n+1-i]$ preserves solutions, for any n
- Therefore, it is justified to add (to the formalization) a constraint $p[1] < p[n]$, for search space pruning
- But how can we know that the problem has symmetries?
  This is a theorem proving task!

Proving Symmetry: Formalization

We need two "copies" (Queens_p) and (Queens_q) of the constraint:

$$n : \mathbb{Z} \tag{Declaration of n}$$

$$p, q : \mathbb{Z} \mapsto \mathbb{Z} \tag{Declaration of p, q}$$

$$perm : \mathbb{Z} \mapsto \mathbb{Z} \tag{Declaration of $perm$}$$

$$\begin{array}{l} \forall i : \mathbb{Z}\ j : \mathbb{Z}\ (1 \leq i \wedge i \leq n \wedge i + 1 \leq j \wedge j < n \Rightarrow \\ \quad p(i) \neq p(j) \wedge p(i) + i \neq p(j) + j \wedge p(i) - i \neq p(j) - j) \end{array} \tag{Queens\_p}$$

$$\begin{array}{l} \forall i : \mathbb{Z}\ j : \mathbb{Z}\ (1 \leq i \wedge i \leq n \wedge i + 1 \leq j \wedge j < n \Rightarrow \\ \quad q(i) \neq q(j) \wedge q(i) + i \neq q(j) + j \wedge q(i) - i \neq q(j) - j) \end{array} \tag{Queens\_q}$$

$$\forall i : \mathbb{Z}\ perm(i) = n + 1 - i \tag{Def. permutation}$$

Logic: Integer arithmetic, quantifiers, "free" symbol p

Reasoning Service: Entailment (logical consequence)

The above entails (Queens_p) $\wedge$ ($\forall i : \mathbb{Z}\ q(i) = p(perm(i))$) $\Rightarrow$ (Queens_q) which expresses the symmetry property. Use a theorem prover

Logics and Reasoning Service - Spectrum

Logics

Base logic: propositional/first-order/higher-order

Syntactic fragments
(Description Logics, Datalog, ...)

Classical/non-monotonic

Modalities (temporal, deontic, ...)

Over structures (finite trees, graphs,...)

Modulo Theories (equality, arithmetic, ...)

Services

Model checking
(evaluation)

Satisfiability
(minimal models)

Validity

Induction

Abduction

Almost any subset of the left column (potentially) makes sense

The challenge is to build "decent" calculi/theorem provers: theoretically analysed, avoiding redundancies, practically useful, meaningful answers (proofs, models), ...

Contents

Introduction

Logics and Reasoning Service (already done)

Methods for Automated Theorem Proving

Overview of some widely used general methods

- Propositional SAT solving
- First-order logic and clause normal forms
- Proof Procedures Based on Herbrand's Theorem
- The Resolution calculus
- Model generation

Theory Reasoning

Methods to reason with specific background theories

- Paramodulation (Equality)
- Satisfiability Modulo Theories (SMT)
- Quantifier elimination for linear real arithmetic
- Combining multiple theories

Propositional Logic – Syntax

Atom	truth symbols $\top$("true") and $\bot$("false")
	propositional variables $P, Q, R, P_1, Q_1, R_1, \cdots$
Literal	atom α or its negation $\neg\alpha$
Formula	literal or application of a
	logical connective to formulae F, F_1, F_2

$\neg F$	"not"	(negation)
$F_1 \wedge F_2$	"and"	(conjunction)
$F_1 \vee F_2$	"or"	(disjunction)
$F_1 \rightarrow F_2$	"implies"	(implication)
$F_1 \leftrightarrow F_2$	"if and only if"	(iff)

Example

Formula $F : (P \land Q) \to (\top \lor \neg Q)$

Atoms: $P, Q, \top$

Literal: $\neg Q$

Subformulas: $P \land Q, \quad \top \lor \neg Q$

Abbreviation (leave parenthesis away)

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

Propositional Logic – Semantics (meaning)

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

Interpretation

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

Evaluation of F under I:

F	$\neg F$
0	1
1	0

where 0 corresponds to value false
1 true

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

Example

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

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

P	Q	$\neg Q$	$P \land Q$	$P \lor \neg Q$	F
1	0	1	0	1	1

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

F evaluates to true under I

Inductive Definition of PL's Semantics

I	$\models$	F	if F evaluates to	true	under I ("I satisfies F")
I	$\not\models$	F		false	under I ("I falsifies F")

Inductive Definition of PL's Semantics

$I \models F$ if F evaluates to true under I ("I satisfies F")

$I \not\models F$ false under I ("I falsifies F")

Base Case:

$I \models \top$

$I \not\models \bot$

$I \models P$ iff $I[P] = \text{true}$

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

Inductive Definition of PL's Semantics

$I \models F$ if F evaluates to true under I ("I satisfies F")

$I \not\models F$ false under I ("I falsifies F")

Base Case:

$$I \models \top$$

$$I \not\models \bot$$

$$I \models P \quad \text{iff} \quad I[P] = \text{true}$$

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

Inductive Case:

$$I \models \neg F \quad \text{iff } I \not\models F$$

$$I \models F_1 \wedge F_2 \quad \text{iff } I \models F_1 \text{ and } I \models F_2$$

$$I \models F_1 \vee F_2 \quad \text{iff } I \models F_1 \text{ or } I \models F_2$$

$$I \models F_1 \rightarrow F_2 \quad \text{iff, if } I \models F_1 \text{ then } I \models F_2$$

$$I \models F_1 \leftrightarrow F_2 \quad \text{iff, } I \models F_1 \text{ and } I \models F_2\text{, or } I \not\models F_1 \text{ and } I \not\models F_2$$

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

Example

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

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

Example

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

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

1. $I \models P$ since $I[P] = \text{true}$

Example

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

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

1. $I \models P$ since $I[P] = \text{true}$
2. $I \not\models Q$ since $I[Q] = \text{false}$

Example

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

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

1.	I	$\models$	P	since $I[P] = \text{true}$
2.	I	$\not\models$	Q	since $I[Q] = \text{false}$
3.	I	$\models$	$\neg Q$	by 2 and $\neg$

Example

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

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

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

Example

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

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

1.	I	$\models$	P	since $I[P] = \text{true}$
2.	I	$\not\models$	Q	since $I[Q] = \text{false}$
3.	I	$\models$	$\neg Q$	by 2 and $\neg$
4.	I	$\not\models$	$P \land Q$	by 2 and $\land$
5.	I	$\models$	$P \lor \neg Q$	by 1 and $\lor$

Example

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

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

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

Example

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

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

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

Thus, F is true under I.

Satisfiability and Validity

F satisfiable iff there exists an interpretation I such that $I \models F$.
In this case I is called a model of F.

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

A formula G entails F iff for all interpretations I, if $I \models G$ then $I \models F$.
Notation: $G \models F$.

Important Facts

F is valid iff $\neg F$ is unsatisfiable

$G \models F$ iff $G \wedge \neg F$ is unsatisfiable

Note: Thus, "validity" and "entailment" can be reduced to unsatisfiability.

Method 1: Truth Tables

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

P Q	$P \land Q$	$\neg Q$	$P \lor \neg Q$	F
0 0	0	1	1	1
0 1	0	0	0	1
1 0	0	1	1	1
1 1	1	0	1	1

Thus F is valid.

Method 1: Truth Tables

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

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

Thus F is satisfiable, but invalid.

Examples

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

1. $F_1 : P \wedge Q$

2. $F_2 : \neg(P \wedge Q)$

3. $F_3 : P \vee \neg P$

4. $F_4 : \neg(P \vee \neg P)$

5. $F_5 : (P \to Q) \wedge (P \vee Q) \wedge \neg Q$

Examples

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

1. $F_1 : P \wedge Q$

   satisfiable, not valid

2. $F_2 : \neg(P \wedge Q)$

3. $F_3 : P \vee \neg P$

4. $F_4 : \neg(P \vee \neg P)$

5. $F_5 : (P \rightarrow Q) \wedge (P \vee Q) \wedge \neg Q$

Examples

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

1. $F_1 : P \wedge Q$

   satisfiable, not valid

2. $F_2 : \neg(P \wedge Q)$

   satisfiable, not valid

3. $F_3 : P \vee \neg P$

4. $F_4 : \neg(P \vee \neg P)$

5. $F_5 : (P \rightarrow Q) \wedge (P \vee Q) \wedge \neg Q$

Examples

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

1. $F_1 : P \wedge Q$

   satisfiable, not valid

2. $F_2 : \neg(P \wedge Q)$

   satisfiable, not valid

3. $F_3 : P \vee \neg P$

   satisfiable, valid

4. $F_4 : \neg(P \vee \neg P)$

5. $F_5 : (P \rightarrow Q) \wedge (P \vee Q) \wedge \neg Q$

Examples

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

1. $F_1 : P \wedge Q$

   satisfiable, not valid
2. $F_2 : \neg(P \wedge Q)$

   satisfiable, not valid
3. $F_3 : P \vee \neg P$

   satisfiable, valid
4. $F_4 : \neg(P \vee \neg P)$

   unsatisfiable, not valid
5. $F_5 : (P \rightarrow Q) \wedge (P \vee Q) \wedge \neg Q$

Examples

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

1. $F_1 : P \wedge Q$

   satisfiable, not valid

2. $F_2 : \neg(P \wedge Q)$

   satisfiable, not valid

3. $F_3 : P \vee \neg P$

   satisfiable, valid

4. $F_4 : \neg(P \vee \neg P)$

   unsatisfiable, not valid

5. $F_5 : (P \rightarrow Q) \wedge (P \vee Q) \wedge \neg Q$

   unsatisfiable, not valid

Method 2: Tableau Calculus (Not Here)

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

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

$$\nwarrow \text{or}$$

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

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

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

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

Method 3: DPLL

Davis/Putnam/Logemann/Loveland, 1960's. Works with clause logic.

Method 3: DPLL

Davis/Putnam/Logemann/Loveland, 1960's. Works with clause logic.

Clause Logic

Method 3: DPLL

Davis/Putnam/Logemann/Loveland, 1960's. Works with clause logic.

Clause Logic

- A (propositional) atom is a propositional variable (but no longer $\top$, $\bot$).
- A literal is either an atom or the negation of an atom.
  Example: A, $\neg A$

Method 3: DPLL

Davis/Putnam/Logemann/Loveland, 1960's. Works with clause logic.

Clause Logic

- A (propositional) atom is a propositional variable (but no longer $\top$, $\bot$).
- A literal is either an atom or the negation of an atom.
  Example: A, $\neg A$
- A clause is a (possibly empty) disjunction of literals (i.e. n-ary "$\vee$" now).
  Example: $\neg B \vee C \vee \neg D$

Method 3: DPLL

Davis/Putnam/Logemann/Loveland, 1960's. Works with clause logic.

Clause Logic

- A (propositional) atom is a propositional variable (but no longer $\top$, $\bot$).
- A literal is either an atom or the negation of an atom.
  Example: A, $\neg A$
- A clause is a (possibly empty) disjunction of literals (i.e. n-ary "$\vee$" now).
  Example: $\neg B \vee C \vee \neg D$
- A formula is in clause normal form, or conjunctive normal form (CNF) iff it is a conjunction of clauses.
  Example: $(\neg A \vee B) \wedge A \wedge (\neg B \vee C \vee \neg D)$

Method 3: DPLL

Davis/Putnam/Logemann/Loveland, 1960's. Works with clause logic.

Clause Logic

- A (propositional) atom is a propositional variable (but no longer $\top$, $\bot$).
- A literal is either an atom or the negation of an atom.
  Example: A, $\neg A$
- A clause is a (possibly empty) disjunction of literals (i.e. n-ary "$\vee$" now).
  Example: $\neg B \vee C \vee \neg D$
- A formula is in clause normal form, or conjunctive normal form (CNF) iff it is a conjunction of clauses.
  Example: $(\neg A \vee B) \wedge A \wedge (\neg B \vee C \vee \neg D)$
- Most automated theorem proving methods work with clause logic.
  Every formula can be transformed into an equivalent CNF.

Method 3: DPLL

Davis/Putnam/Logemann/Loveland, 1960's. Works with clause logic.

Clause Logic

- A (propositional) atom is a propositional variable (but no longer $\top$, $\bot$).
- A literal is either an atom or the negation of an atom.
  Example: A, $\neg A$
- A clause is a (possibly empty) disjunction of literals (i.e. n-ary "$\vee$" now).
  Example: $\neg B \vee C \vee \neg D$
- A formula is in clause normal form, or conjunctive normal form (CNF) iff it is a conjunction of clauses.
  Example: $(\neg A \vee B) \wedge A \wedge (\neg B \vee C \vee \neg D)$
- Most automated theorem proving methods work with clause logic.
  Every formula can be transformed into an equivalent CNF.

Why are the truth symbols $\top$ and $\bot$ not needed?

DPLL Interpretations

DPLL works with trees whose nodes are labelled with literals.

Consistency: No branch contains the labels A and $\neg A$, for no A

Every branch in a tree is taken as a (consistent) set of its literals

A consistent set of literals S is taken as an interpretation:

Positive literals: if $A \in S$ then $(A \mapsto \text{true}) \in I$

Negative literals: if $\neg A \in S$ then $(A \mapsto \text{false}) \in I$

Default: if $A \notin S$ and $\neg A \notin S$ then $(A \mapsto \text{false}) \in I$

DPLL Interpretations

DPLL works with trees whose nodes are labelled with literals.

Consistency: No branch contains the labels A and $\neg A$, for no A

Every branch in a tree is taken as a (consistent) set of its literals

A consistent set of literals S is taken as an interpretation:

Positive literals: if $A \in S$ then $(A \mapsto \text{true}) \in I$

Negative literals: if $\neg A \in S$ then $(A \mapsto \text{false}) \in I$

Default: if $A \notin S$ and $\neg A \notin S$ then $(A \mapsto \text{false}) \in I$

Example

$\{A, \neg B, D\}$ stands for

$I : \{A \mapsto \text{true},\ B \mapsto \text{false},\ C \mapsto \text{false},\ D \mapsto \text{true}\}$

DPLL as a Semantic Tree Method

(1) $A \vee B$ (2) $C \vee \neg A$ (3) $D \vee \neg C \vee \neg A$ (4) $\neg D \vee \neg B$

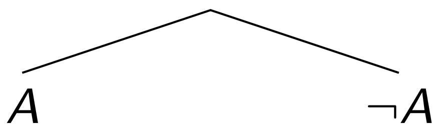
⟨empty tree⟩

$$\{\} \not\models A \vee B$$
$$\{\} \models C \vee \neg A$$
$$\{\} \models D \vee \neg C \vee \neg A$$
$$\{\} \models \neg D \vee \neg B$$

- A Branch stands for an interpretation
- Purpose of splitting: satisfy a clause that is currently falsified
- Close branch if some clause is plainly falsified by it ($\star$)

DPLL as a Semantic Tree Method

(1) $A \vee B$ (2) $C \vee \neg A$ (3) $D \vee \neg C \vee \neg A$ (4) $\neg D \vee \neg B$

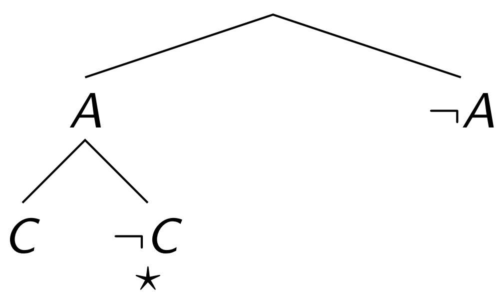


$$\{A\} \models A \vee B$$
$$\{A\} \not\models C \vee \neg A$$
$$\{A\} \models D \vee \neg C \vee \neg A$$
$$\{A\} \models \neg D \vee \neg B$$

- A Branch stands for an interpretation
- Purpose of splitting: satisfy a clause that is currently falsified
- Close branch if some clause is plainly falsified by it ($\star$)

DPLL as a Semantic Tree Method

(1) $A \vee B$ (2) $C \vee \neg A$ (3) $D \vee \neg C \vee \neg A$ (4) $\neg D \vee \neg B$



$\{A, C\} \models A \vee B$

$\{A, C\} \models C \vee \neg A$

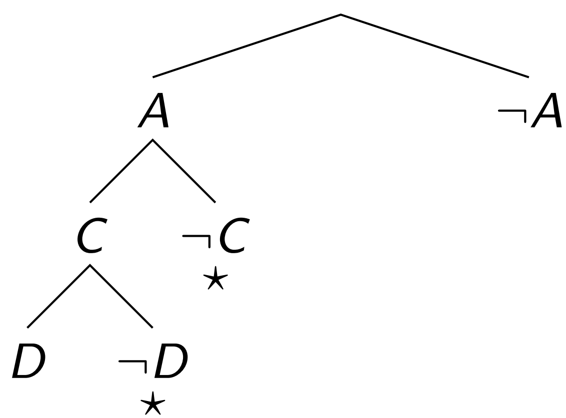
$\{A, C\} \not\models D \vee \neg C \vee \neg A$

$\{A, C\} \models \neg D \vee \neg B$

- A Branch stands for an interpretation
- Purpose of splitting: satisfy a clause that is currently falsified
- Close branch if some clause is plainly falsified by it ($\star$)

DPLL as a Semantic Tree Method

(1) $A \lor B$ (2) $C \lor \neg A$ (3) $D \lor \neg C \lor \neg A$ (4) $\neg D \lor \neg B$



$\{A, C, D\} \models A \lor B$

$\{A, C, D\} \models C \lor \neg A$

$\{A, C, D\} \models D \lor \neg C \lor \neg A$

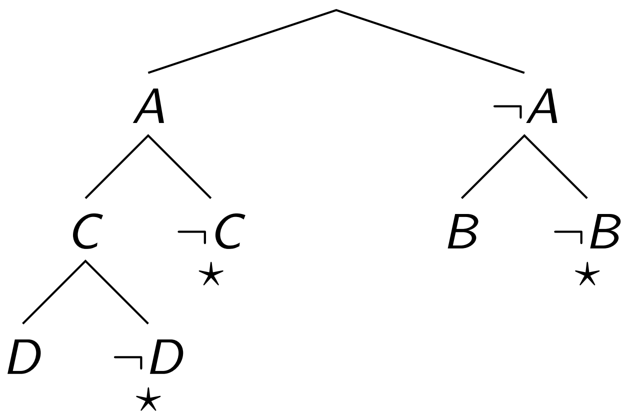
$\{A, C, D\} \models \neg D \lor \neg B$

Model $\{A, C, D\}$ found.

- A Branch stands for an interpretation
- Purpose of splitting: satisfy a clause that is currently falsified
- Close branch if some clause is plainly falsified by it ($\star$)

DPLL as a Semantic Tree Method

(1) $A \vee B$ (2) $C \vee \neg A$ (3) $D \vee \neg C \vee \neg A$ (4) $\neg D \vee \neg B$



$\{B\} \models A \vee B$
$\{B\} \models C \vee \neg A$
$\{B\} \models D \vee \neg C \vee \neg A$
$\{B\} \models \neg D \vee \neg B$

Model $\{B\}$ found.

- A Branch stands for an interpretation
- Purpose of splitting: satisfy a clause that is currently falsified
- Close branch if some clause is plainly falsified by it ($\star$)

DPLL Pseudocode

literal L: a variable A or its negation $\neg A$

clause: a set of literals, e.g., $\{A, \neg B, C\}$, connected by "or"

```
function DPLL(N)    %% N is a set of clauses, connected by "and"
  while N contains a unit clause {L} %% L is a implied
    N := simplify(N, L);
  if N = {} then return true;
  if {} ∈ N then return false;
  L := choose-literal(N); %% L is a decision literal
  if DPLL(simplify(N, L)) then return true;
  else return DPLL(simplify(N, ¬L));
```

```
function simplify(N, L)  %% also called unit propagation
  remove all clauses from N that contain L;
  delete ¬L from all remaining clauses;
  return the resulting clause set;
```

(The semantic tree method does not show unit propagation.)

Simplify Examples

```
function simplify(N, L) %% also called unit propagation
    remove all clauses from N that contain L;
    delete ¬L from all remaining clauses;
    return the resulting clause set;
```

$$\mathtt{simplify}(\{A \vee \neg B,\ C \vee \neg A,\ D \vee \neg C \vee \neg A,\ \neg D \vee \neg B\},\ A)$$

$$=$$

Simplify Examples

```
function simplify(N, L) %% also called unit propagation
    remove all clauses from N that contain L;
    delete ¬L from all remaining clauses;
    return the resulting clause set;
```

$$\mathtt{simplify}(\{A \vee \neg B,\ C \vee \neg A,\ D \vee \neg C \vee \neg A,\ \neg D \vee \neg B\},\ A)$$
$$= \{\quad C \quad,\ D \vee \neg C \quad,\ \neg D \vee \neg B\}$$

Simplify Examples

```
function simplify(N, L) %% also called unit propagation
    remove all clauses from N that contain L;
    delete ¬L from all remaining clauses;
    return the resulting clause set;
```

$$\texttt{simplify}(\{A \vee \neg B,\ C \vee \neg A,\ D \vee \neg C \vee \neg A,\ \neg D \vee \neg B\},\ A)$$
$$= \{\ C,\ D \vee \neg C,\ \neg D \vee \neg B\}$$

$$\texttt{simplify}(\{\ C,\ D \vee \neg C,\ \neg D \vee \neg B\},\ C)$$
$$=$$

Simplify Examples

```
function simplify(N, L) %% also called unit propagation
    remove all clauses from N that contain L;
    delete ¬L from all remaining clauses;
    return the resulting clause set;
```

$$\begin{aligned}\texttt{simplify}(\{A \vee \neg B,\ C \vee \neg A,\ D \vee \neg C \vee \neg A,\ \neg D \vee \neg B\},\ A) \\ = \{ C,\ D \vee \neg C,\ \neg D \vee \neg B\}\end{aligned}$$

$$\begin{aligned}\texttt{simplify}(\{C,\ D \vee \neg C,\ \neg D \vee \neg B\},\ C) \\ = \{ D,\ \neg D \vee \neg B\}\end{aligned}$$

Simplify Examples

```
function simplify(N, L) %% also called unit propagation
    remove all clauses from N that contain L;
    delete ¬L from all remaining clauses;
    return the resulting clause set;
```

$$\begin{aligned}&\texttt{simplify}(\{A \vee \neg B,\ C \vee \neg A,\ D \vee \neg C \vee \neg A,\ \neg D \vee \neg B\},\ A)\\ &\quad = \{C,\ D \vee \neg C,\ \neg D \vee \neg B\}\end{aligned}$$

$$\begin{aligned}&\texttt{simplify}(\{C,\ D \vee \neg C,\ \neg D \vee \neg B\},\ C)\\ &\quad = \{D,\ \neg D \vee \neg B\}\end{aligned}$$

$$\begin{aligned}&\texttt{simplify}(\{D,\ \neg D \vee \neg B\},\ D)\\ &\quad =\end{aligned}$$

Simplify Examples

```
function simplify(N, L) %% also called unit propagation
    remove all clauses from N that contain L;
    delete ¬L from all remaining clauses;
    return the resulting clause set;
```

$$\begin{aligned}&\mathtt{simplify}(\{A \lor \neg B,\ C \lor \neg A,\ D \lor \neg C \lor \neg A,\ \neg D \lor \neg B\},\ A)\\ &\quad = \{C,\ D \lor \neg C,\ \neg D \lor \neg B\}\end{aligned}$$

$$\begin{aligned}&\mathtt{simplify}(\{C,\ D \lor \neg C,\ \neg D \lor \neg B\},\ C)\\ &\quad = \{D,\ \neg D \lor \neg B\}\end{aligned}$$

$$\begin{aligned}&\mathtt{simplify}(\{D,\ \neg D \lor \neg B\},\ D)\\ &\quad = \{\neg B\}\end{aligned}$$

Making DPLL Fast – Overview

Conflict Driven Clause Learning (CDCL) solvers extend DPLL:

Lemma learning: add new clauses to the clause set as branches get closed (“conflict driven”)

Goal: reuse information that is obtained in one branch for subsequent derivation steps.

Backtracking: replace chronological backtracking by “dependency-directed backtracking”, aka “backjumping”: on backtracking, skip splits that are not necessary to close a branch

Randomized restarts: every now and then start over, with learned clauses

Variable selection heuristics: what literal to split on. E.g., use literals that occur often

Make unit-propagation fast: 2-watched literal technique

2-Watched Literals Example

Idea: in an n-literal clause, $n-1$ literals must be assigned false before it can unit-propagate. Defer unit propagation until this is the case.

In a clause, two of its literals are watched. When a literal L is assigned a value, (only) clauses where $\neg L$ is watched are visited.

Invariant: if clause is not satisfied, watched literals are undefined.

Clause $\underline{\neg A} \vee \underline{\neg B} \vee \neg C \vee \neg D \vee E$ (watched literals underlined)

2-Watched Literals Example

Idea: in an n-literal clause, $n - 1$ literals must be assigned false before it can unit-propagate. Defer unit propagation until this is the case.
In a clause, two of its literals are watched. When a literal L is assigned a value, (only) clauses where $\neg L$ is watched are visited.
Invariant: if clause is not satisfied, watched literals are undefined.

Clause $\underline{\neg A} \vee \underline{\neg B} \vee \neg C \vee \neg D \vee E$ (watched literals underlined)

1. Assignments developed in this order C

2-Watched Literals Example

Idea: in an n-literal clause, $n-1$ literals must be assigned false before it can unit-propagate. Defer unit propagation until this is the case.

In a clause, two of its literals are watched. When a literal L is assigned a value, (only) clauses where $\neg L$ is watched are visited.

Invariant: if clause is not satisfied, watched literals are undefined.

Clause $\underline{\neg A} \vee \underline{\neg B} \vee \neg C \vee \neg D \vee E$ (watched literals underlined)

1. Assignments developed in this order $C \quad - \quad D$

2-Watched Literals Example

Idea: in an n-literal clause, $n - 1$ literals must be assigned false before it can unit-propagate. Defer unit propagation until this is the case.

In a clause, two of its literals are watched. When a literal L is assigned a value, (only) clauses where $\neg L$ is watched are visited.

Invariant: if clause is not satisfied, watched literals are undefined.

Clause $\underline{\neg A} \vee \underline{\neg B} \vee \neg C \vee \neg D \vee E$ (watched literals <u>underlined</u>)

1. Assignments developed in this order $C \quad - \quad D \quad - \quad A$

2-Watched Literals Example

Idea: in an n-literal clause, $n-1$ literals must be assigned false before it can unit-propagate. Defer unit propagation until this is the case.
In a clause, two of its literals are watched. When a literal L is assigned a value, (only) clauses where $\neg L$ is watched are visited.
Invariant: if clause is not satisfied, watched literals are undefined.

Clause $\underline{\neg A} \vee \underline{\neg B} \vee \neg C \vee \neg D \vee E$ (watched literals <u>underlined</u>)

1. Assignments developed in this order $C \quad - \quad D \quad - \quad A$
2. Watched literal $\neg A$ is false now $\leadsto$ find another literal to watch

2-Watched Literals Example

Idea: in an n-literal clause, $n-1$ literals must be assigned false before it can unit-propagate. Defer unit propagation until this is the case.
In a clause, two of its literals are watched. When a literal L is assigned a value, (only) clauses where $\neg L$ is watched are visited.
Invariant: if clause is not satisfied, watched literals are undefined.

Clause $\underline{\neg A} \vee \underline{\neg B} \vee \neg C \vee \neg D \vee E$ (watched literals underlined)

1. Assignments developed in this order $C \quad - \quad D \quad - \quad A$
2. Watched literal $\neg A$ is false now $\rightsquigarrow$ find another literal to watch

Clause $\neg A \vee \underline{\neg B} \vee \neg C \vee \neg D \vee \underline{E}$

2-Watched Literals Example

Idea: in an n-literal clause, $n-1$ literals must be assigned false before it can unit-propagate. Defer unit propagation until this is the case.
In a clause, two of its literals are watched. When a literal L is assigned a value, (only) clauses where $\neg L$ is watched are visited.
Invariant: if clause is not satisfied, watched literals are undefined.

Clause $\underline{\neg A} \vee \underline{\neg B} \vee \neg C \vee \neg D \vee E$ (watched literals underlined)

1. Assignments developed in this order $C \quad - \quad D \quad - \quad A$
2. Watched literal $\neg A$ is false now $\rightsquigarrow$ find another literal to watch

Clause $\neg A \vee \underline{\neg B} \vee \neg C \vee \neg D \vee \underline{E}$

3. Extend with decision literal $C \quad - \quad D \quad - \quad A \quad - \quad B$

2-Watched Literals Example

Idea: in an n-literal clause, $n-1$ literals must be assigned false before it can unit-propagate. Defer unit propagation until this is the case.
In a clause, two of its literals are watched. When a literal L is assigned a value, (only) clauses where $\neg L$ is watched are visited.
Invariant: if clause is not satisfied, watched literals are undefined.

Clause $\underline{\neg A} \vee \underline{\neg B} \vee \neg C \vee \neg D \vee E$ (watched literals underlined)

1. Assignments developed in this order $C \quad - \quad D \quad - \quad A$
2. Watched literal $\neg A$ is false now $\rightsquigarrow$ find another literal to watch

Clause $\neg A \vee \underline{\neg B} \vee \neg C \vee \neg D \vee \underline{E}$

3. Extend with decision literal $C \quad - \quad D \quad - \quad A \quad - \quad B$
4. Imposible to watch two literals now $\rightsquigarrow$ E is unit-propagated

2-Watched Literals Example

Idea: in an n-literal clause, $n-1$ literals must be assigned false before it can unit-propagate. Defer unit propagation until this is the case.

In a clause, two of its literals are watched. When a literal L is assigned a value, (only) clauses where $\neg L$ is watched are visited.

Invariant: if clause is not satisfied, watched literals are undefined.

Clause $\underline{\neg A} \vee \underline{\neg B} \vee \neg C \vee \neg D \vee E$ (watched literals underlined)

1. Assignments developed in this order $C \quad - \quad D \quad - \quad A$
2. Watched literal $\neg A$ is false now $\rightsquigarrow$ find another literal to watch

Clause $\neg A \vee \underline{\neg B} \vee \neg C \vee \neg D \vee \underline{E}$

3. Extend with decision literal $C \quad - \quad D \quad - \quad A \quad - \quad B$
4. Imposible to watch two literals now $\rightsquigarrow$ E is unit-propagated

Invariant is (also) maintained on backtracking to $\neg B$ without extra work.

Lemma Learning

"Avoid making the same mistake twice"

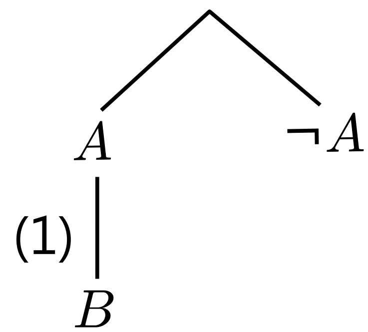
$$\ldots$$

$$B \vee \neg A \qquad (1)$$

$$D \vee \neg C \qquad (2)$$

$$\neg D \vee \neg B \vee \neg C \qquad (3)$$

w/o Lemma

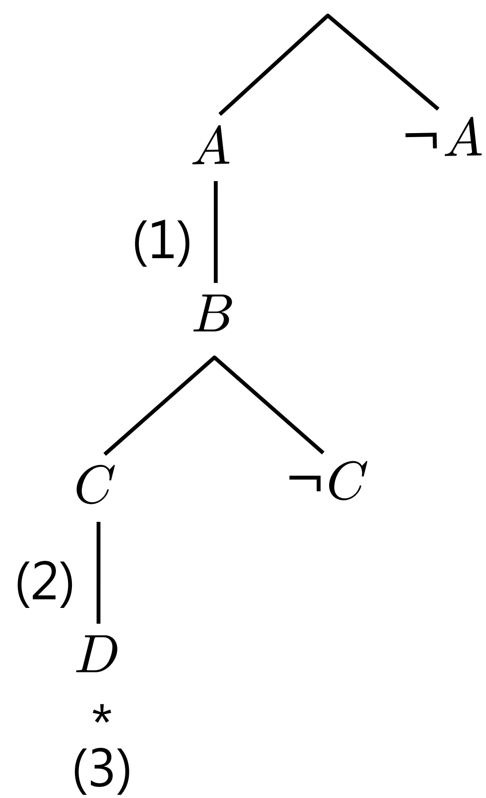


Lemma Learning

"Avoid making the same mistake twice"

$$\ldots$$
$$B \vee \neg A \qquad (1)$$
$$D \vee \neg C \qquad (2)$$
$$\neg D \vee \neg B \vee \neg C \qquad (3)$$

w/o Lemma

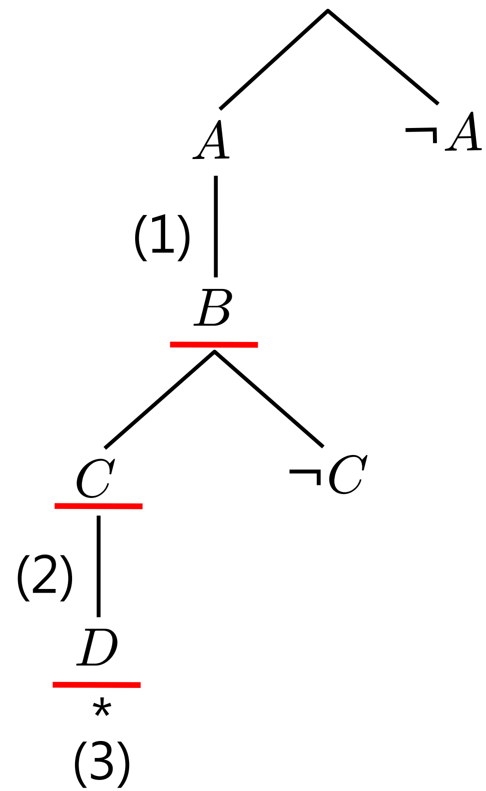


Lemma Learning

"Avoid making the same mistake twice"

$$\ldots$$
$$B \vee \neg A \quad (1)$$
$$D \vee \neg C \quad (2)$$
$$\underline{\neg D} \vee \underline{\neg B} \vee \underline{\neg C} \quad (3)$$

w/o Lemma



Lemma Learning

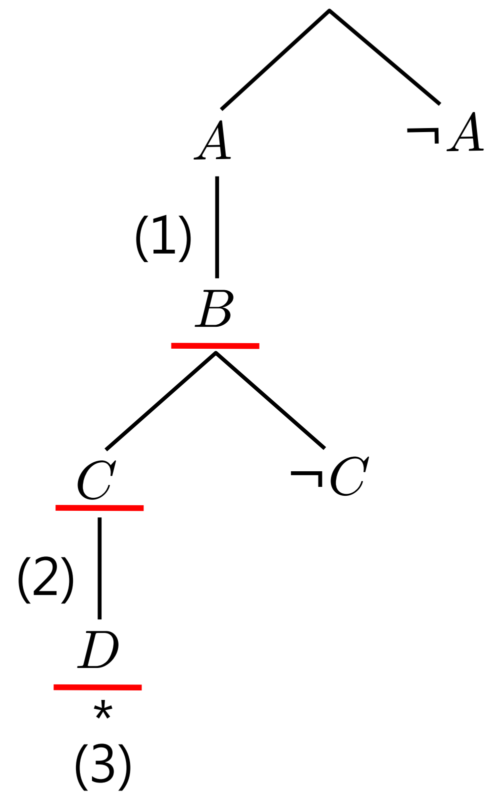
"Avoid making the same mistake twice"

$$\begin{array}{ll} \dots & \\ B \vee \neg A & (1) \\ D \vee \neg C & (2) \\ \underline{\neg D} \vee \underline{\neg B} \vee \underline{\neg C} & (3) \end{array}$$

Lemma Candidates by Resolution:

$\underline{\neg D} \vee \neg B \vee \neg C$

w/o Lemma



Lemma Learning

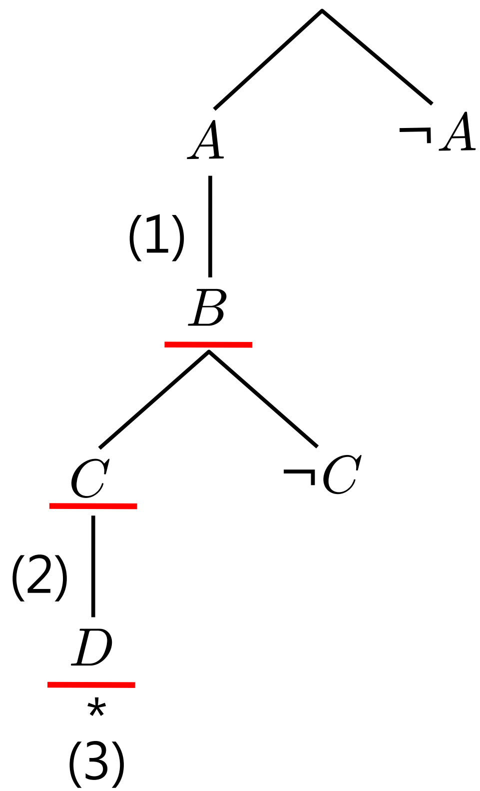
"Avoid making the same mistake twice"

$$\begin{array}{ll} & \ldots \\ B \lor \neg A & (1) \\ D \lor \neg C & (2) \\ \underline{\neg D} \lor \underline{\neg B} \lor \underline{\neg C} & (3) \end{array}$$

Lemma Candidates by Resolution:

$$\frac{\underline{\neg D} \lor \neg B \lor \neg C \qquad \underline{D} \lor \neg C}{\boxed{\underline{\neg B} \lor \neg C}}$$

w/o Lemma



Lemma Learning

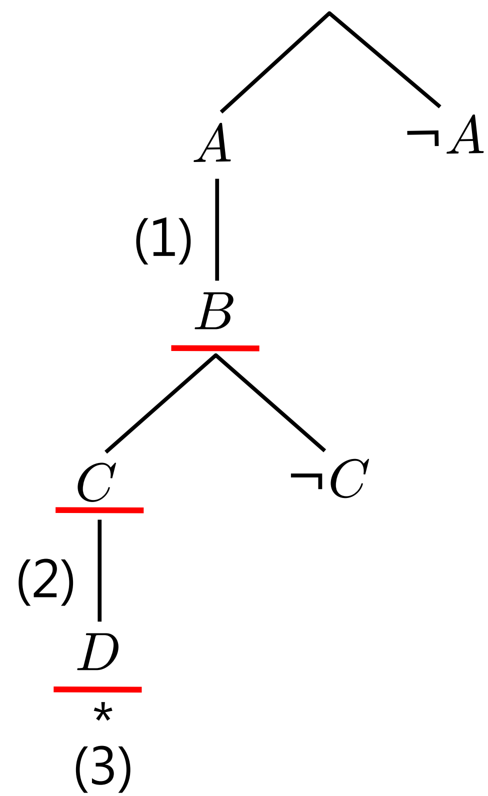
"Avoid making the same mistake twice"

$$\begin{array}{ll} \multicolumn{2}{c}{\dots} \\ B \vee \neg A & (1) \\ D \vee \neg C & (2) \\ \underline{\neg D} \vee \underline{\neg B} \vee \underline{\neg C} & (3) \end{array}$$

Lemma Candidates by Resolution:

$$\frac{\dfrac{\underline{\neg D} \vee \neg B \vee \neg C \qquad \underline{D} \vee \neg C}{\boxed{\underline{\neg B} \vee \neg C}} \qquad \underline{B} \vee \neg A}{\boxed{\neg C \vee \neg A}}$$

w/o Lemma



Lemma Learning

"Avoid making the same mistake twice"

$$\cdots$$

$$B \vee \neg A \qquad (1)$$

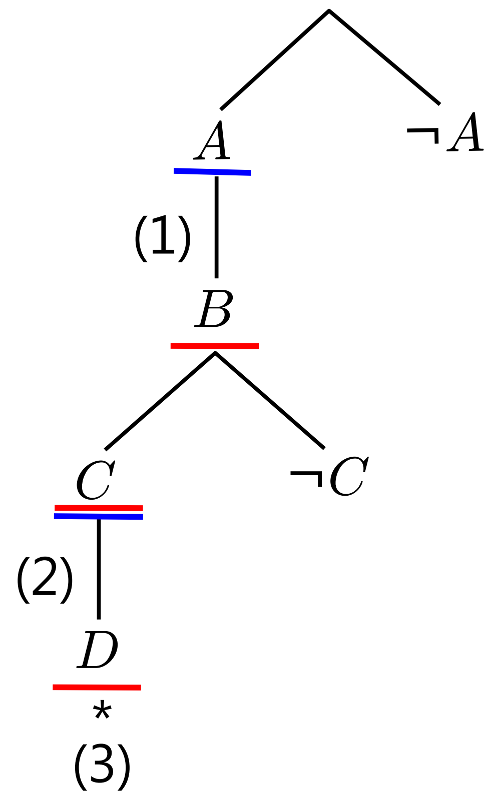
$$D \vee \neg C \qquad (2)$$

$$\underline{\neg D} \vee \underline{\neg B} \vee \underline{\neg C} \qquad (3)$$

Lemma Candidates by Resolution:

$$\dfrac{\dfrac{\underline{\neg D} \vee \neg B \vee \neg C \qquad \underline{D} \vee \neg C}{\boxed{\underline{\neg B} \vee \neg C}} \qquad \underline{B} \vee \neg A}{\boxed{\underline{\neg C \vee \neg A}}}$$

w/o Lemma **With Lemma**



Lemma Learning

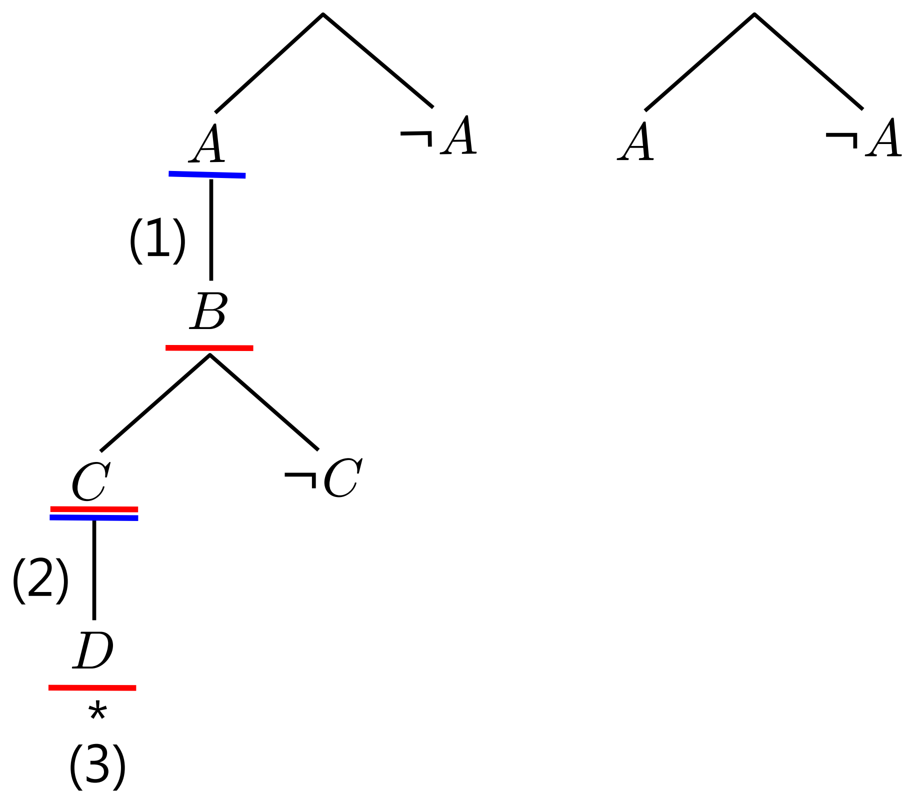
"Avoid making the same mistake twice"

$$\begin{array}{ll} \ldots & \\ B \lor \neg A & (1) \\ D \lor \neg C & (2) \\ \underline{\neg D} \lor \underline{\neg B} \lor \underline{\neg C} & (3) \end{array}$$

Lemma Candidates by Resolution:

$$\dfrac{\dfrac{\underline{\neg D} \lor \neg B \lor \neg C \qquad \underline{D} \lor \neg C}{\boxed{\underline{\neg B} \lor \neg C}} \qquad \underline{B} \lor \neg A}{\boxed{\underline{\neg C \lor \neg A}}}$$

w/o Lemma **With Lemma**



Lemma Learning

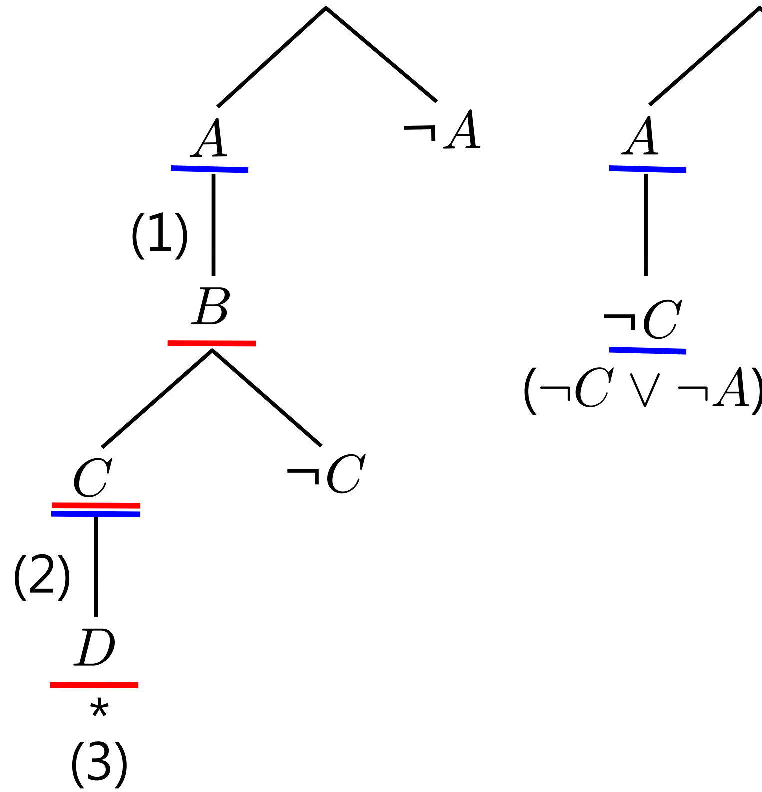
"Avoid making the same mistake twice"

$$\begin{array}{lr} \multicolumn{2}{c}{\cdots} \\ B \vee \neg A & (1) \\ D \vee \neg C & (2) \\ \underline{\neg D} \vee \underline{\neg B} \vee \underline{\neg C} & (3) \end{array}$$

Lemma Candidates by Resolution:

$$\dfrac{\dfrac{\underline{\neg D} \vee \neg B \vee \neg C \qquad \underline{D} \vee \neg C}{\boxed{\underline{\neg B} \vee \neg C}} \qquad \underline{B} \vee \neg A}{\boxed{\underline{\neg C \vee \neg A}}}$$

w/o Lemma **With Lemma**



Further Information

The ideas described so far heve been implemented in the SAT checker zChaff:

Lintao Zhang and Sharad Malik. The Quest for Efficient Boolean Satisfiability Solvers, Proc. CADE-18, LNAI 2392, pp. 295–312, Springer, 2002.

Other Overviews

Robert Nieuwenhuis, Albert Oliveras, Cesare Tinelli. Solvin SAT and SAT Modulo Theories: From an abstract Davis-Putnam-Logemann-Loveland precedure to DPLL(T), pp 937–977, Journal of the ACM, 53(6), 2006.

Armin Biere and Marijn Heule and Hans van Maaren and Toby Walsh. Handbook of Satisability, IOS Press, 2009.

Contents

Introduction

Logics and Reasoning Service (already done)

Methods for Automated Theorem Proving

Overview of some widely used general methods

- Propositional SAT solving
- First-order logic and clause normal forms
- Proof Procedures Based on Herbrand's Theorem
- The Resolution calculus
- Model generation

Theory Reasoning

Methods to reason with specific background theories

- Paramodulation (Equality)
- Satisfiability Modulo Theories (SMT)
- Quantifier elimination for linear real arithmetic
- Combining multiple theories

First-Order Logic (FOL)

A_1: Socrates is a human

A_2: All humans are mortal

Recall: propositional logic: variables are statements ranging over {true/false}

$$\frac{\begin{array}{l} SocratesIsHuman \\ SocratesIsHuman \rightarrow SocratesIsMortal \end{array}}{SocratesIsMortal}$$

FOL: variables range over individual objects

$$\frac{\begin{array}{l} human(socrates) \\ \forall x.\ (human(x) \rightarrow mortal(x)) \end{array}}{mortal(socrates)}$$

First-Order Logic Quiz

A_1: Socrates is a human

A_2: All humans are mortal

Translation into first-order logic:

A_1: $\mathit{human}(\mathit{socrates})$

A_2: $\forall x\ (\mathit{human}(x) \rightarrow \mathit{mortal}(x))$

Which of the following (non-)entailment statements hold true?

1. $\{A_1,\ A_2\} \models \mathit{mortal}(\mathit{socrates})$
2. $\{A_1,\ A_2\} \models \mathit{mortal}(\mathit{apollo})$
3. $\{A_1,\ A_2\} \not\models \mathit{mortal}(\mathit{socrates})$
4. $\{A_1,\ A_2\} \not\models \mathit{mortal}(\mathit{apollo})$
5. $\{A_1,\ A_2\} \models \neg\mathit{mortal}(\mathit{socrates})$
6. $\{A_1,\ A_2\} \models \neg\mathit{mortal}(\mathit{apollo})$

First-Order Logic Reasoning Services

Formula: First-order logic formula ϕ (e.g. the n-queens formulas above)

Usually with equality $=$

Sometimes from syntactically resricted fragment (e.g., Description logics)

Question: Is ϕ formula valid? (satisfiable?, entailed by another formula?)

Calculi: Superposition (Resolution), Instance-based methods, Tableaux, ...

Issues

- Efficient treatment of equality
- Decision procedure for sub-languages or useful reductions?
- Built-in inference rules for arrays, lists, arithmetics (still open research)

First-Order Logic

"The function f is continuous", expressed in (first-order) predicate logic:

$$\forall\varepsilon(0 < \varepsilon \to \forall a \exists \delta(0 < \delta \land \forall x(|x - a| < \delta \to |f(x) - f(a)| < \varepsilon)))$$

Underlying Language

Variables ε, a, δ, x

Function symbols 0, $|\_|$, $\_ - \_$, $f(\_)$

Terms are well-formed expressions over variables and function symbols, e.g. $|f(x) - f(a)|$

Predicate symbols $\_ < \_$, $\_ = \_$

Atoms are applications of predicate symbols to terms, e.g., $|f(x) - f(a)| < \varepsilon$

Boolean connectives $\land$, $\lor$, $\to$, $\neg$

Quantifiers $\forall$, $\exists$

The function symbols and predicate symbols comprise a signature Σ

First-Order Logic

"The function f is continuous", expressed in (first-order) predicate logic:

$$\forall \varepsilon (0 < \varepsilon \rightarrow \forall a \exists \delta (0 < \delta \wedge \forall x (|x - a| < \delta \rightarrow |f(x) - f(a)| < \varepsilon)))$$

Semantics: (Σ-)Algebras, or (Σ-)Interpretations

Universe (aka Domain): Set U

Variables $\mapsto$ values in U (mapping is called "assignment")

Function symbols $\mapsto$ (total) functions over U

Predicate symbols $\mapsto$ relations over U

Boolean connectives $\mapsto$ the usual boolean functions

Quantifiers $\mapsto$ "for all ... holds", "there is a ..., such that"

Terms $\mapsto$ values in U

Formulas $\mapsto$ Boolean (Truth-) values

Semantics - Example

Let Σ_{PA} be the standard signature of Peano Arithmetic

The standard interpretation $\mathbb{N}$ for Peano Arithmetic then is:

$$
\begin{array}{rcl}
U_{\mathbb{N}} & = & \{0, 1, 2, \ldots\} \\
0_{\mathbb{N}} & : & 0 \\
s_{\mathbb{N}} & : & n \mapsto n + 1 \\
+_{\mathbb{N}} & : & (n, m) \mapsto n + m \\
*_{\mathbb{N}} & : & (n, m) \mapsto n * m \\
\leq_{\mathbb{N}} & = & \{(n, m) \mid n \text{ less than or equal to } m\} \\
<_{\mathbb{N}} & = & \{(n, m) \mid n \text{ less than } m\}
\end{array}
$$

Note that $\mathbb{N}$ is just one out of many possible Σ_{PA}-interpretations

Semantics - Example

Evaluation of terms and formulas

Under the interpretation $\mathbb{N}$ and the assignment $\beta : x \mapsto 1, y \mapsto 3$ (to evaluate the free variables) we obtain

$$
\begin{array}{lcll}
(\mathbb{N}, \beta)(s(x) + s(0)) & = & 3 & \\
(\mathbb{N}, \beta)(x + y \doteq s(y)) & = & \mathit{True} & \\
(\mathbb{N}, \beta)(\forall z\; z \leq y) & = & \mathit{False} & \\
(\mathbb{N}, \beta)(\forall x \exists y\; x < y) & = & \mathit{True} & \\
\mathbb{N}(\forall x \exists y\; x < y) & = & \mathit{True} & \quad \text{(Short notation when } \beta \text{ irrelevant)}
\end{array}
$$

Important Basic Notion: Model

If ϕ is a closed formula, then, instead of $I(\phi) = \mathit{True}$ one writes

$$I \models \phi \qquad \text{(“}I\text{ is a model of }\phi\text{”)}$$

E.g. $\mathbb{N} \models \forall x \exists y\; x < y$

Reasoning Services Semantically

E.g. "entailment":

$$\text{Axioms over } \mathbb{R} \wedge \text{continuous}(f) \wedge \text{continuous}(g) \models \text{continuous}(f+g) \text{ ?}$$

Model(I,ϕ): $I \models \phi$? (Is I a model for ϕ?)

Validity(ϕ): $\models \phi$? ($I \models \phi$ for every interpretation?)

Satisfiability(ϕ): ϕ satisfiable? ($I \models \phi$ for some interpretation?)

Entailment(ϕ,ψ): $\phi \models \psi$? (does ϕ entail ψ?, i.e.
for every interpretation I: if $I \models \phi$ then $I \models \psi$?)

Additional complication: fix interpretation of some symbols (as in $\mathbb{N}$ above)

Reasoning Services Semantically

E.g. “entailment”:

$$\text{Axioms over } \mathbb{R} \wedge \text{continuous}(f) \wedge \text{continuous}(g) \models \text{continuous}(f+g) \ ?$$

Model(I,ϕ): $I \models \phi$? (Is I a model for ϕ?)

Validity(ϕ): $\models \phi$? ($I \models \phi$ for every interpretation?)

Satisfiability(ϕ): ϕ satisfiable? ($I \models \phi$ for some interpretation?)

Entailment(ϕ,ψ): $\phi \models \psi$? (does ϕ entail ψ?, i.e.
for every interpretation I: if $I \models \phi$ then $I \models \psi$?)

Additional complication: fix interpretation of some symbols (as in $\mathbb{N}$ above)

In the following focus on “entailment”

Reduction of Entailment to Unsatisfiability

- Suppose we want to prove an entailment $\phi \models \psi$

Reduction of Entailment to Unsatisfiability

- Suppose we want to prove an entailment $\phi \models \psi$
- Equivalently, prove $\models \phi \rightarrow \psi$, i.e. that $\phi \rightarrow \psi$ is valid

Reduction of Entailment to Unsatisfiability

- Suppose we want to prove an entailment $\phi \models \psi$
- Equivalently, prove $\models \phi \rightarrow \psi$, i.e. that $\phi \rightarrow \psi$ is valid
- Equivalently, prove that $\neg(\phi \rightarrow \psi)$ is not satisfiable (unsatisfiable)

Reduction of Entailment to Unsatisfiability

- Suppose we want to prove an entailment $\phi \models \psi$
- Equivalently, prove $\models \phi \rightarrow \psi$, i.e. that $\phi \rightarrow \psi$ is valid
- Equivalently, prove that $\neg(\phi \rightarrow \psi)$ is not satisfiable (unsatisfiable)
- Equivalently, prove that $\phi \wedge \neg\psi$ is unsatisfiable

Reduction of Entailment to Unsatisfiability

- Suppose we want to prove an entailment $\phi \models \psi$
- Equivalently, prove $\models \phi \rightarrow \psi$, i.e. that $\phi \rightarrow \psi$ is valid
- Equivalently, prove that $\neg(\phi \rightarrow \psi)$ is not satisfiable (unsatisfiable)
- Equivalently, prove that $\phi \wedge \neg\psi$ is unsatisfiable

Basis for refutational theorem proving

Reduction of Entailment to Unsatisfiability

- Suppose we want to prove an entailment $\phi \models \psi$
- Equivalently, prove $\models \phi \rightarrow \psi$, i.e. that $\phi \rightarrow \psi$ is valid
- Equivalently, prove that $\neg(\phi \rightarrow \psi)$ is not satisfiable (unsatisfiable)
- Equivalently, prove that $\phi \wedge \neg\psi$ is unsatisfiable

Basis for refutational theorem proving

Dual problem, much harder: to show $\phi \not\models \psi$ find a model of $\phi \wedge \neg\psi$.

Normal Forms

Most first-order theorem provers take formulas in clause normal form

Why Normal Forms?

- Reduction of logical concepts (operators, quantifiers)
- Reduction of syntactical structure (nesting of subformulas)
- Can be exploited for efficient data structures and control

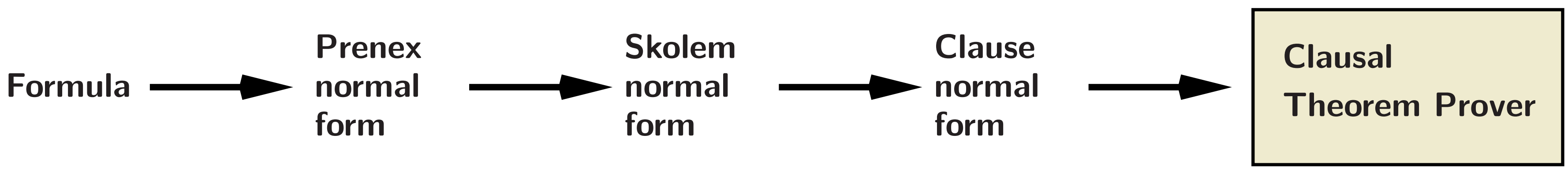
Normal Forms

Most first-order theorem provers take formulas in clause normal form

Why Normal Forms?

- Reduction of logical concepts (operators, quantifiers)
- Reduction of syntactical structure (nesting of subformulas)
- Can be exploited for efficient data structures and control

Translation into Clause Normal Form



Prop: the given formula and its clause normal form are equi-satisfiable

Prenex Normal Form

Prenex formulas have the form

$$Q_1 x_1 \ldots Q_n x_n \; F,$$

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

Prenex Normal Form

Prenex formulas have the form

$$Q_1 x_1 \ldots Q_n x_n \; F,$$

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

Computing prenex normal form by the rewrite relation $\Rightarrow_P$:

$$\begin{array}{rcll}
(F \leftrightarrow G) & \Rightarrow_P & (F \rightarrow G) \wedge (G \rightarrow F) & \\
\neg Q x F & \Rightarrow_P & \overline{Q} x \neg F & (\neg Q) \\
(Q x F \;\rho\; G) & \Rightarrow_P & Q y (F[y/x] \;\rho\; G),\; y \text{ fresh},\; \rho \in \{\wedge, \vee\} & \\
(Q x F \rightarrow G) & \Rightarrow_P & \overline{Q} y (F[y/x] \rightarrow G),\; y \text{ fresh} & \\
(F \;\rho\; Q x G) & \Rightarrow_P & Q y (F \;\rho\; G[y/x]),\; y \text{ fresh},\; \rho \in \{\wedge, \vee, \rightarrow\} &
\end{array}$$

$\overline{Q}$ denotes the quantifier dual to Q, i.e., $\overline{\forall} = \exists$ and $\overline{\exists} = \forall$.
$F[y/x]$ is obtained from F by replacing every free (not bound) occurrence of x in F by y. An occurrence of x in F is bound if this occurrence is within a subformula $Qx\; G$ of F.

In the Example

$$\forall\varepsilon(0 < \varepsilon \to \forall a\exists\delta(0 < \delta \land \forall x(|x - a| < \delta \to |f(x) - f(a)| < \varepsilon)))$$

$$\Rightarrow_P$$

$$\forall\varepsilon\forall a(0 < \varepsilon \to \exists\delta(0 < \delta \land \forall x(|x - a| < \delta \to |f(x) - f(a)| < \varepsilon)))$$

$$\Rightarrow_P$$

$$\forall\varepsilon\forall a\exists\delta(0 < \varepsilon \to 0 < \delta \land \forall x(|x - a| < \delta \to |f(x) - f(a)| < \varepsilon))$$

$$\Rightarrow_P$$

$$\forall\varepsilon\forall a\exists\delta(0 < \varepsilon \to \forall x(0 < \delta \land |x - a| < \delta \to |f(x) - f(a)| < \varepsilon))$$

$$\Rightarrow_P$$

$$\forall\varepsilon\forall a\exists\delta\forall x(0 < \varepsilon \to (0 < \delta \land (|x - a| < \delta \to |f(x) - f(a)| < \varepsilon)))$$

Skolem Normal Form

Formula $\longrightarrow$ Prenex normal form $\longrightarrow$ Skolem normal form $\longrightarrow$ Clause normal form $\longrightarrow$ Clausal Theorem Prover

Intuition: replacement of $\exists y$ by a concrete choice function computing y from all the arguments y depends on.

Transformation $\Rightarrow_S$

$$\forall x_1, \ldots, x_n \exists y \; F \quad \Rightarrow_S \quad \forall x_1, \ldots, x_n \; F[f(x_1, \ldots, x_n)/y]$$

where f/n is a new function symbol (Skolem function).

Skolem Normal Form

Formula $\longrightarrow$ Prenex normal form $\longrightarrow$ Skolem normal form $\longrightarrow$ Clause normal form $\longrightarrow$ **Clausal Theorem Prover**

Intuition: replacement of $\exists y$ by a concrete choice function computing y from all the arguments y depends on.

Transformation $\Rightarrow_S$

$$\forall x_1, \ldots, x_n \exists y \; F \quad \Rightarrow_S \quad \forall x_1, \ldots, x_n \; F[f(x_1, \ldots, x_n)/y]$$

where f/n is a new function symbol (Skolem function).

In the Example

$$\forall \varepsilon \forall a \exists \delta \forall x (0 < \varepsilon \rightarrow 0 < \delta \wedge (|x - a| < \delta \rightarrow |f(x) - f(a)| < \varepsilon))$$

$$\Rightarrow_S$$

$$\forall \varepsilon \forall a \forall x (0 < \varepsilon \rightarrow 0 < d(\varepsilon, a) \wedge (|x - a| < d(\varepsilon, a) \rightarrow |f(x) - f(a)| < \varepsilon))$$

Clausal Normal Form (Conjunctive Normal Form)

Rules to convert the matrix of the formula in Skolem normal form into a conjunction of disjunctions of literals:

$$
\begin{array}{rcl}
(F \leftrightarrow G) & \Rightarrow_K & (F \rightarrow G) \land (G \rightarrow F) \\
(F \rightarrow G) & \Rightarrow_K & (\neg F \lor G) \\
\neg(F \lor G) & \Rightarrow_K & (\neg F \land \neg G) \\
\neg(F \land G) & \Rightarrow_K & (\neg F \lor \neg G) \\
\neg\neg F & \Rightarrow_K & F \\
(F \land G) \lor H & \Rightarrow_K & (F \lor H) \land (G \lor H) \\
(F \land \top) & \Rightarrow_K & F \\
(F \land \bot) & \Rightarrow_K & \bot \\
(F \lor \top) & \Rightarrow_K & \top \\
(F \lor \bot) & \Rightarrow_K & F
\end{array}
$$

They are to be applied modulo commutativity of $\land$ and $\lor$

In the Example

$$\forall\varepsilon\forall a\forall x(0 < \varepsilon \to 0 < d(\varepsilon, a) \wedge (|x - a| < d(\varepsilon, a) \to |f(x) - f(a)| < \varepsilon))$$

$$\Rightarrow_K$$

$$0 < d(\varepsilon, a) \;\vee\; \neg(0 < \varepsilon)$$

$$\neg(|x - a| < d(\varepsilon, a)) \;\vee\; |f(x) - f(a)| < \varepsilon \;\vee\; \neg(0 < \varepsilon)$$

Note: The universal quantifiers for the variables ε, a and x, as well as the conjunction symbol $\wedge$ between the clauses are not written, for convenience

The Complete Picture

$$
\begin{aligned}
F \quad &\overset{*}{\Rightarrow}_P \quad Q_1 y_1 \ldots Q_n y_n \; G && (G \text{ quantifier-free}) \\
&\overset{*}{\Rightarrow}_S \quad \forall x_1, \ldots, x_m \; H && (m \leq n,\ H \text{ quantifier-free}) \\
&\overset{*}{\Rightarrow}_K \quad \underbrace{\forall x_1, \ldots, x_m}_{\text{leave out}} \; \bigwedge_{i=1}^{k} \underbrace{\bigvee_{j=1}^{n_i} L_{ij}}_{\text{clauses } C_i}
\end{aligned}
$$

Notions

An atom is the (arity respecting) application of a predicate symbol to some terms. A literal L is an atom or a negated atom. A clause is a disjunction of literals $L_1 \vee \cdots \vee L_n$, where $n \geq 0$. The empty clause is written as $\square$. A clause set is a set of clauses, The set $N = \{C_1, \ldots, C_k\}$ is called the clausal (normal) form (CNF) of F.

Note: Variables in clauses are implicitly universally quantified

Where are we?

Instead of showing that a formula F is unsatisfiable, the proof problem from now is to show that its CNF N is unsatisfiable

A CNF provides a simple syntactic structure, but does not give a clue how to prove unsatisfiability. The naive approach of "checking all interpretations" does not work: In general, there are infinitely many, even uncountably many interpretations for a signature Σ.

So how to do that? "Herbrand theory" provides the answer.

Contents

Introduction

Logics and Reasoning Service (already done)

Methods for Automated Theorem Proving

Overview of some widely used general methods

- Propositional SAT solving
- First-order logic and clause normal forms
- Proof Procedures Based on Herbrand's Theorem
- The Resolution calculus
- Model generation

Theory Reasoning

Methods to reason with specific background theories

- Paramodulation (Equality)
- Satisfiability Modulo Theories (SMT)
- Quantifier elimination for linear real arithmetic
- Combining multiple theories

Proof Procedures Based on Herbrand's Theorem

Proving unsatisfiabilty of a clause set becomes feasible (semi-decidable) by working with the set of its ground instances instead. Plan of attack:

Definition: A ground instance of a clause is obtained by replacing each of its variables by some variable-free term ("ground term")

Proof Procedures Based on Herbrand's Theorem

Proving unsatisfiabilty of a clause set becomes feasible (semi-decidable) by working with the set of its ground instances instead. Plan of attack:

Definition: A ground instance of a clause is obtained by replacing each of its variables by some variable-free term ("ground term")

Proposition (Herbrand): Let N be a clause set and N^{gr} be the set of all ground instances of all clauses in N.
N is unsatisfiable iff N^{gr} is unsatisfiable wrt. Herbrand interpretations (essentially: propositional-logic unsatisfiable)

Proof Procedures Based on Herbrand's Theorem

Proving unsatisfiabilty of a clause set becomes feasible (semi-decidable) by working with the set of its ground instances instead. Plan of attack:

Definition: A ground instance of a clause is obtained by replacing each of its variables by some variable-free term ("ground term")

Proposition (Herbrand): Let N be a clause set and N^{gr} be the set of all ground instances of all clauses in N.
N is unsatisfiable iff N^{gr} is unsatisfiable wrt. Herbrand interpretations (essentially: propositional-logic unsatisfiable)

Proposition (compactness): N^{gr} is unsatisfiable wrt. Herbrand interpretations iff some finite subset $M \subseteq N^{\mathrm{gr}}$ is unsatisfiable wrt. Herbrand interpretations

Proof Procedures Based on Herbrand's Theorem

Proving unsatisfiabilty of a clause set becomes feasible (semi-decidable) by working with the set of its ground instances instead. Plan of attack:

Definition: A ground instance of a clause is obtained by replacing each of its variables by some variable-free term ("ground term")

Proposition (Herbrand): Let N be a clause set and N^{gr} be the set of all ground instances of all clauses in N.
N is unsatisfiable iff N^{gr} is unsatisfiable wrt. Herbrand interpretations
(essentially: propositional-logic unsatisfiable)

Proposition (compactness): N^{gr} is unsatisfiable wrt. Herbrand interpretations iff some finite subset $M \subseteq N^{\mathrm{gr}}$ is unsatisfiable wrt. Herbrand interpretations

Propositional logic phase: Decide the satisfability of such finite sets M with a SAT solver; Gilmore's method

Proof Procedures Based on Herbrand's Theorem

Proving unsatisfiabilty of a clause set becomes feasible (semi-decidable) by working with the set of its ground instances instead. Plan of attack:

Definition: A ground instance of a clause is obtained by replacing each of its variables by some variable-free term ("ground term")

Proposition (Herbrand): Let N be a clause set and N^{gr} be the set of all ground instances of all clauses in N.
N is unsatisfiable iff N^{gr} is unsatisfiable wrt. Herbrand interpretations (essentially: propositional-logic unsatisfiable)

Proposition (compactness): N^{gr} is unsatisfiable wrt. Herbrand interpretations iff some finite subset $M \subseteq N^{\mathrm{gr}}$ is unsatisfiable wrt. Herbrand interpretations

Propositional logic phase: Decide the satisfability of such finite sets M with a SAT solver; Gilmore's method

The above recasts usual notions of "Herbrand theory" in our application to clause logic. "Herbrand's Theorem" (1930s) is a stronger version of the two propositions above combined

Ground Instances

Example: Let $N = \{P(a),\ \neg P(x) \vee P(f(x)),\ Q(y,z),\ \neg P(f(f(a)))\}$

Ground Instances

Example: Let $N = \{P(a),\ \neg P(x) \vee P(f(x)),\ Q(y, z),\ \neg P(f(f(a)))\}$

The underlying signature is $\Sigma = \{P/1,\ Q/2\} \cup \{a/0,\ f/2\}$

Ground Instances

Example: Let $N = \{P(a),\ \neg P(x) \vee P(f(x)),\ Q(y,z),\ \neg P(f(f(a)))\}$

The underlying signature is $\Sigma = \{P/1,\ Q/2\} \cup \{a/0,\ f/2\}$

The ground terms (of Σ) are $U^H = \{a,\ f(a),\ f(f(a)), f(f(f(a))), \ldots$
(aka Herbrand universe)

Ground Instances

Example: Let $N = \{P(a),\ \neg P(x) \vee P(f(x)),\ Q(y,z),\ \neg P(f(f(a)))\}$

The underlying signature is $\Sigma = \{P/1,\ Q/2\} \cup \{a/0,\ f/2\}$

The ground terms (of Σ) are $U^H = \{a,\ f(a),\ f(f(a)), f(f(f(a))), \ldots$
(aka Herbrand universe)

The ground instances of N is the set

$$
\begin{aligned}
N^{\mathrm{gr}} = {} & \{P(a)\} \\
& \cup \{\neg P(a) \vee P(f(a)),\ \neg P(f(a)) \vee P(f(f(a))), \\
& \qquad \neg P(f(f(a))) \vee P(f(f(f(a)))), \ldots\} \\
& \cup \{Q(a,a),\ Q(a,f(a)),\ Q(f(a),a),\ Q(f(a),f(a)), \ldots\} \\
& \cup \{\neg P(f(f(a)))\}
\end{aligned}
$$

Mapping to Propositional Logic

The Herbrand base, i.e., the set of all ground atoms is

$$HB = \{\underbrace{P(a)}_{A_0},\ \underbrace{P(f(a))}_{A_1},\ \underbrace{P(f(f(a)))}_{A_2},\ \underbrace{P(f(f(f(a))))}_{A_3}, \ldots\}$$

$$\cup\, \{\underbrace{Q(a,a)}_{B_0},\ \underbrace{Q(a,f(a))}_{B_1},\ \underbrace{Q(f(a),a)}_{B_2},\ \underbrace{Q(f(a),f(a))}_{B_3}, \ldots\}$$

By construction, every atom in N^{gr} occurs in HB

Replace in N^{gr} every (ground) atom by its propositional counterpart:

$$\begin{aligned} N^{\mathrm{gr}}_{\mathrm{prop}} = {} & \{A_0\} \\ & \cup \{\neg A_0 \vee A_1,\ \neg A_1 \vee A_2, \neg A_2 \vee A_3, \ldots\} \\ & \cup \{B_0,\ B_1,\ B_2,\ B_3, \ldots\} \\ & \cup \{\neg A_2\} \end{aligned}$$

The subset $\{A_0,\ \neg A_0 \vee A_1,\ \neg A_1 \vee A_2,\ \neg A_2\}$ is unsatisfiable, hence so is N.

Herbrand Proposition

A Herbrand interpretation I is an interpretation such that (in the example)

$$U = U^H = \{a, f(a), f(f(a)), f(f(f(a))), \ldots$$

$$a : a$$

$$f : \; t \mapsto f(t)$$

In every Herbrand interpretation every ground term is always interpreted as "itself", e.g. $I(f(f(a))) = f(f(a))$

The universe U^H of ground terms justifies expanding clauses into their ground instances instead of using a separate mapping β from variables to U

With the universe U and the interpretation of the function symbols uniquely fixed in every Herbrand interpretation, Herbrand interpretations vary only with the interpretation of the predicate symbols.

This justifies to specify a Herbrand interpretation as a subset of HB, those atoms that are *True* by definition. In the example, e.g., $I = \{P(a), Q(a, f(a))\}$

Herbrand Proposition

Prove idea for the non-trivial direction

- Suppose N has a model $J \models N$
  E.g., $U_J = \mathbb{N}$, $a_J : 0$, $f_J : n \mapsto n + 1$, $P_J : n \mapsto n \geq 0$, $Q_J : m, n \mapsto m > n$

Herbrand Proposition

Prove idea for the non-trivial direction

- Suppose N has a model $J \models N$

  E.g., $U_J = \mathbb{N}$, $a_J : 0$, $f_J : n \mapsto n+1$, $P_J : n \mapsto n \geq 0$, $Q_J : m, n \mapsto m > n$

- Define a Herbrand interpretation $I \subseteq HB$ as follows:

  For every ground atom $K \in HB$ put $K \in I$ iff $J(K) = True$

  That is, evaluate K in J to get a (the same) truth value for K in I.

  Example : $P(f(a)) \in I$ as $0 + 1 \geq 0$

Herbrand Proposition

Prove idea for the non-trivial direction

- Suppose N has a model $J \models N$
  E.g., $U_J = \mathbb{N}$, $a_J : 0$, $f_J : n \mapsto n + 1$, $P_J : n \mapsto n \geq 0$, $Q_J : m, n \mapsto m > n$
- Define a Herbrand interpretation $I \subseteq HB$ as follows:

  For every ground atom $K \in HB$ put $K \in I$ iff $J(K) = True$

  That is, evaluate K in J to get a (the same) truth value for K in I.
  Example : $P(f(a)) \in I$ as $0 + 1 \geq 0$
- Given an atom $A[x]$ (with free variables x) and a ground term t.
  Then $I \models A[t]$ iff $(J, [x \mapsto J(t)] \models A[x]$.
  Example: let $A[x] = P(f(x))$ and $t = f(f(a))$

$$I \models P(f(\ f(f(a))\ ))$$

$$\text{iff } J \models P(f(\ f(f(a))\ )) \quad \text{(By definition)}$$

$$\text{iff } J, [x \mapsto J(f(f(a)))] \models P(f(x)) \qquad \text{(Use structural induction)}$$

Herbrand Proposition

Prove idea for the non-trivial direction

- Suppose N has a model $J \models N$
  E.g., $U_J = \mathbb{N}$, $a_J : 0$, $f_J : n \mapsto n+1$, $P_J : n \mapsto n \geq 0$, $Q_J : m, n \mapsto m > n$
- Define a Herbrand interpretation $I \subseteq HB$ as follows:

  For every ground atom $K \in HB$ put $K \in I$ iff $J(K) = True$

  That is, evaluate K in J to get a (the same) truth value for K in I.
  Example : $P(f(a)) \in I$ as $0 + 1 \geq 0$
- Given an atom $A[x]$ (with free variables x) and a ground term t.
  Then $I \models A[t]$ iff $(J, [x \mapsto J(t)] \models A[x]$.
  Example: let $A[x] = P(f(x))$ and $t = f(f(a))$

  $$I \models P(f(\ f(f(a))\ ))$$

  iff $J \models P(f(\ f(f(a))\ ))$ (By definition)

  iff $J, [x \mapsto J(f(f(a)))] \models P(f(x))$ (Use structural induction)
- From that the proposition follows easily. Compactness: see whiteboard

Gilmore's Method - Based on Herbrand's Theorem

Preprocessing:

Given Formula		Clause Form
$\forall x\, \exists y\ P(y,x)$ $\land \forall z\ \neg P(z,a)$	$\longrightarrow$	$P(f(x),x)$ $\neg P(z,a)$

Outer loop:
Grounding

Inner loop:
Propositional
Method

Gilmore's Method - Based on Herbrand's Theorem

Preprocessing:

Given Formula

$\forall x\ \exists y\ P(y, x)$
$\wedge \forall z\ \neg P(z, a)$

$\longrightarrow$

Clause Form

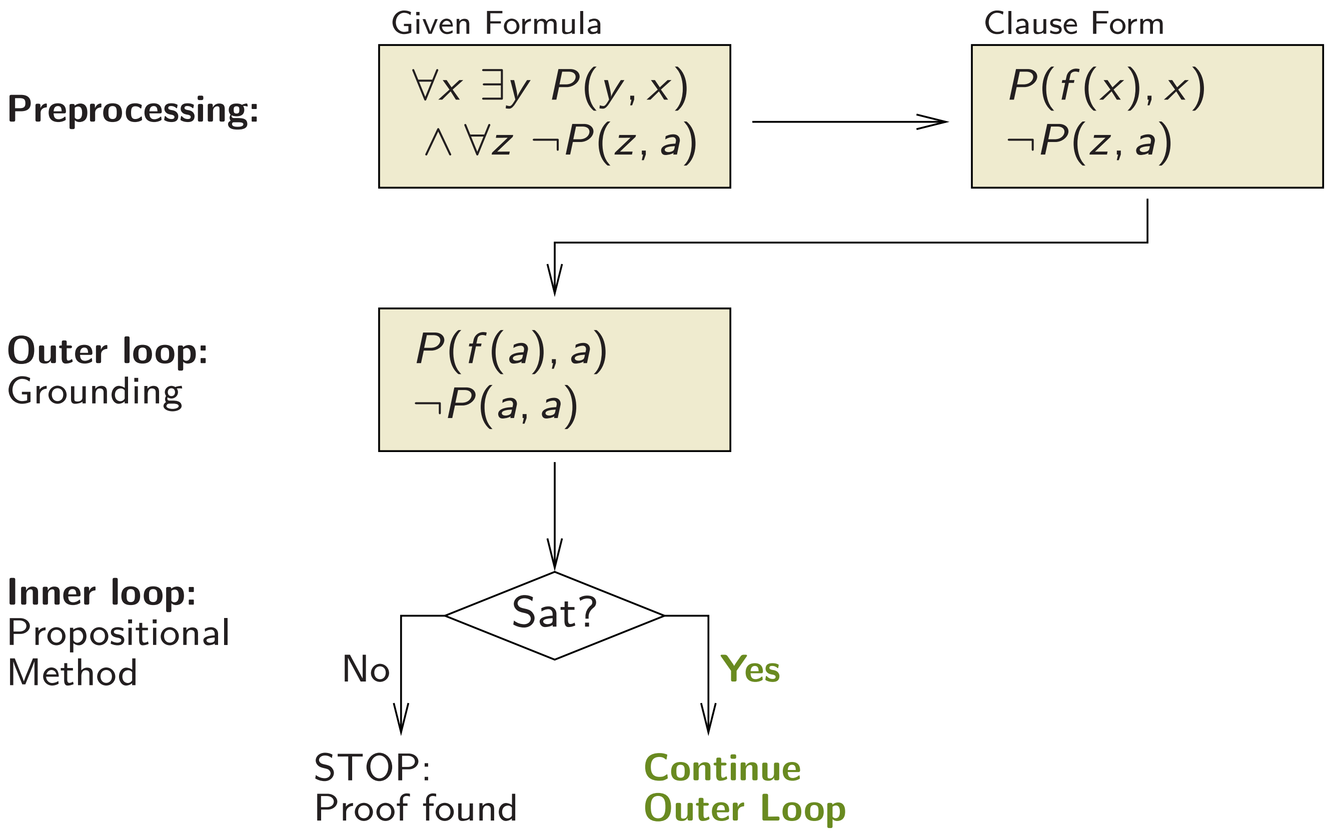
$P(f(x), x)$
$\neg P(z, a)$

Outer loop:
Grounding

$P(f(a), a)$
$\neg P(a, a)$

Inner loop:
Propositional
Method

Gilmore’s Method - Based on Herbrand’s Theorem



Gilmore's Method - Based on Herbrand's Theorem

Preprocessing:

Given Formula

$\forall x \, \exists y \; P(y, x)$
$\wedge \forall z \, \neg P(z, a)$

$\longrightarrow$

Clause Form

$P(f(x), x)$
$\neg P(z, a)$

Outer loop:
Grounding

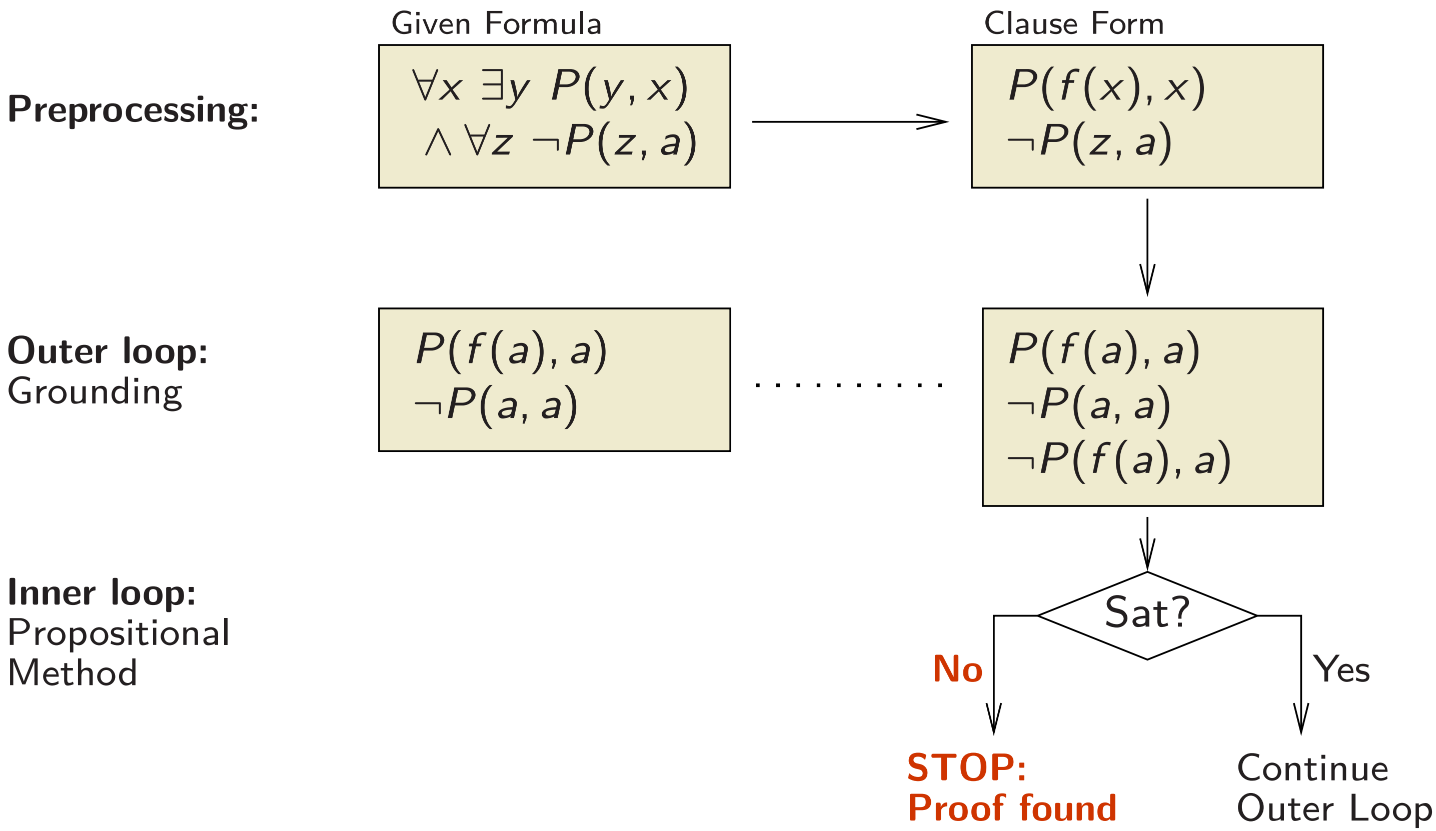
$P(f(a), a)$
$\neg P(a, a)$

. . . .

$P(f(a), a)$
$\neg P(a, a)$
$\neg P(f(a), a)$

Inner loop:
Propositional
Method

Gilmore's Method - Based on Herbrand's Theorem



Contents

Introduction

Logics and Reasoning Service (already done)

Methods for Automated Theorem Proving

Overview of some widely used general methods

- Propositional SAT solving
- First-order logic and clause normal forms
- Proof Procedures Based on Herbrand's Theorem
- The Resolution calculus
- Model generation

Theory Reasoning

Methods to reason with specific background theories

- Paramodulation (Equality)
- Satisfiability Modulo Theories (SMT)
- Quantifier elimination for linear real arithmetic
- Combining multiple theories

The Resolution Calculus

- Gilmore's method reduces proof search in first-order logic to propositional logic unsatisfiability problems

The Resolution Calculus

- Gilmore's method reduces proof search in first-order logic to propositional logic unsatisfiability problems
- Main problem is the unguided generation of (very many) ground clauses

The Resolution Calculus

- Gilmore's method reduces proof search in first-order logic to propositional logic unsatisfiability problems
- Main problem is the unguided generation of (very many) ground clauses
- All modern calculi address this problem in one way or another, e.g.
  - Avoidance: Resolution calculi do not need to generate the ground instances at all

    Resolution inferences operate directly on clauses, not on their ground instances
  - Guidance: Instance-Based Methods are similar to Gilmore's method but generate ground instances in a guided way (see below)

The Resolution Calculus

- Gilmore's method reduces proof search in first-order logic to propositional logic unsatisfiability problems
- Main problem is the unguided generation of (very many) ground clauses
- All modern calculi address this problem in one way or another, e.g.
  - Avoidance: Resolution calculi do not need to generate the ground instances at all

    Resolution inferences operate directly on clauses, not on their ground instances
  - Guidance: Instance-Based Methods are similar to Gilmore's method but generate ground instances in a guided way (see below)

Modern versions of the resolution calculus [Robinson 1965] are (still) the most important calculi for first-order theorem proving today

We first consider the special case for propositional logic

The Propositional Resolution Calculus

Propositional resolution inference rule:

$$\frac{C \vee A \qquad \neg A \vee D}{C \vee D}$$

Terminology: $C \vee D$: resolvent; A: resolved atom

Propositional (positive) factoring inference rule:

$$\frac{C \vee A \vee A}{C \vee A}$$

Terminology: $C \vee A$: factor

These are schematic inference rules:

C and D – propositional clauses

A – propositional atom

"$\vee$" is considered associative and commutative

Derivations

Let $N = \{C_1, \dots, C_k\}$ be a set of input clauses (propositional, for now).
A derivation (from N) is a sequence of the form

$$\underbrace{C_1, \dots, C_k}_{\substack{\text{Input}\\\text{clauses}}}, \underbrace{C_{k+1}, \dots, C_n, \dots}_{\substack{\text{Derived}\\\text{clauses}}}$$

such that for every $n \geq k + 1$

- C_n is a resolvent of C_i and C_j, for some $1 \leq i, j < n$, or
- C_n is a factor of C_i, for some $1 \leq i < n$.

A refutation (of N) is a derivation from N that contains the empty clause $\Box$

Important results:

Soundness: If there is a refutation of N then N is unsatisfiable

Completeness: If N is unsatisfiable then there is a refutation of N

Sample Refutation

1.	$\neg A \vee \neg A \vee B$	(given)
2.	$A \vee B$	(given)
3.	$\neg C \vee \neg B$	(given)
4.	C	(given)

Sample Refutation

1.	$\neg A \vee \neg A \vee B$	(given)
2.	$A \vee B$	(given)
3.	$\neg C \vee \neg B$	(given)
4.	C	(given)
5.	$\neg A \vee B \vee B$	(Res. 2. into 1.)

Sample Refutation

1.	$\neg A \vee \neg A \vee B$	(given)
2.	$A \vee B$	(given)
3.	$\neg C \vee \neg B$	(given)
4.	C	(given)
5.	$\neg A \vee B \vee B$	(Res. 2. into 1.)
6.	$\neg A \vee B$	(Fact. 5.)

Sample Refutation

1.	$\neg A \vee \neg A \vee B$	(given)
2.	$A \vee B$	(given)
3.	$\neg C \vee \neg B$	(given)
4.	C	(given)
5.	$\neg A \vee B \vee B$	(Res. 2. into 1.)
6.	$\neg A \vee B$	(Fact. 5.)
7.	$B \vee B$	(Res. 2. into 6.)

Sample Refutation

1.	$\neg A \vee \neg A \vee B$	(given)
2.	$A \vee B$	(given)
3.	$\neg C \vee \neg B$	(given)
4.	C	(given)
5.	$\neg A \vee B \vee B$	(Res. 2. into 1.)
6.	$\neg A \vee B$	(Fact. 5.)
7.	$B \vee B$	(Res. 2. into 6.)
8.	B	(Fact. 7.)

Sample Refutation

1.	$\neg A \vee \neg A \vee B$	(given)
2.	$A \vee B$	(given)
3.	$\neg C \vee \neg B$	(given)
4.	C	(given)
5.	$\neg A \vee B \vee B$	(Res. 2. into 1.)
6.	$\neg A \vee B$	(Fact. 5.)
7.	$B \vee B$	(Res. 2. into 6.)
8.	B	(Fact. 7.)
9.	$\neg C$	(Res. 8. into 3.)

Sample Refutation

1.	$\neg A \vee \neg A \vee B$	(given)
2.	$A \vee B$	(given)
3.	$\neg C \vee \neg B$	(given)
4.	C	(given)
5.	$\neg A \vee B \vee B$	(Res. 2. into 1.)
6.	$\neg A \vee B$	(Fact. 5.)
7.	$B \vee B$	(Res. 2. into 6.)
8.	B	(Fact. 7.)
9.	$\neg C$	(Res. 8. into 3.)
10.	$\square$	(Res. 4. into 9.)

Soundness of Propositional Resolution

Proposition

Propositional resolution is sound

Proof:

Let I be an interpretation. To be shown:

1. for resolution: $I \models C \vee A,\ I \models D \vee \neg A \ \Rightarrow\ I \models C \vee D$
2. for factoring: $I \models C \vee A \vee A \ \Rightarrow\ I \models C \vee A$

Soundness of Propositional Resolution

Proposition

Propositional resolution is sound

Proof:

Let I be an interpretation. To be shown:

1. for resolution: $I \models C \vee A,\ I \models D \vee \neg A \ \Rightarrow\ I \models C \vee D$
2. for factoring: $I \models C \vee A \vee A \ \Rightarrow\ I \models C \vee A$

Ad (i): Assume premises are valid in I. Two cases need to be considered: (a) A is valid in I, or (b) $\neg A$ is valid in I.

Soundness of Propositional Resolution

Proposition

Propositional resolution is sound

Proof:

Let I be an interpretation. To be shown:

1. for resolution: $I \models C \vee A,\ I \models D \vee \neg A \ \Rightarrow\ I \models C \vee D$
2. for factoring: $I \models C \vee A \vee A \ \Rightarrow\ I \models C \vee A$

Ad (i): Assume premises are valid in I. Two cases need to be considered: (a) A is valid in I, or (b) $\neg A$ is valid in I.

a) $I \models A \Rightarrow I \models D \Rightarrow I \models C \vee D$

Soundness of Propositional Resolution

Proposition

Propositional resolution is sound

Proof:

Let I be an interpretation. To be shown:

1. for resolution: $I \models C \vee A,\ I \models D \vee \neg A \ \Rightarrow\ I \models C \vee D$
2. for factoring: $I \models C \vee A \vee A \ \Rightarrow\ I \models C \vee A$

Ad (i): Assume premises are valid in I. Two cases need to be considered: (a) A is valid in I, or (b) $\neg A$ is valid in I.

a) $I \models A \Rightarrow I \models D \Rightarrow I \models C \vee D$

b) $I \models \neg A \Rightarrow I \models C \Rightarrow I \models C \vee D$

Soundness of Propositional Resolution

Proposition

Propositional resolution is sound

Proof:

Let I be an interpretation. To be shown:

1. for resolution: $I \models C \vee A$, $I \models D \vee \neg A \;\Rightarrow\; I \models C \vee D$
2. for factoring: $I \models C \vee A \vee A \;\Rightarrow\; I \models C \vee A$

Ad (i): Assume premises are valid in I. Two cases need to be considered: (a) A is valid in I, or (b) $\neg A$ is valid in I.

a) $I \models A \Rightarrow I \models D \Rightarrow I \models C \vee D$

b) $I \models \neg A \Rightarrow I \models C \Rightarrow I \models C \vee D$

Ad (ii): even simpler

Completeness of Propositional Resolution

Theorem:

Propositional Resolution is refutationally complete

Completeness of Propositional Resolution

Theorem:

Propositional Resolution is refutationally complete

- That is, if a propositional clause set is unsatisfiable, then Resolution will derive the empty clause $\Box$ eventually

Completeness of Propositional Resolution

Theorem:

Propositional Resolution is refutationally complete

- That is, if a propositional clause set is unsatisfiable, then Resolution will derive the empty clause $\square$ eventually
- More precisely: If a clause set is unsatisfiable and closed under the application of the Resolution and Factoring inference rules, then it contains the empty clause $\square$

Completeness of Propositional Resolution

Theorem:

Propositional Resolution is refutationally complete

- That is, if a propositional clause set is unsatisfiable, then Resolution will derive the empty clause $\square$ eventually
- More precisely: If a clause set is unsatisfiable and closed under the application of the Resolution and Factoring inference rules, then it contains the empty clause $\square$
- Perhaps easiest proof: semantic tree proof technique (see blackboard)

Completeness of Propositional Resolution

Theorem:

Propositional Resolution is refutationally complete

- That is, if a propositional clause set is unsatisfiable, then Resolution will derive the empty clause $\Box$ eventually
- More precisely: If a clause set is unsatisfiable and closed under the application of the Resolution and Factoring inference rules, then it contains the empty clause $\Box$
- Perhaps easiest proof: semantic tree proof technique (see blackboard)
- This result can be considerably strengthened, some strengthenings come for free from the proof

Completeness of Propositional Resolution

Theorem:

Propositional Resolution is refutationally complete

- That is, if a propositional clause set is unsatisfiable, then Resolution will derive the empty clause $\square$ eventually
- More precisely: If a clause set is unsatisfiable and closed under the application of the Resolution and Factoring inference rules, then it contains the empty clause $\square$
- Perhaps easiest proof: semantic tree proof technique (see blackboard)
- This result can be considerably strengthened, some strengthenings come for free from the proof

Propositional resolution is not suitable for first-order clause sets

First-Order Resolution

Propositional resolution:

- refutationally complete,
- in its most naive version: not guaranteed to terminate for satisfiable sets of clauses, (improved versions do terminate, however)
- in practice clearly inferior to the DPLL procedure (even with various improvements).

But: in contrast to the DPLL procedure, resolution can be easily extended to non-ground clauses (but see below First-order DPLL)

First-Order Resolution through Instantiation

Idea: instantiate clauses appropriately:

$$
\begin{tikzcd}
P(z',z') \vee \neg Q(z) \arrow[d, swap, "{[a/z', f(a,b)/z]}"] & & & \neg P(a,y) \arrow[dl, swap, "{[a/y]}"] \arrow[dr, "{[b/y]}"] & & & P(x',b) \vee Q(f(x',x)) \arrow[d, "{[a/x', b/x]}"] \\
P(a,a) \vee \neg Q(f(a,b)) \arrow[dr] & & \neg P(a,a) \arrow[dl] & & \neg P(a,b) \arrow[dr] & & P(a,b) \vee Q(f(a,b)) \arrow[dl] \\
& \neg Q(f(a,b)) \arrow[d] & & & & Q(f(a,b)) \arrow[d] & \\
& {} \arrow[drr] & & & & {} \arrow[dll] & \\
& & & \perp & & &
\end{tikzcd}
$$

First-Order Resolution through Instantiation

Problems:

- More than one instance of a clause can participate in a proof.
- Even worse: There are infinitely many possible instances.

Observation:

- Instantiation must produce complementary literals (so that inferences become possible).

Idea:

- Do not instantiate more than necessary to get complementary literals.

First-Order Resolution through Instantiation

Idea: do not instantiate more than necessary:

$$
\begin{tikzcd}[column sep=small]
P(z',z') \vee \neg Q(z) \arrow[d, "{[a/z']}"'] & & & \neg P(a,y) \arrow[dl, "{[a/y]}"'] \arrow[dr, "{[b/y]}"] & & & P(x',b) \vee Q(f(x',x)) \arrow[d, "{[a/x']}"] \\
P(a,a) \vee \neg Q(z) \arrow[dr] & & \neg P(a,a) \arrow[dl] & & \neg P(a,b) \arrow[dr] & & P(a,b) \vee Q(f(a,x)) \arrow[dl] \\
& \neg Q(z) \arrow[d, "{[f(a,x)/z]}"'] & & & & Q(f(a,x)) \arrow[d] & \\
& \neg Q(f(a,x)) \arrow[drr] & & & & Q(f(a,x)) \arrow[dll] & \\
& & & \bot & & &
\end{tikzcd}
$$

Lifting Principle

Problem: Make saturation of infinite sets of clauses as they arise from taking the (ground) instances of finitely many first-order clauses (with variables) effective and efficient.

Idea (Robinson 1965):

- Resolution for first-order clauses:
- Equality of ground atoms is generalized to unifiability of first-order atoms;
- Only compute most general (minimal) unifiers.

First-Order Resolution through Instantiation

Significance: The advantage of the method in (Robinson 1965) compared with (Gilmore 1960) is that unification enumerates only those instances of clauses that participate in an inference. Moreover, clauses are not right away instantiated into ground clauses. Rather they are instantiated only as far as required for an inference. Inferences with non-ground clauses in general represent infinite sets of ground inferences which are computed simultaneously in a single step.

Substitutions and Unifiers

- A substitution σ is a mapping from variables to terms which is the identity almost everywhere.

  Example: $\sigma = [y \mapsto f(x),\ z \mapsto f(x)]$

Substitutions and Unifiers

- A substitution σ is a mapping from variables to terms which is the identity almost everywhere.

  Example: $\sigma = [y \mapsto f(x),\ z \mapsto f(x)]$

- A substitution σ is applied to a term or atom t by replacing every occurrence of every variable x in t by $\sigma(x)$.
  Instead of $\sigma(t)$ one usually writes $t\sigma$

  Example, with σ is from above: $P(f(x), y)\sigma = P(f(x), f(x))$

Substitutions and Unifiers

- A substitution γ is a unifier of s and t iff $s\gamma = t\gamma$.

  A unifier σ is most general iff for every unifier γ of the same terms there is a substitution δ such that $\gamma = \delta \circ \sigma$ $(=: \sigma\delta)$. Notation: $\sigma = \mathrm{mgu}(s, t)$

  Example:

  $s = car(red, y, z)$

  $t = car(u, v, ferrari)$

Substitutions and Unifiers

- A substitution γ is a unifier of s and t iff $s\gamma = t\gamma$.

  A unifier σ is most general iff for every unifier γ of the same terms there is a substitution δ such that $\gamma = \delta \circ \sigma$ $(=: \sigma\delta)$. Notation: $\sigma = \mathrm{mgu}(s, t)$

  Example:

  $s = car(red, y, z)$

  $t = car(u, v, ferrari)$

  Then

  $\gamma = [u \mapsto red,\ y \mapsto fast,\ v \mapsto fast,\ z \mapsto ferrari]$ is a unifier,

Substitutions and Unifiers

- A substitution γ is a unifier of s and t iff $s\gamma = t\gamma$.

  A unifier σ is most general iff for every unifier γ of the same terms there is a substitution δ such that $\gamma = \delta \circ \sigma$ $(=: \sigma\delta)$. Notation: $\sigma = \mathrm{mgu}(s, t)$

  Example:

  $s = \mathit{car}(\mathit{red}, y, z)$

  $t = \mathit{car}(u, v, \mathit{ferrari})$

  Then

  $\gamma = [u \mapsto \mathit{red},\ y \mapsto \mathit{fast},\ v \mapsto \mathit{fast},\ z \mapsto \mathit{ferrari}]$ is a unifier,

  and

  $\sigma = [u \mapsto \mathit{red},\ y \mapsto v,\ z \mapsto \mathit{ferrari}]$ is a mgu for s and t.

  With $\delta = [v \mapsto \mathit{fast}]$ obtain $\sigma\delta = \gamma$.

Unification of Many Terms

Let $E = \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$ be a multiset of equations, where s_i and t_i are terms or atoms. The set E is called a unification problem.

A substitution σ is called a unifier of E if $s_i\sigma = t_i\sigma$ for all $1 \leq i \leq n$.

If a unifier of E exists, then E is called unifiable.

The rule system on the next slide computes a most general unifer of a unification problems or "fail" ($\perp$) if none exists.

Rule Based Naive Standard Unification

Starting with a given unification problem E, apply the following rules as long as possible. The notation "$s \doteq t,\ E$" means "$\{s \doteq t\} \cup E$".

$$t \doteq t, E \Rightarrow E \tag{Trivial}$$

$$f(s_1, \ldots, s_n) \doteq f(t_1, \ldots, t_n), E \Rightarrow s_1 \doteq t_1, \ldots, s_n \doteq t_n, E \tag{Decompose}$$

$$f(\ldots) \doteq g(\ldots), E \Rightarrow \bot \tag{Clash}$$

$$x \doteq t, E \Rightarrow x \doteq t, E\{x \mapsto t\} \tag{Apply}$$

$$\text{if } x \in var(E),\ x \notin var(t)$$

$$x \doteq t, E \Rightarrow \bot \tag{Occur Check}$$

$$\text{if } x \neq t,\ x \in var(t)$$

$$t \doteq x, E \Rightarrow x \doteq t, E \tag{Orient}$$

$$\text{if } t \text{ is not a variable}$$

Example 1

Let $E_1 = \{f(x, g(x), z) \doteq f(x, y, y)\}$ the unification problem to be solved.
In each step, the selected equation is <u>underlined</u>.

$$E_1 : \quad \underline{f(x, g(x), z) \doteq f(x, y, y)} \qquad \text{(given)}$$

Example 1

Let $E_1 = \{f(x, g(x), z) \doteq f(x, y, y)\}$ the unification problem to be solved.
In each step, the selected equation is underlined.

$$E_1 : \quad \underline{f(x, g(x), z) \doteq f(x, y, y)} \qquad \text{(given)}$$

$$E_2 : \quad \underline{x \doteq x}, \ g(x) \doteq y, \ z \doteq y \qquad \text{(by Decompose)}$$

Example 1

Let $E_1 = \{f(x, g(x), z) \doteq f(x, y, y)\}$ the unification problem to be solved.
In each step, the selected equation is underlined.

$$
\begin{array}{lll}
E_1 : & \underline{f(x, g(x), z) \doteq f(x, y, y)} & \text{(given)} \\
E_2 : & \underline{x \doteq x},\ g(x) \doteq y,\ z \doteq y & \text{(by Decompose)} \\
E_3 : & \underline{g(x) \doteq y},\ z \doteq y & \text{(by Trivial)}
\end{array}
$$

Example 1

Let $E_1 = \{f(x, g(x), z) \doteq f(x, y, y)\}$ the unification problem to be solved. In each step, the selected equation is <u>underlined</u>.

$$
\begin{array}{lll}
E_1 : & \underline{f(x, g(x), z) \doteq f(x, y, y)} & \text{(given)} \\[1ex]
E_2 : & \underline{x \doteq x},\ \ g(x) \doteq y,\ \ z \doteq y & \text{(by Decompose)} \\[1ex]
E_3 : & \underline{g(x) \doteq y},\ \ z \doteq y & \text{(by Trivial)} \\[1ex]
E_4 : & \underline{y \doteq g(x)},\ \ z \doteq y & \text{(by Orient)}
\end{array}
$$

Example 1

Let $E_1 = \{f(x, g(x), z) \doteq f(x, y, y)\}$ the unification problem to be solved. In each step, the selected equation is underlined.

$$
\begin{array}{lll}
E_1: & \underline{f(x, g(x), z) \doteq f(x, y, y)} & \text{(given)} \\
E_2: & \underline{x \doteq x},\ g(x) \doteq y,\ z \doteq y & \text{(by Decompose)} \\
E_3: & \underline{g(x) \doteq y},\ z \doteq y & \text{(by Trivial)} \\
E_4: & \underline{y \doteq g(x)},\ z \doteq y & \text{(by Orient)} \\
E_5: & y \doteq g(x),\ z \doteq g(x) & \text{(by Apply } \{y \mapsto g(x)\}\text{)}
\end{array}
$$

Example 1

Let $E_1 = \{f(x, g(x), z) \doteq f(x, y, y)\}$ the unification problem to be solved.
In each step, the selected equation is underlined.

$$
\begin{array}{lll}
E_1: & \underline{f(x, g(x), z) \doteq f(x, y, y)} & \text{(given)} \\
E_2: & \underline{x \doteq x},\ g(x) \doteq y,\ z \doteq y & \text{(by Decompose)} \\
E_3: & \underline{g(x) \doteq y},\ z \doteq y & \text{(by Trivial)} \\
E_4: & \underline{y \doteq g(x)},\ z \doteq y & \text{(by Orient)} \\
E_5: & y \doteq g(x),\ z \doteq g(x) & \text{(by Apply } \{y \mapsto g(x)\}\text{)}
\end{array}
$$

Result is mgu $\sigma = \{y \mapsto g(x),\ z \mapsto g(x)\}$.

Example 2

Let $E_1 = \{f(x, g(x)) \doteq f(x, x)\}$ the unification problem to be solved.
In each step, the selected equation is underlined.

$$E_1 : \quad \underline{f(x, g(x)) \doteq f(x, x)} \qquad \text{(given)}$$

$$\perp$$

Example 2

Let $E_1 = \{f(x, g(x)) \doteq f(x, x)\}$ the unification problem to be solved.
In each step, the selected equation is underlined.

$$
\begin{array}{lll}
E_1 : & \underline{f(x, g(x)) \doteq f(x, x)} & \text{(given)} \\
E_2 : & \underline{x \doteq x},\ g(x) \doteq x & \text{(by Decompose)} \\
 & & \\
 & \perp &
\end{array}
$$

Example 2

Let $E_1 = \{f(x, g(x)) \doteq f(x, x)\}$ the unification problem to be solved.
In each step, the selected equation is underlined.

$$
\begin{array}{lll}
E_1 : & \underline{f(x, g(x)) \doteq f(x, x)} & \text{(given)} \\
E_2 : & \underline{x \doteq x},\ g(x) \doteq x & \text{(by Decompose)} \\
E_3 : & \underline{g(x) \doteq x} & \text{(by Trivial)} \\
 & & \\
 & \perp &
\end{array}
$$

Example 2

Let $E_1 = \{f(x, g(x)) \doteq f(x, x)\}$ the unification problem to be solved.
In each step, the selected equation is underlined.

$$
\begin{array}{lll}
E_1: & \underline{f(x, g(x)) \doteq f(x, x)} & \text{(given)} \\
E_2: & \underline{x \doteq x},\ g(x) \doteq x & \text{(by Decompose)} \\
E_3: & \underline{g(x) \doteq x} & \text{(by Trivial)} \\
E_4: & \underline{x \doteq g(x)} & \text{(by Orient)} \\
 & \perp &
\end{array}
$$

Example 2

Let $E_1 = \{f(x, g(x)) \doteq f(x, x)\}$ the unification problem to be solved.
In each step, the selected equation is underlined.

$$
\begin{array}{lll}
E_1 : & \underline{f(x, g(x)) \doteq f(x, x)} & \text{(given)} \\
E_2 : & \underline{x \doteq x},\ g(x) \doteq x & \text{(by Decompose)} \\
E_3 : & \underline{g(x) \doteq x} & \text{(by Trivial)} \\
E_4 : & \underline{x \doteq g(x)} & \text{(by Orient)} \\
E_5 : & \perp & \text{(by Occur Check)}
\end{array}
$$

Example 2

Let $E_1 = \{f(x, g(x)) \doteq f(x, x)\}$ the unification problem to be solved.
In each step, the selected equation is <u>underlined</u>.

$$
\begin{array}{lll}
E_1: & \underline{f(x, g(x)) \doteq f(x, x)} & \text{(given)} \\
E_2: & \underline{x \doteq x},\ g(x) \doteq x & \text{(by Decompose)} \\
E_3: & \underline{g(x) \doteq x} & \text{(by Trivial)} \\
E_4: & \underline{x \doteq g(x)} & \text{(by Orient)} \\
E_5: & \perp & \text{(by Occur Check)}
\end{array}
$$

There is no unifier of E_1.

Main Properties

The above unification algorithm is sound and complete:
Given $E = s_1 \doteq t_1, \ldots, s_n \doteq t_n$, exhaustive application of the above rules always terminates, and one of the following holds:

- The result is a set equations in solved form, that is, is of the form

  $$x_1 \doteq u_1, \ldots, x_k \doteq u_k$$

  with x_i pairwise distinct variables, and $x_i \notin var(u_j)$.
  In this case, the solved form represents the substitution $\sigma_E = [x_1 \mapsto u_1, \ldots, x_k \mapsto u_k]$ and it is a mgu for E.
- The result is $\perp$. In this case no unifier for E exists.

First-Order Resolution Inference Rules

$$\frac{C \vee A \qquad D \vee \neg B}{(C \vee D)\sigma} \quad \text{if } \sigma = \mathrm{mgu}(A, B) \quad \text{[resolution]}$$

$$\frac{C \vee A \vee B}{(C \vee A)\sigma} \quad \text{if } \sigma = \mathrm{mgu}(A, B) \quad \text{[factoring]}$$

For the resolution inference rule, the premise clauses have to be renamed apart (made variable disjoint)

First-Order Resolution Inference Rules

$$\frac{C \vee A \qquad D \vee \neg B}{(C \vee D)\sigma} \quad \text{if } \sigma = \mathrm{mgu}(A, B) \quad \text{[resolution]}$$

$$\frac{C \vee A \vee B}{(C \vee A)\sigma} \quad \text{if } \sigma = \mathrm{mgu}(A, B) \quad \text{[factoring]}$$

For the resolution inference rule, the premise clauses have to be renamed apart (made variable disjoint)

Example

$$\frac{Q(z) \vee P(z, z) \qquad \neg P(x, y)}{Q(x)} \quad \text{where } \sigma = [z \mapsto x, y \mapsto x] \quad \text{[resolution]}$$

$$\frac{Q(z) \vee P(z, a) \vee P(a, y)}{Q(a) \vee P(a, a)} \quad \text{where } \sigma = [z \mapsto a, y \mapsto a] \quad \text{[factoring]}$$

Sample Refutation – The Barber Problem

```
set(binary_res). %% This is an "otter" input file
formula_list(sos).
%% Every barber shaves all persons who do not shave themselves:
all x (B(x) -> (all y (-S(y,y) -> S(x,y)))).
%% No barber shaves a person who shaves himself:
all x (B(x) -> (all y (S(y,y) -> -S(x,y)))).
%% Negation of "there are no barbers"
exists x B(x).
end_of_list.
```

otter finds the following refutation (clauses 1 – 3 are the CNF of the above):

```
1 [] -B(x)|S(y,y)|S(x,y).
2 [] -B(x)| -S(y,y)| -S(x,y).
3 [] B($c1).
4 [binary,1.1,3.1] S(x,x)|S($c1,x).
5 [factor,4.1.2] S($c1,$c1).
6 [binary,2.1,3.1] -S(x,x)| -S($c1,x).
10 [factor,6.1.2] -S($c1,$c1).
11 [binary,10.1,5.1] $F.
```

Completeness of First-Order Resolution

Theorem: Resolution is refutationally complete

Completeness of First-Order Resolution

Theorem: Resolution is refutationally complete

- That is, if a clause set is unsatisfiable, then Resolution will derive the empty clause $\square$ eventually

Completeness of First-Order Resolution

Theorem: Resolution is refutationally complete

- That is, if a clause set is unsatisfiable, then Resolution will derive the empty clause $\Box$ eventually
- More precisely: If a clause set is unsatisfiable and closed under the application of the Resolution and Factoring inference rules, then it contains the empty clause $\Box$

Completeness of First-Order Resolution

Theorem: Resolution is refutationally complete

- That is, if a clause set is unsatisfiable, then Resolution will derive the empty clause $\Box$ eventually
- More precisely: If a clause set is unsatisfiable and closed under the application of the Resolution and Factoring inference rules, then it contains the empty clause $\Box$
- Perhaps easiest proof: Herbrand Theorem + Completeness of propositional resolution + Lifting Lemma

Lifting Lemma

Lemma 0.1 *Let C and D be variable-disjoint clauses. If*

$$\begin{array}{ccc} D & & C \\ \big\downarrow \sigma & & \big\downarrow \rho \\ \end{array}$$

$$\frac{D\sigma \qquad\qquad C\rho}{C'} \qquad \textit{[propositional resolution]}$$

then there exists a substitution τ such that

$$\frac{D \qquad\qquad C}{C''} \qquad \textit{[first-order resolution]}$$

$$\begin{array}{c} \big\downarrow \tau \\ C' = C''\tau \end{array}$$

Lifting Lemma

An analogous lifting lemma holds for factoring.

Corollary: if N is a set of clauses closed under resolution and factoring, then also the set of all ground instances of all clauses from N is closed under resolution and factoring.

With this result, it only remains to be shown how a given set of clauses can be closed under resolution and factoring. For this use, e.g., the "Given Clause Loop".

The "Given Clause Loop"

As used in the Otter theorem prover:

Lists of clauses maintained by the algorithm: `usable` and `sos`.

Initialize `sos` with the input clauses, `usable` empty.

Algorithm (straight from the Otter manual):

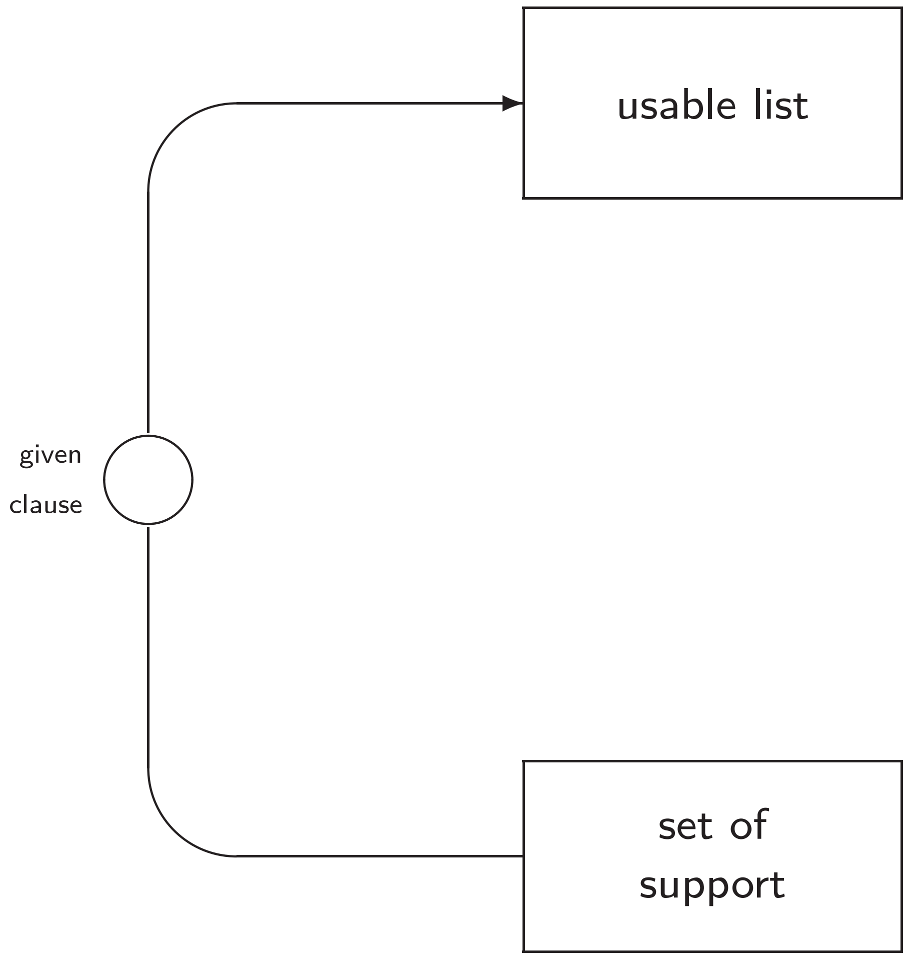
```
While (sos is not empty and no refutation has been found)
  1. Let given_clause be the 'lightest' clause in sos;
  2. Move given_clause from sos to usable;
  3. Infer and process new clauses using the inference rules in
     effect; each new clause must have the given_clause as
     one of its parents and members of usable as its other
     parents;  new clauses that pass the retention tests
     are appended to sos;
End of while loop.
```

Fairness: define clause weight e.g. as "depth + length" of clause.

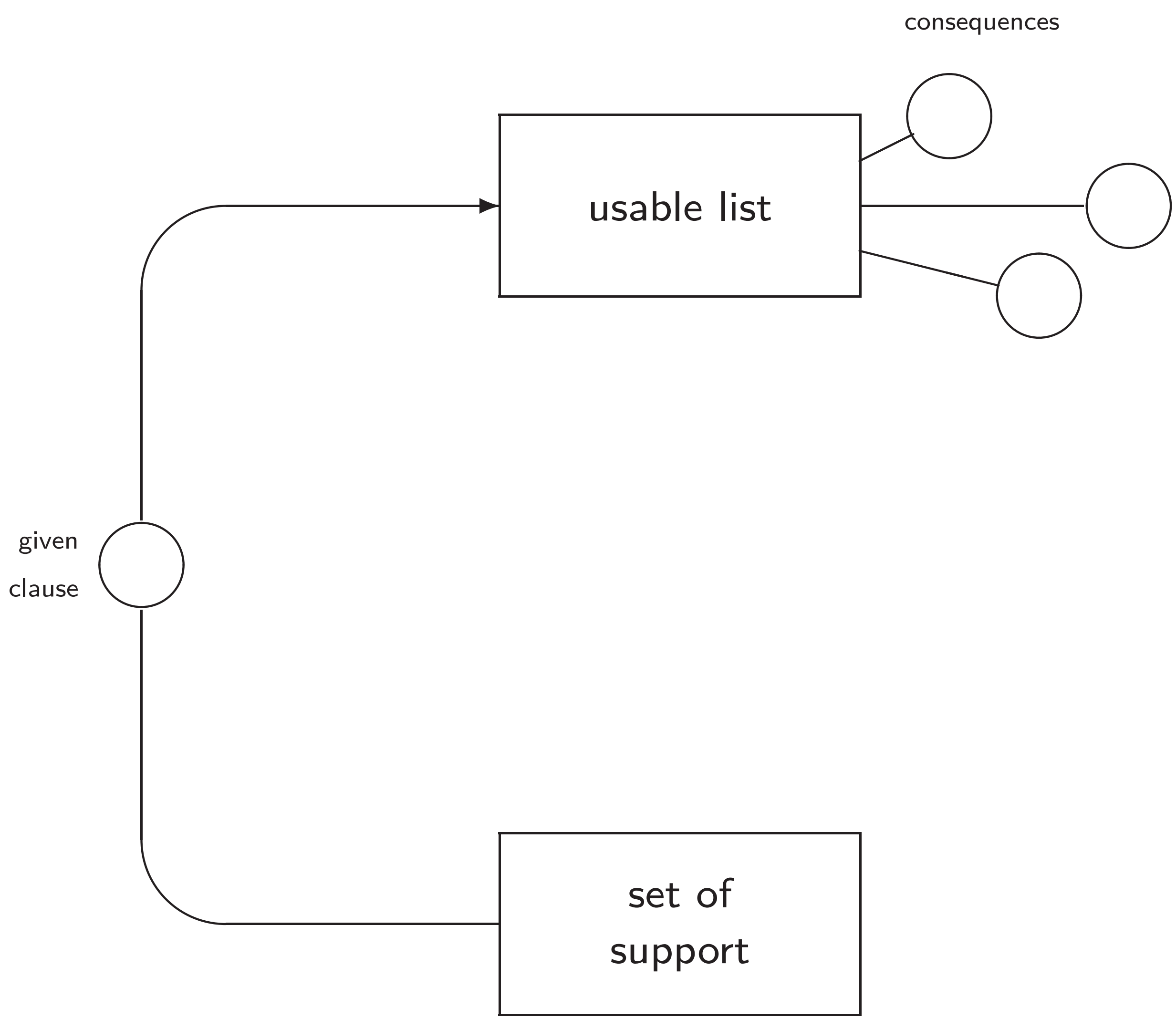
The “Given Clause Loop” - Graphically

The “Given Clause Loop” - Graphically

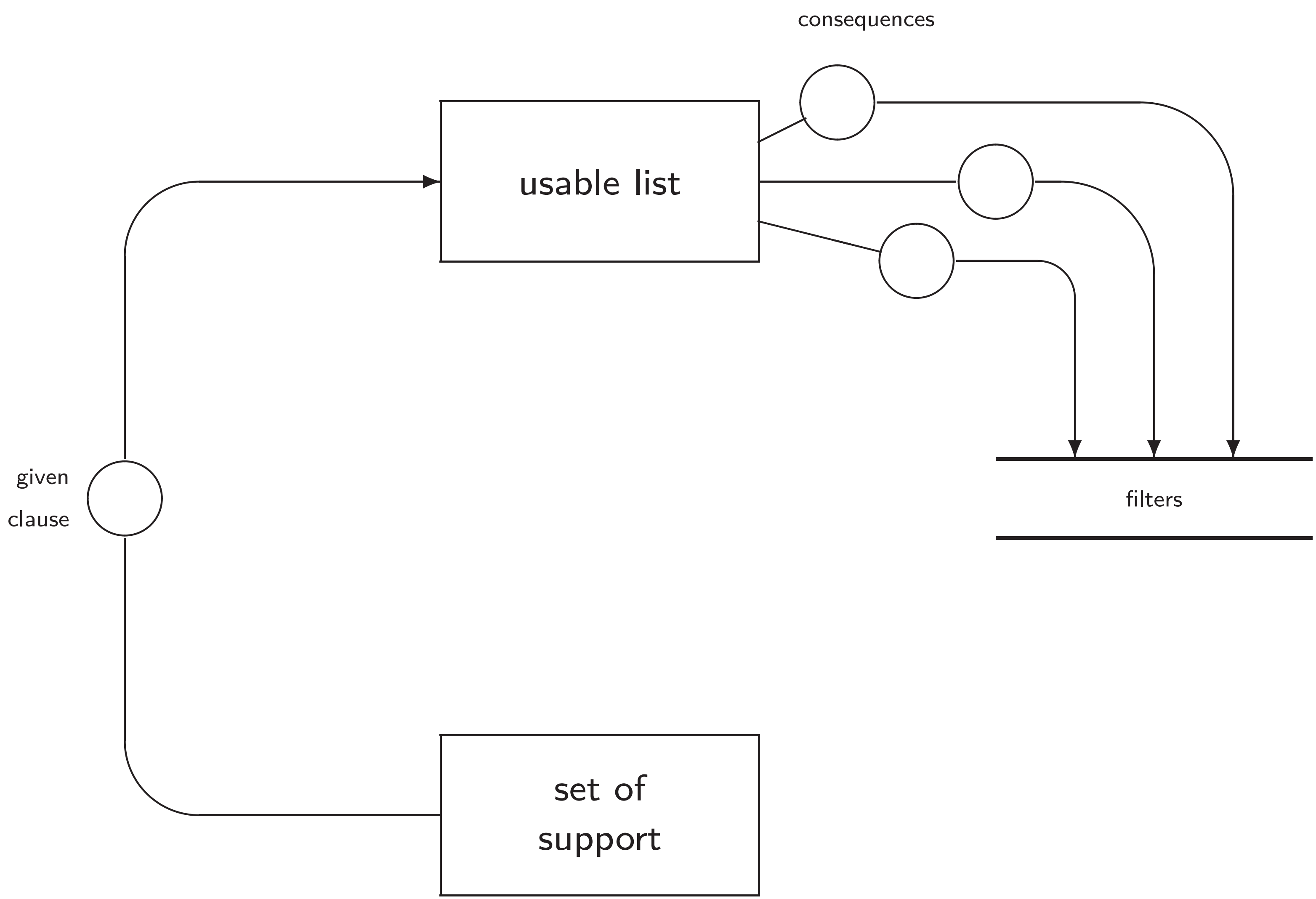
The “Given Clause Loop” - Graphically



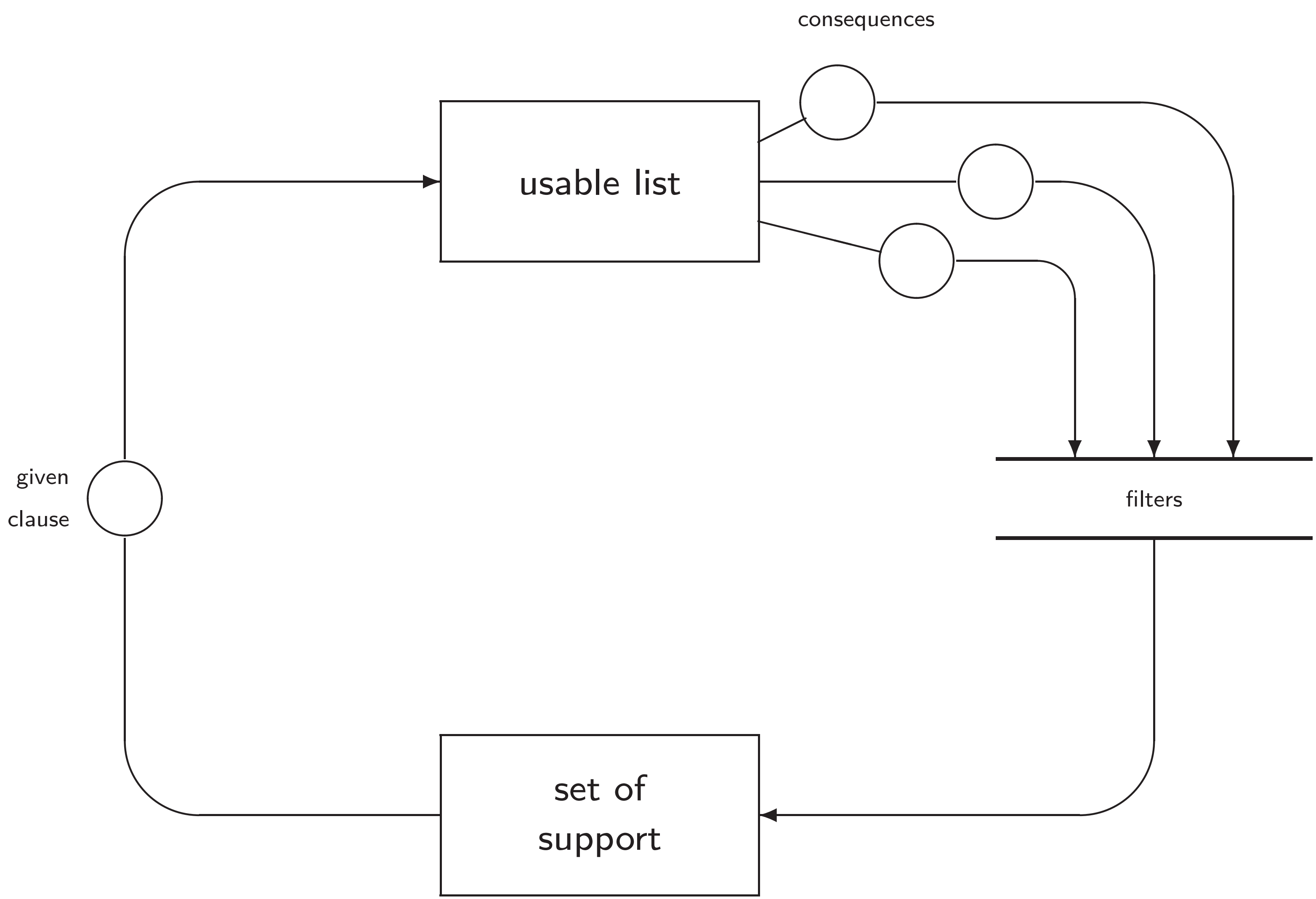
The "Given Clause Loop" - Graphically



The “Given Clause Loop” - Graphically



The "Given Clause Loop" - Graphically



Resolution – Further Topics

Overcoming the search space

- Restricting inference rules, in particular by ordering refinements.

  A-ordered resolution permits resolution inferences only if the literals resolved upon are maximal in their parent clauses.

- Resolution strategies, to compute (hopefully small) subsets of the full closure under inference rule applications.

  Set-of-support, Linear Resolution, Hyperresolution (see below), and more.

- Deleting clauses that are not needed to find a refutation.
  In particular subsumption deletion: delete clause C in presence of a (different) clause D such that $D\sigma \subseteq C$, for some substitution σ.

- Simplification of clauses.

Implementation techniques: in particular term indexing techniques

Hyperresolution

There are many variants of resolution. (We refer to [Bachmair, Ganzinger: Resolution Theorem Proving] for further reading.)

One well-known example is hyperresolution (Robinson 1965):

$$\frac{D_1 \vee B_1 \quad \ldots \quad D_n \vee B_n \qquad C \vee \neg A_1 \vee \ldots \vee \neg A_n}{(D_1 \vee \ldots \vee D_n \vee C)\sigma}$$

with $\sigma = \mathrm{mgu}(A_1 \doteq B_1, \ldots, A_n \doteq B_n)$.

Similarly to resolution, hyperresolution has to be complemented by a factoring inference.

Contents

Introduction

Logics and Reasoning Service (already done)

Methods for Automated Theorem Proving

Overview of some widely used general methods

- Propositional SAT solving
- First-order logic and clause normal forms
- Proof Procedures Based on Herbrand's Theorem
- The Resolution calculus
- Model generation

Theory Reasoning

Methods to reason with specific background theories

- Paramodulation (Equality)
- Satisfiability Modulo Theories (SMT)
- Quantifier elimination for linear real arithmetic
- Combining multiple theories

Model Generation

For every FOL formula F exactly one of these three cases applies:

1. F is unsatisfiable

   (Complete) theorem prover will detect this eventually (in theory)

2. F is satisfiable with only infinite models

   Example: $nat(0)$ $\qquad$ $lt(x, succ(N)) \leftarrow nat(x)$

   $nat(succ(x)) \leftarrow nat(x)$ $\qquad$ $lt(x, z) \leftarrow lt(x, y) \wedge lt(y, z)$

   $\neg lt(x, x)$

   Sometimes resolution refinements help to detect such cases

3. F is satisfiable with a finite model

   A finite model-finder will detect this eventually (in theory)

The rest of this section is concerned with computing finite models.

Model Generation

Two main applications:

- To disprove a "false" theorem by means of a counterexample, i.e., a "countermodel"
- A model provides the expected answer, as in the n-queens puzzle

Some applications

Planning: Can be formalised as propositional satisfiability problem.
[Kautz& Selman, AAAI96; Dimopolous et al, ECP97]

Diagnosis: Minimal models of *abnormal* literals (circumscription). [Reiter, AI87]

Databases: View materialisation, View Updates, Integrity Constraints.

Nonmonotonic reasoning: Various semantics (GCWA, Well-founded, Perfect, Stable,...), all based on minimal models. [Inoue et al, CADE 92]

Software Verification: Counterexamples to conjectured theorems.

Theorem proving: Counterexamples to conjectured theorems.
Finite models of quasigroups, (MGTP/G). [Fujita et al, IJCAI 93]

Example - Discourse Representation

Natural Language Processing:

- Maintain models $\mathfrak{I}_1, \ldots, \mathfrak{I}_n$ as different readings of discourses:

$$\mathfrak{I}_i \models \textit{BG-Knowledge} \cup \textit{Discourse\_so\_far}$$

Example - Discourse Representation

Natural Language Processing:

- Maintain models $\mathcal{I}_1, \ldots, \mathcal{I}_n$ as different readings of discourses:

$$\mathcal{I}_i \models \mathit{BG\text{-}Knowledge} \cup \mathit{Discourse\_so\_far}$$

- Consistency checks ("Mia's husband loves Sally. She is not married.")

$$\begin{array}{ll} & \mathit{BG\text{-}Knowledge} \cup \mathit{Discourse\_so\_far} \not\models \neg \mathit{New\_utterance} \\ \mathit{iff} & \mathit{BG\text{-}Knowledge} \cup \mathit{Discourse\_so\_far} \cup \mathit{New\_utterance} \text{ is satisfiable} \end{array}$$

Example - Discourse Representation

Natural Language Processing:

- Maintain models $\mathcal{I}_1, \ldots, \mathcal{I}_n$ as different readings of discourses:

$$\mathcal{I}_i \models \textit{BG-Knowledge} \cup \textit{Discourse\_so\_far}$$

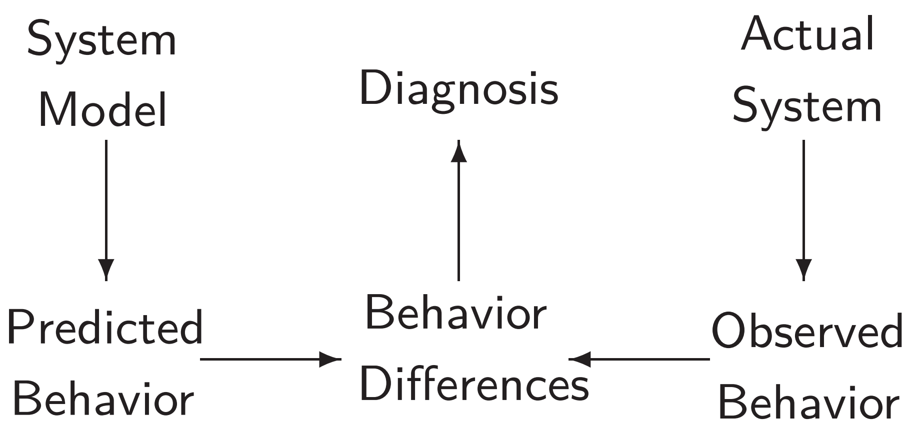
- Consistency checks ("Mia's husband loves Sally. She is not married.")

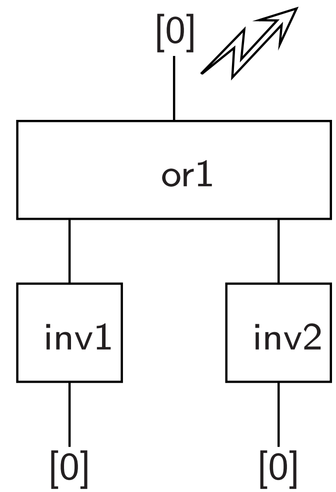
$$\begin{array}{ll} & \textit{BG-Knowledge} \cup \textit{Discourse\_so\_far} \not\models \neg \textit{New\_utterance} \\ \textit{iff} & \textit{BG-Knowledge} \cup \textit{Discourse\_so\_far} \cup \textit{New\_utterance} \text{ is satisfiable} \end{array}$$

- Informativity checks ("Mia's husband loves Sally. She is married.")

$$\begin{array}{ll} & \textit{BG-Knowledge} \cup \textit{Discourse\_so\_far} \not\models \textit{New\_utterance} \\ \textit{iff} & \textit{BG-Knowledge} \cup \textit{Discourse\_so\_far} \cup \neg \textit{New\_utterance} \text{ is satisfiable} \end{array}$$

Example - Model-Based Diagnosis [Reiter 87]





Formal Treatment:

$$COMP = \text{Components}$$

$$SD = \text{System description, components are allowed to perform “abnorma}$$

$$OBS = \text{Observations}$$

Def. **Diagnosis**: Some minimal $\Delta \subseteq COMP$ such that

$$SD \cup OBS \cup \{ab(\Delta)\} \cup \{\neg ab(COMP - \Delta)\} \text{ is satisfiable}$$

Formal Treatment

System Description $SD =$

$$\begin{array}{llll}
\text{OR1:} & \neg(ab(or1)) & \rightarrow & high(or1, o) \leftrightarrow (high(or1, i1) \vee high(or1, i2)) \\
\text{INV1:} & \neg(ab(inv1)) & \rightarrow & high(inv1, o) \leftrightarrow \neg(high(inv1, i)) \\
\text{INV2:} & \neg(ab(inv2)) & \rightarrow & high(inv2, o) \leftrightarrow \neg(high(inv2, i)) \\
\text{CONN1:} & & & high(inv1, o) \leftrightarrow high(or1, i1) \\
\text{CONN2:} & & & high(inv2, o) \leftrightarrow high(or1, i2)
\end{array}$$

Observations $OBS =$

$$\begin{array}{ll}
\text{LOW_INV1_I:} & \neg(high(inv1, i)) \\
\text{LOW_INV1_I:} & \neg(high(inv2, i)) \\
\text{LOW_OR1_O:} & \neg(high(or1, o))
\end{array}$$

Task: Find minimal $\Delta \subseteq \{ab(or1),\ ab(inv1),\ ab(inv2)\}$ such that

$SD \cup OBS \cup \Delta \cup \neg\overline{\Delta}$ is satisfiable

Formal Treatment

System Description $SD =$

OR1:	$\neg(ab(or1))$	$\rightarrow$	$high(or1,o) \leftrightarrow (high(or1,i1) \vee high(or1,i2))$
INV1:	$\neg(ab(inv1))$	$\rightarrow$	$high(inv1,o) \leftrightarrow \neg(high(inv1,i))$
INV2:	$\neg(ab(inv2))$	$\rightarrow$	$high(inv2,o) \leftrightarrow \neg(high(inv2,i))$
CONN1:			$high(inv1,o) \leftrightarrow high(or1,i1)$
CONN2:			$high(inv2,o) \leftrightarrow high(or1,i2)$

Observations $OBS =$

LOW_INV1_I:	$\neg(high(inv1,i))$
LOW_INV1_I:	$\neg(high(inv2,i))$
LOW_OR1_O:	$\neg(high(or1,o))$

Task: Find minimal $\Delta \subseteq \{ab(or1),\ ab(inv1),\ ab(inv2)\}$ such that

$$SD \cup OBS \cup \Delta \cup \neg\overline{\Delta} \text{ is satisfiable}$$

Solutions: (1) $\Delta_1 = \{ab(or1)\}$ and (2) $\Delta_2 = \{ab(inv1),\ ab(inv2)\}$

Example - Group Theory

The following axioms specify a group

$$
\begin{array}{llllll}
\forall x, y, z & : & (x * y) * z & = & x * (y * z) & \text{(associativity)} \\
\forall x & : & e * x & = & x & (\text{left} - \text{identity}) \\
\forall x & : & i(x) * x & = & e & (\text{left} - \text{inverse})
\end{array}
$$

Does

$$
\forall x, y \quad : \quad x * y \quad = \quad y * x \qquad \text{(commutat.)}
$$

follow?

Example - Group Theory

The following axioms specify a group

$$\begin{array}{lcrcll}
\forall x, y, z & : & (x * y) * z & = & x * (y * z) & \text{(associativity)} \\
\forall x & : & e * x & = & x & \text{(left - identity)} \\
\forall x & : & i(x) * x & = & e & \text{(left - inverse)}
\end{array}$$

Does

$$\forall x, y \quad : \quad x * y = y * x \qquad \text{(commutat.)}$$

follow?

No, it does not

Example - Group Theory

Counterexample: a group with finite domain of size 6, where the elements 2 and 3 are not commutative: Domain: $\{1, 2, 3, 4, 5, 6\}$

$e : 1$

$$i : \quad \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 2 & 3 & 5 & 4 & 6 \end{array}$$

$$* : \quad \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 2 & 1 & 4 & 3 & 6 & 5 \\ 3 & 3 & 5 & 1 & 6 & 2 & 4 \\ 4 & 4 & 6 & 2 & 5 & 1 & 3 \\ 5 & 5 & 3 & 6 & 1 & 4 & 2 \\ 6 & 6 & 4 & 5 & 2 & 3 & 1 \end{array}$$

Finite Model Finding

Def: A formula F has the finite model property iff F has a model with a finite domain. (The finite model property is undecidable.)

Question here: how to compute (“efficiently”) finite models?

Today’s finite model finders all follow a generate-and-test approach:

- Given a formula F in clause normal form.
- For each domain size $n = 1, 2, \ldots$ transform F into a clause set $G(F, n)$ such that $G(F, n)$ is satisfiable iff F is satisfiable with the domain $D = \{1, 2, \ldots, n\}$

  For each n, use a theorem prover to determine if $G(F, n)$ is satisfiable.

  If so, stop and report the model. Otherwise continue.

Group Theory Example – $G(F, n)$ as Reduction to SAT

Domain:	$\{1, 2\}$
Clauses:	$\{p(a) \vee f(x) = a\}$
Flattened:	$p(y) \vee f(x) = y \vee a \neq y$
Instances:	$p(1) \vee f(1) = 1 \vee a \neq 1$
	$p(2) \vee f(1) = 1 \vee a \neq 2$
	$p(1) \vee f(2) = 1 \vee a \neq 1$
	$p(2) \vee f(2) = 1 \vee a \neq 2$
Totality:	$a = 1 \vee a = 2$
	$f(1) = 1 \vee f(1) = 2$
	$f(2) = 1 \vee f(2) = 2$
Functionality:	$a \neq 1 \vee a \neq 2$
	$f(1) \neq 1 \vee f(1) \neq 2$
	$f(2) \neq 1 \vee f(2) \neq 2$

A model is obtained by setting the blue literals true

Contents

Introduction

Logics and Reasoning Service (already done)

Methods for Automated Theorem Proving

Overview of some widely used general methods

- Propositional SAT solving
- First-order logic and clause normal forms
- Proof Procedures Based on Herbrand's Theorem
- The Resolution calculus
- Model generation

Theory Reasoning

Methods to reason with specific background theories

- Paramodulation (Equality)
- Satisfiability Modulo Theories (SMT)
- Quantifier elimination for linear real arithmetic
- Combining multiple theories

Theory Reasoning

Let T be a first-order theory of signature Σ and L be a class of Σ-formulas.

- T can be given as a set of axioms (e.g., the theory of groups), or
- T can be given as a class of interpretations (e.g., the standard model of peano arithmetic)

The T-validity Problem

- Given ϕ in L, is it the case that $T \models \phi$? More accurately:
- Given ϕ in L, is it the case that $T \models \forall\, \phi$?

Examples

- "$0/0,\ s/1,\ +/2,\ =/2,\ \leq/2$" $\models \exists y . y > x$
- The theory of equality E $\models \phi$ (ϕ arbitrary formula)
- "An equational theory" $\models \exists\, s_1 = t_1 \wedge \cdots \wedge s_n = t_n$ (E-Unification problem)
- "Some group theory" $\models s = t$ (Word problem)

The T-validity problem is decidable (even semi-decidable) only for restricted L and T

Approaches to Theory Reasoning

Question: Does $T \models \phi$ hold?

Question: Does $Ax \models_T Th$ hold? (Same question, take $\phi = Ax \rightarrow Th$)

Theory-Reasoning in Automated First-Order Theorem Proving

- ϕ is a first-order formula and T is for sub-signature only.
- In general not even semi-decidable.
- Semi-decidable, e.g., for T = equality, using inference rules like paramodulation (see below).

Satisfiability Modulo Theories (SMT)

- ϕ is quantifier-free, i.e. all variables implicitly universally quantified.
- Decidable for many useful theories.
- Applications in particular to formal verification.
  Simple example where T = "arrays+integers":
  $\{m \geq 0 \land a[i] \geq 0\} \quad a[i] := a[i] + m \quad \{a[i] \geq 0\}$

Theory Reasoning – Program Verification Example

```
declare-datatype Tree =
      empty
    | node of val: Int, left: Tree, right: Tree

@pre: searchtree(t)
@post: binSearch(t, v) <-> in(v, t)
def binSearch(t: Tree, v: Int) =
  if (t = empty)
     false
  else {
     if (v = val(t))
        true
     else if (v < val(t))
        binSearch(left(t), v)
     else
        binSearch(right(t), v)
  }
```

Theory Reasoning – Program Verification Example

Partial correctness:

Assume precondition `searchtree(t)`

Proof of postcondition `binSearch(t, v) <-> in(v, t)` by induction:

$Th1 =$

```
forall t:Tree, v:Int
  searchtree(t) ->
  let res =
      if t = empty then
         false
      else if v = val(t) then
         true
      else if v < val(t) then
         in(v, left(t)) // by I.H.
       else
         in(v, right(t)) // by I.H.
   in res <-> in(v, t)
```

Theory Reasoning – Program Verification Example

Need to prove that precondition holds in induction case, so that I.H. can be applied:

$Th2 =$

```
forall t:Tree, v:Int
  searchtree(t) ->
  if t = empty then
     true
  else if v = val(t) then
     true
  else if v < val(t) then
     searchtree(left(t))
  else
     searchtree(right(t))
```

Theory Reasoning – Program Verification Example

To prove $Th1$ and $Th2$ We need to provide axioms for

- the `Tree` datatype,
- the `in`-predicate, and
- the `searchtree`-predicate.

Theory Reasoning – Program Verification Example

```
declare-datatype Tree =
      empty
    | node of val: Int, left: Tree, right: Tree
```

Axioms for `Tree`

```
%% Constructor axiom
forall t: Tree
    t = empty or
    t = node(val(t), left(t), right(t))

%% Injectivity of constructors
forall t1, t2: Tree, v: Int
    empty /= node(v, t1, t2)

%% Selector axioms for val (similarly for left and right)
forall t1, t2: Tree, v: Int
    val(node(v, t1, t2)) = v
```

Theory Reasoning – Program Verification Example

`in(v, t)` holds true iff `v` is the `value` of some node in `t`.

TreeMembershipAxiom =

```
forall: tTree, v:Int
  in(v, t) <->
  if t = empty then
     false
  else if v = val(t) then
     true
  else if in(v, left(t)) then
     true
  else
     in(v, right(t))
```

Theory Reasoning – Program Verification Example

`searchtree(t)` holds true iff `t` is a search tree.

SearchTreeAxiom $=$

```
forall: tTree
  searchtree(t) <->
  if t = empty then
     true
  else
     (forall v: Int
         (if in(v, left(t)) then v =< val(t)) and
         (if in(v, right(t)) then v > val(t))) and
     searchtree(left(t)) and
     searchtree(right(t))
```

Theory Reasoning – Program Verification Example

The proof obligations in full:

1. $\mathit{TreeAxioms} \cup \mathit{SearchTreeAxiom} \cup \mathit{TreeMembershipAxiom} \models_T \mathit{Th}1$
2. $\mathit{TreeAxioms} \cup \mathit{SearchTreeAxiom} \cup \mathit{TreeMembershipAxiom} \models_T \mathit{Th}2$

over first-order logic with equality where $T =$ linear integer arithmetic.

The free symbols (*searchtree*, ...) are not part of T, they are specified by the axioms on the left of $\models_T$.

For automatically proving 1 and 2 we need to extend the resolution calculus by equality reasoning and by reasoning modulo a theory T.

Equality

Reserve a binary predicate symbol $\approx$ ("equality").

Intuitively, we expect that from the clauses

$$P(a) \qquad a \approx b \qquad b \approx c \qquad f(x) \approx x \qquad f(x) \approx g(x)$$

it follows, e.g.,

$$P(g(f(c)))$$

This requires to fix the meaning of $\approx$. Two options:

- Semantically: define $\approx \;=\; \{(d, d) \mid d \in U\}$
  (Recall that predicate symbols are interpreted as relations, U is the universe)
- Syntactically: add equality axioms to the given clause set

The semantic approach cannot be used in conjunction with Herbrand models, but the syntactic approach can.

Handling Equality Naively - Equality Axioms

Let F be a first-order clause set with equality. The clause set $EqAx(F)$ consists of the clauses

$$x \approx x$$

$$x \approx y \rightarrow y \approx x$$

$$x \approx y \wedge y \approx z \rightarrow x \approx z$$

$$x_1 \approx y_1 \wedge \cdots \wedge x_n \approx y_n \rightarrow f(x_1, \ldots, x_n) \approx f(y_1, \ldots, y_n)$$

$$x_1 \approx y_1 \wedge \cdots \wedge x_m \approx y_m \wedge P(x_1, \ldots, x_m) \rightarrow P(y_1, \ldots, y_m)$$

for every n-ary function symbol f occurring in F and every m-ary predicate symbol P occurring in F.

$EqAx(F)$ are the axioms of a congruence relation on terms and atoms.

It holds: F is satisfiable, where $\approx$ is defined semantically as in the previous slide, if and only if $F \cup EqAx(\Sigma)$ is satisfiable, where $\approx$ is left undefined.

Handling Equality Naively - Equality Axioms

By giving the equality axioms explicitly, first-order problems with equality can in principle be solved by a standard resolution prover or instance-based method.

But this is unfortunately not efficient (mainly due to the transitivity and congruence axioms).

Modern systems “build-in” equality by dedicated inference rules, which are (restricted) versions of the Paramodulation inference rule.

Recapitulation: Resolution

Resolution: inference rules:

	Ground case:	Non-ground case:
Resolution:	$\dfrac{D' \lor A \qquad C' \lor \neg A}{D' \lor C'}$	$\dfrac{D' \lor A \qquad C' \lor \neg A'}{(D' \lor C')\sigma}$ where $\sigma = \text{mgu}(A, A')$.
Factoring:	$\dfrac{C' \lor A \lor A}{C' \lor A}$	$\dfrac{C' \lor A \lor A'}{(C' \lor A)\sigma}$ where $\sigma = \text{mgu}(A, A')$.

Paramodulation

Ground inference rules:

Paramodulation: $$\frac{D' \vee t \approx t' \qquad C' \vee L[t]}{D' \vee C' \vee L[t']}$$

Equality Resolution: $$\frac{C' \vee s \not\approx s}{C'}$$

In the Paramodulation rule, $L[t]$ means that the literal L contains the term t, and $L[t']$ means that one occurrence of t in L has been replaced by t'.

Paramodulation

First-order inference rules:

Paramodulation:
$$\frac{D' \vee t \approx t' \qquad C' \vee L[u]}{(D' \vee C' \vee L[t'])\sigma}$$

where $\sigma = \mathrm{mgu}(t, u)$ and
u is not a variable.

Equality Resolution:
$$\frac{C' \vee s \not\approx s'}{C'\sigma}$$

where $\sigma = \mathrm{mgu}(s, s')$.

These are the main inference rules for equality reasoning. Together with the Resolution and Factoring inference rules, and an additional inference rule (not shown here), one obtains a refutationally complete and sound calculus.

The calculus can still be considerably improved by means of ordering restrictions.

Resolution Modulo a Theory (Main Idea)

Problem: unification cannot detect "semantical equality" of T-terms:

$$\frac{P(1+2) \qquad \neg P(2+1)}{?}$$

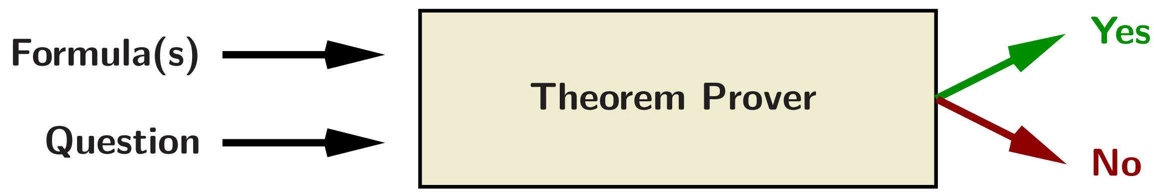
Solution: abstraction for extracting T-terms for separate check later:

$$\cfrac{P(x) \vee \neg(x = 1+2) \qquad \neg P(y) \vee \neg(y = 2+1)}{\cfrac{\neg(x = 1+2) \vee \neg(x = 2+1)}{\Box}\,(T\text{-Close})}$$

The premise of T-Close is a finite set of clauses N over the signature of T. T-Close derives the empty clause $\Box$ from N iff N is unsatisfiable (wrt. all models of T).

Compactness/completeness issue: $\mathbb{Z}$ is not compact: $\{1 < a, 2 < a, 3 < a, \ldots\}$ is unsatisfiable although every finite subset N is satisfiable.

Satisfiability Modulo Theories (SMT)



Formula: first-order logic formula ϕ, over equality and other theories

Question: Is ϕ valid? (satisfiable? entailed by another formula?)

$$\models_{\mathbb{N}\cup\mathbb{L}} \forall l\ (c = 5 \rightarrow \mathrm{car}(\mathrm{cons}(3 + c, l)) \doteq 8)$$

Theorem Prover: DPLL(T), translation into SAT, first-order provers

Issue: essentially undecidable for non-variable free fragment ($\forall$-quantifier left of $\models$):

$$P(0) \wedge (\forall x\ P(x) \rightarrow P(x+1)) \models_{\mathbb{N}} \forall x\ P(x)$$

Design a “good” prover anyways (ongoing research)

Checking Satisfiability Modulo Theories

Given: A quantifier-free formula ϕ (implicitly existentially quantified)
Task: Decide whether ϕ is T-satisfiable
(T-validity via "$T \models \forall\ \phi$" iff "$\exists\ \neg\phi$ is not T-satisfiable")

Approach: eager translation into SAT

- Encode problem into a T-equisatisfiable propositional formula
- Feed formula to a SAT-solver
- Example: $T =$ equality (Ackermann encoding)

Approach: lazy translation into SAT

- Couple a SAT solver with a given decision procedure for T-satisfiability of ground literals, "DPLL(T)"
- For instance if T is "equality" then the Nelson-Oppen congruence closure method can be used
- If T is "linear arithmetic", a quantifier elimination method (see below)

Lazy Translation into SAT

$$g(a) = c \quad \wedge \quad f(g(a)) \neq f(c) \vee g(a) = d \quad \wedge \quad c \neq d$$

Theory: Equality

Lazy Translation into SAT

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}}$$

Lazy Translation into SAT

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}}$$

- Send $\{1,\ \overline{2} \vee 3,\ \overline{4}\}$ to SAT solver.

Lazy Translation into SAT

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}}$$

- Send $\{1,\ \overline{2} \vee 3,\ \overline{4}\}$ to SAT solver.
- SAT solver returns model $\{1,\ \overline{2},\ \overline{4}\}$.
  Theory solver finds $\{1,\ \overline{2}\}$ E-unsatisfiable.

Lazy Translation into SAT

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}}$$

- Send $\{1,\ \overline{2} \vee 3,\ \overline{4}\}$ to SAT solver.
- SAT solver returns model $\{1,\ \overline{2},\ \overline{4}\}$.
  Theory solver finds $\{1,\ \overline{2}\}$ E-unsatisfiable.
- Send $\{1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2\}$ to SAT solver.

Lazy Translation into SAT

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}}$$

- Send $\{1,\ \overline{2} \vee 3,\ \overline{4}\}$ to SAT solver.
- SAT solver returns model $\{1,\ \overline{2},\ \overline{4}\}$.
  Theory solver finds $\{1,\ \overline{2}\}$ E-unsatisfiable.
- Send $\{1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2\}$ to SAT solver.
- SAT solver returns model $\{1,\ 2,\ 3,\ \overline{4}\}$.
  Theory solver finds $\{1,\ 3,\ \overline{4}\}$ E-unsatisfiable.

Lazy Translation into SAT

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}}$$

- Send $\{1,\ \overline{2} \vee 3,\ \overline{4}\}$ to SAT solver.
- SAT solver returns model $\{1,\ \overline{2},\ \overline{4}\}$.
  Theory solver finds $\{1,\ \overline{2}\}$ E-unsatisfiable.
- Send $\{1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2\}$ to SAT solver.
- SAT solver returns model $\{1,\ 2,\ 3,\ \overline{4}\}$.
  Theory solver finds $\{1,\ 3,\ \overline{4}\}$ E-unsatisfiable.
- Send $\{1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4\}$ to SAT solver.
  SAT solver finds $\{1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4\}$ unsatisfiable.

Lazy Translation into SAT: Summary

- Abstract T-atoms as propositional variables
- SAT solver computes a model, i.e. satisfying boolean assignment for propositional abstraction (or fails)
- Solution from SAT solver may not be a T-model. If so,
  - Refine (strengthen) propositional formula by incorporating reason for false solution
  - Start again with computing a model

Optimizations

Theory Consequences

- The theory solver may return consequences (typically literals) to guide the SAT solver

Online SAT solving

- The SAT solver continues its search after accepting additional clauses (rather than restarting from scratch)

Preprocessing atoms

- Atoms are rewritten into normal form, using theory-specific atoms (e.g. associativity, commutativity)

Several layers of decision procedures

- "Cheaper" ones are applied first

Example Theory: Linear Arithmetic

Linear Rational Arithmetic (LRA) is the interpretation

$$I_{\mathrm{LA}} = (\mathbb{Q},\ (+_{\mathcal{A}_{\mathrm{LA}}}, -_{\mathcal{A}_{\mathrm{LA}}}, *_{\mathcal{A}_{\mathrm{LA}}}),\ (\leq_{\mathcal{A}_{\mathrm{LA}}}, \geq_{\mathcal{A}_{\mathrm{LA}}}, <_{\mathcal{A}_{\mathrm{LA}}}, >_{\mathcal{A}_{\mathrm{LA}}}))$$

where $+_{\mathcal{A}_{\mathrm{LA}}}, -_{\mathcal{A}_{\mathrm{LA}}}, *_{\mathcal{A}_{\mathrm{LA}}}, \leq_{\mathcal{A}_{\mathrm{LA}}}, \geq_{\mathcal{A}_{\mathrm{LA}}}, <_{\mathcal{A}_{\mathrm{LA}}}, >_{\mathcal{A}_{\mathrm{LA}}}$ are the "standard" intepretations of $+, -, *, \leq, \geq, <, >$, respectively.

The Problem

Within the DPLL(T) framework it is enough to design a decision procedure for LRA-satisfiability of sets N (conjunctions) of literals. Note that (hence) all variables in N are implicitly existentially quantified

Example:
$N = \{2x \leq y,\ y < 6, 3 < y, 1 < x\}$

Question: Is there an assignment β for the variables x and y such that $(I_{\mathrm{LA}}, \beta) \models N$?

Some Important LA Equivalences

The following equivalences are valid for all LA terms s, t:

$$\neg s \geq t \leftrightarrow s < t$$

$$\neg s \leq t \leftrightarrow s > t \qquad \text{(Negation)}$$

$$(s = t) \leftrightarrow (s \leq t \wedge s \geq t) \qquad \text{(Equality)}$$

$$s \geq t \leftrightarrow t \leq s$$

$$s > t \leftrightarrow t < s \qquad \text{(Swap)}$$

With $\lesssim$ we abbreviate $<$ or $\leq$.

The Fourier-Motzkin Procedure

```
boolean FM(Set N of LA atoms) {
    if (N = ∅) return true;
    elsif (N is ground) return I_LA(N);
    else {
        select a variable x from N;
        transform all atoms in N containing x into s_i ≲ x, x ≲ t_j
        and the subset N′ of atoms not containing x;
        compute N* := {s_i ≲_{i,j} t_j | s_i ≲_i x ∈ N, x ≲_j t_j ∈ N for all i, j}
        where ≲_{i,j} is strict iff at least one of ≲_i, ≲_j is strict
        return FM(N′ ∪ N*);
    }
}
```

Properties of the Fourier-Motzkin Procedure

- Any ground set N of linear arithmetic atoms can be easily decided.
- $\mathrm{FM}(N)$ terminates on any N as in recursive calls N has strictly less variables.
- The set $N' \cup N^*$ is worst case of size $O(|N|^2)$.
- $\mathrm{FM}(N)$=true iff N is satisfiable in I_{LA}.
- The procedure was invented by Fourier (1826), forgotten, and then rediscovered by Dines (1919) and Motzkin (1936).
- There are more efficient methods known, e.g., the simplex algorithm.
- As said, the Fourier-Motzkin Procedure decides the satisfiability of a set (conjunction) of linear arithmetic atoms, which is what is needed to build a sound and complete DPLL(T)-solver.

Combining Theories

Theories:

- $\mathcal{R}$: theory of rationals
  $\Sigma_{\mathcal{R}} = \{\leq, +, -, 0, 1\}$
- $\mathcal{L}$: theory of lists
  $\Sigma_{\mathcal{L}} = \{=, \mathrm{hd}, \mathrm{tl}, \mathrm{nil}, \mathrm{cons}\}$
- $\mathcal{E}$: theory of equality
  Σ: free function and predicate symbols

Problem: Is

$$x \leq y \wedge y \leq x + \mathrm{hd}(\mathrm{cons}(0, \mathrm{nil})) \wedge P(h(x) - h(y)) \wedge \neg P(0)$$

satisfiable in $\mathcal{R} \cup \mathcal{L} \cup \mathcal{E}$?

Nelson-Oppen Combination Method

G. Nelson and D.C. Oppen: *Simplification by cooperating decision procedures*, ACM Trans. on Programming Languages and Systems, 1(2):245-257, 1979.

Given:

- $\mathcal{T}_1$, $\mathcal{T}_2$ first-order theories with signatures Σ_1, Σ_2
- $\Sigma_1 \cap \Sigma_2 = \emptyset$
- ϕ quantifier-free formula over $\Sigma_1 \cup \Sigma_2$

Obtain a decision procedure for satisfiability in $\mathcal{T}_1 \cup \mathcal{T}_2$ from decision procedures for satisfiability in $\mathcal{T}_1$ and $\mathcal{T}_2$.

Nelson-Oppen Combination Method

Variable abstraction + *equality propagation*:

$$x \leq y \wedge y \leq x + \mathrm{hd}(\mathrm{cons}(0, \mathrm{nil})) \wedge P(h(x) - h(y)) \wedge \neg P(0)$$

Nelson-Oppen Combination Method

Variable abstraction + *equality propagation*:

$$x \leq y \wedge y \leq x + \underbrace{\mathrm{hd}(\mathrm{cons}(0, \mathrm{nil}))}_{v_1} \wedge P(\underbrace{\underbrace{h(x)}_{v_3} - \underbrace{h(y)}_{v_4}}_{v_2}) \wedge \neg P(\underbrace{0}_{v_5})$$

Nelson-Oppen Combination Method

Variable abstraction + *equality propagation*:

$$x \leq y \wedge y \leq x + \underbrace{\mathrm{hd}(\mathrm{cons}(0, \mathrm{nil}))}_{v_1} \wedge P(\underbrace{\underbrace{h(x)}_{v_3} - \underbrace{h(y)}_{v_4}}_{v_2}) \wedge \neg P(\underbrace{0}_{v_5})$$

$\mathcal{R}$	$\mathcal{L}$	$\mathcal{E}$
$x \leq y$		$P(v_2)$
$y \leq x + v_1$		$\neg P(v_5)$

Nelson-Oppen Combination Method

Variable abstraction + *equality propagation*:

$$x \le y \land y \le x + \underbrace{\mathrm{hd}(\mathrm{cons}(0, \mathrm{nil}))}_{v_1} \land P(\underbrace{\underbrace{h(x)}_{v_3} - \underbrace{h(y)}_{v_4}}_{v_2}) \land \neg P(\underbrace{0}_{v_5})$$

$\mathcal{R}$	$\mathcal{L}$	$\mathcal{E}$
$x \le y$		$P(v_2)$
$y \le x + v_1$		$\neg P(v_5)$
$v_2 = v_3 - v_4$	$v_1 = \mathrm{hd}(\mathrm{cons}(v_5, \mathrm{nil}))$	$v_3 = h(x)$
$v_5 = 0$		$v_4 = h(y)$

Nelson-Oppen Combination Method

Variable abstraction + *equality propagation*:

$$x \leq y \wedge y \leq x + \underbrace{\mathrm{hd}(\mathrm{cons}(0, \mathrm{nil}))}_{v_1} \wedge P(\underbrace{\underbrace{h(x)}_{v_3} - \underbrace{h(y)}_{v_4}}_{v_2}) \wedge \neg P(\underbrace{0}_{v_5})$$

$\mathcal{R}$	$\mathcal{L}$	$\mathcal{E}$
$x \leq y$		$P(v_2)$
$y \leq x + v_1$		$\neg P(v_5)$
$v_2 = v_3 - v_4$	$v_1 = \mathrm{hd}(\mathrm{cons}(v_5, \mathrm{nil}))$	$v_3 = h(x)$
$v_5 = 0$		$v_4 = h(y)$
	$v_1 = v_5$	

Nelson-Oppen Combination Method

Variable abstraction + *equality propagation*:

$$x \leq y \wedge y \leq x + \underbrace{\mathrm{hd}(\mathrm{cons}(0, \mathrm{nil}))}_{v_1} \wedge P(\underbrace{\underbrace{h(x)}_{v_3} - \underbrace{h(y)}_{v_4}}_{v_2}) \wedge \neg P(\underbrace{0}_{v_5})$$

$\mathcal{R}$	$\mathcal{L}$	$\mathcal{E}$
$x \leq y$		$P(v_2)$
$y \leq x + v_1$		$\neg P(v_5)$
$v_2 = v_3 - v_4$	$v_1 = \mathrm{hd}(\mathrm{cons}(v_5, \mathrm{nil}))$	$v_3 = h(x)$
$v_5 = 0$		$v_4 = h(y)$
$x = y$	$v_1 = v_5$	

Nelson-Oppen Combination Method

Variable abstraction + *equality propagation*:

$$x \le y \wedge y \le x + \underbrace{\mathrm{hd}(\mathrm{cons}(0, \mathrm{nil}))}_{v_1} \wedge P(\underbrace{\underbrace{h(x)}_{v_3} - \underbrace{h(y)}_{v_4}}_{v_2}) \wedge \neg P(\underbrace{0}_{v_5})$$

$\mathcal{R}$	$\mathcal{L}$	$\mathcal{E}$
$x \le y$		$P(v_2)$
$y \le x + v_1$		$\neg P(v_5)$
$v_2 = v_3 - v_4$	$v_1 = \mathrm{hd}(\mathrm{cons}(v_5, \mathrm{nil}))$	$v_3 = h(x)$
$v_5 = 0$		$v_4 = h(y)$
$x = y$	$v_1 = v_5$	$v_3 = v_4$

Nelson-Oppen Combination Method

Variable abstraction + *equality propagation*:

$$x \leq y \wedge y \leq x + \underbrace{\mathrm{hd}(\mathrm{cons}(0, \mathrm{nil}))}_{v_1} \wedge P(\underbrace{\underbrace{h(x)}_{v_3} - \underbrace{h(y)}_{v_4}}_{v_2}) \wedge \neg P(\underbrace{0}_{v_5})$$

$\mathcal{R}$	$\mathcal{L}$	$\mathcal{E}$
$x \leq y$		$P(v_2)$
$y \leq x + v_1$		$\neg P(v_5)$
$v_2 = v_3 - v_4$	$v_1 = \mathrm{hd}(\mathrm{cons}(v_5, \mathrm{nil}))$	$v_3 = h(x)$
$v_5 = 0$		$v_4 = h(y)$
$x = y$	$v_1 = v_5$	$v_3 = v_4$
$v_2 = v_5$		

Nelson-Oppen Combination Method

Variable abstraction + *equality propagation*:

$$x \leq y \wedge y \leq x + \underbrace{\mathrm{hd}(\mathrm{cons}(0, \mathrm{nil}))}_{v_1} \wedge P(\underbrace{\underbrace{h(x)}_{v_3} - \underbrace{h(y)}_{v_4}}_{v_2}) \wedge \neg P(\underbrace{0}_{v_5})$$

$\mathcal{R}$	$\mathcal{L}$	$\mathcal{E}$
$x \leq y$		$P(v_2)$
$y \leq x + v_1$		$\neg P(v_5)$
$v_2 = v_3 - v_4$	$v_1 = \mathrm{hd}(\mathrm{cons}(v_5, \mathrm{nil}))$	$v_3 = h(x)$
$v_5 = 0$		$v_4 = h(y)$
$x = y$	$v_1 = v_5$	$v_3 = v_4$
$v_2 = v_5$		$\perp$

Further Reading

- Wikipedia article on Automated Theorem Proving
  `en.wikipedia.org/wiki/Automated_theorem_proving`
- Wikipedia article on Boolean Satisfiability Problem (propositional logic)
  `en.wikipedia.org/wiki/Boolean_satisfiability_problem`
- Wikipedia article on Satisfiability Modulo Theories (SMT)
  `en.wikipedia.org/wiki/Satisfiability_Modulo_Theories`
- A good textbook with an emphasis on theory reasoning (arithmetic, arrays) for software verification:

  Aaron Bradley and Zohar Manna, The Calculus of Computation, Springer, 2007
- Another good one, on what the title says, comes with OCaml code:

  John Harrison. Handbook of Practical Logic and Automated Reasoning, Cambridge University Press, 2009

Implemented Systems

- The TPTP (Thousands of Problems for Theorem Provers) is a library of test problems for automated theorem proving
  `www.tptp.org`
- The automated theorem prover SPASS is an implementation of the "modern" version of resolution with equality, the superposition calculus, and comes with a comprehensive set of examples and documentation. A good choice to start with.
  `www.spass-prover.org`
- `users.cecs.anu.edu.au/~baumgart/systems/`