Decision Procedures

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

Summer 2012

Organisation

Organisation

Dates

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

To successfully participate, you must

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

THE CALCULUS OF COMPUTATION:
Decision Procedures with
Applications to Verification

by
Aaron Bradley
Zohar Manna

Springer 2007

Motivation

Motivation

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

- in Hoare-style software verification.
- in hardware verification

Motivation (2)

Consider the following program:

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

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

Motivation (3)

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

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

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

Motivation (4)

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

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

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

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

---

holds if and only if:

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

Motivation (5)

We need an algorithm that decides whether a formula holds.

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

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

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

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

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

Contents of Lecture

Topics

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

Foundations: Propositional Logic

Syntax of Propositional Logic

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

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

Example: Syntax

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

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

Semantics (meaning) of PL

Formula F and Interpretation I is evaluated to a truth value 0/1

where 0 corresponds to value false

1 true

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

Evaluation of logical operators:

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

Example: Semantics

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

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

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

F evaluates to true under I

Inductive Definition of PL's Semantics

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

Base Case:

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

$I \models P$ iff $I[P] = 1$
$I \not\models P$ iff $I[P] = 0$

Inductive Case:

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

Example: Inductive Reasoning

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

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

$$\begin{array}{lll}
1. & I \models P & \text{since } I[P] = 1 \\
2. & I \not\models Q & \text{since } I[Q] = 0 \\
3. & I \models \neg Q & \text{by 2, } \neg \\
4. & I \not\models P \wedge Q & \text{by 2, } \wedge \\
5. & I \models P \vee \neg Q & \text{by 1, } \vee \\
6. & I \models F & \text{by 4, } \rightarrow \qquad \text{Why?}
\end{array}$$

Thus, F is true under I.

Satisfiability and Validity

UNI FREIBURG

Definition (Satisfiability)

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

Definition (Validity)

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

Note

F is valid iff $\neg F$ is unsatisfiable

Proof.

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

Decision Procedure: An algorithm for deciding validity or satisfiability.

Examples: Satisfiability and Validity

Now assume, you are a decision procedure.

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

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

Is there a formula that is unsatisfiable and valid?

Method 1: Truth Tables

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

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

Thus F is valid.

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

P	Q	$P \vee Q$	$P \wedge Q$	F	
0	0	0	0	1	$\leftarrow$ satisfying I
0	1	1	0	0	$\leftarrow$ falsifying I
1	0	1	0	0	
1	1	1	1	1	

Thus F is satisfiable, but invalid.

Method 2: Semantic Argument (Semantic Tableaux)

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

Semantic Argument: Proof rules

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

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

$\nwarrow$or

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

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

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

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

Example

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

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

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

Thus F is valid.

Example 2

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

Let's assume that F is not valid.

1.	$I \not\models F$	assumption
2.	$I \models (P \to Q) \land (Q \to R)$	1, Rule $\to$
3.	$I \not\models P \to R$	1, Rule $\to$
4.	$I \models P$	3, Rule $\to$
5.	$I \not\models R$	3, Rule $\to$
6.	$I \models P \to Q$	2, Rule $\land$
7.	$I \models Q \to R$	2, Rule $\land$

$8a.\quad I \not\models P$	$8b.\quad I \models Q \qquad 6 \to$	
$9a.\quad I \models \bot$	$9ba.\quad I \not\models Q$	$9bb.\quad I \models R$
	$10ba.\quad I \models \bot$	$10bb.\quad I \models \bot$

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

Example 3

Is $\quad F : P \lor Q \to P \land Q \quad$ valid?

Let's assume that F is not valid.

$$
\begin{array}{lll}
1. & I \not\models P \lor Q \to P \land Q & \text{assumption} \\
2. & I \models P \lor Q & 1 \text{ and } \to \\
3. & I \not\models P \land Q & 1 \text{ and } \to
\end{array}
$$

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

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

Falsifying interpretation:

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

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

Normal Forms

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

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

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

Equivalence

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

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

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

F_1 implies F_2 ($F_1 \Rightarrow F_2$)

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

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

Equivalence is a Congruence relation

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

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

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

Negation Normal Form (NNF)

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

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

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

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

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

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

Example: Negation Normal Form

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

$$
\begin{aligned}
& (Q_1 \lor \neg\neg R_1) \land (\neg Q_2 \to R_2) \\
\Leftrightarrow\ & (Q_1 \lor R_1) \land (\neg Q_2 \to R_2) \\
\Leftrightarrow\ & (Q_1 \lor R_1) \land (\neg\neg Q_2 \lor R_2) \\
\Leftrightarrow\ & (Q_1 \lor R_1) \land (Q_2 \lor R_2)
\end{aligned}
$$

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

Disjunctive Normal Form (DNF)

Disjunction of conjunctions of literals

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

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

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

Example

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

$$\begin{array}{lll} & (Q_1 \vee \neg\neg R_1) \wedge (\neg Q_2 \rightarrow R_2) & \\ \Leftrightarrow & (Q_1 \vee R_1) \wedge (Q_2 \vee R_2) & \text{in NNF} \\ \Leftrightarrow & (Q_1 \wedge (Q_2 \vee R_2)) \vee (R_1 \wedge (Q_2 \vee R_2)) & \text{dist} \\ \Leftrightarrow & (Q_1 \wedge Q_2) \vee (Q_1 \wedge R_2) \vee (R_1 \wedge Q_2) \vee (R_1 \wedge R_2) & \text{dist} \end{array}$$

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

Conjunctive Normal Form (CNF)

Conjunction of disjunctions of literals

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

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

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

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

Definition (Equisatisfiability)

F and F' are equisatisfiable, iff

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

Every formula is equisatifiable to either $\top$ or $\bot$.
There is a efficient conversion of F to F' where

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

Note: efficient means polynomial in the size of F.

Basic Idea:

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

The formula

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

is equisatisfiable to F.

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

The time to convert one small formula is constant!

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

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

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

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

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

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

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

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

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

Davis-Putnam-Logemann-Loveland (DPLL) Algorithm

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

Decides the satisfiability of PL formulae in CNF

<u>Decision Procedure DPLL</u>: Given F in CNF

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

Unit Propagation (PROP)

If a clause contains one literal ℓ,

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

Based on resolution

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

Pure Literal Propagagion

Pure Literal Propagation (PLP)

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

Example

$$F : (\neg P \lor Q \lor R) \land (\neg Q \lor R) \land (\neg Q \lor \neg R) \land (P \lor \neg Q \lor \neg R)$$

Branching on Q

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

By unit resolution

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

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

On the other branch

$$F\{Q \mapsto \bot\} : (\neg P \lor R)$$
$$F\{Q \mapsto \bot,\ R \mapsto \top,\ P \mapsto \bot\} = \top \Rightarrow \text{true}$$

F is satisfiable with satisfying interpretation

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

Example

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

$$F$$

$Q \mapsto \top$

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

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

$Q \mapsto \bot$

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

$R \mapsto \top$

$$\neg P$$

$P \mapsto \bot$

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

Knight and Knaves

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

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

Knight and Knaves

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

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

In CNF:

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

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

PROP and PLP are not applicable. Decide on A:

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

By PROP we get:

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

Unsatisfiable! Now set A to $\top$:

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

By PROP we get:

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

Satisfying assignment!

Learning is Useful

Consider the following problem:

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

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

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

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

Partial Assignments and Unit/Conflict Clauses

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

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

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

Conflict Driven Clause Learning (CDCL)

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

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

DPLL with CDCL

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

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

Unit propagation

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

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

The last line does resolution:

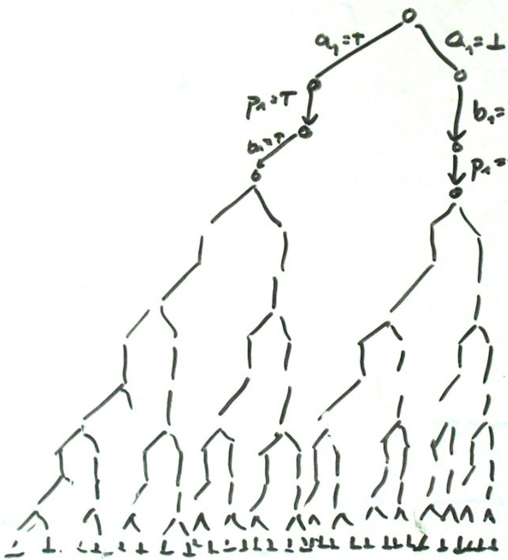
$$\frac{\ell \vee C_1 \qquad \neg\ell \vee C_2}{C_1 \vee C_2}$$

Example

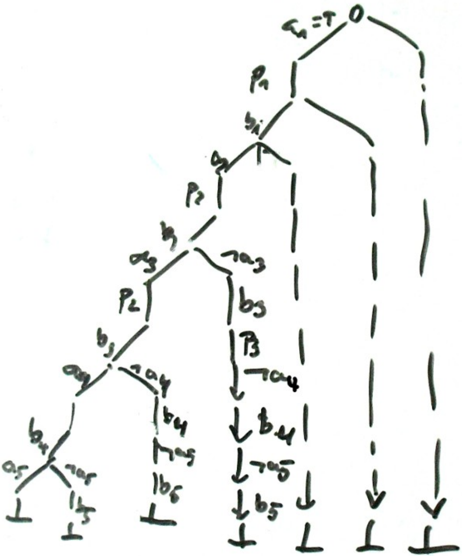
$$\{\{A_1, B_1\}, \{\overline{P_0}, \overline{A_1}, P_1\}, \{\overline{P_0}, \overline{B_1}, P_1\}, \{A_2, B_2\}, \{\overline{P_1}, \overline{A_2}, P_2\}, \{\overline{P_1}, \overline{B_2}, P_2\},$$
$$\dots, \{A_n, B_n\}, \{\overline{P_{n-1}}, \overline{A_n}, P_n\}, \{\overline{P_{n-1}}, \overline{B_n},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

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

DPLL (without Learning)



DPLL with CDCL



Some Notes about DPLL with Learning

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

Summary

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

Further route of this lecture

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

$\Longrightarrow$ Restrictions to decidable fragments of FOL

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

First-Order Logic

Also called Predicate Logic or Predicate Calculus

FOL Syntax

variables	$x, y, z, \cdots$
constants	$a, b, c, \cdots$
functions	$f, g, h, \cdots$ with arity $n > 0$
terms	variables, constants or
	n-ary function applied to n terms as arguments
	$a, x, f(a), g(x, b), f(g(x, f(b)))$
predicates	$p, q, r, \cdots$ with arity $n \geq 0$
atom	$\top$, $\bot$, or an n-ary predicate applied to n terms
literal	atom or its negation
	$p(f(x), g(x, f(x))), \quad \neg p(f(x), g(x, f(x)))$

Note: 0-ary functions: constant

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

Syntax of First-Order Logic (2)

<u>quantifiers</u>

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

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

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

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

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

Example

FOL formula

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

The scope of $\forall x$ is F.
The scope of $\exists y$ is G.
The formula reads:
"for all x,
if $p(f(x), x)$
then there exists a y such that
$p(f(g(x, y)), g(x, y))$ and $q(x, f(x))$"

Famous theorems in FOL

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

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

- Fermat's Last Theorem.

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

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

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

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

FOL Semantics

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

- Domain D_I
  non-empty set of values or objects
  for example $D_I =$ playing cards (finite),
  integers (countable infinite), or
  reals (uncountable infinite)
- Assignment α_I
  - each variable x assigned value $\alpha_I[x] \in D_I$
  - each n-ary function f assigned

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

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

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

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

Example

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

Interpretation $I \ : \ (D_I, \alpha_I)$

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

$$\alpha_I[f] \ : \ D_I^2 \ \rightarrow \ D_I \qquad \alpha_I[g] \ : \ D_I^2 \ \rightarrow \ D_I$$

$$(x,y) \ \mapsto \ x+y \qquad (x,y) \ \mapsto \ x-y$$

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

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

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

Compute the truth value of F under I

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

F is true under I

Semantics: Quantifiers

For a variable x:

Definition (x-variant)

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

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

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

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

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

Example

Consider

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

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

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

What is the truth-value of F?

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

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

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

iff

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

iff

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

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

Example ($\mathbb{Q}$)

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

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

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

iff

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

iff

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

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

Definition (Satisfiability)

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

Definition (Validity)

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

Note

F is valid iff $\neg F$ is unsatisfiable

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

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

should be transformed to

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

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

Substitution

Definition (Substitution)

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

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

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

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

Safe Substitution

Care has to be taken in the presence of quantifiers:

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

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

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

Bounded renaming does not change the models of a formula:

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

Recursive Definition of Substitution

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

$$\begin{aligned} p(t_1, \dots, t_n)\sigma &= p(t_1\sigma, \dots, t_n\sigma) \\ (\neg F)\sigma &= \neg(F\sigma) \\ (F \land G)\sigma &= (F\sigma) \land (G\sigma) \\ \dots \end{aligned}$$

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

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

Example: Safe Substitution $F\sigma$

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

bound by $\forall x$ $\nearrow$ $\nwarrow$ free free $\nearrow$ $\nwarrow$ free

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

$F\sigma$?

1. Rename

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

where x' is a fresh variable

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

Recursive Definition of Substitution

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

$$\begin{aligned}
p(t_1, \ldots, t_n)\sigma &= p(t_1\sigma, \ldots, t_n\sigma) \\
(\neg F)\sigma &= \neg(F\sigma) \\
(F \wedge G)\sigma &= (F\sigma) \wedge (G\sigma) \\
&\ldots
\end{aligned}$$

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

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

Semantic Argument Proof

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

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

Soundness (proof sketch)

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

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

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

Completeness (proof sketch)

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

A (finite or infinite) branch is maximal, if

- it is closed ($I \models \bot$), or
- no new formula can be derived.

A (finite or infinite) tree is maximal, if every branch is maximal.

There is a maximal (possibly infinite) proof tree.

If a branch is closed, it is finite.
If every branch is closed, the tree is finite (Kőnig's Lemma).

In this case, there is a finite semantic argument proof.

Otherwise, there is a maximal (possibly infinite) proof tree with at least one open branch P.

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

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

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

Normal Forms

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

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

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

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

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

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

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

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

Example: Conversion to NNF

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

1. $\forall x.\ (\exists y.\ p(x,y) \land p(x,z)) \to \exists w.\ p(x,w)$
2. $\forall x.\ \neg(\exists y.\ p(x,y) \land p(x,z)) \lor \exists w.\ p(x,w)$

   $F_1 \to F_2 \Leftrightarrow \neg F_1 \lor F_2$
3. $\forall x.\ (\forall y.\ \neg(p(x,y) \land p(x,z))) \lor \exists w.\ p(x,w)$

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

Prenex Normal Form (PNF)

All quantifiers appear at the beginning of the formula

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

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

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

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

Find equivalent PNF of

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

- Write F in NNF

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

- Rename quantified variables to fresh names

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

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

Example: PNF

- Move all quantifiers to the front

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

Alternately,

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

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

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

Note: However $G \;\not\Leftrightarrow\; F$

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

Decidability of FOL

- FOL is undecidable (Turing & Church)
  There does not exist an algorithm for deciding if a FOL formula F is valid, i.e. always halt and says “yes” if F is valid or say “no” if F is invalid.
- FOL is semi-decidable
  There is a procedure that always halts and says “yes” if F is valid, but may not halt if F is invalid.

On the other hand,

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

Similarly for satisfiability