Classical Propositional Logic

Peter Baumgartner

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

Ph: 02 6218 3717

Data61/CSIRO and ANU

July 2017

Classical Logic and Reasoning Problems

A_1: Socrates is a human

A_2: All humans are mortal

Translation into first-order logic:

A_1: human(socrates)

A_2: $\forall X$ (human(X) $\rightarrow$ mortal(X))

Reasoning problems

Which of the following statements hold true? ($\models$ means "entails")

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

Topics of these lectures

- What do these statements *exactly* mean?
- Algorithms/procedures for reasoning problems like the above

Next: some applications

"Application": Mathematical Theorem Proving

First-Order Logic

Can express (mathematical) structures, e.g. groups

$$\forall x\ 1 \cdot x = x \qquad \forall x\ x \cdot 1 = x \qquad \text{(N)}$$

$$\forall x\ x^{-1} \cdot x = 1 \qquad \forall x\ x \cdot x^{-1} = 1 \qquad \text{(I)}$$

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

Reasoning

- Object level: It follows $\forall x\ (x \cdot x) = 1 \rightarrow \forall x, y\ x \cdot y = y \cdot x$
- Meta-level: the word problem for groups is decidable

Automated Reasoning

Computer program to provide the above conclusions *automatically*

Application: Compiler Validation

Problem: prove equivalence of source and target program

```
1:  y := 1
2:  if z = x*x*x
3:    then y := x*x + y
4:  endif
```

```
1: y := 1
2: R1 := x*x
3: R2 := R1*x
4: jmpNE(z,R2,6)
5: y := R1+1
```

To prove: (indexes refer to values at line numbers; index 0 = initial values)

From $\quad y_1 = 1 \ \wedge \ z_0 = x_0 * x_0 * x_0 \ \wedge \ y_3 = x_0 * x_0 + y_1$

and $\quad y'_1 = 1 \ \wedge \ R1_2 = x'_0 * x'_0 \ \wedge \ R2_3 = R1_2 * x'_0 \ \wedge \ z'_0 = R2_3$
$\quad \wedge \ y'_5 = R1_2 + 1 \ \wedge \ x_0 = x'_0 \ \wedge \ y_0 = y'_0 \ \wedge \ z_0 = z'_0$

it follows $\quad y_3 = y'_5$

Application: Constraint Solving

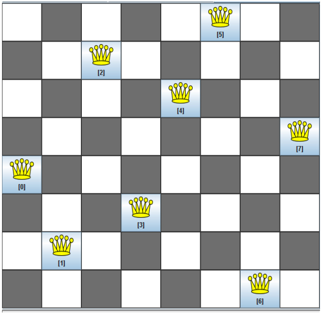
The n-queens problem:

Given: An $n \times n$ chessboard

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

A solution for $n = 8$

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



Application: Constraint Solving

Formalization in sorted first-order logic:

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

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

$$n = 8$$

$$\forall i : \mathbb{Z}\ j : \mathbb{Z}\ (1 \le i \land i \le n \land i + 1 \le j \land j < n \Rightarrow$$
$$p(i) \ne p(j) \land p(i) + i \ne p(j) + j \land p(i) - i \ne p(j) - j) \qquad \text{(Queens)}$$

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

$$\ldots$$

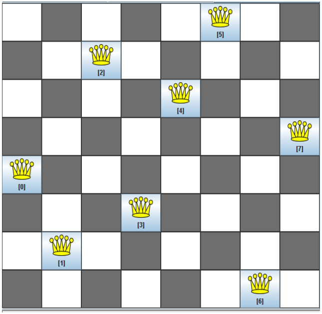
$$p(8) = 1 \lor p(8) = 2 \lor \cdots \lor p(8) = 8 \qquad (p(n) \in \{1, \ldots, n\})$$

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

Task: Find a satisfying interpretation I (a model) and evaluate $I(p(1)), \ldots, I(p(n))$ to read off the answer

Application: System Analysis

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



- 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!

Application: System Analysis

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

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

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

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

$$\forall i : \mathbb{Z}\ j : \mathbb{Z}\ (1 \leq i \land i \leq n \land i + 1 \leq j \land j < n \Rightarrow$$
$$p(i) \neq p(j) \land p(i) + i \neq p(j) + j \land p(i) - i \neq p(j) - j)$$
$$\text{(Queens\_p)}$$

$$\forall i : \mathbb{Z}\ j : \mathbb{Z}\ (1 \leq i \land i \leq n \land i + 1 \leq j \land j < n \Rightarrow$$
$$q(i) \neq q(j) \land q(i) + i \neq q(j) + j \land q(i) - i \neq q(j) - j)$$
$$\text{(Queens\_q)}$$

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

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

Task: Prove logical consequence
(Queens_p) $\land$ ($\forall i : \mathbb{Z}\ q(i) = p(perm(i))$) $\Rightarrow$ (Queens_q)

Issues

- Previous slides gave motivation: *logical analysis of systems*

  System can be "anything that makes sense" and can be described using logic (group theory, computer programs, ...)

- Propositional logic is not very expressive; but it admits *complete* and *terminating* (and sound, and "fast") reasoning procedures
- First-order logic is expressive but not too expressive; it admits *complete* (and sound, and "reasonably fast") reasoning procedures

- So, reasoning with it can be automated on computer. BUT
  - How to do it in the first place: suitable calculi?
  - How to do it efficiently: search space control?
  - How to do it optimally: reasoning support for specific theories like equality and arithmetic?

- The lecture will touch on some of these issues and explain basic approaches to their solution

Contents

Propositional Logic(PL)

PL Syntax

Atom	truth symbols $\top$("true") and $\bot$("false")		
	propositional variables $P, Q, R, P_1, Q_1, R_1, \cdots$		
Literal	atom α or its negation $\neg\alpha$		
Formula	atom 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)

Speaking formally, formulas are defined *inductively*

Example:

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

atoms: $P, Q, \top$

literal: $\neg Q$

subformulas: $P \ \wedge \ Q, \quad \top \ \vee \ \neg Q$

abbreviation (leave parenthesis away)

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

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

Example:

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

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

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")

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 Case:

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

Note:

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

Example:

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

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

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

Thus, F is true under I.

Notation

Extend interpretation I to formulas F:

$$I[F] = \begin{cases} \text{true} & \text{if } I \models F \\ \text{false} & \text{otherwise } (I \not\models F) \end{cases}$$

Inductive Proofs

<u>Induction on the structure of formulas</u>

To prove that a property $\mathcal{P}$ holds for every formula F it suffices to show the following:

Induction start: show that $\mathcal{P}$ holds for every base case formula A

Induction step: Assume that $\mathcal{P}$ holds for arbitrary formulas F_1 and F_2 (*induction hypothesis*).

Show that $\mathcal{P}$ follows for every inductive case formula built with F_1 and F_2

<u>Example</u>

Lemma 1 *Let F be a formula, and I and J be interpretations. If $I[P] = J[P]$ for every propositional variable P occurring in F then $I[F] = J[F]$ (equivalently: $J \models F$ iff $J \models F$).*

Example

Lemma 1 *Let F be a formula, and I and J be interpretations. If $I[P] = J[P]$ for every propositional variable P occurring in F then $I[F] = J[F]$.*

Proof. Assume $I[P] = J[P]$ for every propositional variable P occurring in F. We have to show $I[F] = J[F]$

Induction start
If $F = \top$ or $F = \bot$ then trivially $I[F] = J[F]$.
Otherwise $F = P$ for some propositional variable P. Trivially P occurs in F and hence by assumption $I[F] = I[P] = J[P] = J[F]$.

Example

Lemma 1 *Let F be a formula, and I and J be interpretations. If $I[P] = J[P]$ for every propositional variable P occurring in F then $I[F] = J[F]$.*

Induction step

Case 1: $F = \neg G$ for some formula G.

If $I[F] = \text{true}$ then

$$
\begin{aligned}
I[F] &= \text{true iff} && \\
I[\neg G] &= \text{true iff} && (F = \neg G) \\
I[G] &= \text{false iff} && (\text{Semantics of } \neg) \\
J[G] &= \text{false iff} && (\text{Induction hypothesis}) \\
J[\neg G] &= \text{true iff} && (\text{Semantics of } \neg) \\
J[F] &= \text{true} && (F = \neg G)
\end{aligned}
$$

If $I[F] = \text{false}$: analogously

Example

Lemma 1 *Let F be a formula, and I and J be interpretations. If $I[P] = J[P]$ for every propositional variable P occurring in F then $I[F] = J[F]$.*

Induction step

Case 2: $F = G \wedge H$ for some formulas G and H.
If $I[F] = \text{true}$ then

$$\begin{aligned}
I[F] = \text{true iff} & \\
I[G \wedge H] = \text{true iff} & \qquad (F = G \wedge H) \\
I[G] = \text{true and } I[H] = \text{true iff} & \qquad (\text{Semantics of } \wedge) \\
J[G] = \text{true and } J[H] = \text{true iff} & \qquad (\text{Induction hypothesis 2x}) \\
J[G \wedge H] = \text{true iff} & \qquad (\text{Semantics of } \wedge) \\
J[F] = \text{true} & \qquad (F = G \wedge H)
\end{aligned}$$

If $I[F] = \text{false}$: analogously
Cases 3, 4 and 5 for $\vee$, $\rightarrow$ and $\leftrightarrow$: analogously $\square$

Satisfiability and Validity

F satisfiable iff there exists an interpretation I such that $I \models F$.
F valid iff for all interpretations I, $I \models F$.

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

Method 1: Truth Tables

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

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

Thus F is valid.

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

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. $F_1 : P \land Q$

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

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

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

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

   unsatisfiable, not valid

Method 2: Semantic Argument ("Tableau Calculus")

Proof rules

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

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

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

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

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

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

Example 1: Prove

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

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

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

Thus F is valid.

Example 2: Prove

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

Let's assume that F is not valid.

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

Two cases from 6

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

and

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

Two cases from 7

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

and

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

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

Example 3: Is

$$F : P \lor Q \to P \land Q \quad \text{valid?}$$

Let's assume that F is not valid.

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

Two options

$$\begin{array}{lllllclllll}
4a. & I & \models & P & 2 & \text{or} & 4b. & I & \models & Q & 2 \\
5a. & I & \not\models & Q & 3 & & 5b. & I & \not\models & P & 3
\end{array}$$

We cannot derive a contradiction. F is not valid.

Falsifying interpretation:

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

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

Equivalence

F_1 and F_2 are 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!

Proposition 1 (Substitution Theorem)

Assume $F_1 \Leftrightarrow F_2$. If F is a formula with at least one occurrence of F_1 as a subformula then $F \Leftrightarrow F'$, where F' is obtained from F by replacing some occurrence of F_1 in F by F_2.

Proof.

(Sketch) By induction on the formula structure. For the induction start, if $F = F_1$ then $F' = F_2$, and $F \Leftrightarrow F'$ follows from $F_1 \Leftrightarrow F_2$. The proof of the induction step is similar to the proof of Lemma 1. □

Proposition 1 is relevant for conversion of formulas into normal form, which requires replacing subformulas by equivalent ones

Normal Forms

1. Negation Normal Form (NNF)

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

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

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

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

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

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

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

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

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

2. Disjunctive Normal Form (DNF)

Disjunction of conjunctions of literals

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

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

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

Example: Convert

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

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

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

3. Conjunctive Normal Form (CNF)

Conjunction of disjunctions of literals

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

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

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

Relevance: DPLL and Resolution both work with CNF

Davis-Putnam-Logemann-Loveland (DPLL) Algorithm

Decides the satisfiability of PL formulae in CNF, or clause sets

Clause
A (propositional) clause is a disjunction of literals

Convention
A formula in CNF is taken as a set of clauses. Example:

$(A \lor B)$	$\land$	$(C \lor \neg A)$	$\land$	$(D \lor \neg C \lor \neg A)$	$\land$	$(\neg D \lor \neg B)$	CNF
$\{A \lor B$	,	$C \lor \neg A$	,	$D \lor \neg C \lor \neg A$	,	$\neg D \lor \neg B\}$	Clause Set

Typical Application: Proof by Refutation
To prove the validity of

$$\text{Axiom}_1 \land \cdots \land \text{Axiom}_n \Rightarrow \text{Conjecture}$$

it suffices to prove that the CNF of

$$\text{Axiom}_1 \land \cdots \land \text{Axiom}_n \land \neg\text{Conjecture}$$

is unsatisfiable

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:

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

Example

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

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

Model

A model for a clause set N is an interpretation I such that $I \models N$

DPLL as a Semantic Tree Method

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

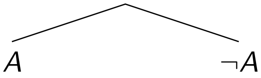
$\langle$empty tree$\rangle$

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

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

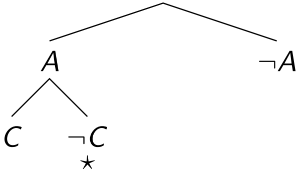


$\{A\} \models A \lor B$
$\{A\} \not\models C \lor \neg A$
$\{A\} \models D \lor \neg C \lor \neg A$
$\{A\} \models \neg D \lor \neg B$

- 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) $\neg D \vee \neg B$

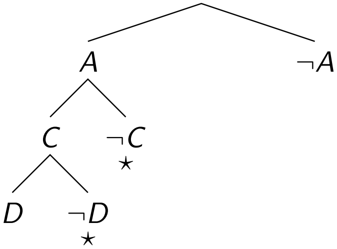


$\{A, C\} \models A \vee B$
$\{A, C\} \models C \vee \neg A$
$\{A, C\} \not\models D \vee \neg C \vee \neg A$
$\{A, C\} \models \neg D \vee \neg B$

- 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) $\neg D \lor \neg B$



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

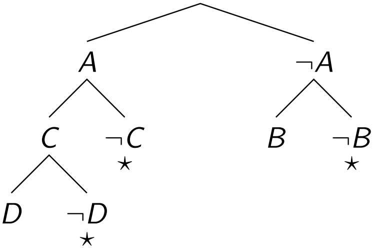
Model $\{A, C, D\}$ found.

- 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

```
function DPLL(N)
  %% N is a set of clauses
  %% returns true if N satisfiable, false otherwise
  while N contains a unit clause {L}
      N := simplify(N, L)
  if N = {} then return true
  if ⊥ ∈ N then return false
  L := choose-literal(N) %% any literal that occurs in N - "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 %% possibly get empty clause ⊥
  return the resulting clause set
```

Simplify Examples

function simplify(N, L) %% also called *unit propagation*
  remove all clauses from N that contain L
  delete $\neg L$ from all remaining clauses %% possibly get empty clause $\perp$
  return the resulting clause set

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

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

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

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

Making DPLL Fast

2-watched literal technique
A technique to implement unit propagation efficiently

- In each clause, select two (currently undefined) "watched" literals.
- For each variable A, keep a list of all clauses in which A is watched and a list of all clauses in which $\neg A$ is watched.
- If an undefined variable is set to false (or to true), check all clauses in which A (or $\neg A$) is watched and watch another literal (that is true or undefined) in this clause if possible.
- As long as there are two watched literals in a n-literal clause, this clause cannot be used for unit propagation, because $n - 1$ of its literals have to be false to provide a unit conclusion.
- Important: Watched literal information need not be restored upon backtracking.

2-Watched Literals Example

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.

Invariant: if clause is not satisfied, watched literals are undefined. Only clauses violating the invariant can unit-propagate

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 $\rightsquigarrow$ 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
5. Now have $C \quad - \quad D \quad - \quad A \quad - \quad B \quad - \quad E$
   Maintains invariant

Invariant is maintained in case of backtracking to $\neg B$:
Then have $C \quad - \quad D \quad - \quad A \quad - \quad \neg B$

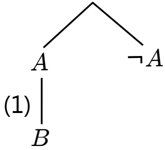
Lemma Learning

"Avoid making the same mistake twice"

$\ldots$

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

w/o Lemma

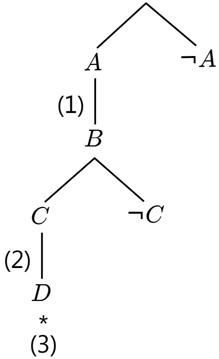


Lemma Learning

"Avoid making the same mistake twice"

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

w/o Lemma



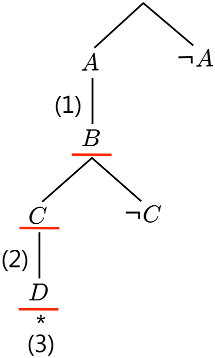
Lemma Learning

"Avoid making the same mistake twice"

$$\ldots$$

$B \lor \neg A$	(1)
$D \lor \neg C$	(2)
$\underline{\neg D} \lor \underline{\neg B} \lor \underline{\neg C}$	(3)

w/o Lemma



Lemma Learning

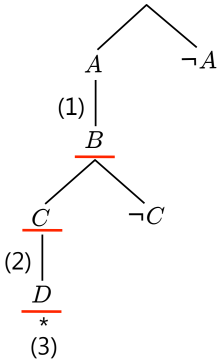
"Avoid making the same mistake twice"

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

Lemma Candidates by Resolution:

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

w/o Lemma



Lemma Learning

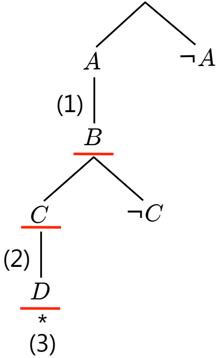
"Avoid making the same mistake twice"

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

Lemma Candidates by Resolution:

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

w/o Lemma



Lemma Learning

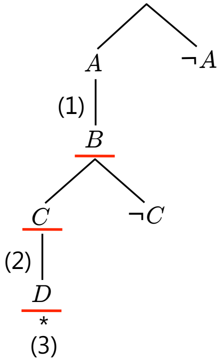
"Avoid making the same mistake twice"

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

Lemma Candidates by Resolution:

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

w/o Lemma



Lemma Learning

"Avoid making the same mistake twice"

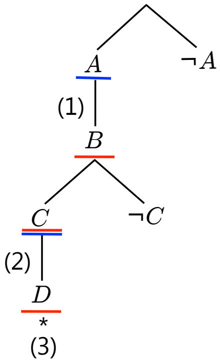
$$\ldots$$

$B \lor \neg A$	(1)
$D \lor \neg C$	(2)
$\underline{\neg D} \lor \underline{\neg B} \lor \underline{\neg C}$	(3)

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

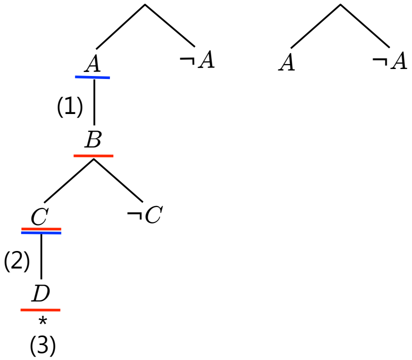
"Avoid making the same mistake twice"

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

Lemma Candidates by Resolution:

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

w/o Lemma **With Lemma**



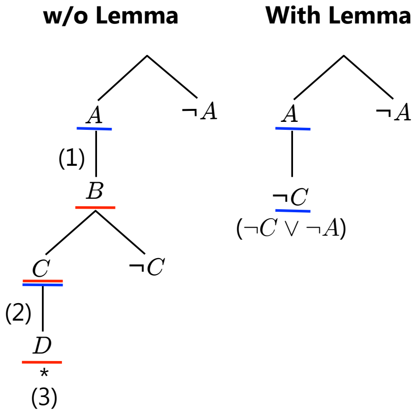
Lemma Learning

"Avoid making the same mistake twice"

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

Lemma Candidates by Resolution:

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



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.

<u>Other Overviews</u>

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.

The Resolution Calculus

DPLL and the refined CDCL algorithm are the practically best methods for PL

The resolution calculus (Robinson 1969) has been introduced as a basis for automated theorem proving in first-order logic. We will see it in detail in the first-order logic part of this lecture

Refined versions are still the practically best methods for first-order logic

The resolution calculus is best introduced first 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*

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 $1 \leq i < n$.

The empty disjunction, or empty clause, is written as $\square$

A refutation (of N) is a derivation from N that contains $\square$

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.	$\Box$	(Res. 4. into 9.)

Soundness and Completeness

Important properties a calculus may or may not have:

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

Deduction completeness:
if N is valid then there is a derivation of N

Refutational completeness:
if N is unsatisfiable then there is a refutation of N

The resolution calculus is sound and refutationally complete, but not deduction complete

Soundness of Propositional Resolution

Theorem 2

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 (1): 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 (2): even simpler □

Completeness of Propositional Resolution

Theorem 3

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 whiteboard)
- This result can be considerably strengthened, some strengthenings come for free from the proof

Semantic Trees

(Robinson 1968, Kowalski and Hayes 1969)

Semantic trees are a convenient device to represent interpretations for possibly infinitely many atoms

Applications

- To prove the completeness of the propositional resolution calculus
- Characterizes a specific, refined resolution calculus
- To prove the compactness theorem of propositional logic. Application: completeness proof of first-order logic Resolution.

Trees

A tree

- is an acyclic, connected, directed graph, where
- every node has at most one incoming edge

A rooted tree has a dedicated node, called root that has no incoming edge

A tree is finite iff it has finitely many vertices (and edges) only

In a finitely branching tree every node has only finitely many edges

A binary tree every node has at most two outgoing edges. It is complete iff every node has either no or two outgoing edges

A path $\mathcal{P}$ in a rooted tree is a possibly infinite sequence of nodes $\mathcal{P} = (\mathcal{N}_0, \mathcal{N}_1, \ldots)$, where $\mathcal{N}_0$ is the root, and $\mathcal{N}_i$ is a direct successor of $\mathcal{N}_{i-1}$, for all $i = 1, \ldots, n$

A path to a node $\mathcal{N}$ is a finite path of the form $(\mathcal{N}_0, \mathcal{N}_1, \ldots, \mathcal{N}_n)$ such that $\mathcal{N} = \mathcal{N}_n$; the value n is the length of the path

The node $\mathcal{N}_{n-1}$ is called the immediate predecessor of $\mathcal{N}$

Every node $\mathcal{N}_0, \mathcal{N}_1, \ldots, \mathcal{N}_{n-1}$ is called a predecessor of $\mathcal{N}$

A (node-)labelled tree is a tree together with a labelling function λ that maps each of its nodes to an element in a given set

Let L be a literal. The complement of L is the literal

$$\overline{L} := \begin{cases} \neg A & \text{if } L \text{ is the atom } A \\ A & \text{if } L \text{ is the negated atom } \neg A. \end{cases}$$

Semantic Trees

A <u>semantic tree $\mathcal{B}$</u> (for a set of atoms $\mathcal{D}$) is a labelled, complete, rooted, binary tree such that

1. the root is labelled by the symbol $\top$
2. for every inner node $\mathcal{N}$, one successor of $\mathcal{N}$ is labeled with the literal A, and the other successor is labeled with the literal $\neg A$, for some $A \in \mathcal{D}$
3. for every node $\mathcal{N}$, there is no literal L such that $L \in \mathcal{I}(\mathcal{N})$ and $\overline{L} \in \mathcal{I}(\mathcal{N})$, where

$$\mathcal{I}(\mathcal{N}) = \{\lambda(\mathcal{N}_i) \mid \mathcal{N}_0, \mathcal{N}_1, \ldots, (\mathcal{N}_n = \mathcal{N}) \text{ is a path to } \mathcal{N} \text{ and } 1 \leq i \leq n\}$$

Semantic Trees

Atom Set

For a clause set N let the atom set (of N) be the set of atoms occurring in clauses in N

A semantic tree for N is a semantic tree for the atom set of N

Path Semantics

For a path $\mathcal{P} = (\mathcal{N}_0, \mathcal{N}_1, \ldots)$ let

$$\mathcal{I}(\mathcal{P}) = \{\lambda(\mathcal{N}_i) \mid i \geq 0\}$$

be the set of all literals along $\mathcal{P}$

Complete Semantic Tree

A semantic tree for $\mathcal{D}$ is complete iff for every $A \in \mathcal{D}$ and every branch $\mathcal{P}$ it holds that

$A \in \mathcal{I}(\mathcal{P})$ or $\neg A \in \mathcal{I}(\mathcal{P})$

Interpretation Induced by a Semantic Tree

Every path $\mathcal{P}$ in a complete semantic tree for $\mathcal{D}$ induces an interpretation $\mathcal{I}_{\mathcal{P}}$ as follows:

$$\mathcal{I}_{\mathcal{P}}[A] = \begin{cases} \text{true} & \text{if } A \in \mathcal{I}_{\mathcal{P}} \\ \text{false} & \text{if } \neg A \in \mathcal{I}_{\mathcal{P}} \end{cases}$$

A complete semantic tree can be seen as an enumeration of all possible interpretations for N (it holds $\mathcal{I}_{\mathcal{P}} \neq \mathcal{I}_{\mathcal{P}'}$ whenever $\mathcal{P} \neq \mathcal{P}'$)

Failure Node

If a clause set N is unsatisfiable (not satisfiable) then, by definition, every interpretation $\mathcal{I}$ falsifies some clause in N, i.e., $\mathcal{I} \not\models C$ for some $C \in N$
This motivates the following definition:

Failure Node
A node $\mathcal{N}$ in a semantic tree for N is a failure node, if

1. there is a clause $C \in N$ such that $\mathcal{I}_N \not\models C$, and
2. for every predecessor $\mathcal{N}'$ of $\mathcal{N}$ it holds:

   there is no clause $C \in N$ such that $\mathcal{I}_{\mathcal{N}'} \not\models C$

Open, Closed

A path $\mathcal{P}$ in a semantic tree for N is closed iff $\mathcal{P}$ contains a failure node, otherwise it is open

A semantic tree $\mathcal{B}$ for M is closed iff every path is closed, otherwise $\mathcal{B}$ is open

Every closed semantic tree can be turned into a finite closed one by removing all subtrees below all failure nodes

Remark
The construction of a (closed or open) finite semantic tree is the core of the propositional DPLL procedure above. Our main application now, however, is to prove compactness of propositional clause logic

Compactness

Theorem 4

A (possibly infinite) clause set N is unsatisfiable iff there is a closed semantic tree for N

Proof.

See whiteboard □

Corollary 5 (Compactness)

A (possibly infinite) clause set N is unsatisfiable iff some finite subset of N is unsatisfiable

Proof.

The if-direction is trivial. For the only-if direction, Theorem 4 gives us a finite unsatisfiable subset of N as identified by the finitely many failure nodes in the semantic tree. □