Propositional Logic

Reading: Chapter 7.1, 7.3 – 7.5

[Partially based on slides from Jerry Zhu, Louis Oliphant and Andrew Moore]

Logic

- If a problem domain can be represented formally, then a decision maker can use logical reasoning to make rational decisions
- Many types of logic
  - Propositional Logic (Boolean logic)
  - First-Order Logic (aka first-order predicate calculus)
  - Non-Monotonic Logic
  - Markov Logic
- A logic includes:
  - **syntax**: what is a correctly-formed sentence?
  - **semantics**: what is the meaning of a sentence?
  - **Inference procedure** (reasoning, entailment): what sentence logically follows given knowledge?

Propositional Logic

- A **symbol** in Propositional Logic (PL) is a symbolic *variable* whose value must be either True or False, and which stands for a natural language statement that could be either true or false
  - A = "Smith has chest pain"
  - B = "Smith is depressed"
  - C = "It is raining"

Propositional Logic Syntax

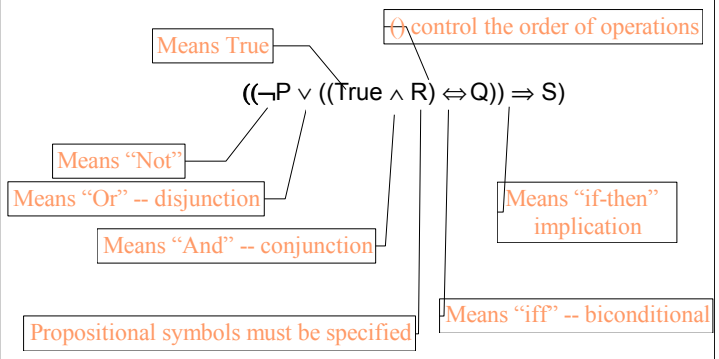
Sentence	→	*AtomicSentence* | *ComplexSentence*
AtomicSentence	→	True | False | *Symbol*
Symbol	→	P | Q | R | . . .
ComplexSentence	→	¬*Sentence*
	|	(*Sentence* ∧ *Sentence*)
	|	(*Sentence* ∨ *Sentence*)
	|	(*Sentence* ⇒ *Sentence*)
	|	(*Sentence* ⇔ *Sentence*)

BNF (Backus-Naur Form) grammar for Propositional Logic

((¬P ∨ ((True ∧ R) ⇔ Q)) ⇒ S) well formed ("wff" or "sentence")

(¬(P ∨ Q) ∧ ⇒ S) not well formed

Propositional Logic Syntax



Propositional Logic Syntax

- Precedence (from highest to lowest):

$$\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$$

- If the order is clear, you can leave off parentheses

¬P ∨ True ∧ R ⇔ Q ⇒ S ok (though not recommended)

P ⇒ Q ⇒ S not ok

Semantics

- An **interpretation** is a complete True / False assignment to ***all*** propositional symbols
  - Example symbols: P means "It is hot", Q means "It is humid", R means "It is raining"
  - There are 8 interpretations (TTT, ..., FFF)
- The semantics (meaning) of a sentence is the set of interpretations in which the sentence evaluates to True
- Example: the semantics of the sentence P ∨ Q is the set of 6 interpretations:
  - P=True, Q=True, R=True or False
  - P=True, Q=False, R=True or False
  - P=False, Q=True, R=True or False
- A **model** of a *set* of sentences is an interpretation in which *all* the sentences are true

Evaluating a Sentence under an Interpretation

- Calculated using the definitions of all the connectives, recursively

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$false$	$false$	$true$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$false$	$true$	$true$	$true$	$true$

- Pay attention to ⇒
  - "5 is even implies 6 is odd" is True!
  - If P is False, regardless of Q, P ⇒ Q is True
  - No causality needed: "5 is odd implies the Sun is a star" is True

Understanding "⇒"

- This is an operator. Although we call it "implies" or "implication," do not try to understand its semantic form from the name. We could have called it "foo" instead and still defined its semantics the same way.
- A ⇒ B "means" A is *sufficient* but not *necessary* to make B true
- Example:
  - Let A be "has a cold" and B be "drink water"
  - A ⇒ B can be interpreted as "should drink water" when "has a cold."
  - However, you can drink water even when you do not have a cold. Thus A ⇒ B is still true when A is ***not*** true.

Example

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

P	Q	R	¬P	Q ∧ R	¬P ∨ (Q ∧ R)	Wff
F	F	F	T	F	T	F
F	F	T	T	F	T	F
F	T	F	T	F	T	T
F	T	T	T	T	T	T
T	F	F	F	F	F	T
T	F	T	F	F	F	T
T	T	F	F	F	F	T
T	T	T	F	T	T	T

Satisfiable: a sentence that is **true** under ***some*** interpretation(s)
Deciding satisfiability of a sentence is NP-complete

Example

$((P \wedge R) \Rightarrow Q) \wedge P \wedge R \wedge \neg Q$

P	Q	R	¬Q	R ∧ ¬Q	P ∧ R ∧ ¬Q	P ∧ R	(P ∧ R) ⇒ Q	Wff
F	F	F	T	F	F	F	T	F
F	F	T	T	T	F	F	T	F
F	T	F	F	F	F	F	T	F
F	T	T	F	F	F	F	T	F
T	F	F	T	F	F	F	T	F
T	F	T	T	T	T	T	F	F
T	T	F	F	F	F	F	T	F
T	T	T	F	F	F	T	T	F

Unsatisfiable: a sentence that is **false** under ***all*** interpretations
Also called **inconsistent** or a **contradiction**

Example

$(P \Rightarrow Q) \vee (P \wedge \neg Q)$

P	Q	R	P ⇒ Q	P ∧ ¬Q	Wff
F	F	F	T	F	T
F	F	T	T	F	T
F	T	F	T	F	T
F	T	T	T	F	T
T	F	F	F	T	T
T	F	T	F	T	T
T	T	F	T	F	T
T	T	T	T	F	T

Valid: a sentence that is **true** under ***all*** interpretations
Also called a **tautology**

Knowledge Base (KB)

- A knowledge base, KB, is a *set of sentences*
  Example KB:
  - ChuckGivingLecture ⇔ (TodayIsTuesday ∨ TodayIsThursday)
  - ¬ChuckGivingLecture
- It is equivalent to a *single* long sentence: the ***conjunction*** of all sentences
  - (ChuckGivingLecture ⇔ (TodayIsTuesday ∨ TodayIsThursday)) ∧ ¬ChuckGivingLecture
- **A model of a KB is an interpretation in which *all* sentences in KB are *true***

Entailment

- **Entailment** is the relation of a sentence β ***logically following*** from other sentences α (e.g., KB)

$$\alpha \models \beta$$

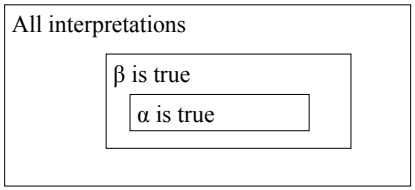
- $\alpha \models \beta$ if and only if, in every interpretation in which α is true, β is also true; i.e., whenever α is true, so is β; **all models of α and also models of β**
- Deduction theorem: **$\alpha \models \beta$ if and only if $\alpha \Rightarrow \beta$ is valid** (always true)
- Proof by contradiction (refutation, *reductio ad absurdum*): **$\alpha \models \beta$ if and only if $\alpha \wedge \neg\beta$ is unsatisfiable**
- There are 2^n interpretations to check, if KB has *n* symbols

Entailment

- Entailment is the relation of a sentence β ***logically following*** from other sentences α (e.g., the KB)

$$\alpha \models \beta$$

- $\alpha \models \beta$ if and only if, in every interpretation in which α is true, β is also true



Deductive Inference

- Say you write a program that, according to you, proves whether a sentence β is entailed by α
- The thing your program does is called ***deductive inference***
- We don't trust your inference program (yet), so we write things your program finds as

$$\alpha \vdash \beta$$

- It reads "β is **derived from** α by your program"
- What properties should your program have?
  - **Soundness**: the inference algorithm only derives entailed sentences. That is, **if $\alpha \vdash \beta$ then $\alpha \models \beta$**
  - **Completeness**: all entailment can be inferred. That is, **if $\alpha \models \beta$ then $\alpha \vdash \beta$**

Soundness and Completeness

- **Soundness** says that any wff that follows deductively from a set of axioms, KB, is valid (i.e., true in all models)
- **Completeness** says that all valid sentences (i.e., true in *all* models of KB), can be proved from KB and hence are theorems

Method 1: Inference by Enumeration

Also called **Model Checking** or **Truth Table Enumeration**

LET: KB = A ∨ C, B ∨ ¬C β = A ∨ B

QUERY: KB ⊨ β ?

A	B	C
false	false	false
false	false	true
false	true	false
false	true	true
true	false	false
true	false	true
true	true	false
true	true	true

NOTE: The computer doesn't know the meaning of the proposition symbols

So, *all* logically distinct cases must be checked to prove that a sentence *can* be derived from KB

Inference by Enumeration

LET: KB = A ∨ C, B ∨ ¬C β = A ∨ B

QUERY: KB ⊨ β ?

A	B	C	A∨C	B∨ ¬C	KB
false	false	false	false	true	false
false	false	true	true	false	false
false	true	false	false	true	false
false	true	true	true	true	true
true	false	false	true	true	true
true	false	true	true	false	false
true	true	false	true	true	true
true	true	true	true	true	true

Rows where *all* of sentences in KB are true are the models of KB

Inference by Enumeration

LET: KB = A ∨ C, B ∨ ¬C β = A ∨ B

QUERY: KB ⊨ β ? ***YES!***

A	B	C	A∨C	B∨ ¬C	KB	A∨B	KB⇒β
false	false	false	false	true	false	false	true
false	false	true	true	false	false	false	true
false	true	false	false	true	false	true	true
false	true	true	true	true	true	true	true
true	false	false	true	true	true	true	true
true	false	true	true	false	false	true	true
true	true	false	true	true	true	true	true
true	true	true	true	true	true	true	true

β is entailed by KB if *all* models of KB are models of β, i.e., *all* rows where KB is true, β is also true

In other words: KB ⇒ β is valid

Inference by Enumeration

- Using inference by enumeration to build a complete truth table in order to determine if a sentence is entailed by KB is a **complete** inference algorithm for Propositional Logic
- But very slow: takes exponential time

Method 2: Natural Deduction using Sound Inference Rules

Goal: Define a more efficient algorithm than enumeration that uses a set of inference rules to *incrementally deduce new sentences* that are true given the initial set of sentences in KB, plus uses all logical equivalences

Logical Equivalences

$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha) \quad \text{commutativity of } \wedge$$
$$(\alpha \vee \beta) \equiv (\beta \vee \alpha) \quad \text{commutativity of } \vee$$
$$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma)) \quad \text{associativity of } \wedge$$
$$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma)) \quad \text{associativity of } \vee$$
$$\neg(\neg\alpha) \equiv \alpha \quad \text{double-negation elimination}$$
$$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha) \quad \text{contraposition}$$
$$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta) \quad \text{implication elimination}$$
$$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination}$$
$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \quad \text{de Morgan}$$
$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \quad \text{de Morgan}$$
$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{distributivity of } \wedge \text{ over } \vee$$
$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{distributivity of } \vee \text{ over } \wedge$$

You can use these equivalences to derive or modify sentences

Sound Inference Rules

- ***Modus Ponens*** (Latin for "mode that affirms")

$$\frac{\alpha \Rightarrow \beta,\ \alpha}{\beta}$$

- **And-Elimination**

$$\frac{\alpha \wedge \beta}{\alpha}$$

Note: Prove that an inference rule is **sound** by using a truth table, e.g.:

P	Q	P	P⇒Q	P ∧ (P⇒Q)	Q	(P ∧ (P⇒Q)) ⇒ Q
T	T	T	T	T	T	T
T	F	T	F	F	F	T
F	T	F	T	F	T	T
F	F	F	T	F	F	T

Some Sound Inference Rules

- **Implication-Elimination, IE (Modus Ponens, MP)** $\dfrac{\alpha \Rightarrow \beta,\ \alpha}{\beta}$
- **And-Elimination, AE** $\dfrac{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}{\alpha_i}$
- **And-Introduction, AI** $\dfrac{\alpha_1,\ \alpha_2,\ \ldots,\ \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}$
- **Or-Introduction, OI** $\dfrac{\alpha_i}{\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n}$
- **Double-Negation Elimination, DNE** $\dfrac{\neg\neg\alpha}{\alpha}$

Inference Rules

- Each inference rule formalizes the idea that
  "A infers B" ($A \vdash B$) in terms of
  "logically follows" ($A \models B$)
- Doesn't say anything about **deducibility** – just says for each interpretation that makes A true, that interpretation also makes B true

Question

What's the difference between

- $\equiv$ (logical equivalence)
- $\models$ (entails)
- $\vdash$ (derived from; infers)

Natural Deduction = Constructing a Proof

- A **Proof** is a sequence of inference steps that leads from α (i.e., KB) to β (i.e., query)
- This is a search problem!

KB:
1. $(P \wedge Q) \Rightarrow R$
2. $(S \wedge T) \Rightarrow Q$
3. S
4. T
5. P

Query:
R

Proof by Natural Deduction

1. S Premise (i.e., given sentence in KB)
2. T Premise
3. S ∧ T Conjunction(1, 2) (And-Introduction)
4. (S ∧ T) ⇒ Q Premise
5. Q Modus Ponens(3, 4)
6. P Premise
7. P ∧ Q Conjunction(5, 6)
8. (P ∧ Q) ⇒ R Premise
9. R Modus Ponens(7, 8) (Last line is query sentence)

Monotonicity Property

- Note that natural deduction relies on the **monotonicity property** of Propositional Logic:

  Deriving a new sentence and adding it to KB does ***NOT*** affect what can be entailed from the *original* KB

- Hence **we can incrementally add new true sentences that are derived in *any order***
- Once something is proved true, it will remain true

Proof by Natural Deduction

KB:
1. ChuckGivingLecture ⇔ (TodayIsTuesday ∨ TodayIsThursday)
2. ¬ ChuckGivingLecture

Query:
¬ TodayIsTuesday

Proof

KB:
1. ChuckGivingLecture ⇔ (TodayIsTuesday ∨ TodayIsThursday)
2. ¬ ChuckGivingLecture

3. (ChuckGivingLecture ⇒ (TodayIsTuesday ∨ TodayIsThursday)) ∧ ((TodayIsTuesday ∨ TodayIsThursday) ⇒ ChuckGivingLecture) iff/biconditional-elimination to 1
4. (TodayIsTuesday ∨ TodayIsThursday) ⇒ ChuckGivingLecture and-elimination to 3
5. ¬ ChuckGivingLecture ⇒ ¬(TodayIsTuesday ∨ TodayIsThursday) contraposition to 4
6. ¬(TodayIsTuesday ∨ TodayIsThursday) Modus Ponens 2,5
7. ¬TodayIsTuesday ∧ ¬TodayIsThursday de Morgan to 6
8. ¬ TodayIsTuesday and-elimination to 7

Resolution Rule of Inference

- **Resolution Rule** of Inference

$$\frac{\alpha \vee \beta, \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

- Examples

$$\frac{A \vee B, \neg B}{A} \qquad \frac{A \vee B \vee \neg C \vee D, \neg A \vee \neg E \vee F}{B \vee \neg C \vee D \vee \neg E \vee F}$$

called "**unit resolution**"

Resolution

- Take any two "clauses" where one contains some symbol, and the other contains its complement (negative)

$$P \vee \mathbf{Q} \vee R \qquad \neg\mathbf{Q} \vee S \vee T$$

- Merge (resolve) them, by throwing away the symbol and its complement, to obtain their **resolvent** clause:

$$P \vee R \vee S \vee T$$

- If two clauses resolve and there's no symbol left, you have derived **False**, aka the ***empty clause***

Method 3: Resolution Refutation

- Show KB $\models \alpha$ by proving that KB $\wedge \neg\alpha$ is *unsatsifiable*, i.e., deducing False from KB $\wedge \neg\alpha$
- Your algorithm can use all the logical equivalences to derive new sentences, plus:
- **Resolution rule**: a ***single*** inference rule
  - **Sound**: only derives entailed sentences
  - **Complete**: can derive any entailed sentence
    - **Resolution is refutation complete**:

      if KB $\models \beta$, then KB $\wedge \neg \beta \vdash$ False
  - But the sentences need to be preprocessed into a special form
  - But all sentences *can* be converted into this form

Resolution Refutation Algorithm

1. Add negation of query to KB
2. Pick 2 sentences that haven't been used before and can be used with the Resolution Rule of inference
3. If none, halt and answer that the query is *NOT* entailed by KB
4. Compute resolvent and add it to KB
5. If False in KB
   - Then halt and answer that the query *IS* entailed by KB
   - Else Goto 2

Conjunctive Normal Form (CNF)

1. Replace all $\Leftrightarrow$ using iff/biconditional elimination
   - $\alpha \Leftrightarrow \beta \equiv (\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
2. Replace all $\Rightarrow$ using implication elimination
   - $\alpha \Rightarrow \beta \equiv \neg\alpha \vee \beta$
3. Move all negations inward using
   - double-negation elimination
     $\neg(\neg\alpha) \equiv \alpha$
   - de Morgan's rule
     $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
     $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$
4. Apply distributivity of $\vee$ over $\wedge$
   - $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$ + 1 more

Convert Sentence into CNF

$A \Leftrightarrow (B \vee C)$	starting sentence
$(A \Rightarrow (B \vee C)) \wedge ((B \vee C) \Rightarrow A)$	iff/biconditional elimination
$(\neg A \vee B \vee C) \wedge (\neg(B \vee C) \vee A)$	implication elimination
$(\neg A \vee B \vee C) \wedge ((\neg B \wedge \neg C) \vee A)$	move negations inward
$(\neg A \vee B \vee C) \wedge (\neg B \vee A) \wedge (\neg C \vee A)$	distribute $\vee$ over $\wedge$

called a "**clause**"

Resolution Refutation Steps

- Given KB and β (query)
- Add $\neg\beta$ to KB, and convert all sentences to CNF
- Show this leads to False (aka "empty clause"). Proof by contradiction

- Example KB:
  - $A \Leftrightarrow (B \vee C)$
  - $\neg A$
- Example query: $\neg B$

Resolution Refutation Preprocessing

- Add $\neg\beta$ to KB, and convert to CNF:

  a1: $\neg A \vee B \vee C$
  a2: $\neg B \vee A$
  a3: $\neg C \vee A$
  b: $\neg A$
  c: B

- Want to reach goal: False (empty clause)

Resolution Refutation Example

a1: $\neg A \vee B \vee C$
a2: $\neg B \vee A$
a3: $\neg C \vee A$
b: $\neg A$
c: B

Step 1: resolve a2, c: A

Step 2: resolve above and b: *empty clause / false*

Example

- Given:
  - $P \vee Q$
  - $P \Rightarrow R$
  - $Q \Rightarrow R$
- Prove: R

Example

- Given:
  - $(P \Rightarrow Q) \Rightarrow Q$
  - $(P \Rightarrow P) \Rightarrow R$
  - $(R \Rightarrow S) \Rightarrow \neg(S \Rightarrow Q)$
- Prove: R

Example

- Given:
  - P
  - $\neg P$
- Prove: R

Efficiency of the Resolution Refutation Algorithm

- Run time can be exponential in the worst case
  - Often much faster
- Factoring: if a new clause contains duplicates of the same symbol, delete the duplicates

  $P \vee R \vee P \vee T \equiv P \vee R \vee T$
- If a clause contains a symbol and its complement, the clause is a tautology and is useless; it can be thrown away

  a1: $(\neg A \vee B \vee C)$

  a2: $(\neg B \vee A)$

  Resolvent of a1 and a2 is: $B \vee C \vee \neg B$

  Which is valid, so throw it away

Resolution Refutation Strategies

- Resolution refutation proofs can be thought of as search:
  - reversed construction of search tree (leaves to root)
  - leaves are KB clauses and $\neg$query
  - resolvent is new node with arcs to parent clauses
  - root is False clause

Resolution Refutation Strategies

- **Breadth-First**
  - level 0 clauses: KB clauses and $\neg$query
  - level k clauses: resolvents computed from 2 clauses:
    - one of which must be from level k-1
    - other from any earlier level
  - compute all possible level 1 clauses, then all possible level 2 clauses, etc.
  - **complete but very inefficient**

Resolution Refutation Strategies

- **Input Resolution**
  - *P* and *Q* can be resolved if at least one is from the set of original clauses, i.e., KB and $\neg$query
  - proof trees have a single "spine" (see Fig. 9.11)
  - Modus Ponens is a form of input resolution since each step is used to generate a new fact
  - **complete for FOL KB with only Horn clauses**

Resolution Refutation Strategies

- **Linear Resolution**
  - a slight generalization of input resolution
  - *P* and *Q* can be resolved if:
    - at least 1 is from the set of original clauses
    - or *P* must be an ancestor of *Q* in the proof tree
  - **complete**

Method 4: Chaining with Horn Clauses

- Resolution is more powerful than we need for many practical situations
- A weaker form: **Horn clauses**
  - **A Horn clause is a disjunction of literals with *at most one positive***

    $\neg R \vee P \vee Q$ no

    $\neg R \vee \neg P \vee Q$ yes

  - KB = conjunction of Horn clauses
  - What's the big deal?

    $\neg R \vee \neg P \vee Q$

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

    $\equiv$?

Horn Clauses

$\neg R \vee \neg P \vee Q$

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

$\equiv$ **$(R \wedge P) \Rightarrow Q$** Every **rule** in KB is in this form

P (special case, no negative literals): **fact**

$\neg R \vee \neg P$ (special case, no positive literal): **goal clause**

- The big deal:
  - KB easy for humans to read
  - Natural forward chaining and backward chaining algorithms; proof easy for humans to read
  - Can decide entailment with Horn clauses in time **linear** with KB size
- But …
  - Can only ask atomic queries

Horn Clauses

Only 1 rule of inference needed:

Generalized Modus Ponens

$$\frac{P,\ Q,\ (P \wedge Q) \Rightarrow R}{R}$$

Forward Chaining

- "Apply" any rule whose premises are satisfied in the KB
- Add its conclusion to the KB until query is derived

KB:

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

query: Q

- Forward chaining with **Horn clause KB** is **complete**

Forward Chaining

1. $\mathbf{P \Rightarrow Q}$
2. $\mathbf{L \wedge M \Rightarrow P}$
3. $\mathbf{B \wedge L \Rightarrow M}$
4. $\mathbf{A \wedge P \Rightarrow L}$
5. $\mathbf{A \wedge B \Rightarrow L}$
6. **A**
7. **B**
8. **L** GMP(5,6,7)
9. **M** GMP(3,7,8)
10. **P** GMP(2,8,9)
11. **Q** GMP(1,10)

Backward Chaining

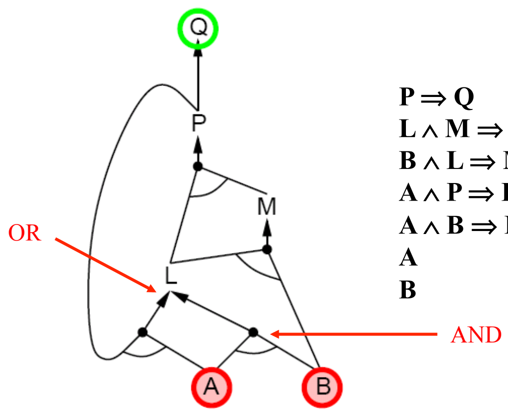
- Forward chaining problem: can generate a lot of irrelevant conclusions
  - Search forward, start state = KB, goal test = state contains query
- Backward chaining
  - **Work backwards from goal to premises**
  - Find all implications of the form

    $(\ldots) \Rightarrow$ query
  - Prove all the premises of one of these implications
  - Avoid loops: check if new subgoal is already on the goal stack
  - Avoid repeated work: check if new subgoal
    1. Has already been proved true, or
    2. Has already failed

Backward Chaining

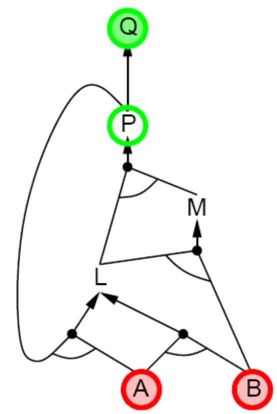
1. $\mathbf{P \Rightarrow Q}$
2. $\mathbf{L \wedge M \Rightarrow P}$
3. $\mathbf{B \wedge L \Rightarrow M}$
4. $\mathbf{A \wedge P \Rightarrow L}$
5. $\mathbf{A \wedge B \Rightarrow L}$
6. **A**
7. **B**
8. **Q** Goal
9. **P** Subgoal(1,8)
10. **L ∧ M** Subgoal(2,9)
11. **L** Subgoal(10)
12. **A ∧ B** Subgoal(5,11)
13. **A** True(6)
14. **B** True(7)
15. **L** True(5,13,14)
16. **M** True(14,15)
17. **P** True(15,16)
18. **Q** True(1,17)

Backward Chaining



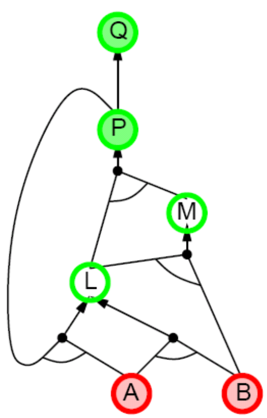
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Backward Chaining



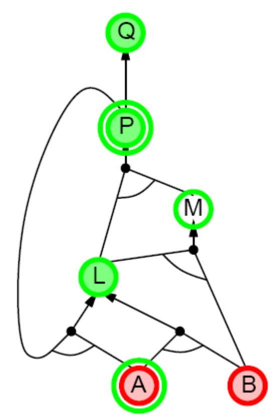
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Backward Chaining



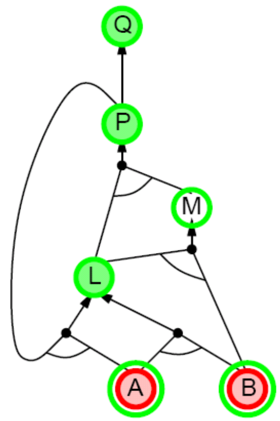
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Backward Chaining



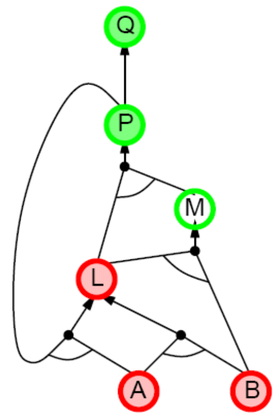
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Backward Chaining



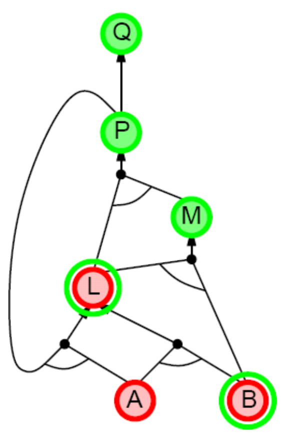
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Backward Chaining



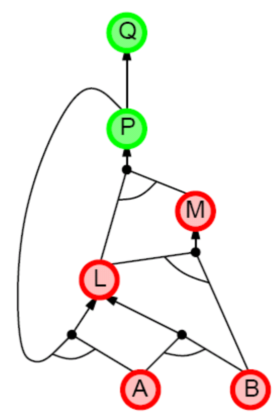
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Backward Chaining



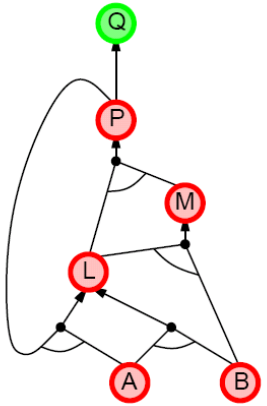
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Backward Chaining



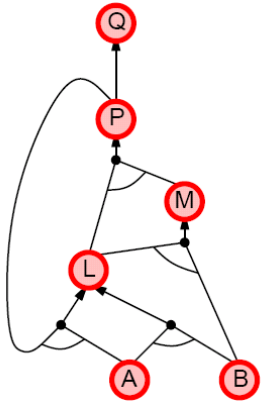
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Backward Chaining



$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Backward Chaining



$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Forward vs. Backward Chaining

- **Forward chaining** is data-driven
  - May perform lots of work irrelevant to the goal
- **Backward chaining** is goal-driven
  - Appropriate for problem solving
  - Time complexity can be much less than linear in size of KB
- Some form of bi-directional search may be even better

Prolog: A Logic Programming Language

- A Program =
  - a set of logic sentences as Horn clauses
    - called the database (DB), i.e., the KB
    - ordered by programmer
  - executed by specifying a query to be proved
    - uses **backward-chaining**
    - uses **depth-first search** on the ordered facts and rules
    - searches until a solution is found

Prolog Basic Syntax

- Database:
  - **Fact:** a positive literal

    FOL: `F(x)` *in Prolog:* `F(X).`

    initial capital variables universally quantified
  - **Rules:** ≥ 1 negative and 1 positive literals

    if antecedent(s) then consequent

    FOL: $A_1 \wedge A_2 \wedge \ldots \wedge A_n \Rightarrow C$ *in Prolog:* `C :-` $A_1, A_2, \ldots, A_n$.
- Query:
  - *FOL:* $Q_1 \wedge Q_2 \wedge \ldots \wedge Q_n$ *in Prolog:* `?-` $Q_1, Q_2, \ldots, Q_n$.
  - query variables implicitly existentially quantified

Some Applications of PL

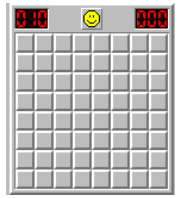
- Puzzles (e.g., Sudoku)
- Scheduling problems
- Layout problems
- Boolean circuit analysis
- Automated theorem provers
- Legal reasoning systems

Weaknesses of PL

- PL is not a very expressive language
- Can't express relations over a group of things, e.g., "All triangles have 3 sides"
- Only deals with "facts," e.g., "It is raining," but does not allow variables where you can express things about them without naming them explicitly. For example, "When you paint a block with green paint, it becomes green."
- You can't quantify things, e.g., talk about all of them, some of them, none of them, without naming them explicitly

Problems with Propositional Logic

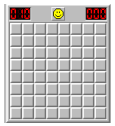
- Consider the game "Minesweeper" on a 10 x 10 field with only one land mine



- How do you express the knowledge, with Propositional Logic, that the squares adjacent to the land mine will display the number 1?

Problems with Propositional Logic

- Consider the game "Minesweeper" on a 10 x 10 field with only one land mine



- How do you express the knowledge, with Propositional Logic, that the squares adjacent to the land mine will display the number 1?
- Intuitively with a rule like

  Landmine(x,y) ⇒ Number1(Neighbors(x,y))

  but Propositional Logic **cannot** do this

Problems with Propositional Logic

- Propositional Logic has to say, e.g. for cell (3, 4):
  - Landmine_3_4 ⇒ Number1_2_3
  - Landmine_3_4 ⇒ Number1_2_4
  - Landmine_3_4 ⇒ Number1_2_5
  - Landmine_3_4 ⇒ Number1_3_3
  - Landmine_3_4 ⇒ Number1_3_5
  - Landmine_3_4 ⇒ Number1_4_3
  - Landmine_3_4 ⇒ Number1_4_4
  - Landmine_3_4 ⇒ Number1_4_5
  - And similarly for *each* of Landmine_1_1, Landmine_1_2, Landmine_1_3, …, Landmine_10_10
- Difficult to express large domains concisely
- Don't have objects and relations
- First-Order Logic is a powerful upgrade

Other Logic Systems

Logics are characterized by what they commit to as "primitives"

Logic	What Exists in World	Knowledge States
Propositional	facts	true/false/unknown
First-Order	facts, objects, relations	true/false/unknown
Temporal	facts, objects, relations, times	true/false/unknown
Probability Theory	facts	degree of belief 0..1
Markov	facts, objects, relations	degree of belief 0..1

First-Order Logic (FOL)

Also known as First-Order Predicate Calculus (FOPC)

- Constants: Bob, 2, Madison, …
- Functions: Income, Address, Sqrt, …
- Predicates: Sister, Teacher, ≤, …
- Variables: x, y, a, b, c, …
- Connectives: ∧ ∨ ¬ ⇒ ⇔
- Equality: =
- Quantifiers: ∀ ∃

FOL Syntax: Quantifiers

Universal quantifier: ∀<variable> <sentence>

- Means the sentence is true **for all** values of *x* in the domain of variable *x*
- Main connective typically ⟹ forming if-then rules
  - **All humans are mammals** **becomes in FOL:**
    ∀x Human(x) ⇒ Mammal(x)
    i.e., for all *x*, if *x* is a human then *x* is a mammal
  - **Mammals must have fur** **becomes in FOL:**
    ∀x Mammal(x) ⇒ HasFur(x)
    for all *x*, if *x* is a mammal then *x* has fur

FOL Syntax: Quantifiers

Existential quantifier: ∃<variable> <sentence>

- Means the sentence is true
  for some value of *x* in the domain of variable *x*
- Main connective is typically ∧
  - **Some humans are old** **becomes in FOL:**
  - ∃x Human(x) ∧ Old(x)
    there exist an *x* such that *x* is a human and *x* is old
  - **Mammals may have arms.** **becomes in FOL:**
  - ∃x Mammal(x) ∧ HasArms(x)
    there exist an x such that x is a mammal and x has arms

Fun with Sentences

- Good people always have friends.
  could mean: All good people have friends.
  ∀x (Person(x) ∧ Good(x)) ⇒ ∃y(Friend(x,y))
- Busy people sometimes have friends.
  could mean: Some busy people have friends.
  ∃x Person(x) ∧ Busy(x) ∧ ∃y(Friend(x,y))
- Bad people never have friends.
  could mean: Bad people have no friends.
  ∀x (Person(x) ∧ Bad(x)) ⇒ ¬∃y(Friend(x,y))
  or equivalently: No bad people have friends.
  ¬∃x Person(x) ∧ Bad(x) ∧ ∃y(Friend(x,y))

Fun with Sentences

- **There is exactly one shoe.**
  ∃x Shoe(x) ∧ ∀y(Shoe(y) ⇒ (x=y))
- **There are exactly two shoes.**
  ∃x,y Shoe(x) ∧ Shoe(y) ∧ ¬(x=y) ∧
  ∀z (Shoe(z) ⇒ (x=z) ∨ (y=z))

What You Should Know

- A lot of terms
- Use truth tables (inference by enumeration)
- Natural deduction proofs
- Conjuctive Normal Form (CNF)
- Resolution Refutation algorithm and proofs
- Horn clauses
- Forward chaining algorithm
- Backward chaining algorithm

That's All Folks!