MONASH University
Information Technology

MONASH University
Information Technology

FIT5047: Fundamentals of AI

Knowledge Representation
Chapters 6, 8-9

Knowledge representation: Learning objectives

- **Knowledge-based agents**
- **Logic in general – models and entailment**
- **Propositional (Boolean) logic**
- **Equivalence, validity, satisfiability**
- **First-order logic**
- **Inference rules and theorem proving**
  - forward chaining
  - backward chaining
  - resolution refutation systems
    - > substitution
    - > unification
    - > resolution

Propositional logic

First order logic

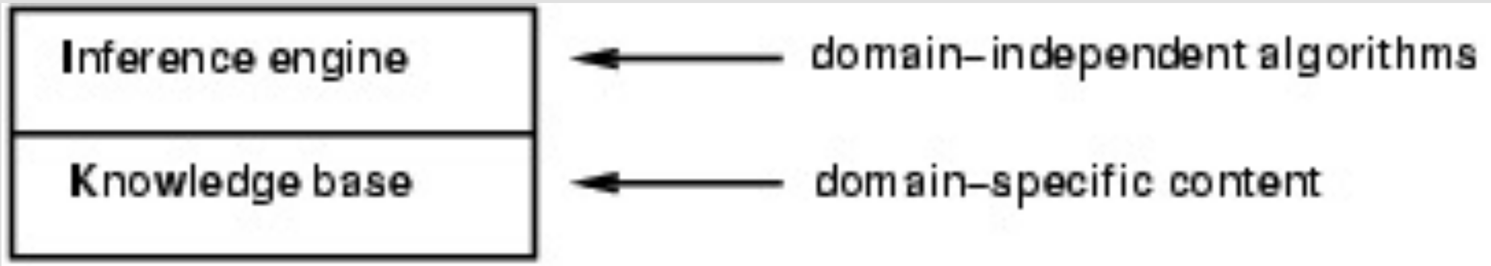
Assumptions about the environment

- **Observable**
- **Known**
- **Single/multi agent**
- **Deterministic**
- **Sequential/episodic**
- **Static**
- **Discrete**

Knowledge representation

- **How do we represent facts about the world?**
- **How do we reason about these facts?**
- **Some widely accepted formal calculi**
  - Propositional logic
  - First-order logic
  - Probability calculus

Knowledge bases



- **Knowledge base = set of sentences in a formal language**
- **Declarative approach to building an agent:**
  - **Tell** it what it needs to know
  - **Ask** it a question – answers follow by inference from the KB
- **Agents may be viewed**
  - at the **knowledge level** – what they know
  - at the **implementation level** – data structures in the KB and algorithms that manipulate them

A simple knowledge-based agent

```
function KB-AGENT(percept) returns an action
   static: KB, a knowledge base
           t, a counter, initially 0, indicating time

   TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
   action ← ASK(KB, MAKE-ACTION-QUERY(t))
   TELL(KB, MAKE-ACTION-SENTENCE(action, t))
   t ← t + 1
   return action
```

- **The agent must be able to:**
  - Represent states, actions, etc
  - Incorporate new percepts
  - Update internal representations of the world
  - Deduce hidden properties of the world
  - Deduce appropriate actions

Logic in general

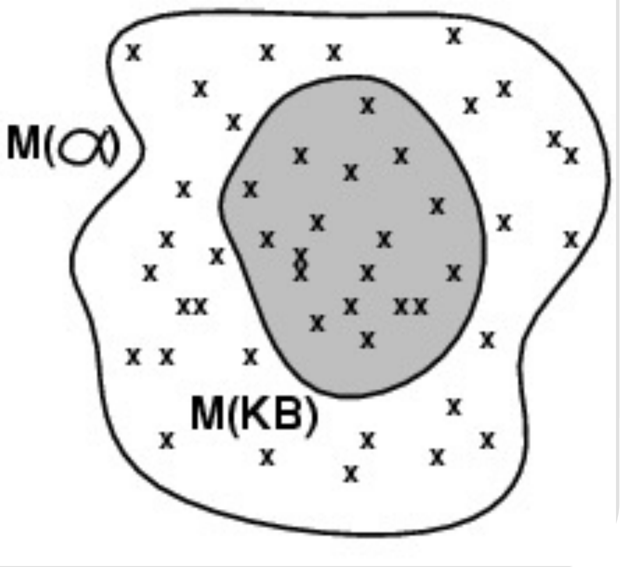
- **Logics are formal languages for representing information such that conclusions can be drawn**
- **Syntax defines the sentences in the language**
- **Semantics define the meaning of sentences**
  - the truth of a sentence in each **possible world**
- **E.g., the language of arithmetic**
  - $x+2 \geq y$ is a sentence; $x2+y > \{\}$ is not a sentence
  - $x+2 \geq y$ is true in all the worlds where the number $x+2$ is no less than the number y
    - $x+2 \geq y$ is true in a world where $x = 7, y = 1$
    - $x+2 \geq y$ is false in a world where $x = 0, y = 6$

Logical entailment

- **Entailment means that one thing follows logically from another:**
  KB $\models$ **α**
  knowledge base *KB* entails sentence α iff α is true in all worlds where *KB* is true
- **E.g.,**
  – the KB containing “the Giants won” and “the Reds won” entails “the Giants won or the Reds won”
  – x+y = 4 entails 4 = x+y
- **Entailment is a relationship between sentences (i.e., syntax) that is based on semantics**

Models

- **Models are formally structured worlds with respect to which truth can be evaluated**
- **If a sentence α is true in a model *m*, we say that**
  - *m* **is a model of** α, or
  - *m* **satisfies** α
- ***M(α)* = the set of all models of α**
  - KB $\models$ α iff $M(KB) \subseteq M(\alpha)$
    - > E.g., KB = Giants won & Reds won
      α = Giants won



Inference

- *KB* $\vdash_i$ α means that sentence α can be derived from *KB* by procedure *i*
- **Soundness**: procedure *i* is sound if whenever *KB* $\vdash_i$ α, it is also true that *KB* $\models$ α
- **Completeness**: procedure *i* is complete if whenever *KB* $\models$ α, it is also true that *KB* $\vdash_i$ α
- First-order logic (FOL) is expressive enough to say almost anything of interest, and there exists a sound and complete inference procedure for it

Grounding

About the connection between logical reasoning processes and real environments

- **How do we know that KB is true in the real world?**
  – By creating a connection using the agent's sensors, i.e., the meaning and truth of percept sentences are defined by the processes of sensing and sentence construction
- **Where do we get the rest of an agent's knowledge?**
  – By learning (generalizing) from experience

MONASH University
Information Technology

FIT5047: Fundamentals of AI

Propositional Logic

Propositional Logic: Some definitions

- **Literal:** a proposition or its negation
  - E.g., P, $\neg$P
- **Clause**: a disjunction of literals
  - E.g., $\neg P \vee Q \vee A$

Propositional Logic: Syntax

- **Propositional logic is the ``simplest'' logic**
- **The proposition symbols P_1, P_2, … are sentences**
  - **Negation**: If S is a sentence, $\neg S$ is a sentence
  - **Conjunction**:
    If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence
  - **Disjunction**:
    If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence
  - **Implication**:
    If S_1 and S_2 are sentences, $S_1 \Rightarrow S_2$ is a sentence
  - **Biconditional**:
    If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence

Propositional Logic: Semantics

- **Each model specifies true/false for each proposition**
  - E.g.,

S_1	S_2	S_3
false	true	false

- **Rules for evaluating truth with respect to a model *m*:**

$\neg S$ is true iff S is false

$S_1 \wedge S_2$ is true iff S_1 is true and S_2 is true

$S_1 \vee S_2$ is true iff S_1 is true or S_2 is true

$S_1 \Rightarrow S_2 \equiv \neg S_1 \vee S_2$ is true iff S_1 is false or S_2 is true

$S_1 \Leftrightarrow S_2$ is true iff $S_1 \Rightarrow S_2$ is true and $S_2 \Rightarrow S_1$ is true

- **Simple recursive process evaluates any sentence**
  - E.g.,

$\neg S_1 \wedge (S_2 \vee S_3) = \mathit{true} \wedge (\mathit{true} \vee \mathit{false}) = \mathit{true} \wedge \mathit{true} = \mathit{true}$

Truth tables for connectives

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

Logical equivalence

- **Two sentences are logically equivalent iff they are both true in the same models: $\alpha \equiv \beta$ iff $\alpha \models \beta$ and $\beta \models \alpha$**

$(\alpha \land \beta)$	$\equiv$	$(\beta \land \alpha)$	commutativity of $\land$
$(\alpha \lor \beta)$	$\equiv$	$(\beta \lor \alpha)$	commutativity of $\lor$
$((\alpha \land \beta) \land \gamma)$	$\equiv$	$(\alpha \land (\beta \land \gamma))$	associativity of $\land$
$((\alpha \lor \beta) \lor \gamma)$	$\equiv$	$(\alpha \lor (\beta \lor \gamma))$	associativity of $\lor$
$\neg(\neg\alpha)$	$\equiv$	α	double-negation elimination
$(\alpha \Rightarrow \beta)$	$\equiv$	$(\neg\beta \Rightarrow \neg\alpha)$	contraposition
$(\alpha \Rightarrow \beta)$	$\equiv$	$(\neg\alpha \lor \beta)$	implication elimination
$(\alpha \Leftrightarrow \beta)$	$\equiv$	$((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha))$	biconditional elimination
$\neg(\alpha \land \beta)$	$\equiv$	$(\neg\alpha \lor \neg\beta)$	de Morgan
$\neg(\alpha \lor \beta)$	$\equiv$	$(\neg\alpha \land \neg\beta)$	de Morgan
$(\alpha \land (\beta \lor \gamma))$	$\equiv$	$((\alpha \land \beta) \lor (\alpha \land \gamma))$	distributivity of $\land$ over $\lor$
$(\alpha \lor (\beta \land \gamma))$	$\equiv$	$((\alpha \lor \beta) \land (\alpha \lor \gamma))$	distributivity of $\lor$ over $\land$

Validity and Satisfiability

- **A sentence is valid if it is true in all models**
  - E.g., *True*, $A \lor \neg A$, $A \Rightarrow A$
- **Validity is connected to inference via the Deduction Theorem**
  - $\alpha \models \beta$ iff $\alpha \Rightarrow \beta$ is valid
- **A sentence is satisfiable if it is true in some model**
  - E.g., $A \lor B$, C
- **A sentence is unsatisfiable if it is true in no model**
  - E.g., $A \land \neg A$
- **Satisfiability and validity are connected**
  - α is valid iff $\neg\alpha$ is unsatisfiable
  - α is satisfiable iff $\neg\alpha$ is not valid
  - $\alpha \models \beta$ iff $\alpha \land \neg\beta$ is unsatisfiable

Validity: Example proof

- **Prove that** $(A \wedge (A \Rightarrow B)) \Rightarrow B$ **is valid**
  - $(A \wedge (\neg A \vee B)) \Rightarrow B$
  - $((A \wedge \neg A) \vee (A \wedge B)) \Rightarrow B$
  - $(A \wedge B) \Rightarrow B$
  - $\neg(A \wedge B) \vee B$
  - $\neg A \vee \neg B \vee B$
  - True

MONASH University
Information Technology

Validity and Satisfiability: Example proofs

- **Prove that α is valid iff ¬α is unsatisfiable**
  - If α is valid, it is true in all models ➔ there does not exist a model for which ¬α is true ➔ ¬α is unsatisfiable
  - If ¬α is unsatisfiable ➔ there does not exist a model for which ¬α is true ➔ α is true in all models ➔ α is valid

- **Prove that α is satisfiable iff ¬α is not valid**
  - If α is satisfiable, it is true in some models ➔ ¬α is not true in these models ➔ ¬α is not valid
  - If ¬α is not valid➔ there exist some models for which ¬α is false ➔ α is true in these models ➔ α is satisfiable

Proof methods

- **Model checking**

  Enumerates all possible models to check that a sentence α is true in all models where KB is true

  – **Truth table enumeration**
  (2^n, where *n* is the number of symbols)

  – **Backtracking**
  (recursive enumeration of all models) with logic-related heuristics

- **Application of inference rules**

  Sound generation of new sentences from old

  – **Proof** = a sequence of inference rule applications

    > use inference rules as operators in a standard search algorithm

    > typically require transformation of sentences into a ***normal form***

Common rules of inference

Parent clauses	Resolvent	Name
P and ¬PvQ	Q	Modus Ponens
¬Q and ¬PvQ	¬P	Modus Tollens
P and Q	P or Q	And Elimination
PvQ and ¬PvQ	Q	
PvQ and ¬Pv¬Q	Qv¬Q or Pv¬P	Tautology
P and ¬P	NIL	
¬PvQ and ¬QvR	¬PvR	Chaining
PvQ and ¬PvR	QvR	

Resolution: Example

1. Eat ⇒ HaveLessMoney
2. ¬Eat ⇒ Hungry

¬HaveLessMoney⇒ Hungry

Resolvent: HaveLessMoney ∨ Hungry

Eat	HLM	Hungry	¬Eat ∨ HLM	Eat ∨ Hungry	HLM∨ Hungry
F	F	F	T	F	F
F	F	T	T	T	T
F	T	F	T	F	T
F	T	T	T	T	T
T	F	F	F	T	F
T	F	T	F	T	T
T	T	F	T	T	T
T	T	T	T	T	T

Proof as search

- **Initial state:** initial KB
- **Actions:** the inference rules applied to all the sentences that match the LHS of the rule
- **Result:** add the sentence on the RHS of a rule to the KB
- **Goal:** a state where the KB contains the sentence we are trying to prove

Monotonicity: the set of entailed sentences can only increase as information is added to the KB

Resolution

- **Resolution – an inference rule applied to clauses that yields a complete inference algorithm when coupled with any complete search algorithm**

parent clauses

$$\frac{l_1 \vee \ldots \vee l_i \vee \ldots \vee l_k \qquad m_1 \vee \ldots \vee m_j \vee \ldots \vee m_n}{l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

resolvent

  - where l_i and m_j are complementary literals ($l_i = \neg m_j$)
  - the resolvent should contain only one copy of each literal
- **Resolution is sound for propositional logic**

Conversion to conjunctive normal form

- **Every sentence in propositional logic is logically equivalent to a conjunction of clauses**
- **Converting a sentence to Conjunctive Normal Form (CNF)**
  1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
  2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$
  3. Move $\neg$ inwards by repeated application of the following equivalences:
     - double-negation: $\neg(\neg \alpha) \equiv \alpha$
     - de Morgan $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$
     - de Morgan $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
  4. Apply the distributivity law ($\wedge$ over $\vee$) and flatten

Conversion to CNF: Example

- $A \Leftrightarrow (B \vee C)$

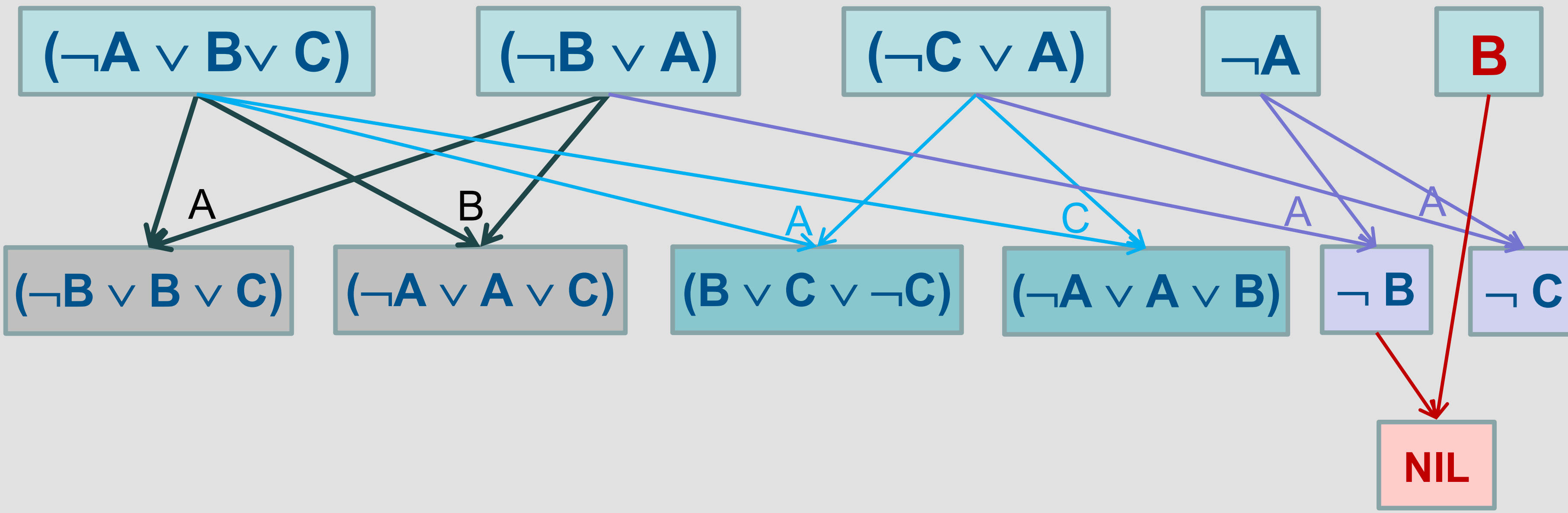
1. Eliminate $\Leftrightarrow$: $(A \Rightarrow (B \vee C)) \wedge ((B \vee C) \Rightarrow A)$
2. Eliminate $\Rightarrow$: $(\neg A \vee (B \vee C)) \wedge (\neg (B \vee C) \vee A)$
3. Move $\neg$ inwards: $(\neg A \vee (B \vee C)) \wedge ((\neg B \wedge \neg C) \vee A)$
   $(\neg A \vee B \vee C) \wedge ((\neg B \wedge \neg C) \vee A)$
4. Apply the distributivity law:
   $(\neg A \vee B \vee C) \wedge (\neg B \vee A) \wedge (\neg C \vee A)$

Resolution-refutation systems

- **Proof by refutation**
  1. Negate the goal and add the negation to the set of clauses
  2. Apply resolution to the clauses in the set of clauses until a contradiction is reached
- **Answer extraction**
  1. Build a tautology by appending the goal itself to the negation of the goal
  2. When the negated goal is contradicted, the answer resides in the goal

Resolution-refutation: Example

- $KB = (A \Leftrightarrow (B \lor C)) \land \neg A$
  $(\neg A \lor B \lor C) \land (\neg B \lor A) \land (\neg C \lor A) \land \neg A$
- **Prove:** $\alpha = \neg B$



Resolution-refutation algorithm

- **Proof by contradiction, i.e., given a goal α, show that *KB*∧¬α is unsatisfiable**

```
function PL-RESOLUTION(KB, α) returns true or false
    clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
    new ← { }
    loop do
        for each C_i, C_j in clauses do
            resolvents ← PL-RESOLVE(C_i, C_j)
            if resolvents contains the empty clause then return true
            new ← new ∪ resolvents
        if new ⊆ clauses then return false
        clauses ← clauses ∪ new
```

Soundness and Completeness

Resolution refutation is <u>sound</u> and <u>complete</u>

- **Resolution Closure *RC(S)* of a set of clauses *S* is the set of <u>all</u> clauses derivable by repeated application of the resolution rule to the clauses in *S* or their derivatives**
  - *RC(S)* is finite ➔ Resolution always terminates
- **Ground resolution theorem: if a set of clauses is unsatisfiable, then the resolution closure of those clauses contains the empty clause**
  ➔Resolution-refutation is complete for propositional logic

Resolution refutation: Example

- **KB:**
  - R1: $P \Rightarrow Q$
  - R2: $L \wedge M \Rightarrow P$
  - R3: $B \wedge L \Rightarrow M$
  - R4: $A \wedge P \Rightarrow L$
  - R5: $A \wedge B \Rightarrow L$
  - A
  - B
- **Prove Q**

- **KB:**
  - R1: $\neg P \vee Q$
  - R2: $\neg L \vee \neg M \vee P$
  - R3: $\neg B \vee \neg L \vee M$
  - R4: $\neg A \vee \neg P \vee L$
  - R5: $\neg A \vee \neg B \vee L$
  - A
  - B
- **$\neg Q$**

$\underline{\neg Q}$ R1: $\neg P \vee \underline{Q}$

$\underline{\neg P}$

R2: $\neg L \vee \neg M \vee \underline{P}$

$\neg L \vee \underline{\neg M}$

R3: $\neg B \vee \neg L \vee \underline{M}$

$\neg B \vee \underline{\neg L}$

R5: $\neg A \vee \neg B \vee \underline{L}$

$\underline{\neg A} \vee \neg B$

$\underline{A}$

$\underline{\neg B}$

$\underline{B}$

nil

Horn clauses and Definite clauses

- **Definite clause – a disjunction of literals of which <u>exactly one</u> is positive**
  – E.g., $\neg A \lor \neg B \lor C$
  – Definite clauses can be written as implications $(A \land B) \Rightarrow C$
- **Horn clause – a disjunction of literals of which <u>at most one</u> is positive**
  – All definite clauses are Horn clauses
  – Clauses with no positive literals are **goal clauses**
  – Inference with Horn clauses can be done with **forward** or **backward** chaining
  – Deciding entailment with Horn clauses can be done in time that is linear on the size of the KB

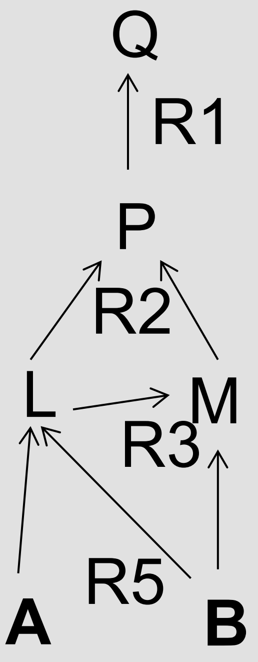
Backward chaining

- **IDEA: Work backwards from the query *q*:**
  – to prove *q* by Backward Chaining, check if *q* is known already, or prove by Backward Chaining all premises of some rule concluding *q*
- **Avoid loops: check if new subgoal is already on the goal stack**
- **Avoid repeated work: check if a new subgoal**
  1. has already been proved true, or
  2. has already failed

Backward chaining is sound and complete for Horn KBs

Backward chaining: Example

- **KB:**
  - R1: $P \Rightarrow Q$
  - R2: $L \wedge M \Rightarrow P$
  - R3: $B \wedge L \Rightarrow M$
  - R4: $A \wedge P \Rightarrow L$
  - R5: $A \wedge B \Rightarrow L$
  - A
  - B
- **Prove Q**



Forward chaining algorithm

- **IDEA: Work forwards from the facts in KB**

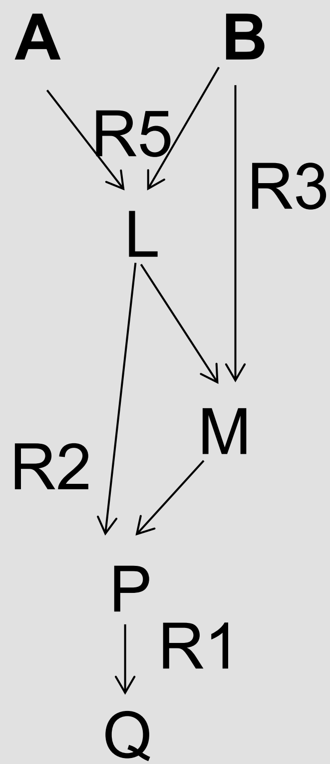
```
function PL-FC-ENTAILS?(KB, q) returns true or false
   local variables: count, a table, indexed by clause, initially the number of premises
                    inferred, a table, indexed by symbol, each entry initially false
                    agenda, a list of symbols, initially the symbols known to be true

   while agenda is not empty do
       p ← POP(agenda)
       unless inferred[p] do
           inferred[p] ← true
           for each Horn clause c in whose premise p appears do
               decrement count[c]
               if count[c] = 0 then do
                   if HEAD[c] = q then return true
                   PUSH(HEAD[c], agenda)
   return false
```

Forward chaining is sound and complete for Horn KBs

Forward chaining: Example

- **KB:**
  - R1: $P \Rightarrow Q$
  - R2: $L \wedge M \Rightarrow P$
  - R3: $B \wedge L \Rightarrow M$
  - R4: $A \wedge P \Rightarrow L$
  - R5: $A \wedge B \Rightarrow L$
  - A
  - B
- **Prove Q**



Agenda	Count	Inferred
	R1 R2 R3 R4 R5	
A B	1 2 2 2 2	

Forward versus Backward chaining

- **Forward chaining**
  - **data-driven**, automatic, unconscious processing
  - may do lots of work that is irrelevant to the goal
- **Backward chaining**
  - **goal-driven**, appropriate for problem-solving
  - complexity can be much less than linear in size of KB
- **Both are sound and complete for Horn KBs**

Pros and Cons of Propositional Logic

- ☺ **declarative**
- ☺ **allows partial/disjunctive/negated information**
- ☺ **compositional:**
  - meaning of $A \wedge B$ is derived from meaning of A and of B
- ☺ **Meaning is context-independent**
- ☹ **Propositional logic has very limited expressive power**
  - E.g., cannot say “all men are mortal“

MONASH University
Information Technology

FIT5047: Fundamentals of AI

First Order Logic

First-order logic (FOL)

- **First-order logic assumes the world contains**
  - Objects: people, houses, numbers, colors, baseball games, wars, …
  - Relations: red, round, prime, brother of, bigger than, part of, comes between, …
  - Functions: father of, best friend, one more than, plus, …

→ **FOL has increased expressive power**

First order logic – Syntax (I)

- **term – constant, variable or function**
- **atomic formula – predicate symbol and terms**
  - Example: MARRIED(*John,Mother*(*x*))
    - > Predicate symbol – MARRIED
    - > Constant – *John* or *A*
    - > Function – *Mother* or *f*
    - > Variable – *x*
- **literal – atomic formula or its negation**
  E.g., ¬MARRIED(*John*,*Mother*(*x*))
- **ground literal – literal without variables**

First order logic – Syntax (II)

- **connectives**
  - disjunction – P(x) v Q(y) v W(x,y)
  - conjunction – P(x) ∧ Q(y)
  - implication – P(x) ⇒ W(x,y) ≡ ¬P(x) v W(x,y)
- **clause – disjunction of literals**
  E.g., MARRIED(*John*,*Mother*(*x*)) v MARRIED(*John*,*y*)
- **conjunctive normal form (CNF) – a conjunction of a finite set of clauses**
  E.g., (P1(x) v P2(y)) ∧ P3(x)

Well formed formulas (wffs)

Legitimate expressions in predicate calculus

1. Any conjunction of wffs is a wff (/\)
2. Any disjunction of wffs is a wff (\/)
3. If both the antecedent and the consequent are wffs, so is the implication (=>)
4. The negation of a wff is also a wff

FOL: Quantification

Quantification

predicate

constant

- Universal ($\forall$) –
  $\forall x$ [ELEPHANT(x) $\Rightarrow$ COLOUR(x,Gray)]
- Existential ($\exists$) –
  $\exists x$ WRITE(x,Computer-Chess)

variable

1. **Any conjunction of wffs is a wff**
2. **Any disjunction of wffs is a wff**
3. **If both the antecedent and the consequent are wffs, so is the implication**
4. **The negation of a wff is a wff**
5. **Any expression obtained by quantifying a wff is a wff**

Which of these expressions is a wff?

$\exists x(\forall y[P(x,y) \wedge Q(x,y)] \Rightarrow R(x))$

$\neg f(A)$

$\neg P(A, g(A, B, A))$

$f(P(A))$

$\forall P\ P(A)$

FOL: Nesting of quantifiers

- **Existential inside the scope of universal**
  E.g., $\forall s \exists c$ Eats(s,c)

- **Universal inside the scope of existential**
  E.g., $\exists c \forall s$ Eats(s,c)

FOL: Some equivalences

- **$\neg(\exists x)P(x) \equiv (\forall x)\,[\neg P(x)]$**
  There does not exist an x such that P(x) is true ≡
  For all x, P(x) is false
- **$\neg(\forall x)P(x) \equiv (\exists x)\,[\neg P(x)]$**
  It is not true that for all x P(x) is true ≡
  There exists an x, such that ¬P(x)
- **$(\forall x)[P(x) \land Q(x)] \equiv (\forall x)P(x) \land (\forall x)Q(x)$**
  For all x, P(x) and Q(x) are true ≡
  For all x, P(x) is true, and, for all x, Q(x) is true
- **$(\exists x)[P(x) \lor Q(x)] \equiv (\exists x)P(x) \lor (\exists x)Q(x)$**
  There is an x, such that, P(x) is true or Q(x) is true ≡
  There is an x, such that, P(x) is true, or
  there is an x, such that, Q(x) is true

Are these wffs equivalent?

$(\forall x)[P(x) \vee Q(x)]$ **$(\forall x)P(x) \vee (\forall x)Q(x)$**

Rules of inference: Example

- **Modus Ponens:**
  [P and P$\Rightarrow$Q] ➔ Q
- **Modus Tollens:**
  [$\neg$Q and P$\Rightarrow$Q] ➔ $\neg$P
- **Universal Specialization**
  $\forall$x W(x) ➔ W(A), where A is a constant
- **Example: Universal Specialization + Modus Ponens**
  1. $\forall$x [DOG(x) $\Rightarrow$ BARKS(x)]
  2. DOG(FIDO)

  From 1 and 2: BARKS(FIDO)

General inference: Resolution refutation

- **Resolution**
  - Unification
    - > Substitution
  - Converting wffs to clauses

Substitution

- **Substitution is a set of ordered pairs $s=\{v_1|t_1, v_2|t_2, \ldots, v_n|t_n\}$ where $v_i|t_i$ means term t_i substitutes variable v_i**
  - Example:
    P(x,y) {x|A, y|B} ➔P(A,B)

- **Semantics: all parts are applied *simultaneously***
  - Example:
    P(w, y, g(z), x) {x|g(y), y|h(z), z|x} ➔P(w, h(z), g(x), g(y))

Composition of substitutions

- **$s_i s_k$ – composition of two substitutions**
  - Apply s_k to the terms of s_i
  - Add any pairs of s_k without variables in s_i

  Example: term variable

  s1 = {z|g(x,y)} s2 = {x|A, y|B, w|C, z|D}

  s1s2 = {z|g(x,y)}{x|A, y|B, w|C, z|D}={z|g(A,B), x|A, y|B, w|C}

  s2s1 = {x|A, y|B, w|C, z|D} {z|g(x,y)} = {x|A, y|B, w|C, z|D}
- **Properties of compositions of substitutions**
  - L(s1s2) = (Ls1)s2
  - Associative: (s1s2)s3 = s1(s2s3)
  - NOT commutative: s1s2 $\neq$ s2s1

Unification

- **Unification is a process that finds substitutions of terms for variables, such that two expressions are identical**
- **A set $\{E_i\}$ of expressions is unifiable, if there exists a substitution *s* such that $E_1s = E_2s = \ldots$**
  In this case, *s* is a unifier of the set $\{E_i\}$
- ***mgu* (most general unifier) – the mgu *g* of $\{E_i\}$ has the property that if *s* is a unifier of $\{E_i\}$, yielding $\{E_i\}s$, then there exists a substitution *s'* such that $\{E_i\}s = \{E_i\}gs'$**
- **Example:**
  s={x|A, y|B} unifies P(x,f(y)) with P(x,f(B))
  but the mgu is {y|B}

Algorithm unify (list-structured expressions)

Algorithm Unify(E1,E2)

1. **If either E1 or E2 is a variable or symbol, then interchange E1 and E2 if necessary, so that E1 is a variable or symbol**
   a. **If** E1 and E2 are identical **then** return **{ }** // no substitution
   b. **If** E1 is a variable **do**
      i. **If** E1 occurs in E2 **then** return **FAIL**
      ii. return **{**E1|E2**}** // E2 may be a compound expression
   c. **If** E2 is a variable **then** return **{**E2|E1**}**
   d. return **FAIL**
2. **F1 ← the first element of E1, T1 ← rest of E1**
3. **F2 ← the first element of E2, T2 ← rest of E2**
4. **Z1 ← Unify(F1,F2)**
5. **If Z1 = FAIL, then return FAIL**
6. **G1← result of applying Z1 to T1; G2 ← result of applying Z1 to T2**
7. **Z2 ← Unify(G1,G2)**
8. **If Z2 = FAIL, then return FAIL**
9. **return the composition of Z1 and Z2**

predicate, function, negation, constant

Unification example: Unify(P(y,g(y)),P(z,g(x)))

(P y (g y)) (P z (g x))

4. Z1 ← Unify(P,P)

1.a P and P are identical ➔ return NIL

7. Z2 ← Unify((y (g y)) , (z (g x)))

4. Z1 ← Unify(y,z): **1.b.ii return {y|z}**

6. G1 ← ((g y)){y|z} ➔((g z)), G2 ← ((g x)){y|z} ➔((g x))

7. Z2 ← Unify(((g z)) , ((g x))):

    4. Z1 ← Unify((g z), (g x))

        4. Z1 ← Unify(g,g)

            1.a g and g are identical ➔ return NIL

        7. Z2 ← Unify((z),(x))

            4. Z1 ← Unify(z,x): **1.b.ii return {z|x}**

9. return composition of {y|z}{z|x} = {y|x,z|x}

Converting wffs into clauses

1. **Eliminate implication symbols**
2. **Reduce scopes of negation symbols**
3. **Standardize variables (for each quantifier)**
4. **Eliminate existential quantifiers (*skolemize*)**
5. **Move all universal quantifiers to the front**
6. **Put result in conjunctive normal form (CNF)**
7. **Eliminate universal quantifiers**
8. **Eliminate $\land$ symbols**
9. **Rename variables (standardize variables <u>apart</u> for each clause)**

Converting wffs into Clauses: Example 1

$\forall x$ [CanRead(x) $\Rightarrow$ Intelligent(x)]

1. **Eliminate $\Rightarrow$: $\forall x$ [$\neg$CanRead(x) v Intelligent(x)]**

7. **Eliminate $\forall$: $\neg$CanRead(x) v Intelligent(x)**

Converting wffs into Clauses: Example 2

$\forall x\, [\neg\, (\forall y)\, [\, P(x,y) \Rightarrow Q(x,y)]\,]$

1. Eliminate $\Rightarrow$: $\forall x\, [\neg\, (\forall y)\, [\, \neg P(x,y) \vee Q(x,y)]\,]$
2. Reduce scope of $\neg$: $\forall x\, [\, \exists y\, \neg\, [\, \neg P(x,y) \vee Q(x,y)]\,]$
   $\forall x\ \ \exists y\ \ [\, P(x,y) \wedge \neg Q(x,y)\,]$
4. Eliminate $\exists$: $\forall x\, [\, P(x,g(x)) \wedge \neg Q(x,g(x))\,]$
7. Eliminate $\forall$: $P(x,g(x)) \wedge \neg Q(x,g(x))$
8. Eliminate $\wedge$ symbols: $\{\, P(x,g(x)),\ \neg Q(x,g(x))\, \}$
9. Standardize variables apart: $\{\, P(x_1,g(x_1)),\ \neg Q(x_2,g(x_2))\, \}$

General resolution

- **Let the prospective parent clauses be $\{L_j\}$ and $\{M_i\}$ (<u>with variables standardized apart</u>)**
- **Suppose that $\{l_j\}$ is a subset of $\{L_j\}$ and that $\{m_i\}$ is a subset of $\{M_i\}$ such that a most general unifier *s* exists for the sets $\{l_j\}$ and $\{\neg m_i\}$**
- **The clauses $\{L_j\}$ and $\{M_i\}$ *resolve* and the new clause $\{\{L_j\}-\{l_j\}\}s \cup \{\{M_i\}-\{m_i\}\}s$ is a *resolvent* of the two clauses**

General resolution: Example

1. **Everyone who can read is literate**

   $\forall x\, [\, CANREAD(x) \Rightarrow LITERATE(x)\,]$

2. **Whoever goes to school can read**

   $\forall x\, [\, GOSCHOOL(x) \Rightarrow CANREAD(x)\,]$

$\neg GOSCHOOL(x_1) \vee \underline{CANREAD(x_1)}$ $\quad$ $\underline{\neg CANREAD(x_2)} \vee LITERATE(x_2)$

mgu $s=\{x_2|x_1\}$

$\neg GOSCHOOL(x_1) \vee LITERATE(x_1)$

Resolution-refutation: Example (I)

1. **If a unit is easy, there are some students who are enrolled in it who are happy**

   $\forall u\ [\ EASY(u) \Rightarrow \exists s\ [\ ENROLLED(s,u) \wedge HAPPY(s)\ ]\ ]$

2. **If a unit has a final exam, no students that are enrolled in it are happy**

   $\forall u\ [\ HASFINAL(u) \Rightarrow \neg \exists s\ [\ ENROLLED(s,u) \wedge HAPPY(s)\ ]\ ]$

   or

   $\forall u\ [\ HASFINAL(u) \Rightarrow \forall s\ [\ ENROLLED(s,u) \Rightarrow \neg\ HAPPY(s)\ ]\ ]$

3. **Prove that if a unit has a final exam, the unit is not easy**

   $\forall u\ [\ HASFINAL(u) \Rightarrow \neg\ EASY(u)\ ]$

Resolution-refutation: Example (II)

Converting to clauses

1. $\forall u$ [EASY(u) $\Rightarrow \exists s$ [ENROLLED(s,u) $\wedge$ HAPPY(s)]]

Eliminate $\Rightarrow$: $\forall u$ [$\neg$ EASY(u) v $\exists s$ [ENROLLED(s,u) $\wedge$ HAPPY(s)]]

Eliminate $\exists$: $\forall u$ [$\neg$ EASY(u) v [ENROLLED(g(u),u) $\wedge$ Happy(g(u))]]

Turn into CNF: $\forall u$ [[$\neg$ EASY(u) v ENROLLED(g(u),u)] $\wedge$
[$\neg$ EASY(u) v HAPPY(g(u))]]

Eliminate $\forall$: [$\neg$ EASY(u) v ENROLLED(g(u),u)] $\wedge$
[$\neg$ EASY(u) v HAPPY(g(u))]

Eliminate $\wedge$ and standardize variables apart:

1.1 $\neg$ EASY(u_1) v ENROLLED(g(u_1),u_1)

1.2 $\neg$ EASY(u_2) v HAPPY(g(u_2))

Resolution-refutation: Example (III)

2. $\forall u$ [HASFINAL(u) $\Rightarrow \neg \exists s$ [ENROLLED(s,u) $\wedge$ HAPPY(s)]]

Eliminate $\Rightarrow$:

$\forall u$ [$\neg$ HASFINAL(u) v $\neg \exists s$ [ENROLLED(s,u) $\wedge$ HAPPY(s)]]

Reduce scope of $\neg$:

$\forall u$ [$\neg$ HASFINAL(u) v $\forall s$ $\neg$[ENROLLED(s,u) $\wedge$ HAPPY(s)]]

$\forall u$ [$\neg$ HASFINAL(u) v $\forall s$[$\neg$ENROLLED(s,u) v $\neg$HAPPY(s)]]

Move $\forall$ to the front:

$\forall u \forall s$ [$\neg$ HASFINAL(u) v $\neg$ ENROLLED(s,u) v $\neg$HAPPY(s)]

Eliminate $\forall$: $\neg$ HASFINAL(u) v $\neg$ENROLLED(s,u) v $\neg$HAPPY(s)

Standardize variables apart:

$\neg$ HASFINAL(u_3) v $\neg$ENROLLED(s_3,u_3) v $\neg$HAPPY(s_3)

Resolution-refutation: Example (IV)

Using resolution to prove statement 3

Negate the goal:

3'. $\neg\forall u$ [HASFINAL(u) $\Rightarrow \neg$ EASY(u)]

Eliminate $\Rightarrow$: $\neg\forall u$ [$\neg$ HASFINAL(u) v $\neg$ EASY(u)]

Reduce scope of $\neg$: $\exists u$ [HASFINAL(u) $\wedge$ EASY(u)]

Eliminate $\exists$: HASFINAL(A) $\wedge$ EASY(A)

Eliminate Λ: 3'.1 HASFINAL(A)

3'.2 EASY(A)

Resolution-refutation: Example (V)

3'.1 $\underline{HASFINAL(A)}$ 2. $\underline{\neg HASFINAL(u_3)} \vee \neg ENROLLED(s_3,u_3) \vee \neg HAPPY(s_3)$

$\{u_3|A\}$

4. $\neg ENROLLED(s_3,A) \vee \underline{\neg HAPPY(s_3)}$ 1.2 $\neg EASY(u_2) \vee \underline{HAPPY(g(u_2))}$

$\{s_3|g(u_2)\}$

1.1 $\neg EASY(u_1) \vee \underline{ENROLLED(g(u_1),u_1)}$ 5. $\underline{\neg ENROLLED(g(u_2),A)} \vee \neg EASY(u_2)$

$\{u_2|u_1\}\{u_1|A\} = \{u_2|A,u_1|A\}$

6. $\underline{\neg EASY(A)}$ 3'.2 $\underline{EASY(A)}$

NIL

Resolution-refutation strategies

- **Unit preference: prefer resolutions where one of the sentences is a unit clause (single literal)**
  - **Unit resolution:** every resolution ***must*** involve a unit clause
- **Set of support: every resolution step should involve at least one element of a special set of clauses, the set of support. The resolvent is added to the set of support**
  - E.g., set of support = $\{\neg Q\}$
- **Input resolution: every resolution combines one of the original input sentences (from the KB or query) with another sentence**
- **Subsumption: eliminates all sentences subsumed by (more specific than) an existing sentence in the KB**

Uses of FOL in AI

- **Theorem proving**
  1. ON(C,A)
  2. ONTable(A), ONTable(B)
  3. CLEAR(C), CLEAR(B)
  4. $\forall x$ [CLEAR(x) $\Rightarrow \neg \exists y$ ON(y,x)]
  5. **Goal:** Prove $\neg\exists y$ ON(y,C)
- **Question answering**
  1. MANAGER(Purchasing-dept., John-Jones)
  2. WORKSIN(Purchasing-dept., Joe-Smith)
  3. $\forall x \forall y \forall z$ [WORKSIN(x,y) $\wedge$ MANAGER(x,z)] $\Rightarrow$ BOSSOF(y,z)
  4. **Goal:** Who is the boss of Joe Smith?

     $\exists x$ BOSSOF(Joe-Smith,x)
- **Planning**

Theorem proving: Example

1. ON(C,A)
2. (a) ONTABLE(A), (b) ONTABLE(B)
3. (a) CLEAR(C), (b) CLEAR(B)
4. $\forall x\ [\text{CLEAR}(x) \Rightarrow \neg\ \exists y\ \text{ON}(y,x)\ ]$

   $\neg\ \text{CLEAR}(x_1) \vee \neg\ \text{ON}(y,x_1)$
5. **Goal:** Prove $\neg\exists y\ \text{ON}(y,C)$

   Negate goal: $\exists y\ \text{ON}(y,C)$

   ON(M,C)

4. $\neg$ CLEAR(x_1) $\vee$ $\neg$ ON(y,x_1) **5. ON(M,C)**

$\{y|M,\ x_1|C\}$

6. $\neg$ CLEAR(C)

NIL

Question answering: Example

3. $\neg$ WORKSIN(x_1,y) v $\neg$ MANAGER(x_1,z) v BOSSOF(y,z)

4. $\neg$ BOSSOF(Joe-Smith,x_2) v BOSSOF(Joe-Smith,x_2)

{y|Joe-Smith, x_2|z}

5. $\neg$ WORKSIN(x_1,Joe-Smith) v $\neg$ MANAGER(x_1,z) BOSSOF(Joe-Smith,z)

1. MANAGER(Purchasing-dept., John-Jones)

{x_1|Purchasing-dept, z|John-Jones}

6. $\neg$ WORKSIN(Purchasing-dept,Joe-Smith) BOSSOF(Joe-Smith,John-Jones)

2. WORKSIN(Purchasing-dept,Joe-Smith)

NIL

Reading

- **Russell, S. and Norvig, P. (2010), *Artificial Intelligence – A Modern Approach* (3rd edition), Prentice Hall**
  - Chapter 6, Sections 6.1-6.5
  - Chapter 8
  - Chapter 9, Sections 9.1, 9.2 and 9.5

Next Lecture Topic

- **Lecture Topic 5**
  - Probability