MONASH University
Information Technology

FIT3080 – Intelligent Systems

Knowledge Representation
Chapters 7-9

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

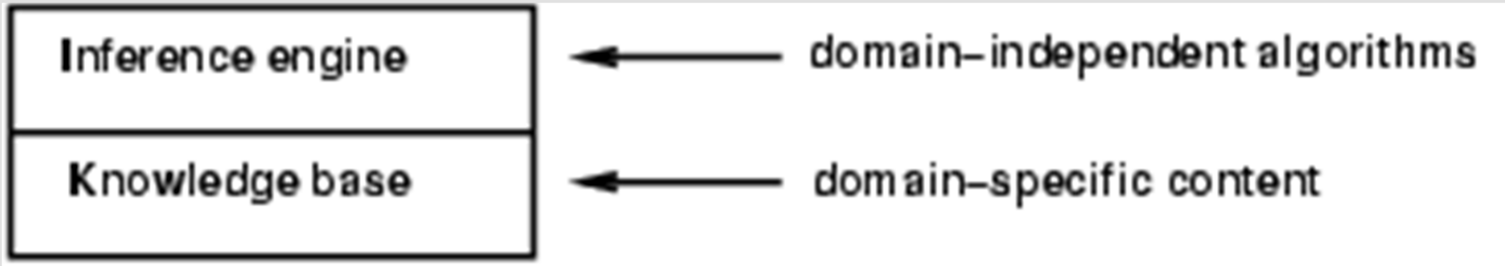
Assumptions about the Environment

- **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

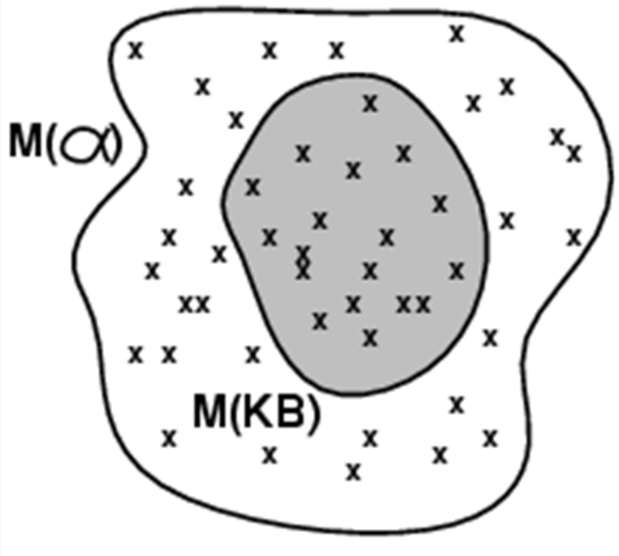
- **Logics are formal languages for representing information such that conclusions can be drawn**
- **Syntax defines the sentences in the language**
- **Semantics define the meaning of sentences,**
  - i.e., the truth of a sentence in each **possible world**
- **E.g., the language of arithmetic**
  - $x+2 \geq y$ 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 $x = 0$, $y = 6$

Entailment

- **Entailment means that one thing follows logically from another:**
  $KB \models \alpha$
  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 ╞ 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

The connection between logical reasoning processes and real environments

- **How do we know that KB is true in the real world?**
  - The agent's sensors create a connection, 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

FIT3080 – Intelligent Systems

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 an arbitrary 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: α ≡ β iff α ⊨ β and β ⊨ α**

$(\alpha \land \beta)$	$\equiv$	$(\beta \land \alpha)$	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 models**
  - 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

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	

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**

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

  – where l_i and m_j are complementary literals ($l_i = \neg m_j$)

  – the resultant clause 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 distributivity law ($\wedge$ over $\vee$) and flatten

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 Algorithm

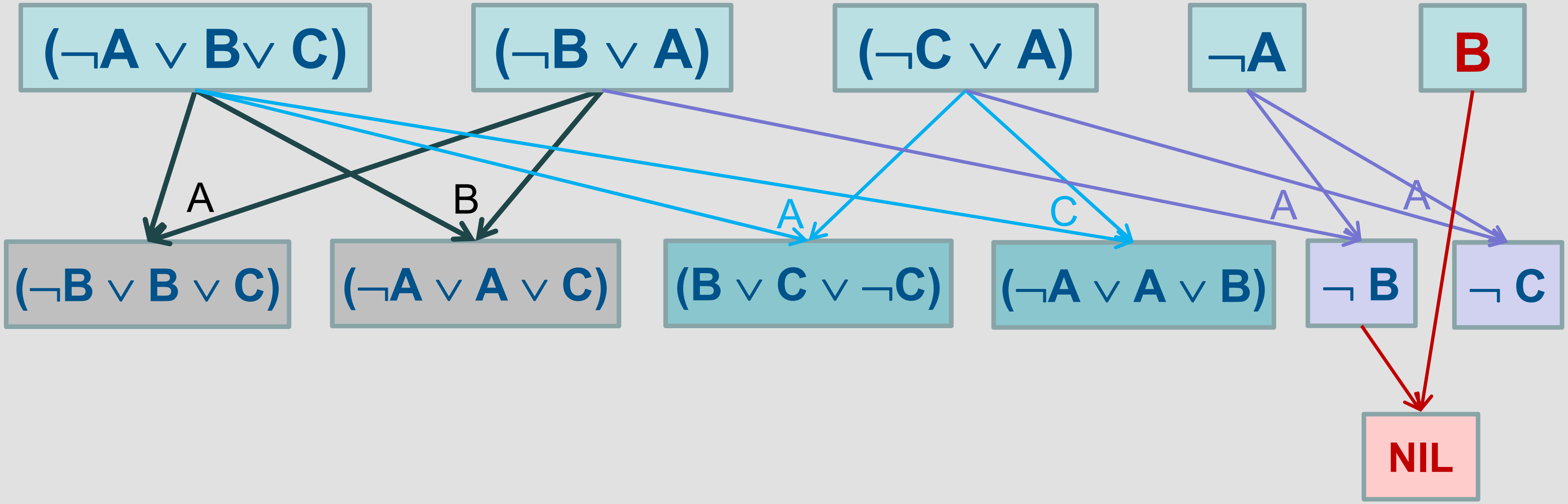
- **Proof by contradiction, i.e., given a goal α, show that *KB*∧¬α is unsatisfiable**

function PL-RESOLUTION(KB, α) **returns** *true* or *false*

    clauses ← the set of clauses in the CNF representation of $KB \land \neg\alpha$
    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*

Resolution-refutation – Example

- ***KB*** **= (A ⇔ (B∨ C)) ∧ ¬A**
  (¬A ∨ B∨ C) ∧ (¬B ∨ A) ∧ (¬C ∨ A) ∧ ¬A
- **Prove: α = ¬B**



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

Horn Clauses and Definite Clauses

- **Definite clause – a disjunction of literals of which exactly one is positive**
  - E.g., $\neg A \vee \neg B \vee C$
  - Definite clauses can be written as implications $(A \wedge B) \Rightarrow C$
- **Horn clause – a disjunction of literals of which at most one 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

Forward Chaining Algorithm

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

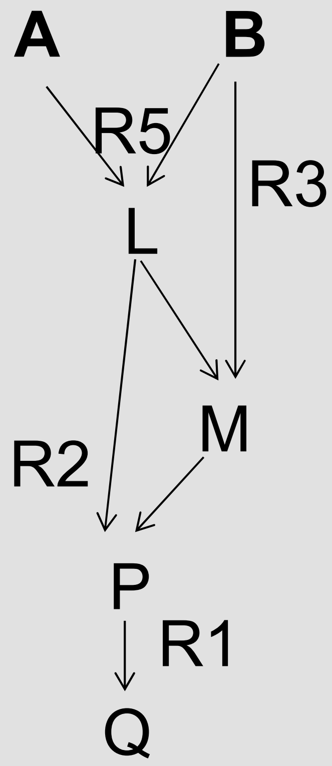
```
function PL-FC-ENTAILS?(KB, q) returns true or false
   local variables: count, a table, indexed by clause, initially the number of premises
                    inferred, a table, indexed by symbol, each entry initially false
                    agenda, a list of symbols, initially the symbols known 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	
B	1 2 2 1 1	A
L		
M		

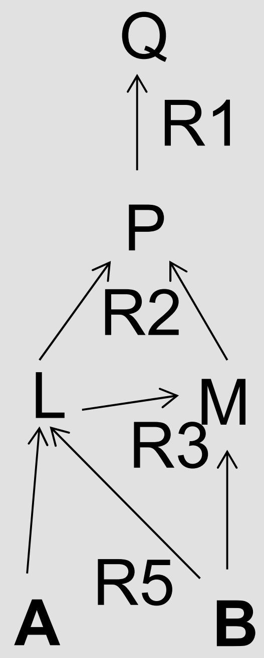
Backward Chaining

- **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**



MONASH University Information Technology

Forward versus Backward Chaining

- **Forward Chaining**
  - **data-driven**, automatic, unconscious processing
    - > e.g., object recognition, routine decisions
  - may do lots of work that is irrelevant to the goal
- **Backward Chaining**
  - **goal-driven**, appropriate for problem-solving
    - > e.g., How do I get into a PhD program?
  - 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

FIT3080 – Intelligent Systems

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*
    - > Function – *Mother*
    - > Variable – *x*
- **literal – atomic formula or negation of an atomic formula**
  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 – a conjunction of a finite set of clauses**
  E.g., (P1(x)vP2(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**

First Order Logic – Quantification

Quantification

- Universal ($\forall$) –
  $\forall x$ [ELEPHANT(x) $\Rightarrow$ COLOUR(x,GRAY)]
- Existential ($\exists$) –
  $\exists x$ WRITE(x,COMPUTER-CHESS)

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**

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)
- **$(\forall x)[P(x) \wedge Q(x)] \equiv (\forall x)P(x) \wedge (\forall y)Q(y)$**
  For all x, P(x) and Q(x) are true ≡
  For all x, P(x) is true, and for all y Q(y) is true
- **$(\exists x)[P(x) \vee Q(x)] \equiv (\exists x)P(x) \vee (\exists y)Q(y)$**
  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 a y, such that Q(y) is true

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 into clauses

Substitution

- **Substitution is a set of ordered pairs $s=\{t_1/v_1, t_2/v_2, \ldots, t_n /v_n\}$ where t_i/v_i means that t_i is substituted for v_i throughout**
  – Example:
    $\{A/x, B/y\}$ P(x,y) ➔P(A,B)
- **Semantics: all elements are applied *simultaneously***
  – Example:
    {g(y)/x, h(z)/y, x/z} P(w, y, g(z), x)➔P(w, h(z), g(x), g(y))

Composition of Substitutions

- **s1s2 – composition of two substitutions**
  - Apply s2 to the terms of s1
  - Add any pairs of s2 having variables not in s1

  Example: term

  variable

  s1 = {g(x,y)/z}

  s2 = {A/x, B/y, C/w, D/z}

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

  s2s1 = {A/x, B/y, C/w, D/z}

- **Properties of compositions of substitutions**
  - L(s1s2) = (Ls1)s2
  - Associative: (s1s2)s3 = s1(s2s3)
  - NOT commutative: s1s2 $\neq$ s2s1

Unification

- **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 $\{E_i\}$
- ***mgu* (most general unifier) – the mgu g of $\{E_i\}$ has the property that if s is any unifier of $\{E_i\}$, yielding $\{E_i\}s$, then there exists a substitution s' such that $\{E_i\}s = \{E_i\}gs'$**
- **Example:**
  s={A/x, B/y} unifies P(x,f(y)) with P(x,f(B))
  but the mgu is {B/y}

Algorithm Unify

Algorithm Unify(E1,E2)

1. **If either E1 or E2 is a symbol (predicate, function, constant, negation or variable), then interchange E1 and E2, so that E1 is a 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 **{**E2/E1**}**
   c. **If** E2 is a variable **then** return **{**E1/E2**}**
   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**

Unification Example: Unify(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 {z/y}

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

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

Converting wffs into Clauses

1. **Eliminate implication symbols**
2. **Reduce scopes of negation symbols**
3. **Standardize variables**
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 $\wedge$ symbols**
9. **Rename variables (standardize variables apart)**

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 $\neg$:** $\forall x\, [\, \exists y\, \neg\, [\, \neg P(x,y) \vee Q(x,y)]\,]$
   $\forall x \quad \exists y \quad [\, P(x,y) \wedge \neg Q(x,y)\,]$
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}* (with variables standardized apart)**
- **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 union of 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 U {{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**

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

$\neg GoSchool(x_1) \vee \underline{CanRead(x_1)}$ $\quad\quad$ $\neg \underline{CanRead(x_2)} \vee Literate(x_2)$

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

$\neg GoSchool(x_1) \vee Literate(x_1)$

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**

   $\forall u\ [\ HasFinal(u) \Rightarrow \neg \exists s\ [\ Enrolled(s,u) \wedge Happy(s)\ ]\ ]$

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 $\Rightarrow$: $\forall u\ [\ \neg\ Easy(u) \vee \exists s\ [\ Enrolled(s,u) \wedge Happy(s)\ ]\ ]$

Eliminate $\exists$: $\forall u\ [\ \neg\ Easy(u) \vee [\ Enrolled(g(u),u) \wedge Happy(g(u))\ ]\ ]$

Turn into CNF: $\forall u\ [\ [\ \neg\ Easy(u) \vee Enrolled(g(u),u)\ ] \wedge$
$[\ \neg\ Easy(u) \vee Happy(g(u))\ ]\ ]$

Eliminate $\forall$: $[\ \neg\ Easy(u) \vee Enrolled(g(u),u)\ ] \wedge$
$[\ \neg\ Easy(u) \vee Happy(g(u))\ ]$

Eliminate $\wedge$ and rename variables:

1.1 $\neg\ Easy(u_1) \vee Enrolled(g(u_1),u_1)$

1.2 $\neg\ Easy(u_2) \vee Happy(g(u_2))$

Resolution-refutation – Example (III)

2. $\forall u\,[\, HasFinal(u) \Rightarrow \neg\, \exists s\,[\, Enrolled(s,u) \wedge Happy(s)\,]\,]$

Eliminate $\Rightarrow$: $\forall u\,[\, \neg\, HasFinal(u) \vee$

$\neg\, \exists s\,[\, Enrolled(s,u) \wedge Happy(s)\,]\,]$

Reduce scope of $\neg$:

$\forall u\,[\neg\, HasFinal(u) \vee \forall s\, \neg[\, Enrolled(s,u) \wedge Happy(s)\,]\,]$

$\forall u\,[\neg\, HasFinal(u) \vee \forall s[\, \neg Enrolled(s,u) \vee \neg Happy(s)\,]\,]$

Move $\forall$ to the front: $\forall u\, \forall s\,[\, \neg\, HasFinal(u) \vee$

$\neg\, Enrolled(s,u) \vee \neg Happy(s)\,]$

Eliminate $\forall$ and rename variables:

$\neg\, HasFinal(u_3) \vee \neg Enrolled(s_3,u_3) \vee \neg Happy(s_3)$

Resolution-refutation – Example (IV)

Using resolution to prove statement 3

Negate the goal:

3'. $\neg \forall u\, [\, HasFinal(u) \Rightarrow \neg\, Easy(u)\,]$

Eliminate $\Rightarrow$: $\neg \forall u\, [\neg\, HasFinal(u) \vee \neg\, Easy(u)\,]$

Reduce scope of $\neg$: $\exists u\, [\, HasFinal(u) \wedge Easy(u)\,]$

Eliminate $\exists$: $HasFinal(A) \wedge Easy(A)$

Eliminate $\wedge$: 3'.1 $HasFinal(A)$

3'.2 $Easy(A)$

Resolution-refutation – Example (V)

3'.1 $\underline{HasFinal(A)}$

2. $\underline{\neg HasFinal(u_3)} \vee \neg Enrolled(s_3,u_3) \vee \neg Happy(s_3)$

$\{A/u_3\}$

4. $\neg Enrolled(s_3,A) \vee \underline{\neg Happy(s_3)}$

1.2 $\neg Easy(u_2) \vee \underline{Happy(g(u_2))}$

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

1.1 $\neg Easy(u_1) \vee \underline{Enrolled(g(u_1),u_1)}$

5. $\underline{\neg Enrolled(g(u_2),A)} \vee \neg Easy(u_2)$

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

6. $\underline{\neg Easy(A)}$

3'.2 $\underline{Easy(A)}$

NIL

Resolution 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 the query) with another sentence**
- **Subsumption: eliminates all sentences that are 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)]

  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)

  Goal: Who is the boss of Joe Smith?

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

Reading

- **Russell, S. and Norvig, P. (2010),** ***Artificial Intelligence – A Modern Approach*** **(3rd edition), Prentice Hall**
  - Chapter 7, Sections 7.1-7.5
  - Chapter 8
  - Chapter 9, Sections 9.1, 9.2 and 9.5