MONASH University
Information Technology

FIT5047 – Intelligent Systems

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   Propositional logic
  - resolution refutation systems
    - substitution
    - unification   First order logic
    - resolution

MONASH University
Information Technology

# 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

Inference engine	← domain–independent algorithms
Knowledge base	← domain–specific content
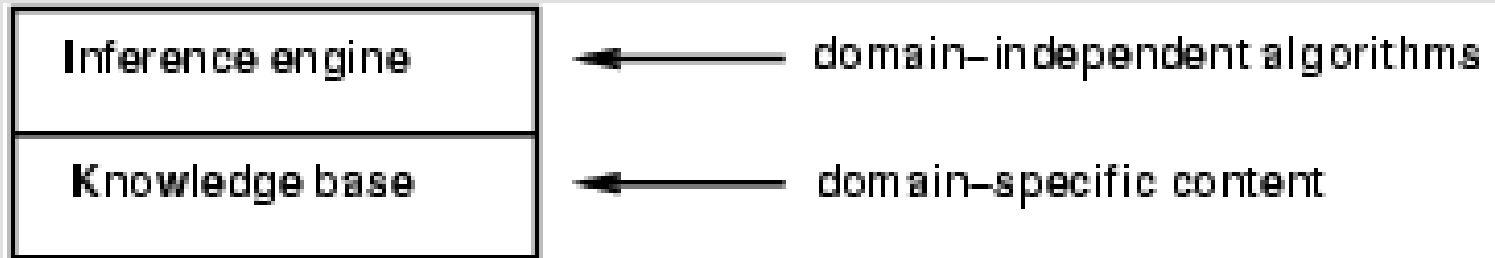
- **Knowledge base = set of sentences in a formal language**
- **Declarative approach to building an agent:**
  - **Tell** it what it needs to know
  - **Ask** it a question – answers follow by inference from the KB
- **Agents may be viewed**
  - at the **knowledge level** – what they know
  - at the **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

MONASH University
Information Technology

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 $x = 7$, $y = 1$
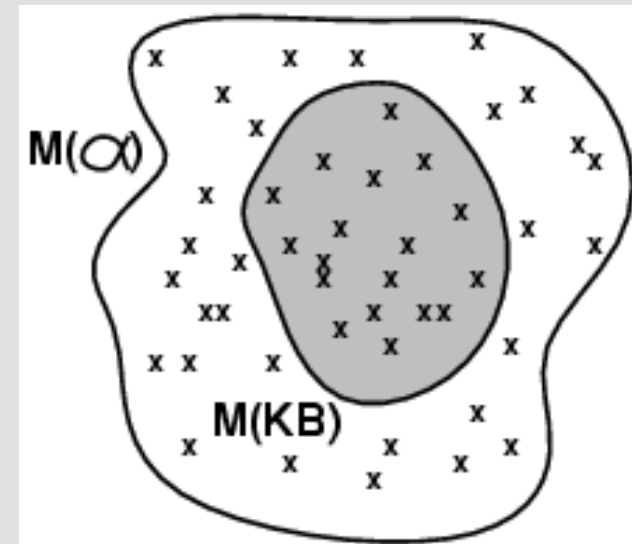   - $x+2 \geq y$ is false in a world where $x = 0$, $y = 6$

Logical Entailment

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

MONASH University
Information Technology

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 $M(KB) \subseteq M(α)$
    - > 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**

MONASH University
Information Technology

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

FIT5047 – 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

MONASH University
Information Technology

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., $\quad S_1 \qquad S_2 \qquad S_3$
  $\qquad\qquad$ false $\quad$ true $\quad$ false
- **Rules for evaluating truth with respect to a model *m*:**
  $\quad \neg S \qquad\qquad$ is true iff $\ S$ is false
  $\quad S_1 \wedge S_2 \qquad$ is true iff $\ S_1$ is true and $\ S_2$ is true
  $\quad S_1 \vee S_2 \qquad$ is true iff $\ S_1$ is true $\ $ or $\ S_2$ is true
  $\quad S_1 \Rightarrow S_2 \equiv \neg S_1 \vee S_2 \ $ is true iff $\ S_1$ is false $\ $ or $\ S_2$ is true
  $\quad S_1 \Leftrightarrow S_2 \quad$ 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.,
  $\neg S_1 \wedge (S_2 \vee S_3) = true \wedge (true \vee false) = true \wedge true = true$

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**

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

Validity and Satisfiability

- **A sentence is valid if it is true in all models**
  - E.g., *True*, $A \vee \neg A$, $A \Rightarrow A$
- **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., $A \vee B$, C
- **A sentence is unsatisfiable if it is true in no model**
  - E.g., $A \wedge \neg A$
- **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 $\alpha \wedge \neg \beta$ is unsatisfiable

Validity – Example Proof

- **Prove that** $(A \land (A \Rightarrow B)) \Rightarrow B$ **is valid**
  - $(A \land (\neg A \lor B)) \Rightarrow B$
  - $((A \land \neg A) \lor (A \land B)) \Rightarrow B$
  - $(A \land B) \Rightarrow B$
  - $\neg(A \land B) \lor B$
  - $\neg A \lor \neg B \lor B$
  - 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

MONASH University
Information Technology

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***

MONASH University
Information Technology

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	

MONASH University
Information Technology

Resolution – Example

1. Eat $\Rightarrow$ HaveLessMoney
2. $\neg$Eat $\Rightarrow$ Hungry

Resolvent: HaveLessMoney $\vee$ Hungry

Eat	HLM	Hungry	$\neg$Eat $\vee$ HLM	Eat $\vee$ Hungry	HLM$\vee$ 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**

<div align="center">

parent clauses

$$l_1 \vee \dots \vee l_i \vee \dots \vee l_k \qquad m_1 \vee \dots \vee m_j \vee \dots \vee m_n$$

$$\overline{\quad l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n \quad}$$

resolvent

</div>

- 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 distributivity law ($\wedge$ over $\vee$) and flatten

MONASH University
Information Technology

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 distributivity law:
     $(\neg A \vee B \vee C) \wedge (\neg B \vee A) \wedge (\neg C \vee A)$

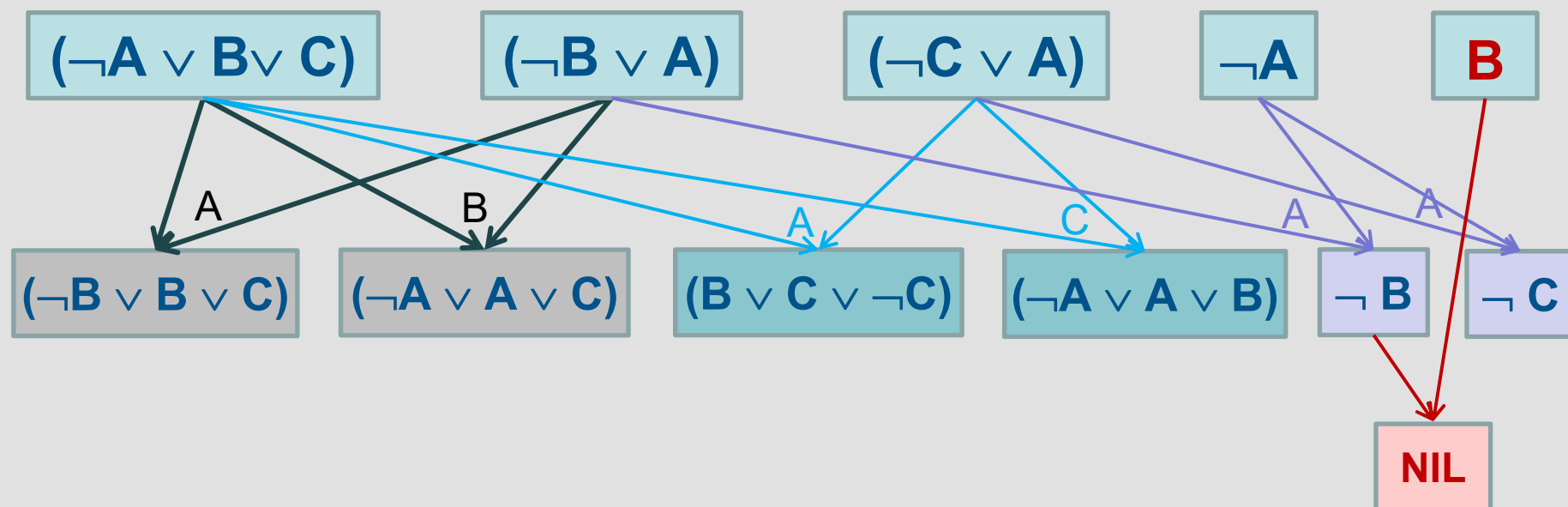# Resolution-refutation Systems

- **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

- $KB = (A \Leftrightarrow (B \vee C)) \wedge \neg A$
  $(\neg A \vee B \vee C) \wedge (\neg B \vee A) \wedge (\neg C \vee A) \wedge \neg A$

- **Prove: α = $\neg B$**

Resolution-refutation Algorithm

- **Proof by contradiction, i.e., given a goal α, show that $KB \wedge \neg\alpha$ is unsatisfiable**

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

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

    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*

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
- **Negate Q: $\neg Q$**

$\underline{\neg Q}$ 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

MONASH University
Information Technology

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
      $p \leftarrow$ POP($agenda$)
      unless $inferred[p]$ do
          $inferred[p] \leftarrow true$
          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 $true$
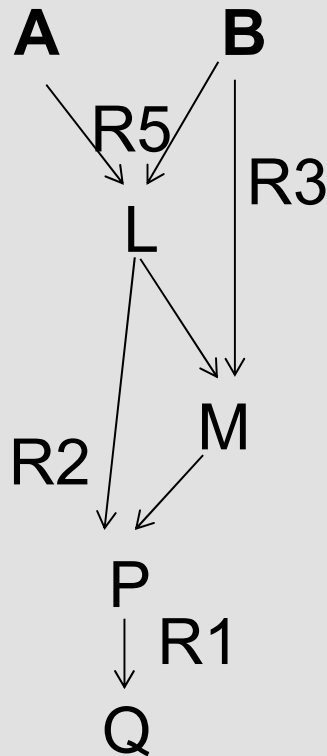   PUSH(HEAD$[c]$, $agenda$)
  return $false$

Forward chaining is sound and complete for Horn KBs

MONASH University
Information Technology

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	

L

M

MONASH University
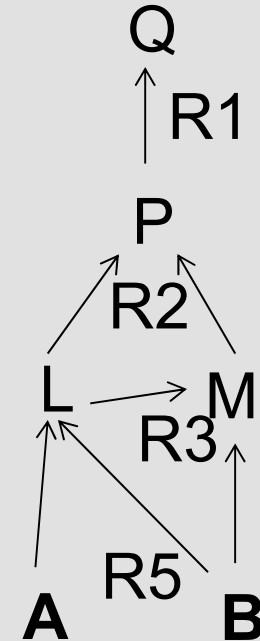Information Technology

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**

Q
R1
P
R2
L → M
R3
R5
A   **B**

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"

FIT5047 – 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**

MONASH University
Information Technology

First Order Logic – Syntax (I)

- **term – constant, variable or function**
- **atomic formula or atom – 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 negation of an atomic formula**
  E.g., ¬MARRIED(*John,Mother*(*x*))
- **ground literal – literal without variables**

MONASH University
Information Technology

First Order Logic – Syntax (II)

- **connectives**
  - disjunction – P(x) ∨ Q(y) ∨ W(x,y)
  - conjunction – P(x) ∧ Q(y)
  - implication – P(x) $\Rightarrow$ W(x,y) $\equiv$ ¬P(x) ∨ W(x,y)

- **clause – disjunction of literals**
  E.g., MARRIED(*John*,*Mother*(*x*)) ∨ MARRIED(*John,y*)

- **conjunctive normal form – a conjunction of a finite set of clauses**
  E.g., (P1(x)∨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

MONASH University
Information Technology

First Order Logic – Quantification

Quantification

- Universal ($\forall$) –
$\forall x$ [ELEPHANT(x) $\Rightarrow$ COLOUR(x,Gray)]

- Existential ($\exists$) –
$\exists x$ WRITE(x,Computer-Chess)

predicate   constant

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 $\equiv$
  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 $\equiv$
  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 $\equiv$
  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 $\equiv$
  There is an x, such that P(x) is true, or
   there is a   y, such that Q(y) is true

Are these wffs equivalent?

$(\forall x)[P(x) \lor Q(x)]$ $\qquad\qquad$ $(\forall x)P(x) \lor (\forall y)Q(y)$

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)

MONASH University
Information Technology

General Inference – Resolution Refutation

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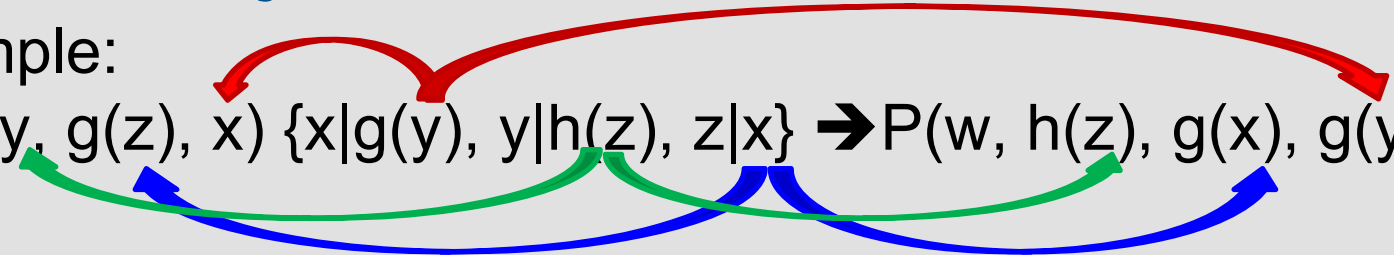 - Converting wffs to clauses

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 that term t_i substitutes variable v_i throughout
  – Example:
  P(x,y) {x|A, y|B} ➔ P(A,B)
- **Semantics: all elements 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))

MONASH University
Information Technology

Composition of Substitutions

- **$s_i s_j$ – composition of two substitutions**
  - Apply s_j to the terms of s_i
  - Add any pairs of s_j having variables not 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

MONASH University
Information Technology

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:**
  $s = \{x|A, y|B\}$ 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**

   predicate, function, negation, constant

   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**

MONASH University
Information Technology

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 {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 $\wedge$ symbols**
9. **Rename variables (standardize variables <u>apart</u> for each clause)**

MONASH University
Information Technology

Converting wffs into Clauses – Example 1

∀x [CanRead(x) ⇒ Intelligent(x)]

1. **Eliminate ⇒:** ∀x [¬CanRead(x) v Intelligent(x)]

7. **Eliminate ∀:** ¬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) v Q(x,y)]]

2. **Reduce scope of $\neg$:** $\forall$x [$\exists$y $\neg$ [$\neg$P(x,y) v 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)) }

MONASH University
Information Technology

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 $\{\{L_j\}\text{-}\{l_j\}\}s \cup \{\{M_i\}\text{-}\{m_i\}\}s$ 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) v CanRead(x_1) $\neg$CanRead(x_2) v Literate(x_2)

mgu s={x_2/x_1}

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

MONASH University
Information Technology

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)]

MONASH University
Information Technology

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))]]

   Convert to 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 standardize variables apart:

   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) 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 $\forall$ and 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) $\lor \neg$ Easy(u)]

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

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

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

               3'.2 Easy(A)

MONASH University
Information Technology

Resolution-refutation – Example (V)

3'.1 <u>HasFinal(A)</u>   2. $\neg$ <u>HasFinal(u_3)</u> v $\neg$Enrolled(s_3,u_3) v $\neg$Happy(s_3)

{u_3/A}

4. $\neg$Enrolled(s_3,A) v <u>$\neg$Happy(s_3)</u>   1.2 $\neg$ Easy(u_2) v <u>Happy($g(u_2)$)</u>

{s_3/$g(u_2)$}

1.1 $\neg$ Easy(u_1) v <u>Enrolled($g(u_1),u_1$)</u>   5. <u>$\neg$Enrolled($g(u_2)$,A)</u> v $\neg$ Easy(u_2)

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

6. $\neg$ <u>Easy(A)</u>   3'.2 <u>Easy(A)</u>

NIL
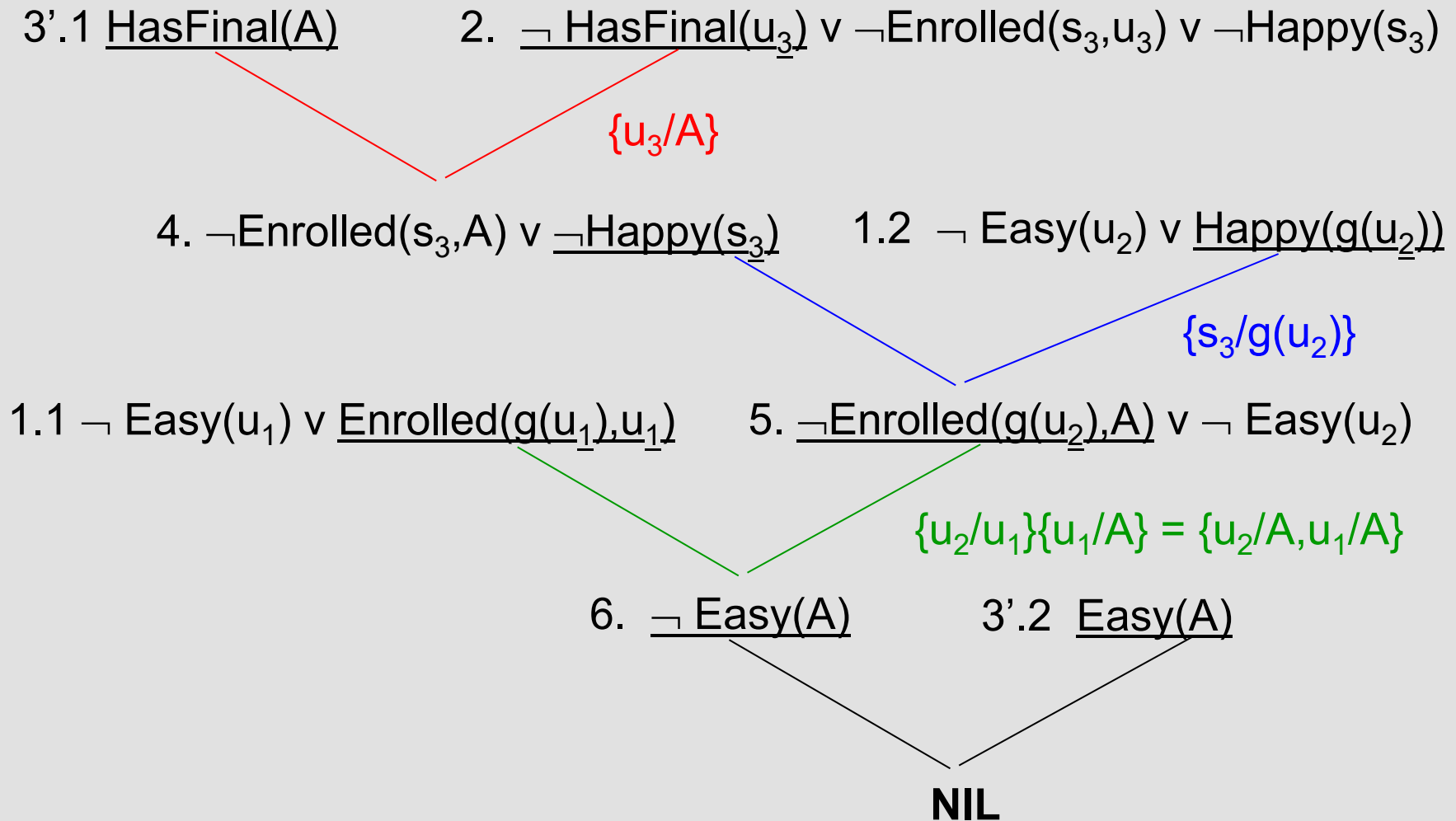
Resolution-refutation Strategies

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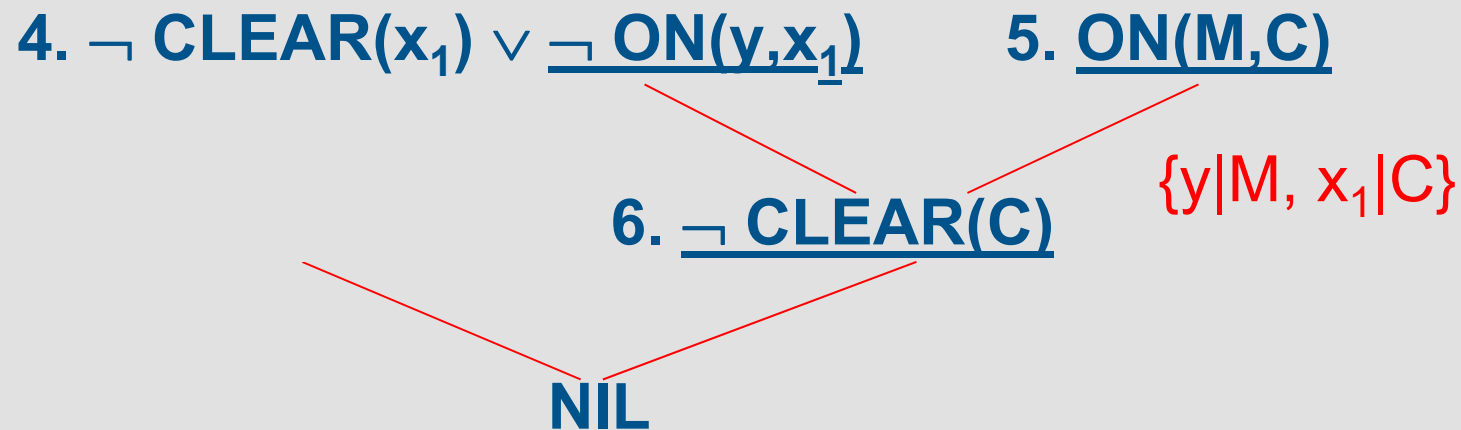
- **Question answering**
  1. MANAGER(Purchasing-dept., John-Jones)
  2. WORKSIN(Purchasing-dept., Joe-Smith)
  3. $\forall x \forall y \forall z$ [WORKSIN(x,y) Λ MANAGER(x,z)] $\Rightarrow$ BOSSOF(y,z)
  4. **Goal:** Who is the boss of Joe Smith?
     $\exists x$ BOSSOF(Joe-Smith,x)

- **Planning**

MONASH University
Information Technology

Theorem Proving – Example

1. ON(C,A)
2. (a) ONTABLE(A), (b) ONTABLE(B)
3. (a) CLEAR(C), (b) CLEAR(B)
4. $\forall$x [CLEAR(x) $\Rightarrow$ $\neg$ $\exists$y ON(y,x)]
   $\neg$ CLEAR(x_1) $\vee$ $\neg$ ON(y,x_1)
5. **Goal:** Prove $\neg\exists$y ON(y,C)
   Negate goal: $\exists$y ON(y,C)
   ON(M,C)

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

6. $\neg$ CLEAR(C)   {y|M, x_1|C}

NIL

MONASH University
Information Technology

Question Answering – Example

3. $\neg$ WORKSIN(x_1,y) v $\neg$ MANAGER(x_1,z) v <u>BOSSOF(y,z)</u>

        4. <u>$\neg$ BOSSOF(Joe-Smith,x_2)</u> v BOSSOF(Joe-Smith,x_2)

        {y|Joe-Smith, x_2|z}

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

        1. <u>MANAGER(Purchasing-dept., John-Jones)</u>

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

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

        2. <u>WORKSIN(Purchasing-dept,Joe-Smith)</u>

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 and Bayesian Networks

MONASH University
Information Technology