CSIT 5900
Knowledge Representation and Reasoning

Department of Computer Science and Engineering
Hong Kong University of Science and Technology

Intelligence and Reasoning

- Copycats are not intelligent:

  Emperor: It's white.
  Minister: It's white.
  Emperor: It's black.
  Minister: It's black.

- Simple pattern-matching-substitution agents are not intelligent either:

  John's program:

```
char *name = "John";
main() {
  int len = 0;
  while (name++ != '0') ++len;}
```

  Jack's program:

```
char *myname = "John";
main() {
  int mylen = 0;
  while (myname++ != '0') ++mylen;}
```

ELIZA - We Now Have ChatGPT

ELIZA (Weizenbaum, 1963), a computer psychotherapist based on pattern matching that fooled a lot of people (from *Machines Who Thinks*, by P. McCorduck, 1979):

DOCTOR: Good afternoon. Please type in your name.

[The visitor did so, and the doctor continued with some polite preliminaries]

DOCTOR: What brought you here to see me today?

VISITOR: Oh, nothing much. I'm feeling a bit tired, that's all.

DOCTOR: Why do you think you're feeling a bit tired?

VISITOR: Well, I've been travelling a lot, and away from home.

DOCTOR: Tell me about your family.

[From here, the conversation became intimate...]

Agents That Reason Logically

The cat or the tiger?

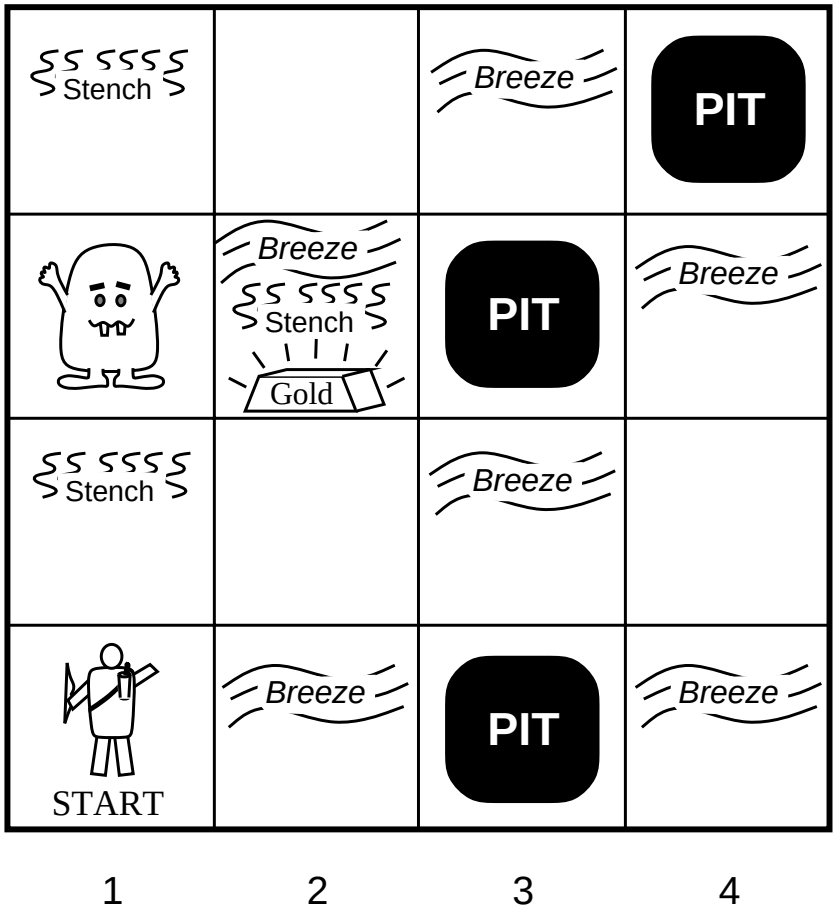
There are two rooms. Each of them is occupied by either a tiger or a cat, but not both. There is also a sign on the door of each room, and the signs are either both true or both false. Here are the signs:

Room I	Room II
either a tiger is in this room or a cat is in the other room	a cat is in the other room

What does Room I contain? How about Room II?

Agents That Reason Dynamically

A typical state in the Wumpus World:



Wumpus World

- Initial state: the agent is in [1,1], facing east (up), and with one arrow.
- Goal: Get the gold and return to [1,1] and climb out of the cave.
- Percepts: given by an array of five sensors [Stench,Breeze,Glitter,Bump,Scream]. Notice that the agent does not know where she is – the agent was put in a cell, and she call that cell [1,1].
- Actions: grab the gold; turn 90 degree clockwise; turn 90 degree counter clockwise; move forward to the next cell; shoot the arrow in the direction that it is facing.

Question: How can the agent figure out that [2,2] are safe to go into?

Agents That Reason about Other Agents Logically

Sarah (the girl) and Ted (the boy) came home from playing outside.

- The mother (to both children): At least one of your foreheads is dirty.
- The boy: I don't know if mine is dirty.
- The girl: My forehead is dirty.
- The boy: My forehead is dirty.

How do they know?

Knowledge-Based Agents

Two main components:

- Knowledge base
  - The collection of known facts about the world.
  - Each item, called a *sentence*, expressed in some representation *language* as a computer-tractable form.
  - Should be updated over time.
- Reasoning (or inference)
  - Reasoning about knowledge in the KB to decide the best action to achieve some given goal.
  - Search algorithms that we have learnt can be used to carry out the reasoning

Knowledge Representation and Reasoning

Knowledge representation is about how to represent things that are needed to solve a problem on a computer:

- A KR language has two components: its syntax and semantics. (Compare it with computer programming language.)
- Syntax determines what are legal expressions (sentences) and semantics tells us the meaning of these legal expressions.
- Good knowledge representation languages should be:
  - *expressive* and *concise* like natural languages
  - *unambiguous* and *precise* like formal languages.

Logical inference: the process of deriving new sentences from old ones.

Two common kinds of logic:

- Propositional logic (also called Boolean logic)
- First-order predicate logic (or simply first-order logic)

Propositional Logic

Language

The language (syntax) of a logic describes how to form sentences.

> *This is like the syntax of a programming language. E.g. the syntax of C says that* `++x` *is a legal expression, but* `**x` *is not.*

To define a language, we need to define its available symbols and formation rules. The former are the basic building blocks (tokens). The latter describes how to form sentences from these symbols.

Language (Continued)

Symbols:

- Logical symbols (those whose meanings are fixed by the logic): parentheses "(" and ")", sentential connectives $\neg$, $\wedge$, $\vee$, $\supset$, $\equiv$.
- Non-logical symbols (those whose meanings depend on interpretations): proposition symbols $p_1, p_2, \ldots$.

Formation rules:

- a proposition symbol is a sentence
- if α and β are sentences, so are $(\neg\alpha)$, $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\alpha \supset \beta)$, and $(\alpha \equiv \beta)$.
- no other expressions are sentences

Convention: the outermost parentheses can be dropped, and we shall use the following precedence orders for the connectives: $\neg \mapsto \wedge, \vee \mapsto \supset \mapsto \equiv$.
$p \vee \neg q \equiv q \supset r$ is $(p \vee (\neg q)) \equiv (q \supset r)$.

Example

The suspect must be released from custody: R
The evidence obtained is admissible: E
The evidence obtained is inadmissible: $\neg E$
The evidence obtained is admissible, and the suspect need not be released from custody: $E \wedge (\neg R)$
Either the evidence is admissible or the suspect must be released from custody

Quiz: $E \vee R$ or $(E \vee R) \wedge \neg(E \wedge R)$?

The evidence is inadmissible, but the suspect need not be released from custody

Quiz: How do we translate "but"?

Semantics - Truth Conditions

To specify semantics of a logic, we need to first assume an interpretation of the non-logical symbols in the language, and then define the meaning of logical symbols in terms of this interpretation.
For propositional logic, an interpretation, often called a truth assignment, is a function from the set of propositional symbols in the language to the set of truth values $\{T, F\}$.
Given such a truth assignment v, we can extend it to the set of sentences using the following truth table

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \supset q$	$p \equiv q$
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

Entailment

We say a truth assignment v satisfies a sentence α iff $v(\alpha) = T$.
We say that a set of sentences Σ (tautologically) implies α, written $\Sigma \models \alpha$, iff every truth assignment that satisfies every member of Σ also satisfies α.
We say that a sentence α is a tautology (valid), written $\models \alpha$, iff it is implied by the empty set of sentences, i.e. $\emptyset \models \alpha$.
Tautologies are all we care about because of the following deduction theorem:
Theorem $\{\alpha_1, ..., \alpha_n\} \models \alpha$ iff $\models \alpha_1 \wedge \cdots \wedge \alpha_n \supset \alpha$.
Sketch of proof. *Only if case:* Suppose v is a truth assignment. There are two cases: (1) v does not satisfies $\alpha_1 \wedge \cdots \wedge \alpha_n$. In this case, v trivially satisfies $\alpha_1 \wedge \cdots \wedge \alpha_n \supset \alpha$. (2) v satisfies $\alpha_1 \wedge \cdots \wedge \alpha_n$. So v also satisfies every member of $\{\alpha_1, ..., \alpha_n\}$. By the assumption that $\{\alpha_1, ..., \alpha_n\} \models \alpha$, v satisfies α as well.
If case: Similar.

Example Tautologies

De Morgan's laws:
$\neg(p \wedge q) \equiv \neg p \vee \neg q$ and $\neg(p \vee q) \equiv \neg p \wedge \neg q$
Distributive laws:
$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$ and $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
Excluded middle: $p \vee \neg p$

Contradiction: $\neg(p \wedge \neg p)$

Contraposition: $p \supset q \equiv \neg q \supset \neg p$

Exportation: $p \wedge q \supset r \equiv p \supset (q \supset r)$
Tautologies that use $\neg$ and $\vee$ to define all other connectives:
$p \wedge q \equiv \neg(\neg p \vee \neg q)$ $\quad\quad$ $p \supset q \equiv \neg p \vee q$

$(p \equiv q) \equiv (p \supset q) \wedge (q \supset p)$
Quiz: Is $((p \supset q) \supset p) \supset q$ a tautology?
How about $((p \supset q) \supset p) \supset p$?

Applications: Query Answering

Suppose: if it is sunny, then you get mail; if it is either rain or sleet, then you still get mail. It will be either raining or sunny tomorrow, would you get mail?

Axiomatization:

- Propositions: $Rain$, $Sunny$, $Sleet$, $Mail$.
- KB: $Rain \vee Sunny$ $\quad$ $Sunny \supset Mail$ $\quad$ $(Rain \vee Sleet) \supset Mail$.
- Query: Does $KB \models Mail$?

Applications: Problem Solving by Finding Models

Consider the graph coloring problem: you have three colors, say black, white, and gray, and you want to assign a color to every node in a graph such that no adjacent nodes have the same color.

Axiomatization:

- Propositions: $n(c)$, for each node n, and each color c, $adj(n_1, n_2)$, for every pair of nodes.
- Axioms:
  - Each node must have exactly one color assigned to it: for each node n,

$$[n(b) \vee n(w) \vee n(g)] \wedge [n(b) \supset (\neg n(w) \wedge \neg n(g))] \wedge$$
$$[n(w) \supset (\neg n(b) \wedge \neg n(g))] \wedge [n(g) \supset (\neg n(w) \wedge \neg n(b))]$$

  - No adjacent nodes can have the same color:

$$adj(n_1, n_2) \supset \neg(n_1(b) \wedge n_2(b)) \wedge \neg(n_1(w) \wedge n_2(w)) \wedge \neg(n_1(g) \wedge n_2(g)).$$

  - A database of $adj(n_1, n_2)$ facts,

Now every model of the above theory is a coloring of the graph.

Applications: Problem Solving by Finding Models

Given the following facts:

1. Lisa is not next to Bob in the ranking
2. Jim is ranked immediately ahead of a biology major
3. Bob is ranked immediately ahead of Jim
4. One of the women (Lisa and Mary) is a biology major
5. One of the women is ranked first

What are possible rankings for the four people?
Axiomatization:

- Propositions: $n(l, b)$, $n(l, j)$, $n(l, m)$, ..., $bm(l)$, $bm(m)$, $bm(b)$, $bm(j)$.

KB:

$$\neg n(l, b) \wedge \neg n(b, l),$$
$$n(j, l) \vee n(j, b) \vee n(j, m) \wedge$$
$$n(j, l) \supset bm(l) \wedge$$
$$n(j, b) \supset bm(b) \wedge$$
$$n(j, m) \supset bm(m),$$
$$n(b, j),$$
$$bm(l) \vee bm(m),$$
$$(\neg n(b, l) \wedge \neg n(m, l) \wedge \neg n(j, l)) \vee (\neg n(b, m) \wedge \neg n(l, m) \wedge \neg n(j, m))$$

We also need to add: each person can have exactly one rank, and no two people can occupy a same rank.
Any other ways to axiomatize the domain?

Applications: Diagnosis

Example:

$$\textit{tennis-elbow} \supset \textit{sore-elbow}$$
$$\textit{tennis-elbow} \supset \textit{tennis-player}$$
$$\textit{arthritis} \wedge \textit{untreated} \supset \textit{sore-joint}$$
$$\textit{sore-joint} \supset \textit{sore-elbow} \wedge \textit{sore-hip}$$

Explain: *sore-elbow*. Possible explanations: *tennis-elbow*, *arthritis* $\wedge$ *untreated*.
Here, $KB \not\models$ *sore-elbow*, but $KB \cup \{$*tennis-elbow*$\} \models$ *sore-elbow*.
Generally, an *abduction* of α under KB is a formula β such that $KB \cup \{\beta\} \models \alpha$. This type of reasoning is wide-spread in many applications.

Clausal Representation

- Represent a formula (boolean function) by a set of *clauses*.
- A clause is a disjunction of *literals*: $l_1 \vee \cdots \vee l_k$.
- A literal is either proposition symbol (variable) (*positive literal*) or its negation (*negative literal*).

CNF and DNF

A set S of clauses is the conjunction of the clauses in S, thus denoting a sentence in Conjunctive Normal Form (CNF): a Conjunction of disjunctions of literals.
Every sentence α can be converted into a CNF α' in such a way that $\models \alpha \equiv \alpha'$:

1. eliminate $\supset$ and $\equiv$ using
   $\alpha \equiv \beta \Rightarrow (\alpha \supset \beta) \wedge (\beta \supset \alpha)$ and $\alpha \supset \beta \Rightarrow \neg\alpha \vee \beta$
2. push $\neg$ inside using
   $\neg(\alpha \wedge \beta) \Rightarrow \neg\alpha \vee \neg\beta$ and $\neg(\alpha \vee \beta) \Rightarrow \neg\alpha \wedge \neg\beta$
3. distribute $\vee$ over $\wedge$ using
   $(\alpha \wedge \beta) \vee \gamma \Rightarrow (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$
4. simplifying items using
   $\alpha \vee \alpha \Rightarrow \alpha$ and $\neg\neg\alpha \Rightarrow \alpha$.
5. eliminate disjunctions that contain both a literal and its compliment.

Example

$$
\begin{aligned}
((p \supset q) \supset p) \supset q \quad &\Rightarrow \quad \neg((p \supset q) \supset p) \vee q \\
&\Rightarrow \quad \neg(\neg(p \supset q) \vee p) \vee q \\
&\Rightarrow \quad \neg(\neg(\neg p \vee q) \vee p) \vee q \\
&\Rightarrow \quad \neg((\neg\neg p \wedge \neg q) \vee p) \vee q \\
&\Rightarrow \quad (\neg(\neg\neg p \wedge \neg q) \wedge \neg p) \vee q \\
&\Rightarrow \quad ((\neg\neg\neg p \vee \neg\neg q) \wedge \neg p) \vee q \\
&\Rightarrow \quad (\neg\neg\neg p \vee \neg\neg q \vee q) \wedge (\neg p \vee q) \\
&\Rightarrow \quad (\neg p \vee q) \wedge (\neg p \vee q) \\
&\Rightarrow \quad \text{clausal form representation:} \\
&\qquad\quad \{\neg p \vee q\}
\end{aligned}
$$

Resolution Rule of Inference

Given two clauses, infer a new clause:

from clauses $p \vee C_1$ and $\neg p \vee C_2$
infer clause $C_1 \vee C_2$. This new clause is called the *resolvent* of input clauses with respect to p.
Example: from clauses $w \vee p \vee q$ and $w \vee s \vee \neg p$ infer $w \vee q \vee s$ as resolvent wrt p.
Special case: p and $\neg p$ resolve to [] (the empty clause, false, contradiction).
A *derivation* of a clause c from a set $\mathcal{S}$ of clauses is a sequence $c_1, c_2, \cdots, c_n$ of clauses, where the last clause $c_n = c$, and for each c_i, either

1. $c_i \in \mathcal{S}$ or
2. c_i is a resolvent of two earlier clauses in the derivation

Write: $\mathcal{S} \rightarrow c$ if there is a derivation of c from $\mathcal{S}$.

Properties of Resolution

- Resolvent is entailed by input clauses:

$$\{(p \vee \alpha), (\neg p \vee \beta)\} \models \alpha \vee \beta$$

- More generally, if $\mathcal{S} \rightarrow c$ then $\mathcal{S} \models c$.
- Special case: If $\mathcal{S} \rightarrow []$, then $\mathcal{S} \models \mathit{FALSE}$, i.e. $\mathcal{S}$ is unsatisfiable, that is, there is no truth assignment that satisfies every member of $\mathcal{S}$.

Proof By refutation

Resolution is sound and complete for []:

Theorem $\mathcal{S}$ is unsatisfiable iff $\mathcal{S} \rightarrow []$
This completeness is also called *refutation complete*, and is all we need:

$\Sigma \models \alpha$ iff $\Sigma \cup \{\neg\alpha\}$ is unsatisfiable iff the CNF $\mathcal{S}$ of $\Sigma \cup \{\neg\alpha\}$ is unsatisfiable iff we can derive [] from $\mathcal{S}$.

Proving $\Sigma \models \alpha$ by checking the satisfiability of $\Sigma \cup \{\alpha\}$ is often called proof by refutation.

A Procedure

To determine if $KB \models \alpha$:

- put KB and $\neg\alpha$ into CNF to get $\mathcal{S}$.
- check if $\mathcal{S} \rightarrow []$:
  1. check if [] is in $\mathcal{S}$. If yes, then return "unsatisfiable"
  2. check if there are two clauses c_1 and c_2 in $\mathcal{S}$ such that they resolve to produce a c_3 not already in $\mathcal{S}$. If no such c_3 can be generated, then return "satisfiable"
  3. add c_3 to $\mathcal{S}$ and go back to the first step.

Note: need only convert KB to CNF once:

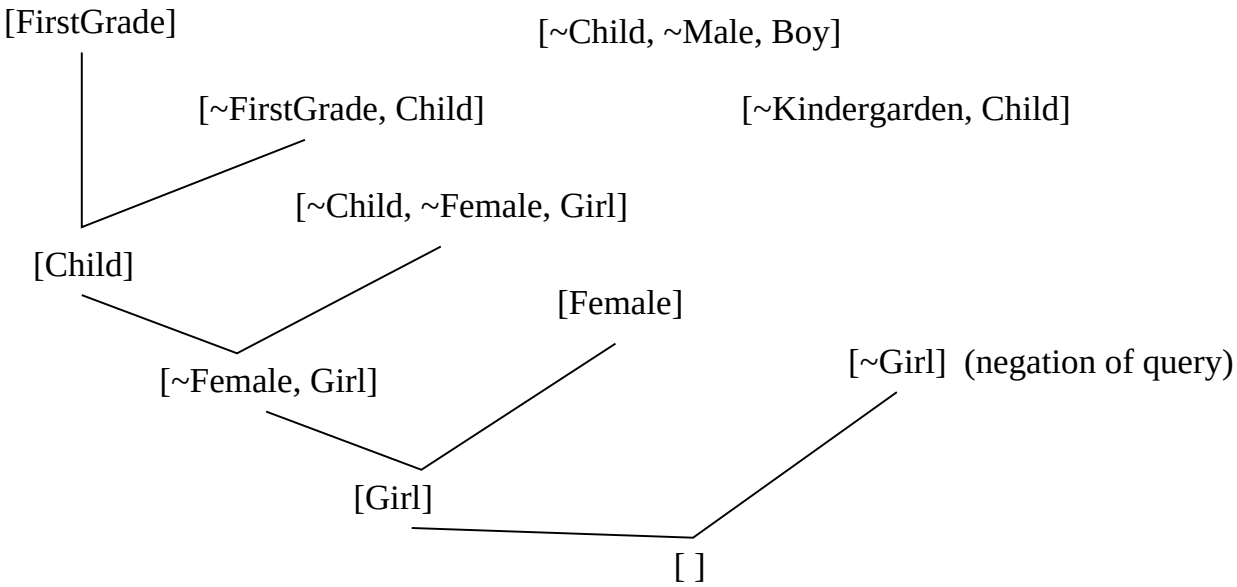
- can handle multiple queries with same KB
- after addition of a new fact β, can simply add new clauses from β's CNF to KB

An Example

KB:

$$FirstGrade,\ FirstGrade \supset Child,\ Child \wedge Male \supset Boy,$$
$$Kindergarden \supset Child,\ Child \wedge Female \supset Girl,\ Female.$$

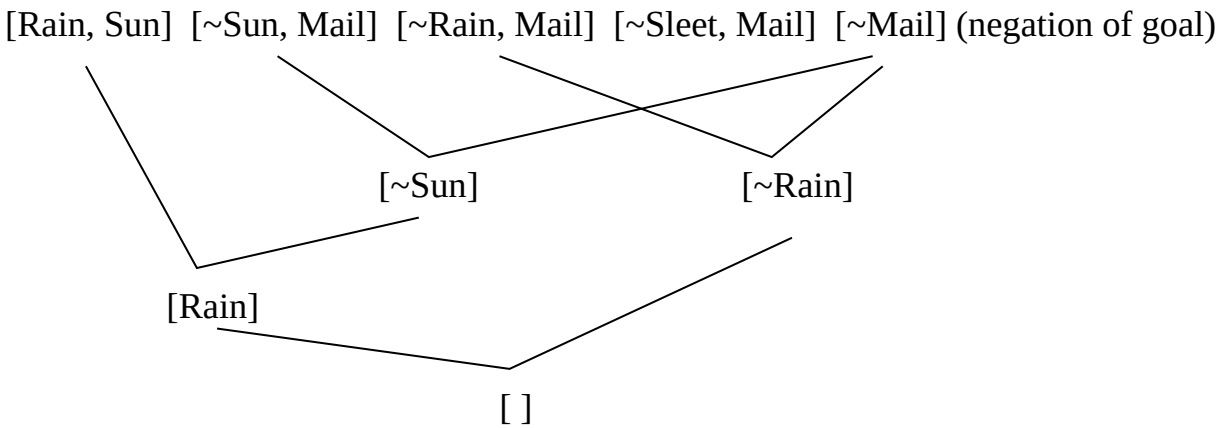
Show that $KB \models Girl$
(In the following drawing, $[l_1, l_2, ..., l_k]$ denotes the clause $l_1 \vee l_2 \vee \cdots \vee l_k$)



An Example

KB: $Rain \vee Sun$ $\quad$ $Sun \supset Mail$ $\quad$ $(Rain \vee Sleet) \supset Mail$

Show $KB \models Mail$



We can also show that $KB \not\models Rain$ by enumerating all possible derivations.

Problems with the propositional agents

Problem: Too many propositions, and too many rules!

Solution: onto first-order logic.

First-Order Logic

- Propositional logic is not expressive and concise enough – the world is a set of facts according to it.
- First-order logic (FOL) is more expressive than propositional logic.
- According to FOL, the world consists of objects. The facts in the world are simply properties about or relations among these objects.
- First-order logic is *universal* - whatever can be said can be written down in FOL, any fact can be thought of as referring to objects and properties or relations:
  - "one plus two equals three" – objects: one two, three, one plus two; relation: equal.
  - "squares neighboring the wumpus are smelly" – objects: wumpus, squares; property: smelly; relation: neighboring.
  - "evil King John ruled England in 1200".

Syntax - Alphabet

Logical Symbols: fixed meaning and use, like keywords in a programming language

- punctuations and parentheses.
- connectives: $\neg$, $\wedge$, $\vee$, $\supset$, $\equiv$.
- quantifiers: $\forall$, $\exists$.
- equality: $=$
- variables: x, x_1, ..., x', x'', ..., y, ..., z, ...

Non-logical symbols: domain-dependent meaning and use, like identifiers in a programming language

- predicate symbols: have arity, i.e. take fixed number of argument. E.g. *Dog* is a 1-ary (unary) predicate, *Dog*(*fido*) means that Fido is a dog. Propositional symbols are 0-ary predicates.
- function symbols: have arity as well. E.g. *plus* is a 2-ary (binary) function, *plus*(x, y) means the sum of x and y. Constant symbols like *fido* are 0-ary functions.

Syntax - Terms and Sentences

A *term* is a logical expression that refers to an object:

- a constant is a term.
- a variable is a term.
- if $t_1, \cdots, t_n$ are terms, and f is an n-ary function symbol, then $f(t_1, ..., t_n)$ is a term.

Example: *fido*, $fatherOf(fido)$,$fatherOf(fatherOf(fido))$. $fatherOf(firstChild(fido, motherOf(dido)))$.

A *sentence* is a logical expression that represents a fact:

- if $t_1, \cdots, t_n$ are terms, and P is an n-ary predicate (relation) symbol, then $P(t_1, ..., t_n)$ is a sentence (*atomic sentence*).
- if t_1 and t_2 are terms, then $t_1 = t_2$ is a sentence (*atomic sentence*).
- if α and β are sentences, then $\neg\alpha$, $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\alpha \supset \beta)$, $(\alpha \equiv \beta)$ are also sentences.
- if α is a sentence, and v is a variable, then $(\forall v\ \alpha)$, $(\exists v\ \alpha)$ are also sentences.

Example Sentences

- Atomic sentences: simple facts about the world. Examples:
  $Brother(richard, john)$,
  $fatherOf(richard) = fatherOf(john)$,
  $Student(firstChild(john, mary), ust)$.
- Quantifiers:
  - Universal quantification ($\forall$):
    - ★ General rules (for *all* objects in universe)
    - ★ Example: "COMP101 students are UST students."
      $\forall x\ Enrolled(x, comp101) \supset Student(x, hkust)$
  - Existential quantification ($\exists$):
    - ★ For *one or more* objects in universe
    - ★ Example: "not all UST students enroll COMP101"
      $\exists x\ Student(x, hkust) \wedge \neg Enrolled(x, comp101)$

Example Sentences

- Quantifiers
  - Nested quantifiers:
    - ★ "brothers are also siblings":

$$\forall x, y \ \mathit{Brother}(x, y) \supset \mathit{Sibling}(x, y)$$

      Here $\forall x, y$ is a shorthand for $\forall x \forall y$.
    - ★ "everybody loves somebody": $\forall x \exists y \ \mathit{Loves}(x, y)$.
    - ★ "a mother is someone who is a female and has a child":

$$\forall x \ [\mathit{Mother}(x) \equiv \mathit{Female}(x) \wedge \exists y \ \mathit{Child}(x, y)]$$

  - Relationships between $\forall$ and $\exists$:
    - ★ $\neg \exists x \ P \equiv \forall x \ \neg P \quad \exists x \ \neg P \equiv \neg \forall x \ P$
    - ★ $\forall x \ P \equiv \neg \exists x \ \neg P \quad \exists x \ P \equiv \neg \forall x \ \neg P$

Some Examples

- All purple mushrooms are poisonous:

$$\forall x\ \mathit{Mushroom}(x) \wedge \mathit{Purple}(x) \supset \mathit{Poisonous}(x)$$

- No purple mushroom is poisonous:

$$\forall x\ \mathit{Mushroom}(x) \wedge \mathit{Purple}(x) \supset \neg \mathit{Poisonous}(x)$$

- All mushrooms are either purple or poisonous:

$$\forall x\ \mathit{Mushroom}(x) \supset \mathit{Purple}(x) \vee \mathit{Poisonous}(x)$$

- All mushrooms are either purple or poisonous but not both:

$$\begin{aligned} \forall x \quad & \mathit{Mushroom}(x) \supset \\ & (\mathit{Purple}(x) \wedge \neg \mathit{Poisonous}(x)) \vee \\ & (\neg \mathit{Purple}(x) \wedge \mathit{Poisonous}(x)) \end{aligned}$$

Examples

- All purple mushrooms except one are poisonous:

$$\begin{aligned}\exists x \quad & \mathit{Purple}(x) \wedge \mathit{Mushroom}(x) \wedge \neg \mathit{Poisonous}(x) \wedge \\ & (\forall y\ \mathit{Purple}(y) \wedge \mathit{Mushroom}(y) \wedge \neg(x = y) \supset \\ & \mathit{Poisonous}(y))\end{aligned}$$

- There are exactly two purple mushrooms:

$$\begin{aligned}\exists x, y \quad & \mathit{Mushroom}(x) \wedge \mathit{Purple}(x) \wedge \\ & \mathit{Mushroom}(y) \wedge \mathit{Purple}(y) \wedge \neg(x = y) \wedge \\ & (\forall z\ \mathit{Mushroom}(z) \wedge \mathit{Purple}(z) \supset (z = x) \vee (z = y))\end{aligned}$$

Interpretation

An interpretation for FOL settles:

- what objects there are
- which of them satisfy a predicate P
- what mapping is denoted by a function f

Given an interpretation I, a sentence α is either true or false in I.

In class example: An interpretation for a langauge with $Friends(x, y)$, $Hates(x, y)$, $Likes(x, y)$, $Person(x)$, $Dog(x)$, $John$, $Lisa$, $Fido$.

Entailment

- An interpretation I is a model of a sentence α (α is satisfied in I) if α is true in I.
- A set of sentences Σ logically entails a sentence α if for any interpretation I, whenever I is a model for all sentences in Σ, then it is a model of α.
- α is valid if it is entailed by the empty set, that is, if it is true in all interpretations.

Refutation (Proof By Contradiction)

- A KB is contradictory if there is a sentence α such that both $KB \models \alpha$ and $KB \models \neg\alpha$.
- A trivial contradictory KB: $\{Man(socrates), \neg Man(socrates)\}$.
- Finding out if a KB is contradictory is as hard as doing other reasoning tasks such as entailment.
- Refutation (proof by contradiction): to show that $KB \models \alpha$, add $\neg\alpha$ to KB and show that the new knowledge base, $KB \cup \{\neg\alpha\}$ is contradictory.
- We do this all the time: If the driver had fastened his seat belt, he wouldn't have been thrown out of the car; since he was thrown out of the car, he must have not fastened his seat belt.

Resolution (First-Order case)

- Example 1 (Modus Ponens):

$$\frac{\neg Man(x) \vee Mortal(x),\ \ Man(socrates)}{Mortal(socrates)}$$

  - The variables in the rule are implicitly universally quantified.
  - The first premise really is

$$\forall x\ \neg Man(x) \vee Mortal(x)$$

    which is equivalent to $\forall x\ Man(x) \supset Mortal(x)$.
  - This rule is the same as:

$$\frac{\neg Man(x) \vee Mortal(x),\ \ Man(socrates)}{\textsc{Subst}(\{x/socrates\}, Mortal(x))}$$

- Example 2:

$$\frac{\neg Man(x) \vee Mortal(x),\ \neg Mortal(y) \vee Die(y)}{\neg Man(z) \vee Die(z)}$$

This rules says: if

$\forall x\ \neg Man(x) \vee Mortal(x)$, "every man is mortal"

$\forall y \neg Mortal(y) \vee Die(y)$, "every mortal thing has to die"

then $\forall z\ \neg Man(z) \vee Die(z)$ ("every man has to die"). Notice that this rule can be written as:

$$\frac{\neg Man(x) \vee Mortal(x),\ \neg Mortal(y) \vee Die(y)}{\textsc{Subst}(\{x/z, y/z\}, \neg Man(x) \vee Die(y))}$$

Resolution (Cont'd)

- Resolution (first-order case): if θ is a substitution such that $\textsc{Subst}(\theta, r) = \textsc{Subst}(\theta, r')$, then

$$\frac{p_1 \vee \cdots p_i \vee r \vee p_{i+1} \cdots \vee p_m, \quad q_1 \vee \cdots q_j \vee \neg r' \vee q_{j+1} \cdots \vee q_n}{\textsc{Subst}(\theta, \ p_1 \vee \cdots \vee p_m \vee q_1 \vee \cdots \vee q_n)}$$

  where $p_1, .., p_m, q_1, ..., q_n$ are literals and r and r' are atoms.
- Example 1 (Modus Ponens):

$$\frac{\neg Man(x) \vee Mortal(x), \ \ Man(socrates)}{Mortal(socrates)}$$

  because

$$\textsc{Subst}(\{x/socrates\}, Man(x)) = \textsc{Subst}(\{x/socrates\}, Man(socrates))$$

  and

$$\textsc{Subst}(\{x/socrates\}, Mortal(x)) = Mortal(socrates).$$

- Example 2:

$$\frac{\neg Man(x) \vee Mortal(x),\ \neg Mortal(y) \vee Die(y)}{\neg Man(z) \vee Die(z)}$$

because

$$\textsc{Subst}(\{x/z, y/z\}, Mortal(x)) = \textsc{Subst}(\{x/z, y/z\}, Mortal(y))$$

and

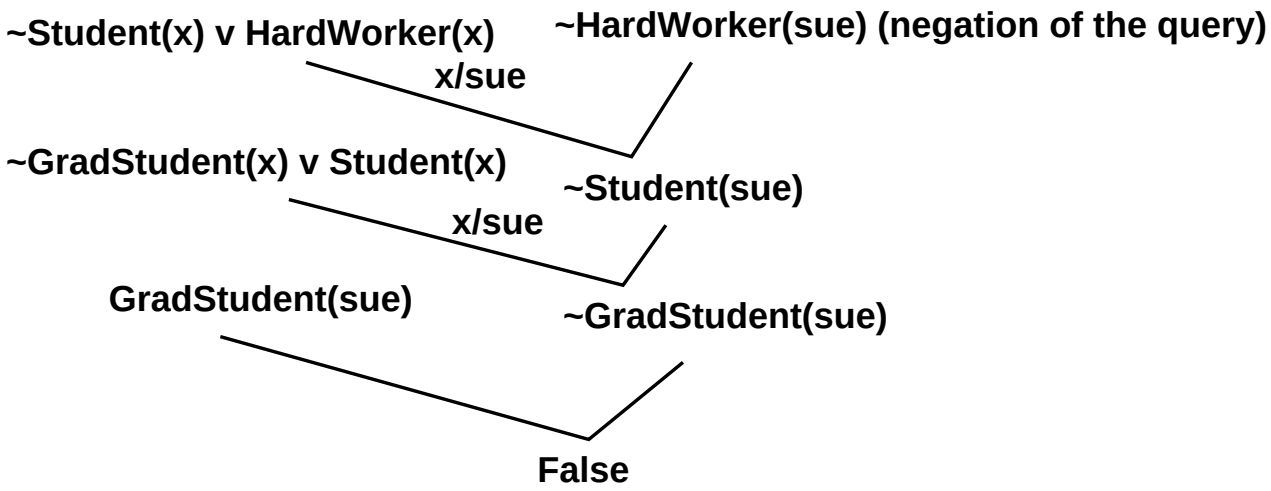
$$\textsc{Subst}(\{x/z, y/z\}, \neg Man(x) \vee Die(y)) = \neg Man(z) \vee Die(z).$$

Example Refutation Using Resolution

KB:

$$\forall x\ GradStudent(x) \supset Student(x),$$
$$\forall x\ Student(x) \supset HardWorker(x),$$
$$GradStudent(sue)$$

Q: $HardWorker(sue)$?



Conversion to Clauses

Converting a first-order sentence to clauses: quite involved in general. We consider only simple cases like the following example:

$$\forall x[\exists y(Dog(y) \wedge Owns(x,y)) \supset AnimalLover(x)] \mapsto \text{ (eliminate implication)}$$

$$\forall x[\neg\exists y(Dog(y) \wedge Owns(x,y)) \vee AnimalLover(x)] \mapsto \text{ (move } \neg \text{ inside)}$$

$$\forall x[\forall y\neg(Dog(y) \wedge Owns(x,y)) \vee AnimalLover(x)] \mapsto \text{ (move } \neg \text{ inside)}$$

$$\forall x[\forall y(\neg Dog(y) \vee \neg Owns(x,y)) \vee AnimalLover(x)] \mapsto \text{ (move quantifiers left)}$$

$$\forall x\forall y[\neg Dog(y) \vee \neg Owns(x,y) \vee AnimalLover(x)] \mapsto \text{ (drop quantifiers)}$$

$$\neg Dog(y) \vee \neg Owns(x,y) \vee AnimalLover(x)$$

Answer Predicate

- Given a KB, to find out an object a that satisfies $P(a)$, we normally pose the query as $\exists x\ P(x)$:

$$KB \models \exists x\ P(x)?$$

- Using resolution, this means that we want to derive *False* from $KB \cup \{\neg P(x)\}$.
- This strategy can be improved by introducing a new predicate A (the answer predicate). Instead of the above query, we ask:

$$KB \models \exists x\ P(x) \wedge \neg A(x)?$$

- Instead of deriving *False*, we want to derive from $KB \cup \{\neg P(x) \vee A(x)\}$ a clause that mentions only A, and then "read" the answer out of it.

The Answer Predicate

- Example 1:
  KB: $\{Stu(john), Stu(jane), Happy(john)\}$
  Q: $\exists x\ Stu(x) \wedge Happy(x)$

  $$\{Stu(john), Stu(jane), Happy(john),$$
  $$\neg Stu(x) \vee \neg Happy(x) \vee A(x)\}$$
  leads to $A(john)$

  So an answer is: John.
- Example 2.
  KB: $\{Stu(john), Stu(jane), Happy(john) \vee Happy(jane)\}$
  Q: $\exists x[Stu(x) \wedge Happy(x)]$

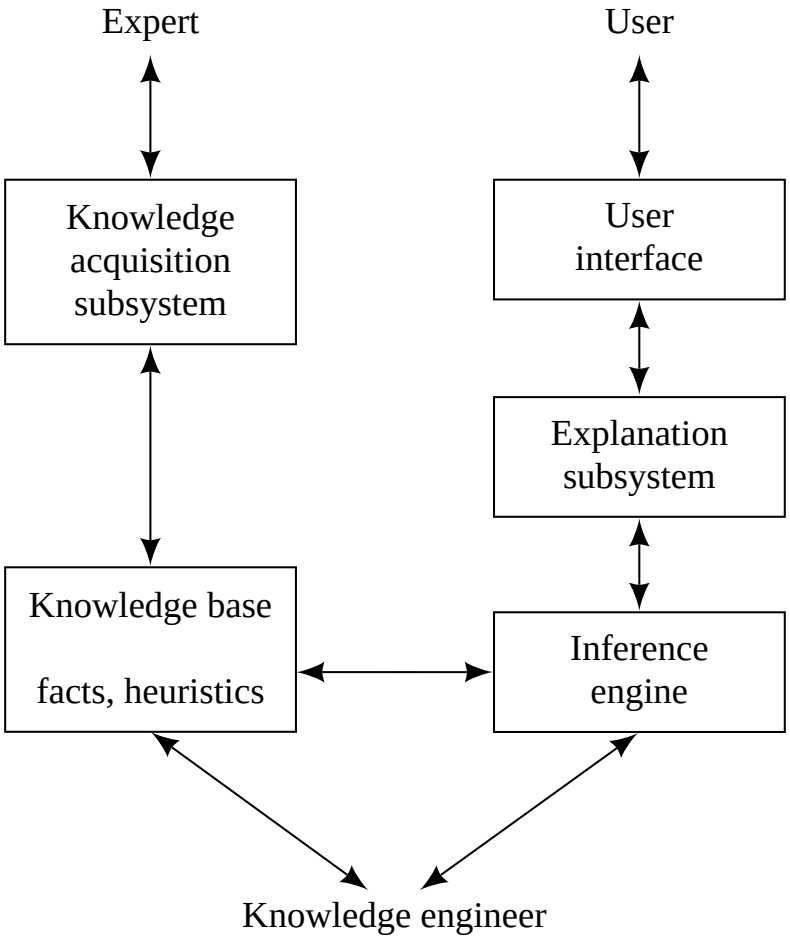
  $$\{Stu(john), Stu(jane), Happy(john) \vee Happy(jane),$$
  $$\neg Stu(x) \vee \neg Happy(x) \vee A(x)\}$$
  leads to $A(john) \vee A(jane)$

  So an answer is: either John or Jane

Rule-Based Expert Systems

- An *expert system* embodies knowledge about a specialized field such as medicine, engineering, or business.
- It is one of the most successful applications of AI reasoning techniques.
- Some well-known expert systems:
  - DENDRAL (1965-83) - one of the earliest expert systems, molecular structure analysis.
  - MYCIN (1972-80) - one of the most influential expert system, medical diagnosis.
  - PROSPECTOR - a mining expert system.
  - XCOM (R1) - a computer configuration system used in DEC.
  - American Express Inc. loan processing expert system.

Basic Architecture



An Example

Rule-based expert systems are often based on Horn clauses (clauses with exactly one positive literal). Consider the following loan-approval system:

1. $COLLAT \wedge PYMT \wedge REP \supset OK$
2. $APP \supset COLLAT$
3. $RATING \supset REP$
4. $INC \supset PYMT$
5. $BAL \wedge REP \supset OK$

where OK means the loan should be approved, $COLLAT$ the collateral for the loan is satisfactory, $PYMT$ the applicant is able to make the loan payments, REP the applicant has a good financial reputation, APP the appraisal on the collateral is sufficiently larger than the loan amount, $RATING$ the applicant has a good credit rating, INC the applicant's income exceeds his/her expenses, and BAL the applicant has an excellent balance sheet.

Rule Learning

- Rule-based expert systems are still widely used.
- It takes a lot of effort to collect these rules from experts.
- An automatic way of acquiring these rules will be very useful.
- We describe an algorithm called *generic separate-and-conquer algorithm* for learning propositional rules.

Loan-Approval Example

Suppose we have following data from bank record:

Individual	APP	RATING	INC	BAL	OK
1	1	0	0	1	0
2	0	0	1	0	0
3	1	1	0	1	1
4	0	1	1	1	1
5	0	1	1	0	0
6	1	1	1	0	1
7	1	1	1	1	1
8	1	0	1	0	0
9	1	1	0	0	0

We want to find out rules about when to approve a loan, i.e. learn rules of the form:

$$\alpha_1 \wedge \cdots \wedge \alpha_n \supset OK$$

where α_i are atoms from the set $\{APP, RATING, INC, BAL\}$.

GSCA

Generic Separate-Conquer Algorithm: Given an atom γ that we want to learn rules about, and Σ, a training set, *GSCA* works as follows:

1. Initialize $\Sigma_{cur} = \Sigma$.
2. Initialize $\pi =$ empty set of rules.
3. **repeat**
   1. Initialize $\Gamma = true$.
   2. Initialize $\tau = \Gamma \supset \gamma$.
   3. **repeat**
      1. Select an atom α from feature set. This is a nondeterministic step, and a backtracking point.
      2. Let Γ be $\Gamma \wedge \alpha$.
   4. **until** τ covers only (or mainly) positive instances in Σ_{cur}.
   5. Let π be π, τ (add the newly learned rule).
   6. Let Σ_{cur} be Σ_{cur} - {the positive instances in Σ_{cur} covered by π}.
4. **until** π covers all (or most) of the positive instance in Σ.

A Heuristic

A common heuristic is to select a feature α that will yield the maximum value of $r_\alpha = n_\alpha^+ / n_\alpha$, where n_α is the total number of instances in Σ_{cur} that is covered by the new rule when α is added to Γ, and n_α^+ the total number of positive instances in Σ_{cur} that is covered by the new rule.

GSCA - The Loan-Approval Example

- We began with the rule: $true \supset OK$.
- It covers all instances, including all negative ones. So we have to narrow them by adding a feature into its antecedent. We do that by choosing one that yields the largest value of r_α:

  $r_{APP} = 3/6 = 0.5, r_{RATING} = 4/6 = 0.667, r_{INC} = 3/6 = 0.5, r_{BAL} = 3/4 = 0.75$

  So we choose BAL, and generate the following rule: $BAL \supset OK$.
- This rule still covers the negative instance 1, so we still need to narrow it. This time we have:

  $$r_{APP} = 2/3, r_{RATING} = 3/3, r_{INC} = 2/2$$

  so we can choose either $RATING$ or INC. Suppose we choose $RATING$, then this yields the rule:

  $$BAL \wedge RATING \supset OK$$

- This rule covers only positive instances, so we have learned one rule. To learn the next one, we eliminate from the sample those rules already cover by it, and continue like above, until no more positive instances are left with.