COMP3211 Lecture 4: Knowledge Representation and Reasoning

Fangzhen Lin

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

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', 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 lady or the tiger?

There are two rooms. Each of them is occupied by either a tiger or a lady, 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 lady is in the other room	a lady is in the other room

What does Room I contain? How about Room II?

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 algoritms 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 $p_1, p_2, ...$

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 droped, 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 inadmisible: $\neg E$
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

Entailments

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 memebr 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 $\neg(p \vee q) \equiv \neg p \wedge \neg q$
Distributie 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: $p \vee \neg p$

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

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 grey, 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 eactly 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

Literal = propositional symbol (*positive literal*) or its negation (*negative literal*)

Notation: If l is a literal, then $\sim l$ is its complement:

$$p \Rightarrow \neg p \qquad\qquad \neg p \Rightarrow p$$

E.g. p is a positive literal, $\neg p$ a negative literal. $\sim p$ is $\neg p$, and $\sim \neg p$ is p.

Clause = set of literals, understood as disjunction of the literals in the set.

Notation: a clause will be written using square brackets. E.g. $[p, \neg q, \neg r]$ is a clause, and stands for the sentence: $p \vee \neg q \vee \neg r$.

Clausal Form = set of clauses, understood as conjunction of the clauses in the set.

Notation: a clausal form will be written using "{" and "}", to distinguish it from a clause. E.g. $\{[p, \neg q], [\neg p, \neg q, r], [r]\}$ is a clausal form, and stands for

$(p \vee \neg q) \wedge (\neg p \vee \neg q \vee r) \wedge r$.

CNF and DNF

A clausal form actually denotes 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

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

Resolution Rule of Inference

Given two clauses, infer a new clause:

from clauses $\{p\} \cup C_1$ and $\{\neg p\} \cup C_2$
infer clause $C_1 \cup C_2$. This new clause is called the *resolvent* of input clauses with respect to p.
Example: from clauses $[w, p, q]$ and $[w, s, \neg p]$ infer $[w, q, s]$ as resolvent wrt p.
Special case: $[p]$ and $[\neg p]$ resolve to []
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 $\mathcal{S}$.

Propeties of Resolution

Resolvent is entailed by input clauses:

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

Proof: Let v be a truth assignment, annd v satisfies $p \vee \alpha$ and $\neg p \vee \beta$.
There are two cases:

- Case 1: v satisfies p. In this case, v satisfies β, so $\alpha \vee \beta$ as well.
- Case 2: v does not satisfies p. In this case, v satisfies α, so $\alpha \vee \beta$ as well.

Either way, v is a model of $\alpha \vee \beta$.
This can be extended to derivations: If $\mathcal{S} \rightarrow c$ then $\mathcal{S} \models c$. (Proof: By induction on the length of the derivation.)
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

So resolution rule of inference is sound. However, it is not complete: can have $\mathcal{S} \models c$ without having $\mathcal{S} \rightarrow c$

$\{[\neg p]\} \models [\neg p, q]$ ($\{\neg p\} \models \neg p \vee q$), but there is no derivation using resolution.
However, 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:

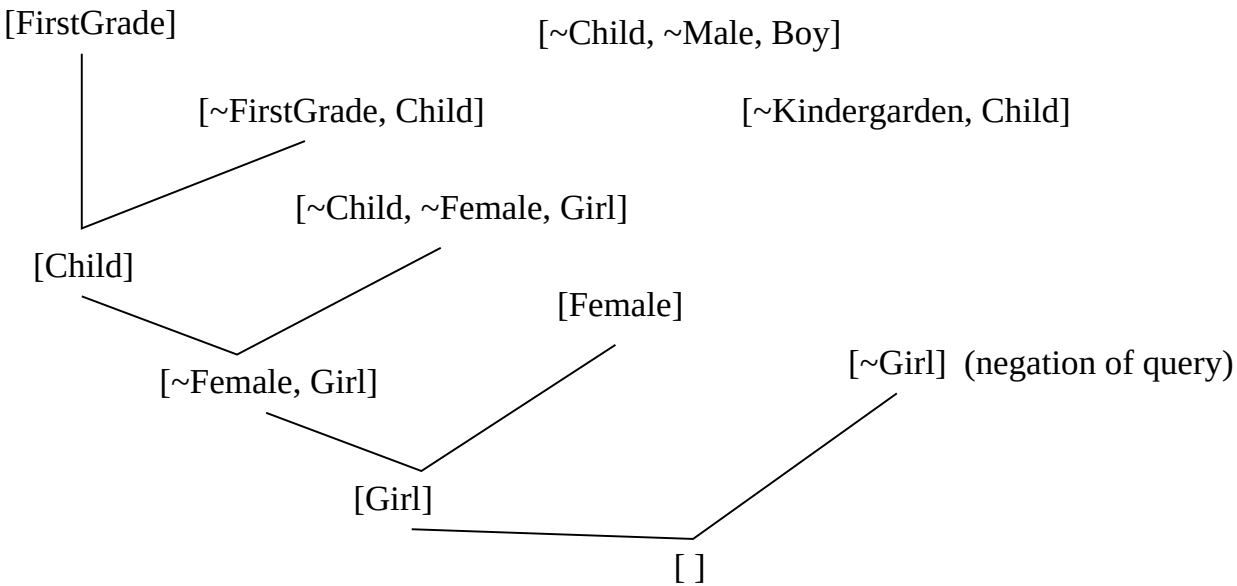
- can handle multiple queries with same KB
- after addition of a new fact β, can simply add new clauses from β's CNF to KB

An Example

KB:

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

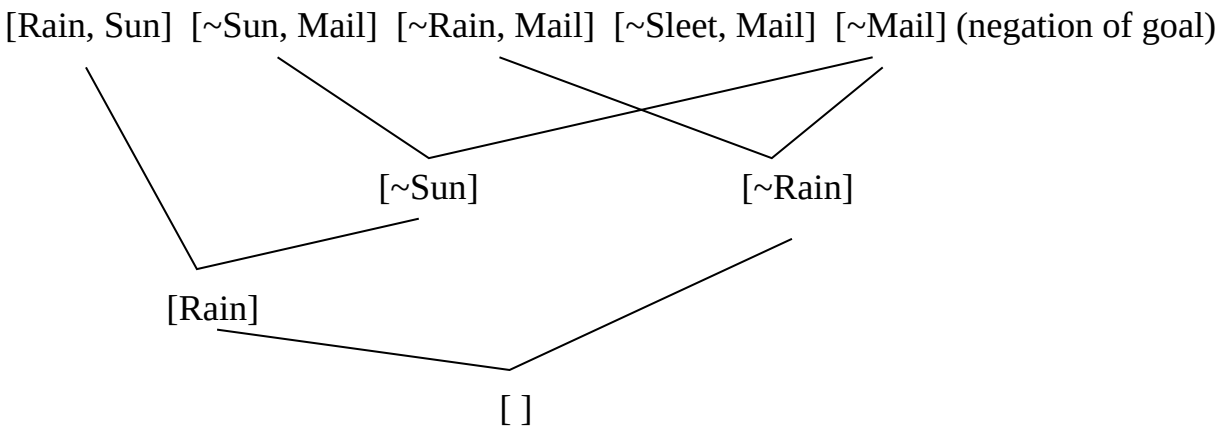
Show that $KB \models Girl$



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.

Satisfiability

- A theory T is satisfiable if there is a truth assignment that satisfies every sentence in T.
- If $\mathcal{S}$ is a CNF of T, then T is satisfiable iff $\mathcal{S}$ is satisfiable.
- SAT refers to the problem of deciding if a set of clauses is satisfiable. It was the first NP-complete problem discovered by Cook, it is also one of the most intensely studied NP-complete problem.
- There is a huge literature on SAT. SAT algorithms have many applications.
- 3SAT referes to the problem of deciding if a set of clauses that have no more than 3 literals is satisfiable.
- In terms of computational complexity, SAT is equivalent to 3SAT.
- A procedure for SAT is sound if whenever it returns yes, the input is indeed satisfiable; it is complete if it will return yes whenever the input is satisfiable. Obviously, we want only sound procedures. But incomplete ones can sometimes be very useful.

DPLL Procedure

The most popular and widely studied compete method for SAT is the so called Davis-Putnam-Longemann-Loveland (DPLL) procedure.
Let α be a set of clauses. If l is a literal occurring in α, then by $\alpha(l)$, we mean the result of letting l true and simplifying: if a clause C does not mention l and $\neg l$, then do nothing; if C mention l, then eliminate it; if C mention $\neg l$, then delete $\neg l$ from C.

Procedure DPLL(CNF: α)

if α is empty, then return yes.
else if there is an empty clause in α, return no.
else if there is a pure literal l in α, then return DPLL($\alpha(l)$).
else if there is a unit clause $\{l\}$ in α, then return DPLL($\alpha(l)$).
else select a variable p in α, and do the following

if DPLL($\alpha(p)$) return yes, then return yes.
else return DPLL($\alpha(\neg p)$).

Where a literal in α is pure if it occurrs only positively or negatively, and a unit clause is one whose length is 1.

GSAT

Incomplete methods cannot be used to check unsatisfiable CNFs. But on satisfiable CNFs, the currently known best incomplete methods are often much faster than the currently known best complete methods.
The following randomized local search algorithm, often called GSAT, is typical of current best known incomplete methods:

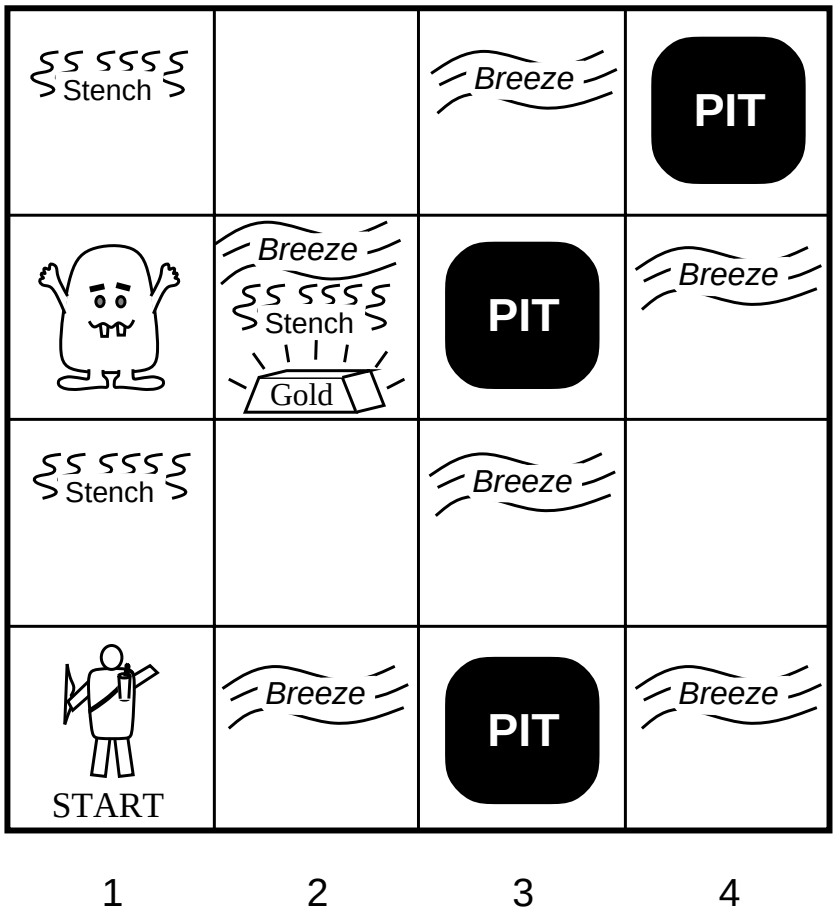
Procedure GSAT(CNF: α, max-restart: mr, max-climb: mc)

for i $= 1$ to mr do
get a randomly generated truth assignment v.
for j $= 1$ to mc do
if v satisfies α, return yes
else let v be a random choice of one the best successors of v
return failure

where a successor of v is an assignment with the truth value of one of the variables flipped, and a best successor is one that satisfies maximum number of clauses in α.

Wumpus World

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.

Acting in the Wumpus World

- Conceptulization: let S_{ij} mean that there is a stench in [i,j], A_{ij} mean that the agent is in [i,j], B_{ij} mean that there is a breeze in [i,j],...
- Axiomatize how the agent should (should not) act based on what she believes:

$$A_{11} \wedge East_A \wedge W_{21} \rightarrow \neg Forward,$$
$$A_{11} \wedge G_{11} \rightarrow Grab,$$
$$\ldots$$

- To decide what to do based on these rules, the agent needs to find out whether, for example, W_{21} is true or not according to her knowledge about the environment and her percepts.

Reasoning in the Wumpus World

Assume that the agent is in the following state, after

(turn-right, forward, turn-left, turn-left, forward, turn-right, forward)

1,4	2,4	3,4	4,4
1,3 **W!**	2,3	3,3	4,3
1,2 **A** **S** **OK**	2,2 **OK**	3,2	4,2
1,1 **V** **OK**	2,1 **B** **V** **OK**	3,1 **P!**	4,1

A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

Assume that the current percept is [Stench,None,None,None,None]. We shall show that at this point. the agent knows that the wumpus is in cell [1,3], so should not heading into that direction.

Reasoning in the Wumpus World (Cont'd)

Knowledge base:

$$
\begin{array}{ll}
 & \neg S_{11}, \quad \neg B_{11} \\
 & \neg S_{21}, \quad B_{21} \\
 & S_{12}, \quad \neg B_{12} \\
R_1 & \neg S_{11} \supset \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{21}, \\
R_2 & \neg S_{21} \supset \neg W_{11} \wedge \neg W_{21} \wedge \neg W_{22} \wedge \neg W_{31}, \\
R_3 & \neg S_{12} \supset \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{22} \wedge \neg W_{13}, \\
R_4 & S_{12} \supset W_{13} \vee W_{12} \vee W_{22} \vee W_{11}
\end{array}
$$

Finding the wumpus:

1. Modus Ponens on $\neg S_{11}$ and R_1, and then AND-Elimination.
2. Modus Ponens on $\neg S_{21}$ and R_2, and then AND-Elimination.
3. Modus Ponens on S_{12} and R_4.
4. Unit resulotion on the resulting formulas.

Problems with the propositional agent

Problem: Too many propositions, and too many rules!

Solution: onto first-order logic.

First-Order Logic

- Propositional logic is not expressive and concise enoguh – 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 sentece (*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":

$$\forall x, y \; Brother(x, y) \supset Sibling(x, y)$$

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

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

  - 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 this, it will be possible to say which sentences of FOL are true and whcih are not.

Formally, an interpretation I is a pair $< D, \Phi >$ where

- D is the *domain* of discourse, and can be any set
- Φ is an *interpretation mapping*:
  - if P is a predicate symbol of arity n, then $\Phi(P)$ is an n-ary relation over D: $\Phi(P) \subseteq D \times \cdots \times D$
  - if f is a function symbol of arity n, then $\Phi(f)$ is an n-ary function over D: $\Phi(f) \in [D \times \cdots \times D \rightarrow D]$

Example

Assume we have: a binary predicate *Less*, a unary function *succ*, and a constant *zero*.

Let the interpretation $I =< D, \Phi >$ be:

- D is the set of natural numbers.
- $\Phi(zero) = 0$.
- $\Phi(succ)(n) = n + 1$ for any number $n \in D$.
- $\Phi(Less)$ is the less than relation over D: for any $< m, n >$, it is in the relation iff $m < n$.

Under this interpretation:

- the constant *zero* denotes the number 0.
- the term $succ(zero)$ denotes the number $0 + 1$ $(= 1)$.
- the term $succ(succ(zero))$ denotes the number $(0 + 1) + 1$.
- The atomic sentence $Less(zero, succ(zero))$ represents the fact "$0 < (0 + 1)$" which is true.

- The sentence

$$Less(zero, succ(zero)) \wedge Less(succ(zero), succ(zero))$$

  is false.
- The sentence $\forall x\ Less(zero, succ(x))$ represents the fact "$0 < (x + 1)$" for all x in the domain D, which is true.

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.

Which of the following sentences are valid?

- $Less(zero, zero) \vee \neg Less(zero, zero)$.
- $Less(zero, succ(zero))$.
- $\neg Less(zero, succ(zero))$.
- $[\forall x\ Less(zero, x)] \supset Less(zero, zero)$.

Denotation

Materials in this and the next slide are for your reference only.
Under an interpretation $I = (D, \Phi)$, terms denote elements of D. For terms with variables, denotation depends on the value of variables. Let s be a mapping from the set of variables to D. Such a mapping is called a *variable assignment.*
The denotation of a term t in interpretation I and under variable assignment s, written $t[I, s]$, is defined as follows:

- if v is a variable, then $t[I, s] = s(v)$.
- recursively, if f is a n-ary function, and $t_1, ..., t_n$ are terms, then $f(t_1, ..., t_n)[I, s] = \Phi(f)(d_1, ..., d_n)$, where $d_i = t_i[I, s]$ for $i = 1, ..., n$. Notice that if f is a constant symbol, then $f[I, s] = \Phi(f)$.

Satisfaction

In terms of I, wffs will be true for some values of the free variables and false for others. We write $I, s \models \alpha$ to mean that α is true in interpretation I under variable assignment s:

- $I, s \models P(t_1, ..., t_n)$ iff $< d_1, ..., d_n > \in \Phi(P)$, where $d_i = t_i[I, s]$ for $i = 1, ..., n$
- $I, s \models t_1 = t_2$ iff $t_1[I, s]$ is the same as $t_2[I, s]$
- $I, s \models \neg\alpha$ iff $I, s \not\models \alpha$
- $I, s \models \alpha \wedge \beta$ iff $I, s \models \alpha$ and $I, s \models \beta$
- $I, s \models \alpha \vee \beta$ iff $I, s \models \alpha$ or $I, s \models \beta$
- $I, s \models \exists v\ \alpha$ iff for some $d \in D$, $I, s(v/d) \models \alpha$
- $I, s \models \forall v\ \alpha$ iff for all $d \in D$, $I, s(v/d) \models \alpha$

where $s' = s(v/d)$ is a variable assigment just like s, except on v, where $s'(v) = d$.

Reasoning By Inference Rules

- An inference rule is an expression of the form:

$$\frac{\alpha_1, ..., \alpha_n}{\beta}$$

  which says that one can infer β given $\alpha_1, ..., \alpha_n$. $\alpha_1, \cdots, \alpha_n$ are called the *premises* and β the *conclusion* of the inference rule.
- Given a set of inference rules, a proof of α from Σ is a sequence of sentences such that the last sentence is α, and each sentence in the sequence is either a premise in Σ or the conclusion of a rule whose premises have already appeared earlier in the sequence.

Example Inference Rules

- Modus ponens (MP):

$$\frac{\alpha \supset \beta, \quad \alpha}{\beta}$$

- Universal elimination (UE):

$$\frac{\forall v \ \alpha}{\textsc{Subst}(\{v/g\}, \alpha)}$$

where v is a variable and g is a ground term (i.e. a term containing no variables). Substitution $\textsc{Subst}(\theta, \alpha)$ is defined as follows:

  - Applying substitution (or binding list) θ to sentence α.
  - Example:

$$\textsc{Subst}(\{x/\mathit{Tom}, y/\mathit{Eva}\}, \mathit{Likes}(x, y)) \ = \ \mathit{Likes}(\mathit{Tom}, \mathit{Eva})$$

Example Proof

- KB:

$$\forall x\ Man(x) \supset Mortal(x),$$
$$Man(socrates)$$

- Query: $Mortal(socrates)$.
- Proof:

(1)	$\forall x\ Man(x) \supset Mortal(x)$	{premise}
(2)	$Man(socrates) \supset Mortal(socrates)$	{(1), UE}
(3)	$Man(socrates)$	{premise}
(4)	$Mortal(socrates)$	{(2), (3), MP}

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 entailments.
- 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 quatified.
  - 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

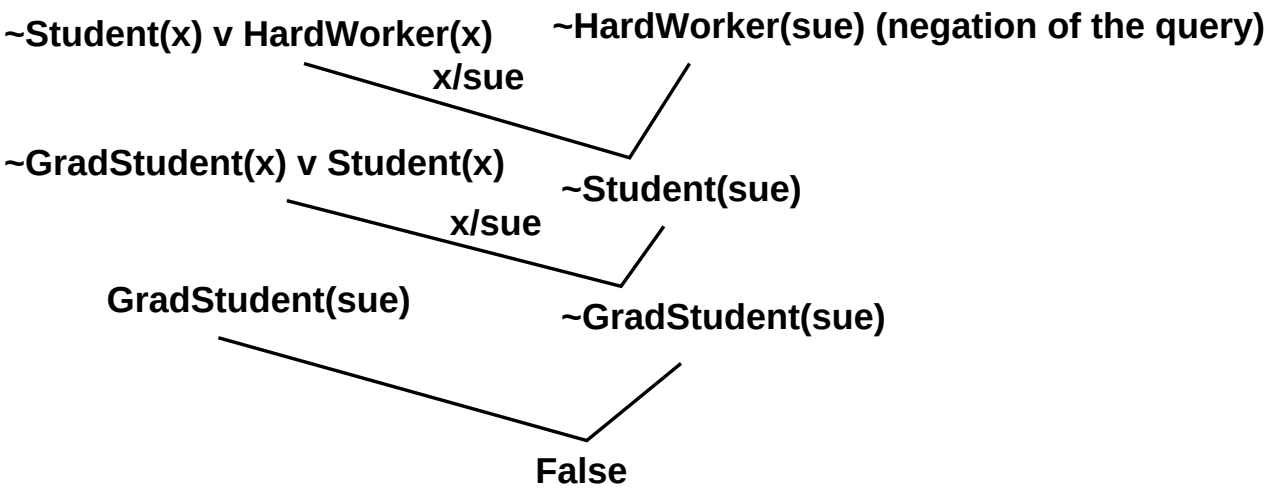
$$\textsc{Subst}(\{x/z, y/z\}, \neg Man(x) \vee Die(y)) = \neg Man(z) \vee Die(z).$$

Example Refutation Using Resolution

KB:

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

Q: $HardWorker(sue)$?



Conjunctive Normal Form (CNF)

- To use resolution, sentences need to be in the form of disjunctions of literals, which are normally called *clauses*.
- A *conjunctive normal form* (CNF) is a conjunction of clauses, i.e. has the following form:

$$
\begin{aligned}
&(p_{11} \vee \cdots \vee p_{1k}) \wedge \\
&(p_{21} \vee \cdots \vee p_{2m}) \wedge \\
&\cdots \wedge \\
&(p_{i1} \vee \cdots \vee p_{in})
\end{aligned}
$$

  where all the p_{xy}'s are literals.
- Just as we normally take a KB to be a set of sentences, we often write a CNF as a set of clauses:

$$\{p_{11} \vee \cdots \vee p_{1k},\ p_{21} \vee \cdots \vee p_{2m},\ \cdots,\ p_{i1} \vee \cdots \vee p_{in}\}$$

- Thoerem proving in first-order logic using resolution

  To prove $\Sigma \models \alpha$, transform Σ and $\neg\alpha$ into a set $\mathcal{S}$ of clauses, and try to generate the empty clause False from $\mathcal{S}$ using resolution.

- We can always transform a sentence into a CNF (a set of clauses).
- The above procedure is sound and complete.

Conversion to CNF

Given a sentence, follow the following steps to transform it into a CNF:

1. Eliminate implication and equivalence:

$$\alpha \supset \beta \mapsto \neg\alpha \vee \beta$$

$$\alpha \equiv \beta \mapsto (\alpha \supset \beta) \wedge (\beta \supset \alpha)$$

2. **move $\neg$ inwards:**

$$\neg(\alpha \vee \beta) \mapsto \neg\alpha \wedge \neg\beta, \quad \neg(\alpha \wedge \beta) \mapsto \neg\alpha \vee \neg\beta$$
$$\neg\neg\alpha \mapsto \alpha \quad \neg\forall x\ \alpha \mapsto \exists x\ \neg\alpha, \quad \neg\exists x\ \alpha \mapsto \forall x\ \neg\alpha.$$

3. **standardize variables:** for sentences like $\forall x P(x) \vee \exists x Q(x)$ that use the same variable twice, change the name of one of the variables, e.g. $\forall x P(x) \vee \exists y Q(y)$. So each quantifier gets a unique variable name. This avoids confusion later when we drop the quantifiers.
4. **Skolemize:** Skolemization is the process of removing existential quantifiers by Skolem functions. See the next slide.
5. **move quantifiers to the front:** using rules $\forall x \alpha \wedge \beta \mapsto \forall x(\alpha \wedge \beta)$ and $\forall x \alpha \vee \beta \mapsto \forall x(\alpha \vee \beta)$, where β does not use x.

Conversion to CNF (Cont'd)

6. **drop quantifiers:** so now variables are implicitly universally quantified.
7. **distribute $\vee$ over $\wedge$:**

$$(\alpha \wedge \beta) \vee \gamma \mapsto (\alpha \vee \gamma) \wedge (\beta \vee \gamma).$$

8. **flatten nested conjunctions and disjunctions**:

$$(\alpha \wedge \beta) \wedge \gamma \mapsto \alpha \wedge \beta \wedge \gamma.$$

$$(\alpha \vee \beta) \vee \gamma \mapsto \alpha \vee \beta \vee \gamma.$$

9. **Remove conjunctions to get a set of clauses**.

Skolemization

A sentence such as $\exists x \forall y \exists z P(x, y, z)$ says: "there is an x such that for all y there is a z such that P(x,y,z) holds".
Suppose we use the constant a to name the x, then the sentence becomes $\forall y \exists z P(a, y, z)$. We can similarly eliminate $\exists z$ but notice that the existence of such z is depended on the y: for different y, we may have a different z. So we use a function, called *Skolem function:* $\forall y P(a, y, sk(y))$.
In general:

$$\forall x_1(...\forall x_2(...\forall x_n(...\exists y[...y...]...)...)...)$$

is replaced by

$$\forall x_1(...\forall x_2(...\forall x_n(...[...f(x_1, x_2, ..., x_n)...]...)...)...)$$

where f is a new function symbol that appears nowhere else.
Skolemization does not preserve equivalence (all of our other rules in transforming a sentence to its CNF do): $\not\models \exists x P(x) \equiv P(a)$. But it does preserve satisfiability: α is satisfiable iff α' is, where α' is the result of skolemization.
This is sufficient for resolution!

Examples

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

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

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

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

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

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

$\forall x\exists y Less(x,y) \wedge \forall x\exists y(double(x) = y) \mapsto$ (standardize variables)

$\forall x\exists y Less(x,y) \wedge \forall x_1\exists y_1(double(x_1) = y_1) \mapsto$ (skolemize)

$\forall x Less(x, sk1(x)) \wedge \forall x_1(double(x_1) = sk2(x_1)) \mapsto$ (move quantifiers left)

$\forall x\forall x_1[Less(x, sk1(x)) \wedge double(x_1) = sk2(x_1)] \mapsto$ (drop quantifiers)

$Less(x, sk1(x)) \wedge double(x_1) = sk2(x_1)$

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

Knowledge Engineering

Baisc ideas:

1. Write down the knowledge required for solving the problem.
2. Pose the problem as a query and try to infer it from the knowledge base (KB) that you constructed in step 1.

An Example

Example: the wumpus world.

KB:

$$A_{11} \wedge East_A \wedge W_{21} \supset \neg Forward,$$
$$A_{11} \wedge G_{11} \supset Grab, ...$$
$$\neg S_{11}, \neg S_{21}, S_{12},$$
$$\neg S_{11} \supset \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{21},$$
$$\neg S_{21} \supset \neg W_{11} \wedge \neg W_{21} \wedge \neg W_{22} \wedge \neg W_{31},$$
$$\neg S_{12} \supset \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{22} \wedge \neg W_{13},$$
$$S_{12} \supset W_{13} \vee W_{12} \vee W_{22} \vee W_{11}, ...$$

Queries:

- Shall I go forward: $KB \models Forward$?
- Shall I avoid going forward: $KB \models \neg Forward$?
- Shall I grab the gold: $KB \models Grab$?

Knowledge Engineering - Five Steps

- Must have enough knowledge about the domain in question.
- Must have enough knowledge about the representation language.
- Must have enough knowledge about the inference procedure to ensure that queries can be answered in a reasonable amount of time.

Five steps in successful knowledge engineering:

- decide what to talk about - what are the relevant objects and facts.
- decide on a vocabulary - constants, functions, predicates.
- encode general knowledge about the domain.
- encode knowledge that is specific to the problem in hand.
- pose queries to the inference procedure and get answers.

The Queens Problem

Problem:

Arrange 2 queens on a 3x3 board so that none of them is attacked by others.

- What to talk about: queens, squares.
- Vocabularies:
  - Constants for queens, rows, and columns: $q1, q2, 1, 2, 3$.
  - Ternary predicate $Q(n, i, j)$ - "Queen n is in square (i, j)".
  - Predicate $Diag(i, j, i', j')$ - "the square (i, j) is diagonal to the square (i', j').

The Queens Problem (Cont'd)

General laws:

- "Each queen has to be in one of the squares":

$$\forall n \exists i, j \; Q(n, i, j).$$

- "Each queen can only occupy one square":

$$\forall n, i, j, i', j' \; Q(n, i, j) \wedge Q(n, i', j') \supset i = i' \wedge j = j'.$$

- "No two queens can be in the same column":

$$\forall n, n', i, j, i' \; Q(n, i, j) \wedge Q(n', i', j) \supset n = n'.$$

- "No two queens can be in the same row":

$$\forall n, n', i, j, j' \; Q(n, i, j) \wedge Q(n', i, j') \supset n = n'.$$

- "No two queens can be in the same diagonal":

$$\forall n, n', i, j, j' \; Q(n, i, j) \wedge Q(n', i', j') \wedge Diag(i, j, i', j') \supset n = n'.$$

The Queens Problem (Cont'd)

Problem specific knowledge:

- We have two different queens:

$$\neg q1 = q2.$$

- There are just 2 queens:

$$\forall n, i, j \ Q(n, i, j) \supset n = q1 \vee n = q2.$$

- We have three different rows and columns:

$$\neg 1 = 2 \wedge \neg 2 = 3 \wedge \neg 1 = 3$$

- There are three rows:

$$\forall n, i, j \ Q(n, i, j) \supset i = 1 \vee i = 2 \vee i = 3.$$

- There are three columns:

$$\forall n, i, j \ Q(n, i, j) \supset j = 1 \vee j = 2 \vee j = 3.$$

- Facts about *Diag*:

$$Diag(1, 1, 2, 2), \ Diag(1, 1, 3, 3), \ Diag(1, 2, 2, 1), ...$$

The Queens Problem (Cont'd)

Reasoning by queries

- Do I have to put q1 in (2,2)? $KB \models Q(q1, 2, 2)$?
- If I put q1 in (1,1), do I have to put q2 in (2,3)?

$$KB \models Q(q1, 1, 1) \supset Q(q2, 2, 3)?$$

Reasoning by model construction:

> *Find an interpretation that satisfies all the sentences in KB. If* $Q(q1, i, j) \wedge Q(q2, i', j')$ *is true in this interpretaton, then a solution is to put q1 in (i,j) and q2 in (i',j').*

Reasoning by contradiction:

- Can I put q1 in (1,1) and q2 in (1,2)?

$$KB \cup \{Q(q1, 1, 1), Q(q2, 1, 2)\} \models \mathit{false}?$$

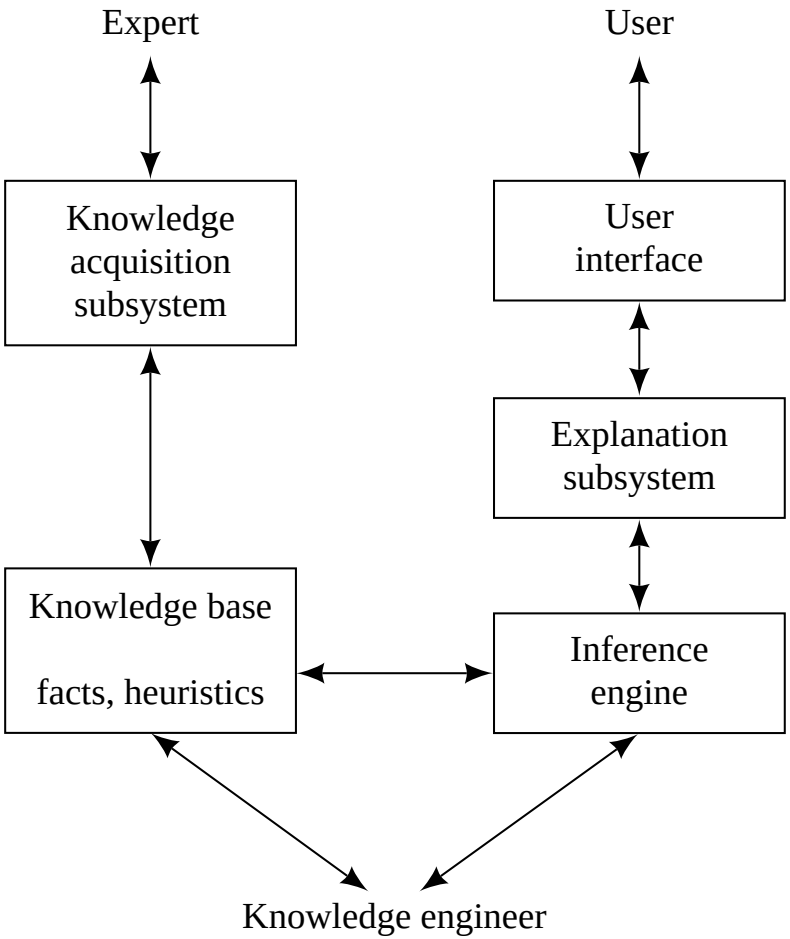
- Can I put q1 in (1,1) and q2 in (2,3)?

$$KB \cup \{Q(q1, 1, 1), Q(q2, 2, 3)\} \models \mathit{false}?$$

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



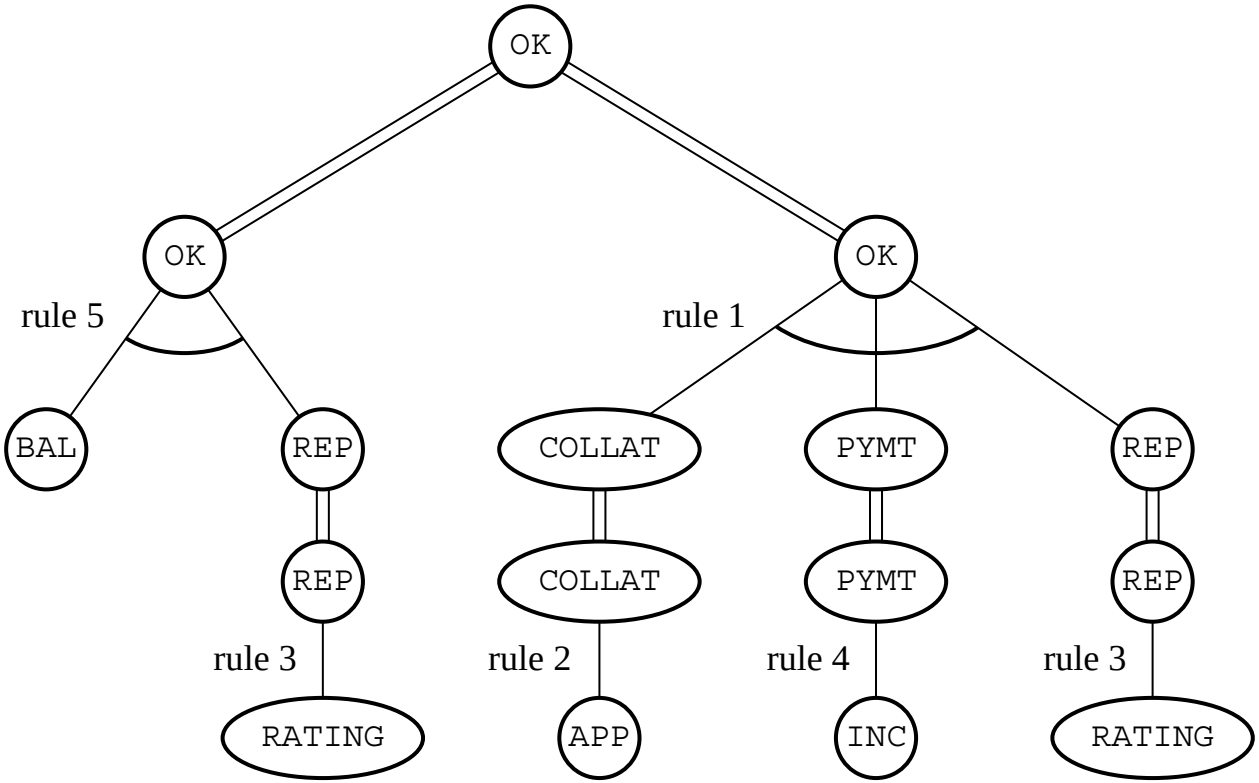
An Example

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 applciant has an excellent balance sheet.

Search Tree



Uncertanty in Rules

Often, experts have only uncertain rules. For instance, in MYCIN, a rule could be:

if (a) the infection is primary-bacteremia, and
(b) the site of the culture is one of the sterile sites, and
(c) the suspected portal of entry is the gastrointestinal tract
then there is evidence (.7) that the infection is bateroid

Here .7 is called certanty factor in MYCIN. Now a better way to deal with uncertainty is by using Bayes networks.

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 algoritm* 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 aready cover by it, and continue like above, until no more positive instances are left with.