Lecture 20
Logical Agents

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Slides by Stuart Russell and Peter Norvig

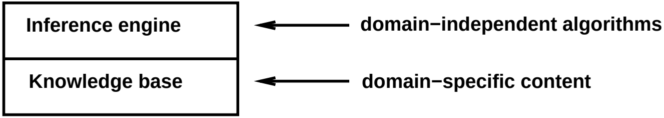
Course Overview

- ✔ Introduction
  - ✔ Artificial Intelligence
  - ✔ Intelligent Agents
- ✔ Search
  - ✔ Uninformed Search
  - ✔ Heuristic Search
- ✔ Uncertain knowledge and Reasoning
  - ✔ Probability and Bayesian approach
  - ✔ Bayesian Networks
  - ✔ Hidden Markov Chains
  - ✔ Kalman Filters
- ✔ Learning
  - ✔ Supervised
    Decision Trees, Neural Networks
    Learning Bayesian Networks
  - ✔ Unsupervised
    EM Algorithm
- ✔ Reinforcement Learning
- ✔ Games and Adversarial Search
  - ✔ Minimax search and Alpha-beta pruning
  - ✔ Multiagent search
- Knowledge representation and Reasoning
  - Propositional logic
  - First order logic
  - Inference
  - Plannning

Outline

Knowledge bases

Knowledge base = set of sentences in a **formal** language



Declarative approach to building an agent (or other system):
    Tell it what it needs to know

Then it can Ask itself what to do—answers should follow from the KB

Agents can be viewed at the knowledge level
    i.e., **what they know**, regardless of how implemented

Or at the implementation level
    i.e., data structures in KB and algorithms that manipulate them

A simple knowledge-based agent

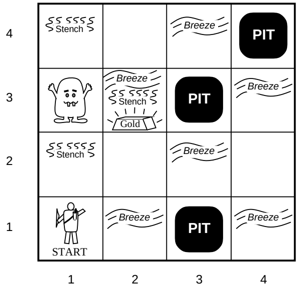
```
function KB-Agent( percept) returns an action
    static: KB, a knowledge base
            t, a counter, initially 0, indicating time

    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

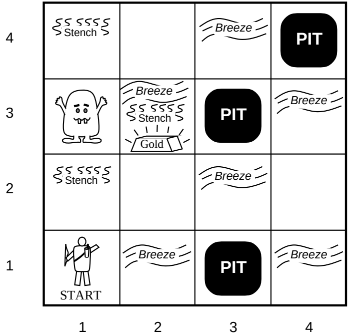
Wumpus World PEAS description

Performance measure
gold +1000, death -1000
-1 per step, -10 for using the arrow
Environment
Squares adjacent to wumpus are smelly
Squares adjacent to pit are breezy
Glitter iff gold is in the same square
Shooting kills wumpus if you are facing it
Shooting uses up the only arrow
Grabbing picks up gold if in same square
Releasing drops the gold in same square
Actuators *LeftTurn*, *RightTurn*,
Forward, *Grab*, *Release*, *Shoot*
Sensors *Breeze*, *Glitter*, *Smell*

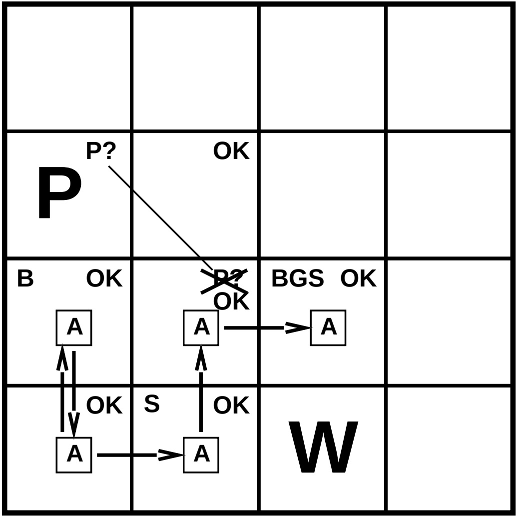


Wumpus world – Properties

Fully vs Partially observable??
No—only local perception
Deterministic vs Stochastic??
Deterministic—outcomes exactly specified
Episodic vs Sequential??
sequential at the level of actions
Static vs Dynamic??
Static—Wumpus and Pits do not move
Discrete vs Continous??
Discrete
Single-agent vs Multi-Agent??
Single—Wumpus is essentially a natural feature



Exploring a wumpus world



Outline

Logic in general

Logics are formal languages for representing information
such that conclusions can be drawn

Syntax defines the sentences in the language

Semantics define the "meaning" of sentences;
i.e., define truth of a sentence in a 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 iff the number $x + 2$ is no less than the number y

$x + 2 \geq y$ is true in a world where $x = 7,\ y = 1$

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

Entailment

Entailment means that one thing **follows from** another:

$$KB \models \alpha$$

Knowledge base KB entails sentence α
if and only if
α is true in all worlds where KB is true

E.g., the KB containing "OB Won" and "KBH won"
entails "Either OB or KBH won"

Entailment is a relationship between sentences (i.e., **syntax**)
that is based on **semantics**

Models

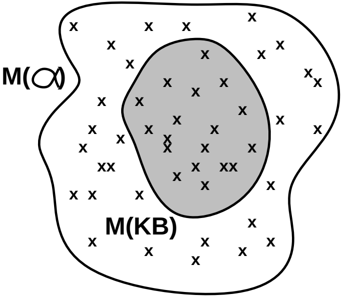
Logicians typically think in terms of models, which are formally structured worlds with respect to which truth can be evaluated

We say m is a model of a sentence α if α is true in m

$M(\alpha)$ is the set of all models of α

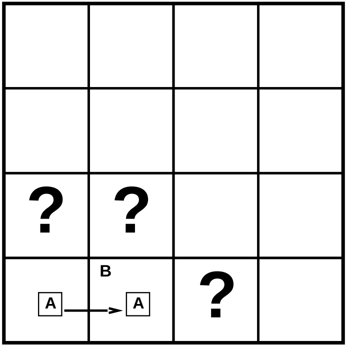
Then $KB \models \alpha$ if and only if $M(KB) \subseteq M(\alpha)$

E.g. KB = OB won and FCK won
α = OB won



Entailment in the wumpus world

Situation after detecting nothing in [1,1], moving right, breeze in [2,1]



Consider possible models for ?s assuming only pits

3 Boolean choices $\implies$ 8 possible models

Inference

$KB \vdash_i \alpha$ = sentence α can be derived from KB by procedure i

Soundness: i is sound if
whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$

Completeness: i is complete if
whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$

Preview: we will define a logic (first-order logic) which is expressive enough to say almost anything of interest, and for which there exists a sound and complete inference procedure.

That is, the procedure will answer any question whose answer follows from what is known by the KB.

Part I

Outline

Outline

Propositional logic: Syntax

Propositional logic is the simplest logic—illustrates basic ideas

The proposition symbols P_1, P_2 etc are sentences

If S is a sentence, $\neg S$ is a sentence (negation)
If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (conjunction)
If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (disjunction)
If S_1 and S_2 are sentences, $S_1 \implies S_2$ is a sentence (implication)
If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)

Propositional logic: Semantics

Each model specifies true/false for each proposition symbol

E.g.	$P_{1,2}$	$P_{2,2}$	$P_{3,1}$
	true	*true*	*false*

(With these symbols, 8 possible models, can be enumerated automatically.)

Rules for evaluating truth with respect to a model m:

$\neg S$	is true iff	S	is false		
$S_1 \wedge S_2$	is true iff	S_1	is true **and**	S_2	is true
$S_1 \vee S_2$	is true iff	S_1	is true **or**	S_2	is true
$S_1 \implies S_2$	is true iff	S_1	is false **or**	S_2	is true
i.e.,	is false iff	S_1	is true **and**	S_2	is false
$S_1 \Leftrightarrow S_2$	is true iff	$S_1 \implies S_2$	is true **and**	$S_2 \implies S_1$	is true

Simple recursive process evaluates an arbitrary sentence, e.g.,

$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = true \wedge (false \vee true) = true \wedge true = true$

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*

Wumpus world sentences

Let $P_{i,j}$ be true if there is a pit in $[i, j]$.
Let $B_{i,j}$ be true if there is a breeze in $[i, j]$.

$$R_1 : \neg P_{1,1}$$
$$R_2 : \neg B_{1,1}$$
$$R_3 : \quad B_{2,1}$$

"Pits cause breezes in adjacent squares"
"A square is breezy **if and only if** there is an adjacent pit"

$$R_4 : B_{1,1} \quad \Leftrightarrow \quad (P_{1,2} \vee P_{2,1})$$
$$R_5 : B_{2,1} \quad \Leftrightarrow \quad (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

Truth tables for inference

$KB \vdash \alpha$

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
false	*true*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	<u>*true*</u>
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	<u>*true*</u>
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	<u>*true*</u>
false	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

Enumerate rows (different assignments to symbols),
if KB is true in row, check that α is too

Inference by enumeration

Depth-first enumeration of all models is sound and complete

$O(2^n)$ for n symbols; problem is **co-NP-complete**

A problem Π is a member of co-NP if and only if its complement $\overline{\Pi}$ is in the complexity class NP.
Class of problems for which efficiently verifiable proofs of no instances, sometimes called counterexamples, exist.

Is α true under KB? To give an answer "no" it is enough to provide a couterexample, which is easily verifiable.

Logical equivalence

Two sentences are logically equivalent iff true in same models:
$\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$

$$
\begin{array}{rcll}
(\alpha \land \beta) & \equiv & (\beta \land \alpha) & \text{commutativity of } \land \\
(\alpha \lor \beta) & \equiv & (\beta \lor \alpha) & \text{commutativity of } \lor \\
((\alpha \land \beta) \land \gamma) & \equiv & (\alpha \land (\beta \land \gamma)) & \text{associativity of } \land \\
((\alpha \lor \beta) \lor \gamma) & \equiv & (\alpha \lor (\beta \lor \gamma)) & \text{associativity of } \lor \\
\neg(\neg\alpha) & \equiv & \alpha & \text{double-negation elimination} \\
(\alpha \implies \beta) & \equiv & (\neg\beta \implies \neg\alpha) & \text{contraposition} \\
(\alpha \implies \beta) & \equiv & (\neg\alpha \lor \beta) & \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) & \equiv & ((\alpha \implies \beta) \land (\beta \implies \alpha)) & \text{bicond. elimination} \\
\neg(\alpha \land \beta) & \equiv & (\neg\alpha \lor \neg\beta) & \text{De Morgan} \\
\neg(\alpha \lor \beta) & \equiv & (\neg\alpha \land \neg\beta) & \text{De Morgan} \\
(\alpha \land (\beta \lor \gamma)) & \equiv & ((\alpha \land \beta) \lor (\alpha \land \gamma)) & \text{distributivity of } \land \text{ over } \lor \\
(\alpha \lor (\beta \land \gamma)) & \equiv & ((\alpha \lor \beta) \land (\alpha \lor \gamma)) & \text{distributivity of } \lor \text{ over } \land
\end{array}
$$

Validity and satisfiability

A sentence is valid if it is true in **all** models,

e.g., $True$, $A \lor \neg A$, $A \implies A$, $(A \land (A \implies B)) \implies B$

Validity is connected to inference via the Deduction Theorem:

$KB \models \alpha$ if and only if $(KB \implies \alpha)$ is valid

A sentence is satisfiable if it is true in **some** model

e.g., $A \lor B$, C

A sentence is unsatisfiable if it is true in **no** models

e.g., $A \land \neg A$

Satisfiability is connected to inference via the following:

$KB \models \alpha$ if and only if $(KB \land \neg\alpha)$ is unsatisfiable

i.e., prove α by *reductio ad absurdum*

Outline

Proof methods

Proof methods divide into (roughly) two kinds:

By resolution (application of inference rules)

- Legitimate (sound) generation of new sentences from old
- Proof = a sequence of inference rule applications
  Can use inference rules as operators in a standard search alg.
- Typically require translation of sentences into a normal form

Model checking

truth table enumeration (always exponential in n)
improved backtracking, e.g., Davis–Putnam–Logemann–Loveland
heuristic search in model space (sound but incomplete)
e.g., min-conflicts-like hill-climbing algorithms

Resolution

Conjunctive Normal Form (CNF—universal)

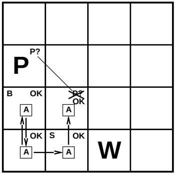
$\textbf{conjunction}$ of $\underbrace{\textbf{disjunctions} \text{ of } \textbf{literals}}_{\textbf{clauses}}$

E.g., $(A \lor \neg B) \land (B \lor \neg C \lor \neg D)$

Resolution inference rule (for CNF): complete for propositional logic

$$\frac{\ell_1 \lor \cdots \lor \ell_k, \qquad m_1 \lor \cdots \lor m_n}{\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n}$$

where ℓ_i and m_j are complementary literals. E.g.:



$$\frac{P_{1,3} \lor P_{2,2}, \qquad \neg P_{2,2}}{P_{1,3}}$$

Resolution is sound and complete for propositional logic $\rightsquigarrow$ can decide whether $\alpha \models \beta$

Conversion to CNF

Resolution rule applies only to clauses (disjunction of literals)
Every sentence in PL is logically equivalent to a conjunction of clauses:

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \implies \beta) \wedge (\beta \implies \alpha)$.

   $(B_{1,1} \implies (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \implies B_{1,1})$

2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$.

   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$

3. Move $\neg$ inwards using de Morgan's rules and double-negation:

   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$

4. Apply distributivity law ($\vee$ over $\wedge$) and flatten:

   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

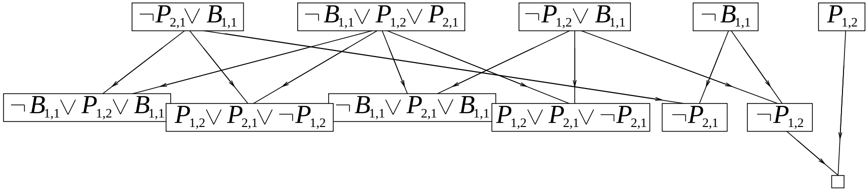
Resolution algorithm

$KB \models \alpha$ Proof by contradiction, i.e., show $KB \wedge \neg\alpha$ unsatisfiable

```
function PL-Resolution(KB, α) returns true or false
   inputs: KB, the knowledge base, a sentence in propositional logic
           α, the query, a sentence in propositional logic

   clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
   new ← { }
   loop do
       for each Ci, Cj in clauses do
           resolvents ← PL-Resolve(Ci, Cj)
           if resolvents contains the empty clause then return true
           new ← new ∪ resolvents
       if new ⊆ clauses then return false
       clauses ← clauses ∪ new
```

Resolution example

$KB = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$
$\alpha = \neg P_{1,2}$



Completeness of Resolution

Theorem

Ground Resolution Theorem
If a set of clauses is unsatisfiable, then the resolution closure of those clauses contains the empty clauses

Proof. by contraposition $RC(S)$ does not contain empty clause $\implies$ S is satisfiable.
Construct a model for S with sutiable ttruth values for $P_1, \ldots, P_k$ as follows

- assign false to P_i if there is a clause in $RC(S)$ containing literal $\neg P_i$ and all its other literals being false under the current assignment
- otherwise, assign P_i true.

Model Checking

$KB \models \alpha$

Forward and backward chaining

Horn Form (restricted)

KB = **conjunction** of **Horn clauses**

Horn clause =

- ♢ proposition symbol; or
- ♢ (conjunction of symbols) $\implies$ symbol

E.g., $C \wedge (B \implies A) \wedge (C \wedge D \implies B)$

Modus Ponens (for Horn Form): complete for Horn KBs

$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \cdots \wedge \alpha_n \implies \beta}{\beta}$$

Can be used with forward chaining or backward chaining.
These algorithms are very natural and run in **linear** time

Forward chaining

Idea: fire any rule whose premises are satisfied in the *KB*,
add its conclusion to the *KB*, until query is found

$P \implies Q$

$L \wedge M \implies P$

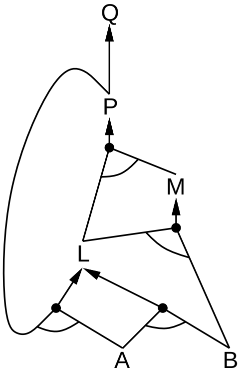
$B \wedge L \implies M$

$A \wedge P \implies L$

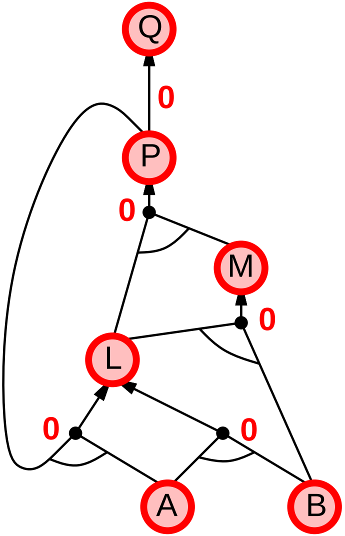
$A \wedge B \implies L$

A

B



Forward chaining example



$P \implies Q$

$L \wedge M \implies P$

$B \wedge L \implies M$

$A \wedge P \implies L$

$A \wedge B \implies L$

A

B

Proof of completeness

FC derives every atomic sentence that is entailed by KB

- FC reaches a fixed point where no new atomic sentences are derived
- Consider the final state as a model m, assigning true/false to symbols
- Every clause in the original KB is true in m
  Proof: Suppose a clause $a_1 \wedge \ldots \wedge a_k \Rightarrow b$ is false in m
  Then $a_1 \wedge \ldots \wedge a_k$ is true in m and b is false in m
  Therefore the algorithm has not reached a fixed point!
- Hence m is a model of KB
- If $KB \models q$, q is true in **every** model of KB, including m

General idea: construct any model of KB by sound inference, check α

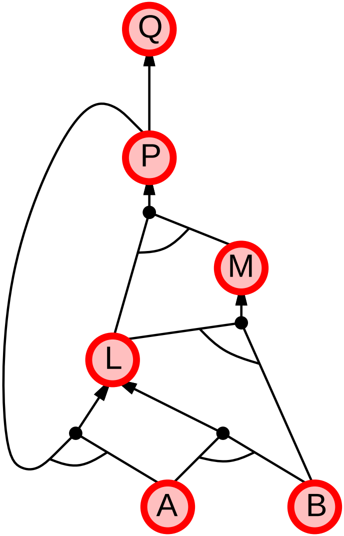
Backward chaining

Idea: work backwards from the query q:
  to prove q by BC,
    check if q is known already, or
    prove by BC all premises of some rule concluding q

Avoid loops: check if new subgoal is already on the goal stack

Avoid repeated work: check if new subgoal
  1) has already been proved true, or
  2) has already failed

Backward chaining example



Forward vs. backward chaining

FC is data-driven, cf. automatic, unconscious processing,
e.g., object recognition, routine decisions

May do lots of work that is irrelevant to the goal

BC is goal-driven, appropriate for problem-solving,
e.g., Where are my keys? How do I get into a PhD program?

Complexity of BC can be **much less** than linear in size of KB

Model Checking

$KB \models \alpha$

General case:

$KB \models \alpha \equiv KB \wedge \neg\beta$ is unsatisfiable hence find a counter example to $KB \wedge \neg\beta$

Model Checking by Backtracking – DPLL

Davis Putnam Logeman and Loveland (1960-1962)

function DPLL-Satisfiable?(s) **returns** *true* or *false*
    inputs: s, a sentence in propositional logic

    clauses ← the set of clauses in the CNF representation of s
    symbols ← a list of the proposition symbols in s
    return DPLL(*clauses*, *symbols*, [ ])

---

function DPLL(*clauses*, *symbols*, *model*) **returns** *true* or *false*

    if every clause in *clauses* is true in *model* **then return** *true*
    if some clause in *clauses* is false in *model* **then return** *true*
    P, *value* ← Find-Pure-Symbol(*symbols*, *clauses*, *model*)
    if P is non-null **then return** DPLL(*clauses*, *symbols*–P, [$P =$ *value*|*model*])
    P, *value* ← Find-Unit-Clause(*clauses*, *model*)
    if P is non-null **then return** DPLL(*clauses*, *symbols*–P, [$P =$ *value*|*model*])
    P ← First(*symbols*); *rest* ← Rest(*symbols*)
        return DPLL(*clauses*, *rest*, [$P =$ *true*|*model*]) **or** DPLL(*clauses*, *rest*, [$P =$ *false*|*model*])

Model Checking by Local Search

Walksat

```
function WalkSAT(clauses, p, max-flips) returns a satisfying model or failure
    inputs: clauses, a set of clauses in propositional logic
            p, the probability of choosing to do a "random walk" move, typically around 0.5
            max-flips, number of flips allowed before giving up

    model ← a random assignment of true/false to the symbols in clauses
    for i = 1 to max-flips do
        if model satisfies clauses then return model
        clause ← a randomly selected clause from clauses that is false in model
        with probability p flip the value in model of a randomly selected symbol from clause
        else flip whichever symbol in clause maximizes the number of satisfied clauses
    return failure
```

Summary

Logical agents apply inference to a knowledge base
to derive new information and make decisions

Basic concepts of logic:
- syntax: formal structure of sentences
- semantics: truth of sentences wrt models
- entailment: necessary truth of one sentence given another
- inference: deriving sentences from other sentences
- soundess: derivations produce only entailed sentences
- completeness: derivations can produce all entailed sentences

Wumpus world requires the ability to represent partial and negated information, reason by cases, etc.

Forward, backward chaining are linear-time, complete for Horn clauses
Resolution is complete for propositional logic

Propositional logic lacks expressive power

Part II

Outline

Outline

Outline

Outline

- Why FOL?
- Syntax and semantics of FOL
- Fun with sentences
- Wumpus world in FOL

Pros and cons of propositional logic

☺ Propositional logic is **declarative**: pieces of syntax correspond to facts

☺ Propositional logic allows partial/disjunctive/negated information (unlike most data structures and databases)

☺ Propositional logic is **compositional**: meaning of $B_{1,1} \wedge P_{1,2}$ is derived from meaning of $B_{1,1}$ and of $P_{1,2}$

☺ Meaning in propositional logic is **context-independent** (unlike natural language, where meaning depends on context)

☹ Propositional logic has very limited expressive power (unlike natural language)
E.g., cannot say "pits cause breezes in adjacent squares"
except by writing one sentence for each square

First-order logic

Whereas propositional logic assumes world contains **facts**,
first-order logic (like natural language) assumes the world contains

- Objects: people, houses, numbers, theories, Ronald McDonald, colors, baseball games, wars, centuries ...
- Relations/Predicates: red, round, bogus, prime, multistoried ..., brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, likes, friends, ...
- Functions: father of, best friend, successor, one more than, times, end of ...

Logics in general

Language	Ontological Commitment	Epistemological Commitment
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief
Fuzzy logic	facts + degree of truth	known interval value

Syntax of FOL: Basic elements

Constants	$KingJohn,\ 2,\ UCB, \ldots$
Variables	$x,\ y,\ a,\ b, \ldots$
Functions	$Sqrt,\ Father \ldots$
Predicates	$BrotherOf,\ >, \ldots$
Connectives	$\wedge\ \vee\ \neg\ \implies\ \Leftrightarrow$
Equality	$=$
Quantifiers	$\forall\ \exists$

Note: constants, variables, predicates are distinguished typically by the case of the letters. Every system/book has differnt conventions in this regard. PROLOG: costants in lower case and variables in upper case.

Atomic sentences

$$\text{Atomic sentence} \quad = \quad \mathit{predicate}(\mathit{term}_1, \ldots, \mathit{term}_n)$$
$$\text{or } \mathit{term}_1 = \mathit{term}_2$$

$$\text{Term} \quad = \quad \mathit{function}(\mathit{term}_1, \ldots, \mathit{term}_n)$$
$$\text{or } \mathit{constant} \text{ or } \mathit{variable}$$

E.g., $\mathit{Brother}(\mathit{KingJohn}, \mathit{RichardTheLionheart})$

$> (\mathit{Length}(\mathit{LeftLegOf}(\mathit{Richard})), \mathit{Length}(\mathit{LeftLegOf}(\mathit{KingJohn})))$

But: E.g., $\mathit{Plus}(2, 3)$ is a function, not an atomic sentence.

Complex sentences

Complex sentences are made from atomic sentences using connectives

$$\neg S, \quad S_1 \wedge S_2, \quad S_1 \vee S_2, \quad S_1 \implies S_2, \quad S_1 \Leftrightarrow S_2$$

E.g. $Sibling(KingJohn, Richard) \implies Sibling(Richard, KingJohn)$

$>(1,2) \vee \leq(1,2)$

$>(1,2) \wedge \neg{>}(1,2)$

E.g., $Equal(Plus(2,3), Seven))$

Semantics in first-order logic

Sentences are true with respect to an interpretation over a domain D.

DEFINITION

INTERPRETATION

Let the domain D be a nonempty set.

An *interpretation* over D is an assignment of the entities of D to each of the constant, variable, predicate, and function symbols of a predicate calculus expression, such that:

1. Each constant is assigned an element of D.
2. Each variable is assigned to a nonempty subset of D; these are the allowable substitutions for that variable.
3. Each function f of arity m is defined on m arguments of D and defines a mapping from D^m into D.
4. Each predicate p of arity n is defined on n arguments from D and defines a mapping from D^n into {T, F}.

Truth Value Assignment

Symbols in FOL are assigned values from the domain D as determined by the interpretation. Each precise assignment is a model

An atomic sentence $predicate(term_1, \ldots, term_n)$ is true
iff the objects referred to by $term_1, \ldots, term_n$
are in the relation referred to by *predicate* in the interpretation
Example:
Consider the interpretation in which

Richard $\rightarrow$ Richard the Lionheart
John $\rightarrow$ the evil King John
Brother $\rightarrow$ the brotherhood relation

Under this interpretation, $Brother(Richard, John)$ is true
just in case Richard the Lionheart and the evil King John
are in the brotherhood relation in the model (the assignment of values of the world to objects according to the interpretation)

Models for FOL: Lots!

Entailment in propositional logic can be computed by enumerating models
We **can** enumerate the FOL models for a given KB vocabulary.

But:

Sentences with quantifiers:
Eg. $\forall X(p(X) \vee q(Y)) \implies r(X))$
It requires checking truth by substituting all values that X can take in the subset of D assigned to X in the interpretation

Since the set maybe infinite predicate calculus is said to be undecidable

Existential quantifiers are not easier to check

Universal quantification

$\forall \langle variables \rangle \ \ \langle sentence \rangle$

Everyone at Berkeley is smart:
$\forall x \ \ At(x, Berkeley) \implies Smart(x)$

$\forall x \ \ P$ is true in a model iff P is true with x being **each** possible object in the model
(**Roughly** speaking, equivalent to the conjunction of instantiations of P)

$$
\begin{array}{ll}
 & (At(KingJohn, Berkeley) \implies Smart(KingJohn)) \\
\wedge & (At(Richard, Berkeley) \implies Smart(Richard)) \\
\wedge & (At(Berkeley, Berkeley) \implies Smart(Berkeley)) \\
\wedge & \ldots
\end{array}
$$

Note: quantifiers are only on objects and variables, not on predicates and functions. This is done in higher order logic.
Eg.: $\forall (Likes) Likes(Geroge, Kate)$

A common mistake to avoid

Typically, $\implies$ is the main connective with $\forall$

Common mistake: using $\land$ as the main connective with $\forall$:

$$\forall x \;\; At(x, Berkeley) \land Smart(x)$$

means "Everyone is at Berkeley and everyone is smart"

Existential quantification

$\exists \langle \mathit{variables} \rangle \; \langle \mathit{sentence} \rangle$

Someone at Stanford is smart:
$\exists x \;\; At(x, Stanford) \wedge Smart(x)$

$\exists x \;\; P$ is true in a model iff P is true with x being **some** possible object in the model
(**Roughly** speaking, equivalent to the disjunction of instantiations of P)

$$
\begin{array}{ll}
 & (At(KingJohn, Stanford) \wedge Smart(KingJohn)) \\
\vee & (At(Richard, Stanford) \wedge Smart(Richard)) \\
\vee & (At(Stanford, Stanford) \wedge Smart(Stanford)) \\
\vee & \ldots
\end{array}
$$

Another common mistake to avoid

Typically, $\wedge$ is the main connective with $\exists$

Common mistake: using $\implies$ as the main connective with $\exists$:

$$\exists x \;\; At(x, Stanford) \implies Smart(x)$$

is true if there is anyone who is not at Stanford!

Properties of quantifiers

- $\forall x \; \forall y$ is the same as $\forall y \; \forall x$
- $\exists x \; \exists y$ is the same as $\exists y \; \exists x$
- $\exists x \; \forall y$ is **not** the same as $\forall y \; \exists x$
- $\exists x \; \forall y \; Loves(x, y)$
  "There is a person who loves everyone in the world"
  $\forall y \; \exists x \; Loves(x, y)$
  "Everyone in the world is loved by at least one person"
- Quantifier duality: each can be expressed using the other
  $\forall x \; Likes(x, IceCream) \qquad \neg\exists x \; \neg Likes(x, IceCream)$
  $\exists x \; Likes(x, Broccoli) \qquad \neg\forall x \; \neg Likes(x, Broccoli)$

Exercise

Translating natural language in FOL

Brothers are siblings
$\forall x, y \;\; Brother(x, y) \implies Sibling(x, y)$.

"Sibling" is symmetric
$\forall x, y \;\; Sibling(x, y) \Leftrightarrow Sibling(y, x)$.

One's mother is one's female parent
$\forall x, y \;\; Mother(x, y) \Leftrightarrow (Female(x) \wedge Parent(x, y))$.

A first cousin is a child of a parent's sibling
$\forall x, y \;\; FirstCousin(x, y) \Leftrightarrow \exists\, p, ps \;\; Parent(p, x) \wedge Sibling(ps, p) \wedge Parent(ps, y)$

Note: there is not an unique way of translating
If it does not rain on Monday, Tom will go to the mountains
$\neg weather(rain, mountain) \implies go(tom, mountains)$

Equality

$term_1 = term_2$ is true under a given interpretation
if and only if $term_1$ and $term_2$ refer to the same object

E.g., $1 = 2$ and $\forall x \ \times(Sqrt(x), Sqrt(x)) = x$ are satisfiable
$2 = 2$ is valid

E.g., definition of (full) *Sibling* in terms of *Parent*:
$\forall x, y \ Sibling(x, y) \Leftrightarrow [\neg(x = y) \wedge \exists m, f \ \neg(m = f) \wedge$
$Parent(m, x) \wedge Parent(f, x) \wedge Parent(m, y) \wedge Parent(f, y)]$

Interacting with FOL KBs

Suppose a wumpus-world agent is using an FOL KB
and perceives a smell and a breeze (but no glitter) at $t = 5$:

$Tell(KB, Percept([Smell, Breeze, None], 5))$
$Ask(KB, \exists\, a\;\; Action(a, 5))$

I.e., does KB entail any particular actions at $t = 5$?

Answer: $Yes,\ \{a/Shoot\}$ $\quad\leftarrow$ substitution (binding list)

Given a sentence S and a substitution σ,
$S\sigma$ denotes the result of plugging σ into S; e.g.,
$S = Smarter(x, y)$
$\sigma = \{x/Hillary, y/Bill\}$
$S\sigma = Smarter(Hillary, Bill)$
$Ask(KB, S)$ returns some/all σ such that $KB \models S\sigma$

Deducing hidden properties

Properties of locations:
$\forall x, t \;\; At(Agent, x, t) \wedge Smelt(t) \implies Smelly(x)$
$\forall x, t \;\; At(Agent, x, t) \wedge Breeze(t) \implies Breezy(x)$

Squares are breezy near a pit:
Diagnostic rule—infer cause from effect

$$\forall y \;\; Breezy(y) \implies \exists x \;\; Pit(x) \wedge Adjacent(x, y)$$

Causal rule—infer effect from cause

$$\forall x, y \;\; Pit(x) \wedge Adjacent(x, y) \implies Breezy(y)$$

Neither of these is complete—e.g., the causal rule doesn't say whether squares far away from pits can be breezy

Definition for the *Breezy* predicate:

$$\forall y \;\; Breezy(y) \Leftrightarrow [\exists x \;\; Pit(x) \wedge Adjacent(x, y)]$$

Outline

Knowledge base for the wumpus world

"Perception"
$\forall\, b, g, t \;\; Percept([Smell, b, g], t) \implies Smelt(t)$
$\forall\, s, b, t \;\; Percept([s, b, Glitter], t) \implies AtGold(t)$

Reflex: $\forall\, t \;\; AtGold(t) \implies Action(Grab, t)$

Reflex with internal state: do we have the gold already?
$\forall\, t \;\; AtGold(t) \land \neg Holding(Gold, t) \implies Action(Grab, t)$

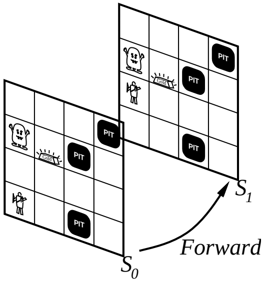
$Holding(Gold, t)$ cannot be observed
$\Rightarrow$ keeping track of change is essential

Keeping track of change

Facts hold in situations, rather than eternally
E.g., $Holding(Gold, Now)$ rather than just $Holding(Gold)$

Situation calculus is one way to represent change in FOL:
Adds a situation argument to each non-eternal predicate
E.g., Now in $Holding(Gold, Now)$ denotes a situation

Situations are connected by the $Result$ function
$Result(a, s)$ is the situation that results from doing a in s



Describing actions I

- "Effect" axiom—describe changes due to action
  $\forall s \;\; AtGold(s) \implies Holding(Gold, Result(Grab, s))$
- "Frame" axiom—describe **non-changes** due to action
  $\forall s \;\; HaveArrow(s) \implies HaveArrow(Result(Grab, s))$

Frame problem: find an elegant way to handle non-change
(a) representation—avoid frame axioms
(b) inference—avoid repeated "copy-overs" to keep track of state

Qualification problem: true descriptions of real actions require endless caveats—what if gold is slippery or nailed down or ...

Ramification problem: real actions have many secondary consequences—what about the dust on the gold, wear and tear on gloves, ...

Describing actions II

Successor-state axioms solve the representational frame problem
Each axiom is "about" a **predicate** (not an action per se):

P true afterwards	$\Leftrightarrow$	[an action made P true
	$\vee$	P true already and no action made P false]

For holding the gold:

$$\forall\, a, s \;\; Holding(Gold, Result(a, s)) \;\Leftrightarrow$$
$$[(a = Grab \wedge AtGold(s))$$
$$\vee\, (Holding(Gold, s) \wedge a \neq Release)]$$

Making plans

Initial condition in KB:
$At(Agent, [1,1], S_0)$
$At(Gold, [1,2], S_0)$

Query: $Ask(KB, \exists\, s \;\; Holding(Gold, s))$
i.e., in what situation will I be holding the gold?

Answer: $\{s/Result(Grab, Result(Forward, S_0))\}$
i.e., go forward and then grab the gold

This assumes that the agent is interested in plans starting at S_0 and that S_0 is the only situation described in the KB

Making plans: A better way

Represent plans as action sequences $p = [a_1, a_2, \ldots, a_n]$

$PlanResult(p, s)$ is the result of executing p in s

Then the query $Ask(KB, \exists\, p \;\; Holding(Gold, PlanResult(p, S_0)))$
has the solution $\{p/[Forward, Grab]\}$

Definition of $PlanResult$ in terms of $Result$:

$\forall\, s \;\; PlanResult([\,], s) = s$

$\forall\, a, p, s \;\; PlanResult([a|p], s) = PlanResult(p, Result(a, s))$

Planning systems are special-purpose reasoners designed to do this type of inference more efficiently than a general-purpose reasoner

Knowledge Engineer

The one just saw is called knowledge engineer process.
It is the production of special-purpose knowledge systems, aka expert systems (eg, in medical diagnosis)

- Identify the task
- Assemble the relevant knowledge
- Decide on a vocabulary of predicates, functions, and constants
- Encode general knowledge about the domain
- Encode a description of the specific problem instance (input data)
  decide what is a constant, a predicate, a function
  leads to definition of the ontology of the domain (what kind of things exist)
- Pose queries to the inference procedure and get answers
- Debug the knowledge base

Summary

First-order logic:
- objects and relations are semantic primitives
- syntax: constants, functions, predicates, equality, quantifiers

Increased expressive power: sufficient to define wumpus world

Situation calculus:
- conventions for describing actions and change in FOL
- can formulate planning as inference on a situation calculus KB