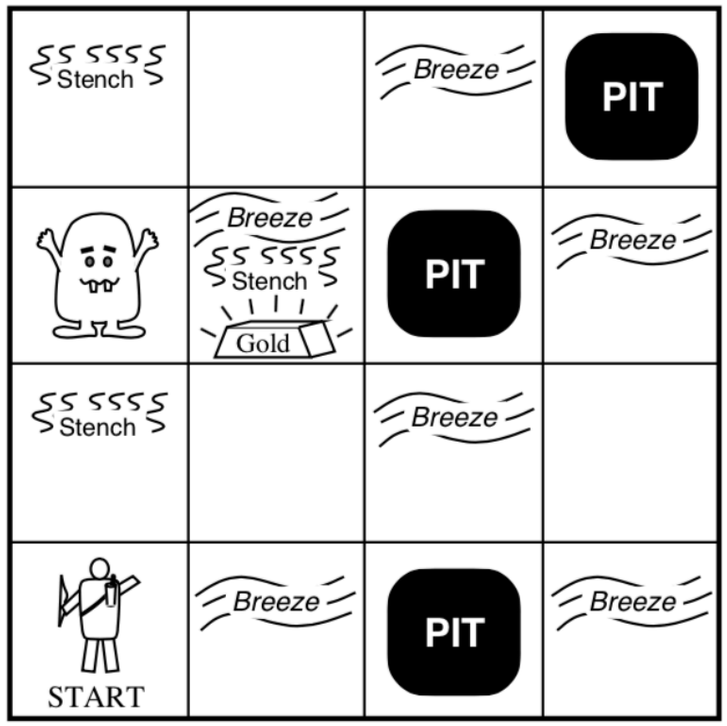
Chapter 7: Logic

- Properties:
  - Partially observable world.
  - Still deterministic, discrete.

Wumpus world



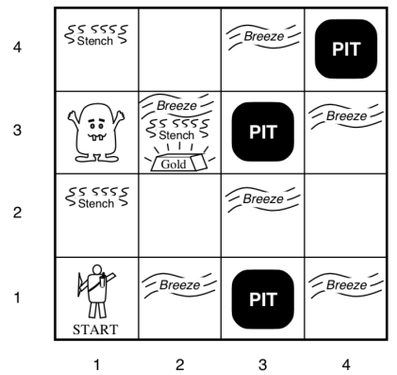
Wumpus world

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 Left turn, Right turn, Forward, Grab, Release, Shoot

Sensors Breeze, Glitter, Smell



Using logic to explore a Wumpus world

- Pit in 3,1; Wumpus in 1,3. Go north first.

Logic in practice: Self-driving car, military, health, law, ...

Logical agents

- Knowledge base = set of sentences in a formal language
- We will focus on two questions:
    - Inference: Given a KB, can we be sure a given sentence is true?
    - Satisfiability: Given a set of sentences, is there a world in which it is true.
    - We will see how we can use an algorithm for satisfiability to do inference.

Logical sentences

We will define a formal language later.
English sentences for now:

- Square 2,2 is breezy
- Squares next to pits are breezy

Models

Model: Specific configuration of the world.

m is a model of a sentence *s* if *m* is true in *s*.

Some model is true; we don't know which one.
A model is kind of like a state.

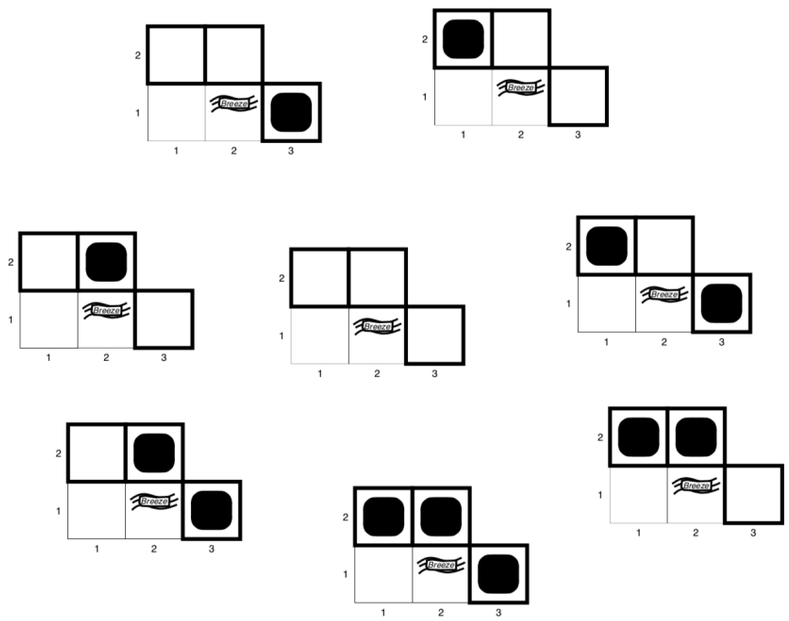
Entailment

Entailment: A |= B means B logically follows from A.
B is true whenever A is true.
Wumpus in 1,1 |= 1,2 is smelly
1,2 is smelly |!= Wumpus in 1,1

Venn diagrams:
- Sentences alpha, beta
- M(alpha): set of models
- m1 in M(alpha), m2 !in M(alpha)
- A |= B iff M(A) subseteq M(B)

Inference: Does KB |= alpha?
Inference by model checking: Enumerate all models, check if M(KB) subseteq M(alpha)

Models



Nothing in 1,1; breeze in 2,1.

KB: wumpus rules + observations
What is M(KB)? left 3 options
alpha1 = 1,2 is safe
Does KB |= alpha1?
alpha2 = 2,2 is safe
Does KB |= alpha2?

Propositional logic

Propositional logic is a system for formally writing logical sentences.

Set of propositional symbols: A, B, C

not A : not A
A ^ B: A and B
A v B: A or B
A => B: A implies B
A <=> B: A iff B

Truth tables

We can build sentences by combining connectives:
not A and (B or C)

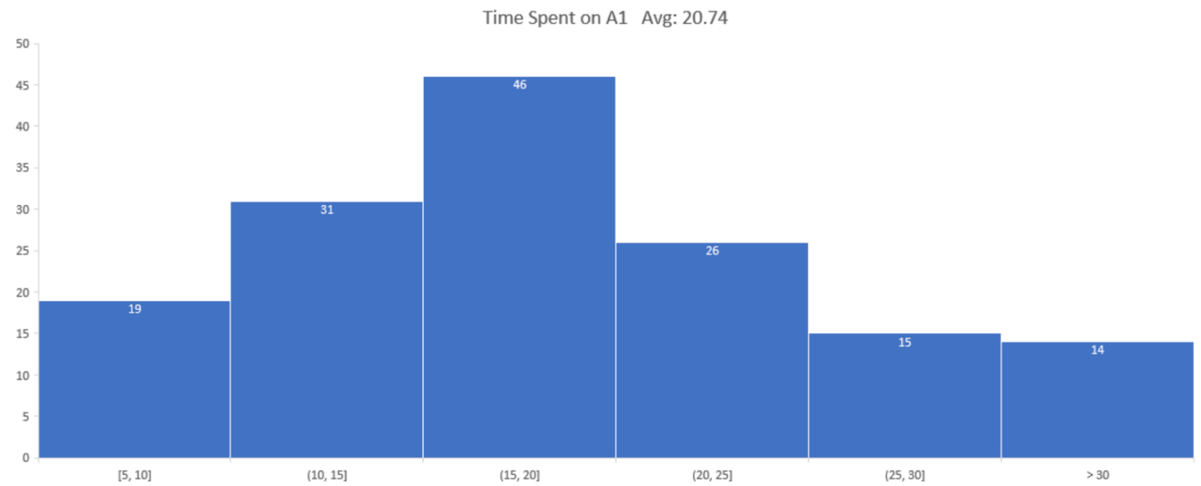
To evaluate whether a model is true in a given sentence, plug in values:
A = true, B = false, C = true: T and (F or T) = T and T = T

There is an order of operations, same as with algebra: not , and , or , implies , iff . (But it's better to use parens for clarity.)

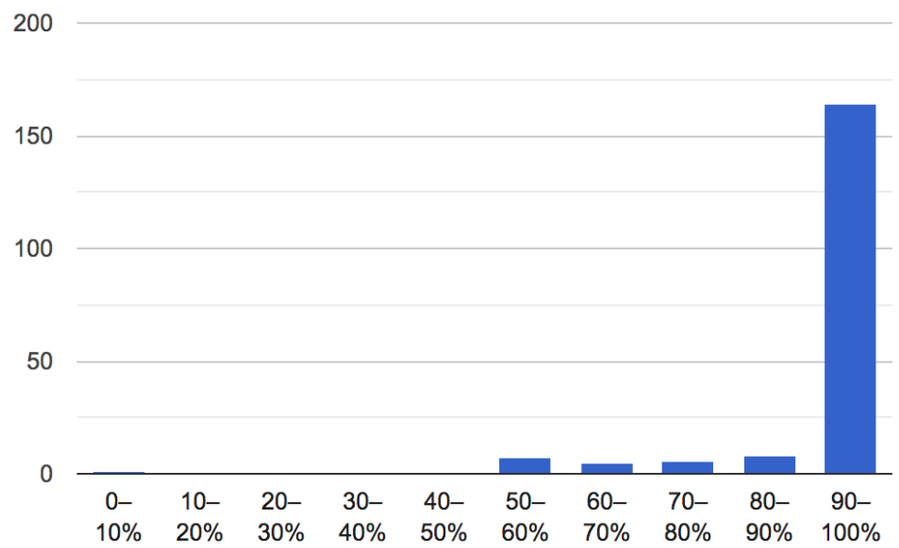
True table for connectives

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*

A1 recap


Time Spent on A1 Avg: 20.74
50
45
40
35
30
25
20
15
10
5
0
19
31
46
26
15
14
[5, 10]
(10, 15]
(15, 20]
(20, 25]
(25, 30]
> 30

A1 recap


200
150
100
50
0
0–10%
10–20%
20–30%
30–40%
40–50%
50–60%
60–70%
70–80%
80–90%
90–100%

Wumpus world rules in propositional logic

"Pits cause breezes in adjacent squares"
B11 <=> (P12 or P21)
P12 <=> (B11 or B22 or B13)

Note iff; if we use normal implies, then we can't prove that a square doesn't have a pit.

Logical equivalence

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

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

We can use equivalence to reduce different expressions to each other, similar to what we would do with algebra.

Validity and satisfiability

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

A sentence is satisfiable if it is true in at least one model.

valid AKA tautology.
Examples of valid sentences?
A or not A
A implies A
(A and (A implies B)) implies B
True

Difference between implication and entailment: They say the same thing, but the implies sentence lives inside

KB |= alpha iff (KB => alpha) is valid

Unsatisfiable AKA contradiction.
Unsatisfiable sentences are true in no models.
Unsatisfiable sentences?
A and not A
False

KB |= alpha iff (KB and not alpha) is unsatisfiable

Conjuctive normal form (CNF)

CNF: conjunction of disjunction of literals
clause: disjunction of literals
Ex: **(A or not B) and (B or not C or not D)**

Any propositional logic expression can be expressed in CNF.
It's easier to write algorithms that work just with CNF.

We usually write CNF expressions as a big list of clauses. The "and"s between clauses are implicit.
Each clause is a list of literals; the "or"s are implicit.

Converting to CNF

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

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

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

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:

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

Logic algorithms

Satisfiability: Is there a model that satisfies a given sentence?

- Model checking
- DPLL
- WalkSAT

Inference: Given a KB, does KB $\models \alpha$?

- Proof by resolution
- Proof by contradiction (using satisfiability algorithm)

Simplest algorithm for satisfiability: Model checking = exhaustively check all models. We will talk about this more later.

We have a KB, we want to know if KB |= alpha
Two kinds:
1) Apply inference rules. Generate new sentences from old until we get alpha. Proof = sequence of rule applications. This is how humans normally do inference.
2) Model checking.

Satisfiability

Satisfiability is just a CSP.
Variables: propositional symbols
Domains: True or False
Constraints: Propositional sentence. In CNF, one constraint per clause.

Simplest algorithm: Use backtracking search. Called model checking.

Better: Backtracking + heuristics, called DPLL algorithm.

Davis–Putnam–Logemann–Loveland (DPLL) algorithm

Backtracking plus two heuristics:

1) Unit clauses. (not C) is a unit clause:
(A or B) and (not C) and (B or C)

Look for unit clauses relative to the existing model. If the current model is (A = false), then (A or B) becomes a unit clauses.
Implementation note: Remove satisfied clauses. Remove literals from clauses that do not satisfy
If you run out of clauses, you're done. If some clause is empty, the given model doesn't work.

2) Pure literals: literals that occur with only one sign. B is a pure literal:
(A or B) and (not C) and (B or C)

Look for pure literals relative to unsatisfied clauses.
(A or not B) and (B or not C) and (C or not A)
current model = {A = true}
Now B is a pure literal.

DPLL algorithm

```
def DPLL(clauses, symbols, model={})
    if every clause is true in model: return model
    if some clause is false in model: return "failure"
    if there is a pure symbol (P, value):
        return DPLL(clauses, symbols - P,
                    model + {symbol = value})
    if there is a unit clause (P, value):
        model[P] = value
        symbols.remove(P)
        return DPLL(clauses, symbols - P,
                    model + {symbol = value})
    P = symbols.remove_first()
    for value in [True,False]
       ret = DPLL(clauses, symbols - P,
                  model + {symbol = value})
       if ret is not "failure": return ret
```

Note that we can return a model that doesn't assign to all symbols. If you want a full assignment, just assign the rest arbitrarily.

Note no "return failure" at the end -- we will always hit one of the top two cases.

WalkSAT

```
def WalkSAT(clauses, symbols)
   model = arbitrary_init(symbols)
   for i in 1:num_iterations:
       clause = pick random unsatisfied clause
       with probability p:
           model = flip a random symbol in clause
       with probability 1-p:
           model = flip the symbol in clause that
                   satisfies the maximum number of
                   clauses
```

WalkSAT = Hill climbing search for satisfiability.

Note that both cases satisfies the clause in question.

Logical inference

We have a KB, we want to know if KB |= alpha
Two kinds:

1) **Proof by contradiction using satisfiability**. If **KB and not alpha** is unsatisfiable, then KB entails alpha.

2) Apply inference rules.
Generate new sentences from old until we get alpha.
Proof = sequence of rule applications.
This is how humans normally do inference.

Rule examples:
A and (A => B) |= B
not A and (A or B) |= B

We can encapsulate all rules into a single rule called resolution.

Resolution

The resolution rule takes two CNF clauses and outputs a new clause that is entailed by the first two.

Resolution rule. Suppose a1 and b1 are complementary literals = the same literal with opposite sign:
(a1 or a2 or a3 or ...)
(b1 or b2 or b3 or ...)
|=
(a2 or a3 or ... or b2 or b3 or ...)
We get rid of the complementary literal and concatenate everything else.

Example:
(A or not B)
(B or not C or not D)
Complementary literal: B / not B
A or not C or not D

Resolution example