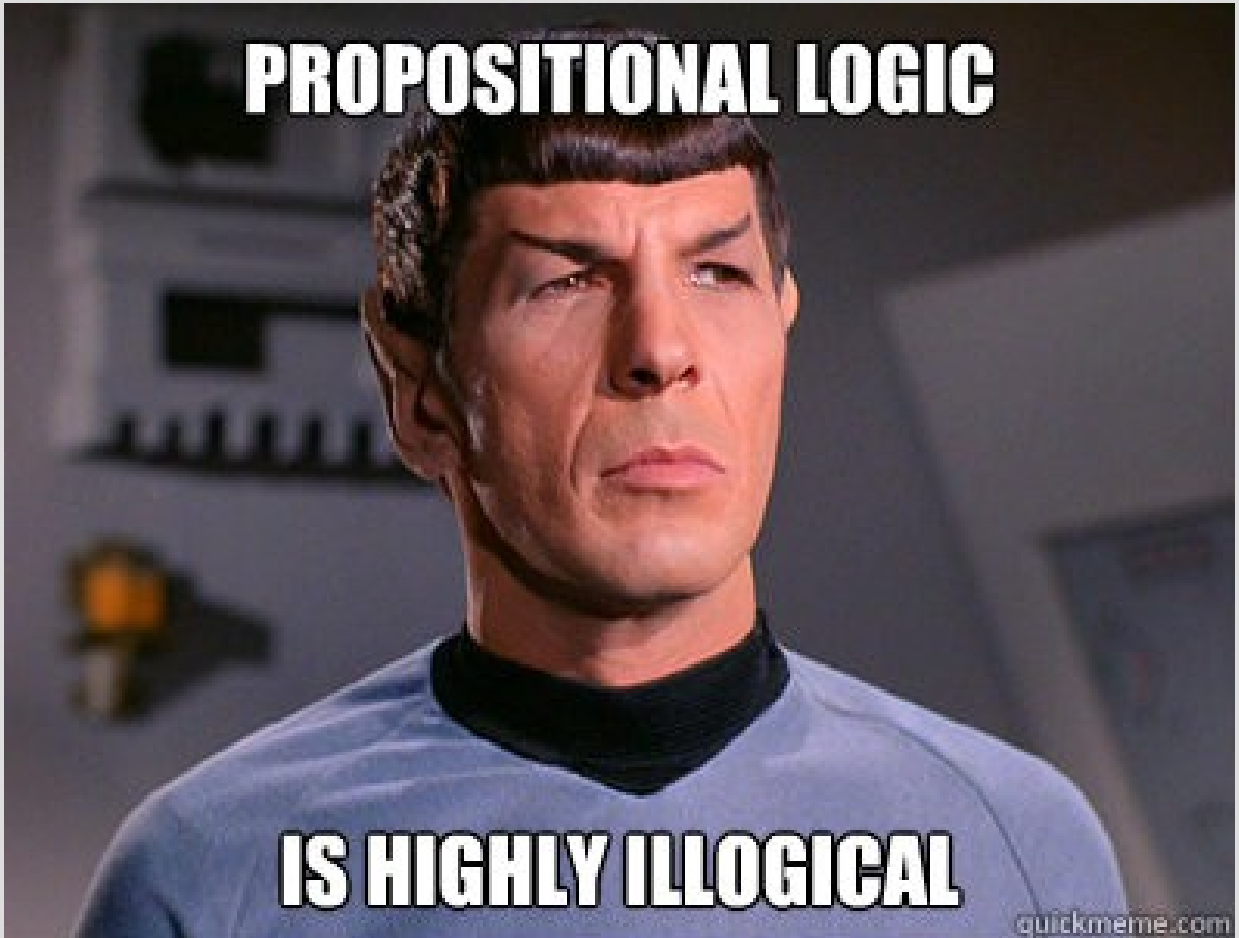
Propositional logic (Ch. 7)



Announcements

HW 3 due Sunday

Representing knowledge

So far we have looked at algorithms to find goals via search, where we are provided with all the knowledge and possibly a heuristic

With CSP we saw how to apply inference to rules to find the goal

Now we will expand more on that and fully represent a knowledge base that we can query and infer proper action from

Representing knowledge

The basic commands the knowledge base needs to have are:

1. Ask() = tells best action
2. Tell() = add information/knowledge to the knowledge base

This new knowledge might be from observations of the environment or from learned rules of the environment

Representing knowledge

There are two ways you can inform agents about the rules of the game:

1. Declarative = use Tell() to inform the agent of all the rules one by one
2. Procedural = code in the fundamental rules and desired behaviors

Typically, there is a mix of both approaches

Representing knowledge

We will follow this general outline:

1. Tell(observations)
2. action = Ask(action options)
3. Tell(picked action)
4. Repeat...

Step 2 can be quite involved and the computer will reason over all of its knowledge

Logic

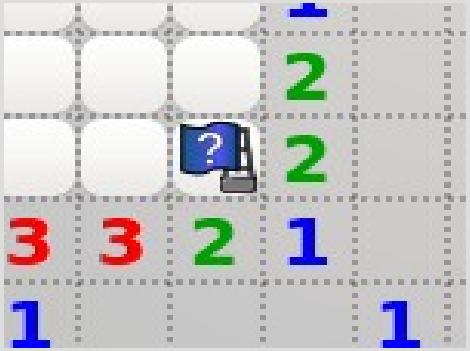
Minesweep?

http://minesweeperonline.com/

Write down any “deductions/rules” you find!

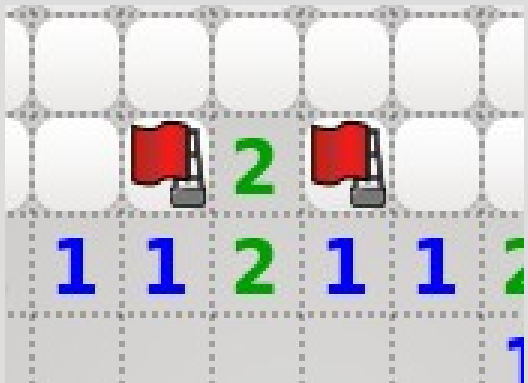
Logic

One example of a simple rule:
The 1 in corner marks flag as a mine



Another rule:
The two can mark the two outer mines if flanked by ones

safe



Logic

The goal is to simply tell the computer about the rules of the game

Then based on what it sees as it plays, it will automatically realize these “safe plays”

This type of reasoning is important in partially observable environments as the agent must often reason on new information

Logic: definitions

A symbol represents a part of the environment (e.g. a minesweep symbol might be if a cell has a mine or not), like math variables

Each single piece of the knowledge base is a sentence involving at least one symbol

A model is a possible assignment of symbols, a “possible outcome” of the environment

Logic: definitions

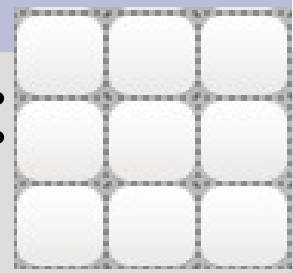
In propositional logic, a symbol is either true or false (as it represents a proposal of a "variable")

If "m" is a model and is "α" a sentence, m satisfies α means α is true in m (also said as "m models α")

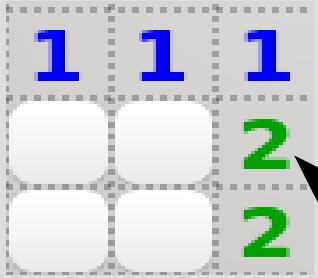
Let M(α) be all models of α

Logic: example

For example, consider a 3x3 minesweep:



After the first play we have:

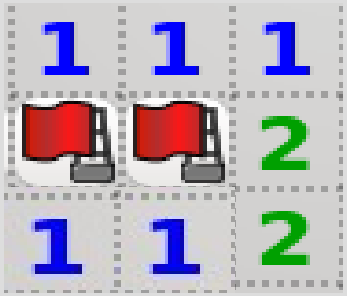


Let us define P2,3,2 as the proposition that row 2, column 3 cell has value 2

After playing the first move, we add to the knowledge base that this proposition is true (this representation has 10^9 states)

Logic: example

Here is one model for the unobserved parts:



This model does **not** satisfy our proposition P2,3,2 as there are only two mines adjacent to row 2, column 3 cell

This means the model does not represent our knowledge base

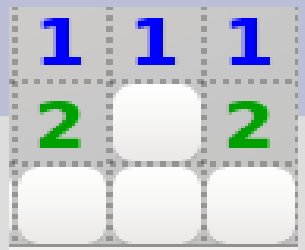
Logic: entailment

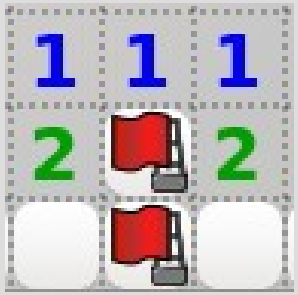
We say α entails β (α ⊨ β) if and only if every model with α true, β is also true

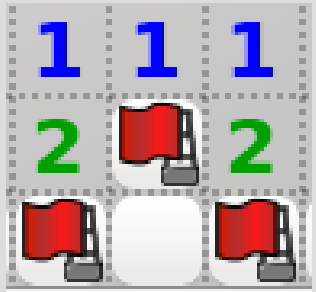
Another definition (mathy):
α ⊨ β if and only if M(α) subset M(β)

This means there are fewer models true with proposition α than β

Logic: entailment

Consider this example: 

There are two valid configurations based on our knowledge base: 

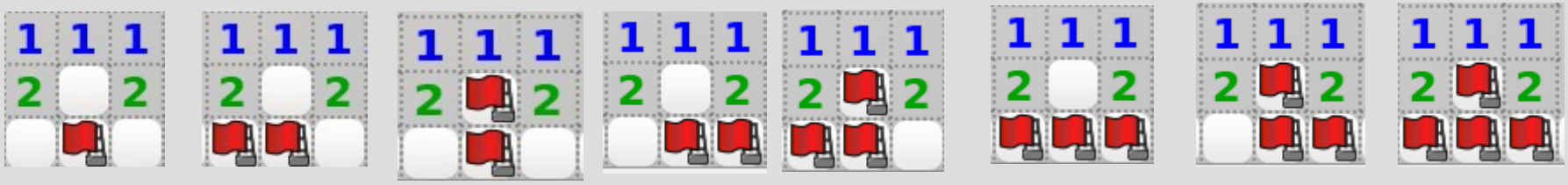


If we let α = mine at (2,2), then in isolation this can mean:

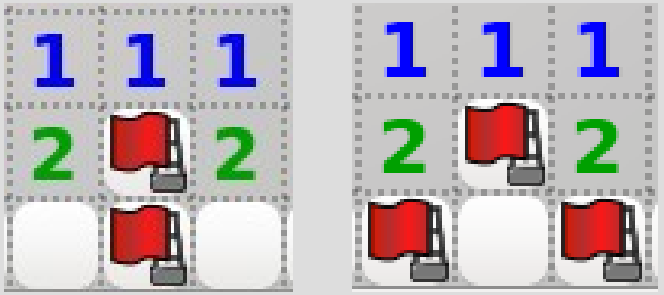
We can see that M(above) subset M(α(below))

Logic: entailment

However, if we let β = mine at (2,3), we get:



M(knowledge base (KB)) is (again):



This is not entailment, as this is not in M(β), thus KB $\not\models$ β (in other words "from the KB, you cannot conclude (2,3) is a mine")

Logic: model checking

Entailment can generate new sentences for our knowledge base(i.e. can add “mine at (2,2)”)

Model checking is when we write out all the actual models (as I did in the last example) then directly check entailment

Although this is exponential, this is true for this type of problem (although some are much worse exponential than others)

Logic: model checking

Model checking is...

1. Preserves truth through inference
2. complete, meaning it can derive any sentence that is entailed (and in finite time)

The “complete” is important as some environments have an infinite number of possible sentences

Logic syntax

In our logic, we allow 5 operations:

$\neg$ = logical negation (i.e. “not” T = F)

$\wedge$ = AND operation

$\vee$ = OR operation (Note: not XOR)

$\Rightarrow$ = “implies” operation

$\Leftrightarrow$ = “if and only if” operation (iff)

The order of operations (wihout parenthesis) is top to bottom

Logic syntax

We mentioned a symbol is P1,3,2 but a literal is either P1,3,2 or ¬ P1,3,2

Two notes:
OR is not XOR (exclusive or), which is not the English “or” (e.g. ordering food)

“implies” only provides information if left hand side is right (e.g. F = cats can fly, B = cats are birds: F implies B is true...)

Logic syntax

Here are the truth tables:

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$
T	T	T	T	T	T
T	F	F	T	F	F
F	T	F	T	T	F
F	F	F	F	T	T

And equivalent laws:

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

Check model

We can make use model checking to make an inference algorithm, much the same way we modified DFS to do backtracking search

1. Enumerate possibilities on a symbol (repeat)
2. Once all symbols are assigned, check if consistent, if not return false (all the way up tree due to recursive call)

Check model

Example: suppose our KB is “P implies Q”
We want to check α = “not P”
Enumerate P: {P = true}, {P = false}
Enumerate Q: {P=T,Q=T}, {P=T,Q=F}, {P=F,Q=T}, {P=F,Q=F}

P	Q	not P	P → Q
T	T	F	T
T	F	F	F
F	T	T	T
F	F	T	T

Consistent?

No! (top row)

“not P” is false when “P implies Q” is true