A complex network diagram with numerous black nodes connected by thin grey lines, forming a dense web-like structure across the entire background.

# Introduction to AI, Spring 2023

## Model Checking

# Logic Review Recap

- Last time we refreshed our memories on logic!
  - Propositional Logic
  - Predicate Logic
  - First Order Logic
  - Boolean Logic to form complex sentences.
- These were different ways of representing facts in a world.

# Logic Review Recap -- Propositional Logic

- Propositional Logic had statements that had truth values:

Sentence:  $\neg P_{A1}$

Semantics: There is no pit in square A1

Sentence:  $B_{A1} \leftrightarrow (P_{A2} \vee P_{B1})$

Semantics: There is a breeze in A1, if and only if there is a Pit in A2, or a Pit in B1

# Logic Review Recap -- Predicate Logic

- Predicate logic gave us the ability to reason about **statements** that pertained to “things” (i.e., **terms**).
- In predicate logic, we construct statements by applying **predicates** to terms.
- A predicate is a relationship between two terms, or a property of a single term.
  - E.g., *Brother(Richard, John)* – “Richard is the brother of John” – Richard and John are terms, Brother is a predicate representing a binary relation.
  - E.g., *King(John)* – “John is King” – John is a Term, King is a predicate for a unary relation (property)

# Logic Review Recap -- Predicate Logic

- Predicate logic also introduced the notion of a **function symbol**.
- Looks very similar to a predicate, but the output is a term (i.e., a “thing”), not a statement (i.e., a truth value).
- *LeftLeg(John)* – specifies the object “John’s Left Leg”

# Logic Review Recap – First Order Logic

- First Order Logic was like predicate logic, but it lets us reason over quantities of objects.
- We achieved this with the universal quantifier and the existential quantifier.

# Logic Review Recap First Order Logic – Universal Quantifiers

- The **Universal Quantifier** ( $\forall$ ) means "for all"
- Will use variable terms, e.g.,

$\forall x \text{ King}(x) \rightarrow \text{Person}(x)$

"For All Objects  $x$ , if  $x$  is a King, then  $x$  is a Person"

$\forall x \text{ Animal}(x) \rightarrow \text{Loves}(\text{John}, x)$

"For All  $x$ , if  $x$  is an animal, then John loves  $x$ "

i.e., "John loves all animals"

# Logic Review Recap -- First Order Logic – Existential Quantifiers

- The **Existential Quantifier** ( $\exists$ ) means "for some" or "there exists"
- Will also use variable terms, e.g.,

$\exists x \text{ Loves}(x, \text{John})$

"For some x, x loves John" -- Someone loves John.

$\exists x \text{ Loves}(\text{John}, x)$

"For some x, John loves x" – John loves someone

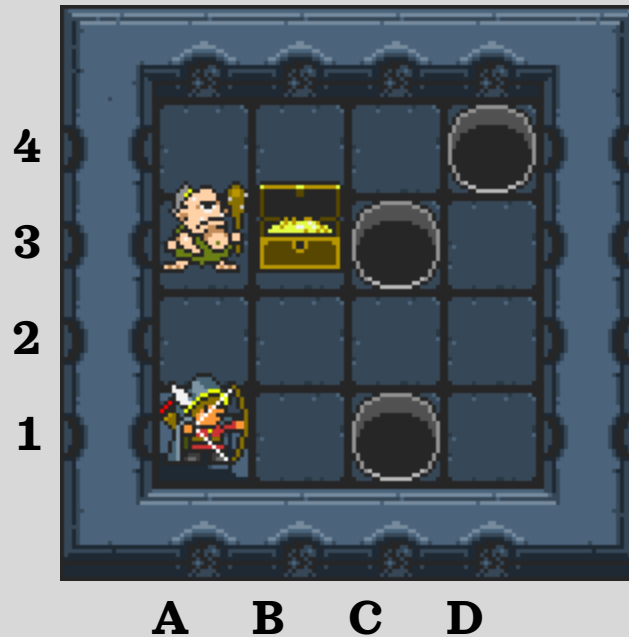
$\exists x \text{ Crown}(x) \wedge \text{OnHead}(x, \text{John})$

"There exists an object x that is a crown, and x is on the head of John"

(John's wearing a crown)



# Logic Review Recap – Wumpus World



- We also introduced the notion of Wumpus World!
- Player can navigate grid world, shoot an arrow..
- Wumpus stands still, eats player if player enters it's square (and it hasn't been killed yet).
- Bottomless Pits
- Chest of Gold
- Get in, get out.

# Logic Review Recap – Wumpus World



- If the player is ever adjacent to the Wumpus, they detect a stench.
- If the player is ever adjacent to a pit they detect a breeze.
- If the player is in the square with gold, they detect a glimmer.

# Logic Review Recap – Knowledge Base

- Knowledge-based agents “work” by having a **knowledge base**; a collection of known facts.
- Often this knowledge base starts with some knowledge in it,
  - E.g., in the form of logic statements/sentences.
- The question of the day: *how to use existing knowledge in the knowledge base to derive new knowledge*

# Logical Inference

- Or in other words... given a knowledge base with facts, can we derive new facts?
- These new facts are **entailed** by the knowledge base.
- To illustrate, we are going to represent (some of) Wumpus World with propositional logic.

# Propositional Representation

Our Proposition Syntax:

- $BA1$  = “There is a breeze at A1.”
  - $BB1$  = “There is a breeze at B1.”
  - $PC1$  = “There is a pit at C1.”
  - $PA2$  = “There is a pit at A2.”
  - $PB2$  = “There is a pit at B2.”
- ... and so on

# Starting Knowledge Base – Establishing "breeze" rules.

## Knowledge Base:

- $BA1 \leftrightarrow PA2 \vee PB1$
- $BB1 \leftrightarrow PB2 \vee PC1$
- ... and so on

"There is a Breeze in A1 if and only if there is a Pit in A2 or B1"

"There is a Breeze in B1, in and only if there is a pit in B2 or C1.



# Starting Knowledge Base – After Sensing

## Knowledge Base:

- $BA1 \leftrightarrow PA2 \vee PB1$

- $BB1 \leftrightarrow PB2 \vee PC1$

... and so on

- $\neg BA1$

“There is a Breeze in A1 if and only if there is a Pit in A2 or B1”

“There is a Breeze in B1, in and only if there is a pit in B2 or C1.

The player detects no breeze in A1 (where they begin), so  $\neg BA1$  is added.



# Starting Knowledge Base – After Moving and Sensing

## Knowledge Base:

- $BA1 \leftrightarrow PA2 \vee PB1$

- $BB1 \leftrightarrow PB2 \vee PC1$

... and so on

- $\neg BA1$

- $BB1$

“There is a Breeze in A1 if and only if there is a Pit in A2 or B1”

“There is a Breeze in B1, in and only if there is a pit in B2 or C1.

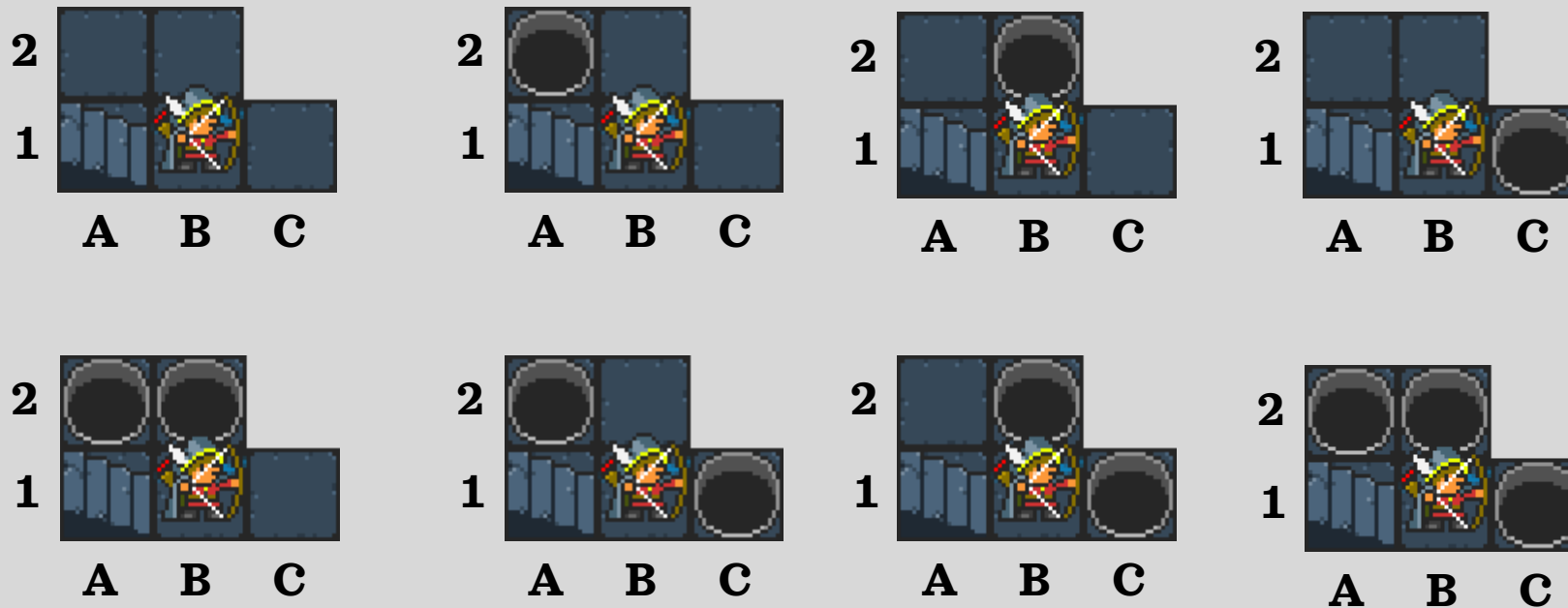
The player moves to B1, and \*does\* detect a breeze so  $BB1$  is added.





# Possible Worlds

Without considering the knowledge base, there are 8 possible worlds for the “presence of pits in squares in A2, B2, and C1”:



# Possible Worlds

But we DO have a knowledge base, and so we can use that to rule out possible worlds that don't fit it!



## Knowledge Base:

- $BA1 \leftrightarrow PA2 \vee PB1$
- $BB1 \leftrightarrow PB2 \vee PC1$
- ... and so on
- $\neg BA1$
- $BB1$

# Possible Worlds

For example, we know that there IS a breeze in B1, and we also know that there is only a Breeze in B1 if either there's a pit in B2, or a pit in C1...

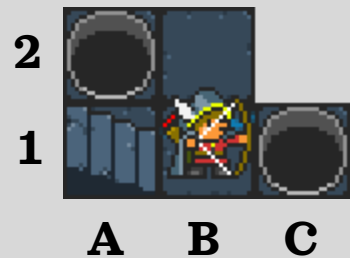
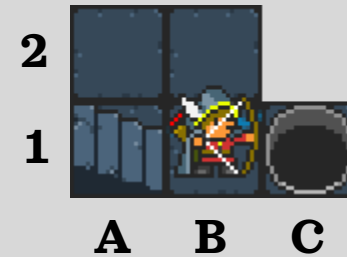


## Knowledge Base:

- $BA1 \leftrightarrow PA2 \vee PB1$
- $BB1 \leftrightarrow PB2 \vee PC1$
- ... and so on
- $\neg BA1$
- $BB1$

# Possible Worlds

So we can safely rule out any possible world (or **model**) that \*doesn't\* have a pit in B2 or a pit in C1!

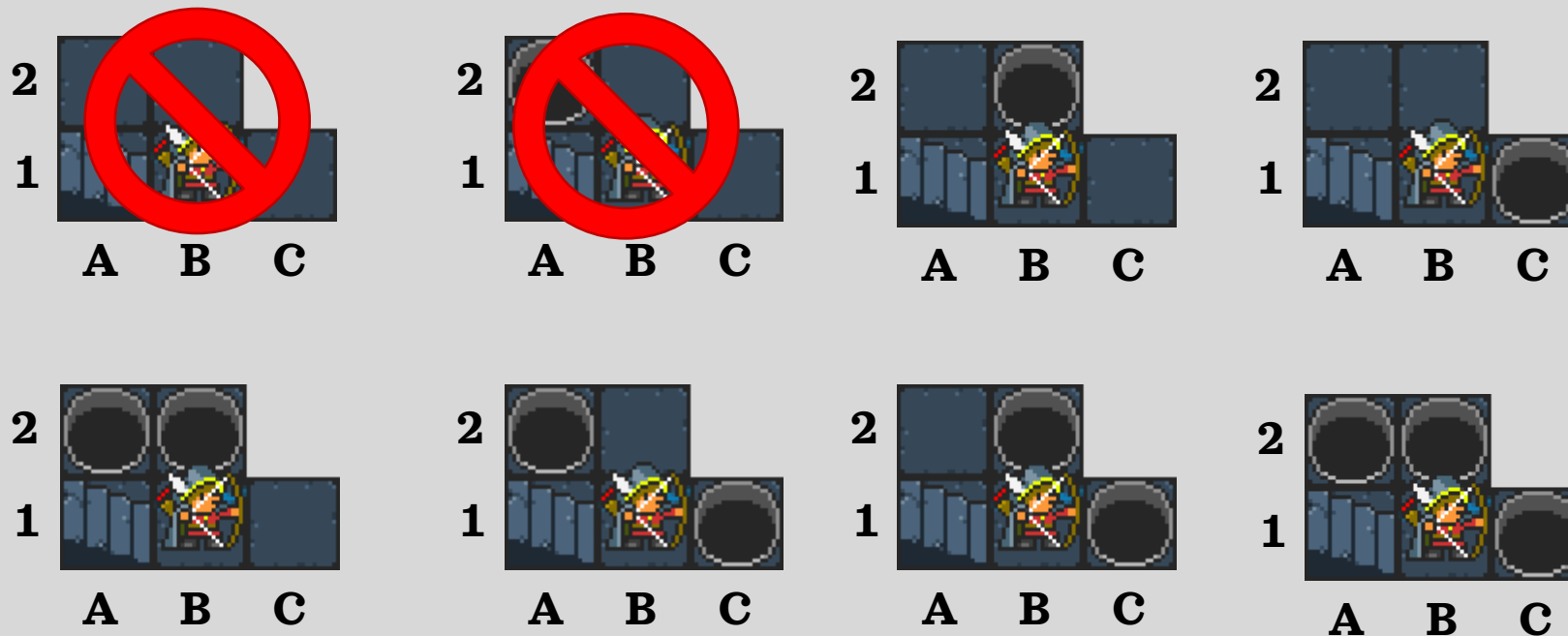


## Knowledge Base:

- $BA1 \leftrightarrow PA2 \vee PB1$
- $BB1 \leftrightarrow PB2 \vee PC1$
- ... and so on
- $\neg BA1$
- $BB1$

# Possible Worlds

Similarly... knowing that there *\*isn't\** a Breeze in A1, and that there can *\*only\** be a Pit in A2 or B1 if and only if there is a breeze in A1...

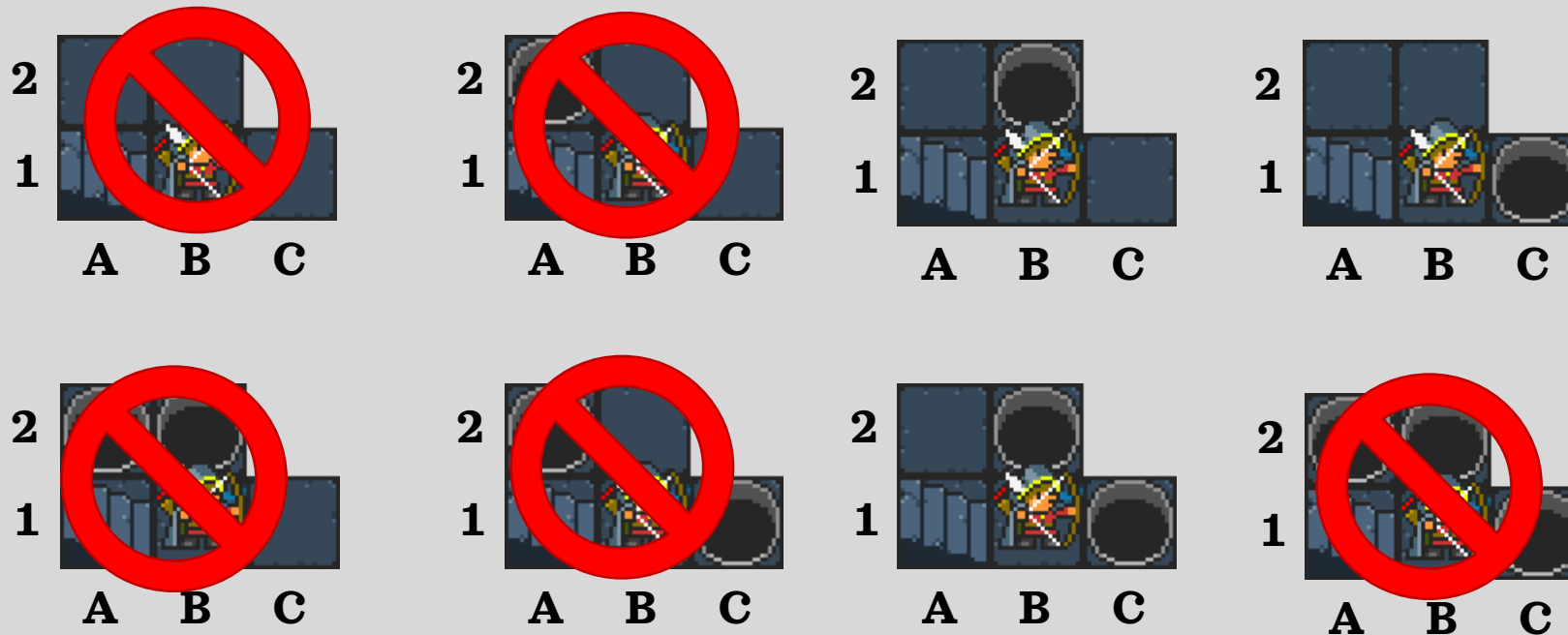


## Knowledge Base:

- $BA1 \leftrightarrow PA2 \vee PB1$
- $BB1 \leftrightarrow PB2 \vee PC1$
- ... and so on
- $\neg BA1$
- $BB1$

# Possible Worlds

Means that there \*can't\* be a pit in either A2 or B1. So we can remove any world that \*does\* have a pit there!



## Knowledge Base:

- $BA1 \leftrightarrow PA2 \vee PB1$
- $BB1 \leftrightarrow PB2 \vee PC1$
- ... and so on
- $\neg BA1$
- $BB1$

# Possible Worlds

Then we can add logical statements that apply to \*every\* remaining possible world to the knowledge base!



## Knowledge Base:

- $BA1 \leftrightarrow PA2 \vee PB1$
- $BB1 \leftrightarrow PB2 \vee PC1$
- ... and so on
- $\neg BA1$
- $BB1$
- $\neg PA2$
- $PB2 \vee PC1$

# Logical Entailment

- A fact is **logically** entailed by a knowledge base if it is true in every possible world.
- The more formal term for a possible world here is a **model**.



# Computational Deduction

- How can we replicate this logical inference process on a computer using simple knowledge representation and a reusable algorithm?
- Begin by putting everything into a common format, such as conjunctive normal form (CNF).
- Then, explore assignments of *true* or *false* to variables until we find an assignment set that works.

# Model Checking

- To be even more precise, then, a **model** isn't just a possible world...
- A model is an assignment (or partial assignment) of truth values to all the propositional variables for a problem.

# Model Checking

- An example of a model, given a set of sentences, like:

$PB2 \vee PC1$

*BB1*

- You list out all of the variables that show up in the set, like so:

PB2

PC1

BB1

# Model Checking

- And then you keep track of their values.

PB2 = ?

PC1 = ?

BB1 = ?

- Each proposition can have one of three values:
  - TRUE, FALSE, or ?
    - ? Means it hasn't been assigned a value yet.
    - You can imagine all of the variables starting off as ?
      - I'm going to write "TRUE" as "T" and "FALSE" as "F"

# Model Checking

- The goal of model checking is to find a model that **satisfies** all the known facts in the world.
  - i.e., makes every individual fact evaluate to true.
- So again, if these are our facts:

$PB2 \vee PC1$

$BB1$

- The goal of model checking is to come up with an assignment of values to the propositions that make each sentence evaluate to true.

# Model Checking

- **So, you tell me! Rattle off at least one model that satisfies these facts (there are a few)!**

$PB2 \vee PC1$

*BB1*

PB2 = ?

PC1 = ?

BB1 = ?

# Model Checking

- Here's one! And we can see why, right?

$PB2 \vee PC1$

*BB1*

$PB2 = T$

$PC1 = F$

$BB1 = T$

# Model Checking

- We sub out the value of each proposition with its value... and we get..
  - T V F (which is True)
  - T (Which is also true).

PB2  $\vee$  PC1

BB1

PB2 = T

PC1 = F

BB1 = T

T V F

T



# Model Checking

- Model checking in this way is somewhat similar to constraint satisfaction problem solving!
  - Each proposition is a variable!
  - Each variable begins with the same domain: {TRUE, FALSE, ?}
  - And the constraint is: “Each ‘fact’ in the knowledgebase needs to evaluate to true”

# Propositional Satisfiability (SAT)

- Given a logical expression in CNF...
  - $(A \text{ OR } B) \text{ AND } (A \text{ OR } \sim B) \text{ AND } (A)$  etc.
- And also given a model...
  - $A = T$
  - $B = T$
- We say that the model **satisfies** the expression if the expression can be reduced to *true*.

# Propositional Satisfiability (SAT) - Fun Fact!

- Propositional SAT was the first problem proven to be NP-Complete!
  - i.e., it is both NP and NP-Hard
- Many problems in computer science are actually satisfiability problems!
  - E.g., all those constraint satisfaction problems were asking if the constraints were satisfiable by some variable assignment.

# Propositional Satisfiability (SAT)

- Just like how CSPs needed a “consistent form”
- Propositional satisfiability needs a consistent form as well.
- That consistent form is CNF.
  - Conjunctive Normal Form.
- Let's take a moment to represent Wumpus stuff in CNF

# Wumpus in CNF

- **Knowledge Base:**

- $BA1 \leftrightarrow PA2 \vee PB1$
- $BB1 \leftrightarrow PB2 \vee PC1$

We want to represent this in CNF.

That is, we want it to be a conjunction of disjunctions!

So things like “bidirectional” have to go!

# Wumpus in CNF

We recall a helpful rule...

- **Knowledge Base:**

- $BA1 \leftrightarrow PA2 \vee PB1$

- $BB1 \leftrightarrow PB2 \vee PC1$

- Remember:  $(x \leftrightarrow y) \leftrightarrow ((x \rightarrow y) \wedge (y \rightarrow x))$

# Wumpus in CNF

- **Knowledge Base:**

- $BA1 \rightarrow PA2 \vee PB1$
- $PA2 \vee PB1 \rightarrow BA1$
- $BB1 \rightarrow PB2 \vee PC1$
- $PB2 \vee PC1 \rightarrow BB1$

And apply it to turn our two statements into four.

(You can think of all of these bullet points as being ANDed together).

We still have a lot of “implies” operators however...

# Wumpus in CNF

But we have another helpful rule!

- **Knowledge Base:**

- $BA1 \rightarrow PA2 \vee PB1$
- $PA2 \vee PB1 \rightarrow BA1$
- $BB1 \rightarrow PB2 \vee PC1$
- $PB2 \vee PC1 \rightarrow BB1$

- Remember:  $(x \rightarrow y) \leftrightarrow (\neg x \vee y)$



# Wumpus in CNF

- **Knowledge Base:**

- $\neg BA1 \vee (PA2 \vee PB1)$
- $\neg(PA2 \vee PB1) \vee BA1$
- $\neg BB1 \vee (PB2 \vee PC1)$
- $\neg(PB2 \vee PC1) \vee BB1$

- Remember:  $(x \rightarrow y) \leftrightarrow (\neg x \vee y)$

Applying it allows us to get rid of all of our implies!

We have a couple of sets of ( ) here that I kept just to highlight how we treated multiple propositions AS y, but we can drop some of them!

# Wumpus in CNF

- **Knowledge Base:**

- $\neg BA1 \vee PA2 \vee PB1$
- $\neg(PA2 \vee PB1) \vee BA1$
- $\neg BB1 \vee PB2 \vee PC1$
- $\neg(PB2 \vee PC1) \vee BB1$

Remember:  $\neg(x \vee y) \leftrightarrow (\neg x \wedge \neg y)$

But we couldn't drop all of them – namely the ones with a negation!

Well, lucky us, we have another rule!

I like to think of this one as the “distributive property” for negation.

# Wumpus in CNF

- **Knowledge Base:**

- $\neg BA1 \vee PA2 \vee PB1$
- $(\neg PA2 \wedge \neg PB1) \vee BA1$
- $\neg BB1 \vee PB2 \vee PC1$
- $(\neg PB2 \wedge \neg PC1) \vee BB1$

Remember:  $\neg(x \vee y) \leftrightarrow (\neg x \wedge \neg y)$

The NOTs have been distributed, but now we have ANDs inside of some of our would-be disjuncts.

We really would prefer it if there were only ORs in there...

# Wumpus in CNF

Well lucky us! Another rule to help us!

- **Knowledge Base:**

- $\neg BA1 \vee PA2 \vee PB1$
- $(\neg PA2 \wedge \neg PB1) \vee BA1$
- $\neg BB1 \vee PB2 \vee PC1$
- $(\neg PB2 \wedge \neg PC1) \vee BB1$

Remember:  $(x \wedge y) \vee z \leftrightarrow (z \vee x) \wedge (z \vee y)$

# Wumpus in CNF

- **Knowledge Base:**

- $\neg BA1 \vee PA2 \vee PB1$
- $(\neg PA2 \wedge \neg PB1) \vee BA1$
- $\neg BB1 \vee PB2 \vee PC1$
- $(\neg PB2 \wedge \neg PC1) \vee BB1$

Remember:  $(x \wedge y) \vee z \leftrightarrow (z \vee x) \wedge (z \vee y)$

And just to be clear why this is helpful for us...

The colors help paint the connections!

# Wumpus in CNF

- **Knowledge Base:**

- $\neg BA1 \vee PA2 \vee PB1$
- $(BA1 \vee \neg PA2) \wedge (BA1 \vee \neg PB1)$
- $\neg BB1 \vee PB2 \vee PC1$
- $(BB1 \vee \neg PB2) \wedge (BB1 \vee \neg PC1)$

This is looking pretty good!

There's two rules that have an AND in them, so for presentability, let's just put each side of the AND on its own line...

Remember:  $(x \wedge y) \vee z \leftrightarrow (z \vee x) \wedge (z \vee y)$

# Wumpus in CNF

- **Knowledge Base:**

- $\neg BA1 \vee PA2 \vee PB1$

- $BA1 \vee \neg PA2$

- $BA1 \vee \neg PB1$

- $\neg BB1 \vee PB2 \vee PC1$

- $BB1 \vee \neg PB2$

- $BB1 \vee \neg PC1$

Together, with all of them ANDed together, they form a conjunction of disjunctive clauses!

(Remember: each line here is called a 'clause' – the whole clause is true when \*at least one\* of its literals is true. That's how OR works, yeah?).

This is a CNF expression!

# Wumpus in CNF

- **This is how the previous looks in the syntax of Project 2 files:**

```
(and (or (not BA1) PA2 PB1)
```

```
      (or BA1 (not PA2))
```

```
      (or BA1 (not PB1))
```

```
      (or (not BB1) PB2 PC1)
```

```
      (or BB1 (not PB2))
```

```
      (or BB1 (not PC1)))
```



# Wumpus in CNF

- **Knowledge Base:**

- $\neg BA1 \vee PA2 \vee PB1$

- $BA1 \vee \neg PA2$

- $BA1 \vee \neg PB1$

- $\neg BB1 \vee PB2 \vee PC1$

- $BB1 \vee \neg PB2$

- $BB1 \vee \neg PC1$

An important question:

Is this initial set of facts (before any sensing occurs) satisfiable?

In other words, does it allow for at least one possible world?

Or in other-other words: can we find a model that makes this true?

# Wumpus in CNF

- **Knowledge Base:**

- $\neg BA1 \vee PA2 \vee PB1$

- $BA1 \vee \neg PA2$

- $BA1 \vee \neg PB1$

- $\neg BB1 \vee PB2 \vee PC1$

- $BB1 \vee \neg PB2$

- $BB1 \vee \neg PC1$

## **Variables:**

$$BA1 = ?$$

$$BB1 = ?$$

$$PA2 = ?$$

$$PB1 = ?$$

$$PB2 = ?$$

$$PC1 = ?$$

Well... let's try and find out!

There are 6 variables in our knowledge base...

Can we assign the variables T and F values that make each clause T?

# Wumpus in CNF

- **Knowledge Base:**

- $\neg BA1 \vee PA2 \vee PB1$
- $BA1 \vee \neg PA2$
- $BA1 \vee \neg PB1$
- $\neg BB1 \vee PB2 \vee PC1$
- $BB1 \vee \neg PB2$
- $BB1 \vee \neg PC1$

**Variables:**

$BA1 = ?$

$BB1 = ?$

$PA2 = ?$

$PB1 = ?$

$PB2 = ?$

$PC1 = ?$

And we are lucky!

Each clause is a disjunction!

That means only ONE variable in it needs to be true!

As soon as we have that, the whole clause is true!

# Wumpus in CNF

- **Knowledge Base:**

- $\neg F \vee PA2 \vee PB1$
- $F \vee \neg PA2$
- $F \vee \neg PB1$
- $\neg BB1 \vee PB2 \vee PC1$
- $BB1 \vee \neg PB2$
- $BB1 \vee \neg PC1$

## Variables:

$BA1 = F$

$BB1 = ?$

$PA2 = ?$

$PB1 = ?$

$PB2 = ?$

$PC1 = ?$

Let's say that our first thing we try is set BA1 to F.

This may or may not be right (Perhaps it turns out BA1 should be T), but we have to start somewhere.

We can replace all BA1 literals with the value 'F' now in our knowledgebase.

# Wumpus in CNF

- **Knowledge Base:**

- $T \vee PA2 \vee PB1$
- $F \vee \neg PA2$
- $F \vee \neg PB1$
- $\neg BB1 \vee PB2 \vee PC1$
- $BB1 \vee \neg PB2$
- $BB1 \vee \neg PC1$

## Variables:

$BA1 = F$

$BB1 = ?$

$PA2 = ?$

$PB1 = ?$

$PB2 = ?$

$PC1 = ?$

Note how in the first clause, we had  $\neg F$

$\neg F$  is, of course, T.

And with that, the first clause is already T!  
Regardless of what PA2 and PB1 end up being!

# Wumpus in CNF

- **Knowledge Base:**

- $T$
- $F \vee \neg PA2$
- $F \vee \neg PB1$
- $\neg BB1 \vee PB2 \vee PC1$
- $BB1 \vee \neg PB2$
- $BB1 \vee \neg PC1$

## Variables:

$BA1 = F$

$BB1 = ?$

$PA2 = ?$

$PB1 = ?$

$PB2 = ?$

$PC1 = ?$

So we can simplify that first clause down!

One of it's literals is T, so the whole thing is T (with our current model).

Likewise, if a literal is F, it can't help us make the clause T, so we can remove it!

# Wumpus in CNF

- **Knowledge Base:**

- $T$
- $\neg PA2$
- $\neg PB1$
- $\neg BB1 \vee PB2 \vee PC1$
- $BB1 \vee \neg PB2$
- $BB1 \vee \neg PC1$

**Variables:**

$BA1 = F$

$BB1 = ?$

$PA2 = ?$

$PB1 = ?$

$PB2 = ?$

$PC1 = ?$

Not too shabby! We simplified things quite a bit!

Another thing to keep in mind: if a clause ever becomes  $F$ , we know we “messed up”

That is, that model doesn't satisfy the expression; there's no possible world where the model exists.

# Wumpus in CNF

- **Knowledge Base:**

- $T$
- $\neg PA2$
- $\neg PB1$
- $\neg BB1 \vee PB2 \vee PC1$
- $BB1 \vee \neg PB2$
- $BB1 \vee \neg PC1$

**Variables:**

$BA1 = F$

$BB1 = ?$

$PA2 = T$

$PB1 = ?$

$PB2 = ?$

$PC1 = ?$

Let's set PA2 to True...



# Wumpus in CNF

- **Knowledge Base:**

- $T$
- $\neg T$
- $\neg PB1$
- $\neg BB1 \vee PB2 \vee PC1$
- $BB1 \vee \neg PB2$
- $BB1 \vee \neg PC1$

## Variables:

$BA1 = F$

$BB1 = ?$

$PA2 = T$

$PB1 = ?$

$PB2 = ?$

$PC1 = ?$

PA2 only shows up in one place...

And it's looking like maybe setting it to true was a bad idea...

# Wumpus in CNF

- **Knowledge Base:**

- $T$
- $F$
- $\neg PB1$
- $\neg BB1 \vee PB2 \vee PC1$
- $BB1 \vee \neg PB2$
- $BB1 \vee \neg PC1$

## Variables:

$BA1 = F$

$BB1 = ?$

$PA2 = T$

$PB1 = ?$

$PB2 = ?$

$PC1 = ?$

Oops! Setting PA2 to T (when BA1 was already F) caused a clause to be F!

We need to backtrack and try a different value for PA2.

# Wumpus in CNF

- **Knowledge Base:**

- $T$
- $\neg PA2$
- $\neg PB1$
- $\neg BB1 \vee PB2 \vee PC1$
- $BB1 \vee \neg PB2$
- $BB1 \vee \neg PC1$

**Variables:**

$BA1 = F$

$BB1 = ?$

$PA2 = F$

$PB1 = ?$

$PB2 = ?$

$PC1 = ?$

Oops! Setting PA2 to T (when BA1 was already F) caused a clause to be F!

We need to backtrack and try a different value for PA2.

Let's try setting it to F

# Wumpus in CNF

- **Knowledge Base:**

- $T$
- $\neg F$
- $\neg PB1$
- $\neg BB1 \vee PB2 \vee PC1$
- $BB1 \vee \neg PB2$
- $BB1 \vee \neg PC1$

**Variables:**

$BA1 = F$

$BB1 = ?$

$PA2 = F$

$PB1 = ?$

$PB2 = ?$

$PC1 = ?$

Doing that, the second clause becomes  $\sim F$

Which means T!

# Wumpus in CNF

- **Knowledge Base:**

- $T$
- $T$
- $\neg PB1$
- $\neg BB1 \vee PB2 \vee PC1$
- $BB1 \vee \neg PB2$
- $BB1 \vee \neg PC1$

## Variables:

$BA1 = F$

$BB1 = ?$

$PA2 = F$

$PB1 = ?$

$PB2 = ?$

$PC1 = ?$

Great!

Let's fast forward a bit,  
to a model that does  
satisfy the CNF  
expression.

# Wumpus in CNF

- **Knowledge Base:**

- T
- T
- T
- T
- T
- T

## **Variables:**

$$BA1 = F$$

$$BB1 = F$$

$$PA2 = F$$

$$PB1 = F$$

$$PB2 = F$$

$$PC1 = F$$

So cool!

This model satisfies the CNF expression!

So there's at least one possible world!

Here, a world where there are no pits and no breezes!

# Wumpus in CNF

- **Knowledge Base:**

- T
- T
- T
- T
- T
- T

**Variables:**

$$BA1 = F$$

$$BB1 = F$$

$$PA2 = F$$

$$PB1 = F$$

$$PB2 = F$$

$$PC1 = F$$

But here's the thing – our initial knowledge base ONLY had our “pit and breeze” rules.

It \*didn't\* have any data that the agent had sensed yet.

That means that, though this is a \*possible\* world...

It might not be the world we find ourselves in!

# Wumpus in CNF

Let's formalize a little bit the process of deciding how to assign "T" and "F" to the variables!

The following will be an algorithm that we'll progressively improve...

Which will be helpful for you for Project 2!



# Naïve SAT Solver

Begin with every variable's value unassigned.

To find a model which satisfies a CNF expression:

- If every clause is true, return true.

- If any clause is empty, return false.

- Choose an unassigned variable  $V$ .

- Set  $V=T$ . Try to find a model that satisfies.

- Set  $V=F$ . Try to find a model that satisfies.

- Return false.

Notice the similarity to the CSP solver!

# Using SAT for Inference

- Let's say that you are given two things:
  - (1) A knowledge base that we know is satisfiable.
  - (2) Some other fact, not in the knowledge base.
- How can we tell if that other fact is entailed by the knowledge base?
- Reminder: entailed means that that the fact would be true in every possible world.

# Using SAT for Inference

- **One approach:** Add the fact to the knowledge base and see if it is still satisfiable.
- Will this work?
  - No, sadly.
  - It *will* tell us if there exists *a* possible world in which the fact is true.
  - But it won't tell us it is true in *all* possible worlds.
    - i.e., it won't tell us if it is *actually* true.

# Proof by Contradiction

- **Another approach:** Given some proposition  $P$  that you want to prove true, assume  $\neg P$  and show that this leads to an absurd conclusion.
- In other words, prove  $P$  by showing that  $\neg P$  is impossible.
- In other-other words: add  $\neg P$  to the knowledge base and show that it becomes unsatisfiable.
  - That there is \*no\* possible world where  $\neg P$  is the case!

# Wumpus SAT – Proof by Contradiction

## Knowledge Base:

- $BA1 \leftrightarrow PA2 \vee PB1$
- $\neg BA1$

**Query:**  $PA2$

We want to see if the knowledge base entails our query via proof by contradiction.

# Wumpus SAT -- Proof by Contradiction

## Knowledge Base:

- $BA1 \leftrightarrow PA2 \vee PB1$
- $\neg BA1$

**Query:**  $PA2$

First step: negate the query and add it to the knowledge base!

# Wumpus SAT -- Proof by Contradiction

## Knowledge Base:

- $BA1 \leftrightarrow PA2 \vee PB1$
- $\neg BA1$
- $\neg PA2$

First step: negate the query and add it to the knowledge base!

# Wumpus SAT -- Proof by Contradiction

## Knowledge Base:

- $BA1 \leftrightarrow PA2 \vee PB1$
- $\neg BA1$
- $\neg PA2$

We'll then convert the knowledge base to CNF (get rid of that pesky bidirectional)...



# Wumpus SAT -- Proof by Contradiction

## Knowledge Base:

- $BA1 \rightarrow PA2 \vee PB1$
- $PA2 \wedge PB1 \rightarrow BA1$
- $\neg BA1$
- $\neg PA2$

We'll then convert the knowledge base to CNF (get rid of those pesky implies)...

# Wumpus SAT -- Proof by Contradiction

## Knowledge Base:

- $\neg BA1 \vee PA2 \vee PB1$
- $\neg(PA2 \vee PB1) \vee BA1$
- $\neg BA1$
- $\neg PA2$

We'll then convert the knowledge base to CNF (distribute the NOT)...

# Wumpus SAT -- Proof by Contradiction

## Knowledge Base:

- $\neg BA1 \vee PA2 \vee PB1$
- $(\neg PA2 \wedge \neg PB1) \vee BA1$
- $\neg BA1$
- $\neg PA2$

We'll then convert the knowledge base to CNF (get rid of that AND)...

# Wumpus SAT -- Proof by Contradiction

## Knowledge Base:

- $\neg BA1 \vee PA2 \vee PB1$
- $(BA1 \vee \neg PA2) \wedge (BA1 \vee \neg PB1)$
- $\neg BA1$
- $\neg PA2$

We'll then convert the knowledge base to CNF (have every clause be on its own line)

# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
If every clause is true, return true.  
If any clause is empty, return false.  
Choose an unassigned variable V.  
Set V=T. Try to find a model that satisfies.  
Set V=F. Try to find a model that satisfies.  
Return false.

## Knowledge Base:

- $\neg BA1 \vee PA2 \vee PB1$
- $BA1 \vee \neg PA2$
- $BA1 \vee \neg PB1$
- $\neg BA1$
- $\neg PA2$

## Variables:

$BA1 = ?$

$PA2 = ?$

$PB1 = ?$

Amazing, we've achieved CNF! Now the proof by contradiction begins!

Our goal is to prove that there's NO model that can possibly exist with this knowledge base.

So we ask ourselves: Are all clauses T? No. Are any clauses full false/empty? No.

# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
If every clause is true, return true.  
If any clause is empty, return false.  
**Choose an unassigned variable V.**  
Set V=T. Try to find a model that satisfies.  
Set V=F. Try to find a model that satisfies.  
Return false.

## Knowledge Base:

- $\neg BA1 \vee PA2 \vee PB1$
- $BA1 \vee \neg PA2$
- $BA1 \vee \neg PB1$
- $\neg BA1$
- $\neg PA2$

## Variables:

$BA1 = ?$

$PA2 = ?$

$PB1 = ?$

Choose some unassigned variable, such as BA1...

# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
If every clause is true, return true.  
If any clause is empty, return false.  
Choose an unassigned variable V.  
Set V=T. Try to find a model that satisfies.  
Set V=F. Try to find a model that satisfies.  
Return false.

## Knowledge Base:

- $\neg BA1 \vee PA2 \vee PB1$
- $BA1 \vee \neg PA2$
- $BA1 \vee \neg PB1$
- $\neg BA1$
- $\neg PA2$

## Variables:

*BA1 = T*

*PA2 = ?*

*PB1 = ?*

And give it a value. Let's start with T (as our naïve SAT solver algorithm has us start).

# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
If every clause is true, return true.  
If any clause is empty, return false.  
Choose an unassigned variable V.  
Set V=T. Try to find a model that satisfies.  
Set V=F. Try to find a model that satisfies.  
Return false.

## Knowledge Base:

- $F \vee PA2 \vee PB1$
- $T \vee \neg PA2$
- $T \vee \neg PB1$
- $F$
- $\neg PA2$

## Variables:

$BA1 = T$

$PA2 = ?$

$PB1 = ?$

We update our clauses based on our current model.



# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
If every clause is true, return true.  
If any clause is empty, return false.  
Choose an unassigned variable V.  
Set V=T. Try to find a model that satisfies.  
Set V=F. Try to find a model that satisfies.  
Return false.

## Knowledge Base:

- $PA2 \vee PB1$
- $T$
- $T$
- $F$
- $\neg PA2$

## Variables:

$BA1 = T$

$PA2 = ?$

$PB1 = ?$

We update our clauses based on our current model. (clause with a T are true, Fs in clauses with unknown literals are removed).

# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
If every clause is true, return true.  
If any clause is empty, return false.  
Choose an unassigned variable V.  
Set V=T. Try to find a model that satisfies.  
Set V=F. Try to find a model that satisfies.  
Return false.

## Knowledge Base:

- $PA2 \vee PB1$
- $T$
- $T$
- $F$
- $\neg PA2$

## Variables:

$BA1 = T$

$PA2 = ?$

$PB1 = ?$

And now this is basically a recursive call! Where we start at the beginning again but with this new model.

All Clauses T? No. Some clause F? **YES**.

# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
If every clause is true, return true.  
If any clause is empty, **return false**.  
Choose an unassigned variable V.  
Set V=T. Try to find a model that satisfies.  
Set V=F. Try to find a model that satisfies.  
Return false.

## Knowledge Base:

- $\neg BA1 \vee PA2 \vee PB1$
- $BA1 \vee \neg PA2$
- $BA1 \vee \neg PB1$
- $\neg BA1$
- $\neg PA2$

## Variables:

$BA1 = ?$

$PA2 = ?$

$PB1 = ?$

So we need to backtrack.

# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
If every clause is true, return true.  
If any clause is empty, return false.  
Choose an unassigned variable V.  
Set V=T. Try to find a model that satisfies.  
Set V=F. Try to find a model that satisfies.  
Return false.

## Knowledge Base:

- $\neg BA1 \vee PA2 \vee PB1$
- $BA1 \vee \neg PA2$
- $BA1 \vee \neg PB1$
- $\neg BA1$
- $\neg PA2$

## Variables:

$BA1 = F$

$PA2 = ?$

$PB1 = ?$

The algorithm wants us to try BA1 as F now.

# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
If every clause is true, return true.  
If any clause is empty, return false.  
Choose an unassigned variable V.  
Set V=T. Try to find a model that satisfies.  
Set V=F. Try to find a model that satisfies.  
Return false.

## Knowledge Base:

- $T$
- $\neg PA2$
- $\neg PB1$
- $T$
- $\neg PA2$

## Variables:

$BA1 = F$

$PA2 = ?$

$PB1 = ?$

We go through the similar reduction process...

# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
If every clause is true, return true.  
If any clause is empty, return false.  
Choose an unassigned variable V.  
Set V=T. Try to find a model that satisfies.  
Set V=F. Try to find a model that satisfies.  
Return false.

## Knowledge Base:

- $T$
- $\neg PA2$
- $\neg PB1$
- $T$
- $\neg PA2$

## Variables:

$BA1 = F$

$PA2 = ?$

$PB1 = ?$

All clauses T? No. Some clause F? No. That means keep going!

# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
If every clause is true, return true.  
If any clause is empty, return false.  
**Choose an unassigned variable V.**  
Set V=T. Try to find a model that satisfies.  
Set V=F. Try to find a model that satisfies.  
Return false.

## Knowledge Base:

- $T$
- $\neg PA2$
- $\neg PB1$
- $T$
- $\neg PA2$

## Variables:

$BA1 = F$

$PA2 = ?$

$PB1 = ?$

Choose some unassigned variable, such as PA2.

# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
If every clause is true, return true.  
If any clause is empty, return false.  
Choose an unassigned variable V.  
Set V=T. Try to find a model that satisfies.  
Set V=F. Try to find a model that satisfies.  
Return false.

## Knowledge Base:

- $T$
- $\neg T$
- $\neg PB1$
- $T$
- $\neg T$

Set it to T...

## Variables:

$BA1 = F$

$PA2 = T$

$PB1 = ?$



# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
If every clause is true, return true.  
If any clause is empty, return false.  
Choose an unassigned variable V.  
Set V=T. Try to find a model that satisfies.  
Set V=F. Try to find a model that satisfies.  
Return false.

## Knowledge Base:

- $T$
- $F$
- $\neg PB1$
- $T$
- $F$

## Variables:

$BA1 = F$

$PA2 = T$

$PB1 = ?$

Set it to T... all clauses T, no, some clause F? **Yes!**

# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
If every clause is true, return true.  
If any clause is empty, return false.  
Choose an unassigned variable  $V$ .  
Set  $V=T$ . Try to find a model that satisfies.  
Set  $V=F$ . Try to find a model that satisfies.  
Return false.

## Knowledge Base:

- $T$
- $\neg PA2$
- $\neg PB1$
- $T$
- $\neg PA2$

## Variables:

$BA1 = F$

$PA2 = ?$

$PB1 = ?$

Backtrack!

# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
If every clause is true, return true.  
If any clause is empty, return false.  
Choose an unassigned variable V.  
Set V=T. Try to find a model that satisfies.  
Set V=F. Try to find a model that satisfies.  
Return false.

## Knowledge Base:

- $T$
- $\neg F$
- $\neg PB1$
- $T$
- $\neg F$

Set PA2 to F

## Variables:

$BA1 = F$

$PA2 = F$

$PB1 = ?$

# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
If every clause is true, return true.  
If any clause is empty, return false.  
Choose an unassigned variable V.  
Set V=T. Try to find a model that satisfies.  
Set V=F. Try to find a model that satisfies.  
Return false.

## Knowledge Base:

- $T$
- $T$
- $\neg PB1$
- $T$
- $T$

## Variables:

$BA1 = F$

$PA2 = F$

$PB1 = ?$

Set PA2 to F. All clause T? No. Some clause F? No...

# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
If every clause is true, return true.  
If any clause is empty, return false.  
**Choose an unassigned variable V.**  
Set V=T. Try to find a model that satisfies.  
Set V=F. Try to find a model that satisfies.  
Return false.

## Knowledge Base:

- $T$
- $T$
- $\neg PB1$
- $T$
- $T$

## Variables:

$BA1 = F$

$PA2 = F$

$PB1 = ?$

Choose an unassigned variable, such as PB1...

# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
If every clause is true, return true.  
If any clause is empty, return false.  
Choose an unassigned variable  $V$ .  
Set  $V=T$ . Try to find a model that satisfies.  
Set  $V=F$ . Try to find a model that satisfies.  
Return false.

## Knowledge Base:

- $T$
- $T$
- $\neg T$
- $T$
- $T$

Set PB1 to T

## Variables:

$$BA1 = F$$

$$PA2 = F$$

$$PB1 = T$$

# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
If every clause is true, return true.  
If any clause is empty, return false.  
Choose an unassigned variable V.  
Set V=T. Try to find a model that satisfies.  
Set V=F. Try to find a model that satisfies.  
Return false.

## Knowledge Base:

- $T$
- $T$
- $F$
- $T$
- $T$

## Variables:

$$BA1 = F$$

$$PA2 = F$$

$$PB1 = T$$

Set PB1 to T. See that that messes us up again!

# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
If every clause is true, return true.  
If any clause is empty, return false.  
Choose an unassigned variable V.  
Set V=T. Try to find a model that satisfies.  
Set V=F. Try to find a model that satisfies.  
Return false.

## Knowledge Base:

- $T$
- $T$
- $\neg PB1$
- $T$
- $T$

## Variables:

$BA1 = F$

$PA2 = F$

$PB1 = ?$

Backtrack again!



# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
If every clause is true, return true.  
If any clause is empty, return false.  
Choose an unassigned variable V.  
Set V=T. Try to find a model that satisfies.  
Set V=F. Try to find a model that satisfies.  
Return false.

## Knowledge Base:

- $T$
- $T$
- $\neg F$
- $T$
- $T$

Set PB1 to F.

## Variables:

$$BA1 = F$$

$$PA2 = F$$

$$PB1 = F$$

# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
If every clause is true, return true.  
If any clause is empty, return false.  
Choose an unassigned variable V.  
Set V=T. Try to find a model that satisfies.  
Set V=F. Try to find a model that satisfies.  
Return false.

## Knowledge Base:

- $T$
- $T$
- $T$
- $T$
- $T$

Set PB1 to F.

## Variables:

$$BA1 = F$$

$$PA2 = F$$

$$PB1 = F$$

# Wumpus SAT -- Proof by Contradiction

Begin with every variable's value unassigned.  
To find a model which satisfies a CNF expression:  
    **If every clause is true, return true.**  
    If any clause is empty, return false.  
    Choose an unassigned variable  $V$ .  
    Set  $V=T$ . Try to find a model that satisfies.  
    Set  $V=F$ . Try to find a model that satisfies.  
    Return false.

## Knowledge Base:

- $T$
- $T$
- $T$
- $T$
- $T$

## Variables:

$$BA1 = F$$

$$PA2 = F$$

$$PB1 = F$$

All clauses are T! So the expression is Satisfiable! What does that mean for us?

# Wumpus SAT -- Proof by Contradiction

## Knowledge Base:

- $T$
- $T$
- $T$
- $T$
- $T$

## Variables:

$$BA1 = F$$

$$PA2 = F$$

$$PB1 = F$$

It means we've shown there exists some possible world in which  $\neg PA2$ . What does that mean?

# Wumpus SAT -- Proof by Contradiction

## Knowledge Base:

- $T$
- $T$
- $T$
- $T$
- $T$

## Variables:

$$BA1 = F$$

$$PA2 = F$$

$$PB1 = F$$

We \*failed\* to conclude PA2. Our knowledge base **did not entail** PA2.

# Wumpus SAT -- Proof by Contradiction

## Knowledge Base:

- $T$
- $T$
- $T$
- $T$
- $T$

## Variables:

$$BA1 = F$$

$$PA2 = F$$

$$PB1 = F$$

We *\*failed\** to conclude PA2. Our knowledge base **did not entail** PA2.

Because we just showed that there's at least one possible world where  $\neg PA2$  is the case. But for PA2 to be entailed, it must be the case in all possible worlds.

# Wumpus SAT -- Proof by Contradiction

- If you were to use the same starting KB for some other queries, and go through a similar process...

**Query:**  $PA1$

KB +  $\neg PA1$  **is** satisfiable. We **cannot** conclude  $PA1$ .

**Query:**  $\neg PA1$

KB +  $PA1$  is **not** satisfiable. We **can** conclude  $\neg PA1$ .

$\neg PA1$  is entailed by the knowledge base.

# Improving the SAT Solver.

Begin with every variable's value unassigned.

To find a model which satisfies a CNF expression:

- If every clause is true, return true.

- If any clause is empty, return false.

- Choose an unassigned variable  $V$ .

- Set  $V=T$ . Try to find a model that satisfies.

- Set  $V=F$ . Try to find a model that satisfies.

- Return false.

This is what we just did...



# Improving the SAT Solver.

Begin with every variable's value unassigned.

To find a model which satisfies a CNF expression:

If every clause is true, return true.

If any clause is empty, return false.

**Choose an unassigned variable  $V$ .**

Set  $V=T$ . Try to find a model that satisfies.

Set  $V=F$ . Try to find a model that satisfies.

Return false.

But perhaps there's a better way...

# Which Variable to Choose? (Unit Clause Example)

- There are two cool tricks for deciding which variable to choose!
- They involve identifying the following:
  - **Unit Clauses**
  - **Pure Symbols**
- Identifying these things will be *\*very\** helpful for your project 2!

# Which Variable to Choose? (Unit Clause Example)

## Knowledge Base:

- $\neg BA1 \vee PA2 \vee PB1$
- $BA1 \vee \neg PA2$
- $BA1 \vee \neg PB1$
- $\neg \mathbf{BA1}$
- $\neg PA2$

## Variables:

$BA1 = ?$

$PA2 = ?$

$PB1 = ?$

Recall: Last time, we chose  $BA1$  first, set it to True, and immediately had to backtrack. In retrospect, it was silly of us to even try setting  $BA1$  to T, when we can see its negation is the only literal in this clause! If  $BA1$  is true, that clause is DEFINITELY GOING to be false.

# Which Variable to Choose? (Unit Clause Example)

## Knowledge Base:

- $\neg BA1 \vee PA2 \vee PB1$
- $BA1 \vee \neg PA2$
- $BA1 \vee \neg PB1$
- $\neg BA1$
- $\neg PA2$

} Unit clauses

## Variables:

$BA1 = ?$

$PA2 = ?$

$PB1 = ?$

When you have a clause with just one literal, that is called a **Unit Clause**.

Unit Clauses are great, because there's no decision to make! Your hands are tied! You *\*need\** to assign variables that show up in unit clauses to the value that makes those unit clauses evaluate to true!

# Which Variable to Choose? (Unit Clause Example)

## Knowledge Base:

- $\neg BA1 \vee PA2 \vee PB1$
  - $BA1 \vee \neg PA2$
  - $BA1 \vee \neg PB1$
  - $\neg BA1$
  - $\neg PA2$
- } Unit clauses

## Variables:

$BA1 = ?$

$PA2 = ?$

$PB1 = ?$

Here, we can see that setting  $BA1$  to T makes the unit clause it is in F.

So  $BA1$  \*can't\* be T. It must be F.

# Which Variable to Choose? (Unit Clause Example)

## Knowledge Base:

- $\neg BA1 \vee PA2 \vee PB1$
  - $BA1 \vee \neg PA2$
  - $BA1 \vee \neg PB1$
  - $\neg BA1$
  - $\neg PA2$
- } Unit clauses

## Variables:

$BA1 = F$

$PA2 = F$

$PB1 = ?$

That means we don't need to "guess" for both BA1 and PA2. We can immediately set them both to what they need to be (here, both F)

# Which Variable to Choose? (Unit Clause Example)

## Knowledge Base:

- $\neg F \vee F \vee PB1$
- $F \vee \neg F$
- $F \vee \neg PB1$
- $\neg F$
- $\neg F$

## Variables:

$BA1 = F$

$PA2 = F$

$PB1 = ?$

Those variables showed up in a lot of clauses. We replace every literal of those variables with these new values. And we can simplify the CNF expression down quite a bit.

# Which Variable to Choose? (Unit Clause Example)

## Knowledge Base:

- $T \vee PB1$
- $T$
- $\neg PB1$
- $T$
- $T$

## Variables:

$$BA1 = F$$

$$PA2 = F$$


$$PB1 = ?$$

- After simplifying it down, do you notice anything interesting?



# Which Variable to Choose? (Unit Clause Example)

## Knowledge Base:

- $T \vee PB1$
- $T$
- $\neg PB1$   Unit clause
- $T$
- $T$

## Variables:

$$BA1 = F$$


$$PA2 = F$$

$$PB1 = ?$$

- We have essentially created a NEW unit clause! This clause still lacks a truth value, and now only has one unassigned proposition. This proposition is that clause's last chance to become true!

# Which Variable to Choose? (Unit Clause Example)

## Knowledge Base:

- $T \vee PB1$
- $T$
- $\neg PB1$   Unit clause
- $T$
- $T$

## Variables:

$$BA1 = F$$

$$PA2 = F$$

$$PB1 = \textcolor{red}{F}$$

- And just as before, it makes our job of assigning the value to the proposition very easy! There's only one valid choice it can possibly be! (Here, F again).

# Which Variable to Choose? (Unit Clause Example)

## Knowledge Base:

- $T \vee F$
- $T$
- $\neg F$
- $T$
- $T$
- And with that...

## Variables:

$$BA1 = F$$

$$PA2 = F$$

$$PB1 = F$$

# Which Variable to Choose? (Unit Clause Example)

## Knowledge Base:

- $T$
- $T$
- $T$
- $T$
- $T$

## Variables:

$$BA1 = F$$

$$PA2 = F$$

$$PB1 = \textcolor{red}{F}$$

- And with that... we have a model that satisfied the expression! And it involved ZERO backtracking! (note: this doesn't change the outcome at all, our proof by contradiction will still be the 'same' end result. We just got there way faster than the initial brute force approach we tried).

# Which Variable to Choose? (Unit Clause Example)

- So, hopefully you see how Unit Clauses reduce the amount of 'bad guesses' you might otherwise make...
- Our two secret weapons:
  - ~~Unit Clauses~~
  - **Pure Symbols**
- Let's chat about pure symbols now!

# Which Variable to Choose? (Unit Clause Example)

- A Pure Symbol refers to a variable that *\*always\** shows up with the same **valence**.
- That is – every literal of that variable is positively valenced
  - e.g.:  $X$
- Or alternatively, every literal of the variable is negatively valenced:
  - e.g.:  $\neg X$

# Which Variable to Choose? (Pure Symbol Example)

## Knowledge Base:

- $\neg X \vee Y \vee Z$
- $\neg Y \vee Z$

## Variables:

$X = ?$

$Y = ?$

$Z = ?$

Jumping to a different example for a second...

**What are the pure symbols here (if any)?**

# Which Variable to Choose? (Pure Symbol Example)

## Knowledge Base:

- $\neg X \vee Y \vee Z$
- $\neg Y \vee Z$

## Variables:

$X = ?$

$Y = ?$

$Z = ?$

The pure symbols here are X and Z.

X only shows up as negatively valenced.

Z only shows up as positively valenced.

Y is not a pure symbol, since it shows up as both positively valenced AND negatively valenced.



# Which Variable to Choose? (Pure Symbol Example)

## Knowledge Base:

- $\neg X \vee Y \vee Z$
- $\neg Y \vee Z$

## Variables:

$X = ?$

$Y = ?$

$Z = ?$

So with that observation under our belts...

**What would be the ‘smart’ value to assign to X?**

**What would be the ‘smart’ value to assign to Z?**

# Which Variable to Choose? (Pure Symbol Example)

## Knowledge Base:

- $\neg X \vee Y \vee Z$
- $\neg Y \vee Z$

## Variables:

$X = ?$

$Y = ?$

$Z = ?$

As we just established, here  $Z$  is an example of **pure symbol**. Across every clause, it always appears with the same **valence**.

That is to say, we only ever see  $Z$ , and we never see  $\neg Z$

Because of this, \*there's no reason to try  $Z = F$ \*. It only makes sense for  $Z = T$ .

# Which Variable to Choose? (Pure Symbol Example)

## Knowledge Base:

- $\neg X \vee Y \vee F$
- $\neg Y \vee F$

## Variables:

$X = ?$

$Y = ?$

$Z = F$

Because let's say that we did make  $Z = F$ . We're just making life more difficult for ourselves needlessly, right?

If we set  $Z$  to  $F$ , now \*neither\* of our clauses are true, and we need to do more variable assignments to find a satisfying model.

# Which Variable to Choose? (Pure Symbol Example)

## Knowledge Base:

- $\neg X \vee Y \vee T$
- $\neg Y \vee T$

## Variables:

$X = ?$

$Y = ?$

$Z = T$

BUT ALTERNATIVELY, If we had set Z to T...

Now both of the clauses are true right then and there, and our model is done!

# Which Variable to Choose? (Pure Symbol Example)

## Knowledge Base:

- $\neg X \vee Y \vee T$
- $\neg Y \vee T$

## Variables:

$X = ?$

$Y = ?$

$Z = T$

The upshot:

- \*If you have a positively valenced pure symbol, you should set that symbol to T
- \*If you have a negatively valenced pure symbol, you should set that symbol to F

# Which Variable to Choose? (Pure Symbol Example)

## Knowledge Base:

- $\neg X \vee Y \vee Z$
- $\neg Y \vee Z$

## Variables:

$$X = F$$

$$Y = ?$$

$$Z = ?$$

And just to be complete...  $X$  is a pure symbol too, right? It only shows up as negatively valenced.

So likewise, it makes no sense to ever set  $X$  to T.

If we set it to F though...

# Which Variable to Choose? (Pure Symbol Example)

## Knowledge Base:

- $T \vee Y \vee Z$
- $\neg Y \vee Z$

## Variables:

$$X = F$$

$$Y = ?$$

$$Z = ?$$

Then, voila, that clause immediately becomes true!

# Which Variable to Choose? (Pure Symbol Example)

- With Unit Clauses and Pure Symbols under our belts, we can improve our SAT algorithm!
- The next slide details the **DPLL** Algorithm
  - Named for the authors of both it and its predecessor,
    - Martin Davis, Hilary Putnam, George Logemann, Donald Loveland
- This is one of the bots you are provided with for your next project!



# DPLL Algorithm

Begin with every variable's value unassigned.

To find a model which satisfies a CNF expression:

- Simplify the model using unit propagation.

- Simplify the model using pure symbols.

- If every clause is true, return true.

- If any clause is empty, return false.

- Choose an unassigned variable  $V$ .

- Set  $V=T$ . Try to find a model that satisfies.

- Set  $V=F$ . Try to find a model that satisfies.

- Return false.

# DPLL Algorithm

Begin with every variable's value unassigned.

To find a model which satisfies a CNF expression:

Simplify the model using **unit propagation**.

Simplify the model using pure symbols.

If every clause is true, return true.

If any clause is empty, return false.

Choose an unassigned variable  $V$ .

Set  $V=T$ . Try to find a model that satisfies.

Set  $V=F$ . Try to find a model that satisfies.

Return false.

Unit propagation means: find unit clauses, and assign variables in those unit clauses the values they need to make the clause true.

(As we saw before, the act of doing this can turn other clauses “into” unit clauses).

# DPLL Algorithm

Begin with every variable's value unassigned.

To find a model which satisfies a CNF expression:

Simplify the model using unit propagation.

Simplify the model using **pure symbols**.

If every clause is true, return true.

If any clause is empty, return false.

Choose an unassigned variable  $V$ .

Set  $V=T$ . Try to find a model that satisfies.

Set  $V=F$ . Try to find a model that satisfies.

Return false.

And then after that, if you have any pure symbols...

i.e., variables that always appear with the same valence...

Assign them the values they need to be to become true.

# DPLL Algorithm

Begin with every variable's value unassigned.

To find a model which satisfies a CNF expression:

Simplify the model using unit propagation.

Simplify the model using **pure symbols**.

If every clause is true, return true.

If any clause is empty, return false.

Choose an unassigned variable  $V$ .

Set  $V=T$ . Try to find a model that satisfies.

Set  $V=F$ . Try to find a model that satisfies.

Return false.

And very similar to  
how we saw with the  
unit clauses...

You really only need  
to consider the  
'unsatisfied clauses;

$X \vee \text{TRUE}$

$\neg X \vee Y$

$\neg X \vee Z$

Is  $X$  a pure symbol  
in the above  
model?

# DPLL Algorithm

Begin with every variable's value unassigned.

To find a model which satisfies a CNF expression:

Simplify the model using unit propagation.

Simplify the model using **pure symbols**.

If every clause is true, return true.

If any clause is empty, return false.

Choose an unassigned variable V.

Set V=T. Try to find a model that satisfies.

Set V=F. Try to find a model that satisfies.

Return false.

Essentially, yes!

$X \vee \text{TRUE}$

$\neg X \vee Y$

$\neg X \vee Z$

That first clause is already satisfied! It doesn't care what X's value is.

So the whole model could be rewritten as:

$\text{TRUE}$

$\neg X \vee Y$

$\neg X \vee Z$

# DPLL Algorithm

Begin with every variable's value unassigned.

To find a model which satisfies a CNF expression:

- Simplify the model using unit propagation.

- Simplify the model using pure symbols.

- If every clause is true, return true.

- If any clause is empty, return false.

- Choose an unassigned variable  $V$ .**

- Set  $V=T$ . Try to find a model that satisfies.

- Set  $V=F$ . Try to find a model that satisfies.

- Return false.

Now by the time you get here, you've done all the "easy stuff" already!

The following slides detail other tricks you can do, too!

# Component Analysis

## Knowledge Base:

- $A \vee B \vee C$
- $\neg A \vee B$
- $X \vee Y \vee Z$
- $\neg Y$

## Variables:

$A = ?$

$B = ?$

$C = ?$

$X = ?$

$Y = ?$

$Z = ?$

Let's say we have this knowledge base with 6 different variables!

# Component Analysis

## Knowledge Base:

$$\circ A \vee B \vee C$$

$$\circ \neg A \vee B$$

$$\circ X \vee Y \vee Z$$

$$\circ \neg Y$$

## Variables set 1:

$$A = ?$$

$$B = ?$$

$$C = ?$$

## Variable set 2:

$$X = ?$$

$$Y = ?$$

$$Z = ?$$

We can separate clauses that share no unassigned variables, and solve these problems in parallel! Each of these subsets is called a **component**.



# Borrowing from CSPs

- Variable and Value Ordering
- Intelligent backtracking and back jumping
- Local Search

# Local Search Recap

- **Traditional Search:**
  - Start with an empty solution (i.e., empty path).
  - At each step, add a thing to the solution.
  - Return solution on success; backtrack on fail.
- **Local Search:**
  - Start with a random (probably bad) solution.
  - At each step, make one change.
  - Run until solution is found or out of time.

# Local Search SAT Solver

Randomly assign T or F to every variable.

Until you run out of time:

- If every clause is true, return solution.

- Choose a variable  $V$ .

- Flip the variable ( $T \rightarrow F$  or  $F \rightarrow T$ ).

# Local Search SAT Solver

Randomly assign T or F to every variable.

Until you run out of time:

If every clause is true, return solution.

**Choose a variable  $V$ .**

Flip the variable ( $T \rightarrow F$  or  $F \rightarrow T$ ).

Once again... perhaps there's  
another way...

# Local Search Recap

- **Local Search:**
  - Start with a random (probably bad) solution.
  - At each step, make one change.
    - Usually make a change that improves the solution.
    - Sometimes make a random change
      - E.g., the simulated annealing “temperature”
    - Restart if stuck in local maxima
      - E.g., the “hill climbing with random restarts”
  - Run until solution is found or out of time.

# Solution Quality (Utility)

- **How do we measure which solutions to a SAT problem are better than others?**
  - Remember: we are living in a world where we've randomly assigned T or F to every variable to start.
- If a problem has  $n$  clauses, then a solution has 0 unsatisfied clauses and  $n$  satisfied clauses.
- The number of satisfied clauses is a good measure of how good a solution is.
  - In other words, solutions where more clauses are satisfied are better than ones where fewer clauses are satisfied.

# GSAT Algorithm (example of Local Search SAT).

Let  $n$  be the "noise parameter," where  $0 < n < 1$ .

Randomly assign T or F to every variable.

Until you run out of time:

- If every clause is true, return solution.

- With probability  $p < n$ :

  - Flip the variable that leads to higher utility.

- Else:

  - Flip a random variable.

# GSAT Example

Problem:     `(and (or A B) (or (not A) B))`

Solution:     `A=? B=?`

Expression: `(and (or ? ?) (or (not ?) ?))`

- Here we see both original problem, current attempted solution, and what the CNF expression looks like with our current solution.
- We start with a random assignment of values to variables.



# GSAT Example

Problem:     `(and (or A B) (or (not A) B))`

Solution:     `A=T B=F`

Expression: `(and (or T F) (or (not T) F))`

- Here is a random assignment of values to variables.
- **Is this a solution?**

# GSAT Example

Problem:     `(and (or A B) (or (not A) B) )`

Solution:     `A=T B=F`

Expression:   `(and (or T F) (or (not T) F) )`



satisfied       Not satisfied

- Is this a solution?
- No, so we need to choose a variable to flip.

# GSAT Example

Problem:        `(and (or A B) (or (not A) B))`

Solution:       `A=T B=F`

Expression:     `(and (or T F) (or (not T) F))`

- GSAT is all about this **noise parameter**,  $n$ . Let's say  $n = 0.25$
- This means 25% of the time, we choose a variable at random to flip.
- Let's say we generate a random number  $p$  between 0 and 1. Let's say we get 0.13.
- Since  $p < n$ , we pick a variable at random. Let's say we get **A**.

# GSAT Example

Problem:     `(and (or A B) (or (not A) B) )`

Solution:     `A=T B=F`

Expression:   `(and (or T F) (or (not T) F) )`

- We flip the value of A. It used to be T...

# GSAT Example

Problem:     `(and (or A B) (or (not A) B) )`

Solution:     `A=F B=F`

Expression:   `(and (or F F) (or (not F) F) )`

- We flip the value of A. It used to be T... and so it becomes F.

# GSAT Example

Problem:     `(and (or A B) (or (not A) B) )`

Solution:     `A=F B=F`

Expression:   `(and (or F F) (or (not F) F) )`

Not satisfied

Satisfied

- Is this a solution?
- No! So we need to choose another variable to flip.

# GSAT Example

Problem:        `(and (or A B) (or (not A) B) )`

Solution:       `A=F B=F`

Expression:    `(and (or F F) (or (not F) F) )`

- We generate another random number, this time let's say  $p = 0.82$ .
- We recall our noise parameter  $n$  was 0.25, so  $p > n$  this time.
- So we won't choose randomly – we'll flip the variable that results in the most clauses becoming satisfied.

# GSAT Example

Problem:        `(and (or A B) (or (not A) B) )`

Solution:       `A=F B=F`

Expression:    `(and (or F F) (or (not F) F) )`

- So we need to figure that out!
- If we flip **A** from **F** to **T**, how many total clauses will be satisfied?    1
- If we flip **B** from **F** to **T**, how many total clauses will be satisfied?    2
- So flipping B is the better choice here!



# GSAT Example

Problem:     `(and (or A B) (or (not A) B) )`

Solution:     `A=F B=T`

Expression:   `(and (or F T) (or (not F) T) )`

- So let's do it! Let's flip **B** from **F** to **T**.

# GSAT Example

Problem:     `(and (or A B) (or (not A) B) )`

Solution:     `A=F B=T`

Expression:   `(and (or F T) (or (not F) T) )`



satisfied                      satisfied

- Is *\*THIS\** a solution?

- Yes! It is!

# Improving GSAT

- GSAT is another one of the “bots” that was given to you for project 2!
- There's two features of GSAT that would be nice to improve:
  - Every step, we take time considering between \*every\* existing variable to flip.
  - The randomness is maybe a little too random.
    - Very easy to accidentally flip a 'good' variable to a worse value!

# WalkSAT Algorithm.

Let  $n$  be the "noise parameter," where  $0 < n < 1$ .

Randomly assign T or F to every variable.

Until you run out of time:

- If every clause is true, return solution.

- Randomly choose an unsatisfied clause  $c$ .

- With probability  $p < n$ :

  - Flip the variable in  $c$  that leads to higher utility.

- Else:

  - Flip a random variable in  $c$ .

# WalkSAT Algorithm.

Let  $n$  be the "noise parameter," where  $0 < n < 1$ .

Randomly assign T or F to every variable.

Until you run out of time:

    If every clause is true, return solution.

    Randomly choose an unsatisfied clause  $c$ .

    With probability  $p < n$ :

        Flip the variable in  $c$  that leads to higher utility.

    Else:

        Flip a random variable in  $c$ .

The basic upshot:

Instead of deciding between \*all\* variables when choosing one to flip...

Only consider variables that exist in "problem clauses"

# GSAT vs. WalkSAT

- The algorithms are very similar. The difference is how they choose which variable to flip.
- GSAT considers every variable every time.
- WalkSAT first chooses an unsatisfied clause, then chooses a variable in that clause.
- Thus, WalkSAT has a greater chance to improve the solution, even when making a random move.

# WalkSAT: One Final Note

- WalkSAT is very good at finding a model that satisfies the problem.
  - i.e., if the problem has a solution, it can find it!
- However, if the problem \*doesn't\* have a solution, it can't always detect it.
  - And we just saw how showing a lack of satisfiability is important for showing entailment.
- We said in the algorithm “until you run out of time” (e.g., maybe you marked the total number of flips to try).
- “I thought about it for an hour, and couldn't find a world where C1 isn't safe”
  - That is promising, but not a proof! C1 might still be deadly!

# Project 2

- You will make your own SAT solver!
- Your deadly, SAT solver foes:
  - “Random Bot”
  - “Brute Bot” (the “naïve SAT solver” from these slides)
  - “GSAT Bot”
  - “WalkSAT Bot”
  - “DPLL Bot”
- You are also given lots and lots of problems in CNF.
- Can your solver best these others in solving these problems?!?