Introduction to AI, Spring 2023

Introduction to Logic

The Art of Knowing Things

- Humans know things.

- This knowledge (can be) used to reason about other things.
  - To use what you already know to figure other new things out.

- This approach in AI is embodied in **knowledge-based agents**.

The Art of Knowing Things

- The agents of the “uninformed” and “informed” search problems (kinda) knew things…
  - Agents that solved problems like 8-queens, 8-puzzle, traveling around Romania, playing chess, etc.

- Transition models of each problem told the agent the result of each action.
  - But very domain-dependent.
  - And basically “only” has the outcome of actions.
  - Doesn’t reveal anything about the nature of the problem.

The Art of Knowing Things

- Constraint Satisfaction Problems introduced lots of good approaches!
  - Representing States as value assignments to variables.
  - Representing problems in domain-independent ways.
  - Allowing for efficient, generalized algorithms.

- We continue this trend with **logic**.
  - A general class of representations to support knowledge-based agents.

Logic

- The major questions logic attempts to answer:

How can we represent knowledge about the world in a general, reusable way?

How can we use existing knowledge to gain new knowledge?

Knowledge-Based Agents

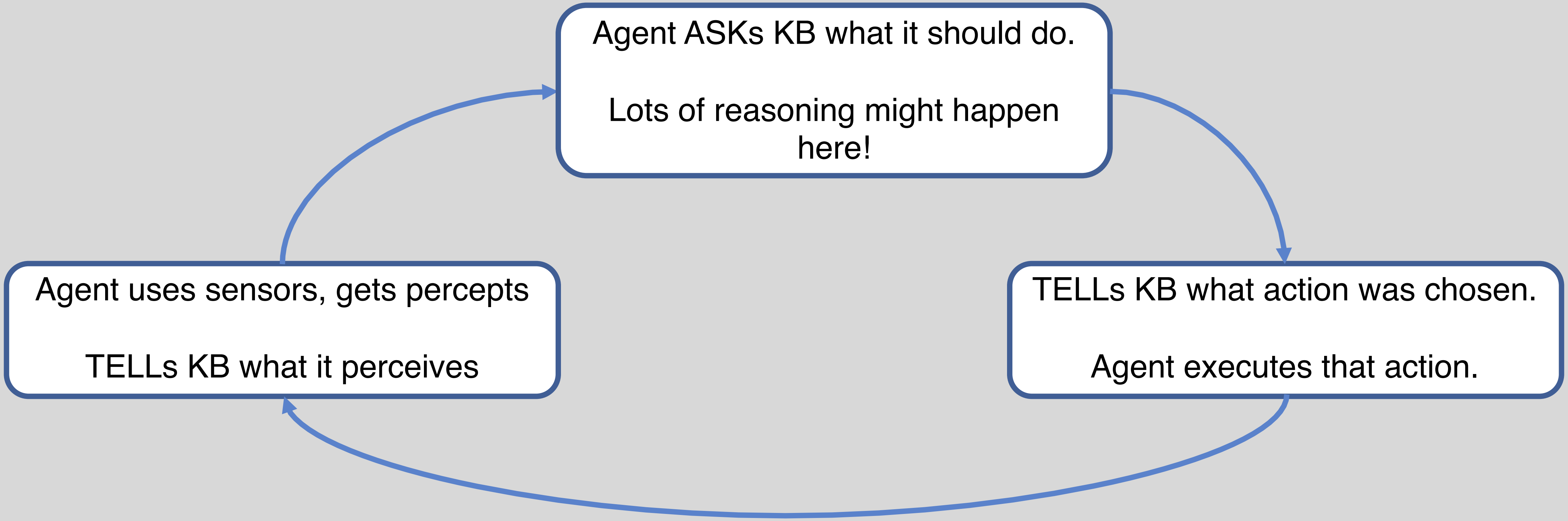
- Central component of a knowledge-based agent is…
  - its **knowledge base (KB)**.

- The knowledge base is a set of **sentences**.
  - Each sentence represents an assertion about the world.
  - Written in a **knowledge representation language**.

- There needs to be a way to add new sentences into the KB.
  - May involve **inference**: deriving new sentences from old.
  - May involve the agent updating it based on sensor input.

Knowledge-Based Agents

- The general flow of KB-Agents is TELL, ASK, TELL


Agent ASKs KB what it should do.
Lots of reasoning might happen here!
TELLs KB what action was chosen.
Agent executes that action.
Agent uses sensors, gets percepts
TELLs KB what it perceives

Knowledge-Based Agents

- You can imagine starting with an empty knowledge base.
- Add to it by TELLing sentences, one by one, until agent knows how to operate.
- This is **declarative** system building.
    - In contrast to **procedural**.

Problem Solving Approaches

- **Procedural**: For each new problem, write a domain-specific algorithm to solve it.
  - Fast/Efficient
  - Not reusable
  - Requires domain expertise.

- **Declarative**: Describe the problem in a common syntax and solve it using general techniques.
  - Slow
  - Reusable
  - Less expertise required.

Syntax vs. Semantics

- The knowledge base is full of sentences.
  - Sentences are expressed in the **syntax** of the representation language.
  - Syntax specifies sentences that are well formed.

- Using the Syntax of ordinary arithmetic…

“x + y = 4” – a well formed sentence!

”x4y+ =“. -- not a well formed sentence!

Syntax vs. Semantics

- If syntax specifies how to structure sentences…
- **Semantics** specifies what any given sentence means.
  - And when that sentence is true or false.

For example: $x + y = 4$

This is true in a world where x is 2 and y is 2.

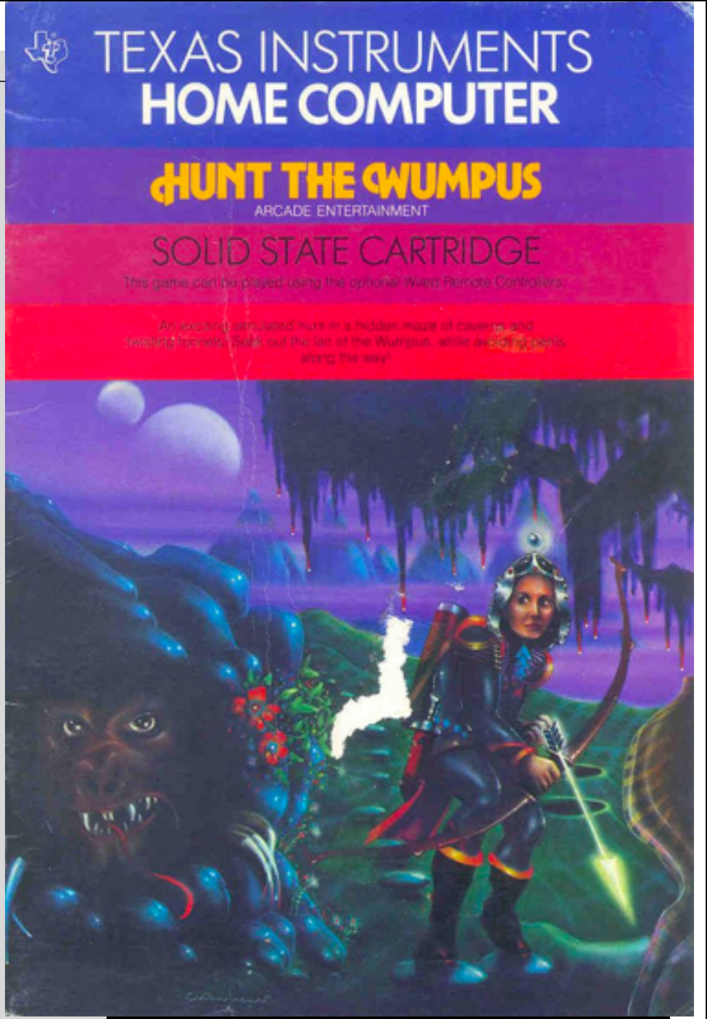
This is false in a world where x is 1 and y is 1.

Logical algorithms manipulate syntax, but the user or agent interprets meaning.

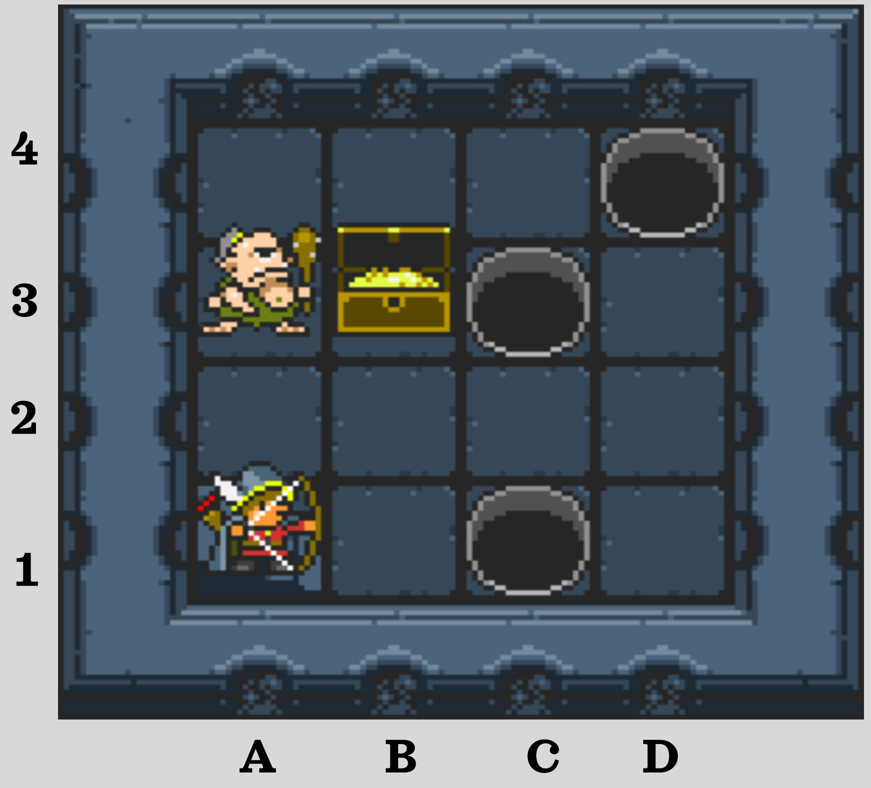
Wumpus World

- There is a classic environment called the Wumpus world.
  - Dark Caves!
  - A Deadly Monster!
  - A Brave Hero!
  - Bottomless Pits!
  - Golden Treasure!
  - A Single Arrow!
  - And most importantly….

  - A classic showcase of knowledge-based agents.

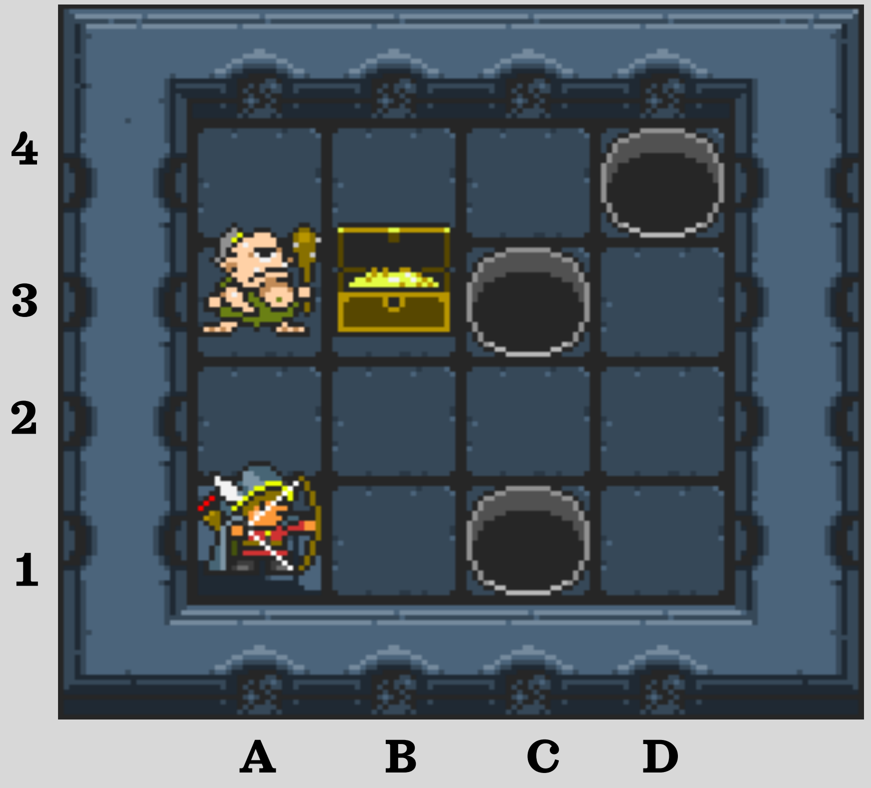


Wumpus World– Performance Measure



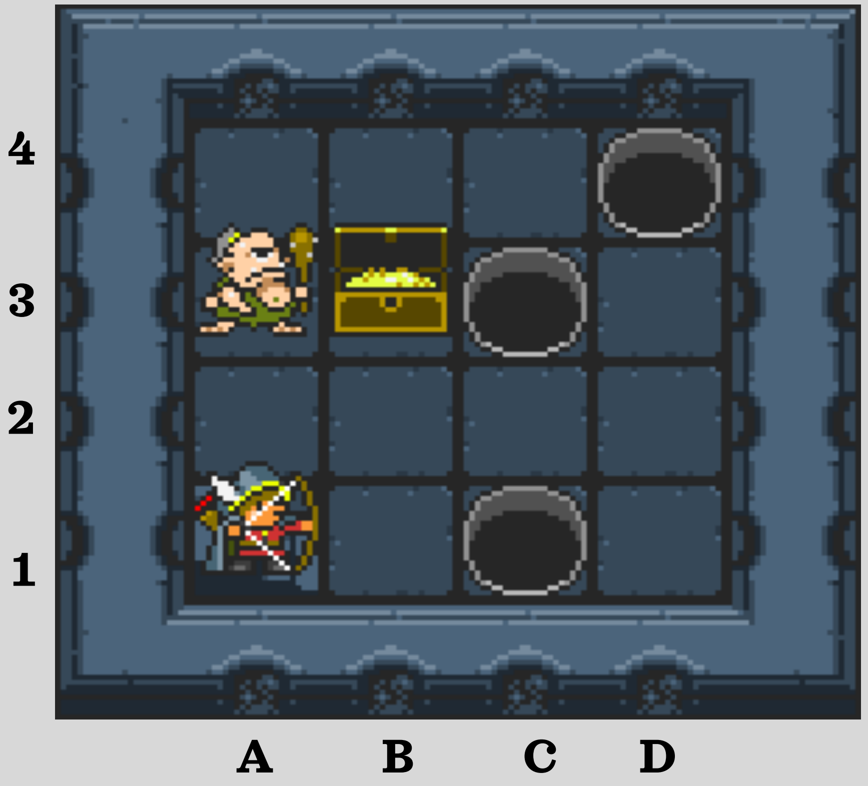
- Goal: get the treasure and get out!
- More formal performance measure:
  - +1000 for escaping cave with gold.
  - -1000 for death (fall in pit or eaten by Wumpus)
  - -1 for each action taken.
  - -10 for using your arrow.

Wumpus World - Environment



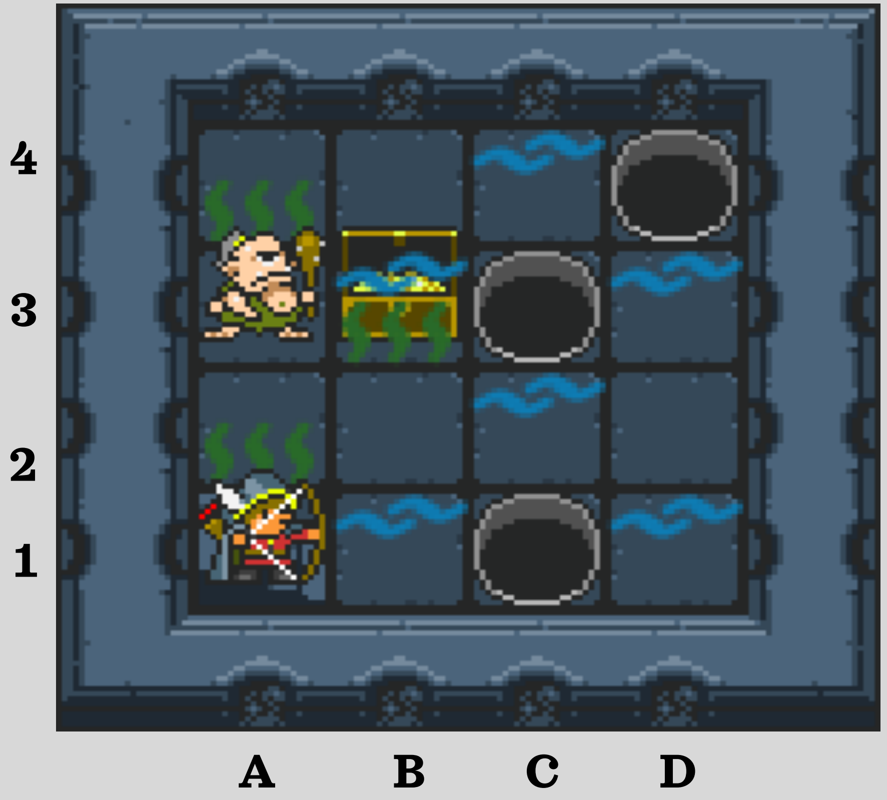
- Environment is a 4x4 grid.
- Agent starts in A1, facing right.
- Location of Wumpus and gold are random (but will never be on start square).
- The Wumpus cannot move.
- Any square other than start may be a pit.

Wumpus World - Actuators



- Agent can:
  - Move **Forward**
  - **TurnLeft** or **TurnRight** 90 degrees
    - Rotation within a square.
  - **Grab** (treasure)
  - **Shoot** arrow (in direction currently facing).
    - Kills Wumpus if connects.
  - **Climb** out (if in square A1)
- Agent dies if entering square with pit or live Wumpus.

Wumpus World - Sensors



- When adjacent to a pit, agent can detect a **breeze**.
- When adjacent to the Wumpus, agent can detect a **stench**.
- When on the square with gold, agent can detect a **glitter**.
- When agent walks into a wall, detects a **bump**.
- When Wumpus is killed, agent detects a **scream**.

Logics

- Several different ways to represent statements or sentences:
  - Propositional Logic
  - Predicate Logic
  - First Order Logic

- We can combine sentences using:
  - Boolean Logic

Propositional Logic

- Every statement about the world which can be true or false is represented as a unique symbol.
  - The ‘symbol’ itself is the syntax, and what the symbol ‘means’ is the semantics.

Syntax:	R
Semantics:	“It is raining outside”

Syntax:	PC1
Semantics:	“There is a pit at square C1”

- These are examples of **atomic sentences.**
  - But we can combine multiple atomic sentences to form **complex sentences** using **Boolean logic**.

Boolean Logic

- Boolean operators provide a way to combine atomic sentences (which can be true or false) into larger expressions
  - Which, themselves, evaluate to true or false.

- You are no doubt familiar with the classics!

- But let’s take a look at some truth tables all the same.

Boolean Logic – Negation (¬)

- The **NOT** operator: ¬
- When the NOT operator (or **negation**) is applied to a term it produces the opposite truth value of the term.

x	$\neg x$
T	F
F	T

Boolean Logic – Conjunction (∧)

- The **AND** operator: ∧   (Looks like an ‘A’ – A for “And”!)
- An AND expression (i.e., a **Conjunction**) is true only when its operands (i.e., **conjuncts**, i.e., it’s “left” and “right) are both true.

x	y	$x \wedge y$
T	T	T
T	F	F
F	T	F
F	F	F

Boolean Logic – Disjunction (∨)

- The **OR** operator: ∨ (Looks like a V for Latin “vel”).
- An OR expression (i.e., a **Disjunction**) is true when *at least one* of its operands (i.e., **disjuncts**, i.e., it’s “left” and “right) are true.

x	y	$x \vee y$
T	T	T
T	F	T
F	T	T
F	F	F

Boolean Logic – Implication (→)

- The **Implication** operator:→
- An Implication expression (aka a **rule** or an **if-then** statement) has a **premise** or **antecedent** (the left hand side) followed by a **conclusion** or **consequent** (the right hand side).

x	y	$x \rightarrow y$
T	T	T
T	F	F
F	T	T
F	F	T

Boolean Logic – Implication (→)

- Perhaps somewhat counter intuitive.
- Implication only evaluates to false when premise is true and conclusion is false.
- Think of it as:

"If premise is true, I claim conclusion is true…

But if premise is false, I make no claim"

x	y	$x \rightarrow y$
T	T	T
T	F	F
F	T	T
F	F	T

Boolean Logic – Biconditional (↔)

- The **Biconditional** operator:↔
- A Biconditional expression (aka **IFF**; If and only if) evaluates to true if both operands have the same value.
- Technically its true when $\boldsymbol{x} \rightarrow \boldsymbol{y}$ *and* $\mathbf{y} \rightarrow \boldsymbol{x}$ are true.
-

x	y	$\boldsymbol{x} \leftrightarrow \boldsymbol{y}$
T	T	T
T	F	F
F	T	F
F	F	T

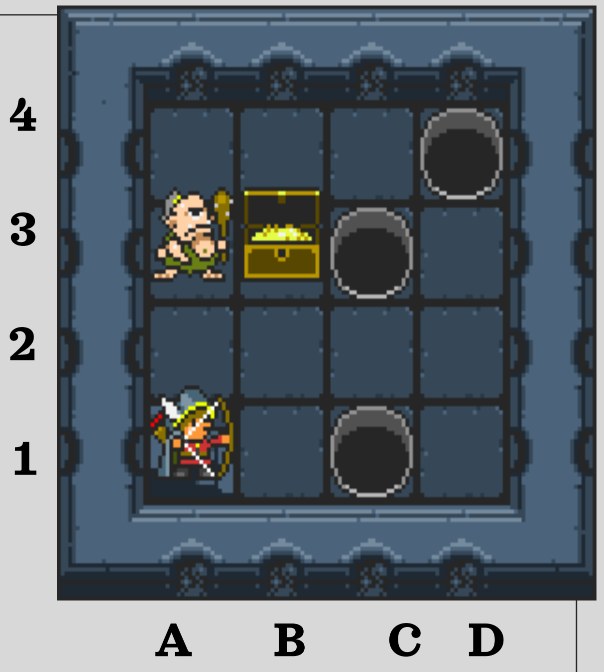
Logics – let’s dig into propositional logic now.

- Several different ways to represent statements or sentences:
  - Propositional Logic
  - Predicate Logic
  - First Order Logic

- We can combine sentences using:
  - ~~Boolean Logic~~

Propositional Logic

- Now that we have these tools, we can begin to write sentences in propositional logic to describe the Wumpus World!



Sentence:	$\neg P_{A1}$
Semantics:	There is no pit in square A1

Sentence:	$B_{A1} \leftrightarrow (P_{A2} \lor P_{B1})$
Semantics:	There is a breeze in A1, if and only if there is a Pit in A2, or a Pit in B1

Propositional Logic

- This is great!
- But gets a little cumbersome…
- We need to specify breeze sentences for every single square…

Sentence: $B_{B1} \leftrightarrow (P_{A1} \lor P_{B2} \lor P_{C1})$

Semantics: There is a breeze in B1, if and only if there is a Pit in A1, or a Pit in B2, or a Pit in C1

Sentence: $B_{C1} \leftrightarrow (P_{B1} \lor P_{C2} \lor P_{D1})$

Semantics: There is a breeze in C1, if and only if there is a Pit in B1, or a Pit in C2, or a Pit in D1

Propositional Logic

- This is a shortcoming of propositional logic.

- The issue is that every statement is its own symbol with a definite truth value.
  - P_{XY}: “There is a pit in such and such square”
  - B_{XY} : “There is a breeze in such and such square”
  - W_{XY}: “There is a Wumpus in such and such square”

Statements vs. Things

◦ It would be nice if we could distinguish between **statements** (that have a truth value) and the **things** those statements are about (which do not have a truth value).

Statements vs. Things

- “It is raining outside.” is a **statement.**
- It is true or false.

- “rain” and “outside” are **things**.
- Statements are made about things.
- Things are not true or false.

Logics – Predicate logic can help us with this distinction between “statements” and “things”!

- Several different ways to represent statements or sentences:
  - ~~Propositional Logic~~
  - Predicate Logic
  - First Order Logic

- We can combine sentences using:
  - ~~Boolean Logic~~

Predicate Logic

- Propositional Logic may not be able to help us out here…
- But **predicate logic** can!

- Predicate logic distinguishes between objects, and statements about those objects.

- **Term** is the official word for these “things”
  - i.e., a term refers to an object.
  - We’ll see there are a couple of different ways to specify terms.

Predicate Logic

- Statements are made by applying a **predicate** to some number of terms.
- Predicates are a **relationship** between objects (**binary relation**) or a property of a single object (**unary relation**).
- Example predicates (i.e., relationships):
- Brother(Richard, John)
  - Typically read as “first term has the relationship with second term”
  - “Richard is the brother of John”
- King(John)
  - John is a King

Predicate Logic

- Often times you get to define whatever predicates and objects you want to use to represent your world.
- The following two slides both represent the “same world”, but with different predicates.

Predicate Logic Example

Predicates:

at = “a thing is at a location.”

breeze = “breeze at”

stench = “stench at”

Objects:

P = “Player”

W = “Wumpus”

P1 = “pit 1”

A1 = “square A1”

Examples:

at(P1, B3) = “Pit 1 is at square B3”

at(P, B2) = “The Player is at square B2”

breeze(B2) = “The player detects a breeze at square B2”

Predicate Logic Example – Alternative!

Predicates:

pit = “pit at”

breeze = “breeze at”

stench = “stench at”

Wumpus = “Wumpus at”

Examples:

breeze(B2) = “The player detects a breeze at B2

pit(B3) = “there is a pit at square B3”

Objects:

A1 = “square A1”

A2 = “square A2”

Predicate Logic

- So far we've seen two types of symbols:

- **Constant Symbols**
  - Refers to a specific object
  - E.g., P = "Player", W = "Wumpus", A1 = "Square A1"

- **Predicate Symbols**
  - Refers to a Relationship or a Property
  - E.g., Brother(Richard, John) – Richard is the Brother of John
  - King(John) – John is a King.
  - Pit(A1) – There is a Pit in square A1

Predicate Logic – Function Symbols

- There is a third type of symbol: A **function symbol**.
- These are kind of tricky!
- Syntactically they *Look Like* Predicates!
- But semantically they are *closer to* Constants!

Predicate Logic – Function Symbols

- Remember: There is a difference between a "*statement*" and a "*term*"
  - Recall: Statements have a truth value.
  - Recall: A term is a 'thing' -- an object that can have properties that we reason about.

- **A Predicate is a statement**
  - That is, it's asserting something that may or may not be true.
    - e.g., Are Richard and John brothers? Is John a King?

- **A function is a term**.
  - That is, it specifies an object (e.g., "John").

Predicate Logic – Function Symbols

- The benefit of having access to functions is that it spares us from having to define a ton of constants.
- There is a somewhat classic / somewhat silly example of this: “Left Legs”
- You can imagine that most humans in the world have a left leg.
- And if we are solving a problem that involves a lot of reasoning about left legs, without functions we’d need to create a new constant symbol for every left leg in the world! In addition to constants for the people who own those left legs! That list would get bloated!
  - Ben
  - LeftLegOfBen
  - John
  - LeftLegOfJohn
  - ChrisSumma
  - LeftLegOfChrisSumma
  - etc., etc.

Predicate Logic – Function Symbols

- But if instead, we had the LeftLeg(person) function…
  - That refers to the left leg of the person you pass in…

- Then we can easily refer to people’s left legs without needing special constants for them!
  - *LeftLeg(John)* – specifies the object “John’s Left Leg”
  - *LeftLeg(Ben)* – specifies the object ‘Ben’s Left Leg’
  - *LeftLeg(Chris)* – specifies the object “Chris’ left leg”

- And remember: These Left Legs are “Things” – they are terms -- *not statements*!
  - It doesn’t make sense to say that John’s Left Leg is true or false.

- And now the only constants we need are for the people themselves!
  - The function provides us with a nice way of mapping a ‘person’ constant to their left leg.

Predicate Logic – Function Symbols

- Other function examples could include:

- *above(B1)* means “the square above square B1”
- *above(x)* means “the square above square x”

- Again, both of those functions map to another object! Map to a term!
  - i.e., they are ”things” – they have no truth value!
- Unlike predicate relationships (e.g., king(John)) which does have truth (e.g., “John is King”).

Predicate Logic – Function Symbols

- And another reminder!

- The relationship between syntax and semantics is up to you/the agent!

King(John) is a predicate relationship that means “John Is King” because I told you it does.

But you could imagine that same syntax, *King(John)* being a **function** that maps to the object that John has sworn fealty to!

i.e., “The King of John”.

Predicate Logic – Function Symbols

- Given the extreme possibility of confusion between, say:
  - King(John) – a predicate that means that John is a king
  - King(John) –a function that returns the person that is John’s king.

- It is on the person who creates the syntax to clearly specify what is a function and what is a predicate.
  - And to also clearly specify what any given predicate or function means.
    - That’s me when making these slides!
    - And that’s *you* when answering some of the homework questions!

Predicate Logic – Types of Terms

- We’ve discussed that a **term** is a logical expression that refers to an object.

- We’ve talked in depth about two of them:

  1.) Constants: Refers to a specific object (e.g., Wumpus, John, etc.).

  2.) Functions: Maps one object to another (e.g., *LeftLeg(John)*).

Predicate Logic – Types of Terms

- In total there are three flavors of terms:
  1.) Constants: Refers to a specific object (e.g., Wumpus, John, etc.).
  2.) Functions: Maps one object to another (e.g., *LeftLeg(John)*).
  3.) Variables: Can (potentially) refer to any term. (e.g., *x*)

Predicate Logic – Ground Expressions

- We say that a logical expression is **ground** when it has no variables.

For example…

King(John) is ground	-- because John is a constant.
King(x) is not ground	-- because x is a variable
King(LeftLeg(Richard)) is ground	-- because LeftLeg(Richard) is a function

Predicate Logic – Ground Expressions

- Also, just for fun… if I tell you that…
  - King(x) is a predicate that means “x is a king”
  - And LeftLeg(x) is a function that refers to the term “the left leg of x”
- **What do the following logical statements mean?**

King(John) John is a King

King(x) x is a King

King(LeftLeg(Richard)) Richard’s Left Leg is a king. (note how nesting works – start with the innermost function and go from there!)

Predicate Logic – Combining Sentences

- And we can use Boolean logic to make more complex sentences!

What do the following predicate logic sentences mean?

King(x) is a predicate that means ‘x is a king’

LeftLeg(x) is a function that returns ‘the left leg of x’

Brother(x,y) is a predicate that means ‘x is the brother of y’

King(Richard) V King(John) Richard is a King OR John is a King

¬King(Richard) → King(John) IF Richard is not a King, THEN John is a King

Brother(Richard, John) ∧ Brother(John, Richard) Richard is the brother of John AND John is the brother of Richard

¬Brother(LeftLeg(Richard), John) The left leg of Richard is NOT the Brother of John

Predicate Logic – Combining Sentences

- While we're on the subject of complex sentences…

- What does the following sentence mean?

(again, Brother(x,y) is a predicate that states "x is the brother of y", and LeftLeg(x) is a function that returns the object of x's left leg).

LeftLeg(Brother(Richard,John))

Predicate Logic – Combining Sentences

- The answer: this sentence is *improperly formed*!

LeftLeg(Brother(Richard,John))

You can think of is as having to do with “mismatched types”

Recall: when you have nested things like this, you start with the innermost first.

That means you start with Brother(Richard,John)

That is a predicate – i.e., a statement about the world with a truth value. You can think of this as “returning” true or false. Let’s say that Richard and John ARE brothers, so this predicate is true...

Predicate Logic – Combining Sentences

- So we can think of the sentence as this…

LeftLeg(true)

And so, if we replace Brother(Richard, John) with a truth value (here, true)…

Hopefully the non-sensical nature of this logic reveals itself!

It doesn't make sense to get the LeftLeg of "true" – "true" isn't a term, it's just a truth value!

So again, you can think of this as being an "input type mismatch exception" – LeftLeg wants a term, but it was handed a truth value.

Predicate Logic – Combining Sentences

- And real quick, while we're thinking about, what about this? What does this sentence mean? Is it properly formed?

Brother(LeftLeg(John), LeftLeg(Richard))

This sentence *is* properly formed!

It is asking "Is John's Left Leg the brother of Richard's Left Leg"

That is, arguably, an odd question to ask.

But at least it is "askable" – John's Left Leg is a term, Richard's Left Leg is a term, and the Brother predicate wants two terms as input. It all checks out!

Logics – OK, so all of that was predicate logic. Let’s recap quickly…

- Several different ways to represent statements or sentences:
  - ~~Propositional Logic~~ //Everything has a symbol and truth value.
  - ~~Predicate Logic~~ //Distinction between ‘terms’ (constants,variables,functions) and ‘statements’ (predicates)
  - First Order Logic

- We can combine sentences using:
  - ~~Boolean Logic~~ //Let’s us combine simple sentences to make complex sentences.

Logics – So we can see where this is going…

- Several different ways to represent statements or sentences:
  - ~~Propositional Logic~~ //Everything has a symbol and truth value.
  - ~~Predicate Logic~~ //Distinction between 'terms' (constants,variables,functions) and 'statements' (predicates)
    // Allows for us to describe relationships between terms.
  - **First Order Logic**

- We can combine sentences using:
  - ~~Boolean Logic~~ //Lets us combine simple sentences to make complex sentences.

First Order Logic

- Specifies not only relationships between objects, but also quantities of objects.

- We achieve this with two standard **quantifiers**:
  - **Universal Quantifier**: ∀
  - **Existential Quantifier**: ∃

First Order Logic – Universal Quantifiers

- The **Universal Quantifier (∀)** means ”for all”
- Will use variable terms, e.g.,

$$\forall x\, King(x)$$

Means, “For all x, x is a king”

Or in “good English”: Everyone is a king.

First Order Logic – Universal Quantifiers

- We can combine the **Universal Quantifier (∀)** with Boolean logic to form complex sentences.

What do the following sentences mean (use reasonable interpretations for the predicates):

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

“For All Objects x, if x is a King, then x is a Person”

or in good English: Every king is a person.

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

“For All x, if x is an animal, then John loves x”

or in good English: “John loves all animals”

First Order Logic – Existential Quantifiers

- The **Existential Quantifier (∃)** means ”for some” or “there exists”
- Will also use variable terms, e.g.,

∃*x Loves(x, John)*

“For some x, x loves John” -- Someone loves John.

∃*x Loves(John,x)*

“For some x, John loves x” – John loves someone

∃x Crown(x) ∧ OnHead(x, John)

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

(John’s wearing a crown)

First Order Logic – Existential Quantifiers

- The **Existential Quantifier (∃)** means ”for some” or “there exists”
- Will also use variable terms.
- For example…

$$\exists x \; King(x)$$

Means…

“There exists an x in which that x is a King”

Or in good English: Someone is a king / There exists a king.

First Order Logic – Existential Quantifiers

- Try translating the following **Existential Quantifier (∃)** sentences into English:
- Note: Loves(x,y) is a predicate that means “x loves y”

∃x Loves(x, John)

“For some x, x loves John” -- Someone loves John.

∃x Loves(John,x)

“For some x, John loves x” – John loves someone

∃x Crown(x) ∧ OnHead(x, John)

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

(John’s wearing a crown)

First Order Logic – Quantifiers

- This solves one of the major limitations of propositional logic!
- Remember how before we had to specify breeze sentences for every single square?

Sentence: $B_{B1} \leftrightarrow (P_{A1} \lor P_{B2} \lor P_{C1})$

Semantics: There is a breeze in B1, if and only if there is a Pit in A1, or a Pit in B2, or a Pit in C1

Sentence: $B_{C1} \leftrightarrow (P_{B1} \lor P_{C2} \lor P_{D1})$

Semantics: There is a breeze in C1, if and only if there is a Pit in B1, or a Pit in C2, or a Pit in D1

The above specifies what it means for there to be a breeze in two squares… we would need a total of 16 of these rules in a 4x4 Wumpus World grid. What a labor to write all of that!

First Order Logic – Quantifiers

- With the power of quantifiers, we can write a *single* sentence that describes properties of *every* square.

- That is, we can now write a single sentence that looks like this:

“All breezy squares must be adjacent to a pit”

What might this sentence look like using first order logic?

First Order Logic – Quantifiers

- Perhaps something that looks like this:

$$\forall x(breeze(x) \rightarrow \exists y\ (Adjacent(x,y) \land Pit(y)))$$

We might translate this as..

 “For all x, if there is a breeze in x, then there exists some y that is adjacent to x and that same y has a pit in it”

Or in “good English”: All breezy squares are adjacent to a pit.

First Order Logic – Quantifiers

- OK… we feeling good about quantifiers so far?
- I’m about to rapid fire introduce a bunch of additional concepts building off of these first-order-logic quantifiers, but if we aren’t feeling like we have a strong foundation of the basics we should reinforce that first before moving on…

First Order Logic – Quantifiers

- OK… here’s a laundry list of the additional quantifier concepts we’re about to cover:
  - Scope of quantifiers
  - Standardizing Apart
  - A convention for denoting Universal Quantifiers
  - Skolemization

First Order Logic – Quantifier Scope

- This example brings up an important point on the **scope** of quantifiers…
- Note the parentheses in use here…

$$\forall x \underbrace{\left(breeze(x) \rightarrow \exists y \underbrace{\left(Adjacent(x,y) \land Pit(y) \right)}_{\textit{The scope of 'y'}} \right)}_{\textit{The scope of 'x'}}$$

First Order Logic – Quantifier Scope

- Technically you are allowed to re-use variable symbols in these “inner” quantifiers.
  - Kind of like having a local variable in a method that has the same name as a class variable in Java. (variable shadowing).

$$\forall x \; (\; breeze(x) \rightarrow \underbrace{\exists y,x}_{\textit{We are re-using the x symbol here!}} \; (\; Adjacent(x,y) \wedge Pit(y) \;) \;)$$

First Order Logic – Quantifier Scope

- But now the universally quantified x is referring to *different things* than the existentially quantified x!

$$\forall x \, (\; breeze(x) \rightarrow \exists y,x \, (\; Adjacent(x,y) \wedge Pit(y) \;) \;)$$

The breeze(x) is referring to the universally quantified x. So this is saying "for all pits, if there is a breeze in them…

But even though it's the same symbol, this x is referring to something totally different! This is just saying "there exists some other square x, that is adjacent to some other square y with a pit."

First Order Logic – Quantifier Scope

- So this new sentence is actually not as useful / powerful a statement as the previous one.:
  - “For every square that has a breeze, there exists *a* square (that might be the one with the breeze or might not be) that is adjacent to a pit”.

$$\forall x \; (\; breeze(x) \rightarrow \exists y,x \; (\; Adjacent(x,y) \wedge Pit(y) \;) \;)$$

First Order Logic – Quantifier Scope

- Here are the two next to each other for comparison purposes…

$\forall x(breeze(x) \rightarrow \exists y\,(Adjacent(x,y) \land Pit(y)))$

The 'good' one.

"All breezy squares are adjacent to a pit."

$\forall x\ (\ breeze(x) \rightarrow \exists y,x\ (\ Adjacent(x,y) \land Pit(y)\ )\ )$

The 'bad' one.

*"For every square that has a breeze, there exists *a* square (that might be the one with the breeze or might not be) that is adjacent to a pit".*

First Order Logic – Quantifier Scope

- The upshot: if you have inner quantifiers, just use different symbols so you don't get confused with the outer symbols!

$$\forall x(breeze(x) \rightarrow \exists y\,(Adjacent(x,y) \land Pit(y)))$$

No confusion! x always means the same thing

"All breezy squares are adjacent to a pit."

$$\forall x\,(\;breeze(x) \rightarrow \exists y,z\,(\;Adjacent(z,y) \land Pit(y)\;)\;)$$

Still no confusion! Using 'z' illustrates that the inner quantifier is referring to a *different* square than the outer quantifier!

*"For every square that has a breeze, there exists *a* square (that might be the one with the breeze or might not be) that is adjacent to a pit".*

First Order Logic – Quantifier Scope

- So hopefully that feels good and reasonable, and isn’t altogether that different from following good variable naming practices that you do in all of your programming classes.

First Order Logic – Quantifiers

- OK… that’s one down… moving on to Standardizing Apart…
  - ~~Scope of quantifiers~~
  - Standardizing Apart
  - A convention for denoting Universal Quantifiers
  - Skolemization

First Order Logic – Standardizing Apart all

$\forall x(breeze(x) \rightarrow \exists y\ (Adjacent(x,y) \land Pit(y)))$

$\forall x(stench(x) \rightarrow \exists y\ (Adjacent(x,y) \land Wumpus(y)))$

So, here are two sentences.

The first sentence is the one we've been working with (all breezy squares are adjacent to a pit).

What is that second sentence saying in English?

First Order Logic – Standardizing Apart

$\forall x(breeze(x) \rightarrow \exists y\ (Adjacent(x,y) \wedge Pit(y)))$

$\forall x(stench(x) \rightarrow \exists y\ (Adjacent(x,y) \wedge Wumpus(y)))$

- The second sentence is saying "all stinky square are adjacent to a Wumpus"

- So these are two totally different sentences. The 'x' and 'y' symbols are being used in both of them, but that's OK…
    - Each sentence is basically it's own self contained unit…

First Order Logic – Standardizing Apart

$\forall x(breeze(x) \rightarrow \exists y\ (Adjacent(x,y) \land Pit(y)))$

$\forall x(stench(x) \rightarrow \exists y\ (Adjacent(x,y) \land Wumpus(y)))$

- BUT….
- Each of these sentences are still contributing to one singular world "The Wumpus World"
  - You could think of it as a system of equations almost…
    - $3x + y = -3$
    - $x = -y + 3$
      - If I tell you that the two above equations are connected, then x and y needs to be the same number in both equations, right?

First Order Logic – Standardizing Apart

$\forall x(breeze(x) \rightarrow \exists y\ (Adjacent(x,y) \wedge Pit(y)))$

$\forall x(stench(x) \rightarrow \exists y\ (Adjacent(x,y) \wedge Wumpus(y)))$

- If we consider the above logical statements as a system, then the fact that we are re-using x and y across the different sentences might indicate that they refer to the SAME squares.
- But that isn’t necessarily true!
- The set of breezy squares that are adjacent to pits is likely totally different from the set of stinky squares adjacent to Wumpuses

First Order Logic – Standardizing Apart

$\forall x(breeze(x) \rightarrow \exists y\ (Adjacent(x,y) \wedge Pit(y)))$

$\forall w(stench(w) \rightarrow \exists z\ (Adjacent(w,z) \wedge Wumpus(z)))$

- SO THE UPSHOT:
- It is common practice to “**Standardize Apart**” i.e., use unique variable names across sentences.
- This ensure there is no confusion that the sentences are referring to different things.
    - Note how the first sentence uses x and y, but now the second one uses w and z. Totally different variable names for different sentences!
- (For now just take my word that this is useful; we’ll see why shortly!)

First Order Logic – Quantifiers

- OK, now we’re going to introduce a special notation convention…
  - ~~Scope of quantifiers~~
  - ~~Standardizing Apart~~
  - A convention for denoting Universal Quantifiers
  - Skolemization

Universal Qualifier Convention

- We will also see situations where it is convenient to not specify *any* quantifiers (but still make use of them).

- When using first order logic, we can assume all variables are universally quantified unless otherwise specified.
  - So we can leave off the ∀ symbol

For example…

∀x Loves(John, x) --- *John loves everyone*

Can be rewritten as:

Loves(John, x) --- *John loves everyone (same meaning!)*

Universal Qualifier Convention

- So given that…
- **What can we reasonable interpret this next sentence to be?**

$$\text{Hungry}(x) \land \mathit{Child}(X) \rightarrow \mathit{Want}(X, \mathit{Pizza})$$

"All Hungry Children Want Pizza"

We assume that X is universally quantified, thus we are referring to not just *a* hungry child, but rather *all* hungry children.

First Order Logic – Quantifiers

- OK, here’s the last addition, and it’s a bit of a doozy… Skolemization.

- We’ll start ‘simply’ with this first example, but then introduce a more fancy case for it later in lecture

    - ~~Scope of quantifiers~~
    - ~~Standardizing Apart~~
    - ~~A convention for denoting Universal Quantifiers~~
    - Skolemization

Existential Instantiation: Skolemization

- Because existential quantification is making a claim about (at least) one object (i.e., “there exists a…”)…

- We can *replace* existentially quantified variables with a “new” **constant** that represents the object in question.
  - This is called a **Skolem constant**.
    - Make sure you give this constant a name/symbol that isn’t already being used!
  - This process is **Skolemization**.
    - It yields a sentence with the *same meaning* as the original, but *without* the use of the existential quantifier!

Existential Instantiation: Skolemization

- For example:

∃x Loves(x, Mosquitoes)   -- (There exists at least one object, x, that loves Mosquitoes)

The idea behind Skolemization is basically giving a specific name to that “one thing” that the existential quantifier is claiming exists.

So, let’s give it a name! Let’s call this one thing that loves mosquitoes is the constant “P”, for person.

With that, we can rewrite the above sentence *without* needing to use the existential quantifier:

Loves(P, Mosquitoes)

Existential Instantiation: Skolemization

- Even though they are written differently:

∃x Loves(x, Mosquitoes)

Loves(P, Mosquitoes)

These two sentences are getting at the same idea: that there is a person that exists that loves mosquitoes.

The second one is just committing to giving them a specific ‘name’ if you will, by using a constant instead of a variable.

Existential Instantiation: Skolemization

- So with that…
- **What is the following logical sentence in English?**
- **And how might you skolemize this sentence?**

$$\exists x(isMansion(x) \land isFixerUpper(x))$$

Existential Instantiation: Skolemization

- This sentence:

$$\exists x(isMansion(x) \land isFixerUpper(x))$$

- In English:
  - “There exists an object that is a mansion and is also a fixer upper”
  - Or in slight better English: “There exists a mansion that is a fixer upper”
- And skolemized:
  - Let’s use the symbol ‘M’ to represent ‘Mansion’ (could have used any name you wished of course).

$$isMansion(M) \land isFixerUpper(M)$$

First Order Logic – Quantifiers

- OK! Phew! We’re first order logic experts now!
  - ~~Scope of quantifiers~~
  - ~~Standardizing Apart~~
  - ~~A convention for denoting Universal Quantifiers~~
  - ~~Skolemization~~

Logics: Review

- **Propositional Logic**: Simple, everything has a truth value. e.g.:

$B_{B1} \leftrightarrow (P_{A1} \lor P_{B2} \lor P_{C1})$ --- *“B_{B1} is true, if and only if P_{A1} is true or P_{B2} is true or P_{C1} is true”*

(good for examples, but not particularly expressive)

- **Predicate Logic**: more expressive, introduced objects and relationships. e.g.:

Brothers(John,Richard) -> Related(John,Richard) ∧ Related(Richard,John) – *If John and Richard are brothers, then John is related to Richard and Richard is related to John*

(and is particularly powerful when combined with)…

- **First Order Logic**: Adds Universal and Existential Quantifiers.

∀x,y Brothers(x,y) -> Related(x,y) ∧ Related(y,x) – *All brothers are related to each other*

And **Boolean logic** can be used with any of the above to add complexity.

Another catch your breath moment…

- OK – we’re about to dive EVEN DEEPER.
- Some of what we’re about to cover next I imagine will be review from other classes you’ve taken, but some might be brand new.
- We’re feeling OK with what we’ve seen so far?

Some Important Equivalencies

OK, here’s the next set of “additional/related” concepts… “Equivalencies” – ways of transforming logical sentences into other, equivalent sentences.

- Negated Quantifiers
- Implications and Disjunctions
- De Morgan’s Laws
- Distribution of OR and AND

Negated Quantifiers

- The Universal and Existential Quantifiers are related: each is the negation of the other.

$\neg \forall x$ loves(John, x) means “it is not true that John loves everything”

- Its negation is:

$\exists x \neg$loves(John, x), which means “there exists something that John does not love.

And hopefully that intuitively makes sense right? If John doesn’t love everything, then there must exist at least one thing that he doesn’t love, yeah?

Negated Quantifiers

- You can kind of think of it as moving the negation "inward"
- And if it "crosses" a Universal quantifier it turns into Existential…
- And if it "crosses" an Existential quantifier, it turns into Universal…

So if we start with:

$\neg \forall x \; loves(John, x)$

We move the negation "inward", and it crosses over the Universal quantifier, transforming it:

$\exists x \; \neg loves(John, x)$

Negated Quantifiers

- **So given that, what is an equivalent for this sentence “There does not exist anyone who loves mosquitoes”?**

$$\neg \exists x \; loves(x, Mosquitoes)$$

Negated Quantifiers

- You move the negation inward, and it turns the existential quantifier into a Universal one:

Starting here:

$\neg \exists x \; loves(x, Mosquitoes)$

Becomes:

$\forall x \; \neg loves(x, Mosquitoes)$

"“Everyone does not love Mosquitoes”

Negated Quantifiers

- And one last one, this time with two positives/two negations:

- **What does this sentence mean?**
- **And what is an equivalent sentence based on negations?**

$$\neg \exists x \neg \mathit{loves}(x, \mathit{IceCream})$$

Negated Quantifiers

- The sentence is a little clunky right now, both in logic and in English.
- In English: "There does not exist anyone who does not love ice cream"

We can make it read a little smoother by addressing that double negative…

$$\neg \exists x \neg \mathit{loves}(x, \mathit{IceCream})$$

Negated Quantifiers

- This ‘inner’ not is already as inner as it can be, so we can’t really do anything with it:

$$\neg \exists x \underbrace{\neg}_{} \mathit{loves}(x, \mathit{IceCream})$$

Negated Quantifiers

- But this ‘outer’ not can be moved inward!

$$\underbrace{\neg}\ \exists x\ \underbrace{\neg}\ \mathit{loves}(x, \mathit{IceCream})$$

Negated Quantifiers

◦ Moving it inward crosses over the existential quantifier, turning it into the universal quantifier.

$$\forall x\ \underbrace{\neg}\ \underbrace{\neg}\ \mathit{loves}(x,\ \mathit{IceCream})$$

Negated Quantifiers

- And now we have two negations right next to each other, which cancel each other out! So we can remove them!

$$\forall x\ \underbrace{\cancel{\neg}}_{}\,\underbrace{\cancel{\neg}}_{}\ \mathit{loves}(x,\ \mathit{IceCream})$$

Negated Quantifiers

- Leaving us with this equivalent sentence! “Everyone loves ice cream!”

$$\forall x \; loves(x, IceCream)$$

- Compare that to the original

$$\neg \exists x \neg loves(x, IceCream)$$

“There does not exist someone that does not love ice cream”

It’s the same sentiment, but conveyed differently (and, IMHO, more clearly!)

Some Important Equivalencies

OK… not so bad… not so bad… moving right along…

- ~~Negated Quantifiers~~
- Implications and Disjunctions
- De Morgan's Laws
- Distribution of OR and AND

Implications and Disjunctions

- Check this out:
- Here’s the completed truth table for implication…
- **What is the truth table for $\neg x \lor y$?**

x	y	$x \to y$
T	T	T
T	F	F
F	T	T
F	F	T

x	y	$\neg x \lor y$
T	T	
T	F	
F	T	
F	F	

Implications and Disjunctions

- It turns out they are the same!
- It turns out that: x → y ↔ $\neg x$ ∨ y

x	y	$x \rightarrow y$
T	T	T
T	F	F
F	T	T
F	F	T

x	y	$\neg x \vee y$
T	T	T
T	F	F
F	T	T
F	F	T

That biconditional above is just a fancy way to say that the expressions $x \rightarrow y$ and $\neg x \vee y$ are equivalent.

x	y	$x \rightarrow y$
T	T	T
T	F	F
F	T	T
F	F	T

x	y	$\neg x \vee y$
T	T	T
T	F	F
F	T	T
F	F	T

Implications and Disjunctior

- That means that when you see a sentence that has an implications in it…

Brothers(Richard,John) -> Related(Richard,John) ∧ Related(John,Richard)

- You can re-write it WITHOUT the implications!
- By negating the 'left hand side' and OR-ing it with the right hand side
- So the above can become

$\neg Brothers(Richard, John) \vee$ *(Related(Richard,John) ∧ Related(John,Richard))*

x	y	$x \to y$
T	T	T
T	F	F
F	T	T
F	F	T

x	y	$\neg x \vee y$
T	T	T
T	F	F
F	T	T
F	F	T

Implications and Disjunctior

◦ The upshot:

Sometimes implications are a pain to work with…

So being able to convert sentences into equivalent ones that *don’t* involve implications is very handy!

Some Important Equivalencies

OK, this next one I’m kind of assuming is review, and I’m just presenting the “what” without going into great detail about the “why”

- ~~Negated Quantifiers~~
- ~~Implications and Disjunctions~~
- De Morgan’s Laws
- Distribution of OR and AND

De Morgan's Laws

$$\neg(x \land y) \leftrightarrow \neg x \lor \neg y$$

$$\neg(x \lor y) \leftrightarrow \neg x \land \neg y$$

$$(x \land y) \leftrightarrow \neg(\neg x \lor \neg y)$$

$$(x \lor y) \leftrightarrow \neg(\neg x \land \neg y)$$

An exercise for you to write truth tables for these!

But if you did, you will find that both sides of each of these biconditionals have the same truth tables!

i.e., it's just like what we looked at before with the "implications and disjunctions" example –

$$x \rightarrow y \leftrightarrow \neg x \lor y$$

if you see one side of the biconditional, you can safely convert it into the other.

De Morgan’s Laws

The upshot of De Morgan’s Laws:

With enough negations (and negation distributions), you can turn an OR into an AND and vice versa.

$\neg(x \wedge y) \leftrightarrow \neg x \vee \neg y$ // “not X AND Y” is the same as “not x or not y”

$\neg(x \vee y) \leftrightarrow \neg x \wedge \neg y$ // “not X OR Y” is the same as “not x AND not y”

$(x \wedge y) \leftrightarrow \neg(\neg x \vee \neg y)$ // “X AND Y” is the same as “the opposite of not x OR not y”

$(x \vee y) \leftrightarrow \neg(\neg x \wedge \neg y)$ // “X OR Y” is the same as “the opposite of not x AND not y”

Some Important Equivalencies

And finally… a way to convert a “disjunction of conjunctions” into a “conjunction of disjunctions”

i.e., : Go from: “a bunch of ANDs being OR-ed together”…

To: “A bunch of Ors being ANDed together”

(which again, we’ll see soon, will be extremely useful for us!)

- ~~Negated Quantifiers~~
- ~~Implications and Disjunctions~~
- ~~De Morgan’s Laws~~
- Distribution of OR and AND

Distribution of OR over AND

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

Another equivalency!

We are distributing the x term to both the y term and the z term.

Note how the left hand side of the biconditional is " THIS OR THAT"

…

And the right hand side of the biconditional is "THIS AND THAT"

You can write a truth table for this too to verify that they are the same.

Distribution of OR over AND

This can be generalized to situations like this (where we start off with a disjunction of conjunctions):

$(w \wedge x) \vee (y \wedge z) \leftrightarrow$

$\big((w \wedge x) \vee y\big) \wedge \big((w \wedge x) \vee z\big)$ Treat ‘w and x’ as a single term, distribute it across the other term

$\big(y \vee (w \wedge x)\big) \wedge \big(z \vee (w \wedge x)\big)$ Simple re-arrangement (moving y and z to the ‘front’)

$\big((y \vee w) \wedge (y \vee x)\big) \wedge \big((z \vee w) \wedge (z \vee x)\big)$ Distribution of OR over AND again (both y and z)

$(y \vee w) \wedge (y \vee x) \wedge (z \vee w) \wedge (z \vee x)$ Cleaning up some parenthesis

$(w \vee y) \wedge (w \vee z) \wedge (x \vee y) \wedge (x \vee z)$ We now have a conjunction of disjunctions!

Some Important Equivalencies

Oh my goodness. Another set of new concepts in the books!

- ~~Negated Quantifiers~~
- ~~Implications and Disjunctions~~
- ~~De Morgan’s Laws~~
- ~~Distribution of OR and AND~~

An Alternative Semantics: Database Semantics

(not what we are working with, but just want you to be introduced the notions)

- **Unique Name Assumption**:
  - Every constant refers to a different object
  - (This means two constants can’t be equal)
- **Domain Closure Assumption**:
  - Only objects that have been specifically named exist.
  - i.e., no objects referred to by functions, only constants!
- **Closed World Assumption**:
  - Facts not explicitly stated as true are assumed to be false.

Skolemization: Revisited

- OK – another deep breath moment.
- We doing OK?
- We're about to see that promised more advanced form of Skolemization…

Skolemization: Revisited

- Consider this statement:

$$\forall x \, \exists y \; breeze(x) \leftrightarrow adjacent(x, y) \land pit(y)$$

- By convention we can drop the Universal Quantifier

$$\exists y \; breeze(x) \leftrightarrow adjacent(x, y) \land pit(y)$$

Can we replace *y* with a Skolem constant?

Skolemization: Revisited

- Let's say that we do: let's replace the 'y' with a constant, S (for 'square'). So that would give us something like this:

- Original: $\forall x\ \exists y\ breeze(x) \leftrightarrow adjacent(x,y) \wedge pit(y)$

- Revised:

$$breeze(x) \leftrightarrow adjacent(x,S) \wedge pit(S)$$

How do we feel about this?

These two statements do not have the same meaning!

Skolemization: Revisited

- Original: $\forall x\ \exists y\ breeze(x) \leftrightarrow adjacent(x,y) \land pit(y)$

"For all squares with a breeze, there exists a square that is adjacent to the square with a breeze and that has a pit."

- Revised: $breeze(x) \leftrightarrow adjacent(x,S) \land pit(S)$

"All squares with a breeze are adjacent to the *same* pit square!"

By using a single constant here, we're now locking in to there being literally a SINGLE object! Which wasn't the meaning of the original sentence!

Skolemization: Revisited

- Original: $\forall x\ \exists y\ breeze(x) \leftrightarrow adjacent(x, y) \wedge pit(y)$

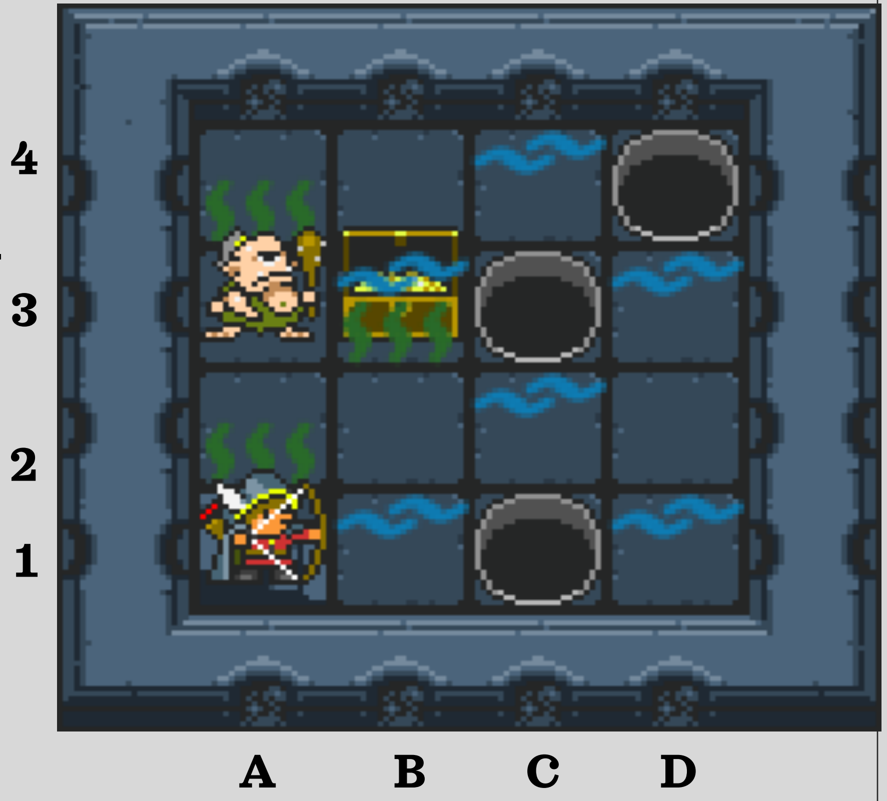
- Due to the fact that the existential quantifier was “inside” of the existential quantifier...
- We’re not saying that there exists a single square that has a pit that is adjacent to all breezy spots….
- We’re saying that, “For Each Square, IF that square is breezy, then it’s adjacent to a pit.

Skolemization: Revisited

- Original: $\forall x\ \exists y\ breeze(x) \leftrightarrow adjacent(x, y) \wedge pit(y)$

- Indeed, you can kind of think of universal quantifiers as 'for loops'.

- So you can say the above is looping through each breeze square, and saying that it is adjacent to a pit. But you can clearly see that not every breeze square is adjacent to the same pit!
  - E.g., the breeze square of B1 is adjacent to the pit of C1
  - But the breeze square of D3 is adjacent to the pit of D4 (and also the pit of C3)



Skolemization: Revisited

- So… how can we address this?
- It would still be nice to use something LIKE using a skolem constant, to get rid of the existential quantifier…
- But we can't just get away with a single constant, since each universally quantified item would need its own constant…

Skolemization: Revisited

- Why… by making use of the notion of ‘functions’ that we introduced with predicate logic!
  - A way to map one object to another!

- That is… when working with existential quantifiers, we won’t convert it into a skolem constant…
  - We’ll convert it into a **skolem function**!

Skolemization: Revisited

- Original: $\forall x \, \exists y \; breeze(x) \leftrightarrow adjacent(x, y) \land pit(y)$

- "Good" Revised with skolem function: $breeze(x) \leftrightarrow adjacent\big(x, \boldsymbol{S}(\boldsymbol{x})\big) \land pit\big(\, \boldsymbol{S}(\boldsymbol{x})\big)$

And this addresses our problem!

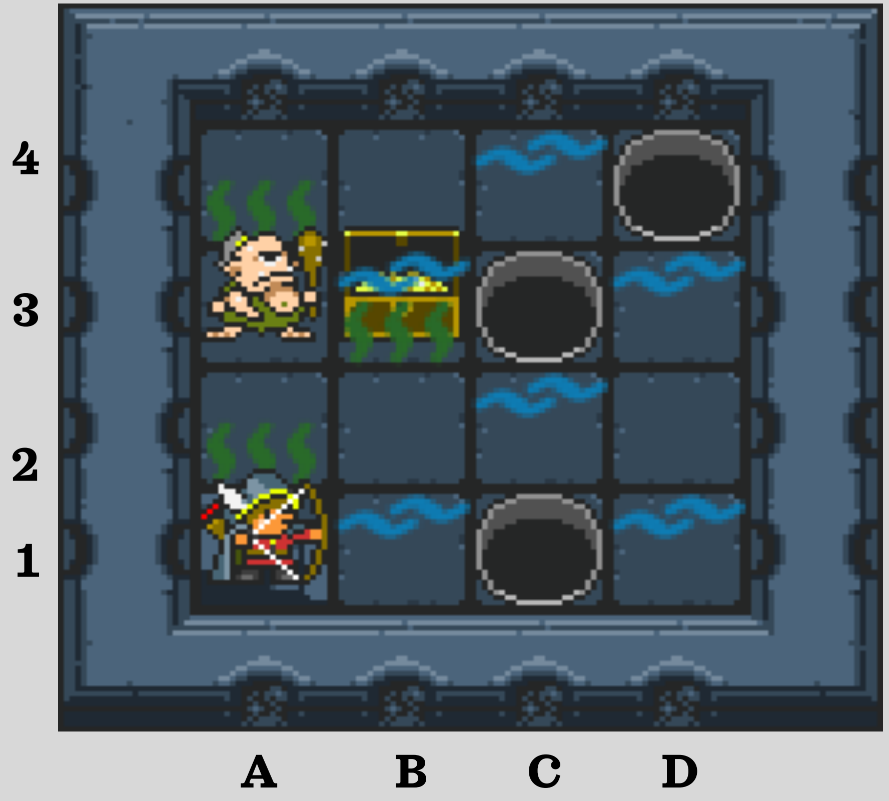
Because as the values of x change, the object that S(x) 'returns' will change too!

(i.e., the function S(x) can refer to any number of squares, instead of the constant S, which only referred to one specific square).

Skolemization: Revisited

$$breeze(x) \leftrightarrow adjacent(x, \boldsymbol{S}(\boldsymbol{x})) \land pit(\boldsymbol{S}(\boldsymbol{x}))$$

- Again, if we think of the Universal quantifier as a for loop, looping through each breezy square…
- Then when 'x' is B1, then S(x) will be C1..
- When 'x' is D1, S(x) will be C1…
- When 'x' is B3, S(x) will be C3…
- When 'x' is C2, S(x) can either be C1 or C3
  - Again, we're just making the claim that there exists "at least one" adjacent square with a pit – there might be more than one!
- And so on and so forth!



Skolemization: Revisited

- The upshot – the important take away:

- When an existential quantifier appears within the scope of one or more universal quantifiers, we don't use a skolem constant, we instead use a **skolem function**.
  - And make the parameters of this function the universally quantified variables.
  - E.g., *x* in the previous slides.

Skolemization: Revisited

- And if it helps you to think about it:

- You can think of the ‘skolem constant’ ALSO as a ‘skolem function’, just a function that takes in 0 parameters!
  - i.e., a function that always returns the same thing.

One Last Breath…

- OK…
- We feeling good one last time?
- Because believe it or not… basically everything that came before is all leading up to this…
- We’re going to start talking about **Conjunctive Normal Form**!
  - Which *itself* has many pieces associated with it!

Conjunctive Normal Form

- Literals
- Disjunctive Clauses
- Conjunctive Normal Form (CNF)
- Converting to CNF

Literals

- A **Literal** is an individual statement (an atomic sentence) or its negation.

Propositional logic example:

- x is a literal
- $\neg x$ is a literal

Predicate logic example:

- $pit(A1)$ is a literal
- $\neg pit(A1)$ is a literal

Disjunctive Clauses

- A **disjunctive clause** is a disjunction (i.e. an OR)…
- Whose **disjunctions** (i.e., its operands, the things you are ORing)…
- Are all literals

e.g., a disjunctive clause looks like this:

$$(X \lor Y)$$

Disjunctive Clauses

- So here are a bunch of disjunctive clauses:

$$King(Richard) \lor King(John)$$

$$\neg Happy(x) \lor Happy(x)$$

$$\neg animal(x) \lor loves(John, x)$$

Disjunctive Clauses

- Note: it doesn't necessarily need to be exactly two literals!
- 1 literal can still be disjunctive clause!
- 3 or more literals can still be a disjunctive clause!
- The following are still disjunctive clauses:

$$animal(x)$$

$$animal(x) \lor vegetable(x) \lor mineral(x)$$

Conjunctive Normal Form

- An expression is in **Conjunctive Normal Form** (CNF) if it is a conjunction of disjunctive clauses.
- i.e., CNF is “ANDing a bunch of ORs”

$$\overbrace{\big(Animal(x)\big)}^{\text{Disjunctive Clause}} \underbrace{\land}_{\text{AND}} \overbrace{\big(\neg Animal(x) \lor Loves(John, x)\big)}^{\text{Disjunctive Clause}}$$

Conjunctive Normal Form

- An expression is in **Conjunctive Normal Form** (CNF) if it is a conjunction of disjunctive clauses.
- i.e., CNF is “ANDing a bunch of Ors”

$$\overbrace{\left(Animal(x)\right) \wedge \left(\neg Animal(x) \vee Loves(John, x)\right)}^{\text{CNF Expression}}$$

Conjunctive Normal Form

- A CNF expression can have as many clauses as you wish.

- **How many clauses are in the following CNF Expression?**

$$(Hungry(x)) \land (Animal(x)) \land (\neg Animal(x) \lor Loves(John, x)) \land (\mathrm{Human(John)} \lor \mathrm{Animal(John)})$$

Conjunctive Normal Form

- A CNF expression can have as many clauses as you wish.

- **How many clauses are in the following CNF Expression?**
  - There are four! Each clause is separated by an ‘AND’

$$\overbrace{(Hungry(x))}^{\text{Clause}} \wedge \overbrace{(Animal(x))}^{\text{Clause}} \wedge \overbrace{(\neg Animal(x) \vee Loves(John, x))}^{\text{Clause}} \wedge \overbrace{(\mathrm{Human(John)} \vee \text{Animal(John)})}^{\text{Clause}}$$

Conjunctive Normal Form

- All logical expressions can be converted into conjunctive normal form.
- This allows us to standardize input to logical algorithms!
- And how do you do the converting?
- Why… by drawing upon EVERYTHING WE’VE LEARNED TODAY!

Converting to CNF

1. Eliminate Implications
2. Move $\neg$ Inwards
3. Standardize Variable Names Apart
4. Skolemize
5. Drop Universal Quantifiers
6. Apply De Morgan's Laws

Converting to CNF

- Convert the following to CNF:
- Note all the things it has!
  - Nested universal quantifiers! TWO implications! An Existential quantifier “within” a Universal Quantifier! The same symbol ‘y’ used in different contexts!

$$\forall x \left(\forall y\ Animal(y) \rightarrow Loves(x, y)\right) \rightarrow \exists y\ Loves(y, x)$$

“Everyone who loves all animals is loved by someone”

Converting to CNF

Step 1: Eliminate the implications

- We remember that $(x \rightarrow y) \leftrightarrow (\neg x \vee y)$

$$\forall x \left(\forall y\ Animal(y) \rightarrow Loves(x,y)\right) \rightarrow (\exists y\ Loves(y,x))$$

Becomes…

Converting to CNF

Step 1: Eliminate the implications

- We remember that $(x \rightarrow y) \leftrightarrow (\neg x \lor y)$

$$\forall x \left(\forall y\ Animal(y) \rightarrow Loves(x,y)\right) \underbrace{\rightarrow}_{\text{Start here}} (\exists y\ Loves(y,x))$$

Becomes…

We have two implications here. Let’s work with the ‘outer’ one first.

Converting to CNF

Step 1: Eliminate the implications

- We remember that $(x \rightarrow y) \leftrightarrow (\neg x \vee y)$

$$\forall x \; \underbrace{\left(\forall y\, Animal(y) \rightarrow Loves(x,y)\right)}_{\substack{\text{The left hand side of the implication} \\ \text{“x”}}} \rightarrow \underbrace{(\exists y\, Loves(y,x))}_{\substack{\text{The right hand side} \\ \text{“y”}}}$$

Becomes…

Converting to CNF

Step 1: Eliminate the implications

- We remember that $(x \rightarrow y) \leftrightarrow (\neg x \lor y)$

$$\forall x\ \left(\forall y\ Animal(y)\ \rightarrow Loves(x,y)\right) \rightarrow (\exists y\ Loves(y,x))$$

Becomes…

$$\forall x\ (\neg\left(\forall y\ Animal(y)\ \rightarrow Loves(x,y)\right)) \lor (\exists y\ Loves(y,x))$$

One implication down, one to go!

Converting to CNF

Step 1: Eliminate the implications

- We remember that $(x \rightarrow y) \leftrightarrow (\neg x \vee y)$

$$\forall x\ (\neg(\ \forall y\ Animal(y)\ \rightarrow Loves(x, y)\ )\ ) \vee (\exists y\ Loves(y, x))$$

Becomes…

We just have that one inner implication now..

Converting to CNF

Step 1: Eliminate the implications

- We remember that $(x \rightarrow y) \leftrightarrow (\neg x \vee y)$

$$\forall x\ (\neg(\ \forall y\ Animal(y)\ \rightarrow Loves(x,y)\ )\ )\vee(\exists y\ Loves(y,x))$$

Becomes…

Converting to CNF

Step 1: Eliminate the implications

- We remember that $(x \rightarrow y) \leftrightarrow (\neg x \vee y)$

$$\forall x\ (\neg(\ \forall y\ Animal(y)\ \rightarrow Loves(x,y)\ )\ ) \vee (\exists y\ Loves(y,x))$$

Becomes…

$$\forall x\ \left(\neg(\forall y\ \neg Animal(y) \vee Loves(x,y))\right) \vee (\exists y\ Loves(y,x))$$

Converting to CNF

Step 2: Move ¬ Inwards

Remember: Negated quantifiers & De Morgan's laws.

$$\forall x \left(\boxed{\neg} (\forall y \, \neg Animal(y) \lor Loves(x,y)) \right) \lor \left(\exists y \, Loves(y,x) \right)$$

This guy is gonna get "distributed"

Becomes…

$$\forall x \left(\exists y \, Animal(y) \land \neg Loves(x,y) \right) \lor (\exists y \, Loves(y,x))$$

"Either there is some animal that x does not love, or someone loves x"

Converting to CNF

Step 3: Standardize Variable Names Apart

Remember: This means using unique variable names

$$\forall x\ \big(\exists y\ Animal(y) \land \neg Loves(x, y)\big) \lor (\exists y\ Loves(y, x))$$

Becomes…

$$\forall x\ \big(\exists y\ Animal(y) \land \neg Loves(x, y)\big) \lor \big(\exists z\ Loves(z, x)\big)$$

We made it so that the second existential quantifier has its own variable.

Converting to CNF

Step 4: Skolemize

Remember: This means subbing out Existential Quantifiers with either constants or functions (functions if universally quantified)

$$\forall x\, \left(\exists y\, Animal(y) \land \neg Loves(x,y)\right) \lor \left(\exists z\, Loves(z,x)\right)$$

Becomes…

$$\forall x\, \left(Animal(F(x)) \land \neg Loves(x,F(x))\right) \lor (Loves(G(x),x))$$

The variable referred to by $\exists y$ was replaced with function *F(x)*

The variable referred to by $\exists z$ was replaced with function *G(x)*

Converting to CNF

Step 5: Drop Universal Quantifiers

Remember: we assume variables are universally quantified

$$\forall x \left(Animal(F(x)) \land \neg Loves(x, F(x))\right) \lor (Loves(G(x), x))$$

Becomes…

$$\left(Animal(F(x)) \land \neg Loves(x, F(x))\right) \lor (Loves(G(x), x))$$

We simply dropped the $\forall x$ symbol

Converting to CNF

Step 6: De Morgan's Laws

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

$$\big(Animal(F(x)) \wedge \neg Loves(x, F(x))\big) \vee (Loves(G(x), x))$$

Becomes…

$(Loves(G(x), x)) \vee \big(Animal(F(x)) \wedge \neg Loves(x, F(x))\big) \quad - -\text{rearrange}$

Converting to CNF

Step 6: De Morgan's Laws

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

$$\big(Animal(F(x)) \wedge \neg Loves(x, F(x))\big) \vee (Loves(G(x), x))$$

Becomes…

$(Loves(G(x), x)) \vee \big(Animal(F(x)) \wedge \neg Loves(x, F(x))\big) \quad - -\text{rearrange}$

Converting to CNF

Step 6: De Morgan's Laws

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

$$\big(Animal(F(x)) \wedge \neg Loves(x, F(x))\big) \vee (Loves(G(x), x))$$

Becomes…

$(Loves(G(x), x)) \vee \big(Animal(F(x)) \wedge \neg Loves(x, F(x))\big) \quad - -\text{rearrange}$

Converting to CNF

Step 6: De Morgan's Laws

Remember: $x \lor (y \land z) \leftrightarrow (x \lor y) \land (x \lor z)$

$$\big(Animal(F(x)) \land \neg Loves(x, F(x))\big) \lor (Loves(G(x), x))$$

Becomes…

$$(Loves(G(x), x)) \lor \big(Animal(F(x)) \land \neg Loves(x, F(x))\big) \quad - -\text{rearrange}$$

$$(Loves(G(x), x) \lor Animal(F(x))) \land (Loves(G(x), x) \lor \neg Loves(x, F(x)))$$

Converting to CNF

Step 6: De Morgan's Laws

Remember: $x \lor (y \land z) \leftrightarrow (x \lor y) \land (x \lor z)$

$$\big(Animal(F(x)) \land \neg Loves(x, F(x))\big) \lor (Loves(G(x), x))$$

Becomes…

$$(Loves(G(x), x)) \lor \big(Animal(F(x)) \land \neg Loves(x, F(x))\big) \quad --\text{rearrange}$$

$$\underbrace{(Loves(G(x), x) \lor Animal(F(x)))}_{\text{Clause}} \land \underbrace{(Loves(G(x), x) \lor \neg Loves(x, F(x)))}_{\text{Clause}}$$

We did it!

Converting to CNF

Step 7: Celebrate

Look how far we came!

Original: $\forall x \left(\forall y \, Animal(y) \rightarrow Loves(x, y)\right) \rightarrow \exists y \, Loves(y, x)$

Final: $(Loves(G(x), x) \vee Animal(F(x))) \wedge (Loves(G(x), x) \vee \neg Loves(x, F(x)))$

And someday soon we'll hopefully see why it was worth the effort!