Mathematical Logic

Why We Overview Logic at This Moment? [1/3]

- Logic을 Computer Science에서 배워야 하는 이유?

- So Many Logic-based Applications in CS!
  - Boolean logic for digital circuit design
  - First order logic for Relational Database Query Languages
  - Various logic derivatives for AI Knowledge Presentation
  - Many many tools for semantics representations in various fields like WWW

- Besides the above reason!
  - Logic의 표현을 이용해야 Formal하고 Theoretical한 내용을 (definition, property, lemma, theorem) 표현할 때에 명쾌하게 의미전달을 할 수 있다!
  - Logic을 이용해야 Mathematical & Formal한 proof를 derive 할 수 있다!

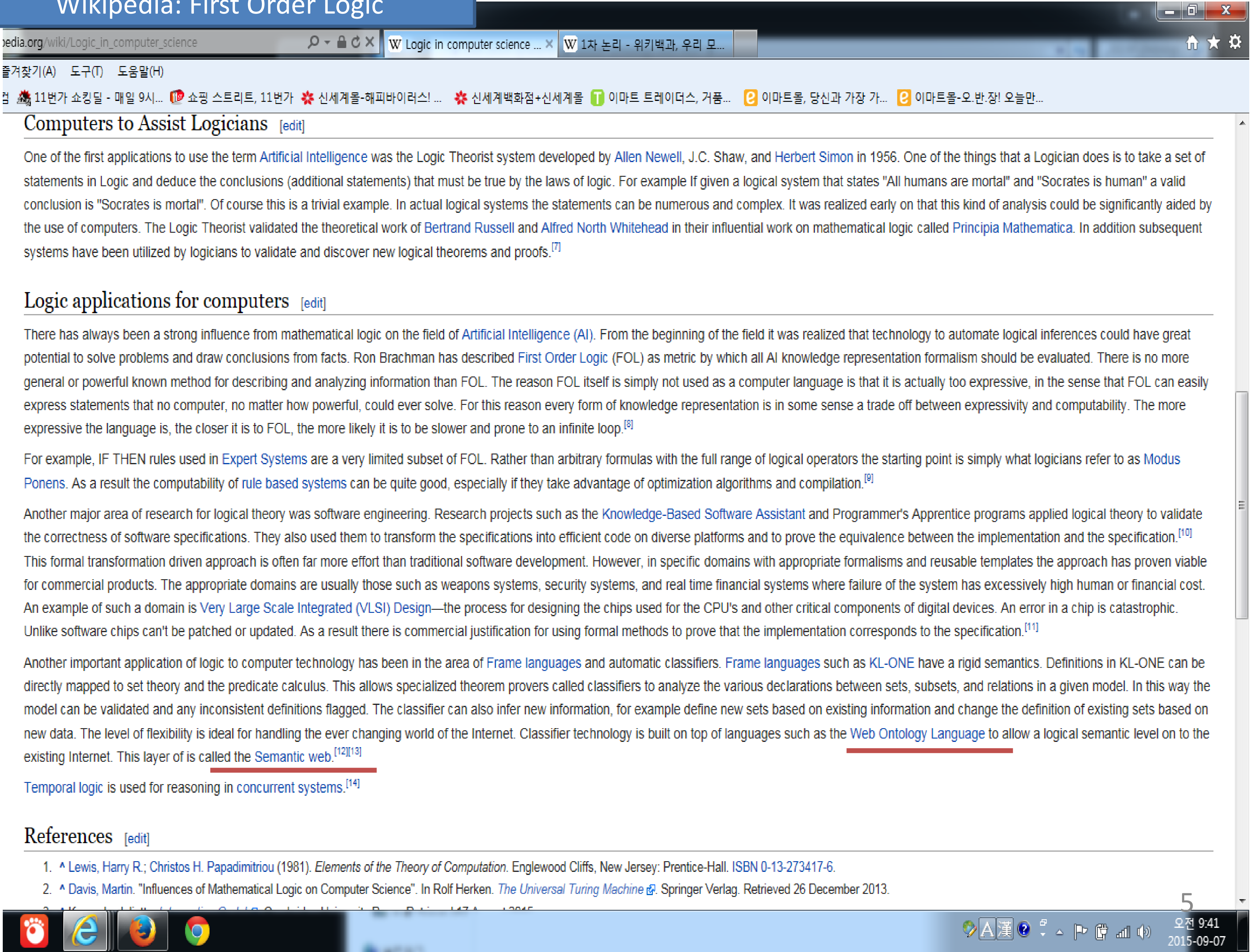
Why We Overview Logic at This Moment? [2/3]

- 그러면 왜 우리과목에서는 Logic을 review하는가?
- Semantic Web에서 Semantics을 capture & represent & utilize하려고 하는데..
- If so, 결국 Logic혹은 Logic derivatives들에 의존해야 하는데.....
- RDF, OWL 등도 전부 root가 Logic에 있는데....
- 가장 기초 Logic은 Propositional Logic, 그 다음 level의 Logic은 First Order Logic인데..
  - 자연어를 Logic으로 Transform하는 framework을 배워야 하고
  - 꼭 명심해야 할 Limits of Computation을 숙지해야 할 필요가 있다.

Why We Overview Logic at This Moment? [3/3]

- Propositional Logic and First Order Logic
- Set theory ➔ 풀수없는 문제가 존재한다는 증명
- Limits of Computations
  - **Satisfiability in propositional logic** is an NP-complete problem
  - **The expressive power of First Order Logic is Turing-complete**
  - **Halting problem in a programming language is undecidable을 증명**
  - **Satisfiability in first-order logic** is undecidable

Wikipedia: First Order Logic


pedia.org/wiki/Logic_in_computer_science
Logic in computer science ...
1차 논리 - 위키백과, 우리 모...
즐겨찾기(A) 도구(T) 도움말(H)
11번가 쇼킹딜 - 매일 9시...
쇼핑 스트리트, 11번가
신세계몰-해피바이러스! ...
신세계백화점+신세계몰
이마트 트레이더스, 거품...
이마트몰, 당신과 가장 가...
이마트몰-오.반.장! 오늘만...
오전 9:41
2015-09-07

Computers to Assist Logicians [edit]

One of the first applications to use the term Artificial Intelligence was the Logic Theorist system developed by Allen Newell, J.C. Shaw, and Herbert Simon in 1956. One of the things that a Logician does is to take a set of statements in Logic and deduce the conclusions (additional statements) that must be true by the laws of logic. For example If given a logical system that states "All humans are mortal" and "Socrates is human" a valid conclusion is "Socrates is mortal". Of course this is a trivial example. In actual logical systems the statements can be numerous and complex. It was realized early on that this kind of analysis could be significantly aided by the use of computers. The Logic Theorist validated the theoretical work of Bertrand Russell and Alfred North Whitehead in their influential work on mathematical logic called Principia Mathematica. In addition subsequent systems have been utilized by logicians to validate and discover new logical theorems and proofs.[7]

Logic applications for computers [edit]

There has always been a strong influence from mathematical logic on the field of Artificial Intelligence (AI). From the beginning of the field it was realized that technology to automate logical inferences could have great potential to solve problems and draw conclusions from facts. Ron Brachman has described First Order Logic (FOL) as metric by which all AI knowledge representation formalism should be evaluated. There is no more general or powerful known method for describing and analyzing information than FOL. The reason FOL itself is simply not used as a computer language is that it is actually too expressive, in the sense that FOL can easily express statements that no computer, no matter how powerful, could ever solve. For this reason every form of knowledge representation is in some sense a trade off between expressivity and computability. The more expressive the language is, the closer it is to FOL, the more likely it is to be slower and prone to an infinite loop.[8]

For example, IF THEN rules used in Expert Systems are a very limited subset of FOL. Rather than arbitrary formulas with the full range of logical operators the starting point is simply what logicians refer to as Modus Ponens. As a result the computability of rule based systems can be quite good, especially if they take advantage of optimization algorithms and compilation.[9]

Another major area of research for logical theory was software engineering. Research projects such as the Knowledge-Based Software Assistant and Programmer's Apprentice programs applied logical theory to validate the correctness of software specifications. They also used them to transform the specifications into efficient code on diverse platforms and to prove the equivalence between the implementation and the specification.[10] This formal transformation driven approach is often far more effort than traditional software development. However, in specific domains with appropriate formalisms and reusable templates the approach has proven viable for commercial products. The appropriate domains are usually those such as weapons systems, security systems, and real time financial systems where failure of the system has excessively high human or financial cost. An example of such a domain is Very Large Scale Integrated (VLSI) Design—the process for designing the chips used for the CPU's and other critical components of digital devices. An error in a chip is catastrophic. Unlike software chips can't be patched or updated. As a result there is commercial justification for using formal methods to prove that the implementation corresponds to the specification.[11]

Another important application of logic to computer technology has been in the area of Frame languages and automatic classifiers. Frame languages such as KL-ONE have a rigid semantics. Definitions in KL-ONE can be directly mapped to set theory and the predicate calculus. This allows specialized theorem provers called classifiers to analyze the various declarations between sets, subsets, and relations in a given model. In this way the model can be validated and any inconsistent definitions flagged. The classifier can also infer new information, for example define new sets based on existing information and change the definition of existing sets based on new data. The level of flexibility is ideal for handling the ever changing world of the Internet. Classifier technology is built on top of languages such as the Web Ontology Language to allow a logical semantic level on to the existing Internet. This layer of is called the Semantic web.[12][13]

Temporal logic is used for reasoning in concurrent systems.[14]

References [edit]

1. ^ Lewis, Harry R.; Christos H. Papadimitriou (1981). *Elements of the Theory of Computation*. Englewood Cliffs, New Jersey: Prentice-Hall. ISBN 0-13-273417-6.
2. ^ Davis, Martin. "Influences of Mathematical Logic on Computer Science". In Rolf Herken. *The Universal Turing Machine*. Springer Verlag. Retrieved 26 December 2013.

Mathematical Logic

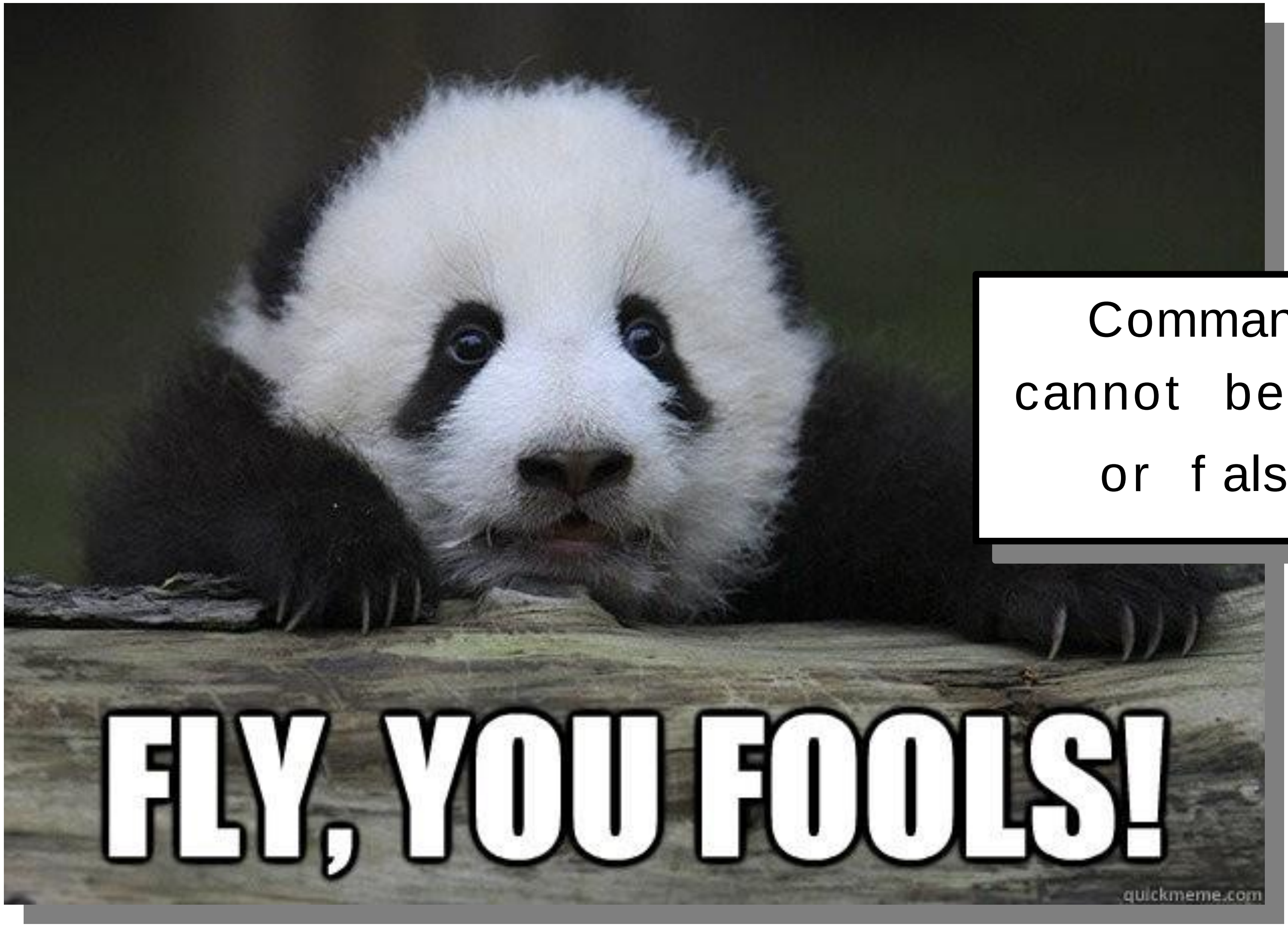
Question: How do we formalize the logic we've been using in our proofs?

- **Propositional Logic**
  - Basic logical connectives.
  - Truth tables.
  - Logical equivalences.
- **First-Order Logic**
  - Reasoning about properties of multiple objects.

Propositional Logic

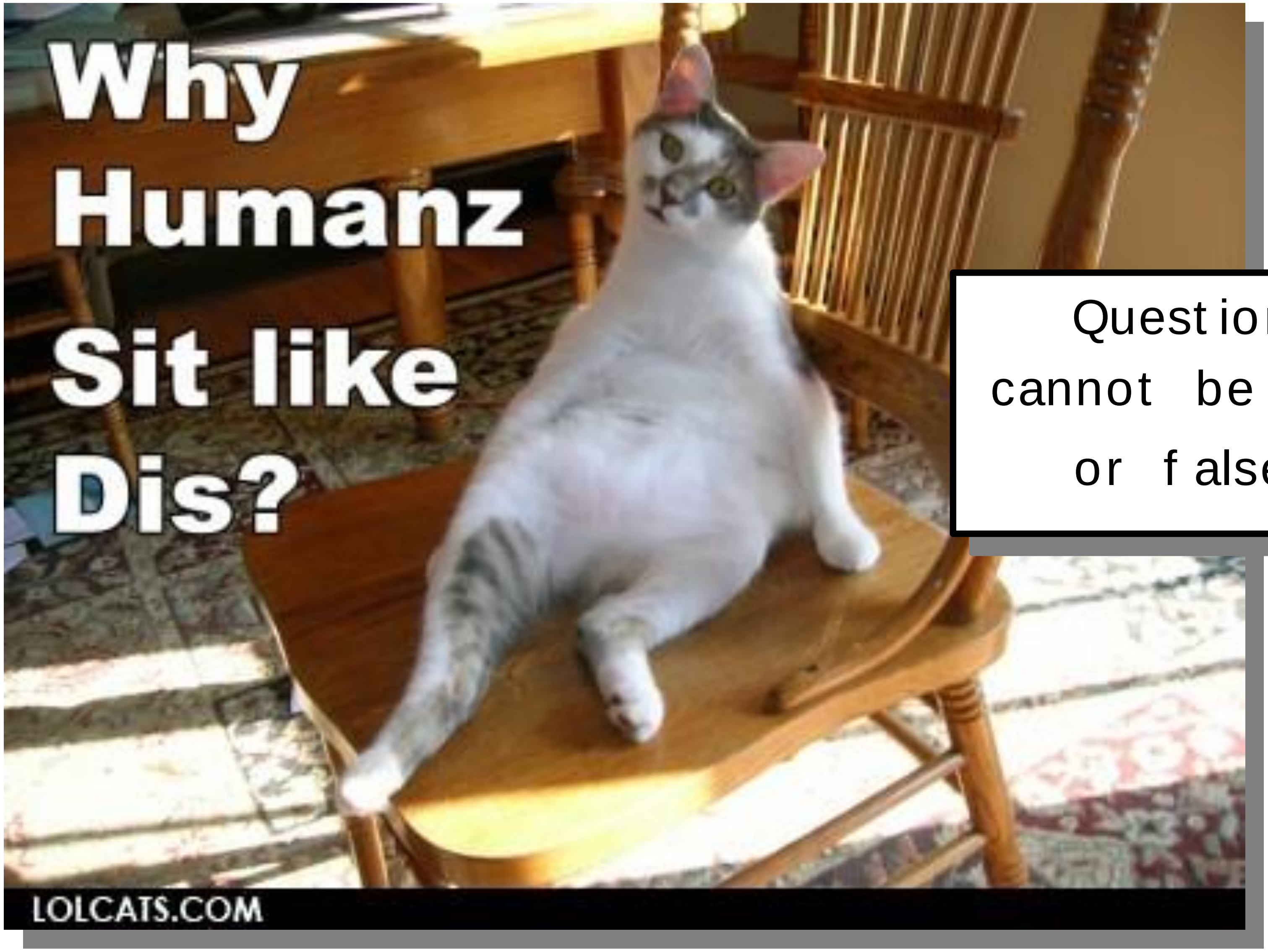
- A ***proposition*** is a statement that is, by itself, either true or false.
- Some Sample Propositions
  - Puppies are cuter than kittens.
  - Kittens are cuter than puppies.
  - Usain Bolt can outrun everyone in this room.
  - CS103 is useful for cocktail parties.
  - This is the last entry on this list.
  - I'm so pretty
  - I'm too hot
  - Called a policeman and a fireman
  - Uptown funk gave it to you

Things That Aren't Propositions



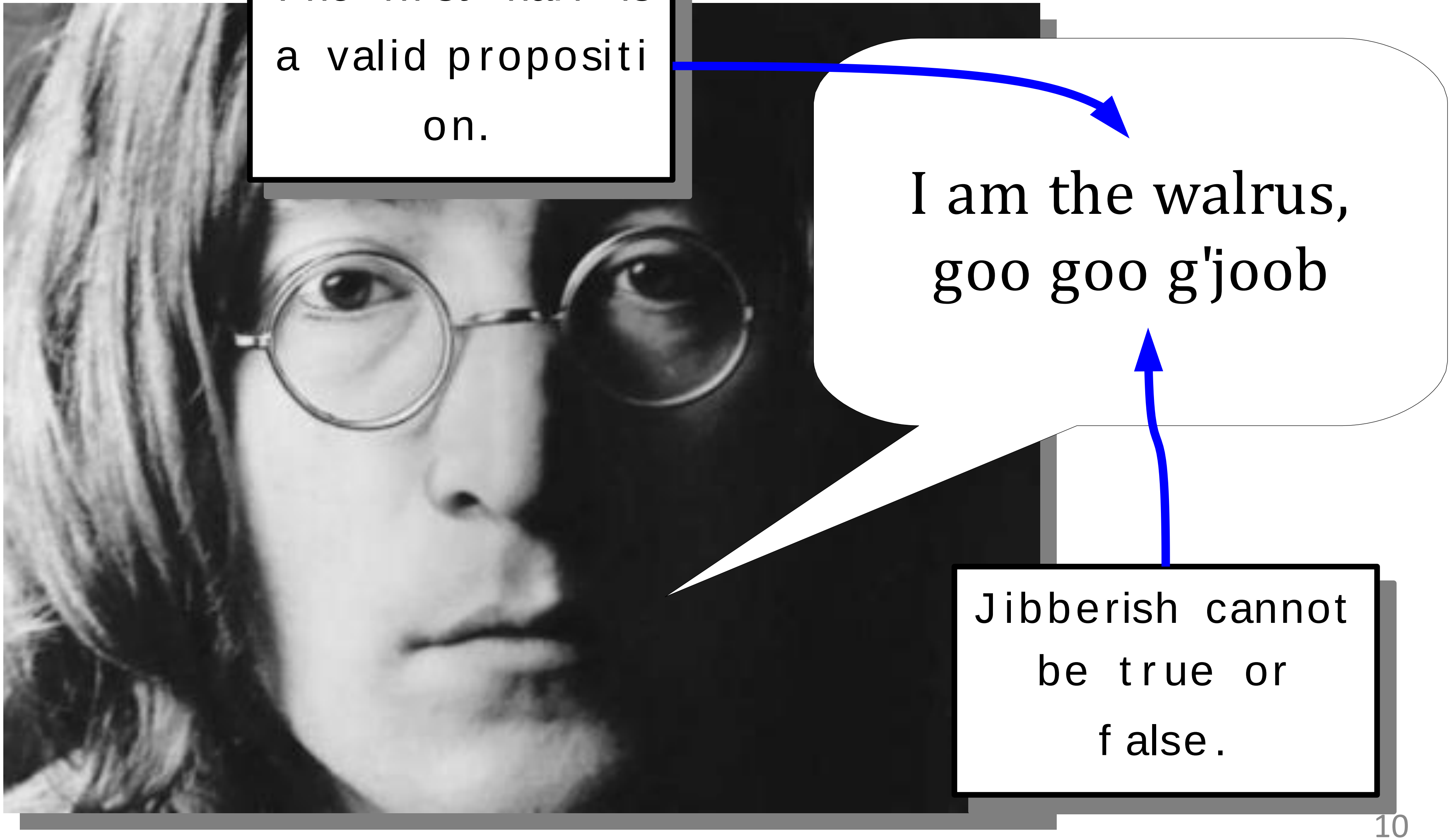
Commands cannot be true or false.

Things That Aren't Propositions



Questions cannot be true or false.

Things That Aren't Propositions



The first half is a valid proposition.

Jibberish cannot be true or false.

Propositional Logic

- ***Propositional logic*** is a mathematical system for reasoning about propositions and how they relate to one another.
- Every statement in propositional logic consists of ***propositional variables*** combined via ***propositional connectives***.
- Each variable represents some proposition, such as "You liked it" or "You should have put a ring on it."
- Connectives encode how propositions are related, such as "If you liked it, then you should have put a ring on it."

Propositional Variables

- Each proposition will be represented by a ***propositionalvariable***.
- Propositional variables are usually represented as lower–case letters, such as *p*, *q*, *r*, *s*, etc.
- Each variable can take on one of two values: true or false.

Propositional Connectives

- **Logical NOT: $\neg p$**
  - Read "***not*** *p*"
  - $\neg p$ is true if and only if p is false.
  - Also called **logical negation**.

- **Logical AND: $p \land q$**
  - Read "*p* ***and*** *q*."
  - $p \land q$ is true if both p and q are true.
  - Also called **logical conjunction**.

- **Logical OR: $p \lor q$**
  - Read "*p* ***or*** *q*."
  - $p \lor q$ is true if at least one of *p* or *q* are true (inclusive OR)
  - Also called **logical disjunction**.

Truth Tables

- A ***truth table*** is a table showing the truth value of a propositional logic formula as a function of its inputs.
- Useful for several reasons:
  - Formally defining what a connective "means."
  - Deciphering what a complex propositional formula means.

The Truth Table Tool

Inclusive OR operator

- The $\vee$ operator is an *inclusive* "or"
- It's true if at least one of the operands is true.
  - Similar to the || operator in C, C++, Java and the **or** operator in Python.
  - If we need an exclusive "or" operator, we can build it out of what we already have.

Truth Table for $\rightarrow$ (Implication)

- The $\rightarrow$ connective is used to represent implications.
  - Its technical name is the ***material conditional*** operator.
- The truth values of the $\rightarrow$ are the way they are because they're *defined* that way.
- The intuition:
  - We want $p \rightarrow q$ to mean "whenever p is true, q is true as well."
  - The only way this *doesn't* happen is if p is true and q is false.
  - In other words, $p \rightarrow q$ should be true whenever $\neg(p \land \neg q)$ is true.
- What's the truth table for $\neg(p \land \neg q)$?

Truth Table for Implication

p	q	$p \to q$
F	F	T
F	T	T
T	F	F
T	T	T

The only way for $p \to q$ to be false is for p to be true and q to be false. Otherwise, $p \to q$ is *by definition* true.

The Biconditional Operator

- The ***biconditional*** connective $p \leftrightarrow q$ is read "*p* if and only if *q*".
- The biconditional operator $\leftrightarrow$ is used to represent a two-directional implication.
  - Specifically, $p \leftrightarrow q$ means that p implies q and q implies p.
  - What should its truth table look like?
- Here's its truth table:

p	q	$p \leftrightarrow q$
F	F	T
F	T	F
T	F	F
T	T	T

One interpretation of $\leftrightarrow$ is to think of it as equality: the two propositions must have equal truth values.

True and False

- There are two more “connectives” to speak of:
  true and false.
- The symbol $\top$ is a value that is always true.
- The symbol $\perp$ is value that is always false.
- These are often called connectives, though they don't connect anything.
- (Or rather, they connect zero things.)

Operator Precedence [1/2]

- How do we parse this statement?

$$\neg x \to y \vee z \to x \vee y \wedge z$$

- Operator precedence for propositional logic:
  - $\neg$
  - $\wedge$
  - $\vee$
  - $\to$
  - $\leftrightarrow$

- All operators are right-associative.
- We can use parentheses to disambiguate.

$$(\neg x) \to ((y \vee z) \to (x \vee (y \wedge z)))$$

Operator Precedence [2/2]

- The main points to remember:
  - $\neg$ binds to whatever immediately follows it.
  - $\wedge$ and $\vee$ bind more tightly than $\rightarrow$.
- We will commonly write expressions like $p \wedge q \rightarrow r$ without adding parentheses.
- For more complex expressions, we'll try to add parentheses.

Recap So Far

- A ***propositional variable*** is a variable that is either true or false.
- The ***propositional connectives*** are
  - Negation: $\neg p$
  - Conjunction: $p \wedge q$
  - Disjunction: $p \vee q$
  - Implication: $p \rightarrow q$
  - Biconditional: $p \leftrightarrow q$
  - True: $\top$
  - False: $\bot$

Translating into Propositional Logic [1/2]

- Some Sample Propositions
  - *a*: I will get up early this morning
  - *b*: There is a lunar eclipse this morning
  - *c*: There are no clouds in the sky this morning
  - *d*: I will see the lunar eclipse

"I won't see the lunar eclipse if I don't get up early this morning."

$$\neg a \rightarrow \neg d$$

Translating into Propositional Logic [2/2]

- Some Sample Propositions
  - *a*: I will get up early this morning
  - *b*: There is a lunar eclipse this morning
  - *c*: There are no clouds in the sky this morning
  - *d*: I will see the lunar eclipse

"If I get up early this morning, but it's cloudy outside, I won't see the lunar eclipse.

$$a \land \neg c \rightarrow \neg d$$

"p if q" translates to $q \rightarrow p$

It does *not* translate to $p \rightarrow q$

"p, not q" translates to $p \wedge \neg q$

"p, but q" translates to $p \wedge q$

The Takeaway Point

- When translating into or out of propositional logic, be very careful not to get tripped up by nuances of the English language.

- In fact, this is one of the reasons we have a symbolic notation in the first place!

- Many prepositional phrases lead to counterintuitive translations;

- So need to make sure to double-check yourself!

Propositional Equivalences

Quick Question

What would I have to show you to convince you that the statement $p \wedge q$ is false?

Quick Question

What would I have to show you to convince you that the statement $p \vee q$ is false?

De Morgan's Laws

- Using truth tables, we concluded that

  $\neg(p \wedge q)$ is equivalent to $\neg p \vee \neg q$

- We also saw that

  $\neg(p \vee q)$ is equivalent to $\neg p \wedge \neg q$

- These two equivalences are called ***De Morgan's Laws***.

Logical Equivalence

- Because $\neg(p \land q)$ and $\neg p \lor \neg q$ have the same truth tables, we say that they're ***equivalent*** to one another.
- We denote this by writing

$$\neg(p \land q) \equiv \neg p \lor \neg q$$

- The $\equiv$ symbol is not a connective.

- The statement $\neg(p \land q) \leftrightarrow (\neg p \lor \neg q)$ is a propositional formula. If you plug in different values of p and q, it will evaluate to a truth value. It just happens to evaluate to true every time.
- The statement $\neg(p \land q) \equiv \neg p \lor \neg q$ means "these two formulas have exactly the same truth table."

- In other words, the notation $\phi \equiv \psi$ means "ϕ and ψ always have the same truth values, regardless of how the variables are assigned."

An Important Equivalence

- Earlier, we talked about the truth table for $p \to q$. We chose it so that

$$p \to q \equiv \neg(p \land \neg q)$$

- Later on, this equivalence will be incredibly useful:

$$\neg(p \to q) \equiv p \land \neg q$$

Another Important Equivalence

- Here's a useful equivalence. Start with

$$p \to q \equiv \neg(p \wedge \neg q)$$

- By De Morgan's laws:

$$\begin{aligned} p \to q &\equiv \neg(p \wedge \neg q) \\ &\equiv \neg p \vee \neg\neg q \\ &\equiv \neg p \vee q \end{aligned}$$

- Thus $p \to q \equiv \neg p \vee q$

If p is false, then $\neg p \vee q$ is true. If p is true, then q has to be true for the whole expression to be true.

The Contrapositive (대우)

- The contrapositive of the statement

$$p \rightarrow q$$

  is the statement

$$\neg q \rightarrow \neg p$$

- These are logically equivalent, which is why proof by contradiction works:

$$p \rightarrow q \quad \equiv \quad \neg q \rightarrow \neg p$$

Recap So Far

- A ***propositional variable*** is a variable that is either true or false.
- The ***propositional connectives*** are
  - Negation: $\neg p$ / Conjunction: $p \land q$ / Disjunction: $p \lor q$
  - Implication: $p \to q$ / Biconditional: $p \leftrightarrow q$ / True: $\top$ / False: $\bot$

- Two propositional formulas ϕ and ψ are called ***equivalent*** if they have the same truth tables.
- We denote this by writing $\phi \equiv \psi$. Some examples
  - $\neg(p \land q) \equiv \neg p \lor \neg q$
  - $\neg(p \lor q) \equiv \neg p \land \neg q$
  - $\neg p \lor q \equiv p \to q$
  - $p \land \neg q \equiv \neg(p \to q)$

Why All This Matters

- Suppose we want to prove the following statement:
  - "If $x + y = 16$, then $x \geq 8$ or $y \geq 8$"

$$x + y = 16 \rightarrow x \geq 8 \lor y \geq 8$$

Using a contrapositive!

$$x < 8 \land y < 8 \rightarrow x + y \neq 16$$

"If $x < 8$ and $y < 8$, then $x + y \neq 16$"

Formally Dressed-Up Writing

Theorem: If $x + y = 16$, then $x \geq 8$ or $y \geq 8$.

Proof: By contrapositive. We prove that if $x < 8$ and $y < 8$, then $x + y \neq 16$. To see this, note that

$$\begin{aligned} x + y &< 8 + y \\ &< 8 + 8 \\ &= 16 \end{aligned}$$

This means that $x + y < 16$, so $x + y \neq 16$, which is what we needed to show. ■

Why All This Matters

- Suppose we want to prove the following statement:

  "If $x + y = 16$, then $x \geq 8$ or $y \geq 8$"

We can rephrase the above statement into

$$x + y = 16 \rightarrow x \geq 8 \vee y \geq 8$$

Suppose the above statement is false,
we can have the below statement

$$x + y = 16 \wedge x < 8 \wedge y < 8$$

"$x + y = 16$, but $x < 8$ and $y < 8$."

But this statement is a contradiction!

Formally Dressed-Up Writing

Theorem: If $x + y = 16$, then $x \geq 8$ or $y \geq 8$.

Proof: Assume for the sake of contradiction that $x + y = 16$, but $x < 8$ and $y < 8$. Then

$$\begin{aligned} x + y &< 8 + y \\ &< 8 + 8 \\ &= 16 \end{aligned}$$

So $x + y < 16$, contradicting that $x + y = 16$. We have reached a contradiction, so our assumption must have been wrong. Therefore if $x + y = 16$, then $x \geq 8$ or $y \geq 8$. ■

Why This Matters

- Propositional logic is a tool for reasoning about how various statements affect one another.
- To better understand how to prove a result, it often helps to translate what you're trying to prove into propositional logic first.
- However as you saw in the previous example....

$$\boxed{\text{If } x + y = 16\text{, then } x \geq 8 \text{ or } y \geq 8.}$$

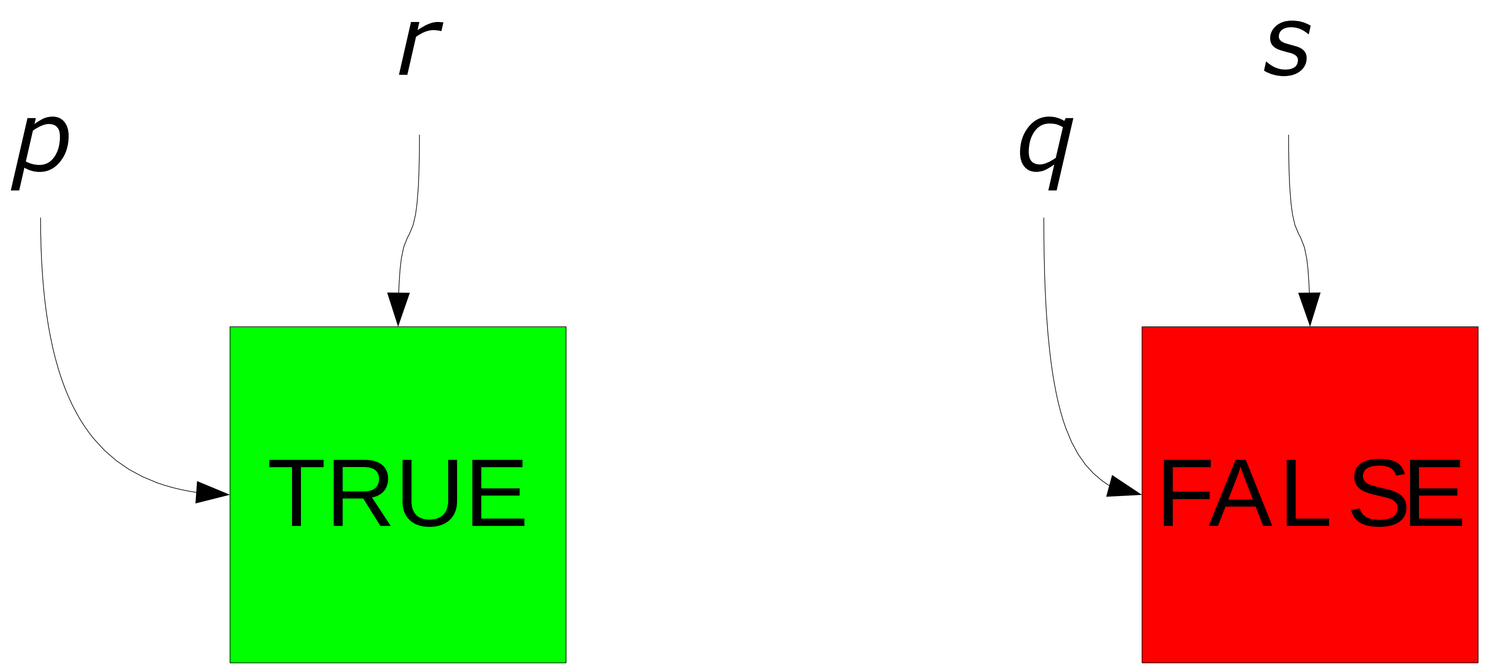
- The propositional logic isn't expressive enough to capture all statements. For that, we need something more powerful.

What is First-Order Logic?

- ***First-order logic*** is a logical system for reasoning about properties of objects.
- Augments the logical connectives from propositional logic with
  - ***predicates*** that describe properties of objects, and
  - ***functions*** that map objects to one another,
  - ***quantifiers*** that allow us to reason about multiple objects simultaneously.

The Universe of Propositional Logic

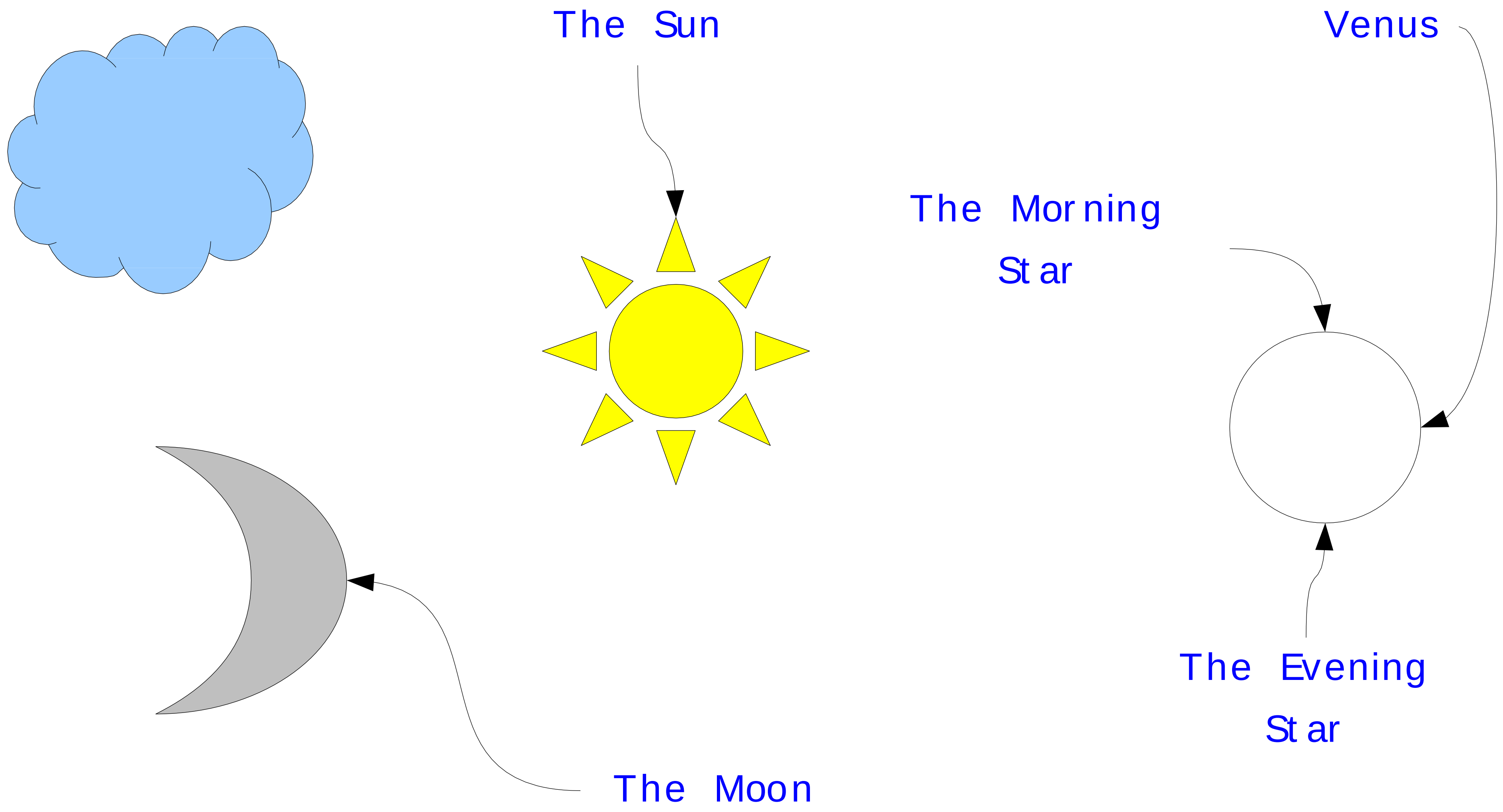
$$p \land q \rightarrow \neg r \lor \neg s$$



Propositional Logic

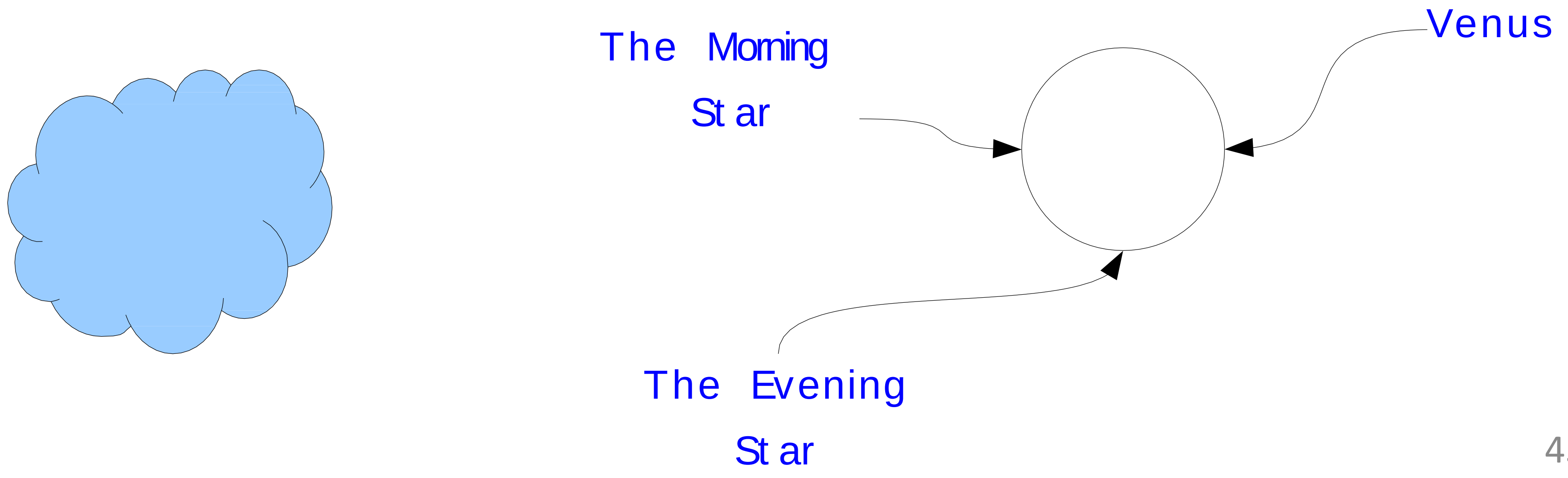
- In propositional logic, each variable represents a ***proposition***, which is either true or false.
- We can directly apply connectives to propositions:
  - $p \rightarrow q$
  - $\neg p \land q$
- The truth of a statement can be determined by plugging in the truth values for the input propositions and computing the result.
- We can see all possible truth values for a statement by checking all possible truth assignments to its variables.

The Universe of First-Order Logic



First-Order Logic

- In first-order logic, each variable refers to some object in a set called the ***domain of discourse***.
- Some objects may have multiple names.
- Some objects may have no name at all



Propositional vs. First-Order Logic

- Because propositional variables are either true or false, we can directly apply connectives to them.

  $p \rightarrow q$ $\neg p \leftrightarrow q \wedge r$

- Because first-order variables refer to arbitrary objects, it does not make sense to apply connectives to them.

  $Venus \rightarrow Sun$ $137 \leftrightarrow \neg 42$

Sentence 단위 vs Object 단위

Reasoning about Objects

- To reason about objects (objec의 TF를 알려면), first-order logic uses ***p redicates***.

  Examples: *ExtremelyCute*(*Quokka*)

  DeadlockEachOther(*House*, *Senate*)

- Predicates can take any number of arguments, but each predicate has a fixed number of arguments (called its ***arity***).
- The arity and meaning of each predicate are typically specified in advance.
- Applying a predicate to arguments produces a proposition, which is either true or false.

First-Order Sentences

- Sentences in first-order logic can be constructed from predicates applied to objects:

$LikesToEat(V, M) \wedge Near(V, M) \rightarrow WillEat(V, M)$

$Cute(t) \rightarrow Dikdik(t) \vee Kitty(t) \vee Puppy(t)$

$\boxed{x < 8 \rightarrow x < 137}$

The notation $x < 8$ is just a shorthand for something like $\mathbf{LessThan}(x, 8)$. Binary predicates in math are often written like this, but symbols like $<$ are not a part of first-order logic.

Equality (=)

- First-order logic is equipped with a special predicate **=** that says whether two objects are equal to one another.
- Equality is a part of first-order logic, just as $\rightarrow$ and $\neg$ are.
- Examples:
  - $MorningStar = EveningStar$
  - $TomMarvoloRiddle = LordVoldemort$
- Equality can only be applied to ***objects***; to see if ***propositions*** are equal, use $\longleftrightarrow$

For notational simplicity, define $\neq$ as

$$x \neq y \equiv \neg(x = y)$$

Functions

$(x < 8 \wedge y < 8) \rightarrow (x + y < 16)$

How is this allowed?

- First-order logic allows ***functions*** that return objects associated with other objects.
- Examples: $x + y$

  LengthOf(*path*)

  MedianOf(x, y, z)

- As with predicates, functions can take in any number of arguments, but each function has a fixed arity.
  - As with predicates, the arity and interpretation of functions are specified in advance.

- Functions evaluate to ***objects***, not ***propositions***.
- There is no syntactic way to distinguish functions and predicates; you'll have to look at how they're used.

Quantifiers

- The biggest change from propositional logic to first-ord er logic is the use of ***quantifiers***.
- A ***quantifier*** is a statement that expresses that some pro perty is true for some or all choices that could be made.
- U seful for statements like “ for every action, there is a n equal and opposite reaction.”
- H ow would we translate the statement into first order logic?

“ For any natural number n, n is even if and only if n^2 is even”

“ S ome muggles are intelligent”

The Universal Quantifier

- A statement of the form $\forall x.\ \psi$ asserts that for ***every*** choice of x in our domain, ψ is true.

"For any natural number n, n is even iff n^2 is even"

$$\forall n.\ (n \in \mathbb{N} \to (Even(n) \leftrightarrow Even(n^2)))$$

$\forall$ is the ***universal quantifier*** and says "for any choice of n, the following is true."

- Examples:
  - $\forall v.\ (Puppy(v) \to Cute(v))$
  - $\forall n.\ (n \in \mathbb{N} \to (Even(n) \leftrightarrow \neg Odd(n)))$
  - $Tallest(SK) \to \forall x.\ (SK \neq x \to ShorterThan(x, SK))$

The Existential Quantifier

- A statement of the form $\exists x.\ \psi$ asserts that for ***some*** choice of x in our domain, ψ is true.

"Some muggles are intelligent."

$$\exists m.\ (Muggle(m) \land Intelligent(m))$$

∃ is the ***existential quantifier*** and says "for some choice of m, the following is true."

- Examples:
  - $\exists x.\ (Even(x) \land Prime(x))$
  - $\exists x.\ (TallerThan(x, me) \land LighterThan(x, me))$
  - $(\exists x.\ Appreciates(x, me)) \to Happy(me)$

Operator Precedence (Again)

- When writing out a formula in first-order logic, the quantifiers $\forall$ and $\exists$ have precedence just below $\neg$

- Thus

$$\forall x.\ P(x) \lor R(x) \to Q(x)$$

is interpreted as the (malformed) statement

$((\forall \boldsymbol{x}.\ P(\boldsymbol{x})) \lor R(\boldsymbol{x})) \to Q(\boldsymbol{x})$

rather than the (intended, valid) statement

$\forall x.\ (P(\boldsymbol{x}) \lor R(\boldsymbol{x}) \to Q(\boldsymbol{x}))$

Translating Into FOL

- First-order logic is an excellent tool for manipulating definitions and theorems to learn more about them.

- Applications:
  - Determining the negation of a complex statement.
  - Figuring out the contrapositive of a tricky implication.

- ***Translating statements into first-order logic is a lot more difficult than it looks.***
- There are a lot of nuances that come up when translating into first-order logic.
- We'll cover examples of both good and bad translations into logic so that you can learn what to watch for.
- We'll also show lots of examples of translations so that you can see the process that goes into it.

Using the predicates

- *Puppy*(*p*), which states that *p* is a puppy, and
- *Cute*(*x*), which states that *x* is cute,

write a sentence in first-order logic that means
"all puppies are cute."

An Incorrect Translation: $\forall x.\ (Puppy(x) \land Cute(x))$

This should work for ***any*** choice of x, including things that aren't puppies.
This logical statement can be made false regardless of whether all puppies are cute. It's therefore not a faithful translation.

A Better Translation

All puppies are cute!

$$\forall x.\ (Puppy(x) \to Cute(x))$$

This should work for ***any*** choice of x, including things that aren't puppies.

"All P's are Q's" translates as $\forall x. (P(x) \to Q(x))$

Useful Intuition:

Universally-quantified statements are true unless there's a counterexample.

$$\forall x. (P(x) \to Q(x))$$

If x is a counterexample, it *must* have property P but not have property Q.

Using the predicates

- *Blobfish*(b), which states that b is a blobfish, and
- *Cute*(x), which states that x is cute,

write a sentence in first-order logic that means "some blobfish is cute."

An Incorrect Translation $\exists x.\ (Blobfish(x) \to Cute(x))$

Although the English statement is false, this logical statement is true. It's therefore not a correct translation.

What happens if

1. The English statement is false, but
2. x refers to a puppy?

A correct Translation

Some blobfish is cute.

$\exists x.\ (Blobfish(x) \land Cute(x))$

"Some ***P*** is a ***Q***" translates as $\exists x.\ (P(x) \land Q(x))$

Useful Intuition:

Existentially-quantified statements are false unless there's a positive example.

$$\exists x.\ (P(x) \land Q(x))$$

If *x* is an example, it *must* have property *P* on top of property *Q*.

Good Pairings

- The $\forall$ quantifier *usually* is paired with $\rightarrow$.
- The $\exists$ quantifier *usually* is paired with $\land$.
- In the case of $\forall$, the $\rightarrow$ connective prevents the statement from being *false* when speaking about some object you don't care about.
- In the case of $\exists$, the $\land$ connective prevents the statement from being *true* when speaking about some object you don't care about.

Using the predicates

- *Tall*(*t*), which states that *t* is tall;
- *Tree*(*t*), which states that *t* is a tree; and
- *Sequoia*(*s*), which states that *s* is a sequoia,

write a sentence in first-order logic that means

“there's a tall tree that's a sequoia.”

Checking a Translation

There's a tall tree that's a sequoia.

$\exists t. (Tree(t) \land (Tall(t) \to Sequoia(t)))$

What if we pick t to be a short tree?

This statement can be true even if no tall sequoias exist.

Checking a Translation

There's a tall tree that's a sequoia.

$\exists t.\ (Tree(t) \land Tall(t) \land Sequoia(t))$

Do you see why this statement doesn't have this problem?

The Universal Quantifier

- A statement of the form **$\forall x.\ \psi$** asserts that for ***every*** choice of *x*, the statement ψ is true.
- Examples:
  - $\forall v.\ (Puppy(v) \to Cute(v))$
  - $\forall n.\ (n \in \mathbb{N} \to (Even(n) \leftrightarrow \neg Odd(n)))$
  - $Tallest(SK) \to \forall x.\ (SK \neq x \to ShorterThan(x, SK))$
- Note the use of the → connective.

The Existential Quantifier

- A statement of the form **$\exists x.\ \psi$** asserts that for ***some*** choice of x, the statement ψ is true.
- Examples:
  - $\exists x.\ (Even(x) \land Prime(x))$
  - $\exists x.\ (TallerThan(x, me) \land LighterThan(x, me))$
  - $(\exists x.\ Appreciates(x, me)) \to Happy(me)$
- Note the use of the $\land$ connective.

Useful Intuition: Universally-quantified statements are true unless there's a counterexample. $\forall x.\ (P(x) \to Q(x))$

If x is a counterexample, it *must* have property P but not have property Q.

Useful Intuition: Existentially-quantified statements are false unless there's a positive example. $\exists x.\ (P(x) \land Q(x))$

If x is an example, it *must* have property P on top of property Q.

Good Pairings

- The ∀ quantifier *usually* is paired with →
- The ∃ quantifier *usually* is paired with ∧.
- In the case of ∀, the → connective prevents the sta
  tement from being *false* when speaking about so
  me object you don't care about.
- In the case of ∃, the ∧ connective prevents the sta
  tement from being *true* when speaking about som
  e object you don't care about.

Using the predicates

- $Person(p)$, which states that p is a person, and
- $Loves(x, y)$, which states that x loves y,

write a sentence in first-order logic that means "everybody loves someone else."

$$
\begin{aligned}
&\forall p.\ (Person(p) \rightarrow \\
&\quad \exists q.\ (Person(q) \land p \neq q \land \\
&\qquad Loves(p, q) \\
&\quad) \\
&)
\end{aligned}
$$

Using the predicates

- *Person*(*p*), which states that *p* is a person, and
- *Loves*(*x*, *y*), which states that *x* loves *y*,

write a sentence in first-order logic that means
"there is someone that everyone else loves."

$$
\begin{aligned}
&\exists p.\ (Person(p)\ \land \\
&\quad \forall q.\ (Person(q)\ \land q \neq p \to \\
&\qquad Loves(q, p) \\
&\quad) \\
&)
\end{aligned}
$$

Combining Quantifiers

- Most interesting statements in first-order logic require a combination of quantifiers
- Example: "Everyone loves someone else."

$$\forall p.\ (Person(p) \rightarrow \exists q.\ (Person(q) \land p \neq q \land Loves(p, q)))$$

For every person,

there is some person

who isn't them

that they love.

Combining Quantifiers

- Most interesting statements in first-order logic require a combination of quantifiers.
- Example: "There is someone everyone else loves."

$$\underbrace{\exists p. (Person(p)}_{\text{There is some person}} \land \underbrace{\forall q. (Person(q)}_{\text{who everyone}} \land \underbrace{p \neq q}_{\text{who isn't them}} \to \underbrace{Loves(q, p)}_{\text{loves.}}))$$

For Comparison

$$\forall p.\ (Person(p) \rightarrow \exists q.\ (Person(q) \land p \neq q \land Loves(p, q)))$$

For every person,

there is some person

who isn't them

that they love.

$$\exists p.\ (Person(p) \land \forall q.\ (Person(q) \land p \neq q \rightarrow Loves(q, p)))$$

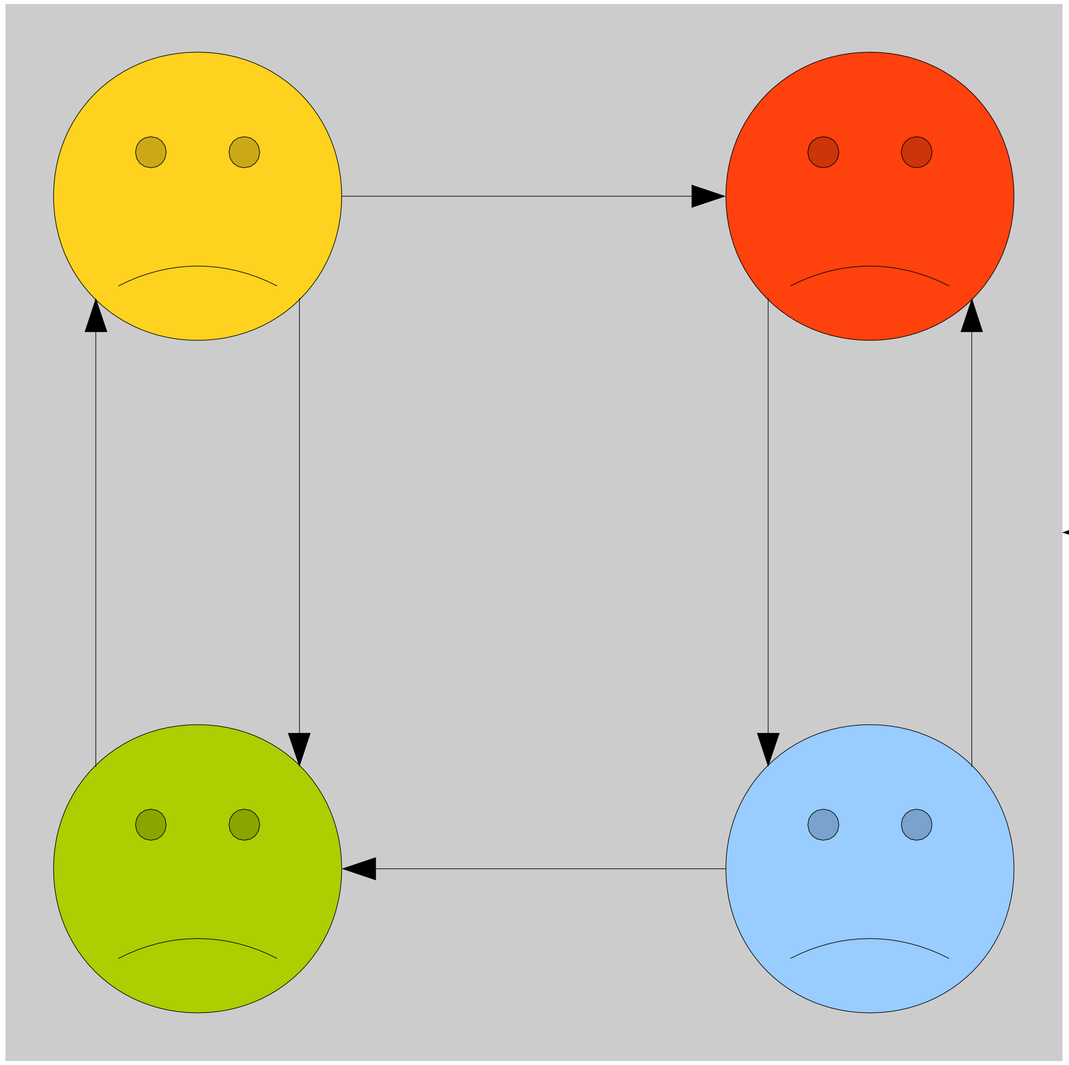
There is some person

who everyone

who isn't them

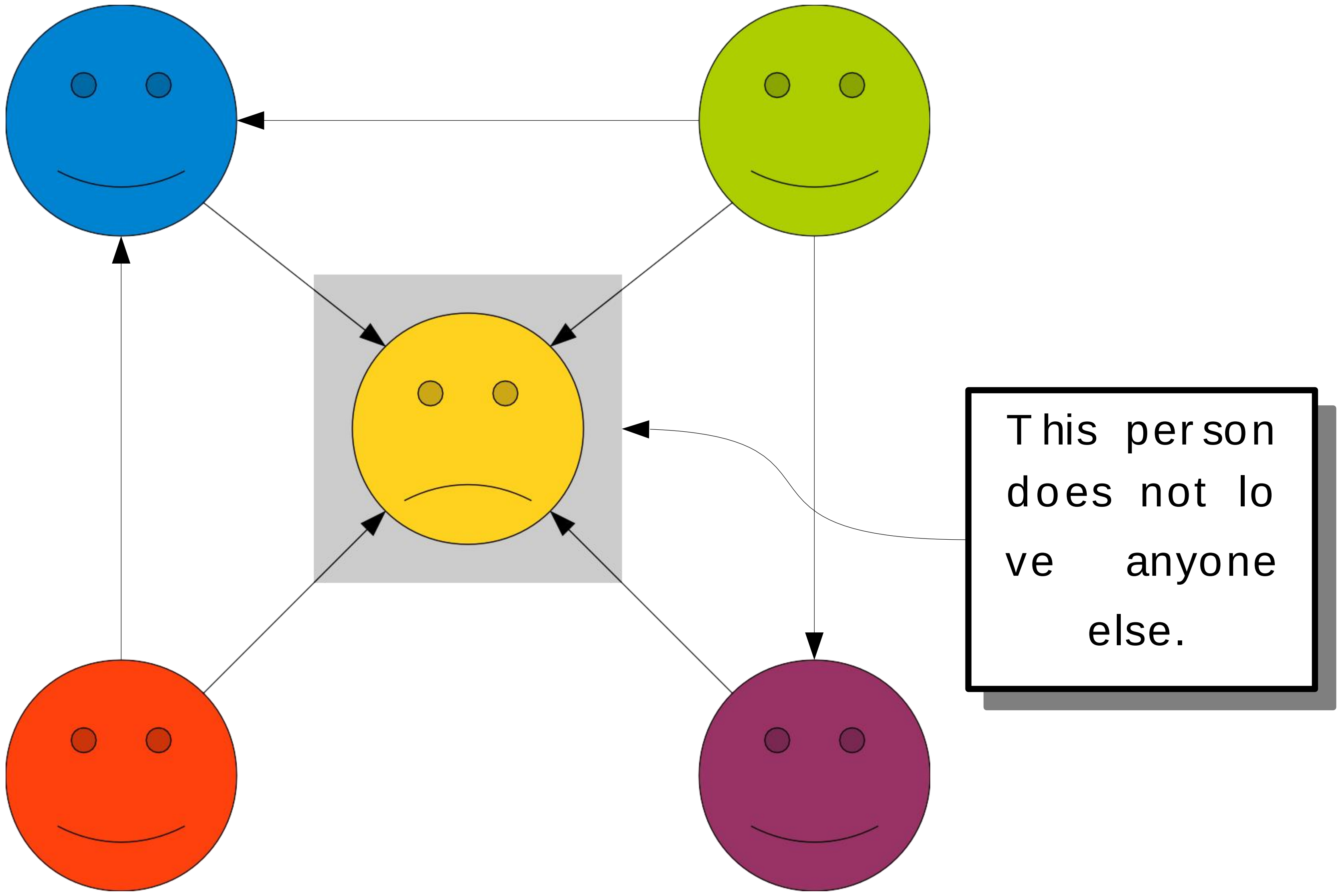
loves.

Everyone Loves Someone Else

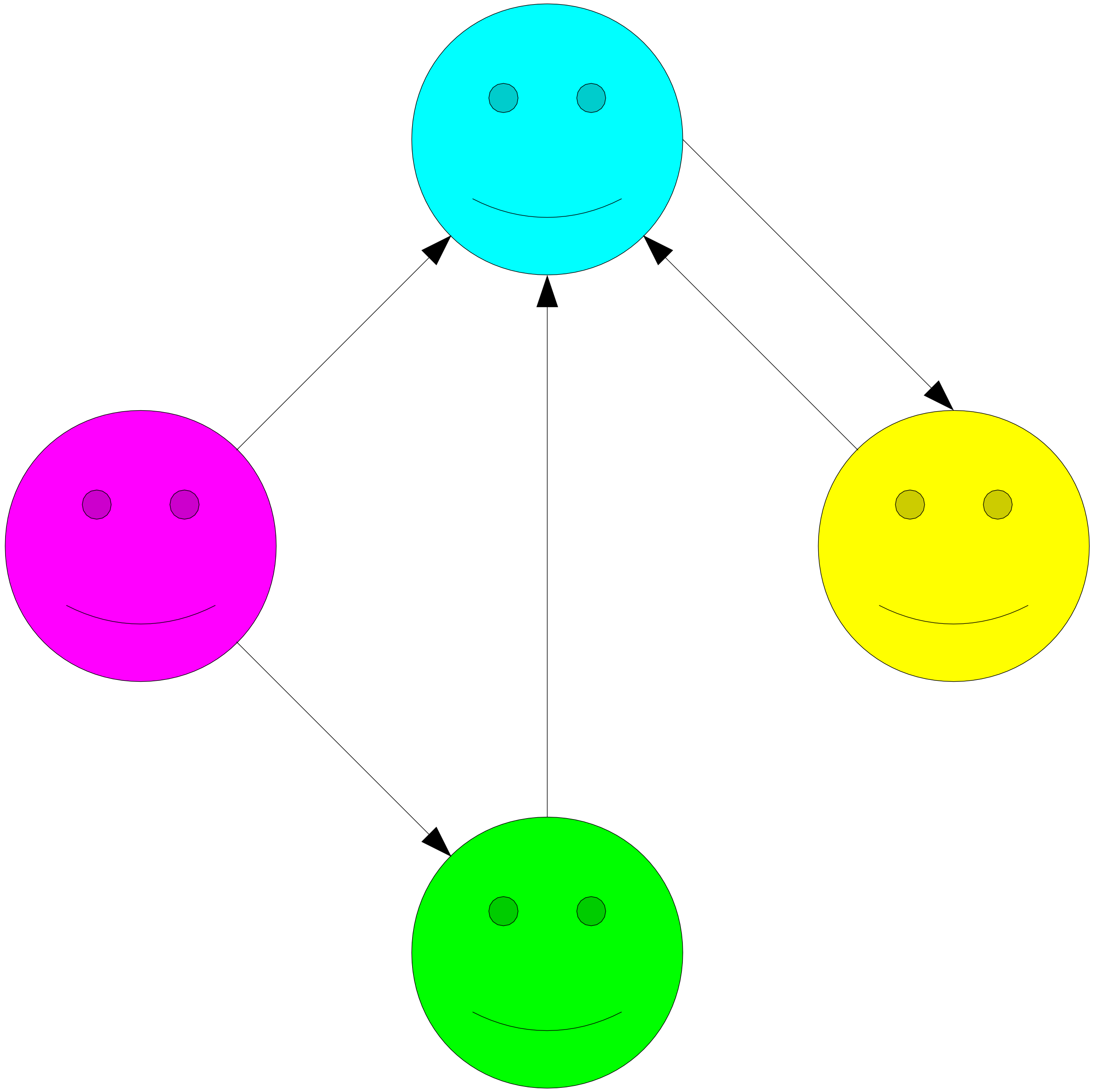


No one here is universally loved.

There is Someone Everyone Else Loves



Everyone Loves Someone Else ***and*** There is Someone Everyone Else Loves



$$\forall p.\ (Person(p) \to \exists q.\ (Person(q) \land p \neq q \land Loves(p, q)))$$

For every person,

there is some person

who isn't them

that they love.

$$\land$$

$$\nexists p.\ (Person(p) \land \forall q.\ (Person(q) \land p \neq q \to Loves(q, p)))$$

There is some person

who everyone

who isn't them

loves.

Quantifier Ordering

- The statement **$\forall x.\ \exists y.\ P(x, y)$** means "for any choice x, there's some y where $P(x, y)$ is true."
  - The choice of y can be different every time and can depend on x.
- The statement **$\exists x.\ \forall y.\ P(x, y)$** means "there is some x where for any choice of y, we get that $P(x, y)$ is true."
  - Since the inner part has to work for any choice of y, this places a lot of constraints on what x can be.

Order matters when mixing existential and universal quantifiers!

Using the predicates

- *Set*(*S*), which states that *S* is a set, and
- $x \in y$, which states that *x* is an element of *y*,

write a sentence in first-order logic that means “the empty set exists.”

First-order logic doesn't have set operators or symbols “built in.” If we only have the predicates given above, how might we describe this?

$$\exists S. \ (Set(S) \land \forall x. \ \neg(x \in S))$$

Using the predicates

- *Tournament*(T), which states that T is a tournament;
- $p \in T$, which states that p is a player in tournament T; an
- *Beat*(p_1, p_2), which states that p_1 beat p_2,

write a sentence in first-order logic that means
"every tournament has a tournament winner."

$$
\begin{array}{l}
\forall T.\ (\mathit{Tournament}(T) \to \\
\quad \exists w.\ (w \in T \land \\
\quad\quad \forall p.\ (p \in T \land p \neq w \to \\
\quad\quad\quad \mathit{Beat}(w, p) \lor \exists q.\ (q \in T \land \mathit{Beat}(w, q) \land \mathit{Beat}(q, p) \\
\quad\quad\quad) \\
\quad\quad) \\
\quad) \\
)
\end{array}
$$

Negating Statements and Negating Quantifiers

- How to negate propositional constructs?
- How do we negate quantifiers?

An Extremely Important Table

	When is this true?	When is this false?
$\forall x.\ P(x)$	For any choice of x, $P(x)$	$\exists x.\ \neg P(x)$
$\exists x.\ P(x)$	For some choice of x, $P(x)$	$\forall x.\ \neg P(x)$
$\forall x.\ \neg P(x)$	For any choice of x, $\neg P(x)$	$\exists x.\ P(x)$
$\exists x.\ \neg P(x)$	For some choice of x, $\neg P(x)$	$\forall x.\ P(x)$

Negating First-Order Statements

- Use the equivalences to negate quantifiers

$$\neg \forall x.\ \varphi \quad \equiv \quad \exists x.\ \neg \varphi$$
$$\neg \exists x.\ \varphi \quad \equiv \quad \forall x.\ \neg \varphi$$

- Mechanically:
  - Push the negation across the quantifier.
  - Change the quantifier from $\forall$ to $\exists$ or vice-versa.
- Use techniques from propositional logic to negate connectives.

Taking a Negation

$$\forall x.\ \exists y.\ Loves(x, y)$$

("Everyone loves someone.")

$$\neg\forall x.\ \exists y.\ Loves(x, y)$$
$$\exists x.\ \neg\exists y.\ Loves(x, y)$$
$$\exists x.\ \forall y.\ \neg Loves(x, y)$$

("There's someone who doesn't love anyone.")

Two Useful Equivalences

- The following equivalences are useful when negating statements in first-order logic:

$$\neg(p \land q) \equiv p \to \neg q$$
$$\neg(p \to q) \equiv p \land \neg q$$

- These identities are useful when negating statements involving quantifiers.
  - $\land$ is used in existentially-quantified statements.
  - $\to$ is used in universally-quantified statements.
- When pushing negations across quantifiers, we *str ongly recommend* using the above equivalences to keep $\to$ with $\forall$ and $\land$ with $\exists$.

Negating Quantifiers

- What is the negation of the following statement, which says “there is a cute puppy”?

$$\exists x.\ (\mathit{Puppy}(x) \land \mathit{Cute}(x))$$

- We can obtain it as follows:

$$\neg\exists x.\ (\mathit{Puppy}(x) \land \mathit{Cute}(x))$$

$$\forall x.\ \neg(\mathit{Puppy}(x) \land \mathit{Cute}(x))$$

$$\forall x.\ (\mathit{Puppy}(x) \to \neg\mathit{Cute}(x))$$

- This says “every puppy is not cute.”
- Do you see why this is the negation of the original statement from both an intuitive and formal persp ective?

$$\exists S.\ (Set(S) \land \forall x.\ \neg(x \in S))$$

("There is a set that doesn't contain anything")

$$\neg\exists S.\ (Set(S) \land \forall x.\ \neg(x \in S))$$
$$\forall S.\ \neg(Set(S) \land \forall x.\ \neg(x \in S))$$
$$\forall S.\ (Set(S) \to \neg\forall x.\ \neg(x \in S))$$
$$\forall S.\ (Set(S) \to \exists x.\ \neg\neg(x \in S))$$
$$\forall S.\ (Set(S) \to \exists x.\ x \in S)$$

("Every set contains at least one element")

These two statements are *not* negations of one another. Can you explain why?

$$\exists S.\ (Set(S) \land \forall x.\ \neg(x \in S))$$

("There is a set that doesn't contain anything")

$$\forall S.\ (Set(S) \land \exists x.\ (x \in S))$$

("Everything is a set that contains something")

Remember: $\forall$ usually goes with $\to$, not $\land$

Quantifying Over Sets

The notation

$$\forall x \in S.\ P(x)$$

means "for any element x of set S, P(x) holds."

This is not technically a part of first-order logic; it is a shorthand for

$$\forall x.\ (x \in S \to P(x))$$

How might we encode this concept?

$$\exists x \in S.\ P(x)$$

Answer: $\exists x.\ (x \in S \land P(x))$.

Note the use of $\land$ instead of $\to$ here.

Quantifying Over Sets

- The syntax

$$\forall x \in S.\ \varphi$$

$$\exists x \in S.\ \varphi$$

- is allowed for quantifying over sets.

- Please don't do things like this:

$$\forall x \text{ with } P(x).\ Q(x)$$

$$\forall y \text{ such that } P(y) \land Q(y).\ R(y).$$

참고자료

First-order logic

- formalizes fundamental mathematical concepts
- is expressive (Turing-complete)
- has a rich structure of decidable fragments
- has a rich model and proof theory

- First-order logic is also called
  - Predicate logic
  - First-order Predicate Calculus

The Undecidability of First Order Logic

참고자료

A first order logic is given by a set of function symbols and a set of predicate symbols. Each function or predicate symbol comes with an arity, which is natural number. Function symbols of arity 0 are known as constant symbols. Now *terms* are recursively defined by

- variables are terms, and
- if t_i are terms for i=1,...,n and f is an n-ary function symbol, then $f(t_1,...,t_n)$ is a term.

Formulas are recursively defined by

- if t_i are terms for i=1,...,n and p is an n-ary predicate symbol, then $p(t_1,...,t_n)$ is a formula.
- if P and Q are formulas, then $_P$ is a formula, P Q is a formula, P Q is a formula, P Q is a formula, and P <=> Q is a formula.
- if P is a formula and x a variable, then $_x$: P and x: P are formulas.

The completeness theorem for first order logic says that a formula is provable from the laws of first order logic (not given here) if and only if it is true in under all possible interpretations, i.e. regardless of the meaning of the function and predicate symbols.

Theorem: It is undecidable whether a first order logic formula is provable (or true under all possible interpretations).