Computing Fundamentals 1
Lecture 5
Predicate Calculus

Lecturer: Patrick Browne
http://www.comp.dit.ie/pbrowne/
Room K308, Lab A115
Based on Chapter 9.
A Logical approach to Discrete Math
By David Gries and Fred B. Schneider
Different Logics in CafeOBJ

- We use CafeOBJ for propositional logic using BOOL module.
- We can use CafeOBJ for predicate logic using FOPL-CLAUSE module.
- We use CafeOBJ as a functional programming language.
The predicate calculus is an extension of the propositional logic that allows the use of variables of types other than Boolean. This extension leads to a logic with enhanced expressive and deductive power.
A predicate calculus formula is a Boolean expression in which some Boolean variables may have been replace by:

1. **Predicates**, which are applications of Boolean functions whose arguments may be types other than Boolean \{true, false\}, e.g. `equal(x:Int, y:Int)`, `max(x:Int, y:Int)`.

2. Universal and existential (\(\forall, \exists\)) quantified variables.
Predicate Calculus

- For all integers, the following is not generally true. But it is true when all variables = 1/3.

\[ a + (b \times c) = (a + b) \times (a + c). \]

- RAT> red \((1/3 + (1/3 \times 1/3)) == (1/3 + 1/3) \times (1/3 + 1/3)\). Gives (true):Bool
- RAT> red \((1 + (1 \times 1)) == (1 + 1) \times (1 + 1)\). Gives false

- The universally quantified version is false.

\((\forall a, b, c \cdot (a + (b \times c) = (a + b) \times (a + c)))\)  
\((\exists a, b, c \cdot (a + (b \times c) = (a + b) \times (a + c)))\)
Predicate Calculus (PC)

• An example formula in the predicate calculus:

\[(x < y) \land (x = z) \Rightarrow q(x, z+x)\]

Where:

- \(x < y\), \(x = z\), \(q(x, z+x)\) are predicates. What about type?
- \(x, y, z,\) and \((z+x)\) are terms.

• Pure PC includes all of the axioms of propositional calculus together with axioms for quantifications \((\forall x \mid R: P)\) and \((\exists x \mid R: P)\)

• The PC inference rules are Substitution, Transitivity, Leibniz, and Leibniz with quantification. CafeOBJ has a module corresponding to the predicate calculus (FOPL-CLAUSE).
Predicate Calculus (PC)

- In pure PC the function symbols are un-interpreted (except for =). The logic provides no specific rules for manipulation the function symbols. This means that we can develop general rules for manipulating function symbols regardless of their meaning (interpretation). The pure predicate calculus is sound in all domains that may be of interest.
Predicate Calculus and relations and functions.

- We must distinguish between relations and function.

- **Function**: for a given element in the domain (input) a function can have one element in the range (output) e.g. $\text{motherOf}(John) = \text{Mary}$ or $<\text{Mother}, \text{John}>$.

- **Relation**: for a given element in the range a relation can have several elements in the range e.g. $\text{parentOf}(John) = \{\text{Liam, Mary}\}$ or $<\text{Mary, John}>$, $<\text{Liam, John}>$. 
We get a theory by adding axioms that give meaning to some of the un-interpreted function symbols.

For example, the theory of integers consists of pure predicate calculus together with axioms for manipulating the operators +, -, <, ≤ (i.e. functions). The axioms say that • (multiply) is symmetric and associative and has the zero 0. See CafeOBJ INT module.

Similarly the theory of sets provides axioms for manipulating set expressions (union etc).

The core of the above theories is PC. The theory provides meaning the logic provides inference.
Zero

- **Z is a zero** of some binary operation \( \circ \) if
  \[
  (\forall x \mid : x \circ Z = Z \circ x = Z)
  \]

- **Z is a left zero** if
  \[
  (\forall x \mid : Z \circ x = Z)
  \]

- **Z is a right zero** if
  \[
  (\forall x \mid : x \circ Z = Z)
  \]

- The term zero comes from the fact that 0 is the zero of multiplication. Think of having the zero property rather than the value 0.
Universal Quantification

• Conjunction ($\wedge$) is symmetric and associative and has identity $\text{true}$.

• $(9.1) \ (\forall x \ | \ R : P)$

• For all $x$ such that $R$ holds, $P$ holds.

• Predicate $R$ is called the range, predicate $P$ the body (or the term), and $x$ the dummy variable.

• $\wedge$ is idempotent ($p \wedge p = p$, $1 \times 1 = 1$, $0 \times 0 = 0$) and satisfies range-split axiom (8.18).

\[(\forall x \ | \ R \lor S : P) = (\forall x \ | \ R : P) \ast (\forall x \ | \ S : P)\]
Existential Quantification

• Disjunction ($\lor$) is symmetric and associative and has identity $\text{false}$.

• $(\forall x \mid R : P)$ written as $(\exists x \mid R : P)$

• There exists an $x$ in the range $R$ such that $P$ holds.

• Predicate $R$ is called the range, predicate $P$ the body (or the term), and $x$ the dummy variable.

• $\lor$ is idempotent ($p \lor p = p$) and satisfies range-split axiom(8.18).
From Lecture 3 recall DeMorgan

• 3.47(a)(b) DeMorgan:

\[ \neg(p \land q) \equiv \neg p \lor \neg q \]

\[ \neg(p \lor q) \equiv \neg p \land \neg q \]
Generalized DeMorgan

\[(9.18)\ (a)\]

\[\neg (\exists x \mid R : \neg P) \equiv (\forall x \mid R : P)\]

\[(9.18)\ (b)\]

\[\neg (\exists x \mid R : P) \equiv (\forall x \mid R : \neg P)\]

\[(9.18)\ (c)\]

\[(\exists x \mid R : \neg P) \equiv \neg (\forall x \mid R : P)\]
English to Predicate Logic

“some integer between 80 and \( n \) is a multiple of \( x \)”.

\[
(\exists i : \mathbb{Z} \mid 80 \leq i \leq n : \text{mult}(n, x))
\]

where \( \text{mult}(n, x) \) denotes “\( n \) is a multiple of \( x \)”.

\[
(\exists m : \mathbb{Z} \mid n = m \cdot x)
\]

• Expressions can be transformed or rewritten using the axioms of a particular theory. For example, using the Theory of Integers(\( \mathbb{Z} \)) the following can be proved:

\[
\text{even}(x) \land (\exists m : \mathbb{Z} \mid n = m \cdot x) \Rightarrow (\exists m : \mathbb{Z} \mid n = m \cdot x / 2)
\]
English to Predicate Logic

• In would be difficult to reason about such statements using only the propositional calculus. With the predicate calculus we can reason over collections of objects.
• Formalizing English or mathematical expressions in predicate logic is useful:
  • Firstly, it enforces precision and exposes ambiguities. Does “between 80 and n” include 80 and n? This question must be answered during formalization.
  • Secondly, once formalized we can use inference rules to reason formally about the objects under consideration.
# English to Predicate Logic

<table>
<thead>
<tr>
<th>English</th>
<th>Quantifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>every</td>
<td>$\forall$</td>
</tr>
<tr>
<td>all</td>
<td>$\forall$</td>
</tr>
<tr>
<td>for all</td>
<td>$\forall$</td>
</tr>
<tr>
<td>for each</td>
<td>$\forall$</td>
</tr>
<tr>
<td>for any</td>
<td>$\forall$</td>
</tr>
<tr>
<td>English</td>
<td>Quantifier</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------</td>
</tr>
<tr>
<td>exists</td>
<td>$\exists$</td>
</tr>
<tr>
<td>some</td>
<td>$\exists$</td>
</tr>
<tr>
<td>there is</td>
<td>$\exists$</td>
</tr>
<tr>
<td>at least one</td>
<td>$\exists$</td>
</tr>
<tr>
<td>for some</td>
<td>$\exists$</td>
</tr>
</tbody>
</table>
English to Predicate Logic

• Every chapter has at least three pages:
  \((\forall c \mid c \in \text{Chapter} \mid : \text{size}(c) \geq 3)\)

• There is a chapter with an even number of pages:
  \((\exists c \mid c \in \text{Chapter} \mid : \text{even}(\text{size}(c)))\)

• Where \text{size}(c)\ is a function returning the number of pages in a chapter, and \text{even}(x:\text{Nat})\ is a predicate taking a natural number argument.
English to Predicate Logic

• Everyone is younger than their father
  \[(\forall x) \ person(x) \Rightarrow younger(x, \ father(x))\]
Negation of Quantified expression

• The correct negation of “All integers are even” is “Not all integers are not even”. Note an incorrect negation is “All integers are not even”. Is the ‘all’ being negated?

• The universal (∀) statement “Not all integers are not even” can be rewritten as an existential (∃) statement “Some (at least one) integers are not even”.

Negation of Quantified expression

• Not( all integers are even)
  = < Formalization in predicate calculus >
  \neg (\forall z: \mathbb{Z} \mid : \text{even}.z )
  = < DeMorgan (9.18c) >
  (\exists z: \mathbb{Z} \mid : \neg \text{even}.z )
  = < Return to English >
  Some integer is not even
Negation of Quantified expression

- Not(all people older than 20) = \( \neg (\forall z: \mathbb{N} \mid \text{age} > 20) \)
  = < Generalized DeMorgan (9.18c) >
  \( (\exists z: \mathbb{N} \mid \neg \text{(age} > 20)) \)
  = < Return to English >
  Some people are not older than 20
  or
  There exists a person who is not older than 20.
Negation of Quantified expression

- For any predicates $P$ and $Q$ over Domain $D$.
- The negation of $\forall x \in D, P(x)$ is $\exists x \in D, \neg P(x)$
- The negation of $\exists x \in D, Q(x)$ is $\forall x \in D, \neg Q(x)$
Quantified Equivalences

• Assume $x$ is the only variable in $F$ or $G$.
• $(\exists x) (F) \lor (\exists x) (G) \equiv (\exists x) (F \lor G)$
• $(\forall x) (F) \land (\forall x) (G) \equiv (\forall x) (F \land G)$
• $\neg (\forall x) (F) \equiv (\exists x) (\neg F)$
• $\neg (\exists x) (F) \equiv (\forall x) (\neg F)$

• The last two equivalences are alternative representations of the negations on the previous slide.
Quantified Equivalences

• Not all birds can fly
  \[ \neg (\forall x: \text{bird}(x) \Rightarrow \text{fly}(x)) \]

  Same as
  \[ \exists x : (\text{bird}(x) \land \neg \text{fly}(x)) \]

• All birds cannot fly.
  \[ \forall x: \text{bird}(x) \Rightarrow \neg \text{fly}(x) \]

  Same as
  \[ \neg (\exists x \text{ bird}(x) \land \text{fly}(x)) \]
Predicate logic in CafeOBJ

- set include FOPL-CLAUSE on
module MORTAL{
  [ Elt ]
pred man : Elt
pred mortal : Elt
op Socrates : -> Elt
op Superman : -> Elt
op CaptainKirk : -> Elt
ax \A[X:Elt] man(X) -> mortal(X) .
ax man(Socrates) .
}
-- goal mortal(Socrates) .

Proof using predicate logic in CafeOBJ

• * PROOF \text{mortal}(\text{Socrates})

  • 1:\[\neg(\text{man}(v)) \lor \text{mortal}(v)\]
  • 2:\[\text{man}(\text{Socrates})\]
  • 3:\[\neg(\text{mortal}(\text{Socrates}))\]
  • 7:[\text{hyper}:2,1] \text{mortal}(\text{Socrates})
  • 8:[\text{binary}:7,3]

Substitution: Socrates for v
Proof by Contradiction using resolution

Suppose we have a complex predicate logic statement, and we want to prove that a certain clause is true. For example the statement may set out various aspects of property law, and the clause express be whether a certain transaction is legal or not.

In automated proof:
1. The system will convert the complex statement(s) to an internal form suitable for processing
2. The system assumes the NEGATION of the original clause
3. The applies resolution (includes substitution and cancellation) several times,
4. If and when a contradiction (A\land\lnot A) is found, the system has proved the original clause.
Suppose we have:

\[(A \lor B \lor D) \land (B \lor \neg D \lor G) \land (A \lor \neg G \lor B)\]

And want to prove \((A \lor B)\) is true.

Start with assuming the negation of what we want to prove.

1) \(\neg A \land \neg B\)

Write down what we already know to be true

2) \((A \lor B \lor D) \land (B \lor \neg D \lor G) \land (A \lor \neg G \lor B)\)

Resolution twice on the 3rd clause of 2, using each part of 1, gives:

3) \(\neg G\)

Resolution using 3 and the second clause of 2 gives:

4) \((B \lor \neg D)\)

Resolution using 4 and the first clause of 2 gives:

5) \((A \lor B)\)

6) This is the negation of 1), so we have a contradiction.
Combining Quantifiers

• What is the precedence or associativity used?
• Is $\text{forall}(x), \text{forall}(y)$ equivalent to $\text{forall}(y), \text{forall}(x)$?
• Assuming standard notation, they are read left to right; two universal quantifiers following one another commute (just like multiplication $a \times b = b \times a$). So they are regarded as being the same i.e. in this order does not matter.
Combining Quantifiers

• Likewise, if we have two existential quantifiers, they commute, because
  • \( \exists x, \exists y, p(x,y) \)
  • is true if and only if
  • \( \exists y, \exists x, p(x,y) \)
  • is true.
Mixing Quantifiers

• But if we have mixed universal and existential quantifiers, we have to be careful. If you write
\( \forall (x), \exists (y), p(x, y) \)
• that means that for all \( x \), there exists a \( y \), which may depend on \( x \), for which \( p(x, y) \) is true.
• But if we write
\( \exists (y), \forall (x), p(x, y) \)
• we are saying that there is a \( y \) which works for any given \( x \) (including \( y=x \)), that makes \( p(x, y) \) true. In this case order does matter.
**Mixing Quantifiers**

Let the predicate $g(x, y)$ be defined as:
\[ g(x, y) = x < y. \]

Write each of the following propositions in words and state whether they are true or false.

1) $\forall x : \mathbb{N}, \exists y : \mathbb{N} \ g(x, y)$
2) $\exists x : \mathbb{N}, \forall y : \mathbb{N} \ g(x, y)$

Solution on next slides.
Mixing Quantifiers

1) $\forall x : \mathbb{N}, \exists y : \mathbb{N} \ g(x, y)$

- For all natural numbers $x$ there exists a $y$ such that $x < y$. The statement is true if we make $y = x + 1$. To prove this statement we only require the existence of one $y$ for a given $x$ that makes the statement true.
2) \( \exists x : \mathbb{N}, \forall y : \mathbb{N}, \ g(x, y) \)

- There exists a natural number \( x \) such that for all \( y \), \( x < y \). The statement is false when \( x > y \) or \( x = y \). To **disprove** that statement, we only had to find one \( y \) for a given \( x \) that makes the statement false.
# Mixing Quantifiers

<table>
<thead>
<tr>
<th>Statement</th>
<th>True When</th>
<th>False When</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall x \forall y P(x, y)$</td>
<td>$P(x, y)$ is true for every pair $x, y$.</td>
<td>There is at least one pair, $x, y$ for which $P(x, y)$ is false.</td>
</tr>
<tr>
<td>$\forall x \exists y P(x, y)$</td>
<td>For every $x$, there is a $y$ for which $P(x, y)$ is true.</td>
<td>There is an $x$ for which $P(x, y)$ is false for every $y$.</td>
</tr>
<tr>
<td>$\exists x \forall y P(x, y)$</td>
<td>There is an $x$ for which $P(x, y)$ is true for every $y$.</td>
<td>For every $x$, there is a $y$ for which $P(x, y)$ is false.</td>
</tr>
<tr>
<td>$\exists x \exists y P(x, y)$</td>
<td>There is at least one pair $x, y$ for which $P(x, y)$ is true.</td>
<td>$P(x, y)$ is false for every pair $x, y$.</td>
</tr>
</tbody>
</table>

$\forall x \exists y \text{ Older}(x, y)$

$\exists x \forall y \text{ Older}(x, y)$
Mixing Quantifiers

• The predicate \( \text{Older}(x, y) \) is true if person \( x \) is older than person \( y \). Translate the following into English. What are their truth values?

\[
\begin{align*}
\forall x \exists y \text{ Older}(x, y) \\
\exists x \forall y \text{ Older}(x, y)
\end{align*}
\]

• What are the truth values of the following?

\[
\begin{align*}
\forall x \exists y \text{ SameAge}(x, y) \\
\exists x \forall y \text{ SameAge}(x, y)
\end{align*}
\]
Skolem Functions (Skolemisation)

- A function can be used as an argument to a predicate e.g. the functions mother, father and s.
- `likes(mother(fred), father(bill))`
- `greater(s(s(x)), s(x))`
- It is possible to re-express existential quantifiers (\(\exists\)) with constants or functions e.g.
  - `\(\forall Y \exists X : likes(Y,X)\)`
  - In CafeOBJ becomes:
    - `likes(Y,skoleml1(Y))`
Logical & Operational meaning of CafeOBJ program.

- Two interpretations of meaning of CafeOBJ program
  - declarative or logical meaning
  - operational meaning as a set of rewrite rules

- The declarative meaning determines logical answers. It is concerned only with the relations and functions that have been defined in the program.
Logical & Operational meaning of CafeOBJ program.

- The operational meaning is concerned with **computing the output**. This means that the order of the equations is significant. Consider the query `parent(X,Emily)` using the following equations:

  - eq [o4] : parent(John,Emily) = true .
  - eq [o5] : parent(Beth,Emily) = true .
Logical & Operational meaning of CafeOBJ program.

The declarative meaning tells us that both John and Beth are parents of Emily. Declaratively, the two equations are the same. Operationally the relation `parent(John, Emily)` occurs before `parent(Beth, Emily)`, means that the first answer returned will be John. We can obtain the second answer (Beth) by using the theorem prover or using sophisticated coding techniques.
Logical and operational meaning of a program.

- mod FAMILY { [Person]
- ops Emily Ken Anne Beth Gran Liz John Ellen Pat : -> Person
- op parent : Person Person -> Bool
- op parentOf : Person -> Person

- vars X Y : Person
- -- Beth has two children
- eq [f1] : parent(Beth,Emily) = true .
- eq [f2] : parent(Beth,Ellen) = true .
- -- Emily has two parents
- eq [f3] : parent(John,Emily) = true .}

- open FAMILY
- -- The theorem prover will find that both John and Beth are parents of Emily. There exists an P such that P is parent of Emily. Set max-proofs to 2 to get both cases.
- goal \E[P:Person] parent(P,Emily) .
- -- Ordinary reduction will not find a value for P.
- red parent(P:Person,Emily) .
Mixing Quantifiers

English: Every human has a mother
Predicate Logic: \( \forall x \exists y [\text{human}(x) \Rightarrow \text{mother}(y, x)] \)

In CafeOBJ it is necessary to use the \( \text{mum}(X) \) a Skolem function to name the individual \( Y \) which exists as a function of \( X \).

Predicate Logic: \( \forall x [\text{human}(x) \Rightarrow \text{Mother}(\text{mum}(x), x)] \)

mod FAMILY {  
    [Being]
    op mother : Being  Being  ->  Bool
    op mum : Being  ->  Being
    op human : Being  ->  Bool
    ops Emily Ellen :  ->  Being
    vars X Y   :  Being

    -- Axiom: Every human has a mother
    eq human(X) = mother( mum(X),X) .

    -- Facts
    eq mum(Emily) = Ellen .
    eq mother(Ellen,Emily) = true .
}

open FAMILY
red human(Emily) .  -- true
Mixing Quantifiers

Example

Every human has a mother.

Standard Form

$$\forall x \exists y [\text{Human}(x) \rightarrow \text{Mother}(y,x)]$$

Changing the order of the quantifiers changes the meaning. The sentence

$$\exists y \forall x [\text{Human}(x) \rightarrow \text{Mother}(y,x)]$$

states there is a single individual who is the mother of us all. The clausal form uses a constant symbol to name the individual.

$$\text{Human}(x) \rightarrow \text{Mother}(\text{Eve},x)$$
Predicates in CafeOBJ equations

1. \( \exists X \) planet\((X) \land bright\((X)\)
2. \( \forall X \) planet\((X) \land bright\((X) \Rightarrow small\((X)\)
3. Therefore \( \exists X \) small\((X)\)

```plaintext
mod* PLANET {
[ Elt ]
ops planet small bright : Elt -> Bool
-- Constant in lower-case for THERE-EXISTS
op x : -> Elt
-- Variable in upper-case FOR-ALL
var X : Elt

eq planet(x) = true .
eq bright(x) = true .
ceq small(X) = true if bright(X) and planet(X) .
} -- Prove that x that is small (we must already know x exists)
open PLANET .
red small(x)  .
Can we prove that THERE-EXISTS an X that is small.
```
Predicates in CafeOBJ equations

• $\exists X \ (\text{planet}(X) \land \text{bright}(X))$
• $\forall X (\text{planet}(X) \land \text{bright}(X) \Rightarrow \text{small}(X))$
• Therefore $\exists X \ \text{small}(X)$

Can we prove that THERE-EXISTS an $X$ that is small.

mod* PLANET {
    [ Elt ]
    ops planet small bright : Elt -> Bool
    -- Constant for THERE-EXISTS, unique within module
    op x : -> Elt
    -- Variable FOR-ALL, unique within equation
    var X : Elt
    eq  planet(x) = true .
    eq  bright(x) = true .
    eq  planet(X) = planet(x) .
    eq  bright(X) = bright(x) .
    ceq small(X) = true if bright(X) and planet(X) .
    -- Prove that there exists an $x$ that is small
    open PLANET .
    red small(X) . -- use a variable, CafeOBJ finds a value
Predicates in CafeOBJ FOPL

- $\exists X \ planet(X) \land bright(X)$
- $\forall X \ planet(X) \land bright(X) \Rightarrow small(X)$
- Therefore $\exists X \ small(X)$

```
mod* PLANET-FOL{
pr(FOPL-CLAUSE)
[ Elt ]
ops planet small bright : Elt -> Bool
-- There exists at least one element x in the universe for which planet(X) is true.
ax \A[X:Elt] planet(X) & bright(X) -> small(X) .
ax \E[Y:Elt] planet(Y) & bright(Y) . } -- Must use & over same Y

open PLANET-FOL .
goal \E[A:Elt] small(A) .
option reset
flag(auto, on)
flag(very-verbose,on)
param(max-proofs, 1)
resolve .
```
More than one answer?

• Suppose we have the following two facts:
  – Ken has a child Beth
  – Ken has a child Emily

• Suppose we wish to find all of Ken’s children. If we use CafeOBJ normal equations and reduction we will only get one result from the first equation of:
  – eq childOf(Ken) = Beth .
  – eq childOf(Ken) = Anne .
More than one answer?

- To find all of Ken’s children we can use CafeOBJ rewrite logic (RWL) and special reduction. RWL equations are called rules.
  - `rl childOf(Ken) => Beth .`
  - `rl childOf(Ken) => Anne .`
- The special reduction command is:
  - `red childOf(Ken) =(*,*)=>* x:Person .`
- Giving Beth and Anne.
Nesting Quantifiers

- You can fool all of the people some of the time.
  \((\forall p : P \mid (\exists t : \mathbb{N} \mid \text{fool}(p, t)))\)

- You can fool some of the people all of the time.
  Using logical operations \(\land\) (and), \(\Rightarrow\) (implies).
  \(\exists x \forall t \ (\text{person}(x) \land \text{time}(t)) \Rightarrow \text{fool}(x, t)\)

- What about the possible cases for implication:
  false \(\Rightarrow\) false, and false \(\Rightarrow\) true?

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>P (\Rightarrow) Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
Nesting Quantifiers

"You can fool some of the people all the time." There are two possibilities. The first is:

\[ \text{FORALL } y \text{ EXISTS } x \{ \text{fool} (x, y) \} \]

**Meaning:** For all times \( y \) there exists a person \( x \) such that you can fool that person \( x \) at time \( y \); probably TRUE. Alternatively:

\[ \text{EXISTS } x \text{ FORALL } y \{ \text{fool} (x, y) \} \]

**Meaning:** There exists a person \( x \) such that for all times \( y \) you can fool person \( x \) at time \( y \); probably FALSE.

Let \( P \) be the set of all people, \( T \) be the set of all times measured in, say, seconds from some initial time.
Let \( F_{pt} \) mean \text{you can fool person } p \text{ at time } t \.

1. You can fool some of the people some of the time. SOLUTION \( (\exists p : P \mid (\exists t : T \mid F_{pt})) \).

2. You can fool all of the people some of the time. SOLUTION \( (\forall p : P \mid (\exists t : T \mid F_{pt})) \).

3. You can't fool all the people all the time. SOLUTION \( \neg (\forall p : P \mid (\forall t : T \mid F_{pt})) \)
Nesting Quantifiers

You can fool some of the people some of the time but you cannot fool all of the people all the time.

$$\exists p (\text{isPerson}(p) \land \forall t (\text{isTime}(t) \rightarrow \text{isFooled}(p, t)))$$

$$\land \exists t (\text{isTime}(t) \land \forall p (\text{isPerson}(p) \implies \text{isFooled}(p, t)))$$

$$\land (\neg \forall t \forall p (\text{isTime}(t) \land \text{isPerson}(p))) \rightarrow (\text{isFooled}(p, t))$$

The result is the following set of clauses:

- isPerson(a)
- $\neg$ isTime(y) $\lor$ isFooled(a,y)
- isTime(b)
- $\neg$ isPerson(x) $\lor$ isFooled(x,b)
- isPerson(c)
Quantifier Examples

Let $O(x, y)$ be the predicate (or propositional function) “$x$ is older than $y$”. The domain of discourse consists of three people Garry, who is 30; Ellen who is 20; and Martin who is 35. Write each of the following propositions in words and state whether they are true or false.
Quantifier Examples(*)

Let $O(x,y)$ be the propositional function “$x$ is older than $y$”. The domain of discourse consists of three people Garry, who is 30; Ellen who is 20; and Martin who is 35. Write each of the following propositional functions in words and state whether they are true or false.

- $\forall x \forall y O(x, y)$ Everyone is older than everyone, False
- $\forall x \exists y O(x, y)$ Everyone is older than someone, False
- $\exists x \forall y O(x, y)$ Someone is older than everyone, False
- $\exists x \exists y O(x, y)$ Someone is older than someone else, True
Quantifiers

What is the truth value of each of the following sentences:

1. $(\forall x: \mathbb{Z}) (\forall y: \mathbb{Z}) (x^2 > y)$
2. $(\exists x: \mathbb{Z}) (\forall y: \mathbb{Z}) (x^2 > y)$
3. $(\forall y: \mathbb{Z}) (\exists x: \mathbb{Z}) (x^2 > y)$
4. $(\exists y: \mathbb{Z}) (\forall x: \mathbb{Z}) (x^2 > y)$
5. $(\forall x: \mathbb{Z}) (\exists y: \mathbb{Z}) (x^2 > y)$
6. $(\exists x: \mathbb{Z}) (\exists y: \mathbb{Z}) (x^2 > y)$
What is the truth value of each of the following sentences:

1. \((\forall x : \mathbb{Z}) \ (\forall y : \mathbb{Z}) \ (x^2 > y)\) \hspace{1cm} \text{false}

2. \((\exists x : \mathbb{Z}) \ (\forall y : \mathbb{Z}) \ (x^2 > y)\) \hspace{1cm} \text{false}

3. \((\forall y : \mathbb{Z}) \ (\exists x : \mathbb{Z}) \ (x^2 > y)\) \hspace{1cm} \text{true}

4. \((\exists y : \mathbb{Z}) \ (\forall x : \mathbb{Z}) \ (x^2 > y)\) \hspace{1cm} \text{true}

5. \((\forall x : \mathbb{Z}) \ (\exists y : \mathbb{Z}) \ (x^2 > y)\) \hspace{1cm} \text{true}

6. \((\exists x : \mathbb{Z}) \ (\exists y : \mathbb{Z}) \ (x^2 > y)\) \hspace{1cm} \text{true}
Quantifiers

Why are the following true or false:

(\forall x: \mathbb{Z}) (\forall y: \mathbb{Z}) (x^2 > y)
false (x=1, y=2)
(\exists x: \mathbb{Z}) (\forall y: \mathbb{Z}) (x^2 > y)
false (no x for every y e.g. x=1, y=2)
(\forall y: \mathbb{Z}) (\exists x: \mathbb{Z}) (x^2 > y)
true, x=|y|+1, where || is absolute value
(\exists y: \mathbb{Z}) (\forall x: \mathbb{Z}) (x^2 > y)
true, y = -1
(\forall x: \mathbb{Z}) (\exists y: \mathbb{Z}) (x^2 > y)
true, y = -1
(\exists x: \mathbb{Z}) (\exists y: \mathbb{Z}) (x^2 > y)
true x = 2, y = 1
Formalizing English Example

• The predicate $\text{Attends}(C, S)$ is interpreted to mean child $C$ attend school $S$. Using this predicate write the following English statements as quantified statements in predicate logic:
  • Every child attends school.
  • Possible negations of the above.
    – At least one child does not attend school.
    – It is not the case that all children attend school.
Formalizing English Example

• Every child attends school.
• What does the English sentence mean? Does it mean:
  • $\forall c \forall s \text{ Attends}(c,s)$
  • or
  • $\forall c \exists s \text{ Attends}(c,s)$
• Most likely the latter i.e. Every child attends a school (at a time!).
Formalizing English Example

• Rephrasing the possible negations of the above.
  – It is not the case that all children attend a school.
  – At least one child does not attend a school.

• \( \neg (\forall c \exists s \text{ Attends}(c,s)) \)
• \( \exists c \forall s \neg (\text{Attends}(c,s)) \)
• “There exists a child that does not attend any school”
• What does the following say;
• \( \neg (\exists s \forall c \text{ Attends}(c,s)) \)
Formalizing English Example

• Do these statements mean the same thing?
  – There exists a child that does not attend any school
  – At least one child does not attend a school.
  – It is not the case that all children attend a school.

\[
\text{not}(\forall \forall \exists \text{Attends}(c,s)) \equiv \\
\exists \forall \exists \text{not}(\text{Attends}(c,s))
\]
During the design of a communication system it is often required to formalise parts of the specification. In this context write the following two sentences in predicate logic:

- Messages sent from one process to another process are received in the order in which they were sent.
- All messages are received in the same order by all processes.
Predicate logic example

• First we must decide on the predicates and their arguments.
  • $\text{sent}(m, p, t)$: message $m$ is sent by process $p$ at time $t$.
  • $\text{rec}(m, p, t)$: message $m$ is received by process $p$ at time $t$. 
• Messages sent from one process to another process are received in the order in which they were sent.

$$(\forall m, p, p', m', t, t', t'', t''') \mid:\n\text{sent}(m, p, t) \land \text{sent}(m', p, t') \land (t < t') \Rightarrow
\text{rec}(m, p', t''') \land (\text{rec}(m', p', t''')) \land (t < t''') \land (t' < t''') \land (t''' < t''''))$$
Formalizing English Example

• English statement: “Every senior student took one mathematics class and passed one programming class”.
Formalizing English Example

• First we write predicates:
  • $\text{taken}(s, c)$: Student $s$ completed class $c$
  • $\text{passed}(s, c)$: Student $s$ passed $c$
  • $\text{senior}(s)$: Student $s$ is a senior
  • $\text{math}(c)$: $c$ is a mathematics class
  • $\text{prog}(c)$: $c$ is a programming class
Formalizing English Example

• The translation of “Every senior student took one mathematics class and passed one programming class” is:

\[(\forall s \mid \text{senior}(s) \quad : \quad (\exists c, c' \mid : \text{math}(c) \land \\
\text{taken}(s, c) \land
\text{prog}(c') \land \\
\text{passed}(s, c'))))\]
Formalizing English Example

• What does this mean?:
• \( (\exists c, c' \mid \exists s \mid \text{senior}(s) : \) \\
  \\
  \( \text{math}(c) \land \text{taken}(s, c) \land \text{prog}(c') \land \text{passed}(s, c') ) ) \)

• All senior students took the same math class and passed the same programming class.
Formalizing English Example

- Translate the following sentence into predicate calculus.
- “Every senior student who took a maths class did not take a programming class”.

\( (\exists c, c' \mid : 
(\forall s \mid \text{senior}(s) : 
\text{maths}(c) \land 
\text{taken}(s, c) \land 
\text{prog}(c') \land 
\neg(\text{taken}(s, c')))) \)
Formalizing English Example

- Does the following predicate calculus expression include the fact the maths class and the programming class are distinct?

\[
(\exists c, c' \mid : \\
(\forall s \mid \text{senior}(s) : \\
\quad \text{taken}(s, c) \land \\
\quad \text{passed}(s, c')))
\]

- Could include a predicate, \text{taken}, as below or state \( c=/=c' \)

\[
(\exists c, c' \mid : \\
(\forall s \mid \text{senior}(s) : \\
\quad \text{math}(c) \land \\
\quad \text{taken}(s, c) \land \\
\quad \text{prog}(c') \land \\
\quad \text{taken}(s, c')))
\]
mod FAMILY {

[Person]

pred mother : Person  Person
pred parent : Person  Person
pred female : Person
op Anne : -> Person
op Ken : -> Person
vars X Y : Person
ax parent(X,Y) & female(X) -> mother(X,Y).
ax parent(Anne,Ken) .
ax female(Anne) .
}
goal mother(Anne, Ken) .
mod FAMILY {                     1. Module name
[Person]                         2. Data sort or type
pred mother : Person  Person    3. predicate by default returns Bool
pred parent : Person  Person    4. takes two persons and returns Bool
pred female : Person            5. takes two persons and returns Bool
op Anne : -> Person             6. takes one person and returns Bool
op Ken : -> Person              7. Constant of type person
vars X Y : Person                8. Constant of type person
ax parent(X,Y) & female(X) -> mother(X,Y). 10. Axiom for mother
ax parent(Anne,Ken) = true .     11. Axiom for fact..
ax female(Anne) = true .         12. Axiom for fact..}
Proof: Anne is Ken’s mother

\textbf{goal(mother(Anne,Ken))}

1: \[\neg(\text{parent(}_v21\text{:Person,}_v22\text{:Person})) \mid \neg(\text{female(}_v21\text{)}) \mid \text{mother(}_v21,\,_v22)\]

7: \[\text{female(Anne) = true}\]

10: \[\text{parent(Anne,Ken) = true}\]

13: \[\neg(\text{mother(Anne,Ken)})\]

44: \[\text{para-from:10,1,demod:7] mother(Anne,Ken)}\]

45: \[\text{binary:44,13}\]
Proof: Beth is Anne's sister

\[ \text{goal}(\text{sister(Beth, Anne)}) \]

\( \text{mod FAMILY} \) 

pr(FOPL-CLAUSE)

\[
\begin{align*}
\text{[Person]} & \\
\text{pred parent: } & \text{Person Person} \\
\text{pred sister: } & \text{Person Person} \\
\text{pred female: } & \text{Person} \\
\text{op Ken : } & \rightarrow \text{Person} \\
\text{op Jim : } & \rightarrow \text{Person} \\
\text{op Anne : } & \rightarrow \text{Person} \\
\text{op Beth : } & \rightarrow \text{Person} \\
\text{vars } X Y Z : & \text{Person} \\
\text{ax female}(X) & \& \text{parent}(Z,X) \& \text{parent}(Z,Y) \rightarrow \text{sister}(X,Y) . \\
\text{ax female}(\text{Anne}) & = \text{true} . \\
\text{ax female}(\text{Beth}) & = \text{true} . \\
\text{ax parent}(\text{Ken,Beth}) & = \text{true} . \\
\text{ax parent}(\text{Ken,Anne}) & = \text{true} . \\
\end{align*}
\]

** PROOF_____________ 

1:[] \neg(\text{female(}_v17:\text{Person})) \mid 

\neg(\text{parent(}_v16:\text{Person, }_v17)) \mid 

\neg(\text{parent(}_v16, \_v18:\text{Person})) \mid \text{sister(}_v17, \_v18) 

3:[] \text{female(Beth)} = \text{true} 

4:[] \text{parent(Ken, Beth)} = \text{true} 

5:[] \text{parent(Ken, Anne)} = \text{true} 

6:[] \neg(\text{sis\text{ter(Beth, Anne)})} 

25:[4,1,3] \neg(\text{parent(Ken, }_v27:\text{Person})) \mid \text{sister(Beth, }_v27) 

33:[25,5] \text{sister(Beth, Anne)} 

34:[33,6] 

** END PROOF_______
Family Tree

We will examine this example when we look at relations later

∀x∀z(∃y(parent(x, y) ∧ ancestor(y, z)) → ancestor(x, z))

We will examine this example when we look at relations later
What can we Prove from equations?

mod* PROF{
  [Elt]
ops p1 p2  faculty prof staff : Elt -> Bool
var X : Elt
op Michael : -> Elt
eq p1(X) = prof(X) implies faculty(X) .
eq p2(X) = faculty(X) implies staff(X) .
eq prof(Michael) = true .
}
open PROF .
red p1(Michael) implies faculty(Michael) .
red p1(Michael) and p2(Michael) implies staff(Michael) .
red (p1(X) and p2(X) and prof(X)) implies staff(X) .
What can we Prove from conditional equations?

mod* PROF{
[Elp]
ops faculty prof staff : Elt -> Bool
var X : Elt
op Michael : -> Elt
ceq faculty(X) = true if prof(X) .
ceq staff(X) = true if faculty(X) .
eq prof(Michael) = true .
}
open PROF .
red faculty(Michael) .
red staff(Michael) .
red (faculty(X) and staff(X) and prof(X)) implies staff(X) .
Skolem Functions

- A Skolem function $f$ is used to construct a new value that depends on the universally quantified variable. For example:
  \[(\forall x)(\exists y)\text{loves}(x, y)\]
  
  Becomes
  \[(\forall x)\text{loves}(x, f(x))\]

- In this case, $f(x)$ specifies the person that $x$ loves.

- In general: $\forall x \exists y A(x, y) \leftrightarrow \exists f \forall x A(x, f(x))$.
Skolem Functions

• In general, the Skolem function is dependent on the universally quantified variables within whose scope it is used.

$$\forall x \exists y A(x, y) \leftrightarrow \exists f \forall x A(x, fx).$$
Skolem Constants & Functions

• Skolemization is the elimination of existential quantifiers through the introduction of new functions or constants, e.g.:

1. \( \exists X. \text{parent}(X,\text{john}) \Rightarrow \text{parent}(p,\text{john}) \)  
   
   [\( p \) is new Skolem Constant]

2. \( \forall X. \text{person}(X) : \exists Y. \text{parent}(Y, X) \Rightarrow \forall X. \text{person}(X) : \text{parent}(\text{parentOf}(X), X) \)  
   
   [\( \text{parent}(\text{parentOf}(X)) \) is new Skolem function]
\[ \forall \text{person}(x) \exists \text{parentOf}(x) \]

mod FAMILY {
[Person]
ops parent ancestor : Person Person \rightarrow Bool
op parentOf : Person \rightarrow Person
ops Emily Jim Anne Beth Gran : \rightarrow Person
vars X Y Z : Person

eq parent(parentOf(X),X) = true. – Not very interesting use of Skolem fun.
eq parent(Emily,Jim) = true .
eq parent(Beth,Emily) = true .
eq parent(Anne,Beth) = true .
eq parent(Gran,Anne) = true .
eq parentOf(Jim) = Emily . 

eq parentOf(Emily) = Beth .
eq parentOf(Beth) = Anne .
eq parentOf(Anne) = Gran . }
open FAMILY .
red parent(parentOf(Jim),Jim) .
Using **Skolem constant** to remove $\exists$

- The existential quantifier ($\exists$) says that some object exists. When we remove the existential quantifier we are giving the object a name. It is safe to move from $\exists x \, \text{dog}(x)$ to $\text{dog}(\text{aSkolemDog})$. 
Unique existential $\exists!$

$\forall x \exists y, z(\text{airline}(x) \land \text{city}(y) \land \text{city}(z) \land \text{flies}(x, y) \land \text{flies}(x, z) \rightarrow (y = z))$.

This asserts that every airline $x$ flies to only one city.

It can be rewritten

$\forall x \exists! y (\text{airline}(x) \land \text{city}(y) \land \text{flies}(x, y))$.

In general $\exists! y \phi(x)$ is shorthand for $\exists x (\phi(x) \land \forall y (\phi(y) \rightarrow y = x))$
What is a logic

- A formal logical system, or logic, is
  - a set of rules defined in terms of a set of symbols,
  - a set of formulas constructed from the symbols,
  - a set of distinguished formulas called axioms, and
  - a set of inference rules.

- The set of formulas is called the language of the logic. The language is defined syntactically; there is no notion of meaning or semantics in a logic per se.
Propositional and Predicate Logic

- The CafeOBJ version of SUPERMAN is expressed in propositional logic i.e. variables can only be of type Boolean. The particular 'flavour' of propositional logic is called 'equational logic' known as E in the course text. This fact is not hugely important on this course, so we will just consider SUPERMAN as being expressed in propositional logic. Propositional logic cannot deal with sentences such as $x > 1$. We cannot tell whether it is true until we know the value or range of $x$. 
Propositional and Predicate Logic

• The version of Family in previous slides uses predicate logic, where variables can be Boolean or any other type e.g. Person. Some authors call the predicates such as $\text{male}(\text{John})$ 'propositional functions'. This terminology links predicate logic back to the simpler propositional logic. Apart from the 'type' of variables used in predicate logic the variables are also quantified. Predicate logic uses 'forAll' ($\forall$) and 'thereExists' ($\exists$) to make statements about variables. Predicate logic is more general than propositional logic and can handle a wider range of problems.
Predicates in CafeOBJ

1. module HOUSE {
2.  [ Person Room Wall ]
3.  pred inRoom : Person
4.  pred inHouse : Person
5.  pred roomHasWall : Room Wall
6.  pred beside : Room Room
7.  op I : -> Person
8.  var X : Person
9.  vars R1 R2 : Room
10. var W1 : Wall
11. ax inRoom(X) -> inHouse(X) .
12  ax inRoom(I) = true .
13  ax roomHasWall(R1,W1) & roomHasWall(R2,W1) 
    -> beside(R1, R2) .
}
•  See description in notes section below.
Predicates in CafeOBJ(*)

- We now explain the result of executing the following commands in CafeOBJ:
  - open HOUSE .
  - goal inHouse(I) .
  - resolve .
- Based on the two axioms CafeOBJ can prove inHouse(I). The axioms in the HOUSE module correspond to a fact and an inference rule in the predicate calculus. The CafeOBJ module definition and the execution of the goal are a software representation of the following argument from predicate calculus.

  - All $X$:Person who is in a room are in a house
  - I am in a room
  - Therefore, I am in a house.
Predicates in CafeOBJ

• Here are 3 operations and two equations that would enable CafeOBJ to prove \( \text{beside(room1,room2)} \) is true.

  • \text{op wall : } \rightarrow \text{ Wall } .
  • \text{op room1 : } \rightarrow \text{ Room } .
  • \text{op room2 : } \rightarrow \text{ Room } .
  • \text{eq roomHasWall(room2, wall) = true .}
  • \text{eq roomHasWall(room1, wall) = true .}
  • -- check if its true
  • \text{goal beside(room1, room2) .}
Contrasting Logics(*)

• SUPERMAN is expressed in propositional logic which uses propositions of type Boolean which can only be true or false.

• HOUSE uses predicate logic in which predicates (or propositional functions) can take types other than Boolean as argument. Also those types can be quantified.

• Contrast: Both logics have a grammar, axioms, and inference rules. The predicate logic is an extension of the propositional logic with a broader range of applications (or possible domains of discourse).
Chapter 7: Logical Consequence

- $G$ is a logical consequence of $X$
- $X \vDash G$

- The symbol $\vDash$ may be read as “semantically implies”, “semantic inference”, “implies”, “entails”.

- Not to be confused with the implication Boolean operator from Chapter 2.

- In model theory, $\text{model } \vDash \text{theory}$ means $\text{model satisfies theory}$ or $\text{model is a model of theory}$. 
Logical Consequence

• ‘$A \vdash_L B$' means: B follows from or is a consequence of A. B is provable from A in logic L.

• ‘$A \not\vdash B$' means: B is a semantic consequence or entailment of A in some model (or valuation system) M (with truth values, etc.) where the argument is valid.

• $\vdash$ is called semantic turnstile

• $\models$ is called syntactic turnstile
Connection World-Representation

Conceptualization

World W

Sentences

represent

Facts about W

entail

Facts about W

represent

hold

hold
Sowa’s view of the world, models and theories

Predicate Logic

http://www.jfsowa.com/logic/theories.htm

Uncle Example Revisited

Facts (model of the world):
BrotherOf(John, Bob).
FatherOf(Bob, Alice).

Rules (theory):
forall P, Q. BrotherOf(P, Q) = BrotherOf(Q, P).
forall P, Q. SiblingOf(P, Q) = SisterOf(P, Q | BrotherOf(P, Q).
forall P, R. UncleOf(P, R) = exists Q. BrotherOf(P, Q) & FatherOf(Q, R).

Query:
SiblingOf(Bob, John) ? := BrotherOf(Bob, John) := BrotherOf(John, Bob).
UncleOf(John, Alice) ? := BrotherOf(John, Bob) & FatherOf(Bob, Alice).

First-Order Predicate Logic (FOL)

- Offers full power but undecidable - "if you can say something, you can say it in FOL"
- Minimal set of constructs but impractical for specification,
- Many super and subsets provide syntactic sugar and/or restrictions.

Great summary of math background
http://www.jfsowa.com/logic/math.htm

Chapter 7: Formal Logic

- **syntax**: formal structure of *sentences*
- **semantics**: *truth* of sentences wrt *models*
- **entailment**: necessary truth of one sentence given another
- **inference**: deriving sentences from other sentences, provides a syntactic mechanism for deriving ‘truth’
- **soundness**: derivations produce only entailed sentences
- **completeness**: derivations can produce all entailed sentences
Formal Logic

• A logic consists of:
  – A set of symbols
  – A set of formulas constructed from the symbols
  – A set of distinguished formulas called axioms
  – A set of inference rules.

• The set of formulas is called the language of the logic, pure syntax.

• Inference rules (premise and conclusion) allow formulas to be derived from other formula.
Formal Logic

• A formula is a *theorem* of the logic if it is an axiom or can be generated from the axioms and already proven theorems.
• A *proof* that a formula is a theorem is an argument that shows how the inference rules are used to generate the formula.
Equality in numbers

• The variables $X$ and $Y$ are used to show a general law for numbers

• $(5+X) \times (5+Y) = (X+Y) \times 10 + (5-X) \times (5-Y)$

• $X = 2, \ Y = 3$ is one example,

• $(5+2) \times (5+3) = 56$ = $(5 \times 10 + 3 \times 2 = 56)$

• Replacing $X$ and $Y$ by an arbitrary number will produce an equality.

• $X = 1, \ Y = 2$ is another example