First Order Predicate Calculus
Propositional logic v. FOPC

Propositional calculus deals only with facts
- P : I-love-all-dogs
- facts are either true or they are false

Predicate calculus makes a stronger commitment to what there is (ontology)
- objects: things in the world (no truth value)
- properties & relations of and between objects (truth value)

FOPC breaks facts down into objects & relations
- can be seen as an extension of propositional logic
Predicate calculus

- **Terms** refer to *objects* in the world
  - John, Mary, etc.
  - *functions* (one-to-one mapping)
  - these terms do *not* have a truth value assigned to them

- **Predicates** [propositions with arguments]
  - marriedTo(John, Mary)

- **Quantifiers**
  - $\forall x[\text{valuable}(x)]$
  - $\exists x[\text{valuable}(x)]$

- **Equality**: do two terms refer to the same object?
Terms

Logical expressions that refer to objects

- **Constants** (by convention, capitalized)
  - e.g., Sue

- **Variables** (by convention, lower case)
  - used with quantifiers
  - e.g., x

- **Functions**
  - MotherOf(Sue)
  - since functions represent objects, we can nest them
    - MotherOf(MotherOf(Ann))
  - don’t need explicit names
    - LeftFootOf(John)
Sentences

Just as in propositional logic, sentences have a truth-value.

In FOPC, only relations (predicates) have truth-values.

Thus, terms alone are not wffs.

They must be (part of) an argument to a predicate.
Making sentences

原子句子: 一个单一的谓词
- married(Sue, FatherOf(Ann))

复杂句子
- 正如在命题逻辑中，我们可以通过连接词或、和、蕴含、等价、否定来组合谓词，从而制作更复杂的句子
  - 或，和，蕴含，等价，否定
- 量词
- 等级
Quantifiers

Allows us to express properties of categories of objects without listing all of the objects

Universal

- $\forall x[P(x)] : T$ if $P(x)$ is true for every object in our interpretation
- e.g., all men are mortal

Existential

- $\exists x[P(x)] : T$ if $P(x)$ is $T$ for some object in our interpretation
- e.g., I love some dog
Equality

• An in-fix predicate
  • but a predicate all the same [returns true or false]
  • e.g., FatherOf(John) = Henry

• termA = termB is shorthand for equal(termA, termB)
  • doesn’t have to be in-fix; convenient

• Will be true if termA & termB refer to the same object
Backus-Naur form

Sentence → AtomicSentence
  | Sentence Connective Sentence
  | Quantifier Variable,… Sentence
  | ¬ Sentence
  | ( Sentence )

Atomic Sentence → Predicate (Term,…)
  | Term = Term
BNF (cont.)

Term $\rightarrow$ Function ( Term,…)
   | Constant
   | Variable
Connective $\rightarrow$ $\Rightarrow$ | $\land$ | $\lor$ | $\equiv$
Quantifier $\rightarrow$ $\forall$ | $\exists$
BNF (cont.)

Constant → A | X₁ | John | . . .
Variable → a | x | s | . . .
Predicate → before | hasColor | raining | . . .
Function → MotherOf | LeftLegOf | . . .
Well Formed Formulas (WFFs)?

tall(john)

MotherOf(john)

mother(john, mary)

John = brother(Bill)

- equal(john, brother(Bill))

F(p(1) \land q(2))
“Every rational number is a real number”

\[ \forall x [\text{rational}(x) \implies \text{real}(x)] \]

What about

\[ \forall x [\text{rational}(x) \land \text{real}(x)] \]?
\[ \exists x [\text{rational}(x) \land \text{real}(x)] \]?
\[ \forall x [\neg \text{real}(x) \lor \text{rational}(x)] \]?
More English → FOPC

★ There is a prime number greater than 100
  ● \( \exists x [\text{prime}(x) \land \text{greaterThan}(x, 100)] \)
  ● \( \exists x [\text{prime}(x) \land x > 100] \)

★ There is no largest prime
  ● no-largest-prime
  ● \( \forall x [\text{prime}(x) \Rightarrow \exists y [\text{prime}(y) \land \text{greaterThan}(y, x)]] \)

★ Every number has an additive inverse
  ● \( \forall x [\text{number}(x) \Rightarrow \exists y [\text{equal}(\text{Plus}(x, y), 0)]] \)
Mixing quantifiers

Everyone likes a dog
- $\forall x[\text{human}(x) \Rightarrow \exists y[\text{dog}(y) \land \text{likes}(x, y)]]$

There’s one dog everyone likes
- $\exists y[\text{dog}(y) \land \forall x[\text{human}(x) \Rightarrow \text{likes}(x, y)]]$

Everyone likes a different dog
- $\forall x[\text{human}(x) \Rightarrow \exists y[\text{dog}(y) \land \text{likes}(x, y) \land \forall z[\text{human}(z) \land \text{likes}(z, y) \Rightarrow x = z]]$
Location of quantifiers

Everyone likes a dog
- $\forall x[\text{human}(x) \Rightarrow \exists y[\text{dog}(y) \land \text{likes}(x, y)]]$

What about
- $\forall x[\exists y[(\text{human}(x) \land \text{dog}(y)) \Rightarrow \text{likes}(x, y)]]$
  - $\text{human}(\text{Jim}) : T; \text{human}(\text{Spot}) : F$
  - $\text{dog}(\text{Jim}) : F; \text{dog}(\text{Spot}) : T$
  - $\text{likes}(\text{Jim,Jim}) : T; \text{likes}(\text{Jim,Spot}) : F$
  - $\text{likes}(\text{Spot,Jim}) : F; \text{likes}(\text{Spot,Spot}) : T$

In general, never use $\exists x$ with $\Rightarrow$, and don’t use $\forall x$ with $\land$
What is the truth-value of FOPC wffs? FOPC interpretations

- The **user** must provide a finite list of objects in the world
  - “universe of discourse”

- For each *function*, a mapping from “parameter setting” to an object in the world
  - e.g., Father(John) maps to “Bill”

- For each *predicate*, a mapping from each “parameter setting” to true or false
Determining truth-value of FOPC wff

- Specify an interpretation, I
- Obtain truth-values of
  - predicates
    - look up functions until only constants remain & then look up the truth value of the predicate
  - termA = termB
    - look up functions until only constants remain; true if same constant; false otherwise
  - wffA connective wffB
    - compute the truth-value of the wffs
    - use connective’s truth table to determine the truth-value of compound wff (same for ¬)
Truth-value for quantifiers

- $\forall x \text{ wff}(x)$
  - successively replace $x$ by each constant in the interpretation
  - if $\text{wff}(\text{constant})$ is true for every case, then $\forall x(\text{wff})$ is true

- $\exists x \text{ wff}(x)$
  - same as above, but $\text{wff}(\text{constant})$ has only to be true once

- assume constants list is never empty
Representing change

- On(BlockA, BlockB)
  - this is either T or F
  - there is no way to change this fact in "basic" FOPC

Solution: "time stamp" wffs
- add one more parameter to all predicates indicating when they are true
- On(BlockA, BlockB, S0)
- On(BlockA, BlockC, S1)
  - where S0 & S1 are situations or states
Changing the world

 Acting (operator applications) changes states (situations) into other states

 We need a name for the new state
  ● Use functions!
  ● In particular, the function $Result$
    • maps an action and a state to a new state
    • $Result(<\text{action}>, <\text{state}>) \Rightarrow <\text{state}>$
    • simply a fancy name for a state, just as $\text{FatherOf}(---)$ is a fancy name for some man
Block-world

Block A

Block C

Table T

On(A,C)

On(B,T)
Block-world example

\[\forall x, y, z, s [\text{block}(x) \land \text{block}(y) \land \text{table}(z) \land \text{state}(s) \land \text{on}(x, z, s) \land \text{clear}(x, s) \land \text{clear}(y, s)] \implies \]
\[\text{state}(\text{Result}(\text{Stack}(x, y), s)) \land \text{on}(x, y, \text{Result}(\text{Stack}(x, y), s)) \land \text{clear}(x, \text{Result}(\text{Stack}(x, y), s)) \land \neg \text{clear}(y, \text{Result}(\text{Stack}(x, y), s))\]

Could now almost use deductions to produce plans (sequences of action)
What’s missing?

What do we know about c in the new state?

This is a case of the frame problem: knowing what stays the same as we move from state to state (like frames in a movie)

“Blocks stay clear unless something is placed on them during stacking”

∀u,x,y,s[clear(u, s) ∧ ¬(u = y) ⇒ clear(u, Result(Stack(x, y), s))]
Example

Painting a house does not change who owns it

∀s,h,p[\text{state}(s) \land \text{house}(h) \land \text{human}(p) \land \text{owns}(p, h, s) \Rightarrow \text{owns}(p, h, \text{Result}(\text{paint}(h), s))]

Alternate approach

Say properties stay the same unless a specific action performed

\[ \forall u, x, s, a \ [\text{block}(u) \land \text{state}(s) \land \text{action}(a) \land \text{clear}(u, s) \land \neg (a = \text{Stack}(x, u)) \land \neg (a = \text{CoverWithBlanket}(u)) \land \neg (a = \text{Smash}(u)) \implies \text{clear}(u, \text{Result}(a, s))] \]

This usually leads to fewer rules, but it is less modular

- when new actions defined, we have to double check every such rule to see if it needs editing
Problems with formalization

- **Qualification** problem
  - can we ever really write down all the “preconditions” for a real-world action?
  - E.g., starting a car

- **Ramification** problem
  - need to represent implicit consequences of actions
  - moving car from A to B also moves its steering wheel, spare tire, etc.
  - can be handled but becomes tedious
Sources

Computer Science Lab
University of Wisconsin, Madison