Organisational Reminders

There is a Blackboard Forum for questions related to course material/labs.

There is a Prolog Help Session at 1pm in 1.10 today.

Lecture 7 First-Order Logic

COMP24412: Symbolic AI

Giles Reger

February 2019

Aim and Learning Outcomes

The aim of this lecture is to:

Introduce you to the general language of first-order logic including its syntax, semantics, and how it relates to Datalog and Prolog.

Learning Outcomes

By the end of this lecture you will be able to:

1. Write formulas in first-order logic modelling real-world situations and describe their meaning
2. Recall the notion of an *interpretation* in first-order logic and apply it to determine the truth of a first-order logic
3. Give examples of why first-order logic is more expressive than Datalog or Prolog

Recap

Representation and Reasoning

Representation	Reasoning	Properties
Datalog: Facts and Rules Database Semantics Rules: $f_1 \wedge \ldots \wedge f_2 \Rightarrow h$ vars in head must be in body function-free, negation-free	**Forward Chaining** Compute all consequences Answer atomic queries Deductive Database	Finite Terminating Single Interpretation
Prolog: extend Datalog with Functions e.g. lists Lift variable restriction Non-logic part	**Backward Chaining** Conjunctive queries Use rules to reduce goal to subgoals (backtracking)	Possibly Infinite Turing-complete

Recap

Representation and Reasoning

Representation	Reasoning	Properties
Datalog: Facts and Rules Database Semantics Rules: $f_1 \wedge \ldots \wedge f_2 \Rightarrow h$ vars in head must be in body function-free, negation-free	**Forward Chaining** Compute all consequences Answer atomic queries Deductive Database	Finite Terminating Single Interpretation
Prolog: extend Datalog with Functions e.g. lists Lift variable restriction Non-logic part	**Backward Chaining** Conjunctive queries Use rules to reduce goal to subgoals (backtracking)	Possibly Infinite Turing-complete
First-order logic No Database Semantics General formulas	**Proof Search** Saturation-based Lots of heuristics	Semi-decidable

Today

(Revising) First-Order (Predicate) Logic - defining formulas

What formulas mean (intuitively)

What formulas mean (formally)

Relating this to things we've seen

First-order logic: Syntax

Definition (Signature)

The signature $\Sigma = \langle \mathcal{F}, \mathcal{P}, \mathsf{arr} \rangle$ of a first-order formula consists of a disjoint sets of function and predicate symbols $\mathcal{F}$ and $\mathcal{P}$, and a function $\mathsf{arr} : \mathcal{F} \cup \mathcal{P} \rightarrow \mathbb{N}_0$ giving the arity (number of parameters) for each symbol.

First-order logic: Syntax

Definition (Signature)

The signature $\Sigma = \langle \mathcal{F}, \mathcal{P}, \text{arr} \rangle$ of a first-order formula consists of a disjoint sets of function and predicate symbols $\mathcal{F}$ and $\mathcal{P}$, and a function $\text{arr} : \mathcal{F} \cup \mathcal{P} \to \mathbb{N}_0$ giving the arity (number of parameters) for each symbol.

Definition (Term - similar to Prolog)

A variable is a term. A constant symbol ($c \in \mathcal{F}$ s.t. $\text{arr}(c) = 0$) is a term. If $f \in \mathcal{F}$ and $\text{arr}(f) = n > 0$ then $f(t_1, \ldots t_n)$ is a term if each t_i is a term.

First-order logic: Syntax

Definition (Signature)

The signature $\Sigma = \langle \mathcal{F}, \mathcal{P}, \text{arr} \rangle$ of a first-order formula consists of a disjoint sets of function and predicate symbols $\mathcal{F}$ and $\mathcal{P}$, and a function $\text{arr} : \mathcal{F} \cup \mathcal{P} \to \mathbb{N}_0$ giving the arity (number of parameters) for each symbol.

Definition (Term - similar to Prolog)

A variable is a term. A constant symbol ($c \in \mathcal{F}$ s.t. $\text{arr}(c) = 0$) is a term. If $f \in \mathcal{F}$ and $\text{arr}(f) = n > 0$ then $f(t_1, \ldots t_n)$ is a term if each t_i is a term.

Terms are similar to Prolog in syntax but not semantics. In first-order logic terms do not automatically have an interpretation as algebraic datatypes.

E.g. it is possible that $f(a, b) = a$, or $add(one, zero) = one$.

First-order logic: Syntax

Definition (Atoms)

A proposition ($p \in \mathcal{P}$ s.t. $\mathrm{arr}(p) = 0$) is an atom. If t_1 and t_2 are terms then $t_1 \neq t_2$ is an atom. If $p \in \mathcal{P}$ and $\mathrm{arr}(p) = n > 0$ then $p(t_1, \ldots t_n)$ is an atom if each t_i is a term.

First-order logic: Syntax

Definition (Atoms)

A proposition ($p \in \mathcal{P}$ s.t. $\text{arr}(p) = 0$) is an atom. If t_1 and t_2 are terms then $t_1 \neq t_2$ is an atom. If $p \in \mathcal{P}$ and $\text{arr}(p) = n > 0$ then $p(t_1, \ldots t_n)$ is an atom if each t_i is a term.

Definition (Formula)

All atoms are formulas. The boolean value true is a formula. If ϕ_1 and ϕ_2 are formulae then so are $\neg\phi_1$, $\phi_1 \wedge \phi_2$, and $\forall x.\phi_1$.

First-order logic: Syntax

Definition (Atoms)

A proposition ($p \in \mathcal{P}$ s.t. $\text{arr}(p) = 0$) is an atom. If t_1 and t_2 are terms then $t_1 \neq t_2$ is an atom. If $p \in \mathcal{P}$ and $\text{arr}(p) = n > 0$ then $p(t_1, \ldots t_n)$ is an atom if each t_i is a term.

Definition (Formula)

All atoms are formulas. The boolean value true is a formula. If ϕ_1 and ϕ_2 are formulae then so are $\neg\phi_1$, $\phi_1 \wedge \phi_2$, and $\forall x.\phi_1$.

We define false $= \neg$true, $\phi_1 \vee \phi_2 = \neg(\phi_1 \wedge \phi_2)$, $\phi_1 \rightarrow \phi_2 = \neg\phi_1 \vee \phi_2$, $\phi_1 \leftrightarrow \phi_2 = ((\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1))$ and $\exists x.\phi = \neg(\forall x.\neg\phi)$.

First-order logic: Syntax

Definition (Atoms)

A proposition ($p \in \mathcal{P}$ s.t. $\text{arr}(p) = 0$) is an atom. If t_1 and t_2 are terms then $t_1 \neq t_2$ is an atom. If $p \in \mathcal{P}$ and $\text{arr}(p) = n > 0$ then $p(t_1, \ldots t_n)$ is an atom if each t_i is a term.

Definition (Formula)

All atoms are formulas. The boolean value true is a formula. If ϕ_1 and ϕ_2 are formulae then so are $\neg\phi_1$, $\phi_1 \wedge \phi_2$, and $\forall x.\phi_1$.

We define false $= \neg$true, $\phi_1 \vee \phi_2 = \neg(\phi_1 \wedge \phi_2)$, $\phi_1 \rightarrow \phi_2 = \neg\phi_1 \vee \phi_2$, $\phi_1 \leftrightarrow \phi_2 = ((\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1))$ and $\exists x.\phi = \neg(\forall x.\neg\phi)$.

In terms of precedence, unary symbols bind tighter than binary symbols. Otherwise, I use parenthesis to disambiguate.

How to Read Formulas

$=$	equals
$\neg$	not
$\wedge$	and
$\vee$	or
$\rightarrow$	implies, or sometimes if *left* then *right*
$\leftrightarrow$	equivalent, bi-implication, if-and-only-if
$\forall$	(for) all, (for) every
$\exists$	exists, some

I will write $\neq$ (read 'not equal') instead of $\neg(t_1 = t_2)$ as it is nicer.

Examples of Universal Quantification

We should be used to Universal quantification by now, it is what we have been implicitly using in rules.

Every man is human
Every red car is cool
Every bird that is not a penguin or an ostrich can fly
If something is either fruit or cheese then it is food
Every car is either petrol or diesel
No car is both petrol and diesel

Examples of Universal Quantification

We should be used to Universal quantification by now, it is what we have been implicitly using in rules.

Every man is human	`human(X) :- man(X).`
Every red car is cool	
Every bird that is not a penguin or an ostrich can fly	
If something is either fruit or cheese then it is food	
Every car is either petrol or diesel	
No car is both petrol and diesel	

Examples of Universal Quantification

We should be used to Universal quantification by now, it is what we have been implicitly using in rules.

Every man is human	$\forall x.(man(x) \rightarrow human(x))$
Every red car is cool	
Every bird that is not a penguin or an ostrich can fly	
If something is either fruit or cheese then it is food	
Every car is either petrol or diesel	
No car is both petrol and diesel	

Examples of Universal Quantification

We should be used to Universal quantification by now, it is what we have been implicitly using in rules.

Every man is human	$\forall x.(man(x) \rightarrow human(x))$
Every red car is cool	`cool(X) :- red(X), car(X).`
Every bird that is not a penguin or an ostrich can fly	
If something is either fruit or cheese then it is food	
Every car is either petrol or diesel	
No car is both petrol and diesel	

Examples of Universal Quantification

We should be used to Universal quantification by now, it is what we have been implicitly using in rules.

Every man is human	$\forall x.(man(x) \rightarrow human(x))$
Every red car is cool	$\forall x.((red(x) \land car(x)) \rightarrow cool(x))$
Every bird that is not a penguin or an ostrich can fly	
If something is either fruit or cheese then it is food	
Every car is either petrol or diesel	
No car is both petrol and diesel	

Examples of Universal Quantification

We should be used to Universal quantification by now, it is what we have been implicitly using in rules.

Every man is human	$\forall x.(man(x) \rightarrow human(x))$
Every red car is cool	$\forall x.((red(x) \land car(x)) \rightarrow cool(x))$
Every bird that is not a penguin or an ostrich can fly	`fly(X) :- bird(X), dif(X,ostrich), dif(X,penguin.`
If something is either fruit or cheese then it is food	
Every car is either petrol or diesel	
No car is both petrol and diesel	

Examples of Universal Quantification

We should be used to Universal quantification by now, it is what we have been implicitly using in rules.

Every man is human	$\forall x.(man(x) \rightarrow human(x))$
Every red car is cool	$\forall x.((red(x) \land car(x)) \rightarrow cool(x))$
Every bird that is not a penguin or an ostrich can fly	$\forall x.(bird(x) \land x \neq penguin \land x \neq ostrich) \rightarrow fly(x)$
If something is either fruit or cheese then it is food	
Every car is either petrol or diesel	
No car is both petrol and diesel	

Examples of Universal Quantification

We should be used to Universal quantification by now, it is what we have been implicitly using in rules.

Every man is human	$\forall x.(man(x) \rightarrow human(x))$
Every red car is cool	$\forall x.((red(x) \wedge car(x)) \rightarrow cool(x))$
Every bird that is not a penguin or an ostrich can fly	$\forall x.(bird(x) \wedge x \neq penguin \wedge x \neq ostrich) \rightarrow fly(x)$
If something is either fruit or cheese then it is food	`food(X) :- fruit(X).` `food(X) :- cheese(X).`
Every car is either petrol or diesel	
No car is both petrol and diesel	

Examples of Universal Quantification

We should be used to Universal quantification by now, it is what we have been implicitly using in rules.

Every man is human	$\forall x.(man(x) \rightarrow human(x))$
Every red car is cool	$\forall x.((red(x) \wedge car(x)) \rightarrow cool(x))$
Every bird that is not a penguin or an ostrich can fly	$\forall x.(bird(x) \wedge x \neq penguin \wedge x \neq ostrich) \rightarrow fly(x)$
If something is either fruit or cheese then it is food	$(\forall x.fruit(x) \rightarrow food(x)) \wedge (\forall x.cheese(x) \rightarrow food(x))$
Every car is either petrol or diesel	
No car is both petrol and diesel	

Examples of Universal Quantification

We should be used to Universal quantification by now, it is what we have been implicitly using in rules.

Every man is human	$\forall x.(man(x) \rightarrow human(x))$
Every red car is cool	$\forall x.((red(x) \wedge car(x)) \rightarrow cool(x))$
Every bird that is not a penguin or an ostrich can fly	$\forall x.(bird(x) \wedge x \neq penguin \wedge x \neq ostrich) \rightarrow fly(x)$
If something is either fruit or cheese then it is food	$\forall x. \left(\begin{array}{c} fruit(x) \vee \\ cheese(x) \end{array} \right) \rightarrow fruit(x)$
Every car is either petrol or diesel	
No car is both petrol and diesel	

Examples of Universal Quantification

We should be used to Universal quantification by now, it is what we have been implicitly using in rules.

Every man is human	$\forall x.(man(x) \rightarrow human(x))$
Every red car is cool	$\forall x.((red(x) \wedge car(x)) \rightarrow cool(x))$
Every bird that is not a penguin or an ostrich can fly	$\forall x.(bird(x) \wedge x \neq penguin \wedge x \neq ostrich) \rightarrow fly(x)$
If something is either fruit or cheese then it is food	$\forall x. \left(\begin{array}{c} fruit(x) \vee \\ cheese(x) \end{array} \right) \rightarrow fruit(x)$
Every car is either petrol or diesel	
No car is both petrol and diesel	Cannot be expressed in Prolog

Examples of Universal Quantification

We should be used to Universal quantification by now, it is what we have been implicitly using in rules.

Every man is human	$\forall x.(man(x) \rightarrow human(x))$
Every red car is cool	$\forall x.((red(x) \wedge car(x)) \rightarrow cool(x))$
Every bird that is not a penguin or an ostrich can fly	$\forall x.(bird(x) \wedge x \neq penguin \wedge x \neq ostrich) \rightarrow fly(x)$
If something is either fruit or cheese then it is food	$\forall x. \begin{pmatrix} fruit(x) \vee \\ cheese(x) \end{pmatrix} \rightarrow fruit(x)$
Every car is either petrol or diesel	$\forall x.car(x) \rightarrow \begin{pmatrix} petrol(x) \vee \\ diesel(x) \end{pmatrix}$
No car is both petrol and diesel	

Examples of Universal Quantification

We should be used to Universal quantification by now, it is what we have been implicitly using in rules.

Every man is human	$\forall x.(man(x) \rightarrow human(x))$
Every red car is cool	$\forall x.((red(x) \wedge car(x)) \rightarrow cool(x))$
Every bird that is not a penguin or an ostrich can fly	$\forall x.(bird(x) \wedge x \neq penguin \wedge x \neq ostrich) \rightarrow fly(x)$
If something is either fruit or cheese then it is food	$\forall x.\left(\begin{array}{c} fruit(x) \vee \\ cheese(x) \end{array}\right) \rightarrow fruit(x)$
Every car is either petrol or diesel	$\forall x.car(x) \rightarrow \left(\begin{array}{c} petrol(x) \vee \\ diesel(x) \end{array}\right)$
No car is both petrol and diesel	`petrol(X) :- not(diesel(X)).`

Examples of Universal Quantification

We should be used to Universal quantification by now, it is what we have been implicitly using in rules.

Every man is human	$\forall x.(man(x) \rightarrow human(x))$
Every red car is cool	$\forall x.((red(x) \wedge car(x)) \rightarrow cool(x))$
Every bird that is not a penguin or an ostrich can fly	$\forall x.(bird(x) \wedge x \neq penguin \wedge x \neq ostrich) \rightarrow fly(x)$
If something is either fruit or cheese then it is food	$\forall x. \left(\begin{array}{l} fruit(x) \vee \\ cheese(x) \end{array} \right) \rightarrow fruit(x)$
Every car is either petrol or diesel	$\forall x.car(x) \rightarrow \left(\begin{array}{l} petrol(x) \vee \\ diesel(x) \end{array} \right)$
No car is both petrol and diesel	$\forall x.\neg(petrol(x) \wedge diesel(x))$

Examples of Universal Quantification

We should be used to Universal quantification by now, it is what we have been implicitly using in rules.

Every man is human	$\forall x.(man(x) \rightarrow human(x))$
Every red car is cool	$\forall x.((red(x) \wedge car(x)) \rightarrow cool(x))$
Every bird that is not a penguin or an ostrich can fly	$\forall x.(bird(x) \wedge x \neq penguin \wedge x \neq ostrich) \rightarrow fly(x)$
If something is either fruit or cheese then it is food	$\forall x. \left(\begin{array}{l} fruit(x) \vee \\ cheese(x) \end{array} \right) \rightarrow fruit(x)$
Every car is either petrol or diesel	$\forall x.car(x) \rightarrow \left(\begin{array}{l} petrol(x) \vee \\ diesel(x) \end{array} \right)$
No car is both petrol and diesel	$\neg\exists x.(petrol(x) \wedge diesel(x))$

Examples of Universal Quantification

We should be used to Universal quantification by now, it is what we have been implicitly using in rules.

Every man is human	$\forall x.(man(x) \rightarrow human(x))$
Every red car is cool	$\forall x.((red(x) \wedge car(x)) \rightarrow cool(x))$
Every bird that is not a penguin or an ostrich can fly	$\forall x.(bird(x) \wedge x \neq penguin \wedge x \neq ostrich) \rightarrow fly(x)$
If something is either fruit or cheese then it is food	$\forall x. \left(\begin{array}{l} fruit(x) \vee \\ cheese(x) \end{array} \right) \rightarrow fruit(x)$
Every car is either petrol or diesel	$\forall x.car(x) \rightarrow \left(\begin{array}{l} petrol(x) \vee \\ diesel(x) \end{array} \right)$
No car is both petrol and diesel	$\neg\exists x.(petrol(x) \wedge diesel(x))$

Remember: Every Yacht that I own is made of gold.

Examples of Existential Quantification

We don't have existential quantification in Prolog as it is negated universal quantification.

Examples of Existential Quantification

We don't have existential quantification in Prolog as it is negated universal quantification.

There is a human man
There are at least two men
Everybody has a mother

If two different people have parents who are siblings then they are cousins

Examples of Existential Quantification

We don't have existential quantification in Prolog as it is negated universal quantification.

There is a human man	$\exists x.(man(x) \land human(x))$
There are at least two men	
Everybody has a mother	

If two different people have parents who are siblings then they are cousins

Examples of Existential Quantification

We don't have existential quantification in Prolog as it is negated universal quantification.

There is a human man	$\exists x.(man(x) \land human(x))$
There are at least two men	$\exists x.\exists y.(man(x) \land man(y) \land x \neq y)$
Everybody has a mother	

If two different people have parents who are siblings then they are cousins

Examples of Existential Quantification

We don't have existential quantification in Prolog as it is negated universal quantification.

There is a human man	$\exists x.(man(x) \wedge human(x))$
There are at least two men	$\exists x.\exists y.(man(x) \wedge man(y) \wedge x \neq y)$
Everybody has a mother	$\forall x.\exists y.(person(x) \rightarrow mother\_of(y, x))$

If two different people have parents who are siblings then they are cousins

Examples of Existential Quantification

We don't have existential quantification in Prolog as it is negated universal quantification.

There is a human man	$\exists x.(man(x) \wedge human(x))$
There are at least two men	$\exists x.\exists y.(man(x) \wedge man(y) \wedge x \neq y)$
Everybody has a mother	$\forall x.\exists y.(person(x) \rightarrow mother\_of(y, x))$

If two different people have parents who are siblings then they are cousins

$$\forall x, y.(x \neq y \wedge (\exists u, v. \begin{pmatrix} parent(u, x) \wedge \\ parent(v, y) \wedge \\ sibling(u, v) \end{pmatrix})) \rightarrow cousins(x, y)$$

Examples of Existential Quantification

We don't have existential quantification in Prolog as it is negated universal quantification.

There is a human man	$\exists x.(man(x) \wedge human(x))$
There are at least two men	$\exists x.\exists y.(man(x) \wedge man(y) \wedge x \neq y)$
Everybody has a mother	$\forall x.\exists y.(person(x) \rightarrow mother\_of(y, x))$

If two different people have parents who are siblings then they are cousins

$$\forall x, y, u, v.(x \neq y \wedge \begin{pmatrix} parent(u, x) \wedge \\ parent(v, y) \wedge \\ sibling(u, v) \end{pmatrix}) \rightarrow cousins(x, y)$$

Examples of Existential Quantification

We don't have existential quantification in Prolog as it is negated universal quantification.

Examples of Existential Quantification

We don't have existential quantification in Prolog as it is negated universal quantification.

Not quite true.

When we ask a query `brother(X,zeus).` we are asking if there exists an object for `X` that makes the fact true.

We'll return to that point later.

Examples of Existential Quantification

Existential quantification over an empty domain is false.

Existential quantification over a finite domain is finite disjunction

$$(\forall x.(x = a \vee x = b)) \rightarrow ((\exists x.p(x)) \leftrightarrow (p(a) \vee p(b)))$$

Which Formula?

Everybody is either a Manchester United Supporter or a Manchester City Supporter and cannot be both.

1. $\forall p.(person(p) \rightarrow (manU(p) \vee manC(p)))$
2. $\forall p.(person(p) \rightarrow ((manU(p) \vee manC(p)) \wedge \neg(manU(p) \wedge manC(p))))$
3. $\exists p_1, p_2.(person(p_1) \wedge person(p_2) \wedge p_1 \neq p_2 \wedge manU(p_1) \wedge manC(p_2))$
4. $\forall p.(person(p) \rightarrow (\exists s.(supports(p, s) \wedge (s = manU \vee s = manC))))$

Which Formula?

Everybody is either a Manchester United Supporter or a Manchester City Supporter and cannot be both.

1. $\forall p.(person(p) \rightarrow (manU(p) \vee manC(p)))$
2. $\forall p.(person(p) \rightarrow ((manU(p) \vee manC(p)) \wedge \neg(manU(p) \wedge manC(p))))$
3. $\exists p_1, p_2.(person(p_1) \wedge person(p_2) \wedge p_1 \neq p_2 \wedge manU(p_1) \wedge manC(p_2))$
4. $\forall p.(person(p) \rightarrow (\exists s.(supports(p, s) \wedge (s = manU \vee s = manC))))$

Which Formula?

Everybody is either a Manchester United Supporter or a Manchester City Supporter and cannot be both.

1. $\forall p.(person(p) \rightarrow (manU(p) \vee manC(p))) \wedge (manU(p) \leftrightarrow \neg manC(p))$
2. $\forall p.(person(p) \rightarrow ((manU(p) \vee manC(p)) \wedge \neg(manU(p) \wedge manC(p))))$
3. $\exists p_1, p_2.(person(p_1) \wedge person(p_2) \wedge p_1 \neq p_2 \wedge manU(p_1) \wedge manC(p_2))$
4. $\forall p.(person(p) \rightarrow (\exists s.(supports(p, s) \wedge (s = manU \vee s = manC))))$

Which Formula?

Everybody is either a Manchester United Supporter or a Manchester City Supporter and cannot be both.

1. $\forall p.(person(p) \rightarrow (manU(p) \vee manC(p))) \wedge (manU(p) \leftrightarrow \neg manC(p))$
2. $\forall p.(person(p) \rightarrow ((manU(p) \vee manC(p)) \wedge \neg(manU(p) \wedge manC(p))))$
3. $\exists p_1, p_2.(person(p_1) \wedge person(p_2) \wedge p_1 \neq p_2 \wedge manU(p_1) \wedge manC(p_2))$
4. $\forall p.(person(p) \rightarrow (\exists s.(supports(p, s) \wedge (s = manU \vee s = manC))))$
5. $\forall p.(person(p) \rightarrow \left(\begin{array}{l} supports(p) = manU \vee \\ supports(p) = manC \end{array} \right)) \wedge manU \neq manC$

Which Formula?

There is a book that if I read it I will score the best mark in all of my exams.

1. $\forall e.\exists b.(read(b) \land (take(e) \rightarrow best(e)))$
2. $\exists b.(read(b) \rightarrow \forall e.(take(e) \rightarrow best(e)))$
3. $(\exists b.read(b)) \land (\forall e.(take(e) \rightarrow best(e)))$
4. $\exists b.(read(b) \rightarrow \forall e.(take(e) \rightarrow \forall p.(takes(p, e) \rightarrow better(e, p))))$

Which Formula?

There is a book that if I read it I will score the best mark in all of my exams.

1. $\forall e.\exists b.(read(b) \wedge (take(e) \rightarrow best(e)))$
2. $\exists b.(read(b) \rightarrow \forall e.(take(e) \rightarrow best(e)))$
3. $(\exists b.read(b)) \wedge (\forall e.(take(e) \rightarrow best(e)))$
4. $\exists b.(read(b) \rightarrow \forall e.(take(e) \rightarrow \forall p.(takes(p, e) \rightarrow better(e, p))))$

Which Formula?

There is a book that if I read it I will score the best mark in all of my exams.

1. $\forall e.\exists b.(read(b) \wedge (take(e) \rightarrow best(e)))$
2. $\exists b.(read(b) \rightarrow \forall e.(take(e) \rightarrow best(e)))$
3. $(\exists b.read(b)) \wedge (\forall e.(take(e) \rightarrow best(e)))$
4. $\exists b.(read(b) \rightarrow \forall e.(take(e) \rightarrow \forall p.(takes(p, e) \rightarrow better(e, p))))$

Why First-Order Logic

The order of a logic is related to the kind of things one can quantify over.

Propositional logic is ‘zero’ ordered as one cannot quantify.

Extensions of propositional logic such as QBF or PLFD are useful for modelling and allow for efficient reasoning but do not increase expressive power.

First-order logic allows one to quantify over individuals

Second-order logic allows one to quantify over sets of individuals or equivalently predicts over individuals.

And so on.

Free Variables, Sentences and Closure

The free variables of a formula f are those not captured by a quantifer. Otherwise they are bound variables.

E.g. x and y are free in $(\forall y.p(x, y)) \land (\exists x.p(x, y))$

Free Variables, Sentences and Closure

The free variables of a formula f are those not captured by a quantifer. Otherwise they are bound variables.

E.g. x and y are free in $(\forall y.p(x, y)) \wedge (\exists x.p(x, y))$

Free Variables, Sentences and Closure

The free variables of a formula f are those not captured by a quantifer. Otherwise they are bound variables.

E.g. x and y are free in $(\forall y.p(x, y)) \wedge (\exists x.p(x, y))$

It's a bit confusing as x and y occur in as bound and unbound. Later we rewrite formulas to avoid this (rectification).

Free Variables, Sentences and Closure

The free variables of a formula f are those not captured by a quantifer. Otherwise they are bound variables.

E.g. x and y are free in $(\forall y.p(x, y)) \wedge (\exists x.p(x, y))$

It's a bit confusing as x and y occur in as bound and unbound. Later we rewrite formulas to avoid this (rectification).

If ϕ has free variables X we might write it $\phi[X]$.
We write $\phi[V]$ for the formula $\phi[X]$ where X is replaced by V.

Free Variables, Sentences and Closure

The free variables of a formula f are those not captured by a quantifer. Otherwise they are bound variables.

E.g. x and y are free in $(\forall y.p(x, y)) \wedge (\exists x.p(x, y))$

It's a bit confusing as x and y occur in as bound and unbound. Later we rewrite formulas to avoid this (rectification).

If ϕ has free variables X we might write it $\phi[X]$.
We write $\phi[V]$ for the formula $\phi[X]$ where X is replaced by V.

A formula is a sentence if it does not contain any free variables

The universal closure of $\phi[X]$ is $\forall X.\phi[X]$

Interpretation

(We already met this general notion in Lecture 3)

An Interpretation allows us to assign a truth value to every sentence.

Let $\langle \mathcal{D}, \mathcal{I} \rangle$ be a structure such that $\mathcal{I}$ is an interpretation over a non-empty (possibly infinite) domain $\mathcal{D}$. We often leave $\mathcal{D}$ implicit and refer directly to $\mathcal{I}$.

The map $\mathcal{I}$ maps

- Every constant symbol to an element of $\mathcal{D}$
- Every function symbol of arity n to a function in $\mathcal{D}^n \to \mathcal{D}$
- Every proposition symbol to a truth value in $\mathbb{B}$
- Every predicate symbol of arity n to a function in $\mathcal{D}^n \to \mathbb{B}$

Interpretation of Atoms

We can then lift interpretations to non-constant terms recursively as

$$\mathcal{I}(f(t_1, \ldots, t_n)) = \mathcal{I}(f)(\mathcal{I}(t_1), \ldots, \mathcal{I}(t_n))$$

(variables are ignored as we only consider ground terms)

We can lift interpretations to non-propositional atoms similarly e.g.

$$\begin{array}{rcl} \mathcal{I}(t_1 = t_2) & = & \mathcal{I}(t_1) = \mathcal{I}(t_2) \\ \mathcal{I}(p(t_1, \ldots, t_n)) & = & \mathcal{I}(p)(\mathcal{I}(t_1), \ldots, \mathcal{I}(t_n)) \end{array}$$

Every atom is interpreted as a truth value.

Interpretation of Atoms

We can then lift interpretations to non-constant terms recursively as

$$\mathcal{I}(f(t_1, \ldots, t_n)) = \mathcal{I}(f)(\mathcal{I}(t_1), \ldots, \mathcal{I}(t_n))$$

(variables are ignored as we only consider ground terms)

We can lift interpretations to non-propositional atoms similarly e.g.

$$\begin{array}{rcl} \mathcal{I}(t_1 = t_2) & = & \mathcal{I}(t_1) = \mathcal{I}(t_2) \\ \mathcal{I}(p(t_1, \ldots, t_n)) & = & \mathcal{I}(p)(\mathcal{I}(t_1), \ldots, \mathcal{I}(t_n)) \end{array}$$

Every atom is interpreted as a truth value.

This mixes three kinds of equality (be careful, I'm being lazy).

Interpreting Equality

Usually FOL is introduced without equality and then extended.

I am introducing FOL with equality directly.

Usually in the extension the set of models is restricted to normal models that interpret equality as equality and such that all domain elements are distinct with respect to equality. We will enforce this straight away here.

Interpreting Equality

Usually FOL is introduced without equality and then extended.

I am introducing FOL with equality directly.

Usually in the extension the set of models is restricted to normal models that interpret equality as equality and such that all domain elements are distinct with respect to equality. We will enforce this straight away here.

We do not *need* to make equality directly part of the language. Adding

$$\forall x. x = x \qquad \forall x, y.(x = y \rightarrow y = x) \qquad \forall x, y, z.((x = y \wedge y = z) \rightarrow x = z)$$
$$\forall x_1, \ldots x_n, y_1, \ldots y_n.((x_1 = y_1 \wedge \ldots x_n = y_n) \rightarrow f(x_1, \ldots x_n) = f(y_1, \ldots, y_n))$$
$$\forall x_1, \ldots x_n, y_1, \ldots y_n.((x_1 = y_1 \wedge \ldots x_n = y_n) \rightarrow p(x_1, \ldots x_n) \leftrightarrow p(y_1, \ldots, y_n))$$

(for all functions f and predicates p) forces $=$ to behave as above.

Interpreting Equality

Usually FOL is introduced without equality and then extended.

I am introducing FOL with equality directly.

Usually in the extension the set of models is restricted to normal models that interpret equality as equality and such that all domain elements are distinct with respect to equality. We will enforce this straight away here.

We do not *need* to make equality directly part of the language. Adding

$$\forall x.x = x \qquad \forall x, y.(x = y \rightarrow y = x) \qquad \forall x, y, z.((x = y \wedge y = z) \rightarrow x = z)$$
$$\forall x_1, \ldots x_n, y_1, \ldots y_n.((x_1 = y_1 \wedge \ldots x_n = y_n) \rightarrow f(x_1, \ldots x_n) = f(y_1, \ldots, y_n))$$
$$\forall x_1, \ldots x_n, y_1, \ldots y_n.((x_1 = y_1 \wedge \ldots x_n = y_n) \rightarrow p(x_1, \ldots x_n) \leftrightarrow p(y_1, \ldots, y_n))$$

(for all functions f and predicates p) forces $=$ to behave as above.

This general approach of restricting the interpretations of interest with respect to some theory is also used in arithmetic (see end of my part).

Interpretation of Formulas

Finally we interpret formulas

$\mathcal{I}(\mathit{true})$	is always true
$\mathcal{I}(\neg\phi)$	*iff* $\mathcal{I}(\phi)$ is not true
$\mathcal{I}(\phi_1 \land \phi_2)$	*iff* both $\mathcal{I}(\phi_1)$ and $\mathcal{I}(\phi_2)$ are true
$\mathcal{I}(\forall x.\phi[x])$	*iff* for every $d \in \mathcal{D}$ we have that $\mathcal{I}(\phi[d])$ is true

Hopefully the first 3 are straightforward. For $\forall$ we take each element d of the domain and see if the quantified formula holds if we replace x by d.

We could extend this for the derived operators but do not need to.

Big Aside

In the next slide I am going to use the syntax

$$\lambda x.\ (\textit{expression using } x)$$

to represent an anonymous function that takes something (x) as input and returns the result of evaluating the expression on that input.

This notation is borrowed from the λ-calculus, a model of computation that is central to computer science but not currently taught in our Undergraduate syllabus. I suggest you at least read the Wikipedia page at some point!

In any case, it is used to represent functions and that's all you need to know for the next slide.

A Formula Can Have Many Interpretations

$$parent(giles, mark)$$
$$man(giles)$$
$$\forall x.((man(x) \land \exists y.parent(x, y)) \rightarrow father(x))$$

We need to interpret $parent, giles, mark, man, father$. Fix $\mathcal{D} = \{1, 2\}$.

A Formula Can Have Many Interpretations

$$
\begin{array}{l}
parent(giles, mark) \\
man(giles) \\
\forall x.((man(x) \land \exists y.parent(x, y)) \rightarrow father(x))
\end{array}
$$

We need to interpret $parent, giles, mark, man, father$. Fix $\mathcal{D} = \{1, 2\}$.

Giles is Mark. Everything is true.

$$
\begin{array}{l}
\mathcal{I}(giles) = 1 \\
\mathcal{I}(mark) = 1 \\
\mathcal{I}(parent) = (\lambda x, y.(true)) \\
\mathcal{I}(man) = (\lambda x.(true)) \\
\mathcal{I}(father) = (\lambda x.(true))
\end{array}
$$

A Formula Can Have Many Interpretations

$$\begin{array}{l} parent(giles, mark) \\ man(giles), mark \neq giles \\ \forall x.((man(x) \land \exists y.parent(x,y)) \rightarrow father(x)) \end{array}$$

We need to interpret $parent, giles, mark, man, father$. Fix $\mathcal{D} = \{1, 2\}$.

Giles and Mark different. Everything is true.

$$\begin{array}{l} \mathcal{I}(giles) = 1 \\ \mathcal{I}(mark) = 2 \\ \mathcal{I}(parent) = (\lambda x, y.(true)) \\ \mathcal{I}(man) = (\lambda x.(true)) \\ \mathcal{I}(father) = (\lambda x.(true)) \end{array}$$

A Formula Can Have Many Interpretations

$$
\begin{array}{l}
parent(giles, mark) \\
man(giles), mark \neq giles \\
\forall x.((man(x) \wedge \exists y.parent(x, y)) \rightarrow father(x))
\end{array}
$$

We need to interpret $parent, giles, mark, man, father$. Fix $\mathcal{D} = \{1, 2\}$.

Giles and Mark different. Least is true.

$$
\begin{array}{l}
\mathcal{I}(giles) = 1 \\
\mathcal{I}(mark) = 2 \\
\mathcal{I}(parent) = (\lambda x, y.(x = 1 \wedge y = 2))) \\
\mathcal{I}(man) = (\lambda x.(x = 2)) \\
\mathcal{I}(father) = (\lambda x.(x = 2))
\end{array}
$$

A Formula Can Have Many Interpretations

$$\begin{array}{l} parent(giles, mark) \\ man(giles), mark \neq giles \\ \forall x.((man(x) \wedge \exists y.parent(x, y)) \rightarrow father(x)) \end{array}$$

We need to interpret $parent, giles, mark, man, father$. Fix $\mathcal{D} = \{1, 2\}$.

Giles and Mark different. A bit more is true.

$$\begin{array}{l} \mathcal{I}(giles) = 1 \\ \mathcal{I}(mark) = 2 \\ \mathcal{I}(parent) = (\lambda x, y.(x = 1 \wedge y = 2))) \\ \mathcal{I}(man) = (\lambda x.(x = 2 \vee x = 1)) \\ \mathcal{I}(father) = (\lambda x.(x = 2)) \end{array}$$

Summary

First-Order Logic has explicit universal and existential quantification and allows arbitrary boolean structure in formulas.

The semantics of a formula is defined in terms of interpretations.

A FOL formula can have many models. It can also have a single model (up to renaming of domain constants) or no models.