Introduction to First Order Logic

Syntax, Comparison to Logic4Fun, Modelling Natural Language, Natural Deduction

Ranald Clouston

Australian National University

March 17, 2025

Beyond Propositional Logic

Logic4Fun has shown that some things are awkward to express with propositional logic.

Other things, although appearing to be logically reasonable, seem impossible to express:

- All cricket fans know the rules of cricket;
- Some Australians do not know the rules of cricket;
- ∴ Some Australians are not cricket fans

This is a perennial problem: if we think parts of language or thought have logical content, but our current logics cannot express that content, we need new logics.

This example, and many more, can be expressed via first order logic, also called predicate logic.

Syntax of First Order Logic

Functions

Logic4Fun is specialised for logic puzzles and automatic model finding, so while its syntax is similar to first order logic it is not the same.

Like Logic4Fun, we do want to talk about *functions*.

Unlike Logic4Fun we will have no sorts, so the only property of a function's syntax is how many arguments it takes, which is called its arity.

- Special case: a nullary (arity 0) function is called a constant.
- Without sort `bool` we will not be able to express a predicate as a sort of function;
- We will see later that sorts, while very convenient, are not essential.

Variables and Terms

A term is a 'thing' in some universe of discourse

- We will be more precise about these universes later; for now, it is a 'collection of things we are interested in'
- Terms are either variables - arbitrary things, written x, y, z etc...
  - Do not confuse with propositional variables, which are arbitrary truth values, not things.
- ... or defined via some function(s), e.g. 'Mountback', 'the horse ridden by Mountback', 'the activity performed by the horse ridden by x'.

In Backus-Naur form:

$$t \quad := \quad x \mid f(t, \ldots, t)$$

where x is any variable, f is a function with arity n, and f has been applied to n terms.

Predicates and Signatures

A predicate is syntax that takes term(s) as input, and given such input has a truth value.

Like functions, predicates have arities.

- A nullary predicate behaves just like a propositional variable.
- unary predicates are assertions about things: '_ is a cricket fan', '_ knows the rules of cricket', '_ is Australian'
- binary predicates are assertions about pairs of things, i.e., a relation between them: '_ is married to _', '_ $=$ _'

When we do first order logic, we always work with a signature: a collection of functions and predicates, each with specified arities.

Formulas

Definition of formula:

$$\varphi \quad := \quad P(t,\ldots,t) \mid t = t \mid \bot \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \to \varphi \mid \forall x\, \varphi \mid \exists x\, \varphi$$

where t are terms, P is a predicate of arity n applied to n terms, and x is a variable.

Connective	**Name**	**Also called**
$=$	equality	identity
$\forall$	universal quantifier	for all, `ALL`
$\exists$	existential quantifier	there exists, `SOME`

$\forall x$ and $\exists x$ bind as tightly as $\neg$, so e.g. $\forall x \bot \wedge \varphi$ is $(\forall x \bot) \wedge \varphi$.
Where a function or predicate, and its subterm, are one letter, omit brackets, e.g. $P\,c$, $f\,x$.

Free and Bound Variables

In mathematics, programming, and logic we have the idea of free and bound variables.

If a mathematics book says ‘Let x be a natural number’, we read x as an arbitrary natural number throughout a certain ‘scope’. If 100 pages later we see x again we do not assume it is the same natural number, or even a natural number at all.

- The construction ‘let x be’ is said to *bind* x.
- In the sentence ‘x is odd’, x is free: it only makes sense in the context of a binder.
- The end of the scope might not be explicit; it might be e.g. the end of a section.
- The name x is essentially arbitrary; it could have been y so long as it was uniformly replaced throughout the scope.

In programming we meet the same phenomenon with local variables.

Free and Bound Variables in First Order Logic

$\forall$ and $\exists$ bind their variables. Formally we define the set of free variables of a formula:

- $FV(P(t_1, \dots, t_n))$ and $FV(t = u)$ are all variables appearing in any of the terms.
- $FV(\bot)$ is $\emptyset$
- $FV(\neg\varphi)$ is $FV(\varphi)$
- $FV(\varphi \bullet \psi)$ is $FV(\varphi) \cup FV(\psi)$ for $\bullet \in \{\wedge, \vee, \rightarrow\}$
- $FV(\heartsuit x\, \varphi)$ is $FV(\varphi) - \{x\}$ for $\heartsuit \in \{\forall, \exists\}$

A variable is bound if it appears but is not free.

Examples:

- $FV(\forall x(P\,x \wedge Q(x, y)))$
- $FV(\forall x\, P\,x \wedge Q(x, y))$

Free and Bound Variables in First Order Logic

$\forall$ and $\exists$ bind their variables. Formally we define the set of free variables of a formula:

- $FV(P(t_1, \ldots, t_n))$ and $FV(t = u)$ are all variables appearing in any of the terms.
- $FV(\bot)$ is $\emptyset$
- $FV(\neg\varphi)$ is $FV(\varphi)$
- $FV(\varphi \bullet \psi)$ is $FV(\varphi) \cup FV(\psi)$ for $\bullet \in \{\wedge, \vee, \to\}$
- $FV(\heartsuit x\, \varphi)$ is $FV(\varphi) - \{x\}$ for $\heartsuit \in \{\forall, \exists\}$

A variable is bound if it appears but is not free.

Examples:

- $FV(\forall x(P\,x \wedge Q(x, y)))$ is $\{y\}$
- $FV(\forall x\, P\,x \wedge Q(x, y))$

Free and Bound Variables in First Order Logic

$\forall$ and $\exists$ bind their variables. Formally we define the set of free variables of a formula:

- $FV(P(t_1, \ldots, t_n))$ and $FV(t = u)$ are all variables appearing in any of the terms.
- $FV(\bot)$ is $\emptyset$
- $FV(\neg\varphi)$ is $FV(\varphi)$
- $FV(\varphi \bullet \psi)$ is $FV(\varphi) \cup FV(\psi)$ for $\bullet \in \{\wedge, \vee, \rightarrow\}$
- $FV(\heartsuit x\, \varphi)$ is $FV(\varphi) - \{x\}$ for $\heartsuit \in \{\forall, \exists\}$

A variable is bound if it appears but is not free.

Examples:

- $FV(\forall x(P\,x \wedge Q(x, y)))$ is $\{y\}$
- $FV(\forall x\, P\,x \wedge Q(x, y))$ is $\{x, y\}$.
  - This formula is needlessly confusing and could be rewritten $\forall z\, P\,z \wedge Q(x, y)$.

Closed Formulas

A formula is closed if it has no free variables.

We are mostly interested in closed formulas; other formulas are in some sense incomplete.

But our closed formulas often have subformulas that are not closed

- e.g. $\forall x P\,x$ is closed, but its subformula $P\,x$ is not...
- so we must develop first order logic in a way that makes sense of free variables.

Comparison to Logic4Fun

Comparison to Logic4Fun

The languages of Logic4Fun and first order logic are similar, but there are a few conveniences (for users) and restrictions (to make automation tractable) in Logic4Fun

Logic4Fun vs First Order Logic:

- Finite and small vs potentially infinite universes of discourse.
- Sorts vs no sorts. First order logic can be extended with sorts, but it is simpler to work with the unsorted base logic. Yet sorts are so useful that we will spend a few slides talking about how to live without them.
- Potentially partial vs always total functions (and predicates).
- No built-in properties of functions like `all_different`.
- Ordered sorts vs no ordering.

During our discussion on sorts we will return to these last three issues.

Working with a Single Sort

Suppose our Logic4Fun example requires only a single sort and does not use the built-in `bool` or `natnum`.

- To be concrete, say we define it in the Sorts box as `Universe`.

Then all our terms will have sort `Universe`, which is no different from having no sorts.

- e.g. instead of defining a function as `f (Universe,Universe) : Universe`, we might as well just say that f has arity 2.

Single Sort by Enumeration

What if our single sort was defined with `enum`?

- e.g. `rider enum : Mountback, Hacking, Klamberon, Topalov.`

Put a constant (nullary function) for each of these elements into the signature.

For each pair of constants, assert they are non-equal, e.g. $\neg(\mathit{Mountback} = \mathit{Hacking})$

Finally, if necessary, declare that there is nothing else:

$$\forall x(x = \mathit{Mountback} \vee x = \mathit{Hacking} \vee x = \mathit{Klamberon} \vee x = \mathit{Topalov})$$

Single Sort by Cardinality

- e.g. `Universe cardinality = 3.`

We can use the same trick as the previous slide with constants called $0, 1, 2$.

Alternatively: observe that if there are 4 things, at least two of them must be equal:

$$\forall w \forall x \forall y \forall z (w = x \lor w = y \lor w = z \lor x = y \lor x = z \lor y = z)$$

and that there exist at least three different things:

$$\exists x \exists y \exists z (\neg(x = y) \land \neg(x = z) \land \neg(y = z))$$

These tricks allow us to define `bool` and the finite `natnum`, should we need them.

Multiple Sorts

What if we do want multiple sorts e.g. `rider`, `horse`, `activity`?

Think of our universe as a set of diverse ‘things’ and introduce the sorts to the signature as unary *predicates*, so e.g. $\mathit{rider}(x)$ is intended to mean that the thing x is a rider.

- Announce that all the constants fulfill the relevant predicate, e.g. $\mathit{rider}(\mathit{Mountback})$.
- To prevent our constants of the same sort being equal things, as before we assert e.g. $\neg(\mathit{Mountback} = \mathit{Hacking})$
- To keep our sorts disjoint, assert e.g. $\forall x(\mathit{rider}(x) \to \neg \mathit{horse}(x))$
- There is nothing else in our universe: $\forall x(\mathit{rider}(x) \vee \mathit{horse}(x) \vee \mathit{activity}(x))$.
- But how to handle quantified variables and functions, that we intuitively assign sorts to?

Quantified Variables

In Logic4Fun when we write `ALL x ...` or `SOME x ...` the variable `x` has an inferred sort.

Now that sorts are replaced by unary predicates, we add an assertion inside our statement that such a predicate holds for our variable.

- If we have $\forall x \varphi$ with intended sort S for x, use implication: $\forall x(S\,x \rightarrow \varphi)$.
- If we have $\exists x \varphi$ with intended sort S, use conjunction: $\exists x(S\,x \wedge \varphi)$.

e.g. to assert there are at most two things in a sort S, write

$$\forall x \forall y \forall z(S\,x \wedge S\,y \wedge S\,z \rightarrow x = y \vee x = z \vee x = z)$$

and to assert there are at least two things:

$$\exists x \exists y(S\,x \wedge S\,y \wedge \neg(x = y))$$

Functions

How to express sorted functions like `rides (rider) : horse`?

- Not as unary functions, because these must be defined on everything in our universe!

We use the fact that functions are a special kind of relation, and relations can be expressed by predicates

- e.g. instead of a unary function, use a binary predicate
- Assert it obeys our sorts (as predicates): $\forall x \forall y (\mathit{rides}(x, y) \rightarrow \mathit{rider}(x) \land \mathit{horse}(y))$.
- Assert everything in our domain is mapped by the function: $\forall x (\mathit{rider}(x) \rightarrow \exists y \; \mathit{rides}(x, y))$. If we omit this our function could be partial.
- Assert that it maps things uniquely: $\forall x \forall y \forall z (\mathit{rides}(x, y) \land \mathit{rides}(x, z) \rightarrow y = z)$

We could similarly make our function `all_different` or `surjective`.

Ordered Sets

In Logic4Fun all sorts are ordered. We do not always use this, so do not always need to encode it into first order logic.

But if we did, we could introduce binary predicates $<, \leq, >, \geq$

- Do we really need all four?
- What assertions of first order logic should they obey? Left as an exercise.

Modelling Natural Language with First Order Logic

Modelling Natural Language

First order logic cannot, of course, model all of natural language

- Nor even model all quantifiers: "For most x", "For finitely many x", ...
- But we hope to model a lot more language than propositional logic could.

Our discussion of moving between Logic4Fun and first order logic brought up many issues that arise with natural language also

- Start with our signature: what are our things, and what operations on, properties of, and relations between them do we wish to discuss?
- Sometimes need to express properties like specific finite size, functionality, injectivity etc.

Extended Example

Take unary predicates F 'plays football / is a footballer', G ‘is a goat’, H ‘is hairy’, and binary predicate K ‘_ kicks _’.

All goats are hairy
Some footballers are hairy
Goats do not play football
Every footballer kicks goats
Every footballer kicks a goat
Only hairy footballers kick goats

Extended Example

Take unary predicates F 'plays football / is a footballer’, G ‘is a goat’, H ‘is hairy’, and binary predicate K ‘_ kicks _’.

All goats are hairy	$\forall x(Gx \rightarrow Hx)$
Some footballers are hairy	
Goats do not play football	
Every footballer kicks goats	
Every footballer kicks a goat	
Only hairy footballers kick goats	

Extended Example

Take unary predicates F 'plays football / is a footballer', G ‘is a goat’, H ‘is hairy’, and binary predicate K ‘_ kicks _’.

All goats are hairy	$\forall x(Gx \to Hx)$
Some footballers are hairy	$\exists x(Fx \wedge Hx)$
Goats do not play football	
Every footballer kicks goats	
Every footballer kicks a goat	
Only hairy footballers kick goats	

Extended Example

Take unary predicates F 'plays football / is a footballer', G ‘is a goat’, H ‘is hairy’, and binary predicate K ‘_ kicks _’.

All goats are hairy	$\forall x(Gx \to Hx)$
Some footballers are hairy	$\exists x(Fx \land Hx)$
Goats do not play football	$\forall x(Gx \to \neg Fx)$
Every footballer kicks goats	
Every footballer kicks a goat	
Only hairy footballers kick goats	

Extended Example

Take unary predicates F 'plays football / is a footballer', G ‘is a goat', H ‘is hairy', and binary predicate K ‘_ kicks _'.

All goats are hairy	$\forall x(Gx \to Hx)$
Some footballers are hairy	$\exists x(Fx \wedge Hx)$
Goats do not play football	$\forall x(Gx \to \neg Fx)$
Every footballer kicks goats	$\forall x(Fx \to \exists y(Gy \wedge Kxy))$
Every footballer kicks a goat	
Only hairy footballers kick goats	

Extended Example

Take unary predicates F 'plays football / is a footballer', G ‘is a goat’, H ‘is hairy’, and binary predicate K ‘_ kicks _’.

All goats are hairy	$\forall x(Gx \to Hx)$
Some footballers are hairy	$\exists x(Fx \wedge Hx)$
Goats do not play football	$\forall x(Gx \to \neg Fx)$
Every footballer kicks goats	$\forall x(Fx \to \exists y(Gy \wedge Kxy))$
Every footballer kicks a goat	As above, or $\exists y(Gy \wedge \forall x(Fx \to Kxy))$?
Only hairy footballers kick goats	

Extended Example

Take unary predicates F 'plays football / is a footballer', G 'is a goat', H 'is hairy', and binary predicate K '_ kicks _'.

All goats are hairy	$\forall x(Gx \rightarrow Hx)$
Some footballers are hairy	$\exists x(Fx \land Hx)$
Goats do not play football	$\forall x(Gx \rightarrow \neg Fx)$
Every footballer kicks goats	$\forall x(Fx \rightarrow \exists y(Gy \land Kxy))$
Every footballer kicks a goat	As above, or $\exists y(Gy \land \forall x(Fx \rightarrow Kxy))$?
Only hairy footballers kick goats	$\forall x(\exists y(Gy \land Kxy) \rightarrow Fx \land Hx)$ or $\forall x(Fx \land \exists y(Gy \land Kxy) \rightarrow Hx)$?

Ambiguity

The last two examples were ambiguous: they could be translated different ways and (in fact) those ways are not logically equivalent.

First order logic has its limitations and flaws, but this is *not* one of them.

Rather, this is a *feature*! We identify potentially dangerous ambiguities in language, e.g. in program specifications or laws, by attempting to formalise them.

Natural Deduction for First Order Logic

Natural Deduction and Predicates

Recall that natural deduction is structured around introduction and elimination rules.

We will not have such rules for atomic predicates $P(t_1, \ldots, t_n)$.

- To prove such a predicate we must use our assumptions, or eliminate other connectives.

However we will have such rules for the very special predicate $=$, which is why it is a basic part of the logic instead of being yet another predicate we can try to define ourselves.

Equality Introduction

Everything is equal to itself:

$$\frac{}{\vdash\ t = t} =I$$

This is our second axiom, after Assumption, so when it appears in our proof we do not cite any previous line.

Equality Elimination

If a formula involving some term t is true, and $t = u$, then the formula with some or all instances of t replaced by u is true:

$$\frac{\Gamma \vdash t = u \quad \Gamma' \vdash \varphi[t/x]}{\Gamma, \Gamma' \vdash \varphi[u/x]} = E$$

This requires us to formally define substitution.

Substitution into Terms

Substitution is the replacement of all appearances of a given variable by a given term.

Substitution inside a term is defined by an easy induction:

- $x[t/x]$ is t
- $y[t/x]$ is y
- $f(t_1, \ldots, t_n)[t/x]$ is $f(t_1[t/x], \ldots, t_n[t/x])$

This is just find-and-replace: every instance of x inside a given term is replaced by t.

- Special case: there is no x in the given term. Then substitution does nothing.

Substitution into Formulas (No Quantifiers)

If we ignore quantifiers, substitution inside a formula is also find-and-replace:

- $P(t_1, \ldots, t_n)[t/x]$ is $P(t_1[t/x], \ldots, t_n[t/x])$
- $(u = u')[t/x]$ is $u[t/x] = u'[t/x]$
- $\bot[t/x]$ is $\bot$
- $(\neg\varphi)[t/x]$ is $\neg(\varphi[t/x])$
- $(\varphi \bullet \psi)[t/x]$ is $(\varphi[t/x]) \bullet (\psi[t/x])$ for $\bullet \in \{\wedge, \vee, \to\}$

Note that this definition uses the definition of substitution into terms.

We will return to the question of substitution in the presence of quantifiers.

Equality Elimination

$$\frac{\Gamma \vdash t = u \quad \Gamma' \vdash \varphi[t/x]}{\Gamma, \Gamma' \vdash \varphi[u/x]} =E$$

The definition of substitution from the last slides allows us to prove a basic example on the whiteboard:

$$t = u \vdash u = t$$

Note: another way to think about substitution is to ignore the variable in the definition, and just replace some (or all) occurrences of some term with another term that is equal to it.

Existential Introduction

If we know something about a specific term, we know that property holds for some term:

$$\frac{\Gamma \vdash \varphi[t/x]}{\Gamma \vdash \exists x \varphi}\ \exists I$$

Universal Elimination

Universal elimination is equally easy to state:

$$\frac{\Gamma \vdash \forall x \varphi}{\Gamma \vdash \varphi[t/x]} \; \forall E$$

Whiteboard proof:

$$\forall x(Fx \to Gx) \vdash \forall x Fx \to \exists x Gx$$

(This example requires $\exists I$ also):

$$\frac{\Gamma \vdash \varphi[t/x]}{\Gamma \vdash \exists x \varphi} \; \exists I$$

Substitution with Quantifiers

Substitution as simple find-and-replace is *not* correct in the presence of quantifiers.

Suppose we have the perfectly reasonable property $\forall x \exists y (x < y)$ and decide to apply $\forall E$.

- We can choose any term to replace x. Let's choose y. Then $\exists y (y < y)$???

Problem: a clash between the term y we chose, and a variable name appearing bound in the formula we are replacing inside.

Solution: change the bound name y in our original formula to z, to get $\forall x \exists z (x < z)$.

- Are you convinced that this does not change the meaning of our formula at all?
- Let's choose y to replace x again. Then $\exists z (y < z)$
- Health warning: never change a free name. Only bound names can be changed like this.

Substitution with Quantifiers

To our earlier almost complete definition of substitution inside a formula we add

- $(\heartsuit y \varphi)[t/x]$ is $\heartsuit y(\varphi[t/x])$ for $\heartsuit \in \{\forall, \exists\}$, so long as x and y are different and y is not a free variable in t

This looks like a partial definition - it only works if two conditions hold - but in fact it is total if we adopt the policy:

- if either condition fails, choose a variable z not used anywhere in φ, x, or t, then rename all occurences of y in $\heartsuit y \varphi$ to z, then proceed with the substitution.

Universal Introduction

If we know something about a completely arbitrary variable, then we know it about anything.

$$a \notin FV(\Gamma, \varphi) : \ \frac{\Gamma \vdash \varphi[a/x]}{\Gamma \vdash \forall x \varphi} \forall I$$

There is a rough convention of choosing variable names from the start of the alphabet to mean 'variable name which is arbitrary in the current context'. Such variables are sometimes called eigenvariables.

Whiteboard example:

$$\forall x Fx, \forall x Gx \vdash \forall x(Fx \wedge Gx)$$

Existential Elimination

How can we use an existential? Recall $\vee E$: if we know a disjunction without knowing which side holds, we show that either side gets us to the same conclusion.

With $\exists$ we might have no idea which term our property holds of, so we show that the same conclusion follows from the assumption of the property on an arbitrary eigenvariable:

$$a \notin FV(\Gamma, \varphi, \Gamma', \psi) : \ \frac{\Gamma \vdash \exists x \varphi \quad \Gamma', \varphi[a/x] \vdash \psi}{\Gamma, \Gamma' \vdash \psi} \exists E$$

From Universal to Existential

An interesting sequent with a short proof: $\forall x \varphi \vdash \exists x \varphi$.

Sanity check: confirm we cannot prove the other direction.

The theorem says that if everything has a certain property, then something has that property.

- This only makes sense if our universe of discourse is not empty.
- First order logic can be modified to have a possibly empty domain of discourse; this is called free logic. We will not pursue this in this course.
- We can have possibly empty 'sorts' (unary predicates): there is no general proof that $\exists x Sx$.

The Classical Relationship between Universal and Existential

Just as $\wedge$ and $\vee$ are related by the De Morgan laws, which require classical proof, we have

- $\exists x \neg\varphi \vdash \neg\forall x\varphi$ and $\neg\forall x\varphi \vdash \exists x\neg\varphi$
- $\forall x\neg\varphi \vdash \neg\exists x\varphi$ and $\neg\exists x\varphi \vdash \forall x\neg\varphi$

Three of these sequents follow by intuitionistically acceptable proofs; let's do the first one.

But the sequent $\neg\forall x\varphi \vdash \exists x\neg\varphi$ requires $\neg\neg E$.

- Hint: first prove $\neg(\exists x\neg\varphi) \vdash \forall x\varphi$, which also required $\neg\neg E$