

Classical first-order predicate logic

This is a powerful extension of propositional logic. It is the most important logic of all.

In the remaining lectures, we will:

- explain predicate logic syntax and semantics carefully
- do English–predicate logic translation, and see examples from computing
- generalise *arguments* and *validity* from propositional logic to predicate logic
- consider ways of establishing validity in predicate logic:
 - truth tables — they don't work
 - direct argument — very useful
 - equivalences — also useful
 - natural deduction (sorry).

106

6. Syntax of predicate logic

6.1 New atomic formulas

Up to now, we have regarded phrases such as *the computer is a Sun* and *Frank bought grapes* as atomic, without internal structure.

Now we look inside them.

We regard being a Sun as a *property* or *attribute* that a computer (and other things) may or may not have. So we introduce:

- A *relation symbol* (or *predicate symbol*) *Sun*.
It takes 1 argument — we say it is *unary* or its 'arity' is 1.
- We can also introduce a relation symbol *bought*.
It takes 2 arguments — we say it is *binary*, or its arity is 2.
- *Constants*, to name objects.
Eg, Heron, Frank, Room-308, grapes.

Then *Sun(Heron)* and *bought(Frank,grapes)* are two new atomic formulas.

108

Why?

Propositional logic is quite nice, but not very expressive. Statements like

- the list is ordered
- every worker has a boss
- there is someone worse off than you

need something more than propositional logic to express.

Propositional logic can't express arguments like this one of De Morgan:

- A horse is an animal.
- Therefore, the head of a horse is the head of an animal.

107

6.2 Quantifiers

So what? You may think that writing

bought(Frank,grapes)

is not much more exciting than what we did in propositional logic — writing

Frank bought grapes.

But predicate logic has machinery to vary the arguments to *bought*.

This allows us to express properties of the relation 'bought'.

The machinery is called *quantifiers*.

109

What are quantifiers?

A quantifier specifies a quantity (of things that have some property).

Examples

- *All* students work hard.
- *Some* students are asleep.
- *Most* lecturers are lazy.
- *Eight out of ten* cats prefer it.
- *Noone* is stupider than me.
- *At least six* students are awake.
- *There are infinitely many* prime numbers.
- *There are more* PCs than there are Macs.

110

6.3 Variables

We will use *variables* to do quantification. We fix an infinite collection (set) V of variables: eg, $x, y, z, u, v, w, x_0, x_1, x_2, \dots$

Sometimes I write x or y to mean ‘any variable’.

As well as formulas like $\text{Sun}(\text{Heron})$, we’ll write ones like $\text{Sun}(x)$.

- Now, to say ‘Everything is a Sun’, we’ll write $\forall x \text{Sun}(x)$.
This is read as: ‘For all x , x is a Sun’.
- ‘Something is a Sun’, can be written $\exists x \text{Sun}(x)$.
‘There exists x such that x is a Sun.’
- ‘Frank bought a Sun’, can be written

$$\exists x(\text{Sun}(x) \wedge \text{bought}(\text{Frank}, x)).$$

‘There is an x such that x is a Sun and Frank bought x .’

Or: ‘For some x , x is a Sun and Frank bought x .’

See how the new internal structure of atoms is used.

We will now make all of this precise.

112

Quantifiers in predicate logic

There are just two:

- \forall (or (A)): ‘for all’
- \exists (or (E)): ‘there exists’ (or ‘some’)

Some other quantifiers can be expressed with these. (They can also express each other.) But quantifiers like *infinitely many* and *more than* cannot be expressed in first-order logic in general. (They can in, e.g., second-order logic.)

How do they work?

We’ve seen expressions like Heron, Frank, etc. These are *constants*, like π , or e .

To express ‘All computers are Suns’ we need *variables* that can range over all computers, not just Heron, Texel, etc.

111

6.4 Signatures

Definition 6.1 (signature) A *signature* is a collection (set) of constants, and relation symbols with specified arities.

Some call it a *similarity type*, or *vocabulary*, or (loosely) *language*.

It replaces the collection of atoms we had in propositional logic.

We usually write L to denote a signature. We often write c, d, \dots for constants, and P, Q, R, S, \dots for relation symbols.

113

A simple signature

Which symbols we put in L depends on what we want to say.

For illustration, we'll use a handy signature L consisting of:

- constants Frank, Susan, Tony, Heron, Texe1, Clyde, Room-308, and c
- unary relation symbols Sun, human, lecturer (arity 1)
- a binary relation symbol bought (arity 2).

Warning: things in L are just symbols — syntax. They don't come with any meaning. To give them meaning, we'll need to work out (later) what a *situation* in predicate logic should be.

114

6.6 Formulas of first-order logic

Definition 6.3 (formula) Fix L as before.

1. If R is an n -ary relation symbol in L , and t_1, \dots, t_n are L -terms, then $R(t_1, \dots, t_n)$ is an atomic L -formula.
2. If t, t' are L -terms then $t = t'$ is an atomic L -formula. (Equality — very useful!)
3. \top, \perp are atomic L -formulas.
4. If A, B are L -formulas then so are $(\neg A)$, $(A \wedge B)$ $(A \vee B)$, $(A \rightarrow B)$, and $(A \leftrightarrow B)$.
5. If A is an L -formula and x a variable, then $(\forall x A)$ and $(\exists x A)$ are L -formulas.
6. Nothing else is an L -formula.

Binding conventions: as for propositional logic, plus: $\forall x, \exists x$ have same strength as \neg .

116

6.5 Terms

Definition 6.2 (term) Fix a signature L .

1. Any constant in L is an L -term.
2. Any variable is an L -term.
3. Nothing else is an L -term.

A **closed term** or (as computer people say) **ground term** is one that doesn't involve a variable.

Examples of terms

Frank, Heron (ground terms)

x, y, x_{56} (not ground terms)

Terms are for *naming objects*.

Terms are not true or false.

Later (§9), we'll throw in function symbols.

115

Examples of formulas

Below, we write them as the cognoscenti do. Use binding conventions to disambiguate.

- $\text{bought}(\text{Frank}, x)$
We read this as: 'Frank bought x .'
- $\exists x \text{bought}(\text{Frank}, x)$
'Frank bought something.'
- $\forall x(\text{lecturer}(x) \rightarrow \text{human}(x))$
'Every lecturer is human.' [Important eg!]
- $\forall x(\text{bought}(\text{Tony}, x) \rightarrow \text{Sun}(x))$
'Everything Tony bought is a Sun.'

117

More examples

- $\forall x(\text{bought}(\text{Tony}, x) \rightarrow \text{bought}(\text{Susan}, x))$
‘Susan bought everything that Tony bought.’
- $\forall x \text{bought}(\text{Tony}, x) \rightarrow \forall x \text{bought}(\text{Susan}, x)$
‘If Tony bought everything, so did Susan.’ Note the difference!
- $\forall x \exists y \text{bought}(x, y)$
‘Everything bought something.’
- $\exists x \forall y \text{bought}(x, y)$
‘Something bought everything.’

You can see that predicate logic is rather powerful — and terse.

118

Example of a structure

Below is a diagram of a particular L -structure, called M (say). There are 12 objects (the 12 dots) in the domain of M . Some are labelled (eg ‘Frank’) to show the meanings of the constants of L (eg Frank). The interpretations (meanings) of Sun , human are drawn as regions. The interpretation of lecturer is indicated by the black dots. The interpretation of bought is shown by the arrows between objects.

120

7. Semantics of predicate logic

7.1 Structures (situations in predicate logic)

Definition 7.1 (structure) Let L be a signature. An L -structure (or sometimes (loosely) a model) M is a thing that

- identifies a non-empty collection (set) of objects (the domain or universe of M , written $\text{dom}(M)$),
- specifies what the symbols of L mean in terms of these objects.

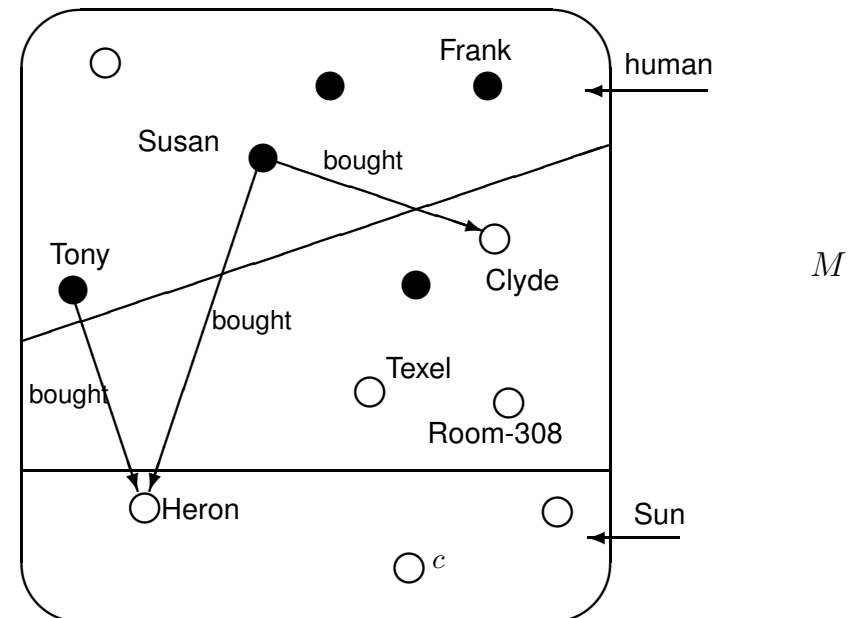
The interpretation in M of a constant is an *object in $\text{dom}(M)$* . The interpretation in M of a relation symbol is a *relation on $\text{dom}(M)$* . You will soon see relations in Discrete Mathematics I, course 142.

For our handy L , an L -structure should say:

- which objects are in its domain
- which of its objects are Tony, Susan, ...
- which objects are Suns, lecturers, human
- which objects bought which.

119

The structure M



121

Tony or Tony?

Do not confuse the object ● marked 'Tony' in $dom(M)$ with the constant Tony in L .

(I use different fonts, to try to help.)

They are quite different things. Tony is syntactic, while ● is semantic. In the context of M , Tony is a *name* for the object ● marked 'Tony'.

The following notation helps to clarify:

Notation 7.2 Let M be an L -structure and c a constant in L . We write c^M for the interpretation of c in M . It is the object in $dom(M)$ that c names in M .

So $Tony^M =$ the object ● marked 'Tony'.

In a different structure, Tony may name (mean) something else.

The meaning of a constant c *IS* the object c^M assigned to it by a structure M . A constant (and any symbol of L) has as many meanings as there are L -structures.

122

7.2 Truth in a structure (a rough guide)

When is a formula true in a structure?

- $Sun(Heron)$ is true in M , because $Heron^M$ is an object ○ that M says is a Sun.

We write this as $M \models Sun(Heron)$.

Can read as ' M says $Sun(Heron)$ '.

Warning: This is a quite different use of \models from definition 3.1. ' \models ' is *overloaded*.

- Similarly, $bought(Susan, Clyde)$ is true in M .
In symbols, $M \models bought(Susan, Clyde)$.
- $bought(Susan, Susan)$ is false in M , because M does not say that the constant Susan names an object ● that bought itself.
In symbols, $M \not\models bought(Susan, Susan)$.

From our knowledge of propositional logic,

- $M \models \neg human(Room-308)$,
- $M \not\models Sun(Tony) \vee bought(Frank, Clyde)$.

124

Drawing other symbols

Our signature L has only constants and unary and binary relation symbols.

For this L , we drew an L -structure M by

- drawing a collection of objects (the domain of M)
- marking which objects are named by which constants in M
- marking which objects M says satisfy the unary relation symbols (human, etc)
- drawing arrows between the objects that M says satisfy the binary relation symbols. The arrow direction matters.

If there were several binary relation symbols in L , we'd have to label the arrows.

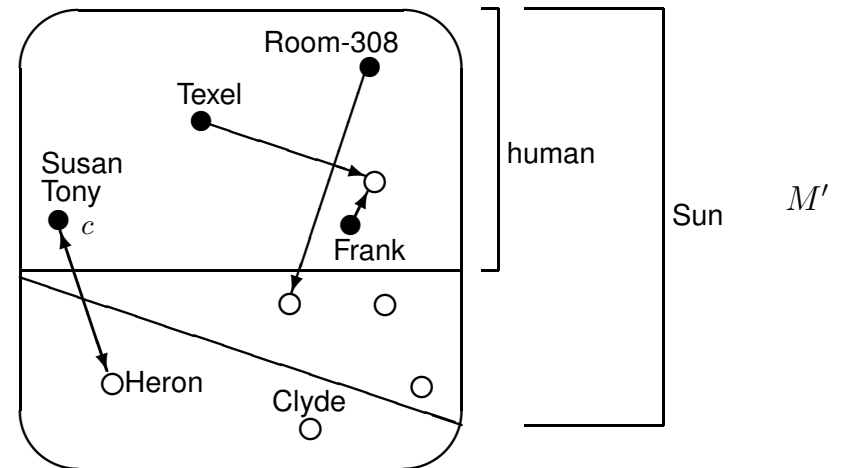
In general, there's no easy way to draw interpretations of 3-ary or higher-arity relation symbols.

0-ary (nullary) relation symbols are the same as propositional atoms.

123

Another structure

Here's another L -structure, called M' .



Now, there are only 10 objects in $dom(M')$.

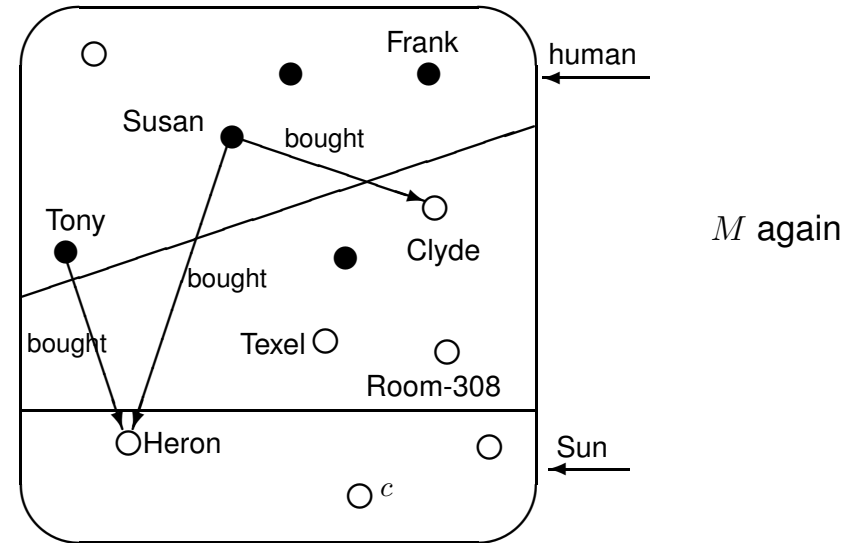
125

How do we work out if a formula with quantifiers is true in a structure?

Some statements about M'

- $M' \not\models \text{bought}(\text{Susan}, \text{Clyde})$ this time.
- $M' \models \text{Susan} = \text{Tony}$.
- $M' \models \text{human}(\text{Texel}) \wedge \text{Sun}(\text{Texel})$.
- $M' \models \text{bought}(\text{Tony}, \text{Heron}) \wedge \text{bought}(\text{Heron}, c)$.

How about $\text{bought}(\text{Susan}, \text{Clyde}) \rightarrow \text{human}(\text{Clyde})$?
 Or $\text{bought}(c, \text{Heron}) \rightarrow \text{Sun}(\text{Clyde}) \vee \neg \text{human}(\text{Texel})$?



Another example: $M \models \forall x(\text{bought}(\text{Tony}, x) \rightarrow \text{bought}(\text{Susan}, x))$

That is, 'for every object x in $\text{dom}(M)$,
 $\text{bought}(\text{Tony}, x) \rightarrow \text{bought}(\text{Susan}, x)$ is true in M' .
 (We evaluate ' \rightarrow ' as in propositional logic.)

In M , there are 12 possible x . We need to check whether
 $\text{bought}(\text{Tony}, x) \rightarrow \text{bought}(\text{Susan}, x)$ is true in M for each of them.
 $\text{bought}(\text{Tony}, x) \rightarrow \text{bought}(\text{Susan}, x)$ will be true in M for any object
 x such that $\text{bought}(\text{Tony}, x)$ is false in M . ('False \rightarrow anything is
 true.') So we only need check the x for which $\text{bought}(\text{Tony}, x)$ is true.

The effect of ' $\text{bought}(\text{Tony}, x) \rightarrow$ ' is to restrict the $\forall x$ to those x that
 Tony bought — here, just Heron^M .

For this object O , $\text{bought}(\text{Susan}, \text{O})$ is true in M . So
 $\text{bought}(\text{Tony}, \text{O}) \rightarrow \text{bought}(\text{Susan}, \text{O})$ is true in M .

So $\text{bought}(\text{Tony}, x) \rightarrow \text{bought}(\text{Susan}, x)$ is true in M for *every* object
 x in M . Hence, $M \models \forall x(\text{bought}(\text{Tony}, x) \rightarrow \text{bought}(\text{Susan}, x))$.

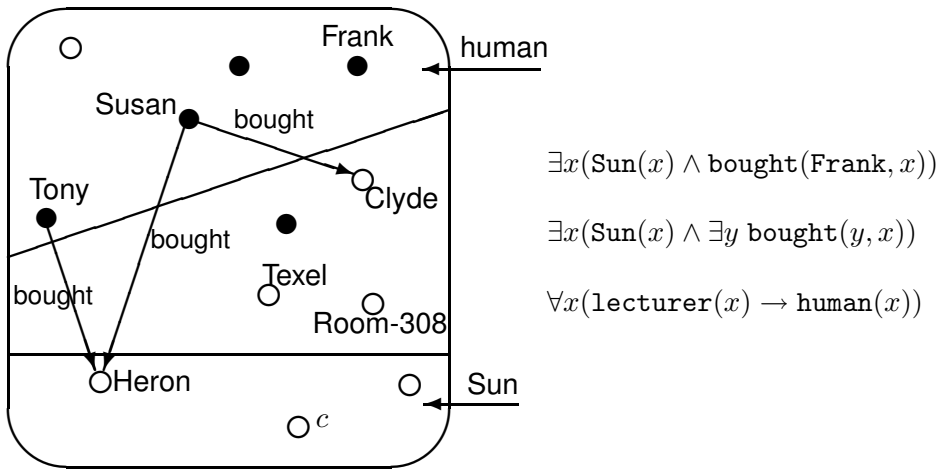
Evaluating quantifiers

$\exists x \text{bought}(x, \text{Heron})$ is true in M .
 In symbols, $M \models \exists x \text{bought}(x, \text{Heron})$.
 In English, 'something bought Heron'.

For this to be so, there must be an object x in $\text{dom}(M)$ such that
 $M \models \text{bought}(x, \text{Heron})$ — that is, M says that $\text{bought}(x, \text{O})$, where
 $\text{O} = \text{Heron}^M$.

There is: we can take (eg.) x to be Tony^M .

Exercise: which are true in M ?



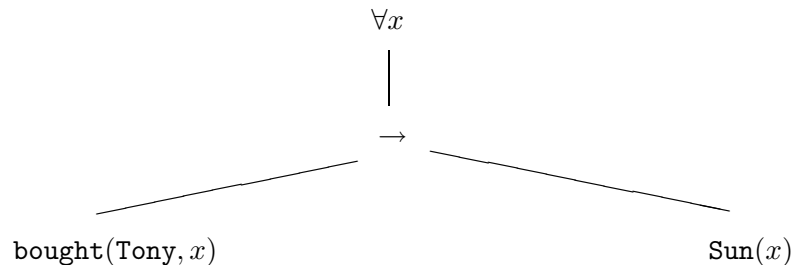
- $\exists x(\text{Sun}(x) \wedge \text{bought}(\text{Frank}, x))$
- $\exists x(\text{Sun}(x) \wedge \exists y \text{bought}(y, x))$
- $\forall x(\text{lecturer}(x) \rightarrow \text{human}(x))$

130

Example

$\forall x(\text{bought}(\text{Tony}, x) \rightarrow \text{Sun}(x))$ is true in M (see slide 130).

Its formation tree is:



- Is $\text{bought}(\text{Tony}, x)$ true in M ?!
- Is $\text{Sun}(x)$ true in M ?!

132

7.3 Truth in a structure — formally!

We saw how to evaluate some formulas in a structure. Now we show how to evaluate arbitrary formulas.

In propositional logic, we calculated the truth value of a formula in a situation by working up through its formation tree — from the atomic subformulas (leaves) up to the root.

For predicate logic, this is not so easy.

Not all formulas of predicate logic are true or false in a structure!

131

Free and bound variables

What's going on?

We'd better investigate how variables can arise in formulas.

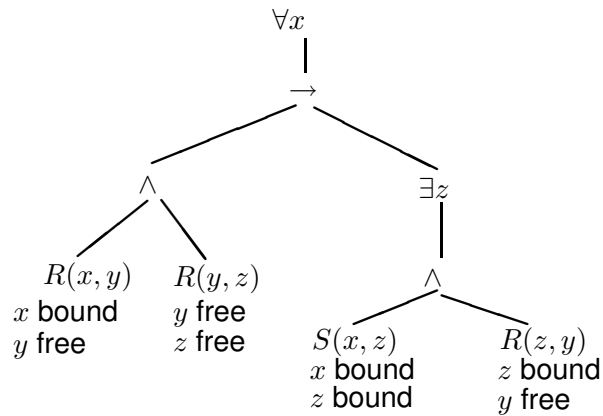
Definition 7.3 Let A be a formula.

1. An occurrence of a variable x in an atomic subformula of A is said to be **bound** if it lies under a quantifier $\forall x$ or $\exists x$ in the formation tree of A .
2. If not, the occurrence is said to be **free**.
3. The **free variables of A** are those variables with free occurrences in A .

133

Example

$$\forall x(R(x, y) \wedge R(y, z) \rightarrow \exists z(S(x, z) \wedge R(z, y)))$$



The free variables of the formula are y, z .

Note: z has both free and bound occurrences.

134

The problem

Sentences are true or false in a structure.

But non-sentences are not!

A formula with free variables is neither true nor false in a structure M , because the free variables have no meaning in M . It's like asking 'is $x = 7$ true?'

We get stuck trying to evaluate a predicate formula in a structure in the same way as a propositional one, because the structure does not fix the meanings of variables that occur free. They are *variables*, after all.

Getting round the problem

So we must specify values for free variables, before evaluating a formula to true or false.

This is so even if it turns out that the values do not affect the answer (like $x = x$).

136

Sentences

Definition 7.4 (sentence) A *sentence* is a formula with no free variables.

Examples

- $\forall x(\text{bought}(\text{Tony}, x) \rightarrow \text{Sun}(x))$ is a sentence.
- Its subformulas

$$\begin{aligned} &\text{bought}(\text{Tony}, x) \rightarrow \text{Sun}(x), \\ &\text{bought}(\text{Tony}, x), \\ &\text{Sun}(x) \end{aligned}$$

are not sentences.

Which are sentences?

- $\text{bought}(\text{Frank}, \text{Texas})$
- $\text{bought}(\text{Susan}, x)$
- $x = x$
- $\forall x(x = y \rightarrow \exists y(y = x))$
- $\forall x \forall y(x = y \rightarrow \forall z(R(x, z) \rightarrow R(y, z)))$

135

Assignments to variables

An *assignment* supplies the missing values of variables.

What a structure does for constants, an assignment does for variables.

Definition 7.5 (assignment) Let M be a structure. An *assignment* (or 'valuation') into M is something that allocates an object in $\text{dom}(M)$ to each variable.

For an assignment h and a variable x , we write $h(x)$ for the object assigned to x by h .

[Formally, $h : V \rightarrow \text{dom}(M)$ is a function.]

Given an L -structure M plus an assignment h into M , we can evaluate:

- any L -term, to an object in $\text{dom}(M)$,
- any L -formula, to true or false.

137

Evaluating terms (easy!)

Definition 7.6 (value of term) Let L be a signature, M an L -structure, h an assignment into M , and t an L -term.

The *value of t in M under h* is the object in M allocated to it by:

- M (if t is a constant) — that is, t^M ,
- h (if t is a variable) — that is, $h(t)$.

138

Semantics of atomic formulas

Fix an L -structure M and an assignment h . We define truth of a formula in M under h by working up the formation tree, as before.

Notation 7.7 (\models) We write $M, h \models A$ if A is true in M under h , and $M, h \not\models A$ if not.

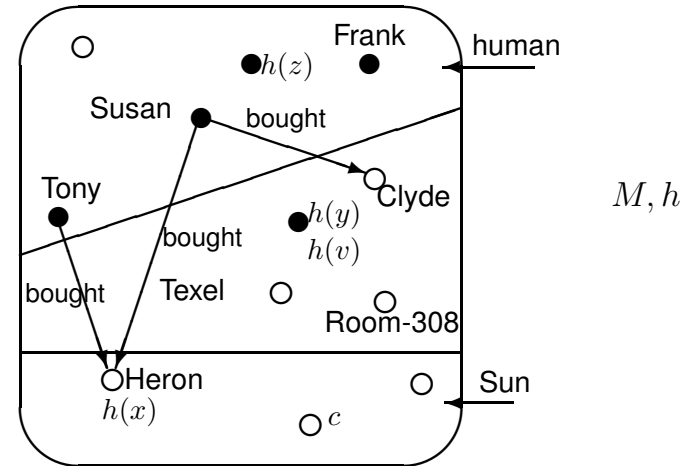
Definition 7.8 (truth in M under h)

1. Let R be an n -ary relation symbol in L , and t_1, \dots, t_n be L -terms. Suppose that the value of t_i in M under h is a_i , for each $i = 1, \dots, n$ (see definition 7.6).
 $M, h \models R(t_1, \dots, t_n)$ if M says that the sequence (a_1, \dots, a_n) is in the relation R .
 If not, then $M, h \not\models R(t_1, \dots, t_n)$.
2. If t, t' are terms, then $M, h \models t = t'$ if t and t' have the same value in M under h .
 If they don't, then $M, h \not\models t = t'$.
3. $M, h \models \top$, and $M, h \not\models \perp$.

140

Example

- (1) The value in M under h (below) of the term Tony is the object \bullet marked 'Tony'. (From now on, I usually write just 'Tony' (or Tony^M , but NOT Tony) for it.)
- (2) The value in M under h of x is Heron.



139

Semantics of non-atomic formulas (definition 7.8 ctd.)

If we have already evaluated formulas A, B in M under h , then

4. $M, h \models A \wedge B$ if $M, h \models A$ and $M, h \models B$.
 Otherwise, $M, h \not\models A \wedge B$.
5. $\neg A, A \vee B, A \rightarrow B, A \leftrightarrow B$
 — similar: just as in propositional logic.

If x is any variable, then

6. $M, h \models \exists x A$ if there is some assignment g that agrees with h on all variables except possibly x , and such that $M, g \models A$.
 If not, then $M, h \not\models \exists x A$.
7. $M, h \models \forall x A$ if $M, g \models A$ for every assignment g that agrees with h on all variables except possibly x .
 If not, then $M, h \not\models \forall x A$.

' g agrees with h on all variables except possibly x ' means that $g(y) = h(y)$ for all variables y other than x . (Maybe $g(x) = h(x)$ too!)

141

7.4 Useful notation for free variables

The books often write things like

‘Let $A(x_1, \dots, x_n)$ be a formula.’

This indicates that the free variables of A are among x_1, \dots, x_n .

Note: x_1, \dots, x_n should all be different. And not all of them need actually occur free in A .

Example: if C is the formula

$$\forall x(R(x, y) \rightarrow \exists yS(y, z)),$$

we could write it as

- $C(y, z)$
- $C(x, z, v, y)$
- C (if we're not using the useful notation)

but not as $C(x)$.

142

Working out \models in this notation

Suppose we have an L -structure M , an L -formula $A(x, y_1, \dots, y_n)$, and objects a_1, \dots, a_n in $\text{dom}(M)$.

- To establish that $M \models (\forall xA)(a_1, \dots, a_n)$ you check that $M \models A(b, a_1, \dots, a_n)$ for each object b in $\text{dom}(M)$.
You have to check even those b with no constants naming them in M . ‘Not just Frank, Texel, ..., but all the other \circ and \bullet too.’
- To establish $M \models (\exists xA)(a_1, \dots, a_n)$, you try to find some object b in the domain of M such that $M \models A(b, a_1, \dots, a_n)$.

A is simpler than $\forall xA$ or $\exists xA$. So you can recursively work out if $M \models A(b, a_1, \dots, a_n)$, in the same way. The process terminates.

144

Notation for assignments

Fact 7.9 For any formula A , whether or not $M, h \models A$ only depends on $h(x)$ for those variables x that occur free in A .

- So for a formula $A(x_1, \dots, x_n)$, if $h(x_i) = a_i$ (each i), it's OK to write $M \models A(a_1, \dots, a_n)$ instead of $M, h \models A$.
- Suppose we are explicitly given a formula C , such as

$$\forall x(R(x, y) \rightarrow \exists yS(y, z)).$$

If $h(y) = a, h(z) = b$, say, we can write

$$M \models \forall x(R(x, a) \rightarrow \exists yS(y, b))$$

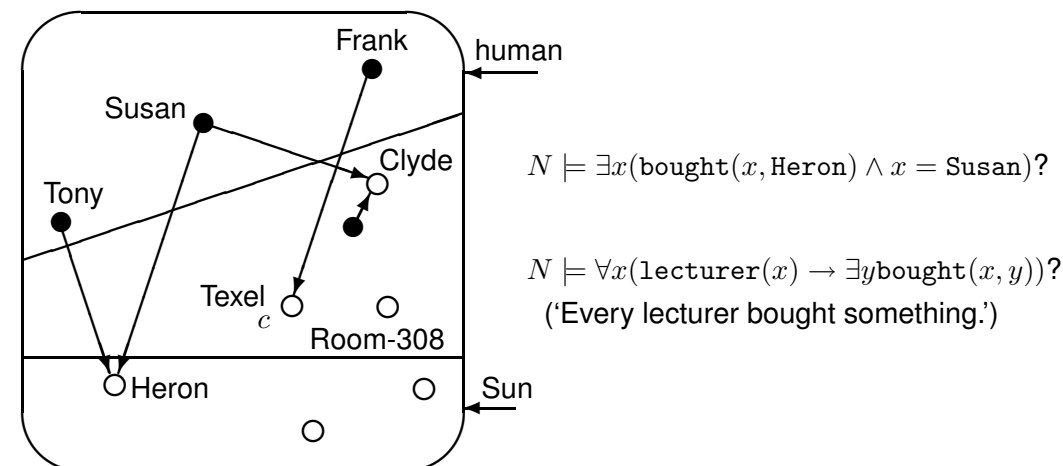
instead of $M, h \models C$. Note: only the *free* occurrences of y in C are replaced by a . The bound y is unchanged.

- For a sentence S , we can just write $M \models S$, because by fact 7.9, whether $M, h \models S$ does not depend on h at all.

143

7.5 Evaluating formulas in practice

Now we do some examples of evaluation. Let's have a new L -structure, say N . The black dots are the lecturers. The arrows indicate the interpretation in N of the relation symbol bought.



145

1. $N \models \exists x(\text{bought}(x, \text{Heron}) \wedge x = \text{Susan})?$

By definition 7.8, the answer is ‘yes’ just in case there is an object b in $\text{dom}(N)$ such that $N \models \text{bought}(b, \text{Heron}) \wedge b = \text{Susan}$.

We can find out by tabulating the results for each b in $\text{dom}(N)$. Write 1 for true, 0 for false.

$N \models \text{bought}(b, \text{Heron}) \wedge b = \text{Susan}$ if (and only if) we can find a b that makes the rightmost column a 1.

b	$\text{bought}(b, \text{Heron})$	$b = \text{Susan}$	both
Tony	1	0	0
Susan	1	1	1
Frank	0	0	0
other ●	0	0	0
Room 308	0	0	0
⋮	0	0	0

2. $N \models \forall x(\text{lecturer}(x) \rightarrow \exists y \text{bought}(x, y))?$

$N \models \forall x(\text{lecturer}(x) \rightarrow \exists y \text{bought}(x, y))$

if and only if $N \models \text{lecturer}(b) \rightarrow \exists y \text{bought}(b, y)$ for all b in $\text{dom}(N)$,
 if and only if for each b in $\text{dom}(N)$, if $N \models \text{lecturer}(b)$ then there is d in $\text{dom}(N)$ such that $N \models \text{bought}(b, d)$.

So $N \models \forall x(\text{lecturer}(x) \rightarrow \exists y \text{bought}(x, y))$ if (and only if) for each b with $\text{lecturer}(b) = 1$ in the table below, we can find a d with $\text{bought}(b, d) = 1$.

But hang on, we only need *one* b making the RH column true. We already got one: Susan.

So yes, $N \models \exists x(\text{bought}(x, \text{Heron}) \wedge x = \text{Susan})$.

A bit of thought would have shown the only b with a chance is Susan. This would have shortened the work.

Moral: read the formula first!

$N \models \forall x(\text{lecturer}(x) \rightarrow \exists y \text{bought}(x, y))$ ctd

b	sample d	$\text{lecturer}(b)$	$\text{bought}(b, d)$
Tony	Heron	1	1
Susan	Tony	1	0
Frank	Clyde	1	0
other ●	Clyde	1	1
Heron	Susan	0	0
Room 308	Tony	0	0
⋮	⋮	0	?

$$N \models \forall x(\text{lecturer}(x) \rightarrow \exists y \text{ bought}(x, y)) \text{ ctd}$$

Reduced table with b just the lecturers:

b	sample d	$L(b)$	$B(b, d)$
Tony	Heron	1	1
Susan	Tony	1	0
Frank	Clyde	1	0
other ●	Clyde	1	1

We don't have 1s all down the RH column.

Does this mean $N \not\models \forall x(\text{lecturer}(x) \rightarrow \exists y \text{ bought}(x, y))$?

Advice

How do we choose the 'right' d in examples like this? We have the following options:

- In $\exists x$ or $\forall x$ cases, tabulate all possible values of d .
In $\forall x \exists y$ cases, etc, tabulate all d for each b : that is, tabulate all pairs (b, d) .
(Very boring; no room to do it above.)
- Try to see what properties d should have (above: being bought by b). Translating into English (see later on) should help. Then go for d with these properties.
- Guess a few d and see what goes wrong. This may lead you to (2).
- Use games. . . coming next.

$$N \models \forall x(\text{lecturer}(x) \rightarrow \exists y \text{ bought}(x, y)) \text{ ctd}$$

Not necessarily.

We might have chosen bad d s for Susan and Frank. $L(b)$ is true for them, so we must try to choose a d that b bought, so that $B(b, d)$ will be true.

And indeed, we can:

b	d	$L(b)$	$B(b, d)$
Tony	Heron	1	1
Susan	Heron	1	1
Frank	Texel	1	1
other ●	Clyde	1	1

shows $N \models \forall x(\text{lecturer}(x) \rightarrow \exists y \text{ bought}(x, y))$.

How hard is it?

In most practical cases, it's easy to do the evaluation mentally, without tables, once used to it.

But in general, evaluation is hard.

Suppose that N is the natural numbers with the usual meanings of *prime*, *even*, $>$, $+$.

No-one knows whether

$$N \models \forall x(\text{even}(x) \wedge x > 2 \rightarrow \exists y \exists z(\text{prime}(y) \wedge \text{prime}(z) \wedge x = y + z)).$$

7.6 Hintikka games

Working out \models by tables is clumsy. Often you can do the evaluation just by looking.

But if it's not immediately clear whether or not $M \models A$, it can help to use a game $G(M, A)$ with two players — me and you, say.

In $G(M, A)$, M is an L -structure, A is an L -sentence. You are trying to show that $M \models A$, and I am testing you to see if you can.

There are two labels, ' \forall ' and ' \exists '.

At the start, I am given the label \forall , and you get label \exists .

154

Winning strategies

A *strategy* for a player in $G(M, A)$ is just a set of rules telling that player what to do in any position.

A strategy is *winning* if its owner wins any play (or match) of the game in which the strategy is used.

Theorem 7.10 *Let M be an L -structure. Then $M \models A$ if and only if you have a winning strategy in $G(M, A)$.*

So your winning once is not enough for $M \models A$. You must be able to win however I play.

156

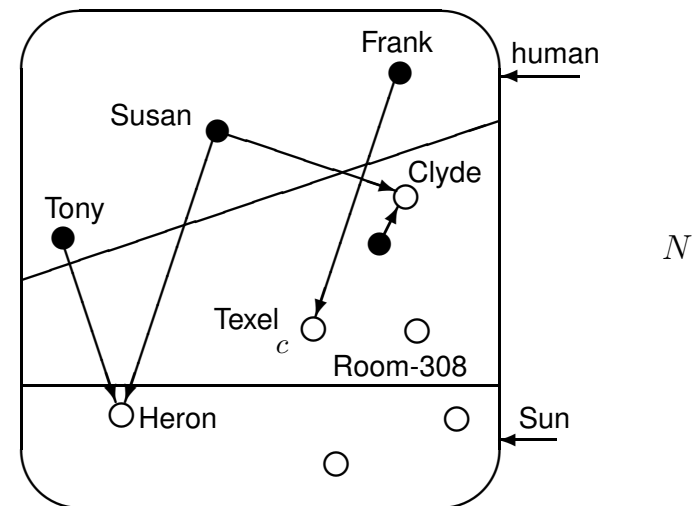
Playing $G(M, A)$

The game starts at the root of the formation tree of A , and works down node by node to the leaves. If the current node is:

- $\forall x$, then (the player labeled) \forall chooses a value (in $\text{dom}(M)$) for the variable x
- $\exists x$, then \exists chooses a value for x
- \wedge , then \forall chooses the next node down
- \vee , then \exists chooses the next node down
- \neg , then we swap labels (\forall and \exists)
- $\rightarrow, \leftrightarrow$ — regard $A \rightarrow B$ as $\neg A \vee B$, and $A \leftrightarrow B$ as $(A \wedge B) \vee (\neg A \wedge \neg B)$.
- an atomic (or quantifier-free) formula, then we stop and evaluate it in M with the current values of variables.
The player currently labeled \exists wins the game if it's true, and the one labeled \forall wins if it's false.

155

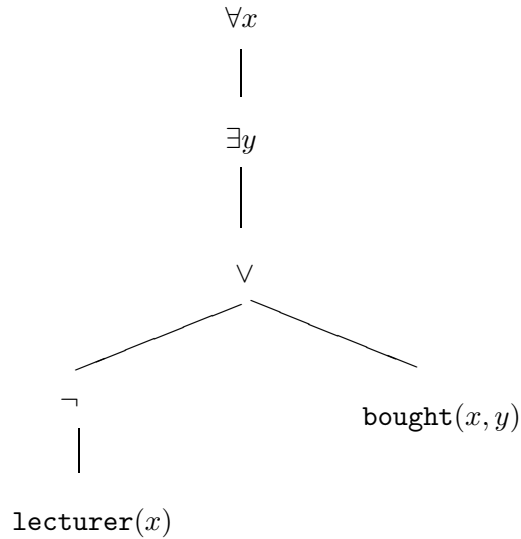
Let's play games on N



The black dots are the lecturers. The arrows indicate the interpretation in N of the relation symbol bought.

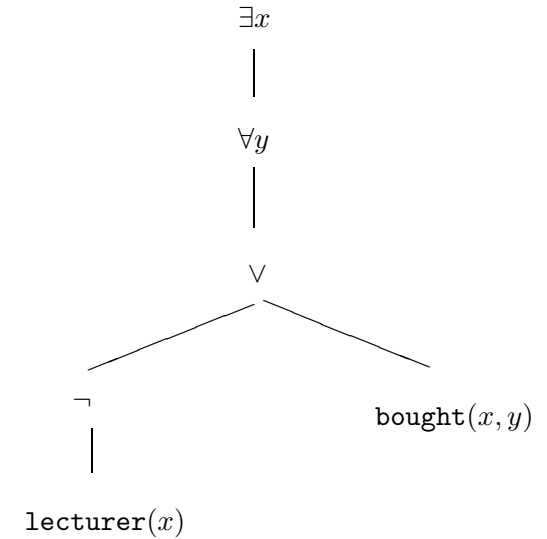
157

$N \models \forall x \exists y (\text{lecturer}(x) \rightarrow \text{bought}(x, y))?$



158

$N \models \exists y \forall x (\text{lecturer}(x) \rightarrow \text{bought}(x, y))?$



159

8. Translation into and out of logic

Translating predicate logic sentences from logic to English is not much harder than in propositional logic. But you can end up with a mess that needs careful simplifying.

$\forall x (\text{lecturer}(x) \wedge \neg(x = \text{Frank}) \rightarrow \text{bought}(x, \text{Texel}))$

'For all x , if x is a lecturer and x is not Frank then x bought Texel.'

'Every lecturer apart from Frank bought Texel.' (Maybe Frank did too.)

$\exists x \exists y \exists z (\text{bought}(x, y) \wedge \text{bought}(x, z) \wedge \neg(y = z))$

'There are x, y, z such that x bought y , x bought z , and y is not z .'

'Something bought at least two different things.'

$\forall x (\exists y \exists z (\text{bought}(x, y) \wedge \text{bought}(x, z) \wedge \neg(y = z)) \rightarrow x = \text{Tony})$

'For all x , if x bought two different things then x is equal to Tony.'

'Anything that bought two different things is Tony.'

CARE: it doesn't imply Tony did actually buy 2 things, just that noone else did.

160

English to logic

Hints for English-to-logic translation: express the sub-concepts in logic. Then build these pieces into a whole logical sentence.

Sample subconcepts:

- x buys y : $\text{bought}(x, y)$.
- x is bought: $\exists y \text{bought}(y, x)$.
- y is bought: $\exists z \text{bought}(z, y)$.
- x is a buyer: $\exists y \text{bought}(x, y)$.
- x buys at least two things:
 $\exists y \exists z (\text{bought}(x, y) \wedge \text{bought}(x, z) \wedge y \neq z)$.
 Here, $y \neq z$ abbreviates $\neg(y = z)$.

161

English-to-logic translation examples

- Every lecturer is human: $\forall x(\text{lecturer}(x) \rightarrow \text{human}(x))$.
- x is bought/has a buyer: $\exists y \text{ bought}(y, x)$.
- Anything bought is not human:
 $\forall x(\exists y \text{ bought}(y, x) \rightarrow \neg \text{human}(x))$.
 Note: $\exists y$ binds tighter than \rightarrow .
- Every Sun has a buyer: $\forall x(\text{Sun}(x) \rightarrow \exists y \text{ bought}(y, x))$.
- Some Sun has a buyer: $\exists x(\text{Sun}(x) \wedge \exists y \text{ bought}(y, x))$.
- All buyers are human lecturers:
 $\forall x(\underbrace{\exists y \text{ bought}(x, y)}_{x \text{ is a buyer}} \rightarrow \text{human}(x) \wedge \text{lecturer}(x))$.
- No lecturer bought a Sun:
 $\neg \exists x(\text{lecturer}(x) \wedge \underbrace{\exists y(\text{bought}(x, y) \wedge \text{Sun}(y))}_{x \text{ bought a Sun}})$.

162

Counting

- There is at least one Sun: $\exists x \text{ Sun}(x)$.
- There are at least two Suns: $\exists x \exists y(\text{Sun}(x) \wedge \text{Sun}(y) \wedge x \neq y)$,
 or (more deviously) $\forall x \exists y(\text{Sun}(y) \wedge y \neq x)$.
- There are at least three Suns:
 $\exists x \exists y \exists z(\text{Sun}(x) \wedge \text{Sun}(y) \wedge \text{Sun}(z) \wedge x \neq y \wedge y \neq z \wedge x \neq z)$,
 or $\forall x \forall y \exists z(\text{Sun}(z) \wedge z \neq x \wedge z \neq y)$.
- There are no Suns: $\neg \exists x \text{ Sun}(x)$
- There is at most one Sun: 3 ways:
 1. $\neg \exists x \exists y(\text{Sun}(x) \wedge \text{Sun}(y) \wedge x \neq y)$
 2. $\forall x \forall y(\text{Sun}(x) \wedge \text{Sun}(y) \rightarrow x = y)$
 3. $\exists x \forall y(\text{Sun}(y) \rightarrow x = y)$
- There's exactly 1 Sun: $\exists x \forall y(\text{Sun}(y) \leftrightarrow y = x)$.
- There are at most two Suns: 3 ways:
 1. $\neg(\text{there are at least 3 Suns})$
 2. $\forall x \forall y \forall z(\text{Sun}(x) \wedge \text{Sun}(y) \wedge \text{Sun}(z) \rightarrow x = y \vee x = z \vee y = z)$
 3. $\exists x \exists y \forall z(\text{Sun}(z) \rightarrow z = x \vee z = y)$

164

More examples

- Everything is a Sun or a lecturer (or both):
 $\forall x(\text{Sun}(x) \vee \text{lecturer}(x))$.
- Nothing is both a Sun and a lecturer:
 $\neg \exists x(\text{Sun}(x) \wedge \text{lecturer}(x))$, or
 $\forall x(\text{Sun}(x) \rightarrow \neg \text{lecturer}(x))$, or
 $\forall x(\text{lecturer}(x) \rightarrow \neg \text{Sun}(x))$, or
 $\forall x \neg(\text{Sun}(x) \wedge \text{lecturer}(x))$.
- Only Susan bought Clyde: $\forall x(\text{bought}(x, \text{Clyde}) \leftrightarrow x = \text{Susan})$.
- If Tony bought everything that Susan bought, and Tony bought a Sun, then Susan didn't buy a Sun:
 $\forall x(\text{bought}(\text{Susan}, x) \rightarrow \text{bought}(\text{Tony}, x))$
 $\wedge \exists y(\text{Sun}(y) \wedge \text{bought}(\text{Tony}, y))$
 $\rightarrow \neg \exists y(\text{Sun}(y) \wedge \text{bought}(\text{Susan}, y))$.
 (This may not be true! But we can still say it.)

163

Common patterns

You often need to say things like:

- 'All lecturers are human': $\forall x(\text{lecturer}(x) \rightarrow \text{human}(x))$.
 NOT $\forall x(\text{lecturer}(x) \wedge \text{human}(x))$.
 NOT $\forall x \text{ lecturer}(x) \rightarrow \forall x \text{ human}(x)$.
- 'All lecturers are human and not Suns':
 $\forall x(\text{lecturer}(x) \rightarrow \text{human}(x) \wedge \neg \text{Sun}(x))$.
- 'All human lecturers are Suns':
 $\forall x(\text{human}(x) \wedge \text{lecturer}(x) \rightarrow \text{Sun}(x))$.
- 'Some lecturer is a Sun': $\exists x(\text{lecturer}(x) \wedge \text{Sun}(x))$.

Patterns like $\forall x(A \rightarrow B)$, $\forall x(A \rightarrow B \wedge C)$, $\forall x(A \rightarrow B \vee C)$, and $\exists x(A \wedge B)$ are therefore common.

$\forall x(B \wedge C)$, $\forall x(B \vee C)$, $\exists x(B \wedge C)$, $\exists x(B \vee C)$ also crop up: they say every/some x is B and/or C .

$\exists x(A \rightarrow B)$ is *extremely rare*. If you write this, check to see if you've made a mistake.

165

9. Function symbols and sorts

— the icing on the cake.

9.1 Function symbols

In arithmetic (and Haskell) we are used to *functions*, such as $+$, $-$, \times , \sqrt{x} , $++$, etc.

Predicate logic can do this too.

A *function symbol* is like a relation symbol or constant, but it is interpreted in a structure as a *function* (to be defined in discr math).

Any function symbol comes with a fixed arity (number of arguments).

We often write f, g for function symbols.

From now on, we adopt the following extension of definition 6.1:

Definition 9.1 (signature) A *signature* is a collection of constants, and relation symbols and function symbols with specified arities.

166

Semantics of function symbols

We need to extend definition 7.1 too: if L has function symbols, an L -structure must additionally define their meaning.

For any n -ary function symbol f in L , an L -structure M *must* say which object (in $\text{dom}(M)$) f associates with any sequence (a_1, \dots, a_n) of n objects in $\text{dom}(M)$. We write this object as $f^M(a_1, \dots, a_n)$.

There must be such a value.

[f^M is a function $f^M : \text{dom}(M)^n \rightarrow \text{dom}(M)$.]

A 0-ary function symbol is like a constant.

Examples

In arithmetic, M *might* say $+$, \times are addition and multiplication of numbers: it associates 4 with $2 + 2$, 8 with 4×2 , etc.

If the objects of M are vectors, M might say $+$ is addition of vectors and \times is cross-product. M doesn't have to say this — it could say $+$ is addition — but we may not want to use the symbol \times in such a case.

168

Terms with function symbols

We can now extend definition 6.2:

Definition 9.2 (term) Fix a signature L .

1. Any constant of L is an L -term.
2. Any variable is an L -term.
3. If f is an n -ary function symbol of L , and t_1, \dots, t_n are L -terms, then $f(t_1, \dots, t_n)$ is an L -term.
4. Nothing else is an L -term.

Example

Let L have a constant c , a unary function symbol f , and a binary function symbol g . Then the following are L -terms:

- c
- $f(c)$
- $g(x, x)$
- $g(f(c), g(x, x))$

167

Evaluating terms with function symbols

We can now extend definition 7.6:

Definition 9.3 (value of term) The value of an L -term t in an L -structure M under an assignment h into M is defined as follows:

- If t is a constant, then its value is the object in M allocated to it by M ,
- If t is a variable, then its value is the object $h(t)$ in M allocated to it by h ,
- If t is $f(t_1, \dots, t_n)$, and the values of the terms t_i in M under h are already known to be a_1, \dots, a_n , respectively, then the value of t in M under h is $f^M(a_1, \dots, a_n)$.

So the value of a term in M under h is always *an object in $\text{dom}(M)$* . *Not true or false!*

Definition 7.8 needs no amendment, apart from using it with the extended definition 9.3.

We now have the standard system of first-order logic (as in books).

169

Arithmetic terms

A useful signature for arithmetic and for programs using numbers is the L consisting of:

- constants $\underline{0}, \underline{1}, \underline{2}, \dots$ (I use underlined typewriter font to avoid confusion with actual numbers $0, 1, \dots$)
- binary function symbols $+, -, \times$
- binary relation symbols $<, \leq, >, \geq$.

We interpret these in a structure with domain $\{0, 1, 2, \dots\}$ in the obvious way. But (eg) $34 - 61$ is unpredictable — can be any number.

We'll abuse notation by writing L -terms and formulas in infix notation:

- $x + y$, rather than $+(x, y)$,
- $x > y$, rather than $>(x, y)$.

Everybody does this, but it's breaking definitions 9.2 and 6.3, and it means we'll need to use brackets.

Some terms: $x + \underline{1}$, $\underline{2} + (x + \underline{5})$, $(\underline{3} \times \underline{7}) + x$.

Formulas: $\underline{3} \times x > \underline{0}$, $\forall x(x > \underline{0} \rightarrow x \times x > x)$.

170

Many-sorted terms

We adjust the definition of 'term' (definition 9.2), to give each term a sort:

- each variable and constant comes with a sort \mathbf{s} , expressed as $c : \mathbf{s}$ and $x : \mathbf{s}$. There are infinitely many variables of each sort.
- each n -ary function symbol f comes with a template

$$f : (\mathbf{s}_1, \dots, \mathbf{s}_n) \rightarrow \mathbf{s},$$

where $\mathbf{s}_1, \dots, \mathbf{s}_n, \mathbf{s}$ are sorts.

Note: $(\mathbf{s}_1, \dots, \mathbf{s}_n) \rightarrow \mathbf{s}$ is not itself a sort.

- For such an f and terms t_1, \dots, t_n , if t_i has sort \mathbf{s}_i (for each i) then $f(t_1, \dots, t_n)$ is a term of sort \mathbf{s} .

Otherwise (if the t_i don't all have the right sorts), $f(t_1, \dots, t_n)$ is not a term — it's just rubbish, like $\forall y \rightarrow$.

172

9.2 Many-sorted logic

As in typed programming languages, it sometimes helps to have structures with objects of different types. In logic, types are called *sorts*.

Eg some objects in a structure M may be lecturers, others may be Suns, numbers, etc.

We can handle this with unary relation symbols, or with '*many-sorted first-order logic*'.

Fix a collection $\mathbf{s}, \mathbf{s}', \mathbf{s}'', \dots$ of sorts. How many, and what they're called, are determined by the application.

These sorts do *not* generate extra sorts, like $\mathbf{s} \rightarrow \mathbf{s}'$ or $(\mathbf{s}, \mathbf{s}')$.

If you want extra sorts like these, add them explicitly to the original list of sorts. (Their meaning would not be automatic, unlike in Haskell.)

171

Formulas in many-sorted logic

- Each n -ary relation symbol R comes with a template $R(\mathbf{s}_1, \dots, \mathbf{s}_n)$, where $\mathbf{s}_1, \dots, \mathbf{s}_n$ are sorts. For terms t_1, \dots, t_n , if t_i has sort \mathbf{s}_i (for each i) then $R(t_1, \dots, t_n)$ is a formula. Otherwise, it's rubbish.
- $t = t'$ is a formula if the terms t, t' have the same sort. Otherwise, it's rubbish.
- Other operations ($\wedge, \neg, \forall, \exists$, etc) are unchanged. But it's polite to indicate the sort of a variable in \forall, \exists by writing

$$\forall x : \mathbf{s} A \quad \text{and} \quad \exists x : \mathbf{s} A$$

instead of just

$$\forall x A \quad \text{and} \quad \exists x A$$

if x has sort \mathbf{s} .

This all sounds complicated, but it's very simple in practice.

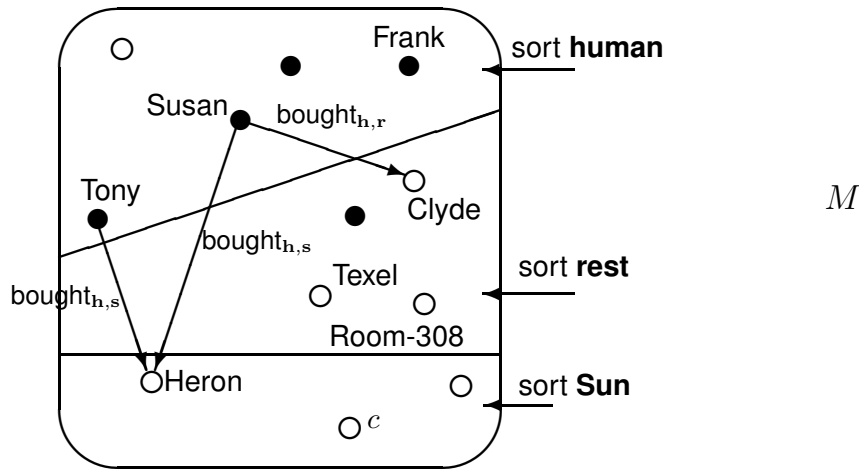
Eg, you can write $\forall x : \text{lecturer} \exists y : \text{Sun}(\text{bought}(x, y))$ instead of $\forall x(\text{lecturer}(x) \rightarrow \exists y(\text{Sun}(y) \wedge \text{bought}(x, y)))$.

173

L-structures for many-sorted L

Let L be a many-sorted signature, with sorts s_1, s_2, \dots

An L -structure is defined as before (definition 7.1 + slide 168), but additionally it allocates *each* object in its domain to *a single sort* (one of s_1, s_2, \dots). So it looks like:



174

We need a binary relation symbol $\text{bought}_{s,s'}$ for *each* pair (s, s') of sorts.

lecturer (black dots) must be implemented as 2 or 3 relation symbols, because (as in Haskell) each object has only 1 sort, not 2. (Alternative: use sorts for human lecturer, non-human lecturer, etc — all possible types of object.)

175

Interpretation of L-symbols

A many-sorted L -structure M must say:

- for each constant $c : s$ in L , which object of sort s in $\text{dom}(M)$ is 'named' by c
- for each relation symbol $R : (s_1, \dots, s_n)$ in L , and all objects a_1, \dots, a_n in $\text{dom}(M)$ of sorts s_1, \dots, s_n , respectively, whether $R(a_1, \dots, a_n)$ is true or not.

It doesn't say anything about $R(b_1, \dots, b_n)$ if b_1, \dots, b_n don't all have the right sorts.

- for each function symbol $f : (s_1, \dots, s_n) \rightarrow s$ in L and all objects a_1, \dots, a_n in $\text{dom}(M)$ of sorts s_1, \dots, s_n , respectively, which object $f^M(a_1, \dots, a_n)$ of sort s is associated with (a_1, \dots, a_n) by f .

It doesn't say anything about $f(b_1, \dots, b_n)$ if b_1, \dots, b_n don't all have the right sorts.

176

10. Application of logic: specifications

A *specification* is a description of what a program should do.

It should state the inputs and outputs (and their types).

It should include conditions on the input under which the program is guaranteed to operate. This is the *pre-condition*.

It should state what is required of the outcome in all cases (output for each input). This is the *post-condition*.

- The type (in the function header) is part of the specification.
- The pre-condition refers to the inputs (only).
- The post-condition refers to the outputs and inputs.

177

Precision is vital

A specification should be unambiguous. It is a *CONTRACT*:

Programmer wants pre-condition and post-condition to be the same — less work to do! The weaker the pre-condition and/or stronger the post-condition, the more work for the programmer — fewer assumptions (so more checks) and more results to produce.

Customer wants weak pre-condition and strong post-condition, for added value — less work before execution of program, more gained after execution of it.

Customer guarantees pre-condition so program will operate.
Programmer guarantees post-condition, provided that the input meets the pre-condition.

If customer (user) provides the pre-condition (on the inputs), then provider (programmer) will guarantee the post-condition (between inputs and outputs).

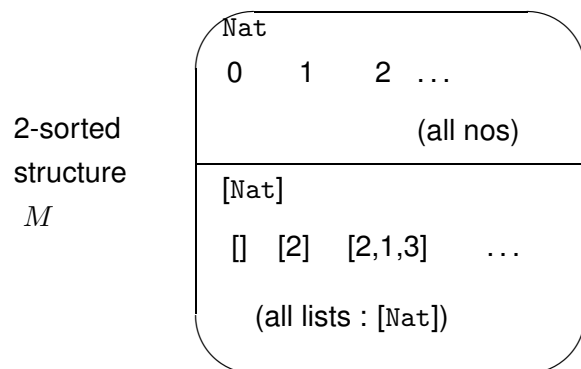
178

Example: lists of type [Nat]

Let's have a sort `Nat`, for $\{0,1,2,\dots\}$, and a sort `[Nat]` for lists of natural numbers.

(Using the real Haskell `Int` is more longwinded: must keep saying $n \geq 0$ etc.)

The idea is that the structure's domain should look like:



180

10.1 Logic for specifying Haskell programs

A very precise way to specify properties of Haskell programs is to use first-order logic.

(Logic can also be used for Java, etc)

We use many-sorted logic, so we can have a sort for each Haskell type we want.

179

10.2 Signature for lists

The signature should be chosen to provide access to the objects in such a structure.

We want `[]`, `:` (cons), `++`, `head`, `tail`, `#!`, `!!`.

How do we represent these using constants, function symbols, or relation symbols?

How about a constant `[] : [Nat]` for the empty list, and function symbols

- `cons : (Nat, [Nat]) → [Nat]`
- `++ : ([Nat], [Nat]) → [Nat]`
- `head : [Nat] → Nat`
- `tail : [Nat] → [Nat]`
- `#! : [Nat] → Nat`
- `!! : ([Nat], Nat) → Nat`

181

Problem: tail etc are partial operations

In first-order logic, a structure *must* provide a meaning for function symbols *on all possible arguments* (of the right sorts).

What is the head or tail of the empty list? What is $xs !! \#(xs)$? What is $34 - 61$?

Two solutions (for tail):

1. Choose an arbitrary value (of the right sort) for $\text{tail}(\square)$, etc.
2. Use a relation symbol $\text{Rtail}([\text{Nat}], [\text{Nat}])$ instead of a function symbol $\text{tail} : [\text{Nat}] \rightarrow [\text{Nat}]$. Make $\text{Rtail}(xs, ys)$ true just when ys is the tail of xs . If xs has no tail, $\text{Rtail}(xs, ys)$ will be false for all ys .

Similarly for head, !!. E.g., use a function symbol

$!! : ([\text{Nat}], \text{Nat}) \rightarrow \text{Nat}$, and choose arbitrary value for $!!(xs, n)$ when $n \geq \#(xs)$. Or use a relation symbol $!!([\text{Nat}], \text{Nat}, \text{Nat})$.

We'll take the function symbol option (1), as it leads to shorter formulas. But we must beware: values of functions on 'invalid' arguments are 'unpredictable'.

182

10.3 Saying things about lists

Now we can say *a lot* about lists.

E.g., the following *L*-sentences, expressing the definitions of the function symbols, are true in *M*, because (as we said) the *L*-symbols are interpreted in *M* in the natural way:

$$\#(\square) = \underline{0}$$

$$\forall x \forall xs ((\#(x:xs) = \#(xs) + \underline{1}) \wedge ((x:xs)!!\underline{0} = x))$$

$$\forall x \forall xs \forall n (n < \#(xs) \rightarrow (x:xs)!!(n + \underline{1}) = xs!!n)$$

Note the ' $n < \#(xs)$ ': $xs!!n$ could be anything if $n \geq \#(xs)$.

$$\forall xs (\#(xs) = \underline{0} \vee \text{head}(xs) = xs!!\underline{0})$$

$$\forall xs (xs \neq \square \rightarrow \#(\text{tail}(xs)) = \#(xs) - \underline{1})$$

$$\forall xs \forall n (\underline{0} < n \wedge n < \#(xs) \rightarrow xs!!n = \text{tail}(xs)!!(n - \underline{1}))$$

$$\forall xs \forall ys \forall zs (xs = ys ++ zs \leftrightarrow$$

$$\begin{aligned} \#(xs) &= \#(ys) + \#(zs) \wedge \forall n (n < \#(ys) \rightarrow xs!!n = ys!!n) \\ &\wedge \forall n (n < \#(zs) \rightarrow xs!!(n + \#(ys)) = zs!!n). \end{aligned}$$

184

Lists in first-order logic: summary

Now we can define a signature *L* suitable for lists of type $[\text{Nat}]$.

- *L* has constants $\underline{0}, \underline{1}, \dots : \text{Nat}$, relation symbols $<, \leq, >, \geq$ of sort (Nat, Nat) , a constant $\square : [\text{Nat}]$, and function symbols $+, -, :$ or $\text{cons}, ++, \text{head}, \text{tail}, \#, !!$, with sorts as specified 2 slides ago.

We write the constants as $\underline{0}, \underline{1}, \dots$ to avoid confusion with actual numbers $0, 1, \dots$. We write symbols in infix notation where appropriate.

- Let x, y, z, k, n, m, \dots be variables of sort Nat , and xs, ys, zs, \dots variables of sort $[\text{Nat}]$.
- Let *M* be an *L*-structure in which the objects of sort Nat are the natural numbers $0, 1, \dots$, the objects of sort $[\text{Nat}]$ are all possible lists of natural numbers, and the *L*-symbols are interpreted in the natural way: $++$ as concatenation of lists, etc. (Define $34 - 61, \text{tail}(\square)$, etc. arbitrarily.)

See figure, 3 slides ago.

183

10.4 Specifying Haskell functions

Now we know how to use logic to say things about lists, we can use logic to specify Haskell functions.

Pre-conditions in logic

These express restrictions on the arguments or parameters that can be legally passed to a function. *You write a formula* $A(a, b)$ that is true if and only if the arguments a, b satisfy the intended pre-condition (are legal).

Eg for the function $\log(x)$, you'd want a pre-condition of $x > \underline{0}$. For \sqrt{x} you'd want $x \geq \underline{0}$.

Pre-conditions are usually very easy to write:

- xs is not empty: use $xs \neq \square$.
- n is non-negative: use $n \geq \underline{0}$.

185

Type information

This is *not* part of the pre-condition.

If there are no restrictions on the arguments beyond their typing information, you can write 'none', or \top , as pre-condition.

This is perfectly normal and is no cause for alarm.

186

Specifying the function 'in'

```
in :: Nat -> [Nat] -> Bool
-- pre:none
-- post: in x xs <--> (E)k:Nat(k<#xs & xs!!k=x)
```

I used (E) and & as can't type \exists, \wedge in Haskell.

Similarly, use $\setminus/$ for \vee , (A) for \forall , ~ for \neg .

188

Post-conditions in logic

These express the required connection between the input and output of a function.

To do post-conditions in logic, *you write a formula* expressing the intended value of a function in terms of its arguments.

The formula should have free variables for the arguments, and should involve the function call so as to describe the required value. The formula should be true if and only if the output is as intended.

```
addone :: Nat -> Nat
-- pre:none
-- post:addone n = n+1
OR, in another commonly-used style,
-- post: z = n+1 where z = addone n
```

187

Existence, non-uniqueness of result

Suppose you have a post-condition $A(x, y, z)$, where the variables x, y represent the input, and z represents the output.

Idea: for inputs a, b in M , the function should return some c such that $M \models A(a, b, c)$.

There is no requirement that c be unique: could have $M \models A(a, b, c) \wedge A(a, b, d) \wedge c \neq d$. Then the function could legally return c or d . It can return *any* value satisfying the post-condition.

But should arrange that $M \models \exists z A(a, b, z)$ whenever a, b meet the pre-condition: otherwise, the function cannot meet its post-condition.

So need $M \models \forall x \forall y (pre(x, y) \rightarrow \exists z post(x, y, z))$, for functions of 2 arguments with pre-, post-conditions given by formulas $pre, post$.

189

10.5 Examples

Saying something is in a list

$\exists k(k < \#(xs) \wedge xs!!k = n)$ says that n occurs in xs . So does $\exists ys \exists zs(xs = ys++(n:zs))$.

Write $in(n, xs)$ for either of these formulas.

Then for any number a and list bs in M , we have $M \models in(a, bs)$ just when a occurs in bs .

So can specify a Haskell function for in :

```
isin :: Nat -> [Nat] -> Bool
-- pre: none
-- post: isin n xs <--> (E)ys,zs(xs=ys++n:zs)
```

The code for `isin` may in the end be very different from the post-condition(!), but `isin` should meet its post-condition.

190

Merge

Informal specification:

```
merge :: [Nat] -> [Nat] -> [Nat] -> Bool
-- pre:none
-- post:merge(xs,ys,zs) holds when xs, ys are
-- merged to give zs, the elements of xs and ys
-- remaining in the same relative order.
```

`merge([1,2],[3,4,5],[1,3,4,2,5])` and
`merge([1,2],[3,4,5],[3,4,1,2,5])` are true.

`merge([1,2],[3,4,5],[1])` and
`merge([1,2],[3,4,5],[5,4,3,2,1])` are false.

192

Least entry

$in(m, xs) \wedge \forall n(n < \#(xs) \rightarrow xs!!n \geq m)$

expresses that (is true in M iff) m is the least entry in list xs .

So could specify a function `least`:

```
least :: [Nat] -> Nat
-- pre: input is non-empty
-- post: in(m,xs) & (A)n(n<#xs -> xs!!n>=m), where m = least xs
```

Ordered (or sorted) lists

$\forall n \forall m(n < m \wedge m < \#(xs) \rightarrow xs!!n \leq xs!!m)$ expresses that list xs is ordered. So does $\forall ys \forall zs \forall m \forall n(xs = ys++(m:(n:zs)) \rightarrow m \leq n)$.

Exercise: specify a function

```
sorted :: [Nat] -> Bool
```

that returns true if and only if its argument is an ordered list.

191

Specifying 'merge'

Quite hard to specify explicitly (challenge for you!).

But can write an implicit specification:

$$\forall xs \forall zs(\text{merge}(xs, [], zs) \leftrightarrow xs = zs)$$

$$\forall ys \forall zs(\text{merge}([], ys, zs) \leftrightarrow ys = zs)$$

$$\forall x \dots zs[\text{merge}(x:xs, y:ys, z:zs) \leftrightarrow (x = z \wedge \text{merge}(xs, y:ys, zs) \vee y = z \wedge \text{merge}(x:xs, ys, zs))]$$

This pins down `merge` exactly: there exists a unique way to interpret a 3-ary relation symbol `merge` in M so that these three sentences are true. So they could form a post-condition.

193

Count

Can use `merge` to specify other things:

```

count : Nat -> [Nat] -> Nat
-- pre:none
-- post (informal): count x xs = number of x's in xs
-- post: (E)ys,zs(merge ys zs xs
--      & (A)n:Nat(in(n,ys) -> n=x)
--      & (A)n:Nat(in(n,zs) -> n<>x)
--      & count x xs = #ys)

```

Idea: `ys` takes all the `x` from `xs`, and `zs` takes the rest. So the number of `x` is $\#(ys)$.

Conclusion

First-order logic is a valuable and powerful way to specify programs precisely, by writing first-order formulas expressing their pre- and post-conditions.

More on this in 141 ‘Reasoning about Programs’ next term.

194

Validity, satisfiability, equivalence

These are defined as in propositional logic.

Definition 11.2 (valid formula) *A formula A is (logically) valid if for every structure M and assignment h into M , we have $M, h \models A$. We write $\models A$ (as above) if A is valid.*

Definition 11.3 (satisfiable formula) *A formula A is satisfiable if for some structure M and assignment h into M , we have $M, h \models A$.*

Definition 11.4 (equivalent formulas)

Formulas A, B are logically equivalent if for every structure M and assignment h into M , we have $M, h \models A$ if and only if $M, h \models B$.

The links between these (page 43) also hold for predicate logic.

So (eg) the notions of valid/satisfiable formula, and equivalence, can all be expressed in terms of valid arguments.

196

11. Arguments, validity

Predicate logic is much more expressive than propositional logic. But our experience with propositional logic tells us how to define ‘valid argument’ etc.

Definition 11.1 (valid argument) *Let L be a signature and A_1, \dots, A_n, B be L -formulas.*

An argument ‘ A_1, \dots, A_n , therefore B ’ is valid if for any L -structure M and assignment h into M ,

if $M, h \models A_1, M, h \models A_2, \dots$, and $M, h \models A_n$, then $M, h \models B$.

We write $A_1, \dots, A_n \models B$ in this case.

This says: in any situation (structure + assignment) in which A_1, \dots, A_n are all true, B must be true too.

Special case: $n = 0$. Then we write just $\models B$. It means that B is true in every L -structure under every assignment into it.

195

Which arguments are valid?

Some examples of valid arguments:

- valid propositional ones: eg, $A \wedge B \models A$.
- many new ones: eg

$$\forall x(\text{lecturer}(x) \rightarrow \text{human}(x)),$$

$$\exists x(\text{lecturer}(x) \wedge \text{bought}(x, \text{Texel}))$$

$$\models \exists x(\text{human}(x) \wedge \text{bought}(x, \text{Texel})).$$

‘All lecturers are human, some lecturer bought Texel
 \models some human bought Texel.’

Deciding if a supposed argument $A_1, \dots, A_n \models B$ is valid is extremely hard in general.

We cannot just check that all L -structures + assignments that make A_1, \dots, A_n true also make B true (like truth tables).

This is because there are infinitely many L -structures.

Theorem 11.5 (Church, 1935) *No computer program can be written to identify precisely the valid arguments of predicate logic.*

197

Useful ways of validating arguments

In spite of theorem 11.5, we can often verify in practice that an argument or formula in predicate logic is valid. Ways to do it include:

- direct reasoning (the easiest, once you get used to it)
- equivalences (also useful)
- proof systems like natural deduction

The same methods work for showing a formula is valid. (A is valid if and only if $\models A$.)

Truth tables no longer work. You can't tabulate all structures — there are infinitely many.

198

Another example

Let's show

$$\begin{aligned} &\forall x(\text{human}(x) \rightarrow \text{lecturer}(x)), \\ &\forall x(\text{Sun}(x) \rightarrow \text{lecturer}(x)), \\ &\forall x(\text{human}(x) \vee \text{Sun}(x)) \quad \models \forall x \text{lecturer}(x). \end{aligned}$$

Take any M such that

- 1) $M \models \forall x(\text{human}(x) \rightarrow \text{lecturer}(x))$,
- 2) $M \models \forall y(\text{Sun}(y) \rightarrow \text{lecturer}(y))$,
- 3) $M \models \forall z(\text{human}(z) \vee \text{Sun}(z))$.

Show $M \models \forall x \text{lecturer}(x)$.

Take arbitrary a in M . We require $M \models \text{lecturer}(a)$.

Well, by (3), $M \models \text{human}(a) \vee \text{Sun}(a)$.

If $M \models \text{human}(a)$, then by (1), $M \models \text{lecturer}(a)$.

Otherwise, $M \models \text{Sun}(a)$. Then by (2), $M \models \text{lecturer}(a)$.

So either way, $M \models \text{lecturer}(a)$, as required.

200

11.1 Direct reasoning

Let's show

$$\begin{aligned} &\forall x(\text{lecturer}(x) \rightarrow \text{human}(x)), \quad \exists x(\text{lecturer}(x) \wedge \text{bought}(x, \text{Texel})) \\ &\quad \models \exists x(\text{human}(x) \wedge \text{bought}(x, \text{Texel})). \end{aligned}$$

Take any L -structure M (where L is as before). Assume that

- 1) $M \models \forall x(\text{lecturer}(x) \rightarrow \text{human}(x))$ and
- 2) $M \models \exists x(\text{lecturer}(x) \wedge \text{bought}(x, \text{Texel}))$.

Show $M \models \exists x(\text{human}(x) \wedge \text{bought}(x, \text{Texel}))$.

So we need to find an a in M such that

$$M \models \text{human}(a) \wedge \text{bought}(a, \text{Texel}).$$

By (2), there is a in M such that

$$M \models \text{lecturer}(a) \wedge \text{bought}(a, \text{Texel}).$$

So $M \models \text{lecturer}(a)$.

By (1), $M \models \text{lecturer}(a) \rightarrow \text{human}(a)$.

So $M \models \text{human}(a)$.

So $M \models \text{human}(a) \wedge \text{bought}(a, \text{Texel})$, as required.

199

Direct reasoning with equality

Let's show $\forall x \forall y (x = y \wedge \exists z R(x, z) \rightarrow \exists v R(y, v))$ is valid.

Take any structure M , and objects a, b in $\text{dom}(M)$. We need to show

$$M \models a = b \wedge \exists z R(a, z) \rightarrow \exists v R(b, v).$$

So we need to show that

IF $M \models a = b \wedge \exists z R(a, z)$ THEN $M \models \exists v R(b, v)$.

But IF $M \models a = b \wedge \exists z R(a, z)$, then a, b are the same object.

So $M \models \exists z R(b, z)$.

So there is an object c in $\text{dom}(M)$ such that $M \models R(b, c)$.

Therefore, $M \models \exists v R(b, v)$. We're done.

201

11.2 Equivalences

As well as the propositional equivalences seen before, we have extra ones for predicate logic. A, B denote arbitrary predicate formulas.

28. $\forall x\forall yA$ is logically equivalent to $\forall y\forall xA$.
29. $\exists x\exists yA$ is (logically) equivalent to $\exists y\exists xA$.
30. $\neg\forall xA$ is equivalent to $\exists x\neg A$.
31. $\neg\exists xA$ is equivalent to $\forall x\neg A$.
32. $\forall x(A \wedge B)$ is equivalent to $\forall xA \wedge \forall xB$.
33. $\exists x(A \vee B)$ is equivalent to $\exists xA \vee \exists xB$.

202

Equivalences/validities involving equality

39. For any term t , $t = t$ is valid.
40. For any terms t, u ,
 $t = u$ is equivalent to $u = t$
41. (Leibniz principle) If A is a formula in which x occurs free, y doesn't occur in A at all, and B is got from A by replacing one or more free occurrences of x by y , then

$$x = y \rightarrow (A \leftrightarrow B)$$

is valid.

Example: $x = y \rightarrow (\forall zR(x, z) \leftrightarrow \forall zR(y, z))$ is valid.

204

34. If x does not occur free in A , then $\forall xA$ and $\exists xA$ are equivalent to A .
35. If x doesn't occur free in A , then
 $\exists x(A \wedge B)$ is equivalent to $A \wedge \exists xB$, and
 $\forall x(A \vee B)$ is equivalent to $A \vee \forall xB$.
36. If x does not occur free in A then
 $\forall x(A \rightarrow B)$ is equivalent to $A \rightarrow \forall xB$.
37. **Note:** if x does not occur free in B then
 $\forall x(A \rightarrow B)$ is equivalent to $\exists xA \rightarrow B$.
38. (Renaming bound variables)
If Q is \forall or \exists , y is a variable that does not occur in A , and B is got from A by replacing all free occurrences of x in A by y , then QxA is equivalent to QyB .
Eg $\forall x\exists y \text{ bought}(x, y)$ is equivalent to $\forall z\exists v \text{ bought}(z, v)$.

203

Examples using equivalences

These equivalences form a toolkit for transforming formulas.

Eg: let's show that if x is not free in A then $\forall x(\exists x\neg B \rightarrow \neg A)$ is equivalent to $\forall x(A \rightarrow B)$.

Well, the following formulas are equivalent:

- $\forall x(\exists x\neg B \rightarrow \neg A)$
- $\exists x\neg B \rightarrow \neg A$ (equivalence 34, since x is not free in $\exists x\neg B \rightarrow \neg A$)
- $\neg\forall xB \rightarrow \neg A$ (equivalence 30)
- $A \rightarrow \forall xB$ (example on p. 59)
- $\forall x(A \rightarrow B)$ (equivalence 36, since x is not free in A).

205

Warning: non-equivalences

Depending on A, B , the following need *NOT* be logically equivalent (though the first \models the second):

- $\forall x(A \rightarrow B)$ and $\forall xA \rightarrow \forall xB$
- $\exists x(A \wedge B)$ and $\exists xA \wedge \exists xB$.
- $\forall xA \vee \forall xB$ and $\forall x(A \vee B)$.

Can you find a ‘countermodel’ for each one? (Find suitable A, B and a structure M such that $M \models$ 2nd but $M \not\models$ 1st.)

206

\exists -introduction, or $\exists I$

To prove a sentence $\exists xA$, you have to prove $A(t)$, for some closed term t of your choice.

⋮		
1	$A(t)$	we got this somehow...
2	$\exists xA$	$\exists I(1)$

Notation 11.6 Here, and below, $A(t)$ is the sentence got from $A(x)$ by replacing all free occurrences of x by t .

Recall a *closed term* is one with no variables — it’s made with only constants and function symbols.

This rule is reasonable. If in some structure, $A(t)$ is true, then so is $\exists xA$, because there exists an object in M (namely, the value in M of t) making A true.

But choosing the ‘right’ t can be hard — that’s why it’s such a good idea to think up a ‘direct argument’ first!

208

11.3 Natural deduction for predicate logic

This is quite easy to set up. We keep the old propositional rules — e.g., $A \vee \neg A$ for any first-order sentence A (‘lemma’) — and add new ones for $\forall, \exists, =$.

You construct natural deduction proofs as for propositional logic: first think of a direct argument, then convert to ND.

This is *even more important than for propositional logic*. There’s quite an art to it.

Validating arguments by predicate ND can sometimes be harder than for propositional ones, because the new rules give you wide choices, and at first you may make the wrong ones! If you find this depressing, remember, it’s a hard problem, there’s no computer program to do it (theorem 11.5)!

207

\exists -elimination, $\exists E$ (tricky!)

Let $A(x)$ be a formula. If you have managed to write down $\exists xA$, you can prove a sentence B from it by

- assuming $A(c)$, where c is a *new* constant not used in B or in the proof so far,
- proving B from this assumption.

During the proof, you can use anything already established. But once you’ve proved B , you cannot use any part of the proof, *including c* , later on. *I mean it!* So we isolate the proof of B from $A(c)$, in a box:

1	$\exists xA$	got this somehow
2	$A(c)$	ass
	⟨the proof⟩	hard struggle
3	B	we made it!
4	B	$\exists E(1, 2, 3)$

c is often called a Skolem constant.

209

Justification of $\exists E$

Basically, ‘we can give any object a name’.

If $\exists xA$ is true in some structure M , then there is an object a in $dom(M)$ such that $M \models A(a)$.

Now a may not be named by a constant in M . But we can add a new constant to name it — say, c — and add the information to M that c names a .

c must be new — the other constants already in use may not name a in M .

So $A(c)$ for new c is really no better or worse than $\exists xA$. If we can prove B from the assumption $A(c)$, it counts as a proof of B from the already-proved $\exists xA$.

210

\forall -introduction, $\forall I$

To introduce the sentence $\forall xA$, for some $A(x)$, you introduce a *new* constant, say c , not used in the proof so far, and prove $A(c)$.

During the proof, you can use anything already established.

But once you’ve proved $A(c)$, you can no longer use the constant c later on.

So isolate the proof of $A(c)$, in a box:

1	c	$\forall I$ const
	\langle the proof \rangle	hard struggle
2	$A(c)$	we made it!
3	$\forall xA$	$\forall I(1, 2)$

This is the *only* time in ND that you write a line (1) containing a *term*, not a formula. And it’s the *only* time a box doesn’t start with a line labelled ‘ass’.

212

Example of \exists -rules

Show $\exists x(P(x) \wedge Q(x)) \vdash \exists xP(x) \wedge \exists xQ(x)$.

1	$\exists x(P(x) \wedge Q(x))$	given
2	$P(c) \wedge Q(c)$	ass
3	$P(c)$	$\wedge E(2)$
4	$\exists xP(x)$	$\exists I(3)$
5	$Q(c)$	$\wedge E(2)$
6	$\exists xQ(x)$	$\exists I(5)$
7	$\exists xP(x) \wedge \exists xQ(x)$	$\wedge I(4, 6)$
8	$\exists xP(x) \wedge \exists xQ(x)$	$\exists E(1, 2, 7)$

In English: Assume $\exists x(P(x) \wedge Q(x))$. Then there is a with $P(a) \wedge Q(a)$.

So $P(a)$ and $Q(a)$. So $\exists xP(x)$ and $\exists xQ(x)$.

So $\exists xP(x) \wedge \exists xQ(x)$, as required.

Note: only sentences occur in ND proofs. They should never involve formulas with free variables!

211

Justification

To show $M \models \forall xA$, we must show $M \models A(a)$ for every object a in $dom(M)$.

So choose an arbitrary a , add a new constant c naming a , and prove $A(c)$. As a is arbitrary, this shows $\forall xA$.

c must be new, because the constants already in use may not name this particular a .

213

\forall -elimination, or $\forall E$

Let $A(x)$ be a formula. If you have managed to write down $\forall xA$, you can go on to write down $A(t)$ for any closed term t . (It's your choice which t !)

:		
1	$\forall xA$	we got this somehow...
2	$A(t)$	$\forall E(1)$

This is easily justified: if $\forall xA$ is true in a structure, then certainly $A(t)$ is true, for any closed term t .

Choosing the 'right' t can be hard — that's why it's such a good idea to think up a 'direct argument' first!

Example with all the quantifier rules

Show $\exists x\forall yG(x, y) \vdash \forall y\exists xG(x, y)$.

1	$\exists x\forall yG(x, y)$	given
2	d	$\forall I$ const
3	$\forall yG(c, y)$	ass
4	$G(c, d)$	$\forall E(3)$
5	$\exists xG(x, d)$	$\exists I(4)$
6	$\exists xG(x, d)$	$\exists E(1, 3, 5)$
7	$\forall y\exists xG(x, y)$	$\forall I(2, 6)$

English: Assume $\exists x\forall yG(x, y)$. Then there is some object c such that $\forall yG(c, y)$.

So for any object d , we have $G(c, d)$, so certainly $\exists xG(x, d)$.

Since d was arbitrary, we have $\forall y\exists xG(x, y)$.

Example of \forall -rules

Let's show $P \rightarrow \forall xQ(x) \vdash \forall x(P \rightarrow Q(x))$.

Here, P is a 0-ary relation symbol — that is, a propositional atom.

1	$P \rightarrow \forall xQ(x)$	given
2	c	$\forall I$ const
3	P	ass
4	$\forall xQ(x)$	$\rightarrow E(3, 1)$
5	$Q(c)$	$\forall E(4)$
6	$P \rightarrow Q(c)$	$\rightarrow I(3, 5)$
7	$\forall x(P \rightarrow Q(x))$	$\forall I(2, 6)$

In English: Assume $P \rightarrow \forall xQ(x)$. Then for any object a , if P then $\forall xQ(x)$, so $Q(a)$.

So for any object a , if P , then $Q(a)$.

That is, for any object a , we have $P \rightarrow Q(a)$. So $\forall x(P \rightarrow Q(x))$.

Derived rule $\forall \rightarrow E$

This is like PC: it collapses two steps into one. Useful, but not essential.

Idea: often we have proved $\forall x(A(x) \rightarrow B(x))$ and $A(t)$, for some formulas $A(x), B(x)$ and some closed term t .

We know we can derive $B(t)$ from this:

1	$\forall x(A(x) \rightarrow B(x))$	(got this somehow)
2	$A(t)$	(this too)
3	$A(t) \rightarrow B(t)$	$\forall E(1)$
4	$B(t)$	$\rightarrow E(2, 3)$

So let's just do it in 1 step:

1	$\forall x(A(x) \rightarrow B(x))$	(got this somehow)
2	$A(t)$	(this too)
3	$B(t)$	$\forall \rightarrow E(2, 1)$

Example of $\forall \rightarrow E$ in action

Show $\forall x \forall y (P(x, y) \rightarrow Q(x, y)), \exists x P(x, a) \vdash \exists y Q(y, a)$.

1	$\forall x \forall y (P(x, y) \rightarrow Q(x, y))$	given
2	$\exists x P(x, a)$	given
3	$P(c, a)$	ass
4	$Q(c, a)$	$\forall \rightarrow E(3, 1)$
5	$\exists y Q(y, a)$	$\exists I(4)$
6	$\exists y Q(y, a)$	$\exists E(2, 3, 5)$

We used $\forall \rightarrow E$ on 2 \forall s at once. This is even more useful.

218

More rules for equality

- Substitution of equal terms (=sub).

If $A(x)$ is a formula, t, u are closed terms, you've proved $A(t)$, and you've also proved either $t = u$ or $u = t$, you can go on to write down $A(u)$.

1	$A(t)$	got this somehow...
2	\vdots	yatter yatter yatter
3	$t = u$... and this
4	$A(u)$	=sub(1, 3)

(Idea: if t, u are equal, there's no harm in replacing t by u as the value of x in A .)

220

Rules for equality

- Reflexivity of equality (refl).

Whenever you feel like it, you can introduce the sentence $t = t$, for any closed L -term t and for any L you like.

\vdots	bla bla bla
1	$t = t$ refl

(Idea: any L -structure makes $t = t$ true, so this is sound.)

219

Examples with equality...

Show $c = d \vdash d = c$. (c, d are constants.)

1	$c = d$	given
2	$d = d$	refl
3	$d = c$	=sub(2, 1)

This is often useful, so make it a derived rule:

1	$c = d$	given
2	$d = c$	=sym(1)

221

More examples with equality...

Show $\vdash \forall x \exists y (y = f(x))$.

1	c	$\forall I$ const
2	$f(c) = f(c)$	refl
3	$\exists y (y = f(c))$	$\exists I(2)$
4	$\forall x \exists y (y = f(x))$	$\forall I(1, 3)$

English: For any object c , we have $f(c) = f(c)$ — $f(c)$ is the same as itself.

So for any c , there is something equal to $f(c)$ — namely, $f(c)$ itself!

So for any c , we have $\exists y (y = f(c))$.

Since c was arbitrary, we get $\forall x \exists y (y = f(x))$.

222

Final remarks

Now you've done sets, relations, and functions in other courses(?), here's what an L -structure M really is.

It consists of the following items:

- a non-empty set, $dom(M)$
- for each constant $c \in L$, an element $c^M \in dom(M)$
- for each n -ary function symbol $f \in L$, an n -ary function $f^M : dom(M)^n \rightarrow dom(M)$
- for each n -ary relation symbol $R \in L$, an n -ary relation R^M on $dom(M)$ — that is, $R^M \subseteq dom(M)^n$.

Recall for a set S , S^n is $\overbrace{S \times S \times \dots \times S}^{n \text{ times}}$.

Another name for a relation (symbol) is a *predicate (symbol)*.

224

Harder example

Show $\exists x \forall y (P(y) \rightarrow y = x), \quad \forall x P(f(x)) \vdash \exists x (x = f(x))$.

1	$\exists x \forall y (P(y) \rightarrow y = x)$	given
2	$\forall x P(f(x))$	given
3	$\forall y (P(y) \rightarrow y = c)$	ass
4	$P(f(c))$	$\forall E(2)$
5	$f(c) = c$	$\forall \rightarrow E(4, 3)$
6	$c = f(c)$	=sym(5)
7	$\exists x (x = f(x))$	$\exists I(6)$
8	$\exists x (x = f(x))$	$\exists E(1, 3, 7)$

English: assume there is an object c such that all objects a satisfying P (if any) are equal to c , and for *any* object b , $f(b)$ satisfies P .

Taking 'b' to be c , $f(c)$ satisfies P , so $f(c)$ is equal to c .

So c is equal to $f(c)$.

As $c = f(c)$, we obviously get $\exists x (x = f(x))$.

223

What we did (all can be in Xmas test!)

Propositional logic

- Syntax
 - Literals, clauses (see Prolog next term!)
- Semantics
- English–logic translations
- Arguments, validity
 - \dagger truth tables
 - direct reasoning
 - equivalences, \dagger normal forms
 - natural deduction

Classical first-order predicate logic

same again (except \dagger), plus

- Many-sorted logic
- Specifications, pre- and post-conditions (continued in Reasoning about Programs)

225

Some of what we didn't do...

- normal forms for first-order logic
- proof of soundness or completeness for natural deduction
- theories, compactness, non-standard models, interpolation
- Gödel's theorem
- non-classical logics, eg. intuitionistic logic, linear logic, modal & temporal logic
- finite structures and computational complexity
- automated theorem proving

Do the 2nd and 4th years for some of these.

Modern logic at research level

- Advanced computing uses classical, modal, temporal, and dynamic logics. Applications in AI, to specify and verify chips, in databases, concurrent and distributed systems, multi-agent systems, protocols, knowledge representation, ... Theoretical computing (complexity, finite model theory) need logic.
- In mathematics, logic is studied in *set theory*, *model theory*, including *non-standard analysis*, and *recursion theory*. Each of these is an entire field, with dozens or hundreds of research workers.
- In philosophy, logic is studied for its contribution to formalising truth, validity, argument, in many settings: eg, involving time, or other possible worlds.
- Logic provides the foundation for several modern theories in linguistics. This is nowadays relevant to computing.