Introduction to Formal Proof

Bernard Sufrin

Trinity Term 2018

3: Predicate Logic Semantics

Propositional logic has limits

- ▷ Consider these two arguments

  S: All software systems are human artefacts
  Some software systems are complex
  All complex human artefacts are incomprehensible
  ∴ Some software systems are incomprehensible

  L: All formalizations of logic are human artefacts
  Some formalizations are complex
  All complex human artefacts are incomprehensible
  ∴ Some formalizations of logic are incomprehensible

- ▷ What do they have in common?
- ▷ Are their conclusions justifiable from their premisses by Natural Deduction?
- ▷ Do you accept their premisses?

Informal predicate language: variables, predicates, quantifiers

- ▷ Atomic Propositions are insufficient to capture the details of the premisses/conclusions
- ▷ Propositional logic and language needs enriching
- ▷ Predicate Logic (sometimes called First Order Logic) is the simplest enrichment that is generally useful[1]
- ▷ It allows the formal expression of statements (sometimes involving "all" and "some") about the properties of things, and about individual things.
- ▷ We express these using *variables*, *predicates* (P, HA, C, I) and *quantifiers* $(\exists \_ \cdot \_, \forall \_ \cdot \_)$

  1: All P things are HA things: $\forall x \cdot P(x) \to HA(x)$
  2: Some P things are C things: $\exists x \cdot P(x) \wedge C(x)$
  3: All C things that are also HA things are I things: $\forall x \cdot C(x) \wedge HA(x) \to I(x)$
  ∴ Some P things are I things: $\therefore \exists x \cdot P(x) \wedge I(x)$

[1] outside of the study of logic itself, that is.

- ▷ A (unary) predicate is used to make statements about individual things.
  - ∘ $Black(x)$, $Strange(y)$, $Incomprehensible(z)$
- ▷ Binary predicates are used to make statements relating two individuals (they are sometimes called binary relations)
  - ∘ $MadeBy(x, y)$
  - ∘ $i < j$, k loves l, $x = y$ (Binary predicates used with infix notation)
- ▷ The theory admits predicates/relations of other arities, *e.g.* $CanFool(who, whom, when)$
- ▷ The resulting language is quite expressive, *e.g.*

$$\forall x \cdot (Integer(x) \to \exists y \cdot (Integer(y) \wedge x < y))$$

$$\forall p \cdot (Person(p) \to \exists t \cdot (Time(t) \wedge \forall p' \cdot Person(p') \to CanFool(p, p', t)))$$

$$\forall p \cdot (Person(p) \to \forall t \cdot (Time(t) \to \exists p' \cdot Person(p') \wedge CanFool(p, p', t)))$$

- ▷ The variables introduced by quantifiers connect places in the text of a formula
- ▷ The textual region in which a name symbolizes the same connector is called its *scope*

$$\underbrace{(Person(*) \to \underbrace{(Time(*) \to \underbrace{Person(*) \wedge CanFool(*, *, *)}_{\exists *}))}_{\forall *}}_{\forall *}$$

- ▷ The names don't matter formally

$$\underbrace{(Person(x) \to \underbrace{(Time(z) \to \underbrace{Person(y) \wedge CanFool(x, y, z)}_{\exists y}))}_{\forall z}}_{\forall x}$$

- ▷ But it's a good idea to use memorable ones

$$\underbrace{(Person(perp) \to \underbrace{(Time(t) \to \underbrace{Person(vict) \wedge CanFool(perp, vict, t)}_{\exists vict}))}_{\forall t}}_{\forall perp}$$

▷ When we want to make a predicate calculus argument about a "real-world" situation, we first:

- Say what our world is going to be
- Choose names for (some) individual things that in our world
- Choose names (or symbols) for predicates that (partly) describe the real world situation

▷ *e.g.* humans

- ♀(x) means "x is genetically female"
- ♂(x) means "x is genetically male"
- $Parent(x, y)$ means "x is a genetic parent of y."
- $Ancestor(x, y)$ means "x is a genetic ancestor of y."

▷ We can then encode statements about our world as predicate language statements, *e.g.*

- Everybody is either genetically male or genetically female, not both:

$$\forall b \cdot (\text{♂}(b) \vee \text{♀}(b)) \wedge \neg(\text{♂}(b) \wedge \text{♀}(b))$$

- Somebody is an ancestor of p if they are a parent of p, or the parent of an ancestor of p.

$$\forall b \cdot \forall p \cdot Ancestor(b, p) \leftrightarrow Parent(b, p) \vee \exists a \cdot Parent(b, a) \wedge Ancestor(a, p)$$

- Nobody is their own ancestor:

$$\neg \exists b \cdot Ancestor(b, b)$$

- Everybody has a unique genetic mother:

$$\forall b \cdot \exists m \cdot \begin{pmatrix} \text{♀}(m) \wedge Parent(m, b) & \wedge \\ \forall m' \cdot (\text{♀}(m') \wedge Parent(m', b) \rightarrow m = m') & \end{pmatrix}$$

Informal predicate language: functions

▷ So far we have only used variables (and names of individuals) to denote things

▷ This leads to some unwieldy circumlocutions – *esp.* about uniqueness

▷ It is more convenient to use function application notation to denote unique things

▷ Humans revisited

- ♀(x) means "x is genetically female"
- ♂(x) means "x is genetically male"
- $Parent(x, y)$ means "x is a genetic parent of y".
- $Ancestor(x, y)$ means "x is a genetic ancestor of y".
- $ma(x)$ denotes *the* genetic mother of x.
- $pa(x)$ denotes *the* genetic father of x.

▷ Facts: (given our present state of knowledge)

- $\forall x \cdot Parent(ma(x), x)$
- $\forall x \cdot Parent(pa(x), x)$
- $\forall x \cdot \text{♂}(pa(x)) \wedge \text{♀}(ma(x))$
- $\forall b \cdot (\text{♂}(b) \vee \text{♀}(b)) \wedge \neg(\text{♂}(b) \wedge \text{♀}(b))$
- $\forall b \cdot \forall p \cdot Ancestor(b, p) \leftrightarrow Parent(b, p) \vee \exists a \cdot Parent(b, a) \wedge Ancestor(a, p)$

- ▷ We may not have coded enough facts to model the world completely enough to prove from them something that's intuitively true, or that's useful. For example in our world of humans as currently encoded:
  - ∘ Can we *prove* that everyone has two distinct parents?
  - ∘ Can we *prove* that everyone has no more than two parents?

Inference systems for predicate logic

- ▷ The goal of an inference system for predicate logic is that if our coding of statements about the world is coherent and the encoded statements about the world are true, then any valid deductions we make from them should also be true in the world.

- ▷ Recall that the rigorous analysis of the soundness and completeness of propositional logic inference systems required us to characterise the truth of composite propositions independently of their provability in the inference system.

  In the same way, a rigorous analysis of predicate logic inference systems will require us to characterise a notion of the truth of composite and quantified statements about a world independently of their provability in the inference system.

- ▷ Such independent characterizations are known as the *semantics* of the logics concerned.

Formal predicate language: grammar

- ▷ There are two sorts of phrase in the language
  - ∘ Terms – denote objects / things
  - ∘ Formulæ – denote declarative statements about things
- ▷ We build phrases in the language from a given initial vocabulary[2] of
  - ∘ constant symbols ("names for individual things") $\mathcal{C}$
  - ∘ predicate symbols ("names of predicates") $\mathcal{P}$
  - ∘ function symbols ("names of functions") $\mathcal{F}$
- ▷ Each function and predicate symbol has an arity (unary, binary, ternary, ...) and "fixity" that is usually indicated informally, *e.g*:
  $\mathcal{C} = \{\mathbf{0}, \mathbf{1}\}$
  $\mathcal{F} = \{\mathsf{s}(\cdot), \mathsf{p}(\cdot), \cdot \oplus \cdot\}$
  $\mathcal{P} = \{\cdot = \cdot, \cdot > \cdot, Even\cdot\}$

[2] Sometimes called a *signature*

- ▷ Terms
  - ∘ Any variable is a term[3]
  - ∘ Any constant is a term
  - ∘ If $t_1, \ldots t_n$ are terms and f is an n-ary function symbol, then $f(t_1, \ldots t_n)$ is a term[4]
  - ∘ Nothing else is a term[5]
- ▷ For example: with $\mathcal{C}, \mathcal{F}, \mathcal{P}$ as on the previous page, the following are terms:

  $$\begin{array}{llllll} \mathbf{0} & \mathbf{1} & \mathsf{s}(\mathbf{0}) & \mathsf{s}(\mathsf{s}(\mathbf{0})) & \mathsf{p}(\mathsf{s}(\mathbf{1})) & \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathbf{0})) \oplus \mathsf{p}(\mathsf{s}(\mathbf{1}))) \\ x & y_0 & z' & \mathsf{s}(\mathsf{s}(y_1)) & \mathsf{s}(z') \oplus (y_1 \oplus x) & \mathsf{s}(\mathsf{s}(z')) \oplus \mathsf{p}(\mathsf{s}(\mathbf{1})) \end{array}$$

  but the following are not terms:

  $$a > \mathbf{0} \qquad Even(x)$$

  and these are not well formed phrases of any kind:

  $$x(x(y)) \qquad Even$$

[3] Here we use sequences of lowercase letters as variables – sometimes decorating them with subscripts and/or dashes.
[4] $x \circledast y$ is syntactic sugar for $\circledast(x, y)$ for every binary infix function symbol $\circledast$. The binding power of such symbols is specified with their fixity.
[5] We allow parentheses in written terms.

▷ Formulæ

- If $t_1, \ldots t_n$ are terms, and P is an n-ary predicate symbol, then $P(t_1, \ldots t_n)$ is a formula[6]

  (these are called **atomic** *formulæ)*

- If ϕ is a formula then so is $\neg\phi$
- If ϕ, ψ are formulæ then so are $\phi \wedge \psi, \phi \vee \psi, \phi \rightarrow \psi, \phi \leftrightarrow \psi$

  (priority[7] of the logical operators is as in propositional logic.

- If α is a variable, and ϕ is a formula then $\forall\alpha \cdot \phi$ and $\exists\alpha \cdot \phi$ are formulae[8]

  (these are called **quantified** *formulæ)*

- Nothing else is a formula[9]

▷ Choose one of the notations for quantification and stick to it:

- $\exists\alpha\cdot$ and $\forall\alpha\cdot$ take the largest well-formed formula (wff) that starts at the right of $\cdot$
- Without the "$\cdot$" $\exists\ \alpha$ and $\forall\ \alpha$ take the smallest wff that starts at the right of α

[6] $x \fallingdotseq y$ is syntactic sugar for $\fallingdotseq(x, y)$ for every binary infix predicate symbol $\fallingdotseq$

[7] Huth and Ryan call it "binding power"

[8] Huth and Ryan write $(\forall\alpha\phi)$ and $(\exists\alpha\phi)$. Others require a $\cdot$ between variable and formula.

[9] We allow parentheses in written formulæ.

Free Variables

▷ If t is a term then every variable v appearing in that term is *free* in t.

▷ If $P(t_1, \ldots, t_n)$ is an atomic formula then the variables *free* in that formula are the variables free in the terms $t_1, \ldots, t_n$.

▷ If ϕ is a formula then the variables free in $\neg\phi$ are the variables free in ϕ

▷ If ϕ, ψ are formulæ then the variables free in $\phi \wedge \psi, \phi \wedge \psi, \phi \rightarrow \psi, \phi \leftrightarrow \psi$, are the variables free in ϕ and the variables free in ψ.

▷ If ϕ is a formula, and α a variable, then the variables free in $\forall\alpha \cdot \phi$ and in $\exists\alpha \cdot \phi$ are the variables, except α, that are free in ϕ.

▷ Examples:

Term	Free
$\mathsf{s}(\mathsf{s}(\mathbf{0}))$	
$\mathsf{s}(\mathsf{s}(z')) \oplus \mathsf{p}(\mathsf{s}(\mathbf{1})) \oplus x$	z', x

Formula	Free
$x > y \rightarrow y > z \rightarrow x > z$	x, y, z
$\forall x \cdot y = \mathsf{s}(\mathsf{s}(z')) \oplus \mathsf{p}(\mathsf{s}(\mathbf{1})) \oplus x$	z', y
$\exists y \cdot \forall x \cdot y = \mathsf{s}(\mathsf{s}(z')) \oplus \mathsf{p}(\mathsf{s}(\mathbf{1})) \oplus x$	z'
$\forall z' \cdot \exists y \cdot \forall x \cdot y = \mathsf{s}(\mathsf{s}(z')) \oplus \mathsf{p}(\mathsf{s}(\mathbf{1})) \oplus x$	
$\exists x \cdot y$ loves $x \wedge (\exists y \cdot x$ loves $y)$	y

▷ Definition: a term/formula with no free variables is called a *closed* term/formula.

Substitution

▷ Definition: if ϕ is a formula, t_i are terms, and α_i are variables, the notation $\phi[t_1, \ldots, t_n/\alpha_1, \ldots, \alpha_n]$ means the formula obtained by simultaneously substituting t_i for every free occurence of α_i in ϕ. The phrase $[t_1, \ldots, t_n/\alpha_1, \ldots, \alpha_n]$ represents a mapping from variables to terms and is called a *substitution*.

▷ Examples:

Formula	Substitution	Equivalent
$(x > y \rightarrow y > z \rightarrow x > z)$	$[\mathsf{s}(x)/x]$	$(\mathsf{s}(x) > y \rightarrow y > z \rightarrow \mathsf{s}(x) > z)$
$(y = \mathsf{s}(\mathsf{s}(z')) \oplus \mathsf{p}(\mathsf{s}(\mathbf{1})) \oplus x)$	$[y, x/x, y]$	$(x = \mathsf{s}(\mathsf{s}(z')) \oplus \mathsf{p}(\mathsf{s}(\mathbf{1})) \oplus y)$
$(y = \mathsf{s}(\mathsf{s}(z')) \oplus \mathsf{p}(\mathsf{s}(\mathbf{1})) \oplus x)$	$[y/x][x/y]$	$(x = \mathsf{s}(\mathsf{s}(z')) \oplus \mathsf{p}(\mathsf{s}(\mathbf{1})) \oplus x)$
$(\exists y \cdot z' = \mathsf{s}(y) \wedge \forall z' \cdot \mathsf{p}(\mathbf{1}) \oplus z' = z')$	$[\mathsf{p}(x)/z']$	$(\exists y \cdot \mathsf{p}(x) = \mathsf{s}(y) \wedge \forall z' \cdot \mathsf{p}(\mathbf{1}) \oplus z' = z')$
$(\exists x \cdot z' = \mathsf{s}(x) \wedge \forall z' \cdot \mathsf{p}(\mathbf{1}) \oplus z' = z')$	$[\mathsf{p}(x)/z']$	$(\exists x \cdot \mathsf{p}(x) = \mathsf{s}(x) \wedge \forall z' \cdot \mathsf{p}(\mathbf{1}) \oplus z' = z')$

▷ Notice that in the last example the x being substituted has been "captured" within the scope of $\exists x \cdot \ldots$. Such "captures" are explicitly prohibited by the formal definition of substitution.

Models and Meanings

- ▷ Knowing only a *signature* $\mathcal{C}, \mathcal{F}, \mathcal{P}$ can tell us nothing about the truth or falsehood of its atomic formulæ.
- ▷ To establish the truth or falsehood of statements denoted by atomic formulæ we need to have a *model*. This tells us:
  - ∘ what the *domain of discourse*[10] is (what world the things come from)
  - ∘ what thing each constant symbol denotes in that world
  - ∘ what function each function symbol means in that world
  - ∘ what predicate each predicate symbol means in that world
- ▷ If M is a model for $\mathcal{C}, \mathcal{F}, \mathcal{P}$ we will write here
  - ∘ $M_{constant}(c)$ to mean the thing the symbol c denotes in the model's domain ($c \in \mathcal{C}$)
  - ∘ $M_{function}(f)$ to mean the function on the domain that the symbol f means ($f \in \mathcal{F}$)
  - ∘ $M_{predicate}(P)$ to mean the predicate on the domain that the symbol P means ($P \in \mathcal{P}$)

[10] Some authors use "Universe of Discourse", many shorten to "Domain"

☜15

- ▷ Example: we are given: $\mathcal{C} = \{\mathbf{0}\}$ $\mathcal{F} = \{\mathsf{s}(\cdot), \mathsf{p}(\cdot), \cdot \oplus \cdot\}$ $\mathcal{P} = \{\cdot = \cdot, \cdot > \cdot, Even\}$
- ▷ One obvious model (Int) has as its domain the integers, with:

  $\mathbf{0}$ denoting the number 0

  s denoting the successor function, and p its inverse

  $\cdot \oplus \cdot$ denoting integer addition

  $\cdot = \cdot, \cdot > \cdot, Even$ denoting the predicates they usually denote for integers
- ▷ Another model (Nat_3) has as its domain the natural numbers modulo 3, with:

  $\mathbf{0}$ denoting the number 0

  s denoting the successor function modulo 3 and p its inverse

  $\cdot \oplus \cdot$ denoting addition modulo 3

  $\cdot = \cdot, \cdot > \cdot, Even$ denoting the predicates they usually denote for natural numbers

The 7PM Model

- ▷ Another model has as its domain the last seven Prime ministers of the UK[11] with

  $\mathbf{0}$ denoting Cameron

  s(x) denoting the Prime Minister who took office immediately after x, and p(x) denoting the Prime Minister who took office immediately before x, and $x \oplus y$ denoting the Prime Minister who took office later.

  $\cdot = \cdot$ meaning identity, $\cdot > \cdot$ meaning "took office later than", and $Even$ being true ovv[12] Prime Ministers educated in state schools.
- ▷ We present this to show that there is no particular reason why the symbols used in a signature should have a mnemonic relationship to the domain, constants, functions and predicates assigned by a model.

[11] In reverse order of appointment they are: Camoron, Brown, Bliar, Major, Thatcher, Callaghan, Wilson.
[12] "Of and only of" (you read it here first, Virginia!)

Evaluation of formulae without variables

- ▷ To discover whether an atomic formula without variables is true:
  - ∘ First translate the formula into the model domain by translating:
    - ∗ every constant symbol into what it denotes in the model
    - ∗ every function symbol into the function it means for the model
    - ∗ every predicate symbol into the predicate it means for the model
  - ∘ Then evaluate the translated atomic formula (in the model domain)
- ▷ For example, in 7PM:

$\mathbf{0}$	>	p	($\mathbf{0}$)
Cameron	took office later than	the immediate predecessor in office of	*Cameron*
Cameron	took office later than	*Brown*	

- ▷ Composite non-quantified formulae without variables ($\neg\phi, \phi \vee \psi, \phi \wedge \psi, \ldots$) are evaluated by evaluating their components, and combining the component values as they would be combined in propositional logic.

Evaluation of Quantified Formulae in the 7PM model

- ▷ Is the formula $\exists x \cdot x > \mathbf{0}$ true?
- ▷ In order to discover this we associate each thing in the 7PM domain with x, then evaluate the formula: $x > \mathbf{0}$. If it is true for any of these formulæ, then the whole formula is true.
- ▷ Is the formula $\forall y \cdot y \neq \mathbf{0} \rightarrow \exists x \cdot x > y$ true?
- ▷ In order to discover this we associate each thing in the 7PM domain with y, then evaluate the formula: $y \neq \mathbf{0} \rightarrow \exists x \cdot x > y$. If it is true for all of these formulæ, then the whole formula is true..

Formalizing “associate with”

- ▷ One way to formalise the idea of “associate ... with x then evaluate ...” is:
  - ∘ *invent* a constant symbol (say $DC, GB, AB, JM, MT, JC, HW$) for each thing
  - ∘ Substitute each constant symbol for x in $x > \mathbf{0}$, then evaluate *the resulting formula-without-variables*:

Formula	Translation into the model	Value
$DC > \mathbf{0}$	*Cameron* took office later than *Cameron*	F
$GB > \mathbf{0}$	*Brown* took office later than *Cameron*	F
$AB > \mathbf{0}$	*Bliar* took office later than *Cameron*	F
$JM > \mathbf{0}$	*Major* took office later than *Cameron*	F
$MT > \mathbf{0}$	*Thatcher* took office later than *Cameron*	F
$JC > \mathbf{0}$	*Callaghan* took office later than *Cameron*	F
$HW > \mathbf{0}$	*Wilson* took office later than *Cameron*	F

  - ∘ So in 7PM, $\exists x \cdot \phi(x)$ means the same as $\phi(DC) \vee \phi(GB) \vee \ldots \vee \phi(HW)$ and $\forall x \cdot \phi(x)$ means the same as $\phi(DC) \wedge \phi(GB) \wedge \ldots \wedge \phi(HW)$

☜16 ☜17

- ▷ To discover the value of $\forall y \cdot y \neq \mathbf{0} \rightarrow \exists x \cdot x > y$ we need to evaluate a substitution-for-y instance of the formula $y \neq \mathbf{0} \rightarrow \exists x \cdot x > y$ for each of our invented constant symbols.

$$
\begin{gathered}
DC \neq \mathbf{0} \rightarrow \exists x \cdot x > DC \\
GB \neq \mathbf{0} \rightarrow \exists x \cdot x > GB \\
\ldots \\
HW \neq \mathbf{0} \rightarrow \exists x \cdot x > HW
\end{gathered}
$$

- ▷ Writing $[\![\ldots]\!]$ for “value”, in the first of these we see

$$
\begin{aligned}
& [\![DC \neq \mathbf{0} \rightarrow \exists x \cdot x > DC]\!] \\
\rightsquigarrow\ & \text{(Evaluation rules for composite non-quantified formulæ)} \\
& [\![DC \neq \mathbf{0}]\!] \rightarrow [\![\exists x \cdot x > DC]\!] \\
\rightsquigarrow\ & \text{(Because } Cameron = Cameron\text{)} \\
& \mathrm{F} \rightarrow [\![\exists x \cdot x > DC]\!] \\
\rightsquigarrow\ & \text{(Definition of } \rightarrow\text{)} \\
& \mathrm{T}
\end{aligned}
$$

- ▷ As for the second (with subsequent formulæ similar) we have

$$
\begin{aligned}
& [\![GB \neq \mathbf{0} \rightarrow \exists x \cdot x > GB]\!] \\
\rightsquigarrow\ & [\![GB \neq \mathbf{0}]\!] \rightarrow [\![\exists x \cdot x > GB]\!] \\
\rightsquigarrow\ & \mathrm{T} \rightarrow [\![\exists x \cdot x > GB]\!] \\
\rightsquigarrow\ & [\![\exists x \cdot x > GB]\!] \\
\rightsquigarrow\ & [\![(DC > GB) \vee (GB > GB) \vee (AB > GB) \vee \ldots \vee (HW > GB)]\!] \\
\rightsquigarrow\ & Cameron \text{ took office later than } Brown\ \vee\ \ldots \\
\rightsquigarrow\ & \mathrm{T}
\end{aligned}
$$

Evaluation rules for formulae in a model M with domain Dom

▷ Our evaluation rules can be simplified if we assume that there is a distinct constant symbol corresponding to every distinct thing v in the domain Dom. Here we use the notation $\langle\langle v\rangle\rangle$ for the constant symbol corresponding to v. In other words: $M_{constant}(\langle\langle v\rangle\rangle)$ is v.

▷ Our assumption effectively means that we need not give rules for the evaluation of terms in which variables appear.

▷ Any metalogical reasoning we do under this assumption can also be done with a slightly more complex collection of rules that provide for the evaluation of terms with variables in them.[13]

[13] Note 9 on p37 gives a Haskell implementation of these rules.

▷ Domain values denoted by terms:

- If c is a constant symbol, then the value denoted by the term c is $M_{constant}(c)$ (so the value denoted by the constant symbol $\langle\langle v\rangle\rangle$ is v)
- If f is a function symbol, and $t_1, \ldots t_n$ are terms, then the value denoted by the term $f(t_1, \ldots t_n)$ is the value of the n-ary function $M_{function}(f)$ at the arguments $(v_1, \ldots v_n)$ where $v_1, \ldots v_n$ are the domain values denoted by the terms $t_1, \ldots t_n$.

▷ Truth values of formulæ without variables

- The formula $P(t_1, \ldots t_n)$ is true iff the n-ary predicate $M_{predicate}(P)$ holds at the n-tuple $(v_1, \ldots v_n)$ where $v_1, \ldots v_n$ are the domain values denoted by the terms $t_1, \ldots t_n$.

▷ Truth values of quantified formulæ.

- $\exists\alpha \cdot \phi$ is true iff $\phi[\langle\langle v\rangle\rangle/\alpha]$ is true for some domain value v
- $\forall\alpha \cdot \phi$ is true iff $\phi[\langle\langle v\rangle\rangle/\alpha]$ is true for every domain value v

▷ Truth values of nonquantified composite formulæ are determined by the truth-valued functions $\wedge, \vee, \rightarrow, \leftrightarrow, \neg$ at the truth values of their components.

▷ Writing $[\![t]\!]$ for "the domain value denoted by the term t", and $[\![\phi]\!]$ for "the truth value of the formula ϕ" we can tabulate the evaluation rules for variable-free terms and formulæ concisely.

$[\![c]\!]$	$M_{constant}(c)$
$[\![f(t_1, \ldots, t_n)]\!]$	$M_{function}(f)([\![t_1]\!], \ldots, [\![t_n]\!])$
$[\![P(t_1, \ldots, t_n)]\!]$	T if and only if $M_{predicate}(P)([\![t_1]\!], \ldots, [\![t_n]\!])$
$[\![\neg\phi]\!]$	$\neg[\![\phi]\!]$
$[\![\phi \wedge \psi]\!]$	$[\![\phi]\!] \wedge [\![\psi]\!]$
$[\![\phi \vee \psi]\!]$	$[\![\phi]\!] \vee [\![\psi]\!]$
$[\![\phi \rightarrow \psi]\!]$	$[\![\phi]\!] \rightarrow [\![\psi]\!]$
$[\![\phi \leftrightarrow \psi]\!]$	$[\![\phi]\!] \leftrightarrow [\![\psi]\!]$
$[\![\forall\alpha \cdot \phi]\!]$	T if and only if $[\![\phi[\langle\langle v\rangle\rangle/\alpha]]\!]$ is T for every domain value v
$[\![\exists\alpha \cdot \phi]\!]$	T if and only if $[\![\phi[\langle\langle v\rangle\rangle/\alpha]]\!]$ is T for some domain value v

▷ Note: here we call the constant terms $\langle\langle v\rangle\rangle$ that correspond to domain values v, the constant terms *synthesized from the domain of* M.

☜18 ☜19

Semantic Entailment

Let $\phi_1, \ldots, \phi_n, \psi$, be formulæ in the predicate language determined by the signature $\mathcal{C}, \mathcal{F}, \mathcal{P}$ and let M be a model for that signature.

▷ We define

$$\phi_1, \ldots, \phi_n \models_M \psi$$

to mean that $\psi\mathcal{S}$ is true in M for every substitution $\mathcal{S}$ of synthesized constant terms (from the domain of M) that makes all of $\phi_1\mathcal{S}, \ldots, \phi_n\mathcal{S}$ true.

▷ We define

$$\phi_1, \ldots, \phi_n \models \psi$$

to mean that *in every such model* M $\phi_1, \ldots, \phi_n \models_M \psi$

▷ It can easily be proven that the only substitutions, S, that are *relevant* to the truth value of a formula are those that assign only to its free variables. This permits the mechanical checking of $\vDash_M$ when M has a finite domain.

▷ For example $\vDash_{\mathsf{Nat}_3} \mathsf{s}(i) \oplus j = \mathsf{s}(i \oplus j)$

Here $n = 2$ and the free variables are i, j, so the relevant substitutions are:

$$[\langle\!\langle 0\rangle\!\rangle, \langle\!\langle 0\rangle\!\rangle/i, j], [\langle\!\langle 0\rangle\!\rangle, \langle\!\langle 1\rangle\!\rangle/i, j], [\langle\!\langle 0\rangle\!\rangle, \langle\!\langle 2\rangle\!\rangle/i, j], \ldots,$$
$$[\langle\!\langle 2\rangle\!\rangle, \langle\!\langle 0\rangle\!\rangle/i, j], [\langle\!\langle 2\rangle\!\rangle, \langle\!\langle 1\rangle\!\rangle/i, j], [\langle\!\langle 2\rangle\!\rangle, \langle\!\langle 2\rangle\!\rangle/i, j]$$

▷ What about

- $i \neq \mathbf{0}, j \neq \mathbf{0} \vDash_{\mathsf{7PM}} \mathsf{s}(i) \oplus j = \mathsf{s}(i \oplus j)$?
- $\vDash_{\mathsf{Int}} \mathsf{s}(i) \oplus j = \mathsf{s}(i \oplus j)$?

▷ $\vDash_M$ cannot be comprehensively checked mechanically for models over non-finite domains.

Satisfiability

▷ Definition: ϕ is *satisfiable* if there is some model $\mathcal{M}$ for which $\vDash_{\mathcal{M}} \phi$

ASIDE: Partial Functions

▷ In the 7PM model the predecessor/successor-in-office functions are *partial*.

▷ This raises the spectre of *terms in the language that do not denote anything in the model*

Two examples are:

$$\mathsf{s}(\mathbf{0}), \mathsf{p}(\mathsf{p}(\mathsf{p}(\mathsf{p}(\mathsf{p}(\mathsf{p}(\mathsf{p}(\mathbf{0})))))))$$

▷ This affects[14] the semantics. For example:

- what is the truth-value of $\forall x \cdot \neg(x = \mathsf{s}(\mathbf{0}))$?
- what is the truth-value of $\exists y \cdot \forall x \cdot \neg(x = \mathsf{s}(y))$?

▷ Discussing the consequences of this in detail is beyond the scope of this course, so we will stick to models in which function symbols denote total functions.

[14] Perhaps we should say *infects*!

Contents

Note 1: On being a "thing" 2 ☞

In English natural language calling an entity a "thing" or "object" sometimes connotes materiality; *i.e.* that the entity has material substance. In these notes using these words does not have such connotations: "234" is a much a "thing" as "my foot"; and "the set of linguistically adept penguins" is as much a "thing" as the paper or screen on which you are reading these notes, or the file(s) in which the pdf representations of the notes are stored in the Lab's filestore.[15] What's more, when we call a thing a thing it does not mean that we believe that it has no internal structure.

Note 2: 3 ☞

Some theoretical studies of predicate logic omit the equality/identity relation at first – considering its detailed study to be of interest in its own right. We don't have sufficient time to do this, so when the time comes we shall simply write introduction and elimination rules for "=" that capture the consensus that is shared by everybody who needs to use logic in a practical way.

Note 3: Terms and Formulae represented in Haskell 13 ☞

Just as we did for propositions, we can define Haskell types to represent formulæ and terms.

```
type Var  = String
type Con  = String
type Pred = String
type Fun  = String

data Form     = Absurd
              | Sat Pred [Term]
              | Not Form
              | Form :/\ Form
              | Form :\/ Form
              | Form :-> Form
              | Form :<-> Form
              | Some Var Form
              | All  Var Form
              deriving (Eq)

data Term     = Var Var
              | Con Con
              | App Fun [Term]
              deriving (Eq)
```

The construction Sat $p\ [t_1, ...t_n]$ represents the formula $p(t_1, ...t_n)$, and App $f\ [t_1, ...t_n]$ represents the function application $f(t_1, ...t_n)$. Absurd will be used to represent a false formula. The intention of the other constructions should be clear.

[15] Incidentally, calling the Department of Computer Science "the Lab" is thought by some to be sentimental or archaic terminology, but the Lab is itself a "thing" despite some people being reluctant to use that name for it, and other people not knowing what "the Lab" refers to.

Notice that this representation has nothing to say about predicate (function) symbols being applied to the correct number of formulæ(terms). When we come to build an evaluator in Haskell we will simply assume that they are.

Note 4: Formal definition of single-variable substitution 16 ☞

The following table defines the substitution of t for x (for constant symbols c, function symbols f, predicate symbols P, variables α, x, and terms $t, t_1, ..., t_n$)

$$\begin{array}{lll}
c & [t/x] \rightsquigarrow & c \\
\alpha & [t/x] \rightsquigarrow & \begin{cases} t, \text{ if } \alpha \text{ is } x \\ \alpha, \text{ if } \alpha \text{ is not } x \end{cases} \\
f(t_1, ..., t_n) & [t/x] \rightsquigarrow & f(t_1[t/x], ..., t_n[t/x]) \\
P(t_1, ..., t_n) & [t/x] \rightsquigarrow & P(t_1[t/x], ..., t_n[t/x]) \\
\phi \wedge \psi & [t/x] \rightsquigarrow & \phi[t/x] \wedge \psi[t/x] \\
... & & \\
\forall\alpha \cdot \phi & [t/x] \rightsquigarrow & \begin{cases} \forall\alpha \cdot \phi, & \text{if } x \text{ is not free in } \phi \text{ or } \alpha \text{ is } x \\ \forall\alpha.\phi[t/x], & \text{if } x \text{ is free in } \phi, \text{ and } \alpha \text{ not free in } t \end{cases} \\
\exists\alpha \cdot \phi & [t/x] \rightsquigarrow & \begin{cases} \exists\alpha \cdot \phi, & \text{if } x \text{ is not free in } \phi \text{ or } \alpha \text{ is } x \\ \exists\alpha.\phi[t/x], & \text{if } x \text{ is free in } \phi, \text{ and } \alpha \text{ not free in } t \end{cases}
\end{array}$$

The precondition: α is not free in t forbids variable capture when a substitution is taken through a quantifier. When this condition is met we (also) say that "α is fresh in t".

Evidently a precondition for $\phi[t/x]$ to be well-defined is that there is no quantified subformula of ϕ that breaks the freshness condition. In this case logicians say that "t is free (to be substituted) for x in ϕ" – nearly always omitting the parenthesized phrase. This omission can startle the unwary – for the different senses of the word "free" are signalled in only a very subdued way in a phrase like: "t is free for x in $\forall\alpha \cdot \phi$ if α is not free in t or x is not free in ϕ".

Note 5: 17 ☞

The context frequently makes it clear which of $M_{constant}, M_{function}, M_{predicate}$ we mean, and in those circumstances we will drop the subscript.

Many logicians (especially those writing only for logicians) use the notations $c^{\mathcal{M}}, f^{\mathcal{M}}, P^{\mathcal{M}}$ for the denotations of c, f, P in the model $\mathcal{M}$. This convention has the virtue of conciseness, but brings little else to the party!

Note 6: 22 ☞

The explanations of $\forall$ (and $\exists$) as a finite conjunction (and disjunction) of formulæ can only be used because the 7PM domain is *finite*. The precise meaning of the ellipsis (...) is evident to most people, and (because the domain is small enough) the ellipsis could be removed by writing the conjunction (disjunction) out in full. This mode of explanation doesn't work for a non-finite domain, and we have to use a more elaborate way of "eliminating" variables from quantified formulae.

Note 7: Models represented in Haskell 22 ☞

Just as we were able to define as a Haskell function the evaluation of propositions represented as Haskell data types, so we are able to define the evaluation of formulæ and terms as Haskell functions.

We first define a type class `Model` that captures the essence of a model. The type parameter, dom, will be instantiated in any particular model by the Haskell type that represents things in the domain of discourse.

```
class Model dom where
      constant::  Con  -> dom
      function::  Fun  -> [dom] -> dom
      predicate:: Pred -> [dom] -> Bool
      universe::  [dom]
```

the $constant, function, predicate$ functions map $Con, Fun, Pred$ symbols to values, functions and predicates. The universe is represented by the list $universe$ – a clue that our Haskell implementation of models will only be capable of computing effectively with models that have*finite universes*.

Here's the Nat_3 model in this encoding

```
instance Model Int where
  universe   = [0,1,2]
  constant  "0"          = 0
  function  "s"    [n]    = (n+1) `mod` 3
  function  "p"    [n]    = (n-1) `mod` 3
  function  "⊕"    [x,y]  = (x+y) `mod` 3
  predicate ">"    [x, y] =  x>y
  predicate "Even" [n]    = even n
  predicate "="    [x,y]  = x==y
```

and here's the $7PM$ model

```
data SevenPM = Wilson | Callaghan | Thatcher | Major |
               Bliar  | Brown     | Cameron
               deriving (Eq, Ord, Show, Enum)

instance Model SevenPM where
  universe     = [Wilson, Callaghan, Thatcher, Major,
                  Bliar,   Brown,    Cameron
                 ]

  constant  "0" = Cameron
  function  "s"    [n]    = succ n
  function  "p"    [n]    = pred n
```

```
function  "⊕"    [x,y]  = if x>y then x else y
predicate ">"    [x, y] = x>y
predicate "Even" [n]    = n /= Bliar || n /= Cameron
predicate "="    [x,y]  = x==y
```

Note 8: From substitution to environment semantics 27

A well-known technique in the implementation of languages for which substitution for variables plays a role in the semantics is to *avoid doing the work of substitution*. This work mostly consists of copying the parts of phrases that are *not* the variable being substituted for).

The technique that is used is to represent phrases by the usual data structures (in which variables can appear) together with an *auxiliary structure* (usually called an *environment*) which *records the substitutions that have been done so far*.

Substitutions for variables appear in exactly two of the evaluation rules given in the main body of the text, namely:

- ▷ $\exists\alpha\cdot\phi$ is true iff $\phi[\langle\langle v\rangle\rangle/\alpha]$ is true for some domain value v
- ▷ $\forall\alpha\cdot\phi$ is true iff $\phi[\langle\langle v\rangle\rangle/\alpha]$ is true for every domain value v

If α occurs free in ϕ, then the recursive evaluation of $\phi[\langle\langle v\rangle\rangle/\alpha]$ will eventually result in the application of the "quoted value" term rule:

- ▷ the value denoted by the constant symbol $\langle\langle v\rangle\rangle$ is v

Following from this, our environments map variables directly to domain values. For if, while applying the evaluation rules, the formula ϕ is evaluated with $\langle\langle v\rangle\rangle$ substituted for the variable α, then that formula will eventually be evaluated in an environment in which α is mapped to the domain value v.

In what follows environments ρ are mappings from variables to domain values, and we write $\rho\oplus\{\alpha\mapsto v\}$ ("extend ρ by mapping α to v") to mean the environment that is identical to ρ except that it maps the variable α to the domain value v.

Writing $[\![\ t\ ]\!]_\rho$ for "the domain value denoted by the term t in environment ρ", and $[\![\ \phi\ ]\!]_\rho$ for "the truth value of the formula ϕ in environment ρ" we can tabulate the evaluation rules for all terms and formulæ concisely.

$[\![\ \alpha\ ]\!]_\rho$	$\rho(\alpha)$ (for the variable α)
$[\![\ c\ ]\!]_\rho$	$M_{constant}(c)$
$[\![\ f(t_1,\ldots,t_n)\ ]\!]_\rho$	$M_{function}(f)([\![\ t_1\ ]\!]_\rho,\ldots,[\![\ t_n\ ]\!]_\rho)$
$[\![\ P(t_1,\ldots,t_n)\ ]\!]_\rho$	T if and only if $M_{predicate}(P)([\![\ t_1\ ]\!]_\rho,\ldots,[\![\ t_n\ ]\!]_\rho)$
$[\![\ \neg\phi\ ]\!]_\rho$	$\neg[\![\ \phi\ ]\!]_\rho$
$[\![\ \phi\wedge\psi\ ]\!]_\rho$	$[\![\ \phi\ ]\!]_\rho\wedge[\![\ \psi\ ]\!]_\rho$
$[\![\ \phi\vee\psi\ ]\!]_\rho$	$[\![\ \phi\ ]\!]_\rho\vee[\![\ \psi\ ]\!]_\rho$
$[\![\ \phi\rightarrow\psi\ ]\!]_\rho$	$[\![\ \phi\ ]\!]_\rho\rightarrow[\![\ \psi\ ]\!]_\rho$
$[\![\ \phi\leftrightarrow\psi\ ]\!]_\rho$	$[\![\ \phi\ ]\!]_\rho\leftrightarrow[\![\ \psi\ ]\!]_\rho$
$[\![\ \forall\alpha\cdot\phi\ ]\!]_\rho$	T if and only if $[\![\ \phi\ ]\!]_{\rho\oplus\{\alpha\mapsto v\}}$ is T for every domain value v
$[\![\ \exists\alpha\cdot\phi\ ]\!]_\rho$	T if and only if $[\![\ \phi\ ]\!]_{\rho\oplus\{\alpha\mapsto v\}}$ is T for some domain value v

Note 9: Evaluation rules in Haskell 27

We can easily translate our environment semantics into a *partial* Haskell implementation of the evaluation rules for predicate calculus formulæ and terms in a model whose domain type is dom. The implementation is partial because it will not terminate for false existential quantifications (or true universal quantifications) over non-finite domains.

```
thingValue:: Model dom => Env dom -> Term -> dom
truthValue:: Model dom => Env dom -> Form -> Bool

thingValue ρ term =
 case term of
  Var   α     -> ρ α
  Con   c     -> constant c
  App f terms -> (function f) (map (thingValue ρ) terms)

truthValue ρ form =
 case form of
   Sat  p terms     -> (predicate p) (map (thingValue ρ) terms)
   Some α φ         -> or  (map (φ ‘asFunOf‘ α) universe)
   All  α φ         -> and (map (φ ‘asFunOf‘ α) universe)
   Not φ            -> not(truthValue ρ φ)
   fl :/\ fr        -> truthValue ρ fl && truthValue ρ fr
   fl :\/ fr        -> truthValue ρ fl || truthValue ρ fr
   fl :-> fr        -> truthValue ρ fl → truthValue ρ fr
   fl :<-> fr       -> truthValue ρ fl == truthValue ρ fr
 where (φ ‘asFunOf‘ α) val = truthValue (extend ρ α val) φ
```

```
p → q = if p then q else True
```

An environment for a domain is a mapping from variables to domain values.

```
type Env dom = Var -> dom
```

The empty environment represents a situation in which no variables have been bound.

```
empty:: Env dom
empty α = error ("Trying to evaluate an unbound variable: "++α)
```

An environment ρ is extended to form a new environment by associating an additional variable with a domain value. The extended environment maps all other variables to the same values as the original did.

```
extend:: Env dom -> Var -> dom -> Env dom
extend ρ α v α' = if α'==α then v else ρ α'
```

Note 10: 31 ☞

There is a straightfoward method of transforming a model M in which some functions are partial into a model $\widehat{M}$ in which all functions are total.

- ▷ Augment the domain of M with a value: $undef$. Call the new domain $\widehat{Dom}$
- ▷ Corresponding to each n-ary function f of the original model, define a "totalized" function $\widehat{f}$ over $\widehat{Dom}^n$ by

$$\begin{array}{ll} \widehat{f}(v_1,\dots,v_n) = f(v_1,\dots,v_n), & \text{if } f \text{ defined at } (v_1,\dots,v_n) \\ \widehat{f}(v_1,\dots,v_n) = undef, & \text{otherwise} \end{array}$$

- ▷ Similarly, corresponding to each n-ary predicate P of the original model, define a predicate $\widehat{P}$ over $\widehat{Dom}^n$ by

$$\begin{array}{ll} \widehat{P}(v_1,\dots,v_n) = P(v_1,\dots,v_n), & \text{if } (v_1,\dots,v_n) \in Dom^n \\ \widehat{P}(v_1,\dots,v_n) = \mathtt{F}, & \text{otherwise} \end{array}$$

This straightfoward transformation will not, in general, preserve the truth values of formulae in the original model.