A Concise Introduction to the Fundamentals of Classical Logic

Brandon Bennett
School of Computing
University of Leeds

brandon@comp.leeds.ac.uk

September 2007

Section 1

Concepts and Uses of Logic

1.1 Logical Form

A *form* of an object is a structure or pattern which it exhibits. A *logical form* of a linguistic expression is an aspect of its structure which is relevant to its behaviour with respect to inference.

To illustrate this we consider a mode of inference which has been recognised since ancient times (the Romans called it *modus ponens*). An example of this type of inference is the following:

If Leeds is in Yorkshire then Leeds is in the UK
Leeds is in Yorkshire

Therefore, Leeds is in the UK

The statements above the line are the *premisses* (or assumptions) upon which the inference depends and the statement below the line is the *conclusion*.

The logical form of an inference is obtained by abstracting away from the specific content of the statements involved so that the essential pattern of the inference is shown. This can be done by replacing complex expressions whose structure is irrelevant to the argument by single letters. Then for this example we shall have:

If P then Q
P

Therefore Q

Examination of inference patterns has revealed that certain constructs such as 'If ... then ...' are of particular significance. To represent logical form in a clear and unambiguous way these constructs are represented by special symbols. In this case the usual representation is:

$(P \to Q)$
P

Q

(The 'therefore' has now been dropped — it can be regarded as the meaning of the horizontal bar.)

1.2 Propositions

The preceding argument can be explained in terms of *propositional* logic.

A proposition is an expression of a fact.

The symbols, P and Q, represent propositions and the logical symbol ' $\rightarrow$ ' is called a propositional connective.

Many systems of propositional logic have been developed. We shall be studying *classical* — i.e. the best established — propositional logic. In classical propositional logic it is taken as a principle that:

Every proposition is either *true* or *false* and not both.

Complex Propositions and Connectives

Propositional logic deals with inferences governed by the meanings of propositional *connectives*. These are expressions which operate on one or more propositions to produce another more complex proposition. The connectives dealt with in standard propositional logic correspond to the natural language constructs:

- '... and ...',
- '... or ...'
- 'it is not the case that...'
- 'if ... then ...'.

Symbols for the Connectives

The propositional connectives are represented by the following symbols:

and	$\wedge$	$(P \wedge Q)$
or	$\vee$	$(P \vee Q)$
if ... then	$\rightarrow$	$(P \rightarrow Q)$
not	$\neg$	$\neg P$

More complex examples: $((P \wedge Q) \vee R)$, $(\neg P \rightarrow \neg(Q \vee R))$

Brackets prevent ambiguity which would otherwise occur in a formula such as '$P \wedge Q \vee R$'.

Propositional Formulae

We can precisely specify the well-formed formulae of propositional logic by the following (recursive) characterisation:

- Each of a set, $\mathcal{P}$, of propositional constants P_i is a formula.
- If α is a formula so is $\neg\alpha$.
- If α and β are formulae so is $(\alpha \wedge \beta)$.

- If α and β are formulae so is $(\alpha \vee \beta)$.
- If α and β are formulae so is $(\alpha \rightarrow \beta)$.

The propositional *connectives* $\neg$, $\wedge$ $\vee$ and $\rightarrow$ are respectively called *negation*, *conjunction*, *disjunction* and *implication*.

1.3 Proposition Symbols and Schematic Variables

The symbols P, Q etc. occurring in propositional formulae should be understood as abbreviations for actual propositions such as 'It is Tuesday' and 'I am bored'. But in defining the class of propositional formulae I used Greek letters (α and β) to stand for arbitrary propositional formulae. These are called *schematic variables*. Schematic variables are used to refer *classes* of expression sharing a common form. Thus they can be used for describing patterns of inference.

For example, the formulae $(A \rightarrow B)$, $((P \wedge Q) \rightarrow R)$, $(G \rightarrow (P \wedge \neg G))$ and $(F \rightarrow F)$ are all instances of the form $(\alpha \rightarrow \beta)$ — and hence can all take part in *modus ponens* inferences.

Note that if a schematic variable occurs more than one in a formula or argument pattern, an instance must match each occurrence with an identical expression (so e.g. $(P \rightarrow Q)$ is *not* an instance of $(\alpha \rightarrow \alpha)$); however, different schematic variables may match identical expressions in an instance (so $(P \rightarrow P)$ is an instance of $(\alpha \rightarrow \beta)$.

(Strictly speaking, in the first section, when I exhibited the 'logical form' of a *modus ponens* argument, I should have used schematic letters. But I think that making this rather subtle distinction right at the beginning would not have been very helpful.)

1.4 From Natural Language to Propositional Logic

In translating a natural language such as English into propositional logic we must be aware of the meanings of all the propositional connectives and be able to recognise them amid the various idiomatic forms of expression used in the language.

Use of the English connectives 'and' and 'or' corresponds quite well with its use in logic; however, we must be aware that these words often occur connecting expressions which are not propositions. In such cases we need to rephrase the sentence in a form where they do connect propositions. For instance 'John loves Mary and Susan' must be rendered as 'John loves Mary and John loves Susan'.

'If ... then ...' constructions are quite easy to spot; but note that the form 'α if β' means 'If β then α' rather than 'if α then β'.

Assertions of the forms 'α if and only if β' and 'α just in case β' are both equivalent to 'If α then β and if β then α'.

A sentence of the form α otherwise β' means the same as $If not \alpha$ then β'; and finally sentences of the form 'If α then β otherwise γ' should be regarded as meaning 'If α then β and if not α then γ'.

Section 2

Inference Rules

2.1 Patterns of Deduction

An inference rule characterises a pattern of valid deduction. In other words, it tells us that if we accept as true a number of propositions — called premisses — which match certain patterns, we can deduce that some further proposition is true — this is called the conclusion. Thus we have seen that from two propositions with the forms $\alpha \to \beta$ and α we can deduce β. We also know that inferences from $(P \to Q)$ and P to Q or from $((A \wedge B) \to \neg C)$ and $(A \wedge B)$ to $\neg C$ (*etc.*) are of this form.

An inference rule can be regarded as a special type of re-write rule: one that preserves the truth of formulae — i.e. if the premisses are true so is the conclusion.

Some Simple Examples

Here are some other simple examples of inference forms:

'And' Elimination

$$\frac{\alpha \wedge \beta}{\alpha} \qquad \frac{\alpha \wedge \beta}{\beta}$$

'And' Introduction

$$\frac{\alpha \qquad \beta}{\alpha \wedge \beta}$$

'Or' Introduction

$$\frac{\alpha}{\alpha \vee \beta} \qquad \frac{\alpha}{\beta \vee \alpha}$$

'Not' Elimination

$$\frac{\neg\neg\alpha}{\alpha}$$

2.2 Logical Arguments and Proofs

We must now be more precise about what is meant by a logical *argument*. A logical *argument* consists of a set of propositions $\{P_1, \ldots, P_n\}$ called premisses and a further proposition C, the conclusion.

Notice that in speaking of an argument we are not concerned with any sequence of inferences by which the conclusion is shown to follow from the premisses. Such a sequence is called a *proof*.

A set of inference rules constitutes a *proof system*. Inference rules specify a class of primitive arguments which are justified by a single inference rule. All other arguments require proof by a series of inference steps.

2.2.1 A 2-Step Proof Example

Suppose we know that 'If it is Tuesday or it is raining John stays in bed all day' then if we also know that 'It is Tuesday' we can conclude that 'John is in Bed'.

Using T, R and B to stand for the propositions involved, this conclusion could be proved in propositional logic as follows:

$$\frac{((T \vee R) \rightarrow B) \qquad \dfrac{T}{(T \vee R)}}{B}$$

Here we have used the 'or introduction' rule followed by good old *modus ponens*

2.2.2 Theorems

A *theorem* of a proof system is a formula which can be proved true without assuming any premisses to be true. For example every formula of the form $(\alpha \wedge \neg\alpha)$ is a theorem of classical propositional logic.

2.3 Provability and Validity

2.3.1 Provability

To assert that C can be proved from premisses $\{P_1, \ldots, P_n\}$ in a proof system S we write:

$$P_1, \ldots, P_n \ \vdash_S \ C$$

This means that C can be derived from the formulae $\{P_1, \ldots, P_n\}$ by a series of inference rules in the proof system S. When it is clear what system is being used we may omit the subscript S on the '$\vdash$' symbol.

2.3.2 Validity

An argument is called *valid* if its conclusion is a consequence of its premisses. Otherwise it is invalid. This needs to be made more precise:

One definition of validity is: An argument is valid if it is not possible for its premisses to be true and its conclusion is false.

Another is: in all possible circumstances in which the premisses are true, the conclusion is also true.

To assert that the argument from premisses $\{P_1, \ldots, P_n\}$ to conclusion C is valid we write:

$$P_1, \ldots, P_n \ \models \ C$$

2.3.3 Provability *vs* Validity

We have defined **provability** as a property of an argument which depends on the inference rules of a logical proof system. **Validity** on the other hand is defined by appealing directly to the meanings of formulae and to the circumstances in which they are true or false.

Later we shall look in more detail at the relationship between validity and provability. This relationship is of central importance in the study of logic. To characterise validity we shall need some precise specification of the 'meanings' of logical formulae. Such a specification is called a *formal semantics*.

Section 3

Predicate Logic

Predicate logic is also commonly called first-order logic. It extends the expressive capabilities of propositional logic by representing *relations* and introducing logical *quantifiers*.

3.1 Relations

In propositional logic the smallest meaningful expression that can be represented is the proposition. However, even atomic propositions (those not involving any propositional connectives) have internal logical structure.

In *predicate* logic atomic propositions are analysed as consisting of a number of *individual constants* (i.e. names of objects) and a *predicate*, which expresses a *relation* between the objects.

$$\mathsf{R}(\mathsf{a}, \mathsf{b}, \mathsf{c}) \qquad \mathsf{Loves}(\mathsf{john}, \mathsf{mary})$$

With many binary (2-place) relations the relation symbol is often written between its operands — e.g. $4 > 3$. Unary (1-place) relations are also called properties — $\mathsf{Tall}(\mathsf{tom})$.

3.1.1 Equality

A particularly significant relation is that of *equality*. Rather than expression this in the usual notational form for relations ($\mathsf{Equals}(x, y)$), this is usually written using the mathematical notation $x = y$.

3.2 Quantification

Useful information often takes the form of statements of general property of entities. For instance, we may know that 'every dog is a mammal'. Such facts are represented in predicate logic by means of *universal quantification*.

Given a complex formula such as $(\mathsf{Dog}(\mathsf{spot}) \rightarrow \mathsf{Mammal}(\mathsf{spot}))$, if we remove one or more instances of some individual constant we obtain an incomplete expression

$(\mathsf{Dog}(\ldots) \rightarrow \mathsf{Mammal}(\ldots))$, which represents a (complex) property. To assert that this property holds of all entities we use the notation:

$$\forall x[\mathsf{Dog}(x) \rightarrow \mathsf{Mammal}(x)]$$

in which '$\forall$' is the universal quantifier symbol and x is a *variable* indicating which property is being quantified.

3.2.1 An Argument Involving Quantification

An argument such as:

> Everything in Yorkshire is in the UK
> Leeds is in Yorkshire
>
> ---
>
> Therefore Leeds is in the UK

can now be represented as follows:

$$\begin{array}{l} \forall x[\mathsf{Inys}(x) \rightarrow \mathsf{Inuk}(x)] \\ \mathsf{Inys}(l) \\ \hline \mathsf{Inuk}(l) \end{array}$$

3.2.2 Existential Quantification

As well as being able to express the fact that every entity x satisfies a given condition $\phi(x)$, it is also useful to be able say that *there exists at least one entity satisfying* ϕ.

To do this we use the *existential quantifier* to construct formulae of the form $\exists x[\phi(x)]$. For example,

$$\exists n[\mathsf{Number}(n) \wedge \mathsf{Prime}(n) \wedge (n > 1,000,000)]$$

3.2.3 Interdefinability of Universal and Existential Quantification

The two forms of quantification are logically very closely related. In fact universal quantification can be *defined* in terms of existential quantification; and conversely existential quantification can be defined in terms of universal.

In saying that one logical symbol can be defined in terms of some other symbols, we mean that in place of the definable symbol, we can write down an equivalent expression involving the other symbols. In the case of the two quantifiers, we can write the following definitions:

$$\exists x[\phi(x)] \equiv_{\text{def}} \neg\forall x[\neg\phi(x)]$$

$$\forall x[\phi(x)] \equiv_{\text{def}} \neg\exists x[\neg\phi(x)]$$

3.3 Translating Natural Language into Predicate Logic

The predicate logic representation allows us to show much more of the structure of natural language statements and to account for many more types of inference than propositional logic. However, translating into predicate logic is a little more complicated. Whilst the idea of a *relation* and its correspondence to natural language forms of expression is fairly straightforward, the use of quantifiers is much more difficult to master because the logical representation of quantifiers is rather different from the way they are expressed in natural language. The reason for this is that the logical representation is designed to be unambiguous, whereas natural languages have evolved for ease of communication and often contain many constructs which are potentially ambiguous (this ambiguity is in practice very often removed by the context of a statement).

To translate from English into predicate logic one must first have a clear understanding of the logical representation and what it is supposed to mean. Then given some English sentence one tries to rephrase it so as to preserve the meaning but state it in a way that is closer to the logical quantifier representation. Thus the sentence 'Every dog barks' could be stated (rather unnaturally) as 'For every entity x, if x is a dog then x barks'. This would then be represented as:

$$\forall x[\mathsf{dog}(x) \rightarrow \mathsf{barks}(x)]$$

Notice how in translating a statement of the form 'Every α βs' or perhaps 'Every α is a β', we not only use the universal quantifier but also introduce an implication, rendering the sentence as 'For every x, if $\alpha(x)$ then $\beta(x)$'.

With existential statements the logical representation again departs from natural language forms of expression. A statement 'There is a black dog' is rendered as 'There is some entity x, such that x is black and x is a *dog*:

$$\exists x[\mathsf{black}(x) \wedge \mathsf{dog}(x)]$$

Here the extra connective 'and' is used to assert two properties of an entity, whereas in English they are simply combined in the expression 'black dog'.

Every red block is on a blue block:

$$\forall x[(\mathsf{red}(x) \wedge \mathsf{block}(x)) \rightarrow \exists y[\mathsf{blue}(y) \wedge \mathsf{block}(y) \wedge \mathsf{on}(x, y)]]$$

Notice here that the existential quantifier conveyed very subtly in the English statement — the existential construct about 'blue blocks' is expressed simply by referring to them in a certain way.

We could investigate at great length the nuances of translation into predicate logic but the underlying principle is always to think of the meaning of a sentence rather than its structure and then try to capture this meaning in the logical representation.

You will be expected to be able to translate sentences of a similar complexity to the last example but no more complex than that.

Section 4

Uses of Logic

Logic has always been important in philosophy and in the foundations of mathematics and science. Here logic plays a foundational role: it can be used to check consistency and other basic properties of precisely formulated theories.

In computer science, logic can also play this role — it can be used to establish general principles of computation; but it can also play a rather different role as a 'component' of computer software: computers can be programmed to carry out logical deductions. Such programs are called *Automated Reasoning* systems.

4.1 Formal Specification of Hardware and Software

Since logical languages provide a flexible but very precise means of description, they can be used as specification language for computer hardware and software. A number of tools have been developed which help developers go from a formal specification of a system to an implementation. However, it must be realised that although a system may completely satisfy a formal specification it may still not behave as intended — there may be errors in the formal specification.

4.2 Formal Verification

As well as being used for specifying hardware or software systems, a logical description can be used to verify properties of systems.

If Θ is a set of formulae describing a computer system and π is a formula expressing a property of the system that we wish to ensure (eg. π might be the formula $\forall x[\mathsf{Employee}(x) \rightarrow \mathsf{age}(x) > 0]$), then we must verify that:

$$\Theta \models \pi$$

We can do this using a proof system S if we can show that:

$$\Theta \vdash_S \pi$$

4.3 Logical Databases

A set of logical formulae can be regarded as a *database*.

A logical database can be *queried* in a very flexible way, since for any formula ϕ, the semantics of the logic precisely specify the conditions under which ϕ is a consequence of the formulae in the database.

Often we may not only want to know whether a proposition is true but to find all those entities for which a particular relation, ρ, holds.

e.g.

query: $\rho(x, y, z)$?
Ans: $\langle x = \text{actress}, y = \text{bishop}, z = \text{cauliflower}\rangle$
or $\langle x = \text{octopus}, y = \text{whelk}, z = \text{sea-cucumber}\rangle$

4.4 Logic and Intelligence

The ability to reason and draw consequences from diverse information may be regarded as fundamental to *intelligence*. As the principal intention in constructing a logical language is to precisely specify correct modes of reasoning, a logical system (i.e. a logical language plus some proof system) might in itself be regarded as a form of *Artificial Intelligence*. However, formal logical inference seems to have a very different character to everyday commonsense reasoning.

These issues go beyond the scope of this course but they should certainly be considered by anyone seriously interested in AI.

Section 5

The Sequent Calculus

A sequent calculus is a proof system of a particular form. Its rules do not operate on ordinary formulae but on special expressions called *sequents*. The use of sequents allows all the necessary rules of inference to be stated in a uniform way. Sequent calculi are very convenient from the point of view of the use of logic in computer science: they are easy to implement in a language such as Prolog and they facilitate systematic methods for proving (or disproving) logical arguments.

5.1 Sequents

A sequent is an expression of the form:

$$\alpha_1,\ \ldots,\ \alpha_m \vdash\ \beta_1,\ \ldots,\ \beta_n$$

(where all the αs and βs are logical formulae).

This asserts that:

If all the αs is true then at least one of the βs is true.

This notation — as we shall soon see — is very useful in presenting inference rules in a concise and uniform way.

The sequent is equivalent to a formula of the form:

$$((\alpha_1 \wedge\ \ldots\ \wedge \alpha_m)\ \rightarrow\ (\beta_1 \vee \ldots \vee\ \beta_n))$$

The use of the '$\vdash$' symbol — as above — is the most common notation used for writing a sequent. However it is a bit confusing because the '$\vdash$' is also used to denote the relation of provability holding between a set of premisses and a conclusion; but a sequent is probably best understood as a special kind of formula.

5.1.1 Special forms of sequent

A sequent with a single formula on the r.h.s.:

$$\alpha_1,\ \ldots,\ \alpha_m \vdash \beta$$

Corresponds to a *logical argument* in the sense that it asserts that if all the αs are true then β is true. The sequent is thus true if the argument with premisses $\alpha_1, \ldots, \alpha_m$ and conclusion β is valid.

A sequent with an empty left-hand side:

$$\vdash \ \beta_1, \ \ldots, \ \beta_n$$

asserts that at least one of the βs is provable without assuming any premisses to be true. Such a formula is called a *logical theorem*.

A sequent with an empty right-hand side:

$$\alpha_1, \ \ldots, \ \alpha_m \vdash$$

asserts that the set of premisses $\{\alpha_1, \ \ldots, \ \alpha_m\}$ is inconsistent, which means that a formula of the form $(\beta \wedge \neg\beta)$ is provable from this set.

5.2 Sequent Calculus Inference Rules

A sequent calculus inference rule specifies a pattern of reasoning in terms of sequents rather than formulae.

Eg. a sequent calculus 'and introduction' is specified by:

$$\frac{\Gamma \ \vdash \alpha, \ \Delta \quad \textit{and} \quad \Gamma \ \vdash \beta, \ \Delta}{\Gamma \ \vdash (\alpha \wedge \beta), \ \Delta} \ [\vdash \ \wedge]$$

where Γ and Δ are any series of formulae.

In a sequent calculus we also have rules which introduce symbols into the premisses:

$$\frac{\alpha, \ \beta, \ \Gamma \ \vdash \ \Delta}{(\alpha \wedge \beta), \ \Gamma \ \vdash \ \Delta} \ [\wedge \ \vdash]$$

5.3 Sequent Calculus Proof Systems

To assert that a sequent is provable in a sequent calculus system, SC, I shall write:

$$\vdash_{\mathrm{SC}} \ \alpha_1, \ \ldots, \ \alpha_m \vdash \ \beta_1, \ \ldots, \ \beta_n$$

Construing a proof system in terms of the provability of sequents allows for much more uniform presentation than can be given in terms of provability of conclusions from premisses.

We start by stipulating that all sequents of the form

$$\alpha, \ \Gamma \ \vdash \ \alpha, \ \Delta$$

are immediately provable. It is clear that if α and some other formulae Γ are true then it must be the case that α or at least one of some formulae Δ must be true. Thus any sequent in which the same formulae occurs on both sides of the '$\vdash$' is valid.

We then specify how each logical symbol can be introduced into the left and right sides of a sequent (see next slide).

5.4 A Propositional Sequent Calculus

Axiom: $\alpha, \Gamma \vdash \alpha, \Delta$ **Re-write**: $\alpha \to \beta \implies \neg\alpha \vee \beta$

Rules:

$$\frac{\alpha,\ \beta,\ \Gamma \vdash \Delta}{(\alpha \wedge \beta),\ \Gamma \vdash \Delta}\,[\wedge \vdash] \qquad \frac{\Gamma \vdash \alpha,\ \Delta \quad \textit{and} \quad \Gamma \vdash \beta,\ \Delta}{\Gamma \vdash (\alpha \wedge \beta),\ \Delta}\,[\vdash \wedge]$$

$$\frac{\alpha,\ \Gamma \vdash \Delta \quad \textit{and} \quad \beta,\ \Gamma \vdash \Delta}{(\alpha \vee \beta),\ \Gamma \vdash \Delta}\,[\vee \vdash] \qquad \frac{\Gamma \vdash \alpha,\ \beta,\ \Delta}{\Gamma \vdash (\alpha \vee \beta),\ \Delta}\,[\vdash \vee]$$

$$\frac{\Gamma \vdash \alpha,\ \Delta}{\neg\alpha,\ \Gamma \vdash \Delta}\,[\neg \vdash] \qquad \frac{\Gamma,\ \alpha \vdash \Delta}{\Gamma \vdash \neg\alpha,\ \Delta}\,[\vdash \neg]$$

(**Exercise:** Show that the rules $[\vee \vdash]$ and $[\vdash \vee]$ can be replaced by the rewrite rule $\alpha \vee \beta \implies \neg(\neg\alpha \wedge \neg\beta)$.)

5.5 Sequent Calculus Proofs

The beauty of the sequent calculus system is its *reversibility*.

To test whether a sequent, $\Gamma \vdash \Delta$, is provable we simply apply the symbol introduction rules backwards. Each time we apply a rule, one connective is eliminated. With some rules two sequents then have to be proved (the proof branches) but eventually every branch will terminate in a sequent containing only atomic propositions. If all these final sequents are axioms, then $\Gamma \vdash \Delta$ is proved, otherwise it is not provable.

Note that the propositional sequent calculus rules can be applied in any order.

This calculus is easy to implement in a computer program.

5.6 Proof Examples

5.6.1 Example 1

This first example shows how a *modus ponens* inference is validated using the sequence calculus. The conclusion is an instance of a *modus ponens*-type argument:

$$\dfrac{\dfrac{\dfrac{\textit{axiom}}{P \vdash Q,\ P}}{\neg P,\ P \vdash Q}\,[\neg \vdash] \quad \textit{and} \quad \dfrac{\textit{axiom}}{Q,\ P \vdash Q}}{\dfrac{(\neg P \vee Q),\ P \vdash Q}{(P \to Q),\ P \vdash Q}\,[\to \text{r.w.}]}\,[\vee \vdash]$$

1. First the re-write rule [$\rightarrow$ r.w.] is applied to replace the '$\rightarrow$' formula with an equivalent disjunction (the equivalence of these formulae is explained in the section on formal semantics for propositional logic).

2. Next, the rule [$\vee \vdash$] is applied to eliminate the disjunction: to prove a sequent with a disjunction on the left, we have to prove each of the two sequents having just one of the disjuncts on the left. In this case one of these sequents is an 'axiom' since it contains Q on both sides.

3. Finally the [$\neg \vdash$] rule is applied. The $\neg P$ formula is moved over to the right and the negation eliminated. The result is another axiom — a sequent with P on both sides.

5.6.2 Example 2

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\textit{Axiom}}{P \ \vdash \ Q, \ P} \quad \textit{and} \quad \dfrac{\dfrac{\textit{Axiom}}{P, \ Q \ \vdash \ Q}}{P \ \vdash \ Q, \ \neg Q}\,[\vdash \neg]}{P \ \vdash \ Q, \ (P \wedge \neg Q)}\,[\vdash \wedge]}{\vdash \ \neg P, \ Q, \ (P \wedge \neg Q)}\,[\vdash \neg]}{\neg(P \wedge \neg Q) \ \vdash \ \neg P, \ Q}\,[\neg \vdash]}{\neg(P \wedge \neg Q) \ \vdash \ (\neg P \vee Q)}\,[\vdash \vee]}{\neg(P \wedge \neg Q) \ \vdash \ (P \rightarrow Q)}\,[\rightarrow \text{r.w.}]$$

5.6.3 Example 3

Now we shall look quite a long proof of the following sequent:

$$((P \vee Q) \vee R), \ (\neg P \vee S), \ \neg(Q \wedge \neg S) \ \vdash \ (R \vee S)$$

Although the proof is long and rather tedious there is nothing particularly clever about it — it is just a case of applying rules until each branch of the proof terminates in a sequent which is an axiom.

I have broken the proof tree up into four parts. This is just because the whole proof tree would be far too wide to fit onto the page.

$$\dfrac{\dfrac{}{Q,\ \neg P\ \vdash\ R,\ S,\ Q}\ \mathit{Axiom} \qquad \mathit{and} \qquad \dfrac{\dfrac{}{S,\ Q,\ \neg P\ \vdash\ R,\ S}\ \mathit{Axiom}}{Q,\ \neg P\ \vdash\ R,\ S,\ \neg S}\ [\vdash \neg]}{Q,\ \neg P\ \vdash\ R,\ S,\ (Q \wedge \neg S)}\ [\vdash \wedge]$$

$$\dfrac{\dfrac{\dfrac{}{P\ \vdash\ R,\ S,\ (Q \wedge \neg S),\ P}\ \mathit{Axiom}}{P,\ \neg P\ \vdash\ R,\ S,\ (Q \wedge \neg S)}\ [\neg \vdash] \qquad \mathit{and} \qquad \dfrac{:}{Q,\ \neg P\ \vdash\ R,\ S,\ (Q \wedge \neg S)}}{(P \vee Q),\ \neg P\ \vdash\ R,\ S,\ (Q \wedge \neg S)}\ [\vee \vdash]$$

$$\dfrac{\dfrac{:}{(P \vee Q),\ \neg P\ \vdash\ R,\ S,\ (Q \wedge \neg S)} \qquad \mathit{and} \qquad \dfrac{\dfrac{}{(P \vee Q),\ S\ \vdash\ R,\ S,\ (Q \wedge \neg S)}\ \mathit{Axiom}}{(P \vee Q),\ (\neg P \vee S)\ \vdash\ R,\ S,\ (Q \wedge \neg S)}\ [\vee \vdash]$$

$$\dfrac{\dfrac{:}{(P \vee Q),\ (\neg P \vee S)\ \vdash\ R,\ S,\ (Q \wedge \neg S)} \qquad \mathit{and} \qquad \dfrac{\dfrac{}{R,\ (\neg P \vee S)\ \vdash\ R,\ S,\ (Q \wedge \neg S)}\ \mathit{Axiom}}{\dfrac{\dfrac{((P \vee Q) \vee R),\ (\neg P \vee S)\ \vdash\ R,\ S,\ (Q \wedge \neg S)}{((P \vee Q) \vee R),\ (\neg P \vee S),\ \neg(Q \wedge \neg S)\ \vdash\ R,\ S}\ [\neg \vdash]}{((P \vee Q) \vee R),\ (\neg P \vee S),\ \neg(Q \wedge \neg S)\ \vdash\ (R \vee S)}\ [\vdash \vee]}\ [\vee \vdash]$$

The rules of the propositional sequent calculus can be applied in any order and this will not affect the possibility of proving any sequent. However, the size of the resulting proof tree may vary with different sequences of rule applications. In the above example I have applied non-branching rules before branching rules; and when applying a branching rule I have always been able to find an application where one of the branches is an axiom (this is not possible in all proofs). This strategy minimises the size of a proof tree — but it may still be very large.

5.6.4 Example 4

For the final example we shall look at what happens when we try to prove an invalid sequent.

$$\dfrac{\dfrac{\neg P,\ Q\ \vdash\ P \qquad \mathit{and} \qquad Q,\ Q\ \vdash\ P}{(\neg P \vee Q),\ Q\ \vdash\ P}\ [\vee \vdash]}{(P \rightarrow Q),\ Q\ \vdash\ P}\ [\rightarrow \text{r.w.}]$$

For all invalid propositional sequents, after applying enough rules, we will end up with (at least) one branch ending in a sequent consisting entirely of atomic propositions (i.e. no connectives), which is not an axiom — i.e. no proposition occurs on both sides of the sequent.

Section 6

Formal Semantics

We have seen that a notion of *validity* can be defined independently of the notion of *provability*:

> An argument is valid if it is not possible for its premisses to be true and its conclusion is false.

We could make this precise if we could somehow specify the conditions under which a logical formulae is true. Such a specification is called a *formal semantics* or an *interpretation* for a logical language.

6.1 Interpretation of Propositional Calculus

To specify a formal semantics for propositional calculus we take literally the idea that 'a proposition is either true or false'.

We say that the *semantic value* of every propositional formula is one of the two values **t** or **f**— which are called *truth-values*.

For the atomic propositions this value will depend on the particular fact that the proposition asserts and whether this is true. Since propositional logic does not further analyse atomic propositions we must simply assume there is some way of determining the truth values of these propositions.

The connectives are then interpreted as *truth-functions* which completely determine the truth-values of complex propositions in terms of the values of their atomic constituents.

6.2 Truth-Tables

The truth-functions corresponding to the propositional connectives $\neg$, $\wedge$ and $\vee$ can be defined by the following tables:

α	$\neg\alpha$
f	**t**
t	**f**

α	β	$(\alpha \wedge \beta)$
f	**f**	**f**
f	**t**	**f**
t	**f**	**f**
t	**t**	**t**

α	β	$(\alpha \vee \beta)$
f	**f**	**f**
f	**t**	**t**
t	**f**	**t**
t	**t**	**t**

These give the truth-value of the complex proposition formed by the connective for all possible truth-values of the component propositions.

6.2.1 The Truth-Function for '→'

The truth-function for ' → ' is defined so that a formulae $(\alpha \rightarrow \beta)$ is always true except when α is true and β is false:

α	β	$(\alpha \rightarrow \beta)$
f	**f**	**t**
f	**t**	**t**
t	**f**	**f**
t	**t**	**t**

So the statement 'If logic is interesting then pigs can fly' is true if either 'Logic is interesting' is false or 'Pigs can fly is true'.

Thus a formula $(\alpha \rightarrow \beta)$ is *truth-functionally equivalent* to

$$(\neg\alpha \vee \beta).$$

6.3 Propositional Models

A propositional *model* for a propositional calculus in which the propositions are denoted by the symbols $P_1, \ldots, P_n$, is a specification assigning a truth-value to each of these proposition symbols. It might by represented by, e.g.:

$$\{\langle P_1 = \mathbf{t}\rangle, \langle P_2 = \mathbf{f}\rangle, \langle P_3 = \mathbf{f}\rangle, \langle P_4 = \mathbf{t}\rangle, \ldots\}$$

Such a model determines the truth of all propositions built up from the atomic propositions $P_1, \ldots, P_n$. (The truth-value of the atoms is given directly and the values of complex formulae are determined by the truth-functions.)

If a model, $\mathcal{M}$, makes a formula, ϕ, true then we say that

$$\mathcal{M} \textit{ satisfies } \phi.$$

6.4 Validity in terms of Models

Recall that an argument's being valid means that: in all possible circumstances in which the premisses are true the conclusion is also true.

From the point of view of truth-functional semantics each model represents a possible circumstance — i.e. a possible set of truth values for the atomic propositions.

To assert that an argument is *truth-functionally valid* we write

$$P_1, \ldots, P_n \models_{TF} C$$

and we define this to mean that ALL models which satisfy ALL of the premisses, $P_1, \ldots, P_n$ also satisfy the conclusion C.

6.5 Semantics for Predicate Logic

6.5.1 The Domain of Individuals

Whereas a model for propositional logic assigns truth values directly to propositional variables, in predicate logic the truth of a proposition depends on the meaning of its constituent predicate and argument(s). The arguments of a predicate may be either *constant names* (a, b, ...) or *variables* (u, v, ..., z).

To formalise the meaning of these argument symbols each predicate logic model is associated with a set of entities that is usually called the *domain of individuals* or the *domain of quantification*. (Note: Individuals may be anything — either animate or inanimate, physical or abstract.) Each constant name *denotes* an element of the domain of individuals and variables are said to *range over* this domain.

6.5.2 Semantics for Property Predication

Let us start by considering the semantics of property predication. A property is formalised as a 1-place predicate — i.e. a predicate applied to one argument. For instance $\mathsf{Happy}(\mathsf{jane})$ ascribes the property denoted by Happy to the individual denoted by jane. To give the conditions under which this assertion is true, we specify that Happy denotes the *set* of all those individuals in the domain that are happy. Then $\mathsf{Happy}(\mathsf{jane})$ is true just in case the individual denoted by jane is a member of the set of individuals denoted by Happy.

In order to make this semantic interpretation more precise, we introduce a formal *model* structure, which is a little more complex than the propositional models considered above.

6.5.3 Predicate Logic Model Structures

A predicate logic model is a tuple

$$\mathcal{M} = \langle \mathcal{D}, \delta \rangle \ ,$$

where:

- $\mathcal{D}$ is a non-empty set (the domain of individuals) —
  i.e. $\mathcal{D} = \{i_1, i_2, \ldots\}$, where each i_n represents some entity.
- δ is an assignment function, which gives a value to each constant name and to each predicate symbol.

The kind of value given to a symbol σ by the assignment function δ depends on the type of σ:

- If σ is a constant name then $\delta(\sigma)$ is simply an element of $\mathcal{D}$.
  (E.g. $\delta(\mathsf{john})$ denotes an individual called 'John'.)
- If σ is a property, then $\delta(\sigma)$ denotes a *subset* of the elements of $\mathcal{D}$.
  This is the subset of all those elements that possess the property σ. (E.g. $\delta(\mathsf{Red})$ would denote the set of all red things in the domain.)

- If σ is a binary relation, then $\delta(\sigma)$ denotes a *set of pairs* of elements of $\mathcal{D}$.
  For example we might have
  $$\delta(R) = \{\langle i_1, i_2\rangle, \langle i_3, i_1\rangle, \langle i_7, i_2\rangle, \ldots\}$$
  The value $\delta(R)$ denotes the set of all pairs of individuals that are related by the relation R.
  (Note that we may have $\langle i_m, i_n\rangle \in \delta(R)$ but $\langle i_n, i_m\rangle \notin \delta(R)$ — e.g. John loves Mary but Mary does not love John.)

- More generally, if σ is an n-ary relation, then $\delta(\sigma)$ denotes a *set of n-tuples* of elements of $\mathcal{D}$.
  (E.g. $\delta(\mathsf{Between})$ might denote the set of all triples of points, $\langle p_x, p_y, p_z\rangle$, such that p_y lies between p_x and p_z.)

6.5.4 The Semantics of Predication

We have seen how the denotation function δ assigns a value to each individual constant and each property and relation symbol in a predicate logic language. The purpose of this is to define the conditions under which a predicative proposition is true. Specifically, a predication of the form $\rho(\alpha_1, \ldots \alpha_n)$ is true according to δ if and only if

$$\langle \delta(\sigma_1), \ldots \delta(\sigma_n)\rangle \in \delta(\rho)$$

For instance, $\mathsf{Loves}(\mathsf{john}, \mathsf{mary})$ is true if and only if the pair $\langle \delta(\mathsf{john}), \delta(\mathsf{mary})\rangle$ (the pair of individuals denoted by the two names) is an element of $\delta(\mathsf{Loves})$ (the set of all pairs, $\langle i_m, i_n\rangle$, such that i_m loves i_n).

6.5.5 Variable Assignments and Augmented Models

In order to specify the truth conditions of quantified formulae we will have to interpret variables in terms of their possible values. Given a model $\mathcal{M} = \langle \mathcal{D}, \delta\rangle$, Let V be a function from variable symbols to entities in the domain $\mathcal{D}$. I will call a pair $\langle \mathcal{M}, \mathcal{V}\rangle$ an *augmented model*, where V is a variable assignment over the domain of $\mathcal{M}$.

If an assignment V' gives the same values as V to all variables except possibly to the variable x, I write this as:

$$V' \approx_{(x)} V \ .$$

This notation will be used in specifying the semantics of quantification.

6.5.6 Truth and Denotation in Augmented Models

We will use augmented models to specify the truth conditions of predicate logic formulae, by stipulating that ϕ is true in $\mathcal{M}$ if and only if ϕ is true in a corresponding augmented model $\langle \mathcal{M}, \mathcal{V}\rangle$. It will turn out that if a formula is true in *any* augmented model of $\mathcal{M}$, then it is true in *every* augmented model of $\mathcal{M}$. The purpose of the augmented models is to give a denotation for variables.

From an augmented model $\langle \mathcal{M}, \mathcal{V}\rangle$, where $\mathcal{M} = \langle \mathcal{D}, \delta\rangle$, we define the function δ_V, which gives a denotation for both constant names and variable symbols. Specifically:

- $\delta_V(\alpha) = \delta(\alpha)$, where α is a constant;
- $\delta_V(\xi) = V(\xi)$, where ξ is a variable.

6.5.7 Semantics for Quantifiers

We are now in a position to specify the conditions under which a quantified formula is true in an augmented model:

- $\forall x[\phi(x)]$ is true in $\langle \mathcal{M}, \mathcal{V} \rangle$
  iff $\phi(x)$ is true in every $\langle \mathcal{M}, \mathcal{V}' \rangle$, such that $V' \approx_{(x)} V$.
- $\forall x[\phi(x)]$ is true in $\langle \mathcal{M}, \mathcal{V} \rangle$
  iff $\phi(x)$ is true in some $\langle \mathcal{M}, \mathcal{V}' \rangle$, such that $V' \approx_{(x)} V$.

In other words $\forall x[\phi(x)]$ is true in a model just in case the sub-formula $\phi(x)$ is true whatever entity is assigned as the value of variable x, while keeping constant any values already assigned to other variables in ϕ. And $\exists x[\phi(x)]$ is true just in case there is some entity that can be assigned as the value of x which will make the sub-formula $\phi(x)$ true.

6.5.8 Semantics of Equality

In predicate logic, it is very common to make use of the special relation of *equality*, '='. The meaning of '=' can be captured by specifying axioms such as

$$\forall x \forall y[((x = y) \wedge \mathsf{P}(x)) \rightarrow \mathsf{P}(y)]$$

or by means of more general inference rules such as, from $(\alpha = \beta)$ and $\phi(\alpha)$ derive $\phi(\beta)$. We can also specify the truth conditions of equality formulae using our augmented model structures:

- $(\alpha = \beta)$ is true in $\langle M, V \rangle$, where $\mathcal{M} = \langle \mathcal{D}, \delta \rangle$, iff $\delta_V(\alpha)$ is the same entity as $\delta_V(\beta)$.

6.5.9 Full Semantics for Predicate Logic

To summarise, the semantics for predicate logic can be specified as follows:

- $\rho(\alpha_1, \ldots \alpha_n)$ is true in $\langle \mathcal{M}, \mathcal{V} \rangle$, where $\mathcal{M} = \langle \mathcal{D}, \delta \rangle$,
  iff $\langle \delta_V(\sigma_1), \ldots \delta_V(\sigma_n) \rangle \in \delta(\rho)$.
- $(\alpha = \beta)$ is true in $\langle M, V \rangle$, where $\mathcal{M} = \langle \mathcal{D}, \delta \rangle$, iff $\delta_V(\alpha) = \delta_V(\beta)$.
- $\neg\phi$ is true in $\langle \mathcal{M}, \mathcal{V} \rangle$ iff ϕ is *not* true in $\langle \mathcal{M}, \mathcal{V} \rangle$
- $(\phi \wedge \psi)$ is true in $\langle \mathcal{M}, \mathcal{V} \rangle$ iff both ϕ and ψ are true in $\langle \mathcal{M}, \mathcal{V} \rangle$
- $(\phi \vee \psi)$ is true in $\langle \mathcal{M}, \mathcal{V} \rangle$ iff either ϕ or ψ is true in $\langle \mathcal{M}, \mathcal{V} \rangle$
- $\forall x[\phi(x)]$ is true in $\langle \mathcal{M}, \mathcal{V} \rangle$
  iff $\phi(x)$ is true in every $\langle \mathcal{M}, \mathcal{V}' \rangle$, such that $V' \approx_{(x)} V$.

Section 7

Soundness and Completeness

7.1 Soundness

A proof system is *sound* with respect to a formal semantics if:

> *Every argument which is provable with the system is also valid according to the semantics.*

Using the symbols '$\vdash$' and '$\models$' the soundness of a proof system S with respect to a semantics M can be formally defined by stating that:

$$\text{If} \quad \Gamma \vdash_S C \quad \text{then} \quad \Gamma \models_M C.$$

7.2 Completeness

A proof system is *complete* with respect to a formal semantics if:

> *Every argument which is valid according to the semantics is also provable using the proof system.*

So the completeness of a proof system S with respect to a semantics M can be formally defined by stating that:

$$\text{If} \quad \Gamma \models_M C \quad \text{then} \quad \Gamma \vdash_S C.$$

7.3 Semantic Adequacy of a Proof System

The notions of soundness and completeness are of fundamental importance to the analysis of logical proof systems. If a proof system is to adequately reflect a formal semantics it must be both sound and complete. I.e. we require that:

$$\Gamma \vdash_S C \quad \textit{if and only if} \quad \Gamma \models_M C.$$

For such a system S, the set of arguments which are provable is exactly the same as the set of arguments which are valid with respect to the formal semantics M.

7.4 Soundness and Completeness of the Sequent Calculus

It can be shown that the system of sequent calculus rules, SC, is both sound and complete with respect to the truth-functional semantics for propositional formulae.

Thus, $\vdash_{\text{SC}} \Gamma \vdash C$ if and only if $\Gamma \models_{TF} C$.

(How this can be show is beyond the scope of this course.)

Section 8

Reasoning in Predicate Logic

8.1 Defining $\exists$ in Terms of $\forall$

We shall shortly look at sequent rules for handling the universal quantifier.

Predicate logic formulae will in general contain both universal ($\forall$) and existential ($\exists$) quantifiers. However, in the same way that in propositional logic we saw that $(\alpha \rightarrow \beta)$ can be replaced by $(\neg\alpha \vee \beta)$, the existential quantifier can be eliminated by the following re-write rule.

$$\exists\upsilon[\phi(\upsilon)] \qquad \Longrightarrow \qquad \neg\forall\upsilon[\neg\phi(\upsilon)]$$

These two forms of formula are equivalent in meaning.

8.2 The Sequent Rule for $\vdash \forall$

$$\frac{\Gamma \ \vdash \ \phi(\kappa),\ \Delta}{\Gamma \ \vdash \ \forall\upsilon[\phi(\upsilon)],\ \Delta}\ [\vdash \forall]^{*}$$

The * indicates a special condition:

The constant κ must not occur anywhere else in the sequent.

This restriction is needed because if κ occurred in another formulae of the sequent then it could be that $\phi(\kappa)$ is a consequence which can only be proved in the special case of κ. On the other hand if κ is not mentioned elsewhere it can be regarded as an *arbitrary* object with no special properties. Thus if the property $\phi(\ldots)$ can be proven true of an arbitrary object it must be true of all objects.

8.2.1 Examples

As an example we now prove that the formula $\forall x[P(x) \vee \neg P(x)]$ is a theorem of predicate logic. This formula asserts that every object either has or does not have the property $P(\ldots)$.

$$\cfrac{\cfrac{\cfrac{P(a) \ \vdash \ P(a)}{\vdash \ P(a),\ \neg P(a)}\ [\vdash \neg]}{\vdash \ (P(a) \vee \neg P(a))}\ [\vdash \ \vee\ ]}{\vdash \ \forall x[P(x) \vee \neg P(x)]}\ [\vdash \forall]$$

Here the (reverse) application of the $[\vdash \forall]$ rule could have been used to introduce not only a but any name, since no names occur on the LHS of the sequent.

Now consider the following illegal application of $[\vdash \forall]$:

$$\frac{P(b) \ \vdash \ P(b)]}{P(b) \ \vdash \ \forall x[P(x)]} \, [\vdash \forall]^{\dagger}$$

† This is an incorrect application of the rule, since b already occurs on the LHS of the sequent.

(Just because b has the property $P(\ldots)$ we cannot conclude that everything has this property.)

8.3 A Sequent Rule for $\forall \vdash$

A formula of the form $\forall \upsilon[\phi(\upsilon)]$ clearly entails $\phi(\kappa)$ for any name κ. Hence the following sequent rule clearly preserves validity:

$$\frac{\phi(\kappa), \ \Gamma \ \vdash \ \Delta}{\forall \upsilon[\phi(\upsilon)], \ \Gamma \ \vdash \ \Delta} \, [\forall \vdash]$$

But, the formulae $\phi(\kappa)$ makes a much weaker claim than $\forall \upsilon[\phi(\upsilon)]$. This means that this rule is not *reversible* since, the bottom sequent may be valid but not the top one.

Consider the case:

$$\frac{F(a) \ \vdash \ (F(a) \wedge F(b))}{\forall x[F(x)] \ \vdash \ (F(a) \wedge F(b))} \, [\forall \vdash]$$

8.4 A Reversible Version

A quantified formula $\forall \upsilon[\phi(\upsilon)]$ has as consequences all formulae of the form $\phi(\kappa)$; and, in proving a sequent involving a universal premiss, we may need to employ many of these *instances*.

A simple way of allowing this is by using the following rule:

$$\frac{\phi(\kappa), \ \forall \upsilon[\phi(\upsilon)], \ \Gamma \ \vdash \ \Delta}{\forall \upsilon[\phi(\upsilon)], \ \Gamma \ \vdash \ \Delta} \, [\forall \vdash]$$

When applying this rule backwards to test a sequent we find a universal formulae on the LHS and add some instance of this formula to the LHS.

Note that the universal formula is not removed because we may later need to apply the rule again to add a different instance.

8.4.1 An Example Needing 2 Instantiations

We can now see how the sequent we considered earlier can be proved by applying the $[\forall \vdash]$ rule twice, to instantiate the same universally quantified property with two different names.

$$\dfrac{\dfrac{\dfrac{F(a), F(b), \forall x[F(x)] \vdash F(a) \quad \textit{and} \quad F(a), F(b), \forall x[F(x)] \vdash F(b)}{F(a),\ F(b) \forall x[F(x)]\ \vdash\ (F(a) \wedge F(b))}\,[\vdash \wedge]}{F(a),\ \forall x[F(x)]\ \vdash\ (F(a) \wedge F(b))}\,[\forall \vdash]}{\forall x[F(x)]\ \vdash\ (F(a) \wedge F(b))}\,[\forall \vdash]$$

8.5 Termination Problem

We now have the problem that the (reverse) application of $[\forall \vdash]$ results in a more complex rather than a simpler sequent. Furthermore, in any application of $[\forall \vdash]$ we must choose one of (infinitely) many names to instantiate. Although there are various clever things that we can do to pick instances that are likely to lead to a proof, these problems are fundamentally insurmountable. This means that unlike with propositional sequent calculus, there is no general purpose automatic procedure for testing the validity of sequents containing quantified formulae.

In propositional logic we could always tell if a sequent was invalid because after applying sufficiently many rules we would end up with a sequent containing only atomic propositions which was not an axiom. However, with predicate logic this need not happen because when we the $[\forall \vdash]$ rule, the universal quantifier cannot be eliminated because we may need to apply it again later to yield a different instance of the formula.

Section 9

Decision Procedures and Decidability

A *decision procedure* for some class of problems is an algorithm which can solve any problem in that class in a finite time (i.e. by means of a finite number of computational steps).

Generally we will be interested in some infinite class of similar problems such as:

1. problems of adding any two integers together
2. problems of solving any polynomial equation
3. problems of testing validity of any propositional logic sequent
4. problems of testing validity of any predicate logic sequent

9.1 Decidability

A class of problems is *decidable* if there is a decision procedure for that class; otherwise it is *undecidable*. Problem classes 1–3 of the previous slide are decidable, whereas class 4 is known to be undecidable.

Undecidability of testing validity of entailments in a logical language is clearly a major problem if the language is to be used in a computer system: a function call to a procedure used to test entailments will not necessarily terminate.

9.1.1 Semi-Decidability

A problem class, where we want a result *Yes* or *No* for each problem, is called *(positively) semi-decidable* if every positive case can be verified in finite time but there is no procedure which will refute every negative case in finite time.

Despite the fact that predicate logic is undecidable, the rules that we have given for the quantifiers do give us a *complete* proof system for predicate logic. Furthermore, it is even possible to devise a strategy for picking instants in applying the $[\forall \vdash]$ rule, such that every *valid* sequent is provable in finite time. However, there is no procedure that will demonstrate the *invalidity* of every *invalid* sequent in finite time. Hence, although predicate logic is not decidable it is semi-decidable.

Section 10

Possible Questions on Classical Logic

If you have understood these notes you should be able to answer the following types of questions about classical logic:

- **Concepts:** You may be asked to define one or more of the logical concepts which have been introduced in the course. These include: proposition, inference rule, proof, validity, provability, propositional model, truth-functional semantics, soundness, completeness, decidability and semi-decidability.
- **Logical Representation:** You may be asked to translate natural language sentences into either propositional or predicate logic.
- **Semantics:** You may be asked about the formal semantics of propositional and predicate logic. You need to know the truth-tables for each connective and understand the semantics for quantification. Given a *model* you should be able to determine whether a formula is made true or false by the model and whether the model satisfies a given formula (it satisfies just those formulae which it makes true).
- **Sequent Calculus:** You may be asked to carry out a short proof using the sequent calculus (possibly including the quantifier rules). You should give a proof tree with the sequent to be proved at the root of the tree. In these notes proof trees have been given with the root at the bottom but, if you find it more convenient, you may put the root (i.e. the sequent to be proved) at the top and expand the proof tree downwards.
- **Uses of Logic:** You may be asked about uses of logic in computer systems or about difficulties associated with the use of logic.

Be aware that these notes do *not* cover everything you need to know about logic for the AR34 course. This material is intended only to provide a basic grounding in classical logic which provides a foundation for understanding Knowledge Representation and Reasoning techniques used in AI.

Section 11

Exercises

1. (a) Translate the following sentences into propositional logic. Give a key indicating the meaning of each propositional letter.
   i. If the mug was dropped and it was fragile it is now broken. (1 mark)
   ii. Toby is neither happy nor sad — he is confused. (1 mark)
   iii. Jeremy eats toast for breakfast except on Friday. (1 mark)

   (b) Give truth-tables for the propositional connectives '$\neg$' and '$\vee$'. (2 marks)

   (c) A model $\mathcal{M}$ for a propositional language is represented by the following structure:
   $$\{A = \mathbf{t},\ B = \mathbf{f},\ C = \mathbf{f}\}$$
   i. What truth-value is assigned to the formula $((A \rightarrow B) \vee C)$ by $\mathcal{M}$? (1 mark)
   ii. Does $\mathcal{M}$ satisfy the formula $(A \vee (B \vee C))$? (1 mark)

   (d) Using the rules of the sequent calculus prove the following propositonal argument:
   $$P,\ (P \rightarrow Q)\ \vdash\ (Q \vee R) \qquad \text{(3 marks)}$$

   (e) Represent the following statement in the notation of predicate logic:
   'A traveller visited every country in Europe'. (3 marks)

   (f) Using the sequent calculus prove that the following predicate logic argument is valid:
   $$\forall x[F(x) \wedge \neg G(x)]\ \vdash\ (F(a) \wedge \neg G(b)) \qquad \text{(5 marks)}$$

   (g) Explain what is meant by saying that the predicate calculus is *semi-decidable*. (2 marks)

2. (a) Translate the following sentences into propositional logic:
   i. It is the end of term if tomorrow is Friday (1 mark)
   ii. On Wednesdays and Fridays Susan goes climbing (1 mark)

(b) What does it mean to say that a proof system for a logical language is *sound* with respect to some formal semantics for that language? (2 marks)

(c) Explain by means of truth-tables, why in the truth-functional semantics for propositional logic the formula $\neg(P \wedge \neg Q)$ is equivalent to $(P \rightarrow Q)$. (3 marks)

(d) Using the rules of the sequent calculus prove the propositonal argument:

$$\neg(P \wedge \neg Q) \ \vdash \ (P \rightarrow Q) \qquad \text{(3 marks)}$$

(e) Represent the following statements in the notation of predicate logic:

i. Every number is either odd or even (2 marks)

ii. No student enjoys every lecture (2 marks)

(f) Using the sequent calculus prove that the following predicate logic argument is valid:

$$R(a), \ \forall x[R(x) \rightarrow S(x)] \ \vdash \ \exists x[S(x)] \qquad \text{(6 marks)}$$