COM3412: Logic and Computation

Marking guidelines for the 2007 exam

29th May 2008

1. (a) **OE** says that if x and y both overlap exactly the same things, then they are equal.

   (3 marks)

   PO says that x is part of y if and only if everything which overlaps x also overlaps y.

   (3 marks)

   (b) **PR** From $O(z, x)$ we can infer $O(z, x)$, hence $O(z, x) \to O(z, x)$ is true, and since this applies to any z we have $\forall x(O(z, x) \to O(z, x))$. By **PO**, this implies $P(x, x)$, and since x is general we have $\forall x P(x, x)$.

   (3 marks)

   PT Suppose $P(x, y) \wedge P(y, z)$. If $O(u, x)$, then by **PO**, we have $O(u, y)$ from $P(x, y)$, and hence also $O(u, z)$ from $P(y, z)$. Hence we have $O(u, x) \to O(u, z)$, and since this holds for all u we have $P(x, z)$ by **PO**. Hence $P(x, y) \wedge P(y, z) \to P(x, z)$. This holds for all x, y, z, giving **PT**.

   (4 marks)

   PA Suppose $P(x, y) \wedge P(y, x)$, and let $O(x, z)$. By **OS** this means that $O(z, x)$, and by **PO**, since we have $P(x, y)$, we obtain $O(z, y)$. By **OS**, this gives $O(y, z)$, and hence we have $O(x, z) \to O(y, z)$. Similarly, using $P(y, x)$ we get $O(y, z) \to O(x, z)$. Together, these give $O(x, z) \leftrightarrow O(y, z)$, and since this holds for all z, we get $x = y$ by **OE**. Hence $P(x, y) \wedge P(y, x) \to x = y$, and this holds for all x, y, giving **PA**.

   (5 marks)

   (c) For any non-empty set X, we have $X \cap X = X \neq \emptyset$. Hence if $I(x) = X$ we have $O(x, x)$, verifying **OR**.

   (2 marks)

   Clearly $X \cap Y \neq \emptyset$ implies $Y \cap X \neq \emptyset$, and hence if $I(x) = X$ and $I(y) = y$, we have $O(x, y) \to O(y, x)$, verifying **OS**.

   (3 marks)

   Now suppose $\forall z(O(x, z) \leftrightarrow O(y, z))$. Let $X = I(x)$ and $Y = I(y)$. Also, let $I(z) = \{a\}$. If $a \in X$ then $\{a\} \cap X \neq \emptyset$, so $O(x, z)$, hence $O(y, z)$, so $Y \cap \{a\} \neq \emptyset$, so $a \in Y$. Therefore $X \subseteq Y$. Similarly $Y \subseteq X$. Hence $X = Y$, and therefore we have $\forall z(O(x, z) \leftrightarrow O(y, z)) \to x = y$, verifying **OE**.

   (5 marks)

   (d) $P(x, y)$ means, if **PO** is to be true, that for any set Z, if $Z \cap X \neq \emptyset$, then $Z \cap Y \neq \emptyset$.

   (3 marks)

   In particular, for each $a \in X$ we have $\{a\} \cap X \neq \emptyset$, giving $\{a\} \cap Y \neq \emptyset$, so $a \in Y$. Hence

$X \subseteq Y$. Thus $P(x, y)$ should be interpreted to mean $X \subseteq Y$.

(3 marks)

(e) In the suggested interpretation we can have, for example,

$$X = \{0, 1\},\ Y = \{1, 2\},\ Z = \{2, 3\}.$$

Then $X \cap Y \neq \emptyset$, $Y \cap Z \neq \emptyset$, $X \cap Z = \emptyset$, so the formula

$$O(x, y) \wedge O(y, z) \wedge \neg O(x, z)$$

is satisfied, contradicting **OT**. Thus **OT** is not satisfied by every model for $\{$**OR**, **OS**, **SE**$\}$, and is therefore not a logical consequence of them.

(6 marks)

2. (a) Assume the following non-logical vocabulary:

Constant: 0
Unary function symbol: s
Binary function symbols: $+$, $\times$

The first-order theory of arithmetic consists of all formulae over this vocabulary, in the first-order predicate calculus with identity, which are satisfied under the standard interpretation with domain $\mathbb{N}$ and

0 means zero
s means the successor function
$+$, $\times$ mean addition and multiplication respectively.

(6 marks)

(b) It means that there is no finitely-specifiable set of first-order formulae such that the first-order theory of arithmetic consists of all and only the logical consequences of those formulae.

(3 marks)

The key step in the proof is the 'arithmetisation of syntax', by which formulae and sequences of formulae are assigned natural numbers ('Gödel numbers') in such a way that the sequence of symbols in each formula (or sequence of formulae) is recoverable from the sequence of exponents in the unique prime factorisation of its Gödel number. This enables one to construct formulae which may be interpreted as saying things about other formulae (or indeed themselves). It is then possible to write down a formula which says that (i.e., is true if and only if) that very formula cannot be proved in the formal system under consideration. If true, this formula is *true but unprovable*, testifying to the incompleteness of the system, while if it is false, it is *false but provable*, testifying to the unsoundness of the system — either way the system cannot be both sound and complete.

(7 marks)

(c) By **M1** we have

(I) $s0 * 0 = 0$.

(This is the base case.) Suppose we have

(IH) $s0 * a = a$.

(This is the induction hypothesis.) Then

$$\begin{array}{rcll} s0 * sa & = & (s0 * a) + s0 & \text{(by } \mathbf{M2}\text{)} \\ & = & a + s0 & \text{(by } \mathbf{IH}\text{)} \\ & = & s(a + 0) & \text{(by } \mathbf{A2}\text{)} \\ & = & sa & \text{(by } \mathbf{A1}\text{)} \end{array}$$

Hence $s0 * a = a \rightarrow s0 * sa = sa$, and since this holds for all values of a we have

(II) $\forall x(s0 * x = x \rightarrow s0 * sx = sx)$

By **Ind**, with $\Phi(x) \equiv s0 * x = x$, we have

(III) $s0 * 0 = 0 \wedge \forall x(s0 * x = x \rightarrow s0 * sx = sx) \rightarrow \forall x(s0 * x = x)$.

Together, **I**, **II**, and **III** imply $\forall x(s0 * x = x)$ as required.

(9 marks)

(d) The axioms (including the infinitely many substitution-instances of the induction schema) of Peano Arithmetic constitute a finitely-specifiable set of formulae. By Gödel’s theorem, the complete set of logical consequences of these formulae cannot be identical with the first-order theory of arithmetic. It follows that *either* there are formulae of Peano Arithmetic which are not true statements about natural numbers (and hence that at least one of the axioms is false), *or* there are true statements about the natural numbers, expressible in the language of Peano Arithmetic, which cannot be proved within Peano Arithmetic.

(5 marks)

3. (a) The rule has the effect that once the machine has arrived in the halt state it stays there with no further changes.

(1 mark)

It is needed so that T, U, and D are total functions, enabling them to be represented using function symbols in the logical representation.

(2 marks)

(b) q is the current state.
a is the symbol on the currently scanned square.
b is the string of symbols running left from the scanned square as far as the leftmost non-blank square (or the empty string if there are no non-blank squares to the left of the scanned square.
c is the same as b but with 'left' replaced by 'right'.

(1.5 marks each)

(c)
$$\begin{array}{rcl} q(n+1) & = & T(q(n),a(n)) \quad \text{(The next state)} \\ a(n+1) & = & \begin{cases} head(b(n)) & \text{(if } D(q(n),a(n)) = -1 \text{ — moving left)} \\ U(q(n),a(n)) & \text{(if } D(q(n),a(n)) = 0 \text{ — staying still)} \\ head(c(n)) & \text{(if } D(q(n),a(n)) = 1 \text{ — moving right)} \end{cases} \\ b(n+1) & = & \begin{cases} tail(b(n)) & \text{(if } D(q(n),a(n)) = -1\text{)} \\ b(n) & \text{(if } D(q(n),a(n)) = 0\text{)} \\ U(q(n),a(n))^\frown b(n) & \text{(if } D(q(n),a(n)) = 1\text{)} \end{cases} \\ c(n+1) & = & \begin{cases} U(q(n),a(n))^\frown c(n) & \text{(if } D(q(n),a(n)) = -1\text{)} \\ c(n) & \text{(if } D(q(n),a(n)) = 0\text{)} \\ tail(c(n)) & \text{(if } D(q(n),a(n)) = 1\text{)} \end{cases} \end{array}$$

(10 marks)

(d) (i) $q(n) = 0$.

(2 marks)

(ii) $\exists n(q(n) = 0)$.

(2 marks)

(e) The rules given above must be supplemented with general rules for handling numbers (for the states), lists (for b and c), etc. [See details in notes]. The premisses of the inference are all the above rules and a statement of the initial configuration, in the form

$$C(0) = (q(0), a(0), b(0), c(0)).$$

The conclusion is the statement that the machine eventually halts:

$$\exists a, b, c, i(C(i) = (0, a, b, c)).$$

If every case of the Entscheidungsproblem were solvable then it could be determined by mechanical means whether or not this inference was valid, i.e., whether or not the machine halts. But Turing had already shown that the Halting Problem is not solvable by mechanical means, and this therefore shows that the same is true of the Entscheidungsproblem.

(7 marks)

4. (a) A decision problem is in NP if there is a polynomial-time algorithm which, given any instance of the problem, and a candidate solution (or, more generally, certificate), will verify whether or not the candidate solution is in fact a solution for that instance.

(3 marks)

For the given problem, a candidate solution is a subset U of S. All the algorithm has to do is to add up the members of U, and check whether the result is equal to n. This can be done in time that is a linear function of the total size of S (as measured by, e.g., the total number of digits in all the members of S — note the cardinality of S alone is not suitable since the numbers themselves may be arbitrarily large).

(3 marks)

(b) An NP problem is NP-complete if it is also NP-hard, which means that any other NP problem can be reduced to it in polynomial time.

(4 marks)

Examples of NP-complete problems include Travelling Salesman, Hamiltonian Circuit, Graph Colouring, etc.

(1 marks)

Examples of NP problems which aren't NP complete include searching, sorting, matrix multiplication, etc.

(1 marks)

(c) SAT: Given a set of propositional clauses, to determine whether there is a truth-assignment which makes them all true.

(2 marks)

Here I'm looking for a brief description of Cook's proof, by which he showed how to encode as Propositional Calculus clauses both the relevant facts about the problem instance and the Turing machine which does the certificate-checking, in such a way that the resulting set of clauses is satisfiable if and only if the original problem instance is positive. Thus the problem of determining the latter is reduced to the problem of determining the former.

(6 marks)

(d) If a given NP problem Q is such that a known NP-complete problem such as SAT can be reduced to it in polynomial time, this shows that Q itself must be NP complete, since an arbitrary NP problem can be reduced first to SAT (or the other known NP problem) and then to Q, both reductions, and hence their sequential composition, being achievableE in polynomial time.

(5 marks)

(e) Points to note here: Many problems of practical interest are NP-complete; since the fastest-known exact algorithms for NP-complete problems run in exponential time or worse, it is infeasable to run them on 'industrial scale' inputs (here a reference to the 'P=NP?' problem might be appropriate). Hence the practical thing to do is to work with algorithms which are approximate in the sense that either they deliver results that approximate to the correct results, or they deliver the correct results with a certain quantifiable (and acceptable) probability.

(5 marks)

5. (a) The given description certainly applies to much (if not most) of what goes on in a computer, although it does not apply very comfortably to the ongoing computation of a reactive system such as an operating system. On the other hand, the process by which analogue informaion is digitised (e.g., in sound recording or scanning of images) might be described as a kind of computation, and yet it certainly does not come under the description given. I'm looking for a discussion of what processes should count as computational, of what processes can be described in the way suggested in the question, and of the relationship between these. [This material was discussed in the module as a preliminary to talking about the Turing machine model of computation.]

(30 marks)

(b) I'm expecting an account of what Gödel's theorems say, of the use that has been made of them by people such as Lucas and Penrose to argue against the possibility of strong AI, and a critique or endorsement of those arguments. [This was discussed in the module after the technical work on Gödel's theorems.]

(30 marks)