CSC 486/2502, 2000 – Assignment 1

Due: Friday, October 13

---

1. [**20 Marks**] (Logic Puzzle)
   Donald and Daisy Duck took their nephews aged 4, 5 and 6 on an outing. Each boy wore a tee-shirt with a different design on it and of a different colour. You are also given the following information:

   - Huey is younger than the boy in the green tee-shirt
   - The five year-old wore the tee-shirt with the camel design
   - Dewey's tee-shirt was yellow
   - Louie's tee-shirt bore the giraffe design
   - The panda design was not featured on the white tee-shirt

   (a) Represent these facts as sentences in first-order logic.

   (b) Using your formalisation in part (1a), is it possible to conclude the age of each boy together with the colour and design of the tee-shirt they're wearing? Show semantically how you determined your answer.

   (c) If your answer to part (1b) was 'no', indicate what further sentences you would need to add to your formalisation so that you could conclude the age of each boy together with the colour and design of the tee-shirt they're wearing.

2. [**40 Marks**] (Resolution)
   It is possible to consider several variants to the *resolution* proof procedure, some of which were mentioned in lectures. We shall consider two variants here.

   - A *unit resolution* is one in which a resolvent is obtained using at least one parent which is a unit clause (i.e., a single literal) or a unit factor[1] of a parent clause.
   - An *input resolution* is one in which at least one of the two parent clauses comes from the original knowledge base of clauses.
   - A *unit* (respectively, *input*) *deduction* is one in which every resolution is a unit (respectively, input) resolution.
   - A *unit* (respectively, *input*) *refutation* is a unit (respectively, input) deduction of the empty clause.

   (a) While completeness is an important consideration for automated deduction, so is efficiency. At times we may be willing to forego completeness for the sake of efficiency. Show that unit resolution is incomplete. [Hint: exhibit a set of clauses and explain your answer.]

   (b) Given a finite set S of ground clauses, show that there is a unit refutation from S if and only if there is an input refutation from S. [Hint: use mathematical induction on the number of elements in the Herbrand base of S.]

   (c) Given a finite set S of clauses, use the result you just proved in (2b) to show that there is a unit refutation from S if and only if there is an input refutation from S. [Hint: You may find the following "lifting lemma" useful: *If C_1' and C_2' are instances of C_1 and C_2, respectively, and if C' is a resolvent of C_1' and C_2', then there is a resolvent C of C_1 and C_2 such that C' is an instance of C.*]

---

[1] If two (or more) literals of a clause C have a most general unifier σ, then $C\sigma$ is known as a *factor* of C. For example, $C = [Q(x),\ Q(g(y)),\ \neg R(x)]$ has a most general unifier $\sigma = \{x/g(y)\}$ giving the factor $[Q(g(y)), \neg R(g(y))]$.

(d) Consider combining resolution strategies to create new resolution rules. If we combine unit resolution and input resolution into a single new rule that says we can only obtain a resolvent using at least one parent which is a unit input clauses or a factor of a unit input clause, does this make it *impossible* to prove some things that are provable by unit resolution alone? If not, prove that there is no difference.

3. [**40 Marks**] (Undecidability of first-order logic)
In this question we formalise a proof of the *undecidability of first-order logic*. The proof involves formalising a Turing machine in first-order logic and appealing to the well-known *halting problem*. More precisely, given a description of a Turing machine and any input n, we can write down a finite set of sentences Γ and a sentence H such that $\Gamma \models H$ if and only if the given machine eventually halts on input n. The proof consists of two parts: (a) formalisation and (b) verification.

(a) Formalisation
For each state s_i of the Turing machine introduce a two-place predicate letter S_i. The intended interpretation is that $S_i(t,\ x)$ is true if and only if the Turing machine is in state s_i at time t scanning the tape square numbered x (time is taken to be discrete). For each symbol (or token) t_j introduce a two-place predicate T_j whose intended interpretation is that $T_j(t,\ x)$ is true if and only if the Turing machine is scanning symbol t_j at time t in square x. We use the predicate T_{blank} to denote the blank symbol. We also introduce the two place predicate $<$ with the usual intended interpretation, the constant symbol 0 and the successor function $'$.

We shall now formalise the sentences comprising Γ. In general, many sentences must be added to Γ depending on the nature of the given Turing machine. However, we wish to prove a general result for all Turing machines. For the following, give the *general form* (i.e., a schema) of the sentence to be added.

i. Introduce one sentence corresponding to writing a symbol in the current square
ii. Introduce two sentences corresponding to shifting the read/write head left and right
iii. Introduce one sentence corresponding to the initial state of the Turing machine

You will also require the following two sentences (dealing with arithmetic):

$$\forall z \exists x . z = x' \wedge \forall x \forall y \forall z . (z = x' \wedge z = y' \supset x = y)$$

$$\forall x \forall y \forall z . (x < y \wedge y < z \supset x < z) \wedge \forall x \forall y . (x' = y \supset x < y) \wedge \forall x \forall y . (x < y \supset x \neq y)$$

This concludes the composition of the set Γ. Now we need to specify the halting sentence H.

iv. Introduce the sentence H corresponding to the Turing machine being in a halting state at some time t (recall, there may be more than one halting state)

(b) Verification
We now wish to show:

$\Gamma \models H$ if and only if the Turing machine eventually halts when given input n.

This proceeds in two parts.

i. Prove the 'only if' part of this result. (This should be relatively short.)
ii. Prove the 'if' part of this result.
At any (non-negative) time s, the state of the Turing machine can be described by a sentence of the form (0^x represents the xth successor of 0):

$$S_i(0^s,\ 0^p) \wedge T_{j_1}(0^s,\ 0^{p_1}) \wedge \ldots \wedge T_j(0^s,\ 0^p) \wedge \ldots \wedge T_{j_m}(0^s,\ 0^{p_n}) \wedge$$

$$\forall y . [(y \neq 0^{p_1} \wedge \ldots \wedge y \neq 0^p \wedge \ldots \wedge y \neq 0^{p_n}) \supset T_{blank}(0^s,\ y)]$$

Since the Turing machine eventually halts (at time t say), we would like to show that Γ entails a description of time t which entails the halting sentence H. We need to show that Γ entails each description of time s, $0 \leq s \leq t$ [Hint: use mathematical induction]). We also need to show that the description of time t entails H.

Having verified the above we can now appeal to the proof of the Halting problem to show that there is no (mechanical) procedure for deciding whether $\Gamma \models H$ and therefore first-order logic is undecidable. (**Note:** this is not how undecidability was first proved.)

4. [**50 Marks**] (Program)
The problem with the simple back-chaining procedure presented in lectures can be remedied by keeping a list of the currently active goals (to avoid infinite loops), and marking atoms as T or F as the procedure progresses (to avoid working on the same goal twice). This results in a procedure like the following:

```
SOLVE[a1, ..., an; active]:
If n = 0 then return YES
else if a1 is marked T then return SOLVE[a2,..., an; active]
else if a1 is marked F then return NO
else if there is a rule r of the form (a1 ← b1, ..., bm)
    such that r ∉ active and
    SOLVE[b1, ..., bm, active ∪ {r}]
then mark a1 as T and return SOLVE[a1, ..., an; active]
else mark a1 as F and return NO
```

(a) Implement a version of this procedure and test it on your own examples. You should provide documentation in the form of program comments and any additional documentation necessary to understand how your program works and the form of input that it accepts.

(b) Explain how a back-chaining procedure like the one above can be used to test satisfiability of an arbitrary collection of propositional Horn clauses, positive and negative. Extend your program as necessary to handle such sets of clauses, and test it on some of your own examples.

(c) Argue that the back-chaining procedure above can be made to run in time that is linear in the size of the knowledge base using special data structures. In particular, assume that for any atom, you can obtain a list of clauses where it appears positive in constant time, and for any clause you can obtain a list of atoms that appear in it negatively in constant time. It is not necessary to implement these data structures.