CS 472/572 **Spring 1997**

Programming Project 3
AUTOMATED THEOREM PROVING

In this project, you will write programs to pass the graduate-level AI Qualifying Exam questions on logic :-)

1. (Warning: This part is relatively easy.)

   (a) Given the following rules for converting a wff into clause form (also called Skolem standard form (SSF)), write a Lisp function that takes a wff as input and returns an equivalent wff in SSF:

      i. Eliminate all occurrences of $\Leftrightarrow$ by rewriting all occurrences of
         $(\mathbf{P} \Leftrightarrow \mathbf{Q})$ as $((\mathbf{P} \Rightarrow \mathbf{Q}) \wedge (\mathbf{Q} \Rightarrow \mathbf{P}))$.

      ii. Eliminate all occurrences of $\Rightarrow$ by rewriting all occurrences of
         $(\mathbf{P} \Rightarrow \mathbf{Q})$ as $(\neg\, \mathbf{P} \vee \mathbf{Q})$.

      iii. Reduce the scope of all $\neg$ signs to apply to at most one atomic wff by using these rewrite rules:[1]

$$\begin{array}{lcl} \neg\,(\mathbf{P} \vee \mathbf{Q}) & \rightarrow & (\neg\, \mathbf{P} \wedge \neg\, \mathbf{Q}) \\ \neg\,(\mathbf{P} \wedge \mathbf{Q}) & \rightarrow & (\neg\, \mathbf{P} \vee \neg\, \mathbf{Q}) \\ \neg\, \forall \mathrm{v} \mathbf{P} & \rightarrow & \exists \mathrm{v}[\neg\, \mathbf{P}] \\ \neg\, \exists \mathrm{v} \mathbf{P} & \rightarrow & \forall \mathrm{v}[\neg\, \mathbf{P}] \\ \neg\, \neg\, \mathbf{P} & \rightarrow & \mathbf{P} \end{array}$$

      iv. Rename variables so that every quantifier binds a unique variable,
         using rewrite rules of the following form:

$$(Q_1 v_1 F(v_1^*) \;\#\; Q_2 v_1 G(v_1^*)) \rightarrow (Q_1 v_1 F(v_1^*) \;\#\; Q_2 v_2 G(v_2^*))$$

         where the Q_i are quantifiers (either the same or different), # is either $\vee$ or $\wedge$, the v_i are variables such that 'v_1' $\neq$ 'v_2', and '$F(v_1^*)$' represents a wff containing 0 or more occurrences of 'v_1'. An example would be:

$$(\forall x P(a,x) \wedge \exists x R(a)) \rightarrow (\forall x P(a,x) \wedge \exists y R(a))$$

      v. Put the wff in Skolem Normal Form (SNF) by eliminating all existential quantifiers and introducing appropriate Skolem constants and Skolem functions.

      vi. Put the wff in Prenex Normal Form (PNF) by moving all universal quantifiers to the left. This can be done mechanically, but it is offically justified by the following equivalences:
         - $((\forall \mathrm{x}[\mathrm{F(x)}] \vee \forall \mathrm{y}[\mathrm{G(y)}]) \Leftrightarrow \forall \mathrm{x} \forall \mathrm{y}[\mathrm{F(x)} \vee \mathrm{G(y)}])$
         - $((\forall \mathrm{x}[\mathrm{F(x)}] \wedge \forall \mathrm{y}[\mathrm{G(y)}]) \Leftrightarrow \forall \mathrm{x} \forall \mathrm{y}[\mathrm{F(x)} \wedge \mathrm{G(y)}])$
         - $((\forall \mathrm{x}[\mathrm{F(x)}] \vee \mathrm{H}\{\mathrm{x}\}) \Leftrightarrow \forall \mathrm{x}[\mathrm{F(x)} \vee \mathrm{H}\{\mathrm{x}\}])$
         - $((\forall \mathrm{x}[\mathrm{F(x)}] \wedge \mathrm{H}\{\mathrm{x}\}) \Leftrightarrow \forall \mathrm{x}[\mathrm{F(x)} \wedge \mathrm{H}\{\mathrm{x}\}])$

         where 'F(x)' and 'G(x)' denote wffs containing a free variable x, and 'H{x}' denotes a wff that does *not* contain the variable x.

      vii. Eliminate all universal quantifiers.

      viii. Put the result into Skolem Standard Form (SSF) by rewriting
         $(\mathrm{P} \vee (\mathrm{Q} \wedge \mathrm{R}))$ as $((\mathrm{P} \vee \mathrm{Q}) \wedge (\mathrm{P} \vee \mathrm{R}))$.

   (b) Apply your algorithm to the following wff:

$$\forall x[Animal(x) \Rightarrow (Predator(x) \Leftrightarrow \exists y[Animal(y) \wedge Eats(x,y)])]$$

[1] The notation '$\alpha \rightarrow \beta$' means: "rewrite α as β".

2. (Warning: This part is relatively hard.)

   (a) Implement a unification algorithm (either the one in the text, the one given in lecture, or—if you did it correctly—your pattern-matcher from Project 1 (perhaps suitably modified)). More precisely, your algorithm should take a pair of wffs as input and either return their MGU if they are unifiable or else return a message such as "NOT UNIFIABLE". You may assume that the notation $f(x, g(x))$ can be understood as: `(f x (g x))`, if you prefer using Lispish notation.

   (b) Use your algorithm to answer this question. For each of the following pairs of terms, if they unify, show a most general unifier (mgu); if they don't, say so, and state why. Assume that u, v, x, y, and z are variables, and that a, b, and c are individual constants.

      i. $P(a, x, c)$ and $P(y, b, z)$
      ii. $P(a, x, c)$ and $P(y, b, y)$
      iii. $P(x, x, c)$ and $P(u, v, u)$
      iv. $P(x, f(x), f(y))$ and $P(f(a), f(z), z)$
      v. $P(x, f(x), f(a))$ and $P(f(z), f(z), z)$

3. (a) (Warning: The computational implementation of this part is relatively hard.)
   THE COMPUTATIONAL IMPLEMENTATION OF THIS PART IS OPTIONAL

   Write a resolution + unification theorem prover for *first-order predicate* logic.

   (b) Using resolution, show that the following set of clauses is inconsistent. Assume that a, b, and c are individual constants, and that x and y are variables. **DO THIS PART (AT LEAST BY HAND) WHETHER OR NOT YOU DO PART 3a.**

      i. $\{On(a, b)\}$
      ii. $\{On(b, c)\}$
      iii. $\{Red(a)\}$
      iv. $\{Green(c)\}$
      v. $\{Red(b), Green(b)\}$
      vi. $\{\neg Red(x), \neg Green(y), \neg On(x, y)\}$

NOTE: Please do *all* exercises at least *by hand* (in addition to, or instead of, using Lisp) as part of your *report*. I will hand out a tentative grading scheme to make it easier for you to organize your final report.

DUE AT START OF LECTURE, THURSDAY, APRIL 10.

NOTE: Starting with this project, I will institute a policy of what some people call "slip" days (I prefer the term "grace period"). You now have 3 "slip" days that you can use on this project or the next one. This works as follows: You may turn this project in up to 3 days late, with no penalty (where a "day" is a 24-hour time period; thus, you may turn this project in as late April 13). If you "spend" all your slip days on this project, then you will have none for the next one. If you turn this one in 1 day late ("spending" 1 of your slip days), then you will only have 2 slip days for the next project, and so on. Any projects turned in late after your slip days are used up will be accepted at the discount mentioned in the syllabus.