COSC 4P79 Expert systems **Assignment #1** B. Ross

Due date: Friday February 3, 12:00 noon; lates until Monday Feb 6, 12:00 noon (-25%).

Objectives: Prolog programming practice! All questions use Sicstus Prolog interpreter. (SWI Prolog and others should work too, with some modifications). For programming questions, hand in a source listing as well as a dialog listing (eg. use the 'script' utility of unix).

1. (a) Construct Prolog rules defining the following family relations:

mother
daughter
grandfather
grandchild
niece
second-cousin*
father
child_of
grandson
aunt
nephew
sibling
son
grandmother
granddaughter
uncle
cousin
great-grandparent

(* more complicated than you might think -- see Wikipedia page!)

(b) Add to the database information about your own family tree, and run some queries to test the different predicates. Make sure a person is not his or her own sibling!

2. Write a Prolog program that solves the following arithmetic puzzle. The 16 empty squares represent the integers 1 through 16, with no repeats. No value is used more than once. The first row represents the condition,

(**A + B - C) / D = 9**

Your program should report the solution integers in row-major order (row 1, row 2, etc.). With backtracking, your program should report multiple solutions of the puzzle if they exist. (Note: in Sicstus, the "/" operator always returns floats, while "//" returns integers).

	+		-		/		9
-	■	-	■	+	■	+	
	X		+		+		49
+	■	-	■	-	■	/	
	X		-		+		24
/	■	X	■	+	■	-	
	-		+		+		27
2		-6		16		-14	

3. (a) Take the attached code from Clocksin and Mellish which does symbolic differentiation, type it into Prolog, and try it on some example expressions.

(b) Write a new predicate simplify(Old, New). Old is an arithmetic expression as generated by the symbolic differentiation routine in (a). New is a simplified expression. The types of simplifying transformations can look as follows:

E + 0 → E
E * 0 → 0
E - E → 0
E + E → 2*E
E / 1 → E
etc.

Think of a reasonable number of simplifying transformations, and implement them. You can test simplify/2 without using the differentiation routine. In general, you will have better results with your simplifier if you recursively simplify arguments before the parent expression. For example,

```
5*((2*3)-(2*3)) → 5*0   (using "E-E")
                → 0     (using E*0)
```