C SC/MATH 473 **hw 1** **Thu 2 Sep 2010**

DUE: Thu 16 Sep, by 3:15 (in class unless an on-line student)

Reading

Sipser Text, Chapter 1: Section 1.1 Finite Automata and Section 1.2 Nondeterminism

Problems

Clarity and brevity of answers wins points, so revise your work before writing it up for submission. Begin each problem on a separate sheet and state the original problem briefly. Submit your solution as the printout of a PDF file, in the labeled envelope provided in class. (On-line students may submit by email attachment to `pete@cs.arizona.edu`).

1. Relation Properties

Draw directed graphs representing relations of the following types:

(a) Reflexive, transitive and *antisymmetric*. Antisymmetry of R means that $(\forall a,b)$ if $(a,b) \in R$ and $a \neq b$ then $(b,a) \notin R$.

(b) Reflexive, transitive, and *neither* symmetric nor antisymmetric.

2. Closure

What is the the reflexive, transitive closure R^* of $R = \{(a,b),(a,d),(d,c),(d,e)\}$? Express R^* as a set, and also draw a directed graph representing the relation R^*.

3. Ackermann's Function

Consider the following recursively (inductively) defined function, where $m,\ n$ are non-negative integers.

$$\begin{aligned} A(n,\ 0) &= n + 1 \\ A(0,\ m+1) &= A(1,\ m) \\ A(n+1,\ m+1) &= A(\ A(n,\ m+1)\ ,\ m) \end{aligned}$$

In a modern programming language, this function A would be specified more like this:

function $A(n, m) =$ **if** $m = 0$ **then** $A := n+1$ **else** **if** $n = 0$ **then** $A := A(1, m-1)$ **else** $A := A(\ A(n-1, m)\ ,\ m-1)$

Discover a simple, closed-form expressions for the following functions (use inference and experiment to make a discovery), and then *prove* by induction that your expression is correct (use deduction to confirm your discovery).

(a) $A(n,\ 1)$ (b) $A(n,\ 2)$ (c) $A(n,\ 3)$

Caution: If, as part of the discovery process, you write a program to explore this function's behavior, be sure to use only small values of the argument *n*, or we could be waiting forever for you to return. Especially in part (c).

4. False Induction

Point out the flaw in the reasoning below. *Find and point out the precise step or steps that are in error*, and show the error by giving a concrete example that, when traced through the ‘‘proof" above, shows this step is in error. It is recommended that in debugging this proof that you try stepping through the argument with *very small* sets of marbles.

Theorem: All marbles have the same color.

Proof: Let A be any set of n marbles, where $n \geq 1$. The proof is by induction on n.

Base: If $n = 1$ all marbles in A clearly have the same color.

Step: Assume that if A is any set of k marbles, then all marbles in A have the same color. Let A' be a set of $k+1$ marbles, where $k \geq 1$. Remove one marble m from A'. We are then left with a set A'' of k marbles. By the induction hypothesis, A'' has all marbles of the same color. Remove from A'' a second marble m', and then add back to A'' the marble m originally removed. We again have a set of k marbles, which by the induction hypothesis has marbles all the same color. Thus the two marbles m, m' removed must have been the same color, and so the set A' must contain marbles all the same color.

By the principle of induction, applied to the above basis and step, we conclude that in any set of $n \geq 1$ marbles, all marbles are the same color. **QED**.

5. Languages

Show each of the following:

(a) $\{\varepsilon\}^* = \{\varepsilon\}$.

(b) For any language L, $L^*L^* = L^*$.

(c) For any language L, $L^* = (L^*)^*$.

Remember that to show two sets A and B are the same, you have two things to do: (1) show $A \subseteq B$ and (2) show $B \subseteq A$.

6. Scheme

In the *Scheme* programming language†, the syntactic category of ''Numbers'' is (partially) defined as follows‡:

```
<num>     → <prefix><real>
<real>    → <sign><ureal>
<ureal>   → <uint> | <uint>/<uint> | <decimal>
<uint>    → <digit>+
<prefix>  → ε | #d
<digit>   → 0 |  · · ·  | 9
<decimal> → <uint><exp> | .<digit>+<suffix> | <digit>+.<digit>*<suffix>
<suffix>  → ε | <exp>
<exp>     → <marker><sign><digit>+
<sign>    → ε | + | -
<marker>  → e | s | f | d | l
```

(The markers **s, f, d, l** override the default floating point (**e**) representation, in implementations that support short, single precision, double and long.)

Problem: Draw the state diagram that recognizes a well-formed **<num>** in Scheme. Your machine can be non-deterministic. To save space, label edges—where possible—with a *class* of symbols, such as <digit>, rather than introducing an edge for each symbol.

†Dybvig, R.K., *The Scheme Programming Language, 3rd Ed.*, Cambridge: The MIT Press, 2003.

‡Scheme also provides for numbers in radix 2, 8 and 16. In addition, Scheme employs a special ''digit'' called **#** to indicate ''inexact'' digits, and calculations must then report on resulting digits that are inexact given the information provided. For simplicity, we have suppressed everything except the rules for *decimal exact numbers* in this exercise.