Coláiste na Tríonóide, Baile Átha Cliath
Trinity College Dublin
Ollscoil Átha Cliath | The University of Dublin

**Faculty of Engineering, Mathematics & Science**

**School of Computer Science & Statistics**

**Integrated Computer Science** **Sample**
**Computer Science & Business**
**Computer Science & Language**
**Mathematics**

# Symbolic Programming

**Thu, 15 Dec 2022** **RDS SIM COURT** **14:00 – 16:00**

**Dr Tim Fernando**

**Instructions to Candidates:**

Answer both questions. Each question is 50 points (for a total of 100).

You may not start this examination until you are instructed to do so by the Invigilator.

**Materials permitted for this examination:**

Non-programmable calculators are permitted for this examination — please indicate the make and model of your calculator on each answer book used.

## Question 1

(a) Consider the English sentence

(†) *Wizards are magic.*

Let us agree to translate *magic* as a Prolog predicate `magic/1` of arity 1.

(i) Give a Prolog **rule** translating (†), and describe how a Prolog interpreter consulting this rule would respond to the query

```
?- magic(X).
```

[5 marks]

(ii) Give a Prolog **fact** translating (†), and a Prolog query that can be answered on the basis of this fact.

[5 marks]

(iii) Next, consider the English sentence

(‡) *Magic is magic.*

Translate (‡) in Prolog and describe how the Prolog interpreter consulting this translation responds to the query

```
?- magic(X).
```

Can you translate (‡) in Prolog so that the query above does not lead to a loop?

[5 marks]

(b) Recall that the Prolog predicate =/2 is unification without the occurs check. As a result, there is a term `X` such that `X=[X]`. Is the `X` such that `X=[X]` a list, and if so what are its members?

[5 marks]

(c) Next, consider the term `Y` such that `Y=[Y|Y]`. Is this term a list, and if so what are its members?

[5 marks]

(d) Recall that the non-negative integers 0,1,2, ... can be encoded as the numerals `0`, `succ(0)`, `succ(succ(0))`, ... as described in

```
numeral(0).
```

```
numeral(succ(X)) :- numeral(X).
```

To represent the numerals in binary notation, define a binary predicate `n2bs` that converts numerals into bit-strings so that, for example,

```
?- n2bs(0,S).
S = [0] ;
false

?- n2bs(succ(0),S).
S = [1] ;
false

?- n2bs(succ(succ(0)),S).
S = [1,0] ;
false.

?- n2bs(succ(succ(succ(0))),S).
S = [1,1] ;
false.

?- n2bs(succ(succ(succ(succ(0)))),S).
S = [1,0,0] ;
false.
```

For full credit, make sure all recursive predicates you define are tail-recursive.

[25 marks]

## Question 2

(a) Recall that the Prolog predicate `member(X,L)` says `X` is a member of the list `L`.

(i) Give the Prolog clauses that define `member(X,L)`.

[4 marks]

(ii) Let `memb(X,L)` be obtained from `member(X,L)` by putting a cut in the base case.

```
memb(X,[X|_]) :- !.
memb(X,[_|Y]) :- memb(X,Y).
```

Give Prolog's answer to the query

```
?- findall(X,memb(X,[1,2,3],L).
```

[3 marks]

(iii) Another variant of `member(X,L)` is the predicate `me(X,L)` obtained by putting a cut in the inductive case.

```
me(X,[X|_]).
me(X,[_|Y]) :- me(X,Y), !.
```

Give Prolog's answer to the query

```
?- findall(X,me(X,[1,2,3],L).
```

[3 marks]

(b) Consider the regular expressions over the alphabet $\{1, 2\}$. An example, with alternation (or choice) written $|$ (also sometimes written $+$), is $1(1|22)^*22$ which picks out the set of strings of the form

$$1^{n_1}2^{2m_1}1^{n_2}2^{2m_2}\cdots 1^{n_k}2^{2m_k}$$

for some positive integer $k$, and positive integers $n_1, m_1, n_2, m_2, \ldots, n_k, m_k$. For example, the shortest string in this set is 122, which we shall represent in Prolog as the list `[1,2,2]`.

Define a DCG that generates the aforementioned set of strings so that, for example,

```
?- s([1,2,2,1,1,1,2,2],L).
L = [1,1,1,2,2] ? ;
```

```
L = [] ? ;
false
```

[10 marks]

(c) To generalize the construction of the DCG in part (b) to arbitrary regular expressions over the alphabet $\{1, 2\}$, let us agree to use the binary functors `c`, `a` and `k` for concatenation, alternation and Kleene star (respectively) so that, for example, $1|22$ can be encoded as `a(1,c(2,2))`, and $(1|22)^*$ can be encoded as `k(a(1,c(2,2)))`. For completeness, let us use the constant `e` for the empty set (consisting of no strings), and `n` for the set consisting (solely) of the string `[]` of length 0. Now, the idea is to add an argument to the symbol `s` in the part (a), which we can fill by any regular expression over $\{1, 2\}$ (under the encoding above) so that, for example,

```
?- s(c(2,2),L,[]).
L = [2,2] ? ;
false
?- s(a(1,c(2,2)),L,[]).
L = [1] ? ;
L = [2,2] ? ;
false.
?- s(k(a(1,c(2,2))),[1,2,2],T).
T = [1,2,2] ? ;
T = [2,2] ? ;
T = [] ? ;
false.
```

Define a DCG for this 3-ary predicate `s/3` that works for all regular expressions over the alphabet $\{1, 2\}$.

[15 marks]

(d) A regular expression such as $1^*2^*$, encoded above as `c(k(1),k(2))`, has infinitely many strings, not all of which may appear as Prolog answers the query below.

```
?- s(c(k(1),k(2)),L,[]).
L = [] ? ;
L = [2] ? ;
L = [2,2] ? ;
```

```
L = [2,2,2] ? ;
...
```

Missing from the enumeration above is `[1,2]` even though

```
?- s(c(k(1),k(2)),[1,2],[]).
true.
```

Revise the predicate `s` to a predicate `sr` so that for any regular expression *R* and any string *x* in *R*, we need only type ; enough times, as the Prolog interpreter processes the query `sr(`*R*`,L)` before `L` is set to *x*. For example, the string `[1,1,1,2,2]` should be bound to `L` at some finite point below.

```
?- sr(c(k(1),k(2)),L).
L = []  ;
...
L = [1,1,1,2,2]
```

[15 marks]