CSU34011 Symbolic Programming

Second of Two Assessed Assignments

Submit to Blackboard by Fri, Nov 15 (23:59).
For full marks, make your code as simple as possible.

Problem 1 (20 points) Write a DCG for a 3-ary predicate `odd(L,A,B)` that given a list `L`, encodes all the strings of odd length over the alphabet `L` as difference lists `A-B`. For example,

```
?- odd([0,1],[1,0,1],[]).
true.

?- odd([a,b],[a,b,b,b,c],B).
B = [b,b,b,c] ;
B = [b,c] ;
false.

?- odd([c],[c,c],[]).
false
```

Problem 2 (20 points) The DCG below encodes all bit strings as difference lists `A-B` such that `s(A,B)`.

```
s --> [].
s --> a, s.
a --> [0].
a --> [1].
```

Notice we get only strings of `0` from the Prolog query

```
?- s(L,[]).
L = [] ;
L = [0] ;
L = [0,0] ;
L = [0,0,0] ;
...
```

To include strings with `1`, define a unary predicate `p(L)` so that we can enumerate all bit strings in order of increasing length through the query

```
?- p(L), s(L,[]).
L = [] ;
L = [0] ;
L = [1] ;
L = [0,0] ;
```

```
L = [0,1] ;
L = [1,0] ;
L = [1,1] ;
L = [0,0,0] ;
L = [0,0,1] ;
...
```

Problem 3 (20 points) The nth *Fibonacci number* F_n is, for any integer $n \geq 0$, defined by

$$\begin{aligned} F_0 &:= 0 \\ F_1 &:= 1 \\ F_{n+2} &:= F_n + F_{n+1} \end{aligned}$$

giving $F_2 = 1,\ F_3 = 2,\ F_4 = 3,\ F_5 = 5$, etc. Define a DCG that generates for every $n \geq 1$, lists $[F_0, F_1, \ldots, F_n]$ so that, for example,

```
?- fib(L,[]).
L = [0,1] ;
L = [0,1,1] ;
L = [0,1,1,2] ;
L = [0,1,1,2,3] ;
L = [0,1,1,2,3,5] ;
...
```

Problem 4 (40 points) The regular expression

$$(0+1)^*\,1(0+1)(0+1)$$

denotes the set

$$L_3 := \{s \in \{0,1\}^* \mid s \text{ has length} \geq 3 \text{ and its third to the last bit is } 1\}$$

of bitstrings that end with one of the four strings 100, 101, 110, 111 from $1(0+1)(0+1)$. Recall from lecture that the predicate `accept/1` defined below is true of strings accepted by a finite automaton with transitions given by `tran/3` and final states given by `final/1`.

```
accept(L) :- steps(q0,L,F), final(F).
steps(Q,[],Q).
steps(Q,[H|T],Q2) :- tran(Q,H,Qn), steps(Qn,T,Q2).
```

Define the predicates `tran` and `final` to accept precisely the strings in L_3 so that, for example,

```
?- accept([0,0,Z,0,0]).
Z = 1 ;
false.
```

Turn your transitions into a DCG for L_3 so that, for example,

```
?- q0([0,0,Z,0,0],[]).
Z = 1 ;
false.
```

Finally, define a predicate `l3(String,Numeral)` that holds if `String` belongs to L_3 and has length `Numeral` and `numeral(Numeral)`, where

```
numeral(0).
numeral(succ(X)) :- numeral(X).
```

For example,

```
?- l3(String, succ(0)).
false.

?- l3(String, succ(succ(succ(succ(0))))).
String = [0, 1, 0, 0] ;
String = [0, 1, 0, 1] ;
String = [0, 1, 1, 0] ;
String = [0, 1, 1, 1] ;
String = [1, 1, 0, 0] ;
String = [1, 1, 0, 1] ;
String = [1, 1, 1, 0] ;
String = [1, 1, 1, 1] ;
false.
```