/

p4q4
ACN
p11q4

1 Compiler Construction 2000

Consider the grammar:

```
S -> E <eof>
E -> T + E
E -> T
T -> x
```

where S is the starting symbol, `<eof>` is a special token marking end of input and x is a terminal.

Explain and find the left, right and follow sets for all non-terminals in the grammar.

(5 marks)

Suppose that an SLR parser for this grammar is required. One stage on the way to constructing the parsing tables is to create the *characteristic finite state machine* (sometimes known as the LR(0) states). Do this, explaining your working clearly. You do not need to complete the SRL parsing tables.

(10 marks)

Now, imagining that the parsing tables have been constructed show what values will be placed on a stack and comment about internal state as an SLR parser using this grammar processed the input text `x+x+x<eof>`.

(5 marks)

1.1 Marking notes

```
   S -> E <eof>
   E -> T + E
   E -> T
   T -> x

Explanation as per bookwork!

        left     right    follow
for S:  E T x    <eof>
for E:  T x      E T x    <eof>
for T:  x        x        + <eof>


For LR(0) states need to consider
   S -> .E <eof>
   S -> E .<eof>
   E -> .T + E
   E -> T .+ E
   E -> T + .E
   E -> T + E.
   E -> .T
   E -> T.
   T -> .x
   T -> x.
```

```
and our states will be various subsets of these.  See page 64 of Appel
for this very example worked through!


For behaviour of LR parser eating x+x+x$ it has to go something like
   x    reduce T->x
   +    shift
   x    reduce T->x
   +    shift
   x    reduce T->x
        reduce again using E->T
        reduce again using E->T+E
        reduce again using E->T+E
        reduce using S->E$
        accept
```

2 Foundations of Functional Programming

Give a brief account of how **four** of the following features of general programming systems can be modelled in terms of a form of un-typed functional programming where none of the mentioned facilities are provided as built-in features:

1. Tuples (it will be sufficient to consider just the case of pairs);
2. Boolean quantities and an *if/then/else* construct;
3. Lists (both empty and non-empty);
4. Recursive function definitions;
5. The numbers 0,1,2,..., with the associated operations of a zero test, addition and multiplication.

(4 marks for each part attempted)
Select the cases you describe so that in one of the cases your modelling could still be carried out in a polymorphically typed functiona language and in one case it could not, and explain briefly which is which and why.
(4 marks)

2.1 Marking notes

```
If I use ML-like syntax

fun p1 a b = a;  // projection functions
fun p2 a b = b;

fun pair a b f = f a b;
```