## 2. Prolog (30 marks total)

**2.1 Unification** (8 marks)
What is the first result of the following queries in Prolog? Give the overall outcome (circle one of: **yes** if query succeeds, **no** if query fails, **infinite-loop**, or **other-error**) as well as the variable bindings. Use names such as _1, _2, _3,... for newly created variables, if necessary. Assume that no user-defined program has been loaded before executing your queries.

Example: **?- Z=0.**
outcome: yes
bindings: Z=0

**2.1.1** (2 marks)

**?- p(X) = p(5).**

outcome: yes

bindings: X=5

**2.1.2** (2 marks)

**?- X = 20, f(Y) = f(g(Z)), Z = g(T).**

outcome: yes

bindings:
```
X = 20
Y = g(g(_1))
Z = g(_1)
T = _1
```

**2.1.3** (2 marks)

**?- X is 5+3, Y=X+X.**

outcome: yes

bindings:
```
X = 8
Y = 8+8
```

**2.1.4** (2 marks)

**?- X = 5+3, Y is X+Z.**

outcome: other-error

bindings: (none)

**2.2 Understanding a Prolog Program and Backtracking** (8 marks)
Consider the following Prolog program:

```
tag(1).
tag(2).
label(a).
label(b).
label(c).

p([]).
p([T,L|R]) :- tag(T), label(L), p(R).
q([L|R]) :- label(L), p(R).
r(X) :- p(X).
r(X) :- q(X).
```

What is the first result for the following Prolog query, and what are the results of asking for four more answers by pressing semicolon **;** four times?

**2.2.1** (2 marks)

**?- p(X).**

first: `X = []`

next four:

```
X = [1,a] ;
X = [1,a,1,a] ;
X = [1,a,1,a,1,a] ;
X = [1,a,1,a,1,a,1,a]
```

**2.2.2** (2 marks)

**?- q(X).**

first: `X = [a]`

next four:

```
X = [a,1,a] ;
X = [a,1,a,1,a] ;
X = [a,1,a,1,a,1,a] ;
X = [a,1,a,1,a,1,a,1,a]
```

**2.2.3** (2 marks)

**?- r(X).**

first: `X = []`

next four:

```
X = [1,a] ;
X = [1,a,1,a] ;
X = [1,a,1,a,1,a] ;
X = [1,a,1,a,1,a,1,a]
```

**2.2.4** (2 marks) Assume that in the program above, the order of the two clauses for r(X) is swapped:

```
r(X) :- q(X).
r(X) :- p(X).
```

Now what are the results of the query **?- r(X).**

first: `X = [a]`

next four:

```
X = [a,1,a] ;
X = [a,1,a,1,a] ;
X = [a,1,a,1,a,1,a] ;
```

**2.3 Understanding and Changing Prolog Programs** (14 marks)

Consider the following Prolog program:

```
member(A,[A|_]).
member(A,[B|L]) :- A \== B, member(A,L).
a(X,L,L) :- member(X,L), !.
a(X,L,[X|L]).
```

**2.3.1** (6x1 mark)
Determine the first answer for the following queries, and give the bindings for the variables.
If a query fails, explain why.

**2.3.1.1** **?- a(3, [1,2], X).** `X = [3,1,2]`

**2.3.1.2** **?- a(3, [3,1,2], X).** `X = [3,1,2]`

**2.3.1.3** **?- a(3, [1,2,3,3], X).** `X = [1,2,3,3]`

**2.3.1.4** **?- a(3, [1,[3],2], X).** `X = [3,1,[3],2]`

**2.3.1.5** **?- a(3, X, X).** `X = [3|_1]`

**2.3.1.6** **?- a(3, X, [2|Y]).** `X = [2,3|_1], Y = [3|_1]`

**2.3.2** (5 marks)
Write a Prolog predicate **a1** that computes exactly the same first answer as **a** but does *not* use the cut. You must code all the helper functions that you may want to use, except for **member**.

```
not_member(X, []).
not_member(X, [Y|L]) :- not_member(X, L), X \== Y.

a1(X, L, L) :- member(X, L).
a1(X, L, [X|L]) :- not_member(X, L).
```

**2.3.3** (1 mark)
What about the efficiency of the two predicates? Circle one of the three given answers.
a) Predicate **a** is more efficient.
a1) Predicate **a1** is more efficient. **a** is more efficient
same) Both are about the same.

**2.3.4** (2x1 mark) answer true or false.

**2.3.4.1** false Predicate **a** can generate more than one answer with backtracking.
**2.3.4.2** true Predicate **a1** can generate more than one answer with backtracking.

## 3. Constraint Programming (20 marks total)
## 3.1 Completing a Constraint Program (10 marks)

Professor C.S.Train had to schedule five meetings on the same day, and solved this problem by using Sicstus Prolog and the CLPFD package.
She used the following 5 variables to represent the starting times of meetings:
S1 .. meet student1 (duration: 1 hour)
S2 .. meet student2 (duration: 2 hours)
R .. research group meeting (duration: 2 hours)
L .. lunch with colleagues (duration: 1 hour)
B .. breakfast with visitor (duration: 1 hour)

Here are the constraints:
- all meetings start at full hours (e.g. 8:00, but not 8:30)
- no meetings before 8:00
- all meetings must end at 19:00 at the latest
- order: breakfast first, meet student1 before lunch, student2 after lunch, research meeting last.
- lunch should start either at 11:00, 12:00, or 13:00.

**3.1.1** (2x1 mark)
Compute two different solutions to the constraint problem by hand.

Solution 1: S1 = S2 = R = L = B =
many solutions, e.g. M = [9,13,15,12,8]

Solution 2: S1 = S2 = R = L = B =
many solutions, e.g. M = [9,13,16,12,8]

**3.1.2** (8 marks)
Here is Professor Train's program, with some parts left out. Write the missing parts in the space provided after the program.

```
MISSING-PART-1    % load CLPFD package

todaysmeetings(M) :-
      M = [S1,S2,R,L,B],      % list of all meetings
      MISSING-PART-2,    % define variables and domains
      constrain(M),              % check given constraints
      MISSING-PART-3.    % solve constraint problem (use default settings)

constrain(M) :-
      MISSING-PART-4.    % check given constraints
```

Write the missing parts here:
**3.1.2.1** (1 mark) MISSING-PART-1
```
:- use_module(library(clpfd)).
```

**3.1.2.2** (2 marks) MISSING-PART-2
```
domain(M,8,17)
```

**3.1.2.3** (1 mark) MISSING-PART-3
```
labeling([],M)
```

**3.1.2.4** (4 marks) MISSING-PART-4
```
      M = [S1,S2,R,L,B],
B #< S1, S1 #< L, L #< S2, S2 #< R - 1, L #>= 11, L #=< 13.
```

**3.2 Questions on Constraint Programming** (10x1 mark)

Are the following statements true or false? Circle **true** or **false** only if you know the answer. In the questions, CLP stands for "constraint logic program" or "constraint logic programming".

**3.2.1** true — Prolog's backtracking mechanism is useful for a CLP solver.

**3.2.2** false — CLP is a more general problem-solving method than logic programming.

**3.2.3** false — All variables in a constraint problem have the same domain.

**3.2.4** true — A variable domain in CLPFD must be a range of consecutive integers.

**3.2.5** true — In principle, constraint problems could be solved by using C or Java.

**3.2.6** true — Current CLP systems are useful for industrial applications.

**3.2.7** false — A problem with two contradictory constraints can still have a solution.

**3.2.8** false — CLP is designed to replace all previous forms of logic programming.

**3.2.9** true — A CLP can be 1 million times more efficient than Prolog backtracking.

**3.2.10** false — CLPFD contains one fixed strategy for assigning values to variables.