Midterm Exam, COMP3031, Fall 2021

Date	Oct 19, 2021 Tuesday
Time	9:00-10:20
Instructions:	(a) This exam is closed-book. Ask the instructor if you have any questions. (b) Write ALL answers in the exam book, or in blank papers for NIHK students.

	Problem	Points
Name:	Problem	Points
Student ID:	1.	
ITSC Account:	2.	
	3.	
	4.	
	5.	

Total:

Problem 1 (10 pts) SML function type and execution.

(a) Deduce the type of the function n. Briefly show the steps of your deduction.

```
fun n [] x = []
| n ((x1,x2)::t) x =
if x1 = x then x2::(n t x) else n t x;

Answer:
```

(b) Deduce the output of the following SML expression. Show the subsequent steps after the given first step.

```
 n [(1,2), (1,3), (2,3)] 1;

Answer:
n [(1,2), (1,3), (2,3)] 1;
= n ((1,2)::[(1,3),(2,3)]) 1;
=
```

Problem 2 (15 pts) More SML function type and execution.

(a) Deduce the function type of m. Briefly show the steps of your deduction.

```
 fun m [] n x f = x |
     m (h::t) n x f =  if (f n h) > x
                       then m t n (f n h) f
                       else m t n x f;
```

Answer:

(b) Deduce the output of calling function m. Show the steps of executing m (no need to show the execution of c).

```
fun c [] x = 0 |
    c (h::t) x = if x=h then 1+(c t x) else c t x;
m [true, false, true] [true, false] 0 c;
```

```
Answer:
m [true, false, true] [true, false] 0 c;
=
```

Problem 3 (30 pts) Write SML functions. Use the following data types defined in Assignment1:

```
datatype flight = F of int * int;
datatype flights = Fs of flight list;
```

The data type *flight* is constructed on a two-element tuple consisting of integers x and y representing a direct flight from a city x to another city y. Assume x and y in the tuple in *flight* are of different values. For example, the tuple $(0, 1)$ represents a direct flight from city 0 to city 1. The data type *flights* is constructed on a list of elements of the *flight* type. Assume all tuples in the flight list are unique.

Also, you can assume the function *reachable* is provided: *reachable*$(fs, (a, b))$ is true if there is a direct flight or connecting flights from city a to city b in fs.

```
val reachable = fn : flights * (int * int) -> bool
```

(a) `val search_sources = fn : flights * int  -> int list`

Write a function *search_sources* that returns a list of cities from each of which the given city *dst* is reachable. All cities in the output list are unique, but the order of the cities in the output list is unimportant.

Examples:

```
- search_sources(Fs [], 0);
val it = [] : int list

- search_sources(Fs [F(0,1), F(1,0)], 0);
val it = [0,1] : int list

- search_sources(Fs [F(0,2), F(1,0), F(2,1)], 0);
val it = [1,2,0] : int list
```

(b) `val is_connected = fn : flights -> bool`

This function will return true if for any pair of cities x and y in the flights, x is reachable to y or y is reachable to x.

Examples:

```
- is_connected(Fs []);
val it = false : bool

- is_connected(Fs [F(0,1), F(1,2), F(2, 0)]);
val it = true : bool

- is_connected(Fs [F(0,1), F(1,0), F(2,3), F(3,2)]);
val it = false : bool
```

Problem 4 (15 pts) Consider the following grammar in BNF with `<S>` being the starting symbol:

```
<S>::= [<empty>] | [<S1>]
<S1>::= F (<D>, <D>) | F (<D>, <D>), <S1>
<D>::= <D2> | <D1><D>
<D1>::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<D2>::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

(a) Determine whether the string "[F(0,10), F(10,2), F(2, 0)]" belongs to the language generated by the grammar. If your answer is yes, draw a parse tree of the string based on the BNF grammar; If your answer is no, just say so and no explanation is needed.

(b) Determine whether the language generated by the grammar is a regular language. If your answer is yes, write a regular expression to represent this language; if your answer is no, just say so and no explanation is needed.

Problem 5 (30 pts) Consider the following definition of type expressions:

```
- "A", "B", "C", "D", "X", "Y", "Z" are type expressions.
- Given type expression A, A* is a type expression.
- Given type expression A, A& is a type expression.
- Given type expression A and B, A::B is a type expression.
- Given type expressions A, B, and C, A[B,C] is a type expression.
```

The operators of these type expressions observe the following rules in **decreasing precedence** (operators on the same line have the same level of precedence):

```
::          (right associative)
* & [,]       (left associative)
```

(a) Write an **unambiguous** context-free grammar in BNF for such type expressions, preserving the precedence and associativity of the operators.

(b) Draw the **tree representation** of the following type expression:

`A::B[X[Y*,Z],A::C]::D&`

Extra Page