SC-T-501-FMAL Programming languages
Final exam
Spring 2018

Teacher: Tarmo Uustalu

Write answers in English or Icelandic.

Write clearly.

No phones, no computers.

No books, notes, no other additional material, except for a cheat sheet prepared by the teacher.

You can use additional blank paper for calculations.

This exam gives max 100 p, which will account for 60 pct of your final grade for this course.

Best of luck!

Your name: ____

Kennitala: ____

Multiple-choice questions on all material

Encircle the correct answer. Only encircle one answer. If you encircle two or three, this is 0 p.

1. (2 p) A context-free grammar is unambiguous if

   a. every string has at least one derivation tree
   b. every string has at most one derivation tree
   c. the grammar can be understood by a Martian

2. (2 p) Call by value is

   a. a scope rule
   b. a parameter passing mode
   c. a compiler optimization

3. (2 p) In call by value-result, upon exit from the function

   a. the l-value of the actual parameter is updated
   b. the r-value of the actual parameter is updated (but the l-value remains unchanged)
   c. neither

4. (2 p)
   a. C allows defining arbitrarily nested functions.
   b. It does not.

5. (2 p)
   a. C has explicit pointer types and dereferencing.
   b. C uses the "reference model" for variables of certain types.

6. (2 p) Locks and keys is

   a. a parameter passing mode
   b. a memory safety mechanism
   c. a garbage collection technique

7. (2 p) In reference counting based garbage collection, an object can be removed if its reference counter has become

   a. 0
   b. 1
   c. 0, but further conditions have to be met

8. (2 p) SML has:

   a. type checking, no type inference
   b. both type checking and inference

9. (2 p) The operation + (op+) in SML is

   a. ad hoc polymorphic (i.e., overloaded)
   b. constrained parametrically polymorphic
   c. universally parametrically polymorphic

10. (2 p) SML allows implicit conversions.

    a. Yes
    b. No
    c. Only from booleans to integers

11. (2 p) Prolog is a typed language.
    a. Yes
    b. No

12. (2 p) Prolog's search strategy is:
    a. depth-first
    b. breadth-first
    c. iterative deepening

Context-free grammars

13. (4 p) Consider the following context-free grammar. The terminals are $\{a, b\}$, the nonterminals are $\{S, A\}$, the start symbol is S. The rules are

$$\begin{aligned} &S \to aAAbb \\ &A \to a \\ &A \to ba \end{aligned}$$

Write down all strings derivable in this grammar (you do not have to show the derivation trees).

14. (4 p) Consider the following context-free grammar. The terminals are $\{a, b\}$, the nonterminals are $\{S, A, B\}$, the start symbol is S. The rules are

$$\begin{aligned} &S \to Ab \\ &A \to AB \\ &A \to a \\ &B \to b \\ &B \to (\mathit{empty}) \end{aligned}$$

Draw two different derivation trees for the string abb (there are infinitely many).

Imperative programming

15. Consider the following program in the toy language of the book:

```
{
  int x = 13;

  int p () {
    x = x + 2;
    return (x);
  }

  print (p () - x);
}
```

What is printed under
(a) (2 p) left-to-right evaluation order of expressions,

(b) (2 p) right-to-left evaluation order.

16. Consider the following program:

```
{
  int x = 1;

  void p () {
    x = x + 5;
  }

  {
    int x = 2;
    p ();
    print (x);
  }
  print (x);
}
```

What is printed under
(a) (3 p) the static scope rule,

(b) (3 p) the dynamic scope rule.

17. Consider this program.

```
{
  int x = 8;

  void p (int y) {
    y = y + 3;
    print (x);
    print (y);
 }

  p (x);
  print (x);
}
```

What is printed if parameter passing in this language is
(a) (3 p) call-by-value,

(b) (3 p) call-by-reference,

(c) (3 p) call-by-value-result.

18. (3 p) Consider this program in the version of the toy language with pointer types and explicit dereferencing:

```
{
  int* p = new int;
  int* q = new int;

  *p = 17;
  *q = 42;
  p = q;
  print (*p);
  *p = *q + 1;
  print (*p);
  print (*q);
}
```

What does the program print?

Functional programming

19. (a) (3 p) Normalize this untyped lambda-term:

    $z\,((\lambda x.\, y\, x)\,(u\, v))$

    (b) (3 p) Also normalize this one. Make sure you do the substitution correctly, avoiding free variable capture.

    $z\,((\lambda x.\, y\,(\lambda u.\, x\, u))\,(u\, x))$

20. Consider this SML program.

```
fun f g x = g (x, x)
```

(a) (3 p) What is the type of `f`?

(b) (2 p) What is the value of this expression? (`op* : int * int -> int` is multiplication of integers.)

```
- f op* 5;
```

21. The library functions `map` and `filter` are defined like this.

```
fun map f [] = []
  | map f (x :: xs) = f x :: map f xs

fun filter p [] = []
  | filter p (x :: xs) = if p x then x :: filter p xs else filter p xs
```

(a) (2 p) What is the value of this expression?

```
- filter (fn x => x mod 5 = 0) (map (fn x => x + 3) [7,8,2,5,6]);
```

(b) (2 p) And this expression?

```
- map (fn x => x + 3) (filter (fn x => x mod 5 = 0) [7,8,2,5,6]);
```

22. (4 p) Write a function `applyToSnd : ('a -> 'a) -> ('a list -> 'a list)` that applies the given function to the 2nd element of the given list.

If the list is empty or has only one element, the function should leave it unchanged.

E.g.,

```
- applyToSnd (fn x => x * x) [4,5,6];
val it = [4,25,6] : int list

- applyToSnd (fn x => 0) [17];
val it = [17] : int list
```

23. (4 p) Write a function `predFirst0 : int list -> int` which, given a list of integers, returns the integer preceding the first 0 in the list.

    If there are no 0's, it should return the last element of the list.

    The function can give an error (due to pattern match failure) on the empty list.

    E.g.,

```
- predFirst0 [4,3,0,8,7,0,5];
val it = 3 : int


- predFirst0 [4,3,8,7,5];
val it = 5 : int
```

Logic programming

24. What does Prolog answer to the following queries?

    If a query gives rise to an error, you don't need to know the precise error message. Just say this query gives an error.

    But when a query fails (i.e., returns false), say so. This is not the same as an error.

    Remember that `=/2` only tries to unify the two arguments, but `is/2` first evaluates the 2nd argument arithmetically (which may be impossible, if the argument is not a closed arithmetical expression) and then tries to unify the 1st argument and the value of the 2nd argument.

    (a) (2 p)

```
?- [X, a, c] = [b | W].
```

    (b) (2 p)

```
?- [d, [a, Y]] = [Y, [W]].
```

(c) (2 p)

```
?- X + 4 = 3 + 5.
```

(d) (2 p)

```
?- X + 3 = 2 + Z.
```

(e) (2 p)

```
?- 4 is Y + 3.
```

(f) (2 p)

```
?- Y + 3 is 4.
```

25. (4 p) Consider the following program:

```
b(p).
b(q).

a(X, m).
a(q, Y) :- !, b(Y).
a(X, c) :- b(X).
```

What does Prolog answer to the following query? List all answers, in the order Prolog gives them!

```
?- a(Z, Z).
```

26. (3 p) Write a predicate `ok/1` in Prolog for checking that the given term is a list with five elements whereof the 2nd is the atom `correct` and the 1st and 4th elements are the same. (You do not need recursion for this.)

```
?- ok(2 + X).
false.

?- ok([1,correct,3,1,5,6]).
false.

?- ok([this, correct, that, this, X]).
true.

?- ok([this, correct, this, that, this]).
false.
```

27. (4 p) Write a predicate `nonEmptyList/1` that checks that the given term is a nonempty list, i.e., a list with at least one element. NB! Not only does there have to be a head, but the tail has to be a well-formed list. (You do not need negation, i.e., `\+/1`, for this. But recursion is necessary. You may find it useful to define a helper function `list/1`.)

```
?- nonEmptyList(17).
false.

?- nonEmptyList([a,b,c|d]).
false.

?- nonEmptyList([]).
false.

?- nonEmptyList([hey,this,works,1000]).
true
```