Midterm exam - TDT4165 Programming Languages - 10.25-11.40, October 16, 2012

(With solutions as of 29. october.)

No printed or handwritten aids allowed. Contact during the exam: Øystein Nytrø t.: 91897606
All answers should be **short** and written on ordinary, plain, answer sheets. All programs must be written in Oz. All tasks contribute 1 (one) point. A wrong answer (more than one answer for a task) will result in 0 points. The grade from this exam will only count in the final grade by 30% if it improves the final grade, otherwise it will count as 0%.

Task 1

A grammar …

a. reads a sequence of characters and outputs a sequence of tokens.
b. defines a programming language
c. reads a sequence of tokens and outputs an abstract syntax tree.
d. is a prerequisite for implementing a parser
e. traverses the syntax tree and generates low-level instructions for a real or abstract machine.

Answer: d (a programming language also needs semantics & pragmatics to be fully defined. "b" is not correct because it lacks: "syntax" at the end.)

Task 2

A parser for a programming language …

a. reads a sequence of characters and outputs a sequence of tokens.
b. translates a sequence of characters into a sequence of low-level instructions that can be executed on a machine.
c. will uncover semantic errors in a program.
d. reads code input as text and evaluates and prints the result of executing the code.
e. traverses the syntax tree and generates low-level instructions for a real or an abstract machine.
f. will detect syntax errors.

Answer: f

Task 3

Interpreting the following expression in Mozart….

```
{Browse local Y in local X=[1 2] in X.1 = Y.1 end end},
```

will display:

a. 1
b. [1]
c. nil
d. will display nothing, since it will suspend forever
e. will display nothing, but give a syntax error warning
f. none of the above.

Answer: d (since Y.1 does not yet exist, but try to unify Y = [_] (afterwards) and the suspense will be lifted!)

Task 4

Interpreting the following expression in Mozart…

```
{Browse local Y in  local X = [1 2 ] in Y = X.1|X.2 end end}
```

will display:

a. [1 2]
b. will display nothing, since it will suspend forever
c. will display nothing, but give a syntax error warning
d. 1 | 2
e. will raise a unification error
f. none of the above

Answer: a (since X.2 = [2])

Task 5
Implement a function "{Drop List N}" that removes the first N elements of a List and returns the rest. Remember to handle N=0 (leaving the list intact) and N larger than the length of the list (yielding the empty list).
Answer: For example…

```
fun {Drop List N}
          if N =< 0 then List
          else
           case List of nil then nil
           [] Head|Tail then {Drop Tail N-1}
           end
          end
   end
```

Task 6
What will happen when the following program is evaluated in Mozart?

```
local Dodat X = 2 Y in
      proc {Dodat Z} Z = 1 end
      {ByNeed Dodat X}
      {Browse Y}
      Y = X + 1 end %(1)
```

Answer: Browse prints "3" in all tested Oz implementations. Either, the interpreter notices that X is already bound, so doesn't store a trigger, or Dodat remains uncalled since X is bound at (1). **Since this exact example is explained ambiguously in lecture 13, all answers get full score!**

Task 7
What will happen when the following program is evaluated in Mozart? Explain.

```
local X in
      thread if X==1 then skip else X=0 end end % (1)
      thread if X==0 then skip else X=1 end end % (2)
      {Browse X} end
```

Answer: The first explicit thread, (1) or (2) freezes on using == with an unbound variable, so X does not get bound. Similarly with the other explicit thread. Browse prints "_", - ie. X is unbound, and the two threads is indefinitely suspended.

Task 8
Which one of the following statements about implementations of abstract datatypes are true?
a. *Bundled* implementations restrict access to data fields.
b. *Unbundled* implementations prevent direct access to data fields.
c. In *secure* implementations, operations can be defined separately and externally to the data implementation.
d. *Insecure* implementations allow free access to all operations.
e. None of the above
Answer: a and b are false, because bundledness does not relate to access. d are false, security relates to accessibility of data fields. **c is true**, because bundledness ("defined separately") is not related to security.

Task 9
Which one of the following statements are true?
a. Eager and lazy evaluation are only meaningful with concurrency.
b. Data-driven concurrency is not declarative.
c. Declarative concurrency is not necessarily parallel.
d. Declarative concurrency does not rely on fair scheduling.
e. None of the above.
Answer: c

Task 10
What is the next state of the execution stack after the state:
[(raise B end, {A:v1, B:v2, C:v3}), (C=3, {A:v1, B:v2, C:v3}), (catch E {Browse E} end, {A:v1, B:v2, C:v3}),

({Browse A#B#C}, {A:v1, B:v2, C:v3})] { v1=1, v2=2, v3 }

Answer:

Observe that by itself, catch … is a statement to be skipped.
However, evaluation of a "raise B" statement, includes removal of the stack content up to and including the corresponding catch-statement, and binding of the caught variable. So the next statement after a raise is the statement in the "catch"-statement. Ie.:

*[({Browse E}, {A:v1, B:v2, C:v3, **E:v2**}),*
({Browse A#B#C}, {A:v1, B:v2, C:v3})] { v1=1, v2=2, v3 }

Task 11

Which one of the following statements are true?

a. Oz allows catch-statements with pattern-matching.
b. Oz can only raise throwable literal values.
c. Oz can only raise fully bound values.
d. Exceptions are never declarative.
e. None of the above statements are true.

Answer: a

Task 12

Which of the following programs will run with constant stack size?

1. `fun {Fi A B} if A==0 then B else {Fi A-1 B+A} end end`
2. `fun {Fu A B} if A\=0 then {Fu A-1 B+A} else B end end`

a. Both
b. Neither
c. 1, but not 2
d. 2, but not 1
e. 1, and sometimes 2.

Answer: a

Task 13

Which one of the following statements are true?
Dataflow computation

a. is the same as lazy evaluation
b. implies lazy evaluation
c. requires threads
d. may delay unification
e. cannot delay procedure invocation

Answer: d

Task 14

Implement the function {Map List Fun} that applies Fun to every element of List to return a new list.

Answer: For example:

```
fun {Mymap List Fun}
      case List of nil then nil
      [] Head|Tail then {Fun Head}|{Mymap Tail Fun}
      end
  end
```

Task 15

Given the grammar

<expression> ::= <integer>
| (<expression> <operator> <expression>)
<operator> ::= +|-||/*
<integer> ::= <nonzero digit> { <digit> }
<digit> ::= 0 | <nonzero digit>
<nonzero digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

How many parse trees can be generated for 1+2*3 with this grammar?

Answer: Intended as a trick task, with correct answer 0, none, since the phrase is not an expression (it does not contain parentheses). It can be argued that terminals should be quoted in some way, as some dialects of EBNF. However, in this example, no terminals are quoted, only non-terminals (“<” and “>”) and no other use of “(” or “)” is found, so they must be terminals of the defined language.

However, this is not sufficient, since the same interpretation should be applied to “{” and “}”, leading to integers being defined as containing “{...}”. So essentially, don’t make tasks that tries to fool people by blurring defined and defining language..., you only end up fooling yourself...

So: Both 0 and 2 will get full score on this task.