NTNU
Norwegian University of Science and Technology

Department of Computer and Information Sciences

Only english version available!

Midterm exam
TDT4165 Programming Languages
12.25-13.40, October 17, 2011

No printed or handwritten aids allowed. Contact during the exam: Øystein Nytrø t.: 91897606

Answers should be written on ordinary, plain, answer sheets. There is no special form. Some of the tasks are multiple-choice, while others require you to write a short answer. For each multiple-choice question there is only one correct alternative, and it will give you 1 (one) point. Every wrong answer will contribute 0 (zero) points. For the questions that require a written answer, the maximum number of points will be stated in each task (unless it is 1 point). 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%. The total number of tasks is 15.

Comments to the solutions: Two of the tasks were perceived as ambiguous, so two alternatives will be graded as correct.

Task 1

An interpreter…
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. reads a sequence of tokens and outputs an abstract syntax tree.
d. reads program 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 abstract machine.

Solution: **d**

Task 2

A parser …
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. reads a sequence of tokens and outputs an abstract syntax tree
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.

Solution: **c**

Task 3

Interpreting the following expression in Mozart….

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

will display:
a. 1
b. [1]
c. nil
d. will display nothing but raise an unification error
e. will display nothing, since it is syntactically malformed

Solution: **a**

Task 4 (2 points)

When is a grammar ambiguous? Give an example grammar (no reference to Task 15 allowed).

Solution: Part 1: A grammar is ambiguous when two or more different systematic (eg. leftmost) derivations generate the same sentence form, or equivalently, a sentence form can yield two or more different parse/syntax trees. Part 2: An example grammar that is ambiguous is

S → S + S | S – S | digit

Task 5

Consider the following state in the execution of a program expressed in the declarative kernel language on the abstract machine:

([({A A}, {A→a})], {a→(proc {$ A} {A A} end, {})})

What is the next state?
a. There will be no next state in the execution because of infinite recursion.
b. (nil, {a→(proc{$ A} {A A} end, {})})
c. ([({A A},{A→a})],{a→(proc {$A}{A A} end,{})})
d. ([({A A},{A→a})({A A},{A→a})],{a→(proc {$ A}{A A}end,{})})
e. An exception will be thrown because of unification error

Solution: **c**, continuing forever due to indirect (through parameter) infinite recursion

Task 6

Translate the following partial expression to the proper (which?) kernel language:

```
… lazy fun {Foo X} X end
```

Solution: Correct syntax is `lazy <statement>`, so the answer is that it is not translatable! The Mozart compiler anwers:

```
%** expression at statement position
```

However, any solution that interprets the implicit declaration of the result parameter, and marking it as the trigged variable in a "ByNeed"-call gets full score (underlined). The Mozart compiler produces a named function, instead of as in the lecture, an anonymous thunk, a function that will compute the result when needed.

```
proc {Foo Result1 Result2}
   local Fun1 in
      proc {Fun1 Result3}
         case Result1 of X then
            X = Result3
         end
      end
      {`Value.byNeed` Fun1 Result2}
   end
end
```

Task 7

Given the following state during the execution of a program:

([({X Y R}, {X→a, Y→b, Z→c, R→d})], {a→(proc {$ Y R} R=Y+Z end, {Z→e}) b→5, c→7, d, e→3})

the next execution state be:

a. Termination of computation
b. Suspension of computation
c. *([ ], {a→(proc {$ Y R} R=Y+Z end, {Z→e}), b→5, c→7, d→8, e→3})*
d. *([(R=Y+Z, {Y→b, Z→e, R→d})], {a→(proc {$ Y R} R=Y+Z end, {Z→e}) b→5, c→7, d, e→3})*
e. *([ ], {a→ (proc {$ Y R} R=Y+Z end, {Z→e}), b→5, c→7, d→10, e→3})*

Solution: **d.** The body of the procedure replaces the application, and the environment in the closure is added to the current environment (*E* ∪ *{Z→e}*). The state in c) is the next state after d.

Task 8

Consider the following program, declaration and invocation:

```
declare fun {Dobop L}
   case L of
      _|T then 1 + {Dobop T}
   else 0
   end
end
declare X = 1|2|3|nil

{Browse {Dobop 1|2|3|X}}
```

What will the browse-window show?

a. 3
b. 4
c. 6
d. 7
e. nothing

Solution: **c**

Task 9

We change only the following declaration

```
declare X = 1|2|3|_
```

What will the browse-window show?

a. Nothing, - but an unification error will be raised
b. 4
c. 6
d. 7
e. nothing

Solution: **e,** because the computation freezes

Task 10

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

1. `fun {F1 A B} if A==0 then B else {F1 A-1 B+A} end end`
2. `fun {F2 A B} if A\=0 then {F1 A-1 B+A} else B end end`

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

Solution: **a,** since the last call in each recursion is another call, if it does not terminate. (In program 2., the recursive function should have been “F2”. Points for commenting and answering taking that into account.)

Task 11

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

Solution: **d**

Task 12

Given the following program:

```
declare
fun {FoldL L F U}
   case L
        of nil then U
        [] X|L2 then
           {FoldL L2 F {F U X}}
        end
      end

{Browse {FoldL [1 2 3] fun{$ X Y} Y-X end 0}}
```

What will be displayed?
a. -6
b. 2
c. 6
d. None of the other alternatives
e.
```
%************************ syntax error ***********************
        %**
        %** expression at statement position
        %**
        %** in file ``Oz", line 11, column 34
        %** ------------------ rejected (1 error)
```

Solution: **b**

Task 13

Given a concurrent definition of a function that computes Fibonacci numbers:

```
fun {Fibonacci N}
   if N<2 then N
   else thread {Fibonacci N-1} end
   + thread {Fibonacci N-2} end
   end
```

```
end
```

Which of the following statements about an application of the function Fibonacci to an argument N is correct:

a. if N is 10, then it is not possible that at some time there will be less than 10 threads running concurrently
b. if N is 10, then it is possible that at some time there will be more than 100 threads running concurrently
c. if N is 2, then the total number of threads created during the execution is 4
d. if N is 5, then it is not possible that at some time there will be more than 10 threads running concurrently

Solution: **b.** a and c are certainly false. d is wrong (count threads), so **b** is right. The computation is O(2^n).

Task 14

We declare:

```
     declare fun lazy {CentiPlus N M} N*100 + M end
```

If we execute `{Browse {CentiPlus 10 20}},` what happens?

a. The Browser window opens, but with no result, since the expression partially terminates without a need to generate a result
b. The expression terminates normally, returning 1020, because unifying the result of a call with an undetermined dataflow variable generates a need
c. The expression fails and throws an exception.
d. Nothing at all happens.

Solution: **a & d. a** is correct, but **d** also gives full score, because Mozart was not specificed as execution environment.

Task 15

Given the following Oz-representation (using records and Oz integers) of a grammar for arithmetical expressions:

```
<expr> ::= plus( <expr> <expr> )
     | minus( <expr> <expr> )
     | mult( <expr> <expr> )
     | div( <expr> <expr> )
     | <int>
```

Which statement is correct?

a. All operators have the same precedence
b. The grammar has no left-associative operators
c. The grammar is ambiguous
d. The grammar representation is unsuited, because is not tail-recursive.

Solution: **a & b.** Some may have been surprised about seeing Oz-terms as terminals like "`minus (`" and "`)` ", but this is just another grammar. The grammar is not ambiguous (c) and no grammars are in general "unsuited" (d). The grammar has no "operators" at all, so **a)** was intended as correct, since all of the missing operators have the same precedence, but accepting that as a bit subtle, also **b)** is accepted.