i

Front page

NTNU
Faculty of Information Technology
and Electrical Engineering
Department of Computer Science

Midterm examination paper for **TDT4165 Programming Languages**

Academic contact during examination: *Øystein Nytrø*
Phone: *+47 91897606,* Email: *nytroe@ntnu.no*

Examination date: **October 12, 2020**
Examination time (from-to): **10.00-12.00**
Permitted examination support material: **Code E: None**

This course has only an english version of the exam.
This examination has 17 tasks. All tasks have the same weight (1/17).
Some tasks require program or prose writing.
The tasks are in no particular order wrt. curriculum.
Wrong answers are not scored negatively.

There is an ungraded text-entry at the end of the exam that you can use for comments.

Students will find the examination results (a score 0-100) in Studentweb after scoring has been completed. Please contact the department if you have questions about your results. This midterm counts 33% towards a final score that will be tranformed to a final grade for the entire course.

1

Programming paradigms

Which one of completions to "Declarative programming ..." makes the sentence **false**?
Select one alternative:

- can only be achieved in a declarative programming language.
- is definitionally declarative when written in the declarative sequential kernel language (a subset of Oz).
- is either descriptive, observational or definitional.
- in Oz requires "declare"-sentences. ✔
- makes reasoning about programs easier.

2 Parsing

Select the (most) correct completion of the sentence: 'A syntax analyzer ...'
Select one alternative:

- is required in all programming language translation/interpretation.
- will produce tree-structured output. ✔
- will produce a sequence of tokens
- reads parse-trees
- is not specified by a grammatical definition.

3 Grammar comprehension

Consider the following grammar for statements in a language similar to Oz.

<s> ::= skip
  | <s> <s>
  | local <x> in <s>
  | <x> = <x>
  | <x> = <v>
  | if <x> then <s> else <s>

<s> is a sentence, and is also the start symbol.
<x> is an identifier, as in Oz.
<v> is a value expression, as in Oz.

Which alternative is **not a valid** sentence generated by the grammar?
Select one alternative:

- local A in if A then A=A else A=A
- local A in if B then B=A else A=B
- local A in local B in A=A B=B if A then A=B else B=A
- local A in local B in A=B B=A if A then 3 else 0 ✔
- All alternatives are syntactically valid.

4 Grammar properties

Consider the following grammar for statements in a language similar to Oz.
<s> ::= skip
| <s> <s>
| local <x> in <s>
| <x> = <x>
| if <x> then <s> else <s>

<s> is a sentence, and is also the start symbol.
<x> is an identifier, as in Oz.
<v> is a value expression, as in Oz.

Which one completion of "The grammar is ..." makes the sentence **false?**
Select one alternative:

- context sensitive
- not ambiguous. ✔
- context free
- not regular
- recursive

5 Semantic stacks and procedures

Consider the following state in the execution of a program in the declarative kernel language on the abstract machine (variable names are given as v1, v2, ...

([({A A}, {A->v1})], {v1=(proc{$ A} {A A} end, {})})

What will the next state (if existing) in the execution be?
Select one alternative:

- An error is reported.
- ([], {v1=(proc{$ A} {A A} end, {})})
- The next state is identical, ie. unchanged. ✔
- ([({A A},{A->v1})({A A},{A->v1})],{v1=(proc{$A}{A A} end,{})})
- None of the other alternatives

6

Program comprehension

If you run/consult the following code in Mozart:

```
local Y T Z=2 in
   try
      local    X=bar(Z)  Y=boom  T Z in
         try
            raise X end
            Z = 1
         catch bar(X) then {Browse a#T#Z} end
      end
   catch bar(X) then {Browse b#T#Z} end end
```

What would the browser window show?

Select one alternative:

- a#_#baz
- None of the other alternatives.
- a#_#2
- b#_#1
- a#_#_ ✔

7

Identifier scopes in Oz etc.

Which completion of "Oz has..." is true?
Select one alternative:

- no scope rules.
- static typing.
- no typing.
- dynamic scoping.
- static scoping. ✔

8 Paradigm understanding

Complete with the correct alternative: 'Dataflow computation ...'
Select one alternative:

- is the same as lazy evaluation.
- implies lazy evaluation.
- requires exceptions.
- may delay unification. ✔
- is not declarative

9 List representation

Given the Oz values

1. [1 2 3]
2. 1|2|3
3. '|'(1 '|'(2 '|'(3 nil)))

Which of the values represent the same data structure?
Select one alternative:

- None.
- 1 and 2.
- 2 and 3.
- 1 and 3. ✔
- All.

10 Semantic stack and procedures

Consider the following state in the execution of a program in the declarative kernel language on the abstract machine (variable names are given as v1, v2, ...

([({X Y R}, {X → v1, Y → v2, Z→v3, R→v4})],
{v1 = (proc {$ Y R} R=Y+Z end, {Z→v5}), v2=5, v3=7, v4, v5=3})

observe that the formal and actual parameter **identifiers** for the procedure value are equal.

What will the next state be?
Select one alternative:

- ([ ], {v1=(proc {$ Y R} R=Y+Z end, {Z→v5}), v2=5, v3=7, v4=8, v5=3})
- ([ ], {v1=(proc {$ Y R} R=Y+Z end, {Z→v5}), v2=5, v3=7, v4=10, v5=3})
- ([(R=Y+Z, {Y→v2, Z→v5, R→v4})], {v1=(proc {$ Y R} R=Y+Z end, {Z→v5}), v2=5, v3=7, v4, v5: ✔)
- Computation will terminate.
- Computation will suspend/freeze.

11 Explain run time behaviour

Explain the important computational and efficiency features of the implementation of the function Reverse (and implicitly Reverse2) as shown below. (Do not translate Oz to prose!)

```
declare Reverse Reverse2
   fun {Reverse2 Rs Ys}
      case Ys
      of nil then Rs
      [] Y|Yr then {Reverse2 Y|Rs Yr} end
   end
   fun {Reverse Xs} {Reverse2 nil Xs}
   end
```

Write no more than 5 lines of text.

- tail recursive, thus constant stack size
- time used linearly proportional to length of input list

12 Higher-order program comprehension

Given the following definitions

```
declare FoldR FoldL G1 G2

fun {FoldR X F S}
   case X of E|Xr then {F E {FoldR Xr F S}} else S
   end
end

fun {FoldL X F Ac}
   case X of E|Xr then {FoldL Xr F {F Ac E}} else Ac
   end
end

fun {G1 L R} L|R end

fun {G2 L R} R|L end
```

Which of the following calls will give the result [1 2 3]?

Select one alternative:

- {FoldL [1 2 3] G1 nil}
- {FoldL [1 2 3] G2 nil}
- {FoldR [1 2 3] G2 nil}
- {FoldR [1 2 3] G1 nil} ✔
- None of the other alternatives.

13 Program comprehension

What is the result of feeding the following program to Mozart?

```
declare Bar X Y

fun {Bar X Y}
   (A#B)#(C#D) = X#Y
in
   B=C
   A#D
end

{Browse {Bar [f o o | X]#X Y#Y}}
```

Select the correct alternative

- [f o o]
- [f o o | _]#_ ✔
- No reaction (it will suspend)
- Unification error during runtime.
- None of the other alternatives

14 Programming with higher-order Programming

Rewrite the following program fragment so that it only uses FoldL (ie. replace the Map-function):

```
Ys={FoldL {Map Xs F} G S}
```

Fill in your answer here

Solution. The idea is to apply F before G is applied by FoldL:

```
Ys={FoldL Xs fun {$ S X}
                {G S {F X}}
             end S}
```

It is analogous for FoldR (beware of the order of arguments for G).

15 Higher order programming properties

Four basic principles underlie higher-order programming. Map Term with definition

Please match the values:

	Genericity	Instantiation	Procedural abstraction	Embedding
the ability to convert any statement into a procedure value.	○	○	○ ✔	○
the ability to return procedure values as results from a procedure call.	○	○ ✔	○	○
the ability to pass procedure values as arguments to a procedure call.	○ ✔	○	○	○
the ability to put procedure values in data structures.	○	○	○	○ ✔

16 Unification

If you feed
  declare X Y = X#Y {Browse Y}
to Mozart, what will happen?

Select one alternative:

- None of the other alternatives.
- It will complain that Y is not introduced.
- It will show 'X#Y'
- It will show '_#_'
- It will show something like '_#(_#(,,,#,,,))' ✔

17 Programming with difference lists

In the textbook, Difference lists are explained like:

3.4.4 Difference lists

A difference list is a pair of two lists, each of which might have an unbound tail. The two lists have a special relationship: it must be possible to get the second list from the first by removing zero or more elements from the front. Here are some examples:

```
X#X                 % Represents the empty list
nil#nil             % idem
[a]#[a]             % idem
(a|b|c|X)#X         % Represents [a b c]
(a|b|c|d|X)#(d|X)   % idem
[a b c d]#[d]       % idem
```

A difference list is a representation of a standard list. We will talk of the difference list sometimes as a data structure by itself, and sometimes as representing a standard list. Be careful not to confuse these two viewpoints. The difference list `[a b c d]#[d]` might contain the lists `[a b c d]` and `[d]`, but it represents neither of these. It represents the list `[a b c]`.

Define the function {AppendD DL1 DL2} that computes a difference list which is the difference list DL2 appended to DL1.

```

fun {AppendD DL1 DL2}
    (A#B)#(C#D) = DL1#DL2 in
    B = C
    A#D
end
```

18 Comments

This section is not marked or graded, leave comments or clarifications if needed here.