



Faculty of Information Technology
and Electrical Engineering
Department of Computer Science

Examination paper for **TDT4165 Programming Languages**

Academic contact during examination: *Øystein Nytrø*

Phone: +47 91897606, Email: *nytroe@ntnu.no*

Examination date: **December 17, 2018**

Examination time (from-to): **09.00-13.00**

Permitted examination support material: **Code C: Specified (NONE) printed and hand-written support material is allowed. A specified basic calculator is allowed (but not at all needed).**

This course has only an english version of the exam.

This examination has 21 tasks. All tasks have the same weight (1/21). Wrong answers are not scored negatively.

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

Students will find the examination results in Studentweb. Please contact the department if you have questions about your results. The Examinations Office will not be able to answer this.

- 1 Let $a :: A$, $f :: A \rightarrow B$ and let x, y, z denote type variables. Infer the type of

$f\ a$

Select one alternative:

- ☒ B
- ☐ $A \rightarrow B$
- ☐ $B \rightarrow A$
- ☐ A

Maximum marks: 1

- 2 Let $a :: A$, $f :: A \rightarrow B$ and let x, y, z denote type variables. Infer the types of

$\text{map } f$

Select one alternative:

- ☐ [B]
- ☐ $[A \rightarrow B]$
- ☒ $[A] \rightarrow [B]$
- ☐ [A]

- 3 Let $a :: A$, $f :: A \rightarrow B$ and let x, y, z denote type variables. Infer the type of $\lambda u \rightarrow a$

Select one alternative:

- ☐ A
☒ $x \rightarrow A$
☐ Does not compile, or makes no sense
☐ $B \rightarrow A$

Maximum marks: 1

- 4 Let $a :: A$, $f :: A \rightarrow B$ and let x, y, z denote type variables. Infer the type of $\lambda u \rightarrow \lambda v \rightarrow (u, v)$

Select one alternative:

- ☐ $x \rightarrow y \rightarrow (y, y)$
☐ $x \rightarrow y \rightarrow (x, x)$
☐ $z \rightarrow z \rightarrow (z, z)$
☒ $x \rightarrow y \rightarrow (x, y)$

Maximum marks: 1

- 5 Let $a :: A$, $f :: A \rightarrow B$ and let x, y, z denote type variables. Infer the type of uncurry from the definition:
 $\text{uncurry } f(u, v) = f \ u \ v$

Select one alternative:

- ☐ $\text{uncurry} :: (x \rightarrow y) \rightarrow x \rightarrow y$
☒ $\text{uncurry} :: (x \rightarrow y \rightarrow z) \rightarrow (x, y) \rightarrow z$
☐ $\text{uncurry} :: (x \rightarrow y \rightarrow (x, y)) \rightarrow z$
☐ $\text{uncurry} :: (x, y) \rightarrow z \rightarrow (x \rightarrow y \rightarrow z)$

Maximum marks: 1

- 6 Let $a :: A$, $f :: A \rightarrow B$ and let x, y, z denote type variables. Infer the type of $\text{curry } f \ u \ v = f(u, v)$

Select one alternative:

- ☒ `curry :: ((x,y) -> z) -> x -> y -> z`
- ☐ `curry :: ((x,y) -> z) -> z`
- ☐ `curry :: (x -> y -> z) -> (x,y)`
- ☐ `curry :: (x -> y -> z) -> (x,y) -> z`

Maximum marks: 1

- 7 Let `g :: (x,y) -> z`, and let `a :: (x,y)` in Haskell, and let `x, y, z` denote type variables. Which of the following expressions does not evaluate to `g a`?

Select one alternative:

- ☐ `(uncurry $ curry g) a`
- ☐ `head . map g . (\x -> [x]) $ a`
- ☐ `(\x -> g x) $ a`
- ☒ `g . (\x -> (x,x)) $ a`

Maximum marks: 1

- 8 Given

`f :: [a] -> a``f [x] = x``f (x:xs) = f xs`

then

`f [1,2,3]`

evaluates to

Select one alternative:

- ☒ 3
- ☐ [3]
- ☐ [1,2,3]
- ☐ 1

Maximum marks: 1

- 9 The definition of `f` as

`f :: [a] -> a``f [x] = x``f (x:xs) = f xs`

has one important flaw. What is it?

Select one alternative:

- ☒ The function is only partially defined over lists.
- ☐ Unused variables should not be bound
- ☐ Implicit recursion should be avoided
- ☐ f crashes when input is not a list.

Maximum marks: 1

10 The definition of f as

f :: [a] -> a**f [x] = x****f (x:xs) = f xs**

has one important flaw. How would you define a better version?

Select one alternative:

- ☐ the signature should be Num a => [a] -> a
- ☒

```
f :: [a] -> Maybe a
f [] = Nothing
f [x] = Just x
f (_:xs) = f xs
```
- ☐ f = foldl (\x y -> [y]) []
- ☐ Second pattern-match should be (_:xs)

Maximum marks: 1

11 Recall the foldr-function which folds lists and other foldable structures from the right. I.e:

foldr (-) 0 [1,2,3] will compute **(1 - (2 - (3 - 0))) = 2**

Which of the following is the signature of foldr?

Select one alternative:

- ☒ foldr :: Foldable t => (a -> b -> b) -> b -> t a -> b
- ☐ foldr :: Foldable t => (a -> a -> a) -> a -> t a -> a
- ☐ foldr :: Foldable t => (a -> b -> a) -> t b -> a -> t a
- ☐ foldr :: Foldable t, Num a => (b -> a -> a) -> a -> t b -> a

Maximum marks: 1

12 Closely related to foldr is foldl, which folds from the left, e.g:

foldl (-) 0 [1, 2, 3] will yield **((0 -1) -2) -3 = -6**

Given f

f :: [a] -> a**f [x] = x****f (x:xs) = f xs**

try to define it equivalently using foldl.

Which alternative is correct?

- ☐ f xs = foldl (\b a -> a) 0 xs
- ☒ f (x:xs) = foldl scnd x xs where scnd a b = b
- ☐ f l@(x:xs) = foldl (\b a -> b) x l
- ☐ f xs = foldl (\ys y -> y : ys) [] xs

Maximum marks: 1

13 The following is a fragment of Scala code:

```
object ThreadStuff extends App {
  val t = thread { log("one") }
  log("two")
  log("three")
  t.join()
  log("four")
}
```

What is correct?

Select one alternative:

- ☐ The log will show the sequence "one two three four"
- ☐ The log will show the sequence "two three one four"
- ☒ The log will always show "one" before "four"
- ☐ The sequence cannot be predicted

Maximum marks: 1

14 The following is a fragment of Scala code:

```
object MoreThreadStuff extends App {
  var loc: String = "snow"
  val t = thread { loc = "ice" }
  t.join()
  log(loc)
}
```

Select the correct answer:

Select one alternative:

- ☒ The log will show "ice"
- ☐ The log will show either "snow" or "ice", determined by the implementation
- ☐ The log may show either "snow" or "ice" nondeterministically
- ☐ The log will show "snow"

Maximum marks: 1

15 A race condition in a concurrent program is:

Select one alternative:

- ☐ The situation in which different threads depletes each others resources
- ☐ The situation equivalent to a deadlock
- ☐ The conditional test that a thread is not waiting
- ☒ The situation that the effect, or output, of depends on execution scheduling

Maximum marks: 1**16** The abstract "execute" method of Scala's ExecutionContext trait**Select one alternative:**

- ☐ does not reduce memory usage to achieve concurrency
- ☐ takes a runnable and executes it immediately
- ☐ makes concurrency deterministic
- ☒ has a default global execution context

Maximum marks: 1**17** Let s be a string, and consider the following code:

```
val x = scala.util.Try(s.toLong).toEither.left.map(t => 0)
```

Which one is the correct statement?

- ☒ If we want to apply a function f(n: Long) to the parse result, we should use Either's map method
- ☐ If s.toLong succeeds, then x is s.toLong
- ☐ x is a long
- ☐ If s.toLong fails, then x is 0

Maximum marks: 1**18** Implement list reversal in Prolog. Implement list append if you need it.

1	<pre> % Efficient reverse, O(N) accRev([], A, A). accRev([HIT], A, R) :- accRev(T, [HIA], R). reverse1(L, R) :- accRev(L, [], R). % Naïve reverse, O(N**2) append2([], L, L). append2([HIT], L, [HITL]) :- append2(T, L, TL). reverse2([], []). reverse2([HIT], RL) :- reverse2(T, TR), append2(TR, [H], RL). </pre>
---	--

Maximum marks: 1

- 19 Given the following interaction with a Prolog interpreter,
match the line numbers to the right answers (each correct answer counts 1/6 point):

1 :- use_module(library(clpfd)).

2 test(X, Y) :-

3 X in 0..10,

4 Y #> 4,

5 X #> Y.

6 ?- test(X, Y).

7 X in 6..10,

8 Y#=<X+-1,

9 Y in 5..9

Match explanation to the line numbers

	8	4	5	1	9	7	6	3	2
X in the answer is constrained to an interval.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The head of a defined clause (rule)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
Two variables in the answer are constrained by a relationship.	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The variable must be constrained to bounded interval	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
A query to the interpreter.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
A directive to the interpreter	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Maximum marks: 1

20 What answer would the Prolog interpreter give to the unification in the following query?

? - [g(g(Y),X), Z] = [Z, g(A,B)].

Select one alternative:

☐ A = B,
B = X,
X = g(Y),
Z = g(g(Y), g(Y))

☐ A = g(Y),
B = X,
X = Y,
Z = g(g(Y), Y)

☒ A = g(Y),
B = X,
Z = g(g(Y), X)

☐ "No" (ie. the query cannot be satisfied).

Maximum marks: 1

- 21 Define a DCG-grammar that will parse arithmetic expressions (represented as Prolog lists) with operators '*' and '+', and integers as operands. Do not introduce parentheses. Let the grammar accumulate terms representing the parse tree.

Eg.:

?- phrase(E, [1,+,2,*,3,+,4]).

should result in something like:

E = expr(add(1, add(mult(2, 3), 4)))

(depending on your grammar definition).

Fill in your answer here

1	<pre> %% DCG (Equal to assignment 8) expr(T) --> term(T). expr(add(T,E)) --> term(T), [+], expr(E). term(M) --> mult(M). term(mult(M,T)) --> mult(M), [*], term(T). mult(M) --> [M], {number(M)}. %% The following was not required % Must use brackets not already part of Prolog syntax, ie. introduce mult(M) --> [leftbr], expr(M), [rightbr]. % or use '(' mult(M) --> ['(', expr(M), [')'].</pre>
---	--

Maximum marks: 1

- 22 If you have additional comments to any of the questions, this is the place to write them! (This text is not graded in any way.)

Please refer to question number if relevant:

Maximum marks: 0