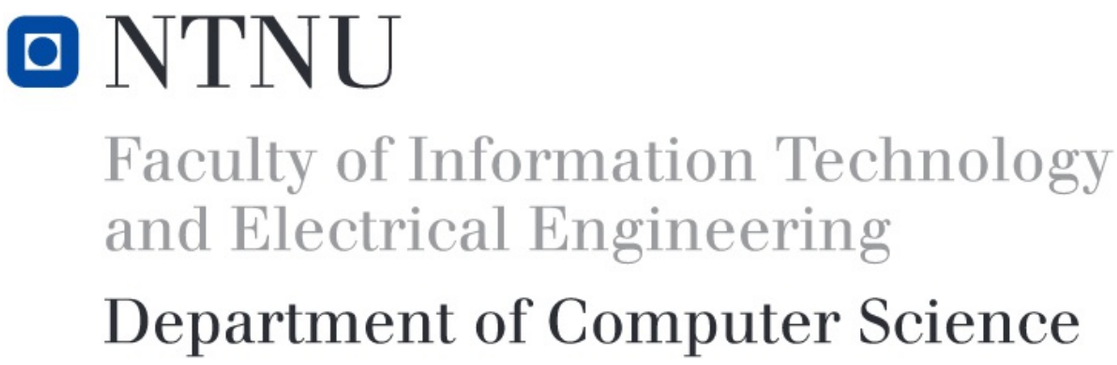
Front page



`Grading aid and solutions to the`
Endterm test for **TDT4165 Programming Languages**

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

Test date: **November 16, 2020**
Test time (from-to): **10.00-12.00**
Permitted support material:
A: All printed and hand-written support material is allowed.
All calculators are allowed, including your Oz-calculator.
Collaboration, stand-ins or person-to-person communication during the test is regarded as cheating.

This course has only an english version of the test.
This test has 11 tasks, of which 6 are multiple choice, 3 require prose explanations and 2 programming.
9 tasks score 10p max, 2 has 5p as max score.
Maximum score is 100p.
Wrong answers are not scored negatively.

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

Your answers will not be available until after scoring is completed. Students will find the results in Blackboard. The score will count 33% towards the final course grade.

1 Threads_MC

Interpret the following according to the formal semantics of Oz and expected behaviour of 'Browse'. What is the most correct desciption of the result?

```
local B in
   thread B=true end
   thread B=false end
   if B then {Browse yes} end
end
```

Select one alternative:

Execution fails always.

Execution may fail.

'yes' or nothing is shown

'yes' is always shown.

Nothing is shown.

Maximum marks: 10

2 Thread sequencing MC

Given the following code, with numbered statements:

```
local B in            %1
   thread             %2
      B=true          %6
   end
   thread             %3
      B=false         %7
   end
   if B               %4
   then {Browse yes}  %5
   end
end
```

Which execution sequence below is not possible, with the program either running until completion, or ending by failing?

Select one alternative

1 2 6 3 4 5 7

1 2 3 7 6

1 2 3 6 4 5 7

1 2 6 3 7 4 5

1 2 6 3 7

Replace one statement to avoid failed computation

Select one alternative

%1 local B=3 in ...(unchanged)

%4 skip

%7 B=false

%6 B=false

%5 skip

Maximum marks: 10

3 Three threads MC

The Browse function runs as a separate thread when called and shows the current value of the parameter in a Browser-window. Which explanation is most correct when interpreting the following snippet in Mozart?

```
local X in
            thread if X==1 then skip else X=0 end end % thread A
            thread if X==0 then skip else X=1 end end % thread B
            thread X=1 end % thread C
            {Browse X}
end
```

Select one alternative:

Nothing is printed and one or two of threads A-C are suspended.

'1' is printed and at least one thread suspends.

'1' or '0' is printed, but that cannot be determined from the code only.

'1' is printed and all thread terminate.

'_' is printed and both threads A and B will be suspended.

Maximum marks: 10

4 Lazyness Prose

Given the definition of R1 and R2:

```
fun lazy {R1 S}
   fun {FnX S R}
      case S of nil then R
      [] X|S2 then {FnX S2 X|R} end
   end
in
   {FnX S nil} end

fun lazy {R2 S}
   fun lazy {FnX S R}
      case S of nil then R
      [] X|S2 then {FnX S2 X|R} end
   end
in {FnX S nil} end
```

What functions do R1 and R2 realise? What is the difference in behaviour between {R1 [1 2 3]} and {R2 [1 2 3]}? Do R1 and R2 calculate the same result? In a lazy program, which of R1 and R2 would you use? Explain your answers.

Fill in your answer here

1) Both R1 and R2 do list reversal upon the need of the function return value. (Both lazy). 3p
2) The difference is that the recursive inner function is also made lazy for R2,
   but this is wasted, since any computation of part of the reversed list requires
   the traversal of the entire list. Cfr. monolithic functions in CTMCP P. 297 2p
3) Both calcualate the same result 2p
4) R1 more efficient, the R2 adds extra thread administration and thus will consume
   more time and resources. 3p

Maximum marks: 10

5 LzyFilt Program

Program the lazy function {LzyFilt Xs F}, where Xs is a list and F a unary function. {LzyFilt Xs F} should return a list which contains all elements of the list Xs for which F returns true.

Fill in your answer here

```
1
```

```
fun lazy {LzyFilt Xs F }
   case Xs  of nil then nil
   [] X|L2 then
      if {F X} then X|{LzyFilt L2 F} else {LzyFilt L2 F} end
   end
end

Scoring:
  Laziness: 1p
  Arguments: 1p
  Application of function: 2p
  Recursion/traversal: 2p
  Overall correctness: 2p
  Simple(st) solution 2p
```

Maximum marks: 10

6

ByNeed MC

Consider the following code snippet in Oz. The interpreter will produce specific output with delay between (we assume you are able to decude what the output is).

```
local X Y in
   {Browse X}
   thread {ByNeed proc {$ A} A = 3 end X} end
   thread {Delay 10} X = Y end
   thread {Delay 10000} if X == Y then {Browse true} end end
end
```

Based on the specific sequence of output, which of the statements below would be the most correct?

Select one alternative:

The first output does not change because it is not needed.

Changing the delay would make the computation non-deterministic.

Exchanging 'true' for '1+X' would not change the first output.

The output indicates that at least one of the explicit three threads is still suspended after the longest delay.

The first output does not change because it is subject to the first unification.

Maximum marks: 10

7 Nondeterministic lazyness with exceptions Programming

The following is an implementation of a useful control abstraction

```
proc {Somethingily S1 S2}
B Y in
   try {S1} B=false catch X then B=true Y=X end
   {S2}
   if B then raise Y end end
end
```

Write an example application of the Somethingily procedure that is nondeterministic (in Oz semantics), by implementing the S1 and S2 procedures using threads or lazyness. Make a short comment explaining how Somethingily, exception handling and S1 and S2 interact.

Fill in your answer here

```
1
```

```
A detailed discussion of this is outside the curriculum, but the actual problem
is quite straightforward when reasoning with the specific code sample.

Somethingily implements try-finally using a flag variable which tries S1,
executes S2 whatever happens to S1, and re-raises whatever S1 may have
thrown if it failed.

If S1 is lazy or a separate thread, it may not throw an exception before S2
has executed. Even worse, if S1 exits at once, S2 will never be executed.
In general, concurrency and exceptions in combinaton is not declarative!

Scoring:
   Recognizing try finally: 4p
   Explaining TryFinally ("the relationship between S1...": 4p
   A sample program containing eg. treads, delays, exits or similar: 2p
   Concurrency and exceptions non-declarativeness: 2p

See also p. 343 in CTMCP, exercise 4.11.
```

Maximum marks: 10

8 Scala safety MC

Which statement about thread safety in Scala is **true**?
Select one alternative:

Mechanisms for ensuring thread safety always increases efficient use of parallellism.

Thread safety is not a problem in Scala programs because the JVM is safe.

Thread safety can be achieved by using "synchronized' in Scala.

Thread safe programs may result in deadlock.

Thread unsafe programs are observationally declarative.

Maximum marks: 10

9 Scala concurrency Prose

In your own words, explain the declarative and non-declarative features and constructs of Scala concurrency by comparing to Oz declarative concurrency.
Fill in your answer here

Format

```
Most of the declarative features of Scala are generally not related to concurrency.
Concurrency in Scala is basically not declarative, since inherited from state-based
synchronization.
However Oz, using logic dataflow variables for almost all concurrency is declarative.

Scoring:
- Basic explanation of declarative concurrency: "A concurrent dataflow program
is declarative if for all possible inputs the following holds. For all
possible executions with a given input, one of two results is possible.
(1) They all do not terminate, or (2) they all eventually reach partial
termination and give results that are logically equivalent." 2p

- Declarativeness may be observational (ie not a language, but program feature): 2p

- State-based communication and "var"-s are not declarative. Oz has by-need,
lazy dataflow computation, …. 2p

- Functional Scala features (functional IO, type system, val-s, comprehensions,
 recursion) enables overall declarativeness, and thus concurrent declarativeness 2p

- Java-features disables declarativeness: OO, mutable variables, infinite domains,
statefulness, iteration, side effects. 2p

(Read more: Doeraene, Van Roy: A New Concurrency Model for Scala Based
on a Declarative Dataflow Core)
```

Words: 0

Maximum marks: 10

10 Copy 2019 MC 5p

Consider the following code written in Scala. Assume that the 'log()'-method will print a string along with the name of the thread calling log. *(This is an identical copy of a 2019 endterm task. Note that you´ll only get 5p for being able to look up the solutions to that test. The next tasks asks you to explain the answers...)*

```
object ThreadsCommunicate extends App {
     var result: String = null
     val t = thread { result = "\nTitle\n" + "=" * 5 }
     // val s = thread { result = "\nTitle\n" + "=" * 2 }
     t.join()
     log(result)
}
```

Which statement is true?

Select one alternative:

't.join' may suspend the thread t.

't.join' ensures that log never prints null.

ThreadsCommunicate is an example of nondeterministic concurrency.

If we uncomment the line starting with '// val s', the program may deadlock.

If we uncomment the line starting with '// val s', t.join ensures that we can predict log output exactly.

Maximum marks: 5

11

Scala Prose 5p

This task was a part of the endterm 2019 (and also in 2020, with very low score):

Consider the following code written in Scala. Assume that the 'log()'-method will print a string along with the name of the thread calling log.

```
object ThreadsCommunicate extends App {
    var result: String = null
    val t = thread { result = "\nTitle\n" + "=" * 5 }
    // val s = thread { result = "\nTitle\n" + "=" * 2 }
    t.join()
    log(result)
}
```

The following alternative statements were provided.

Select one alternative:

- 't.join' ensures that log never prints null.
- 't.join' may suspend the thread t.
- ThreadsCommunicate is an example of nondeterministic concurrency.
- If we uncomment the line starting with '// val s', t.join ensures that we can predict log output exactly.
- If we uncomment the line starting with '// val s', the program may deadlock.

Explain, in your own words, why the alternatives are correct or not correct statements.

Fill in your answer here.

```
ip pr correct explanation

1. "'t.join' ensures that log never prints null" is correct because the join
method suspends  execution until the thread t havs completed execution,
thus ensuring that the var result is non-null.

2." 't.join' may suspend the thread t", is not correct since t.join only
controls the main thread, and not t. The main thread continues
after t has completed.

3."ThreadsCommunicate is an example of nondeterministic concurrency" is not
correct, because  this particular program is observationally deterministic.

4." if we uncomment the line starting with  // val s', t.join ensures that we
can predict log output exactly" is not correct, because the s-thread
may set result both before and after 't.join' has completed, and before
and after the call to 'log'. Ie. the logged result is not possible to predict.
This is a race condition.

5."if we uncomment the line starting with '// val s', the program may deadlock"
is not correct, because there are no shared resources on which to deadlock, ie.
there are no executions with mutual (declared) dependency or locking (eg. two
synchronizations).
```

Maximum marks: 5

12

Comment section

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

Maximum marks: 0