Exam Datalogi II (2G1512)
2003-04-26, 10:00-15:00

Firstname: _____

Lastname: _____

Personnummer: _____

Rules

You are not allowed to bring any material or equipment (such as laptops, PDAs, or mobile phones) with you. The only exceptions are an English to "your favorite language" dictionary and pencils.

Instructions

- The exam has 300 points and takes 300 minutes. The points for each task should help you to judge how much time you use for each task.
- Please read the entire exam first!
- Write on these sheets of paper. Use the free space after each assignment for your answer.
- Write your name and "personnummer" on each page of the exam.

- If you need more space for your answers, use the additional sheets you have been provided with.

  Answers on additional sheets will only be considered if the sheets are marked with your name and “personnummer” and if you have noted in the space left after the question that part of your answer is on an additional sheet.

- You have to hand in the *complete* exam, you are not allowed to take home part of it (this also refers to the extra sheets).

- Write your answers in English or Swedish.

- Tables you might need are at the end of the exam.

Grading

The grades depend on the sum of exam and bonus points n:

$$\begin{array}{rcll} & n & < 150 & \text{fail (U)} \\ 150 \leq & n & < 200 & \text{grade 3} \\ 200 \leq & n & < 250 & \text{grade 4} \\ 250 \leq & n & & \text{grade 5} \end{array}$$

Points

Please do not write here, this is for correcting the exam.

Task	1	2	3	4	5	6	7	8	9	Σ
Max	45	25	20	40	45	20	40	40	25	300
Points										

Bonuspoints:

Totalpoints:

Grade:

Name:_____ Personnummer:_____

1 Questions (45 points)

1. Give a statement $\langle s \rangle$ such that for any environment E the execution of the semantic statement $(\langle s \rangle, E)$ yields a semantic statement $(\langle s \rangle', E)$ for some statement $\langle s \rangle'$ (with other words, the environment E remains unchanged) (6 points).

   Solution. `if true then skip else skip end`

2. Is the **`local`**-statement (variable creation) suspendable (3 points)?

   Solution. No.

3. Assume that a thread has $(\langle s \rangle, E)$ as its topmost semantic statement where $\langle s \rangle$ is a suspendable statement and that execution suspends. What can you say about the activation condition of the statement $\langle s \rangle$ (3 points)?

   Solution. It is false.

4. Give a unary procedure `P` and a value `X` such that execution of `{P X}` never terminates (3 points).

   Solution.

   ```
   proc {P X} {X} end
   proc {X} {X} end
   ```

5. Assume a nullary procedure `P` with no external references. Is it possible that execution of `{P}` never terminates (6 points)?

   Solution. Yes, for example

   ```
   proc {P} proc {Q} {Q} end in {Q} end
   ```

6. Are environments of semantic statements shared among the semantic statements of a single thread (3 points)?

Solution. No.

7. Assume execution of the following statement (where $\langle s \rangle$ is also some statement):

```
local X in
   local Y in skip end
   local Z in
      local Y in
         ⟨s⟩
      end
   end
end
```

Give the variable identifiers that occur in the environment E when the semantic statement $(\langle s \rangle, E)$ is executed (3 points).

Solution. X, Y, Z.

8. Give the external references of the following procedure definition (3 points):

```
P = proc {$ X Y}
       if Z then local U in {X {U} Y} end else U=4 end
    end
```

Solution. Z, U

9. Give the external references of the following procedure definition (3 points):

```
P = proc {$ X Y}
       local Y in {Q {P {X Z} Q}} end
    end
```

Solution. P, Q, Z

10. Consider the following statement $\langle s \rangle$:

```
P = proc {$ X Z}
       if X==Y then Z=a else Z=P end
    end
```

Give the contextual environment created by execution of the semantic statement $(\langle s \rangle, \{\mathsf{Z} \mapsto z, \mathsf{P} \mapsto p, \mathsf{X} \mapsto x, \mathsf{Y} \mapsto y\})$ (3 points).

Solution. $\{\mathsf{Y} \mapsto y, \mathsf{P} \mapsto p\}$

11. Give the value of `U` after execution of the following statement (3 points):

```
local X Y Z U in
   thread if X==2 then Z=2 else Y=3 end end
   thread if Z==3 then U=4 else U=5 end end
   X=2
end
```

Solution. `U` is `5`.

12. Give the value of `Z` after execution of the following statement (6 points):

```
local A B C X Y Z in
   fun {A X}
      false
   end
   fun {B R F X}
      fun {$ G}
         if F==G then X else {R G} end
      end
   end
   fun {C R X}
      {R X}
   end
   X={B {B A a 1} b 2}
   Y={B X a X}
   Z=[{{C Y a} a} {Y c}]
end
```

Solution. `[1 false]`

2 Accumulators (25 points)

The following function `{Pick Xs}` takes a list of lists `Xs` as input:

```
fun {Pick Xs}
   case Xs
   of nil  then nil
   [] X|Xr then
      Y={Pick Xr}
   in
      if {Length X}>{Length Y} then X else Y end
   end
end
```

2.1 Examples (5 points)

What does `{Pick [[a]]}` return?

Solution. `[a]`
What does `{Pick [[a] [b] [b c]]}` return?

Solution. `[b c]`
What does `{Pick [[b a] [b] [b c]]}` return?

Solution. `[b c]`

2.2 Tail-Recursion (20 points)

Give an equivalent tail-recursive function `{PickAcc Xs Y}` that uses `Y` as accumulator.

Solution.

```
fun {PickAcc Xs Y}
   case Xs
   of nil then Y
   [] X|Xr then
      {PickAcc Xr if {Length X}>{Length Y} then X else Y end}
   end
end
```

Name:_____ Personnummer:_____

3 Higher-Order Programming (20 points)

Write a function `{SelectMap Xs F G H}` that takes a list `Xs` and three unary functions `F`, `G`, and `H` as input. It returns a list where the elements are obtained by applying either `G` or `H` to the elements in `Xs` dependening on whether application of `F` to the element has returned **`true`** (in this case `G` is applied) or **`false`** (in this case `H` is applied).
For example, with the definitions

```
fun {IsPos N} N>=0 end
fun {Inc N} N+1 end
fun {Dec N} N-1 end
```

the call `{SelectMap [~1 1 ~2 2] IsPos Inc Dec}` returns `[~2 2 ~3 3]`.
As another example, with the definitions

```
fun {IsLess X} X.1<X.2 end
fun {Fst X} X.1 end
fun {Snd X} X.2 end
```

the call `{SelectMap [1#2 4#3 7#5] IsLess Fst Snd}` returns `[1 3 5]`.
The function must be tail-recursive and you are not allowed to use other functions.

Solution.

```
fun {SelectMap Xs F G H}
   case Xs
   of nil then nil
   [] X|Xr then
      {if {F X} then G else H end X}|{SelectMap Xr F G H}
   end
end
```

4 Runtime (40 points)

Consider the following function

```
fun {Choose Xs}
   case Xs
   of X|Y|Xr then
      if X<Y then {Choose X|Xr} else {Choose Y|Xr} end
   [] X|nil then X
   end
end
```

Tables for execution times and asymptotic complexity are at the end of the exam.

4.1 Kernel Syntax (5 points)

Transform `Choose` into kernel-syntax.

Solution.

```
proc {Choose Xs Z}
   case Xs of X|Y|Xr then B in
      B = (X<Y)
      if B then Zr in
         Zr=X|Xr {Choose Zr Z}
      else Zr in
         Zr=Y|Xr {Choose Zr Z}
      end
   [] X|nil then Z=X
   end
end
```

4.2 Input Argument and Size Function (5 points)

Give the input argument (that is, the I function) of `{Choose Xs Y}` and an appropriate size function.

Solution. Input argument is `Xs`, size function is length of list.

4.3 Recurrence Equation (15 points)

Give a recurrence equation for the runtime $T(n)$ of `Choose`.

Solution. $T(n) = c + T(n-1)$

4.4 Asymptotic Complexity (5 points)

Give the asymptotic complexity of `Choose`.

Solution. $O(n)$.

4.5 The Opposite of `Choose` (10 points)

`{Choose Xs}` returns the smallest element of the list `Xs`. Give a similar definition `Opposite` to choose that returns the largest element of a list.
The function must be tail-recursive and you are not allowed to use other functions.

Solution.

```
fun {Opposite Xs}
   case Xs
   of X|Y|Xr then
      if X>Y then {Opposite X|Xr} else {Opposite Y|Xr} end
   [] X|nil then X
   end
end
```

5 Runtime (45 points)

Consider the following function:

```
fun {Scan Xs}
   case Xs
   of nil  then nil
   [] _|Xr then {Append {Scan Xr} Xs}
   end
end
```

Tables for execution times and asymptotic complexity are at the end of the exam.

5.1 Examples (6 points)

What does `{Scan [a]}` return?

Solution. `[a]`
What does `{Scan [a b]}` return?

Solution. `[b a b]`
What does `{Scan [a b c]}` return?

Solution. `[c b c a b c]`

5.2 The First Element (4 points)

Assume a list `Xs` with elements `X1`, $\cdots$, `XN` in that order.
What does `{Scan Xs}.1` return?

Solution. `XN`

5.3 Kernel Syntax (5 points)

Transform `Scan` to kernel syntax.

Solution.

```
proc {Scan Xs Ys}
   case Xs
   of nil  then Ys=nil
   [] _|Xr then Zs in Zs={Scan Xr} {Append Zs Xs Ys}
   end
end
```

5.4 Input Argument and Size Function (5 points)

Give the input argument (that is, the I function) of `Scan` and an appropriate size function.

Solution. Input argument is `Xs`, size function is length of list `Xs`.

5.5 Recurrence Equation (20 points)

Give a recurrence equation for the runtime $T(n)$ of `Scan`.

Solution. $T(n) = c_1 + c_2 n + T(n-1)$

5.6 Asymptotic Complexity (5 points)

Give the asymptotic complexity of `Scan`.

Solution. $O(n^2)$

6 Demand-driven Execution (20 points)

6.1 Generating Numbers (6 points)

Write a lazy function `{AltScale N}` that lazily computes the stream of numbers

`~N|3*N|~9*N|27*N|~81*N|`$\cdots$

Solution.

```
fun lazy {AltScale N}
   ~N|{AltScale ~3*N}
end
```

6.2 Lazy Filter (10 points)

Give a lazy `{LazyFilter Xs F}` function, where `Xs` is a list and `F` a unary function. Remember: `{LazyFilter Xs F}` returns a list which contains all elements of the list `Xs` for which `F` returns **`true`**.

Solution.

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

6.3 Lazy Testing (4 points)

Consider the following two definitions of test functions:

```
fun {EagerTest X} X==0 end
fun lazy {LazyTest X} X==0 end
```

Is the result of

```
{LazyFilter [~1 0 1] EagerTest}.1
```

determined?

Solution. Yes!
Is the result of

```
{LazyFilter [~1 0 1] LazyTest}.1
```

determined?

Solution. Yes!

7 Abstract Datatypes (40 points)

In the following you are going to use and implement the abstract data type *ordered collection*, which is abbreviated as *ocoll* in the following.
Ocolls can contain items that can be ordered with respect to the builtin comparison operator `<` (that is, numbers and atoms). The same item can also be contained multiple times in the same ocoll.
The interface for ocolls is defined by the following functions:

- `{NewOcoll}` returns a new ocoll.
- `{IsEmpty O}` tests whether the ocoll `O` contains an element.
- `{Smallest O}` returns the smallest element of the ocoll `O`.
- `{Insert O X}` returns an ocoll that includes `X`. If `X` is already included n-times in `O`, it must be contained $(n+1)$-times in the returned ocoll.
- `{Remove O}` returns an ocoll where the smallest element has been removed from `O`.

For example, after constructing an ocoll `O` by

```
O={Insert {Insert {Insert {NewOcoll} 1} 2} 1}
```

we have that `{Smallest O}` returns `1`, `{Smallest {Remove O}}` returns `1`, and `{Smallest {Remove {Remove O}}}` returns `2`.
Hint: the tasks start on the next page!

7.1 Inserting List Elements (8 points)

Write a function `{InsertAll Xs O}` that returns an ocoll according to the above definition with all elements of the list `Xs` added to those already in the ocoll `O`. The function must be tail-recursive.
You are only allowed to use the functions defined by the interface above.

Solution.

```
fun {InsertAll Xs S}
   case Xs
   of nil then S
   [] X|Xr then {InsertAll Xr {Insert S X}}
   end
end
```

7.2 Sorting to a List (8 points)

Write a function `{SortAll O}` that returns a list of all elements of the ocoll `O` in increasing order with respect to `<`. The function must be tail-recursive.
For example, for the ocoll `O` computed by

```
O={Insert {Insert {Insert {NewOcoll} a} b} a}
```

it holds that `{SortAll O}=[a a b]`.
You are only allowed to use the functions defined by the interface above.

Solution.

```
fun {SortAll O}
   if {IsEmpty O} then nil
   else {Smallest O}|{SortAll {Remove O}}
   end
end
```

7.3 Sorting Lists (4 points)

Implement a function `{Sort Xs}` that returns a list with the elements of the list `Xs` sorted in increasing order.
You must implement `Sort` with the functions `InsertAll` and `SortAll` from above.

Solution.

```
fun {Sort Xs}
   {SortAll {InsertAll Xs {NewOcoll}}}
end
```

7.4 Implementing Ocolls (20 points)

Give an implementation of the *ordered collection* abstract data type that uses lists. All functions must be tail-recursive and no helper functions are allowed.

Solution.

```
fun {NewOcoll}
   nil
end
fun {IsEmpty O}
   O==nil
end
fun {Smallest O}
```

```
   O.1
end
fun {Remove O}
   O.2
end
fun {Insert O X}
   case O
   of nil then [X]
   [] Y|Yr then
      if X<Y then X|O else Y|{Insert Yr X} end
   end
end
```

8 Agents with State: Message Repeater (40 points)

This assignment develops a repeat agent managing repeated sending of messages to another agent. The agent allows to change the receiving agent, the message, as well as the repeat time (the time between sending two messages).
A repeat agent understands the following messages:

- `agent(A)` Change the receiving agent to `A`.
- `message(M)` Change the message to be sent to `M`.
- `repeat(T)` Change the repeat time to the integer `T` (in milliseconds).
- `tick` The agent sends the current message `M` to the current agent `A` after the current repeat time `T`. After that it sends itself another `tick` message.

  The agent is not allowed to block during the time period `T`, it must immediately be able to process incoming messages.

The state of a repeat agent `R` must contain the receiving agent `A`, the message `M`, the repeat time `T`, as well as the repeat agent `R` itself. The repeat agent `R` is needed so that it can send itself a `tick` message.
The function `NewAgent` creating agents is as follows:

```
fun {NewAgent Process InitState}
   Port Stream
   proc {Execute Stream State}
      case Stream of Message|Rest then
         {Execute Rest {Process State Message}}
      end
   end
```

```
in
   Port={NewPort Stream}
   thread {Execute Stream InitState} end
   Port
end
```

8.1 Agent Processing Function (20 points)

Implement a function `{Process S M}` which takes a state `S` (containing an agent, a message, an integer for the repeat-time, and the repeat agent itself), a message `M`, and returns a new state. The function must implement processing of the messages as described above.
Use the procedure `{Delay T}` that suspends the current thread for `T` milliseconds

Solution.

```
fun {Process S M}
   case M
   of agent(A) then {AdjoinAt S agent A}
   [] message(M) then {AdjoinAt S message M}
   [] time(T) then {AdjoinAt S time T}
   [] tick then
      thread
         {Delay S.time}
         {Send S.agent S.message}
         {Send S.myself tick}
      end
      S
   end
end
```

8.2 Agent Creation (20 points)

Implement a function `{NewRepeater A M T}` that:

- Creates a new repeat agent `R` with the help of `NewAgent` and the `Process` function from above with an initial state containing `A`, `M`, `T`, and `R`.
- Initializes the agent such that it immediately starts sending messages.
- Returns the created agent.

Solution.

```
fun {NewRepeater A M T}
   G={NewAgent Process
               state(agent:A message:M time:T myself:G)}
in
   {Send G tick}
   G
end
```

9 Increment Functions (25 points)

In the following you are going to develop an increment function which returns `1` when called the first time, `2` the second time, and so on.

9.1 Increment Agent (10 points)

Give the processing function `{Process S M}` of an increment agent. The increment agent should maintain the current integer value in its state and understand an increment message `inc(N)` to increment its current value and return the old value in `N`.

Solution.

```
fun {Process S M}
   case M
   of inc(N) then N=S S+1
   end
end
```

9.2 Increment Function Creation (15 points)

Implement a function `{NewInc}` which returns a new increment function.
For example, the function obtained by

```
F={NewInc}
```

works as follows. Calling `F` by `{F}` the first returns `1`, the second time `2`, and so on.
Make sure that all functions created by `NewInc` are independent from each other!
Use the `Process` function from above and the `NewAgent` function from the previous assignment.

Solution.

```
fun {NewInc}
   IncAgent = {NewAgent Process 1}
in
   fun {$}
      N
   in
      {Send IncAgent inc(N)}
      N
   end
end
```

A Execution Times for Statements

Statement	Execution Time
skip	0
⟨x⟩=⟨y⟩	c
⟨x⟩=⟨v⟩	c
⟨s⟩$_1$ ⟨2⟩$_2$	$T(\langle s\rangle_1) + T(\langle s\rangle_2)$
local ⟨x⟩ **in** ⟨s⟩ **end**	$c + T(\langle s\rangle)$
if ⟨x⟩ **then** ⟨s⟩$_1$ **else** ⟨s⟩$_2$ **end**	$c + \max(T(\langle s\rangle_1), T(\langle s\rangle_2))$
case ⟨x⟩ **of** ⟨p⟩ **then** ⟨s⟩$_1$ **else** ⟨s⟩$_2$ **end**	$c + \max(T(\langle s\rangle_1), T(\langle s\rangle_2))$
{⟨x⟩ ⟨y⟩$_1$ ⋯ ⟨y⟩$_n$}	$T_{\langle x\rangle}(\text{size}(I(\{\langle y\rangle_1), \ldots, \langle y\rangle_n\})))$

B Asymptotic Complexity for Recurrence Equations

Recurrence Equation	Asymptotic Complexity
$T(n) = c + T(n-1)$	$O(n)$
$T(n) = c_1 + c_2 n + T(n-1)$	$O(n^2)$
$T(n) = c + T(n/2)$	$O(\log n)$
$T(n) = c_1 + c_2 n + T(n/2)$	$O(n)$
$T(n) = c + 2T(n/2)$	$O(n)$
$T(n) = c_1 + c_2 n + 2T(n/2)$	$O(n \log n)$