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 $n$:

- $n < 150$ fail (U)
- $150 \leq n < 200$ grade 3
- $200 \leq n < 250$ grade 4
- $250 \leq n$ grade 5

Points

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

<table>
<thead>
<tr>
<th>Task</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>Σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>45</td>
<td>25</td>
<td>20</td>
<td>40</td>
<td>45</td>
<td>20</td>
<td>40</td>
<td>40</td>
<td>25</td>
<td>300</td>
</tr>
<tr>
<td>Points</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bonuspoints:

Totalpoints:

Grade:
1 Questions (45 points)

1. Are environments shared among all statements of a single thread (3 points)?

Solution. No. Threads (semantic stacks) consist of semantic statements being pairs of statements and environments.

2. Is the case-statement suspendable (3 points)?

Solution. Yes.

3. Consider the environments

\[ E_1 = \{ X \mapsto x_1, Z \mapsto x_2 \} \]
\[ E_2 = \{ X \mapsto x_3, Y \mapsto x_4, Z \mapsto x_5 \} \]

Give the environment \( E_1 + E_2 \) (3 points):

Solution.

\( \{ X \mapsto x_3, Y \mapsto x_4, Z \mapsto x_5 \} \)

Give the environment \( E_2 + E_1 \) (3 points):

Solution.

\( \{ X \mapsto x_1, Y \mapsto x_4, Z \mapsto x_2 \} \)

4. Give the activation condition of the semantic statement

\[(\langle x \rangle \langle y \rangle_1 \cdots \langle y \rangle_n, E)\]

(3 points).

Solution. \( E(\langle x \rangle) \) is determined.

5. Do infinite computations (computations that never terminate such as an agent that receives an infinite number of messages) always require infinite stack space (3 points)?
6. Assume a recursive procedure \( P \) (it is directly recursive, that is \( P \) calls \( P \)). Give one variable identifier that must be an external reference of the definition of \( P \) (3 points).

**Solution.** \( P \) itself.

7. Has each semantic stack in the abstract machine a private store (3 points)?

**Solution.** No. Semantic stacks (threads) actually communicate by using the single shared store.

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

\[
\text{local } X \text{ in } \\
\quad \text{local } Z \text{ in skip end} \\
\quad \text{local } Y \text{ in } \\
\quad \quad \text{local } Y \text{ in } \langle s \rangle \end{end} \\
\end{end}
\]

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

**Solution.** \( X, Y \).

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

\[
P = \text{proc } \{ \$ X Y \} \\
\quad \text{local } Z \text{ in } \{ X \{ U Z \} Y \} \text{ end} \\
\end{end}
\]

**Solution.** \( U \)

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

\[
P = \text{proc } \{ \$ X Y \} \\
\quad \text{local } Y \text{ in } \{ P \{ X Z \} Y \} \text{ end} \\
\end{end}
\]
Solution.  $P, Z$

11. Consider the following statement $(s)$:

$P = \text{proc } \{$ $X Y$ \}$

\begin{verbatim}
   \text{if } X==Y \text{ then } Z=a \text{ else } Z=P \text{ end}
\end{verbatim}

end

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

Solution.  $\{Z \mapsto z, P \mapsto p\}$

12. Give the value of $U$ after execution of the following statement (3 points):

local $X$ $Y$ $Z$ $U$ in
thread if $X==1$ 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 unbound.

13. Give the value of $Z$ after execution of the following statement (6 points):

local $C$ $X$ $Y$ $Z$ in
fun $\{C A B\}$
fun $\{$ $F$ $\}$
\begin{verbatim}
    \{F A B\}
end
end
fun $\{X A B\}$
$A$
end
fun $\{Y A B\}$
$B$
end
$Z=\{\{C \{C a b\} \{C c d\}\} Y\} X\}$
end

Solution.  $c$
2 Accumulators (25 points)

The following function \{Longest Xs\} takes a list of lists Xs as input:

\[
\text{fun } \{\text{Longest } Xs\} \\
\text{case } Xs \\
\text{of } \text{nil } \text{then } 0 \\
[\_] X | Xr \text{ then} \\
\quad N=\{\text{Length } X\} \quad M=\{\text{Longest } Xr\} \\
\quad \text{in} \\
\quad \quad \text{if } N>M \text{ then } N \text{ else } M \text{ end} \\
\quad \text{end} \\
\text{end}
\]

2.1 Examples (5 points)

What does \{Longest nil\} return?

Solution. 0

What does \{Longest [[a] [b] [b c]]\} return?

Solution. 2

2.2 Tail-Recursion (20 points)

Give an equivalent tail-recursive function \{LongestAcc Xs N\} that uses N as accumulator.

Solution.

\[
\text{fun } \{\text{LongestAcc } Xs \ N\} \\
\text{case } Xs \\
\text{of } \text{nil then } N \\
[\_] X | Xr \text{ then} \\
\quad M=\{\text{Length } X\} \\
\quad NM=\text{if } N>M \text{ then } N \text{ else } M \text{ end} \\
\quad \text{in} \\
\quad \quad \{\text{LongestAcc } Xr \ NM\} \\
\quad \text{end} \\
\text{end}
\]
3 Higher-Order Programming (20 points)

Write a function \( \{\text{SwitchMap} \ Xs \ F \ G\} \) that takes a list \( Xs \) and two unary functions \( F \) and \( G \) as input. It returns a list where the elements at odd positions are obtained by applying \( F \) to the element at the same position in \( Xs \) and the elements at even positions are obtained by applying \( G \) to the element at the same position.

For example, with the definitions

\[
\begin{align*}
\text{fun } \{\text{Inc} \ N\} & \ N+1 \ \text{end} \\
\text{fun } \{\text{Dec} \ N\} & \ N-1 \ \text{end}
\end{align*}
\]

the call \( \{\text{SwitchMap} \ [1 \ 1 \ 2 \ 2] \ \text{Inc} \ \text{Dec}\} \) returns \([2 \ 0 \ 3 \ 1]\).

The function must be tail-recursive and you are not allowed to use other functions.

Solution.

\[
\begin{align*}
\text{fun } \{\text{SwitchMap} \ Xs \ F \ G\} \\
\text{case } Xs \text{ of nil then nil} \\
\text{[] } X|Xr \text{ then } \{F \ X\}|\{\text{SwitchMap} \ Xr \ G \ F\} \\
\text{end}
\end{align*}
\]

4 Runtime (40 points)

Consider the following function

\[
\begin{align*}
\text{fun } \{\text{Last} \ Xs\} \\
\text{case } Xs \text{ of } X|Xr \text{ then} \\
\text{if } Xr==\text{nil} \text{ then } X \text{ else } \{\text{Last} \ Xr\} \text{ end} \\
\text{end}
\end{align*}
\]

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

4.1 Kernel Syntax (5 points)

Transform \( \text{Last} \) into kernel-syntax.

Solution.

\[
\begin{align*}
\text{proc } \{\text{Last} \ Xs \ Y\} \\
\text{case } Xs \text{ of } X|Xr \text{ then } B \text{ in} \\
\text{B } = (Xr==\text{nil}) \\
\text{if } B \text{ then } Y=X \\
\text{else } \{\text{Last} \ Xr \ Y\}
\end{align*}
\]
4.2 Input Argument and Size Function (5 points)

Give the input argument (that is, the \( I \) function) of \( \text{Last } Xs \ Y \) 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 \( \text{Last} \).

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

4.4 Asymptotic Complexity (5 points)

Give the asymptotic complexity of \( \text{Last} \).

Solution. \( O(n) \).

4.5 \( \text{Last} \) With a Single \text{case}-Statement (10 points)

Give an equivalent definition of \( \text{Last} \) that only has a single \text{case}-statement but no \text{if}-statement. You can assume that \( \text{Last} \) is only called with lists that have at least one element.

Solution.

\[
\begin{align*}
\text{fun } \{\text{Last } Xs\} \\
\text{  case } Xs \\
\text{    of } [X] \text{ then } X \\
\text{    [] } \_ \text{ Xr then } \{\text{Last } Xr\} \\
\text{  end} \\
\text{end}
\end{align*}
\]
5  Runtime (45 points)

Consider the following function:

```plaintext
fun {Pile Xs}
  case Xs
  of nil then nil
      [] X|Xr then {Append Xs {Pile Xr}}
  end
end
```

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

5.1  Examples (6 points)

What does \{Pile [a]\} return?

Solution.  [a]

What does \{Pile [a b]\} return?

Solution.  [a b b]

What does \{Pile [a b c]\} return?

Solution.  [a b c b c c]

5.2  Kernel Syntax (5 points)

Transform \texttt{Pile} to kernel syntax.

Solution.

```plaintext
proc {Pile Xs Ys}
  case Xs
  of nil then nil
      [] X|Xr then Yr in
        {Pile Xr Yr}
        {Append Xs Yr Ys}
  end
end
```

5.3  Input Argument and Size Function (5 points)

Give the input argument (that is, the \textit{I} function) of \texttt{Pile} and an appropriate size function.
Solution. Input argument is $x_s$, size function is length of list $x_s$.

### 5.4 Recurrence Equation (20 points)

Give a recurrence equation for the runtime $T(n)$ of $\text{Pile}$.

Solution. $T(n) = c_1 + c_2n + T(n-1)$

### 5.5 Asymptotic Complexity (5 points)

Give the asymptotic complexity of $\text{Pile}$.

Solution. $O(n^2)$

### 5.6 The Last Element (4 points)

Give an expression that computes $\{\text{Last } \{\text{Pile } x_s\}\}$ in linear time. For the definition of $\text{Last}$ see Question 4.

Solution.

$\{\text{Last } x_s\}$

### 6 Demand-driven Execution (20 points)

#### 6.1 Generating Numbers (6 points)

Write a lazy function $\{\text{Double } N\}$ that lazily computes the stream of numbers $\text{N}\ |	ext{2*N}\ |	ext{4*N}\ |	ext{8*N}\ |\cdots$

Solution.

```
fun lazy {Double N}
  N|{Double 2*N}
end
```

#### 6.2 Lazy Map (10 points)

Give a lazy $\{\text{LazyMap } x_s F\}$ function, where $x_s$ is a list and $F$ a unary function.
Solution.

\[
\text{fun lazy } \{ \text{LazyMap } Xs \ F \} \\
\quad \text{case } Xs \text{ of nil then nil} \\
\quad \qquad [] X|Xr \text{ then } \{ F \ X \} \{ \text{LazyMap } Xr \ F \} \end{case} \\
\end{fun}
\]

### 6.3 Scaling (4 points)

Consider the following two definitions which multiply lists of numbers \( Ns \) with a number \( M \):

\[
\text{fun } \{ \text{EagerScale } Ns \ M \} \\
\quad \{ \text{LazyMap } Ns \ \text{fun} \ \$ \ N \ N*M \ \text{end} \end{fun}
\]

\[
\text{fun lazy } \{ \text{LazyScale } Ns \ M \} \\
\quad \{ \text{LazyMap } Ns \ \text{fun} \ \$ \ N \ N*M \ \text{end} \end{fun}
\]

Both return a list of numbers.

How many list elements are computed by

\( \{ \text{EagerScale } [1 \ 2 \ 3] \ 3 \} \)

**Solution.** 0

How many list elements are computed by

\( \{ \text{LazyScale } [1 \ 2 \ 3] \ 3 \} \)

**Solution.** 0

### 7 Abstract Datatypes (40 points)

In the following we are going to use and implement an abstract data type \textit{stack}. The interface for stacks is defined by the following functions:

- \( \{ \text{NewStack} \} \) returns an empty stack.
- \( \{ \text{IsEmpty } S \} \) tests whether the stack is empty.
- \( \{ \text{Top } S \} \) returns the topmost element of stack \( S \).
- \( \{ \text{Push } S \ X \} \) returns a stack with \( X \) pushed on top of stack \( S \).
- \( \{ \text{Pop } S \} \) returns a stack where the top-element of stack \( S \) has been popped.
7.1 Pushing List Elements (10 points)

Write a function \( \{\text{PushAll Xs S}\} \) that returns a stack according to the above definition with all elements of the list \( Xs \) pushed on the stack \( S \). The first element of \( Xs \) must be pushed first. The function must be tail-recursive.
For example, for the stack \( S \) computed by
\[
S = \{\text{PushAll} \ [a \ b] \ \{\text{NewStack}\}\}
\]
it holds that \( \{\text{Top} \ S\} = \text{b} \) and \( \{\text{Top} \ \{\text{Pop} \ S\}\} = \text{a} \).
You are only allowed to use the functions defined by the interface above.

Solution.

\[
\text{fun} \ \{\text{PushAll} \ Xs \ S\} \\
\quad \text{case} \ Xs \\
\quad \quad \text{of} \ \text{nil} \ \text{then} \ S \\
\quad \quad \ [\ ] \ X|Xr \ \text{then} \ \{\text{PushAll} \ Xr \ \{\text{Push} \ S \ X\}\}
\end{array}
\end{array}
end
end
\]

7.2 Popping to a List (10 points)

Write a function \( \{\text{PopAll} \ S\} \) that returns a list of all elements of the stack \( S \) as obtained by popping them from the stack (in that order). The function must be tail-recursive.
For example, for the stack \( S \) computed by
\[
S = \{\text{Push} \ \{\text{Push} \ \{\text{NewStack}\} \ a\ b\}\}
\]
it holds that \( \{\text{PopAll} \ S\} = [b \ a] \).
You are only allowed to use the functions defined by the interface above.

Solution.

\[
\text{fun} \ \{\text{PopAll} \ S\} \\
\quad \text{if} \ \{\text{IsEmpty} \ S\} \ \text{then} \ \text{nil} \\
\quad \text{else} \ \{\text{Top} \ S\} | \{\text{PopAll} \ \{\text{Pop} \ S\}\}
\end{array}
\end{array}
end
end
\]

7.3 Reversing Lists (5 points)

Implement a function \( \{\text{Reverse} \ Xs\} \) that returns a list with the elements of the list \( Xs \) in reverse order.
You must implement \text{Reverse} with the functions \text{PushAll} and \text{PopAll} from above.
Solution.

fun {Reverse Xs}
    {PopAll {PushAll Xs {NewStack}}}
end

7.4 Implementing Stacks (15 points)

Give an implementation of the stack abstract data type that uses lists.

Solution.

fun {NewStack}
    nil
end

fun {IsEmpty S}
    S==nil
end

fun {Top S}
    S.1
end

fun {Pop S}
    S.2
end

fun {Push S X}
    X|S
end

8 Agents with State: Group Multicast (40 points)

This assignment develops an agent that manages multicast groups. The agent understands the following messages:

- subscribe(A) The agent A is subscribed to the multicast group.
- unsubscribe(A) The agent A is unsubscribed from the multicast group.
- multicast(M) The message M is send to all members of the multicast group.

The state of the agent consists of a multicast group. A multicast group is implemented as a list of agents. Initially, the agent starts with an empty multicast group.

The function NewAgent to create 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  
end  
in  
  Port={NewPort Stream}  
thread {Execute Stream InitState} end Port  
end

The function FoldL for left folding lists is as follows:

fun {FoldL Xs F S}  
  case Xs of  
    nil then S  
    [] X Xr then {FoldL Xr F {F S X}}  
  end  
end

8.1 NewAgent with FoldL (4 points)

Give a function NewAgent that is equivalent to the above definition but where Execute is implemented with FoldL.

Solution.

fun {NewAgent Process InitState}  
  Port Stream  
in  
  Port={NewPort Stream}  
  thread _={FoldL Stream Process InitState} end Port  
end

8.2 Subscribing (8 points)

Implement a function {Subscribe G A} which returns a multicast group where the agent A is added to the multicast group G only if A is not yet a member of G. As agents are represented by ports, testing whether two agents are the same can be done by the normal equality test. For example, if A1 and A2 are two agents, then A1==A2 returns true, if and only if A1 and A2 are the same agent. You are not allowed to use the built-in Member function.
Name: Personnummer: 

Solution.

fun {Subscribe G A}
  case G
  of nil then [A]
  [] A1|R then
    if A1==A then G else A1|{Subscribe R A} end
  end
end

8.3 Unsubscribing (8 points)

Implement a function {UnSubscribe G A} which returns a multicast group where the agent A is removed from the multicast group G. You are not allowed to use the built-in Remove function.

Solution.

fun {UnSubscribe G A}
  case G
  of nil then nil
  [] A1|R then
    if A1==A then R else A1|{UnSubscribe R A} end
  end
end

8.4 Multicasting (8 points)

Implement a procedure {MultiCast G M} where G is a multicast group and M is a message. The procedure sends M to all agents contained in G. You are allowed to use ForAll.

Solution.

proc {MultiCast G M}
  {ForAll G proc {$ A}
    {Send A M}
    end}
end

Without ForAll:

proc {Multicast G M}
  case G
  of nil then skip
  [] A1|R then {Send A M} {MultiCast R M}
8.5 Agent Processing Function (8 points)

Implement a function \{Process S M\} which takes a state S (a multicast group), a message M, and returns a new state. The function must implement processing of the messages as described above.

Solution.

\[
\text{fun } \{\text{MCP S M}\} \text{ case M of } \{\text{Subscribe S A}\} \text{ then } \{\text{Subscribe S A}\} \text{ end } \{\text{UnSubscribe S A}\} \text{ end } \{\text{MultiCast S M}\} \text{ S end end}
\]

8.6 Agent Creation (4 points)

Define a function \{NewGroupAgent\} that returns an agent for multicast groups. The agent should be created with an empty multicast group.

Solution.

\[
\text{fun } \{\text{NewGroupAgent}\} \text{ end}
\]

9 Checking Runtime (25 points)

9.1 Result Sending (10 points)

Implement a procedure \{RunAndSend F P\} that executes the nullary function F in a newly created thread and sends its result to the port P after execution of F has terminated.
Solution.

\[\text{proc } \{\text{RunAndNotify } F \ P\}\]
\[\text{thread } R=F \text{ in } \{\text{Send } P \ R\} \text{ end}\]
\[\text{end}\]

9.2 Checking (10 points)

Implement a function \{Check P T\} which takes a nullary procedure P and a time value T in milliseconds. It returns \text{true}, if execution of the procedure P in a thread of its own has finished in less than or equal to T milliseconds. Otherwise, it returns \text{false}.

**Hint:** Use a port where the first message that arrives determines the function’s result. The procedure \{Delay T\} suspends the executing thread for T milliseconds.

Solution.

\[\text{fun } \{\text{Check P T}\}\]
\[\text{Po Stream}\]
\[\text{in}\]
\[\text{Po=\{NewPort Stream\}}\]
\[\{\text{RunAndNotify } \text{fun } \{\$\} \{P\} \text{ true end } \text{ Po}\}\]
\[\{\text{RunAndNotify } \text{fun } \{\$\} \{\text{Delay T} \text{ false end } \text{ Po}\}\]
\[\text{Stream.1}\]
\[\text{end}\]

9.3 Cheating (5 points)

Cheat the Check function from above! Give a nullary procedure Cheat such that \{Check Cheat 1000\} returns \text{true} even though there are still statements from Cheat being executed.

Solution.

\[\text{local}\]
\[\text{proc } \{\text{Spin}\} \{\text{Spin}\} \text{ end}\]
\[\text{in}\]
\[\text{proc } \{\text{Cheat}\} \text{ thread } \{\text{Spin}\} \text{ end end}\]
\[\text{end}\]
A  Execution Times for Statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>skip</td>
<td>0</td>
</tr>
<tr>
<td>( (x) = (y) )</td>
<td>( c )</td>
</tr>
<tr>
<td>( (x) = (v) )</td>
<td>( c )</td>
</tr>
<tr>
<td>( \langle s \rangle_1 \langle 2 \rangle_2 )</td>
<td>( T(\langle s \rangle_1) + T(\langle s \rangle_2) )</td>
</tr>
<tr>
<td>local ( (x) ) in ( \langle s \rangle ) end</td>
<td>( c + T(\langle s \rangle) )</td>
</tr>
<tr>
<td>if ( (x) ) then ( \langle s \rangle_1 ) else ( \langle s \rangle_2 ) end</td>
<td>( c + \max { T(\langle s \rangle_1), T(\langle s \rangle_2) } )</td>
</tr>
<tr>
<td>case ( (x) ) of ( (p) ) then ( \langle s \rangle_1 ) else ( \langle s \rangle_2 ) end</td>
<td>( c + \max { T(\langle s \rangle_1), T(\langle s \rangle_2) } )</td>
</tr>
<tr>
<td>( \langle x \rangle \langle y \rangle_1 \cdots \langle y \rangle_n )</td>
<td>( T(\langle x \rangle)(\text{size}(\langle {y}_1, \ldots, {y}_n})) )</td>
</tr>
</tbody>
</table>

B  Asymptotic Complexity for Recurrence Equations

<table>
<thead>
<tr>
<th>Recurrence Equation</th>
<th>Asymptotic Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(n) = c + T(n - 1) )</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>( T(n) = c_1 + c_2 n + T(n - 1) )</td>
<td>( O(n^2) )</td>
</tr>
<tr>
<td>( T(n) = c + T(n/2) )</td>
<td>( O(\log n) )</td>
</tr>
<tr>
<td>( T(n) = c_1 + c_2 n + T(n/2) )</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>( T(n) = c + 2T(n/2) )</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>( T(n) = c_1 + c_2 n + 2T(n/2) )</td>
<td>( O(n \log n) )</td>
</tr>
</tbody>
</table>