MIDTERM EXAM
TDT4165 PROGRAMMING LANGUAGES

Materials: D (No printed or handwritten materials are allowed. An approved pocket calculator is allowed.)

22. October 2009

Some of the tasks are multiple-choice, while others require you to write a **short** answer. For each multiple-choice question there is only one correct alternative, and it will give you 1 (one) point. Every wrong answer will contribute 0 (zero) points. If you mark more than one alternative, you will receive 0 (zero) points for that task. For the questions that require a written answer, the maximum number of points will be stated in each task.

Task 1

```
fun {F N1 N2}
    if N1==N2 then N2
    else
      N1+{F N1+1 N2}
    end
end
```

Given the code above. What does `{F 1 4}` return?

a) 7

b) 8

c) 9

d) 10

Answer: d)

Task 2

A tokenizer is a program that...

a) reads a sequence of characters and outputs a sequence of tokens.

b) translates a sequence of characters into a sequence of low-level instructions that can be executed on a machine.

c) reads a sequence of tokens and outputs an abstract syntax tree.

d) traverses the syntax tree and generates low-level instructions for a real machine or an abstract machine.

Answer: a)

Task 3

A parser is a program that...

a) reads a sequence of characters and outputs a sequence of tokens.

b) translates a sequence of characters into a sequence of low-level instructions that can be executed on a machine.

c) reads a sequence of tokens and outputs an abstract syntax tree.

d) traverses the syntax tree and generates low-level instructions for a real machine or an abstract machine

Answer: c)

Task 4

1) `List=[1 2 3]`

2) `List= 1|2|3|nil`

3) `List= '|'(1 '|'(2 '|'(3 nil)))`

Which of the lists above represent the same data structure?

a) None

b) 1 and 2

c) 1 and 3

d) All

Answer: d)

Task 5

Function 1:

```
fun {Function1 A}
    case A of nil then
       0
    [] _|T then
       1+{Function1 T}
    end
end
```

Function 2:

```
fun {Function2 A B}
    if B<1 then
       nil
    else
       case A of nil then
            nil
       [] H|T then
            H|{Function2 T B-1}
       end
    end
end
```

Function 3:

```
fun {Function3 A B}
    if B<1 then
       A
    else
       case A of nil then
            nil
       [] _|T then
            {Function3 T B-1}
       end
    end
end
```

Function 4:

```
fun {Function4 A B}
    case A of nil then
        B
    [] H|T then
        H|{Function4 T B}
    end
end
```

{`Drop Xs N`} returns Xs without the first N elements. If N is bigger than the number of elements in Xs will Xs be returned.
Which of the functions above will do the same thing as {`Drop Xs N`}?

a) `Function1`

b) `Function2`

c) `Function3`

d) `Function4`

Answer: c)

Task 6

What will {`Function2 [1 2 3 4] 2`} return?

a) 3

b) [3 4]

c) nil

d) [1 2]

Answer: d)

Task 7

```
proc {A B C}
     {D B E}
end
```

What external references do we have above?

a) A and D

b) B and C

c) D and E

d) B and E

Answer: c)

Task 8 (max 1 point)

```
local X in
     X=3
     X=2
end
```

Describe what will happen if the code above is run in Oz.

Answer: Unification failure on X=2 (The unification attempt 3=2 fails)

Task 9

```
        declare
        fun lazy {MakeOnes} 1.0|{MakeOnes} end
        fun lazy {StreamMap S F}
            case S of Sh|St then
                 {F Sh}|{StreamMap St F}
            [] nil then
                 nil
            end
        end
        A={StreamMap {MakeOnes} fun {$ X} X+1.0 end}
        B={StreamMap A fun {$ X} X+1.0 end}
        C={StreamMap B fun {$ X} X+1.0 end}
{Browse A}
{Browse B}
{Browse C}
```

What is printed?

a) `2.0|2.0|2.0|2.0|,,,`
   B
   C

b) `2.0|2.0|2.0|2.0|,,,`
   `3.0|3.0|3.0|3.0|,,,`
   C

c) `2.0|2.0|2.0|2.0|,,,`
   `3.0|3.0|3.0|3.0|,,,`
   `4.0|4.0|4.0|4.0|,,,`

d) A
   B
   C

Answer: d)

Task 10

What does the higher-order programming concept of "genericity" mean?

a) The ability to convert any statement into a procedure value

b) The ability to pass procedure values as arguments to a procedure call

c) The ability to return procedure values as results from a procedure call

d) The ability to put procedure values in data structures

Answer: b)

Task 11 (max 3 points)

`{StreamMap S F}` will use `F` on every element of the stream `S` and return this as a new stream.

```
{StreamMap 1|2|3|4|_ MultiplyWithFive}
```

will return `5|10|15|20|_`

Write this function in Oz.

Answer:

```
fun lazy {StreamMap S F}
   case S of H|T then
      {F H}|{StreamMap T F}
   [] nil then
      nil
   end
end
```

Since the text does not specify whether the streams should be eager or lazy, the `lazy` keyword is optional here.

Task 12

Which one of these grammars can produce the string 1010?

a)
```
<number> ::= <digit> <number>
<digit> ::= 0|1
```

b)
```
<number> ::= <number> | <digit> | ε
<digit> ::= 0|1
```

c)
```
<number> ::= <digit> | <digit><number>| ε
<digit> ::= 0|1
```

d)
```
<number> ::= <number> <digit>|<number>
<digit> ::= 0|1
```

where ε is the empty string.

Answer: c)

Task 13

Given the grammar

```
<expression>    ::= <integer>
                    |  <expression> <operator> <expression>
<operator>      ::= + | - | * | /
<integer>       ::= <nonzero digit> { <digit> }
<digit>         ::= 0 | <nonzero digit>
<nonzero digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

How many parse trees can be generated for `1+2*3` with this grammar?

a) 1

b) 2

c) 3

d) 4

Answer: b)

Task 14

What do the terms "linguistic abstraction" and "syntactic sugar" have in common?

a) Both are translation schemes from the full programming language into the kernel language

b) Both describes a function of the kernel language

c) Both extend the kernel language

d) Both are a shortcut notation for frequently occurring language idioms

Answer: a)

Task 15

What is syntactic sugar?

a) A way of improving program readability

b) A shortcut notation for frequently occurring language idioms

c) A translation scheme from the full programming language into the kernel language

d) All of the above

Answer: d)

Task 16

Which of the following is wrong?

a) A linguistic abstraction is a translation scheme from the full programming language into the kernel language

b) A linguistic abstraction is a language idiom

c) A linguistic abstraction is a way of defining new language extensions

d) A linguistic abstraction can be translated into the kernel language

Answer: Since a given linguistic abstraction only makes up part of a translation scheme, and no precise definition of "a language idiom" has been given, both a) and b) are accepted as correct answers.

Many students asked during the midterm for a definition of "language idiom". The book uses this term on several occasions, but does not actually define it. See `http://en.wikipedia.org/wiki/Programming_idiom` or `http://en.wiktionary.org/wiki/idiom#Noun` (alt 5., programming) for a definition.

Task 17 (max 1 point)

What is meant by the syntax of a programming language?

Answer: The syntax of a language (usually specified by a grammar) defines the set of legal programs in the language, ie. programs that can be successfully parsed.

Task 18 (max 1 point)

What is meant by the semantics of a programming language?

Answer: The semantics of a language defines what a legal program, written in the language, does when it executes (the program's *meaning*).

Task 19 (max 2 points)

What are the defining characteristics of a declarative program?
Answer:

a) variables are declarative (can only be assigned once)

b) supports unification of partial values

c) supports higher order programming

d) deterministic

e) no side effects

Needed to specify at least three characteristics to get full score here.

Task 20 (max 2 points)

What are the properties of the declarative computation model?
Answer:

a) The model is sequential

b) All computations are independent of any external state

c) All computations are stateless and deterministic

d) The model supports both pure functional programming pure deterministic logic programming

Needed to specify at least two central properties of the model to get full score here.

Task 21 (max 2 points)

```
fun {Sum Xs}
    case Xs of H|T then
       H+{Sum T}
    else
       0
    end
end
```

Rewrite this to be a tail recursive function.

Answer: Rewrite to use an accumulator function:

```
fun {Sum2 Xs}
   fun {SumAcc Ys Acc}
     case Ys of H|T then
        {SumAcc T Acc+H}
      else
        Acc
      end
   end
in
   {SumAcc Xs 0}
end
```

See Ch. 2.5.1 page 72 in the book for an explanation of what is meant by "tail recursion" or "tail recursive function".