CS4215 Programming Language Implementation

You have 45 minutes to complete the exam. Use a B2 pencil to fill up the provided MCQ form. Leave Section A blank. Fill up Sections B and C.

After finishing, place the MCQ sheet on top of the question sheet and leave both on the table, when you exit the room.

Question 1: Sometimes, it is a matter of perspective if a particular language processor is seen as a compiler or as an interpreter. Which of the following statements is incorrect?

1 [A] A web browser that displays HTML web pages can be seen as an interpreter of HTML code.

1 [B] A web browser that allows for printing of web pages to a file in PostScript format can be seen as a compiler from HTML to PostScript.

1 [C] A web browser that provides a "print screen" function, which saves the current display to a JPG file, can be seen as a compiler from HTML to JPG.

1 [D] A web browser that handles JavaScript programs that are embedded in HTML web pages can be seen as a compiler from JavaScript to HTML.

1 [E] A web browser that uses XSL style sheets to display XML files can be seen as an interpreter of both XSLT style sheets and XML files.

Answer 1:

1 [D] A web browser that handles JavaScript programs that are embedded in HTML web pages can be seen as an interpreter of JavaScript programs. There is no compilation going on here. All other alternatives make sense and highlight that it is often a matter of perspective, which view to take.

Question 2: Which of the following statements is correct?

2 [A] `fun x -> y end`$[\texttt{x} \leftarrow \texttt{y}]$`fun w -> w end`

2 [B] `fun x -> y end`$[\texttt{x} \leftarrow \texttt{y}]$`fun w -> x end`

2 [C] `fun x -> y end`$[\texttt{x} \leftarrow \texttt{y}]$`fun x -> x end`

2 [D] `fun x -> y end`$[\texttt{x} \leftarrow \texttt{y}]$`fun w -> y end`

2 [E] `fun x -> y end`$[\texttt{x} \leftarrow \texttt{y}]$`fun x -> y end`

Answer 2:

2 [E] There is no capturing here.

Question 3: The lectures introduce a specific way of translating `let` expressions in simPL to the application of a function definition. For a given simPL expression E, let us denote by $L(E)$ the result of eliminating all `let` expressions in E using this technique.
Consider an alternative translation $L'(E)$, defined as follows.

$$\texttt{let } x_1 = E_1 \texttt{ in } E \texttt{ end}$$

stands for

$$\texttt{(fun } x_1 \texttt{ -> } E \texttt{ end } E_1\texttt{)}$$

and for $n > 1$,

$$\texttt{let } x_1 = E_1 \;\; x_2 = E_2 \cdots \; x_n = E_n \texttt{ in } E \texttt{ end}$$

stands for

$$\texttt{let } x_1 = E_1 \texttt{ in let } x_2 = E_2 \cdots \; x_n = E_n \texttt{ in } E \texttt{ end}$$

Note that you may have to apply the transformation L' multiple times for a given `let` expression. Which of the following statements is correct?

3 [A] For every simPL expression E, if $L(E)$ has a free variable, then $L'(E)$ also has a free variable.

3 [B] For every simPL expression E, if $L'(E)$ has a free variable, then $L(E)$ also has a free variable.

3 [C] For every simPL expression E, if $L(E)$ evaluates to a value according to dynamic semantics, then $L'(E)$ also evaluates to a value.

3 [D] For every simPL expression E, if $L'(E)$ evaluates to a value according to dynamic semantics, then $L(E)$ also evaluates to a value.

3 [E] None of the above statements is true.

Answer 3:

3 [B] This new definition of `let` widens the scope of identifiers so that it applies to some other equations between `let` and `in`. Alternatives 3 and 4 are not correct, since 1 is not correct. Therefore, you can construct examples that behave differently in either way.

Question 4: Consider the static semantics of simPL as introduced in the lectures. How many type expressions T exist such that

```
fun {T} x y z -> (z x + y) & (x = y) end
```

is well-typed?

4 [A] 0

4 [B] 1

4 [C] 2

4 [D] 3

4 [E] infinitely many

Answer 4:

4 [B] The only type expression possible is

```
int * int * (int -> bool) -> bool
```

Question 5: Consider the static semantics of simPL as introduced in the lectures. How many type expressions T exist such that

```
fun {T} x y z -> (z (x y)) + (z y) + (z x) end
```

is well-typed?

5 [A] 0

5 [B] 1

5 [C] 2

5 [D] 3

5 [E] infinitely many

Answer 5:

5 [A] The types of `x` and `y` must be the same, since `z` is applied to both. However, `x` is

applied to `y`. That means that the argument type of `x` is the same as the type of `x`, which is clearly impossible.

Question 6: Consider the static semantics of simPL as introduced in the lectures. How many type expressions T exist such that

```
fun {T} x y z -> (z (y z)) end
```

is well-typed?

6 [A] 0

6 [B] 1

6 [C] 2

6 [D] 3

6 [E] infinitely many

Answer 6:

6 [E] Examples:

```
int * ((int->int)->int) * (int->int) -> int
bool * ((int->int)->int) * (int->int) -> int
(int -> int) * ((int->int)->int) * (int->int) -> int
(bool -> bool) * ((int->int)->int) * (int->int) -> int
```

Question 7: Consider the static semantics of simPL as introduced in the lectures. How many type expressions T exist such that

```
fun {T} x y z -> x + (y z & true) end
```

is well-typed?

7 [A] 0

7 [B] 1

7 [C] 2

7 [D] 3

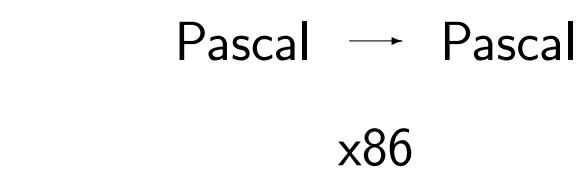
7 [E] infinitely many

Answer 7:

7 [B] The only valid type expression is

```
int * (bool -> int) * bool -> int
```

Question 8: Consider the following T-diagram of a compiler.



Which one of the following statements is true?

8 [A] This compiler is easy to implement, because it simply needs to output the same program that it receives as input.

8 [B] This compiler is impossible to implement, because you cannot translate a language to itself.

8 [C] This compiler must originally be implemented in Pascal. It was then compiled to x86 machine code (possibly using another language as intermediate step).

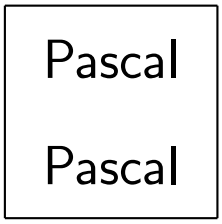
8 [D] This compiler cannot originally be implemented in Pascal; it was either implemented in a different high-level language, such as C++, or directly written in x86 machine code.

8 [E] This compiler cannot originally be implemented in a high-level language; it was directly written in x86 machine code.

Answer 8:

8 [A] When the source language is the same as the target language, the compiler just needs to churn out what you give it as input. Trivial! All other statements are nonsense; discussion please in Forum Midterm.

Question 9: Consider the following T-diagram of an interpreter.



Which one of the following statements is true?

9 [A] This interpreter is impossible to implement, because a language cannot interprete itself.

9 [B] This interpreter must originally be implemented in a different language, such as C++, and then translated to Pascal.

9 [C] This interpreter cannot be used to run programs, even if you use another Pascal interpreter.

9 [D] This interpreter is useless, in a sense that any T-diagram that uses this interpreter can be changed to a T-diagram that does not use it.

9 [E] This interpreter is easy to implement. It just needs to print out the program that it is receiving as parameter.

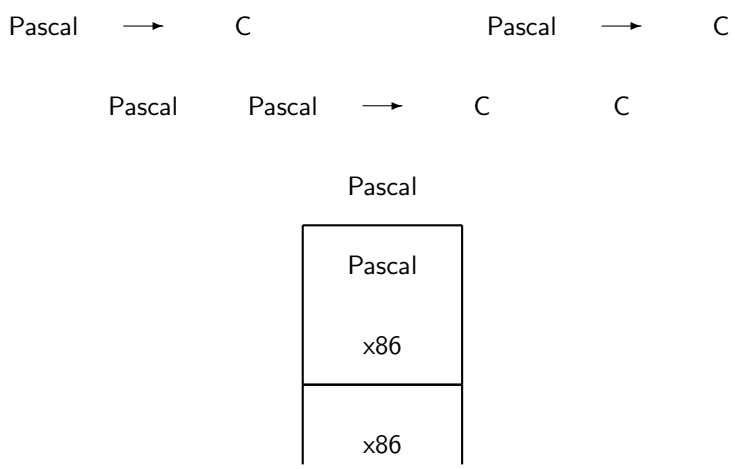
Answer 9:

9 [D] You can just take out such an interpreter and run directly on the underlying machine (or interpreter). In that sense, this is useless.

Such interpreters are called "meta-circular interpreters", and are used for programming language experiments.

Alternative A is directly contradicted by this. Alternative B and C are nonsense. Alternative E: Such interpreters are not tivial; you will have a chance to try your luck for oPL in a future challenge assignment.

Question 10: Consider the following T-diagram.



Which one of the following statements is true?

10 [A] This compilation is not possible, because a compiler cannot take itself as input.

10 [B] You can implement a compiler for a source language (here Pascal) in the source language itself and compile it, provided that you have an interpreter for the source language already.

10 [C] This compilation is useless, because in order to compile the compiler, you must have the compiler already compiled.

10 [D] It is trivial to implement a compiler for a source language (here Pascal) in the language itself. The compiler simply needs to print out its own code, whenever it is run.

10 [E] It is impossible to implement a compiler for a source language (here Pascal) in the language itself, because the program constructs of the language can only be processed in a language that is "higher" than the source language.

Answer 10:

10 [B] The compilation is perfectly fine. This approach is not useless; there are several examples of compilers written in their source language. Search for Java compilers written in Java, for example. As with writing any compiler, the task is not trivial, however. (Try it!)

Question 11: The language ePL does not require the programmer to declare the type of each expression. However, it is possible to type-check ePL programs. Which of the following statements is **false**?

11 [A] We could change ePL such that the type of every expression must be declared by the programmer. In this case, type checking could verify that the actual type is the same as the declared type.

11 [B] Let us assume we have a complicated well-typed ePL expression E. The type of E can be inferred just by looking at the topmost operator in the syntax tree for E.

11 [C] Let us assume we have a version of ePL, where the programmer declares the type of every expression using `{...}` as in simPL. Let us assume we have an ePL expression `{int} 1 + ({bool}` E`)`. In this case, the type checker does not have to analyse E in order produce the correct answer.

11 [D] In order to check whether a given program is well-typed, the type checker needs to traverse the entire expression. This would be the case even if the programmer would be required to declare the type of each expression.

11 [E] We could change ePL such that the type of every expression must be declared by the programmer. In this case, type checking would become much more difficult because every expression would have to be checked multiple times.

Answer 11:

11 [E] Type checking would not become more difficult. The type checker computes the type of the expression anyway. The only difference would be that this type needs to be compared with the declared type.

All other alternatives are true; discussions please in Forum Midterm Results.

Question 12: Compilers for high-level languages such as Java often simplify expressions in order to speed up the generated code. For example, the Java expression

```
x = x + 2 + 2 * x;
```

can be simplified to

```
x = 3 * x + 2;
```

Which of the following statements is **false**?

12 [A] The compiler from ePL to eVML could simplify any given complex well-typed expression by evaluating the entire expression, provided that division by zero remains in the program.

12 [B] The compiler could transform the ePL expression `1 + 2 + 3 + 4` such that the resulting eVML code consists of only two instructions.

12 [C] The compiler can transform any well-typed ePL expression such that the resulting eVML code consists of at most four instructions.

12 [D] The compiler from ePL to eVML cannot simplify any well-typed expression, because division by zero could occur at runtime.

12 [E] The compiler can transform any well-typed ePL expression, in which no integer constants appear, such that the resulting in eVML code consists of only two instructions.

Answer 12:

12 [D] This alternative contradicts A, which is true. A is true, because evaluation of ePL programs always terminates with a runtime that is proportional to the size of the program.

Alternative B is true; the two instructions are:

```
LDCI 10
DONE
```

Alternative C is true; in case of division by zero, the instructions have to be code like this:

```
LDCI 1
LDCI 0
DIV
DONE
```

Alternative E is true, since well-typed expressions in which no integer constants appear cannot have division by zero. In fact, they will always evaluate to a boolean value, which means they can be compiled to:

```
LDCB b
DONE
```

where `b` is either `true` or `false`. Further discussions in Forum Midterm Results.

Question 13: Let expressions in simPL are translated to expressions without let. What is the result of translating the following simPL expression to simPL without let?

```
let {int -> int} f = fun {int -> int} x -> 2 * x end
in {bool}
   let {bool} y = true
   in {bool}
      (f 3) = 4 & y
   end
end
```

13 [A]
```
(fun {bool -> bool} y ->
   (fun {(int -> int) -> bool} f ->
       (f 3) = 4 & y
    end
    fun {int -> int} x -> 2 * x end)
end
true)
```

13 [B]
```
(fun {(int -> int) * bool -> bool} f y ->
   (f 3) = 4 & y
end
fun {int -> int} x -> 2 * x end
true)
```

13 [C]
```
(fun {bool * (int -> int) -> bool} y f ->
   (f 3) = 4 & y
end
true
fun {int -> int} x -> 2 * x end)
```

13 [D]
```
(fun {(int -> int) -> bool} f ->
   (fun {bool -> bool} y ->
       (f 3) = 4 & y
    end
    true)
end
fun {int -> int} x -> 2 * x end)
```

13 [E]
```
((fun {bool -> int -> bool} y ->
    fun {int -> bool} x ->
       (fun {int -> int} x -> 2 * x end x) = 4
       & y
    end
 end
 true)
3)
```

Answer 13:

13 [D] All other alternatives are false. Discuss in Forum Midterm Results.

Question 14: Which one of the following statements about well-typed simPL expressions is true?

14 [A] There are well-typed simPL expressions that have free variables, and can be evaluated to a value according to the dynamic semantics of simPL.

14 [B] There are well-typed simPL expressions that have free variables, and whose evaluation according to the dynamic semantics of simPL does not terminate.

14 [C] The evaluation of well-typed simPL expressions according the dynamic semantics of simPL always terminates.

14 [D] During evaluation of a well-typed simPL expression, division by zero cannot occur.

14 [E] There are well-typed simPL expressions whose evaluation according to the dynamic semantics of simPL terminates, but does not produce a value.

Answer 14:

14 [E] The example is a well-typed program in which division by zero occurs. Evaluation according to dynamic semantics gets stuck in this case.

Alternative D contradicts this statement, and is thus false. A and B are false because well-typed expressions cannot contain free variables. C is false due to the counter example

```
(recfun f {int -> int} x -> (f x) end 0)
```

Question 15: In the definition of the dynamic semantics of simPL, the substitution applied to function definition introduces a new identifier in some cases. Which of the following statement is **true**?

15 [A] During the evaluation of well-typed expressions according to the dynamic semantics of simPL, new identifiers are introduced in some cases, where the formal parameter of a function definition appears as a free variable in a function argument.

15 [B] During the evaluation of well-typed expressions according to the dynamic semantics of simPL, new identifiers are introduced in some cases, where the body of a function definition has a free variable that appears in the function argument, as well.

15 [C] During the evaluation of well-typed expressions according to the dynamic semantics of simPL, new identifiers are introduced in some cases, where the argument of a function application is a function definition with free variables.

15 [D] There will not be any new identifier introduced during the evaluation of well-typed expressions according to the dynamic semantics of simPL.

15 [E] During the evaluation of simPL expressions according to the dynamic semantics of simPL, new identifiers are introduced whenever the formal parameter of a function definition appears in the function argument.

Answer 15:

15 [D] The only correct answer is D. The reason for this is that our dynamic semantics never evaluates within the body of a function. Therefore, arguments of function applications that are being executed never have free variables, and thus, the "capturing" case for substitution never applies.

All other alternatives are false, because they contradict with D.

Question 16: Type environment extension is a fundamental operation on type environments for the static semantics of simPL. Let us say we have type environments Γ and Γ' for which the following holds.

$$\Gamma[\texttt{x} \leftarrow \texttt{int}]\Gamma'$$

Which one of the following statements is **false**?

16 [A] There exist Γ and Γ' such that the above holds, and $\Gamma'(\texttt{x}) = \texttt{bool}$.

16 [B] There exist Γ and Γ' such that the above holds, and $\Gamma(\texttt{x}) = \texttt{bool}$.

16 [C] There exist Γ and Γ' such that the above holds, and $\Gamma(\texttt{x}) = \texttt{int}$.

16 [D] There exist Γ and Γ' such that the above holds, and $\Gamma(\texttt{x}) = \Gamma'(\texttt{x})$.

16 [E] There exist Γ and Γ' such that the above holds, and $\Gamma(\texttt{x}) \neq \Gamma'(\texttt{x})$.

Answer 16:

16 [A] After extension by a binding of `x` to `int`, the environment must return `int` when applied to `x`, and must not return `bool`. All other statements are true. Examples as homework, please discuss in Forum Midterm Results.

Question 17: What is the type of the following simPL expression, according to the static semantics presented in the lectures?

```
((fun {int -> int -> int -> int} x ->
     fun {int -> int -> int} y ->
        fun {int -> int} z ->
           z + 1
        end
     end
  end
  2)
 3)
```

17 [A] `int`

17 [B] `int -> int`

17 [C] `int -> int -> int`

17 [D] `int -> int -> int -> int`

17 [E] not well-typed

Answer 17:

17 [B] The correct type is `int -> int`.