Programming Paradigms

—Final Exam—

Department of Computer Science
University of Stuttgart

Summer Semester 2023, September 7, 2023

Note: The solutions provided here may not be the only valid solutions.

Part 1 [4 points]

1. Which of the following statements is true? (Only one statement is true.)
   - ☐ "Test-and-set" is an implicit type conversion that converts a Boolean value into a set.
   - ☒ "Test-and-set" is hardware instruction that atomically sets a variable to true and returns whether it was false before.
   - ☐ "Test-and-set" is an operation in Python that converts a value into a Boolean and sets it to true.
   - ☐ "Test-and-set" is a synchronization mechanism that checks whether a thread-local, Boolean variable has been set to true in all threads.
   - ☐ "Test-and-set" is a primitive in unit testing that checks whether a set contains the expected values.

2. Which of the following statements is true? (Only one statement is true.)
   - ☐ A memory leak occurs when a program statically allocates too much memory, e.g., in the form of large compile-time constants.
   - ☐ A memory leak occurs when a program allocates more memory on one execution path than on another execution path.
   - ☐ A memory leak occurs when a program frees memory that actually has not been allocated.
   - ☒ A memory leak occurs when dynamically allocated memory does not get freed even though it is not needed anymore.
   - ☐ A memory leak occurs when a loop writes beyond the bounds of a fixed-size data structure, such as an array.

3. Which of the following statements is true? (Only one statement is true.)
   - ☐ The scoping rules of a language define which part of a program's memory (static, stack, heap) a local variable gets stored in.
   - ☐ The scoping rules of a language define how to handle uncaught runtime exceptions.
   - ☐ The scoping rules of a language define whether the variables in a language may alias.
   - ☒ The scoping rules of a language define which bindings are active at a given program point.
   - ☐ The scoping rules of a language define which names are legal for variables and functions.

4. Which of the following statements is true? (Only one statement is true.)
   - ☐ A recursive type has at least two fields, one of which must be another object type.
   - ☐ A recursive type results in the same value when being passed to the address-of operator.
   - ☐ A recursive type allows values to be only statically allocated to ensure that all values fit into the available memory.
   - ☒ A recursive type is a composite type that references objects of the same type.
   - ☐ A recursive type is a type traversed by a recursive function.

Part 2 [12 points]

Consider the following grammar, where "(" and ")" are the terminals, ⟨S⟩ and ⟨X⟩ are the non-terminals, and ⟨S⟩ is the start symbol.

⟨S⟩ ::= ⟨X⟩ (rule 1)

⟨X⟩ ::= (⟨X⟩) (rule 2)
| () (rule 3)

Your task is to parse the following input using an LR(1) parser: (())

The following are the tables required for LR(1) parsing:

Action table:

State	(	)	EOF
0	shift 2	error	error
1	error	error	accept
2	shift 2	shift 5	error
3	error	shift 4	error
4	reduce 2	reduce 2	reduce 2
5	reduce 3	reduce 3	reduce 3

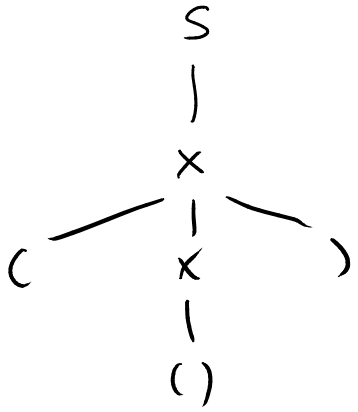
Goto table:

State	⟨X⟩
0	1
1	
2	3
3	
4	
5	

1. Provide the steps taken by the LR(1) parsing algorithm. For each step, indicate the current stack, the currently remaining input, and the action taken based on the stack and input. Use the following table as a template for your answer.

Stack	Remaining input	Action
EOF, 0	(, (,),), EOF	shift 2
EOF, 0, (, 2	(,),), EOF	shift 2
EOF, 0, (, 2, (, 2	),), EOF	shift 5
EOF, 0, (, 2, (, 2,), 5	), EOF	reduce 3
EOF, 0, (, 2, X, 3	), EOF	shift 4
EOF, 0, (, 2, X, 3,), 4	EOF	reduce 2
EOF, 0, X, 1	EOF	accept

2. Draw the parse tree created by the parser.



Part 3 [10 points]

Consider the following program, which uses a JavaScript-inspired syntax:

```
1  x = 2
2  y = 3
3
4  function foo() {
5    y = 4
6    function bar() {
7      print(x)
8      print(y)
9    }
10   bar()
11   return bar
12 }
13
14 function baz() {
15   x = 5
16   g = foo()
17   g()
18 }
19
20 baz()
```

We consider two semantics for executing the program:

- S_1, which uses static scoping and deep binding.
- S_2, which uses dynamic scoping and shallow binding.

1. Give the sequence of lines that get executed by the program. You can ignore lines that define a function. When a function gets called, first give the line that calls the function, then the lines in the body of the called function, and then the next line executed after the function call.

   Solution:

   1, 2, 20, 15, 16, 5, 10, 7, 8, 11, 17, 7, 8

2. Under S_1, what variables do `x` and `y` refer to when `bar` is executed for the first time? To answer the question, give the lines where the variables get defined.

   - `x` refers to the variable defined at line: 1
   - y refers to the variable defined at line: 5

3. Under S_2, what variables do `x` and `y` refer to when `bar` is executed for the first time? To answer the question, give the lines where the variables get defined.

   - x refers to the variable defined at line: 15
   - y refers to the variable defined at line: 5

4. The program creates a reference to a function, then passes on this reference, and eventually invokes the function.

   - The function reference is created at line: 11
   - The function gets called via this reference at line: 17

5. Under S_1, what variables do `x` and `y` refer to when `bar` is executed for the second time? To answer the question, give the lines where the variables get defined.

   - `x` refers to the variable defined at line: 1
   - `y` refers to the variable defined at line: 5

6. Under S_2, what variables do `x` and `y` refer to when `bar` is executed for the second time? To answer the question, give the lines where the variables get defined.

   - `x` refers to the variable defined at line: 15
   - `y` refers to the variable defined at line: 2

7. What does the program print?

   - Under S_1, the program prints:
     Solution:
     2
     4
     2
     4
   - Under S_2, the program prints:
     Solution:
     5
     4
     5
     3

Part 4 [12 points]

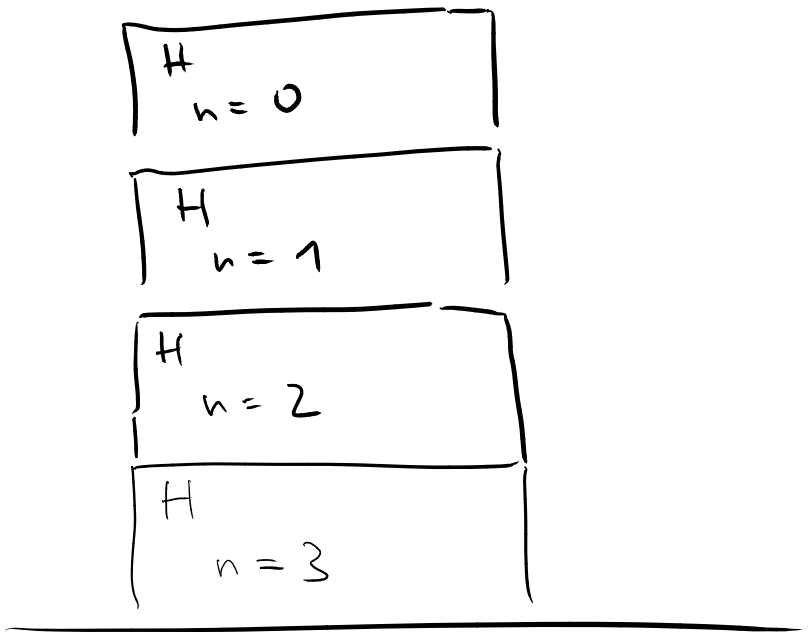
Consider the following Scheme code, which defines a function `H`.

```
(define H
    (lambda (n)
        (if (= n 0)
            0.0
            (+ (/ 1.0 n)
               (H (- n 1))))))
```

1. Provide a mathematical formula that describes the behavior of `H`.

   $H(n) = \sum_{i=1}^{n} \frac{1}{i}$

2. Suppose we invoke `H` as follows: `(H1 3)`.
   Draw the function stack that is created during this invocation at the point in time when the stack has reached its maximum size. Your drawing should include only stack frames of `H`, i.e., not of any of the built-in functions/operators, such as `+`. Use the following template, which illustrates what to show in a stack frame: The name of the invoked function (`H` in the given stack frame) and the values of any local variables and parameters (`n=3` in the given stack frame).



3. Suppose we invoke `H` with a large positive number, e.g., `(H 1000000)`. What kind of problem do you expect to occur?

   When invoked with n, `H` will create a stack of size $n + 1$. That is, when invoked with a large positive number, the stack will grow very large, and hence, the program takes very long and/or eventually runs out of memory.

4. Provide an alternative version of `H` that does not suffer from the problem you identified in the previous question. Give your answer as Scheme code.

```
(define H
    (lambda (n acc)
    (if (= n 0)
        acc
        (H (- n 1) (+ acc (/ 1.0 n))))))
```

   (Not needed to obtain the points, but FYI: Unlike the original `H`, this version of `H` is tail-recursive. This property allows the Scheme interpreter to reuse the existing stack frame when invoking `H` recursively. Hence, the stack size does not grow with the input size.)

Part 5 [10 points]

This question is about arithmetic expressions and the lambda calculus.

1. In the following program, mark all unbound variables with a dashed box and all bound variables with a solid box.

$$(\lambda\ a\ b\ c.\ if(iszero(pred\ \boxed{\text{a}}))\ succ\ \boxed{\text{c}}\ else\ \boxed{\text{x}})\ pred\ \boxed{\text{a}}$$

2. Evaluate the following program by providing the semantics of the above program, as a step-by-step sequence of operations.

   Solution:

   $(\lambda\ z.((\lambda\ x.\ true)\ z))\ 0$

   $\rightarrow (\lambda\ x.\ true)\ 0$

   $\rightarrow true$

   Alternative solution (the order of applying functions does not matter):

   $(\lambda\ z.((\lambda\ x.\ true)\ z))\ 0$

   $\rightarrow (\lambda\ z.\ true)\ 0$

   $\rightarrow true\ 0$

3. A program with n occurrences of the λ symbol may take a number of steps that is not equal to n. Give an example a program with two occurrences of the λ symbol that takes only one step to evaluate.

   Solution:
   Any program that evaluates to a function, e.g., $(\lambda\ x.\ x)(\lambda\ y.\ y)$

4. Let's now consider the typed lambda calculus. Under which typing context Γ does the following typing relation hold?

   $\Gamma \vdash (\lambda\ a.\ a)\ f : Bool \rightarrow (Bool \rightarrow Bool)$

   Solution:

   $f : Bool \rightarrow (Bool \rightarrow Bool)$

Part 6 [12 points]

1. Consider the following Prolog program:

```
student(john).
student_year(eve, 2).
student_year(john, 2).
course_year(pse, 1).
course_year(pp, 2).
has_exam(pp).
has_exam(pse).
takes(S, C) :- student(S), student_year(S, Y), course_year(C, Y).
takes_exam(S, C) :- takes(S, C), has_exam(C).
```

   (a) How many occurrences of constants, variables, and structures exist in the program? ("Occurrences" means that if a variable, etc. appears twice, it counts twice.)

   - Constants: 11
   - Variables: 12
   - Structures: 14

   (b) How many clauses, facts, rules, and goals/queries exist in the program?

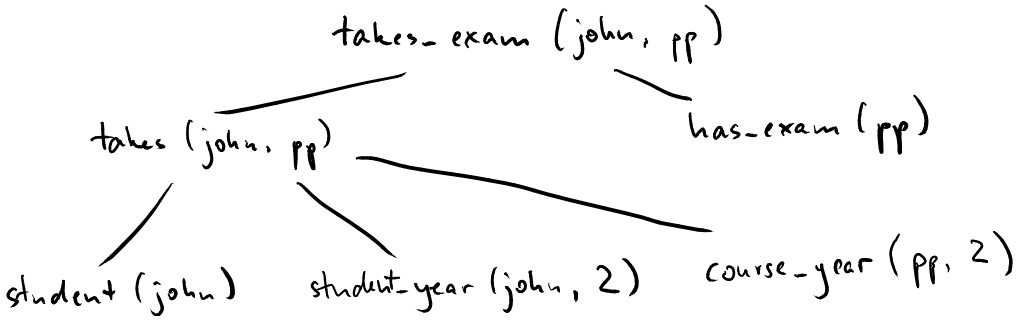
   - Clauses: 9
   - Facts: 7
   - Rules: 2
   - Goals/queries: 0

   (c) What is the output of the query `?- takes_exam(john, pp).`?
   Solution: `true.`

   (d) To answer the query in question (c), Prolog either constructs a proof tree or determines that no such tree exists. Which of the two cases applies here? In the former case, provide the proof tree. In the latter case, explain why no proof tree exists.

   Solution:

   Prolog constructs the following proof tree:



(e) What is the output of the query `?- takes_exam(john, eve).`?
Solution: `false.`

(f) To answer the query in question (e), Prolog either constructs a proof tree or determines that no such tree exists. Which of the two cases applies here? In the former case, provide the proof tree. In the latter case, explain why no proof tree exists.

Solution:

Prolog fails to construct a proof tree because the query cannot be unified with any of the facts or rules. In particular, Prolog cannot find any way to show that `student(eve)` is true, which is required to show that `takes(eve, pp)` holds.