IN4303 — Compiler Construction

Exam

January 31, 2017

---

1. You can make each question in WebLab or on paper.
2. When you answer a question in WebLab, make sure you save and submit the answer. Most questions come with several subquestions. Make sure you save each of your answers separately. Answers will be lost, if you only submit at the end of the page.
3. When you answer a question on paper, answer each question on its own page (each sheet has four pages).
4. Put your name, your student ID and the number of the question on top of each page.
5. If you need more than one sheet to answer a question, number your pages and state the overall number of pages for this question on the first page.
6. Take care of your time. This exam has 10 questions, for a total of 150 points. Try to answer a question worth 10 points in 10 minutes.
7. Keep your answers short and precise. Do not waste your time on essay writing. If you cannot answer a particular question, admit it and move on.
8. Hand in the answers together with this form (including the questions).

Good luck!

Name: _____ Student ID: _____

Question	Points	Score
Language specification	5	
Formal Grammar	10	
Lexical analysis	20	
LR parsing	20	
Term rewriting	10	
Name resolution	20	
Java Virtual Machine	10	
Dataflow Analysis	20	
Register allocation	15	
Garbage collection	20	
Total	150	

Question 1: Language specification (5 points)

(a) What is a *language workbench*? (1)

Solution: A language workbench is a tool for the construction of languages using high-level declarative meta-languages from which various language implementation artifacts such as IDEs, type checkers, interpreters, and/or compilers are generated.

(b) Name three different aspects of a programming language, which need to be specified or implemented. Explain, what each aspect is about. (2)

Solution:

- Syntax: defines the structure of well-formed programs of a language.
- Static semantics: defines which well-formed sentences have correct name binding and typing.
- Dynamic semantics: defines the execution behavior of well-typed programs.

(c) How can each of the different aspects of a programming language be formally specified? (2)

Solution:

- Syntax: context-free grammars
- Static semantics: Well-formedness and well-typedness judgements with inference rules.
- Dynamic semantics: Evaluation judgements with inference rules for small-step or big-step semantics

Question 2: Formal Grammar (10 points)

Let G be a formal grammar with nonterminal symbols E and B, terminal symbols **id**, $+$, $-$, $*$, and $/$, and the following production rules (annotated with constructor symbols in the style of SDF3):

$$\begin{aligned} E.I &\rightarrow \textbf{id} \\ E.B &\rightarrow E\ B\ E \\ B.P &\rightarrow + \\ B.S &\rightarrow - \\ B.M &\rightarrow * \\ B.D &\rightarrow / \end{aligned}$$

(a) Demonstrate that this grammar is ambiguous by giving two different abstract syntax trees for the sentence `id + id * id` and the left-most derivations that give rise to these trees. (3)

Solution:

```
Tree: B(B(I(), P(), I()), M(), I())

Derivation:
E
=> E B E
=> E B E B E
=> id B E B E
=> id + E B E
=> id + id B E
=> id + id * E
=> id + id * id
```

```
Tree: B(I(), P(), B(I(), M(), I()))

Derivation:
E
=> E B E
=> id B E
=> id + E
=> id + E B E
=> id + id B E
=> id + id * E
=> id + id * id
```

(b) Rewrite the grammar in SDF3 notation and use declarative priority and associativity rules to disambiguate it following standard rules. Define a translation from the ASTs of this grammar to the AST (3)

Solution:

```
context-free syntax
  E.I = ID
  E.P = E "+" E {left}
  E.S = E "-" E {left}
  E.M = E "*" E {left}
  E.D = E "/" E {left}
context-free priorities
  {left: E.M ED} > {left: E.P E.S}
```

(c) Instead of using separate disambiguation rules, transform the grammar to an unambiguous context-free grammar. (4)

Solution:

```
context-free syntax
  E.P = E "+" T
  E.S = E "-" T
  E   = T
  T.M = T "*" F
  T.D = T "/" F
  T   = F
  F.I = ID
```

Question 3: Lexical analysis (20 points)

Consider the following regular expression

$$r_1 \quad = \quad (\mathbf{0}|\ldots|\mathbf{9})^+(.(\mathbf{0}|\ldots|\mathbf{9})^+)^?(\mathbf{E}(\mathbf{0}|\ldots|\mathbf{9})^+)^?$$

(a) Describe the language defined by r_1 in English. (2)

Solution: The language describes floating point numbers with at least one digit before the decimal point, an optional decimal point followed by at least one digit, and an optional exponent consisting of at least one digit.

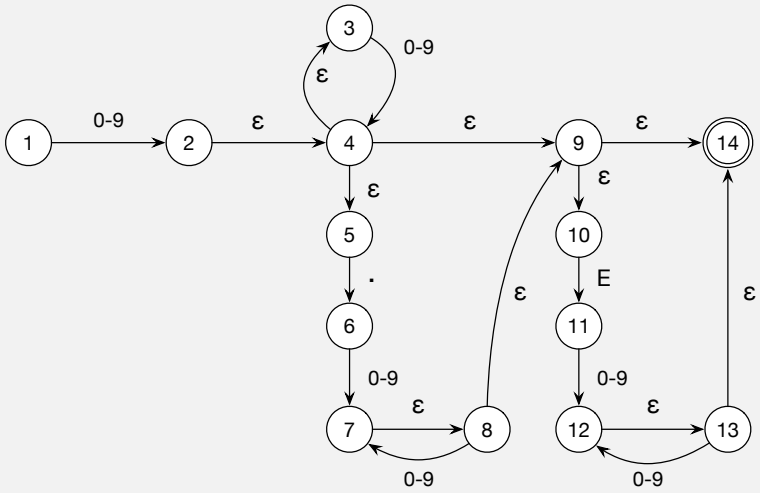
(b) Transform the regular expression r_1 to the equivalent expression r_2 that does not use the ? and + operators (4)

Solution:

$$r_2 \quad = \quad (\mathbf{0}|\ldots|\mathbf{9})(\mathbf{0}|\ldots|\mathbf{9})^*(\epsilon|(.(\mathbf{0}|\ldots|\mathbf{9})(\mathbf{0}|\ldots|\mathbf{9})^*))(\epsilon|(\mathbf{E}(\mathbf{0}|\ldots|\mathbf{9})(\mathbf{0}|\ldots|\mathbf{9})^*))$$

(c) Systematically construct an equivalent finite automaton for the r_2 expression (4)

Solution:

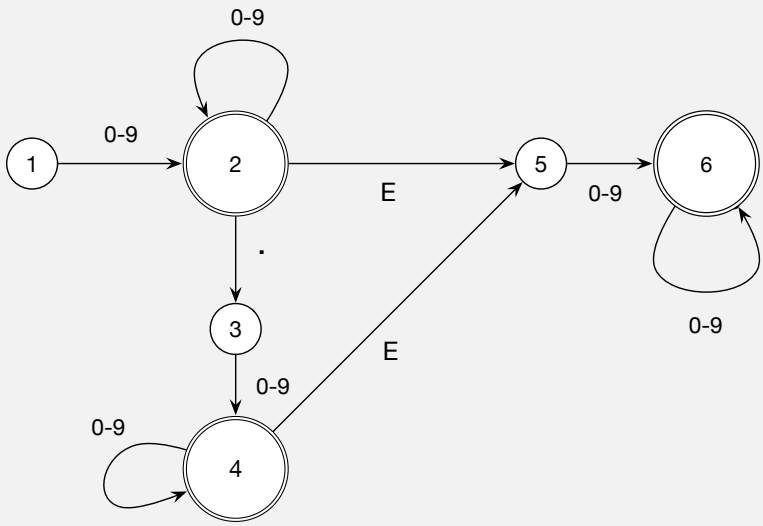


(d) What does equivalence in this case mean? (1)

Solution: The regular expression and automaton accept the same language.

(e) Transform the automaton to an equivalent automaton without ε-moves. (4)

Solution:



(f) Is the resulting automaton deterministic? Why (not)? (1)

Solution: Yes, the automaton is determistic, since the transitions from states are non-overlapping.

(g) Use the automaton to generate a word with at least five characters. Enumerate the states passed during the generation. (2)

Solution:

```
[1] 0 [2] . [3] 5 [4] 6 [4] E [5] 9 [6]  ==> 0.56E9
```

(h) Use the automaton to recognize the word **3.14E12**. Enumerate the states passed during the recognition. (2)

Solution:

```
[1] 3 [2] . [3] 1 [4] 4 [4] E [5] 1 [6] 2 [6]
```

Question 4: LR parsing (20 points)

Let G be a formal grammar with nonterminal symbols S and E, terminal symbols **id**, +, (,) and \$, start symbol S, and the following production rules:

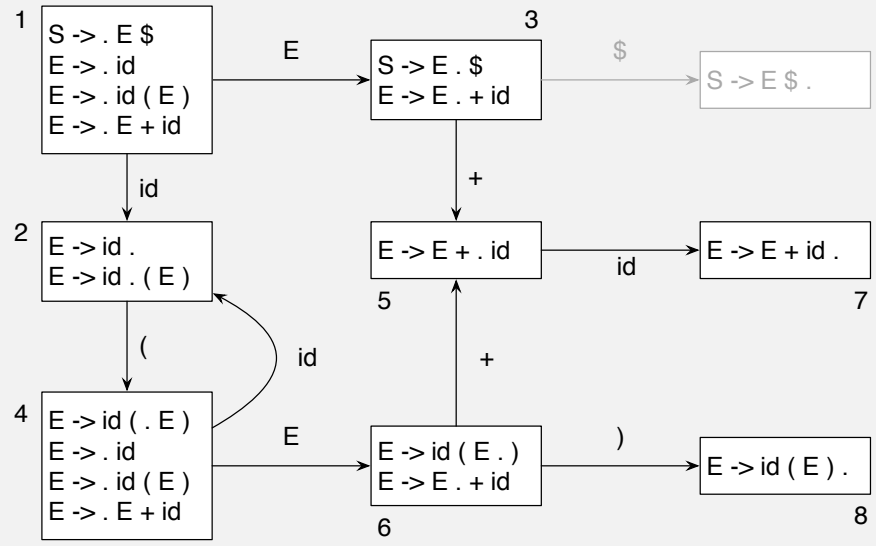
$$\begin{array}{lcl} S & \rightarrow & E\ \$ \\ E & \rightarrow & \textbf{id} \\ E & \rightarrow & \textbf{id}\ (\ E\ ) \\ E & \rightarrow & E\ +\ \textbf{id} \end{array}$$

(a) Explain the role of the terminal symbol \$. (1)

Solution: The terminal symbol \$ indicates the last character of the input.

(b) Draw a diagram with the LR(0) states and transitions for grammar G (8)

Solution:



(c) Build the LR(0) parse table for grammar G (5)

Solution:

	id	+	(	)	\$	S	E
1	s2						g3
2	r2	r2	r2,s4	r2	r2		
3		s5			a		
4	s2						g6
5	s7						
6		s5		s8			
7	r4	r4	r4	r4	r4		
8	r3	r3	r3	r3	r3		

(d) Is this an SLR grammar? Why (not)? (2)

Solution: Yes. The FOLLOW set of non-terminal E is $\{\$, +\}$. Thus, production 2 is only reduced when the next character is \$ or +, which solves the shift-reduce conflict for state 2.

(e) Use the parse table to recognize the sentence `id(id(id) + id)$`. Show the stack and the remaining input after each step. (4)

Solution:

```
   [1] id(id(id) + id)$
s2
   [1, 2] (id(id) + id)$
s4
```

```
   [1, 2, 4] id(id) + id)$
s2
   [1, 2, 4, 2] (id) + id)$
s4
   [1, 2, 4, 2] (id) + id)$
s4
   [1, 2, 4, 2, 4] id) + id)$
s2
   [1, 2, 4, 2, 4, 2] ) + id)$
r2, g6
   [1, 2, 4, 2, 4, 6] ) + id)$
s8
   [1, 2, 4, 2, 4, 6, 8] + id)$
r3, g6
   [1, 2, 4, 6] + id)$
s5
   [1, 2, 4, 6, 5] id)$
s7
   [1, 2, 4, 6, 5, 7] )$
r4, g6
   [1, 2, 4, 6] )$
s8
   [1, 2, 4, 6, 8] $
r3, g3
   [1, 3] $
a
```

Question 5: Term rewriting (10 points)

Consider the following algebraic signature representing the abstract syntax of an expression language in *Stratego* :

```
signature
  constructors
    Var : String -> E
    Int : Int -> E
    Add : E * E -> E
    Mul : E * E -> E
    And : E * E -> E
    Or  : E * E -> E
    If  : E * E * E -> E
```

In this language integers are used to represent Boolean values, with `Int(0)` representing false and all other integers representing true. The Boolean operators `And` and `Or` are short-circuit operations.

(a) Define in Stratego a desugaring transformation to eliminate the `And` and `Or` operators. (2)

Solution:

```
rules

  desugar : And(e1, e2) -> If(e1, e2, Int(0))

  desugar : Or(e1, e2) -> If(e1, Int(1), e2)

  desugar-all = innermost(desugar)
```

(b) Give the term resulting from desugaring the term `And(Var("x"), Or(Int(1), Var("y")))` (2)

Solution:

```
<desugar-all> And(Var("x"), Or(Int(1), Var("y")))
=> If(Var("x"), If(Int(1), Int(1), Var("y")), Int(0))
```

(c) Define in Stratego a strategy that, given a transformation that maps variables to expressions, applies this transformation to all variables in an expression; use only basic operators. (2)

Solution:

```
strategies

  substitute(s) = s <+ all(substitute(s))
```

(d) Define in Stratego a constant folding transformation for desugared expressions. (4)

Solution:

```
strategies

  eval : Add(Int(i), Int(j)) -> Int(<add>(i, j))
  eval : Mul(Int(i), Int(j)) -> Int(<mul>(i, j))

  eval : If(Int(0), e2, e3) -> e3
  eval : If(Int(i), e2, e3) -> e2 where not(<eq>(i, 0))

  eval-all = bottom-up(try(eval))
```

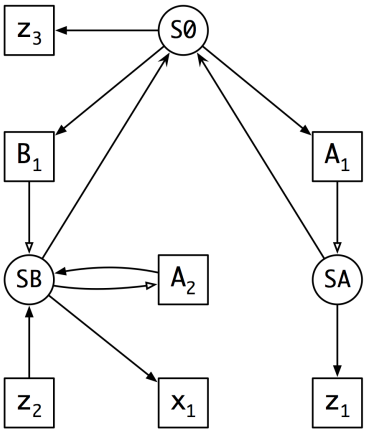
Question 6: Name resolution (20 points)

(a) Explain how the scope graph approach to name resolution works. (5)

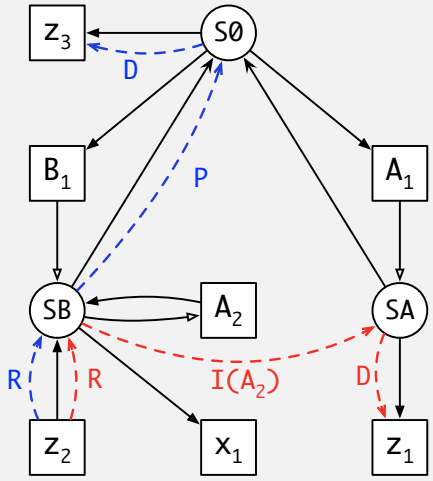
Solution: Include at least the following elements

- AST is mapped to a scope graph
- Scope graph consists of scopes connected by scope edges
- References and declarations corresponding to name occurrences in program are associated with a scope
- Name resolution consists of finding a path from a reference to a declaration with the same name following the directed edges in the graph
- Multiple paths for the same reference can be disambiguated using path specificity ordering and path well-formedness predicate

(b) Given the following scope graph, list all reachability paths for reference z_2 and indicate the path selected according to visibility rules. Motivate your answer. (5)



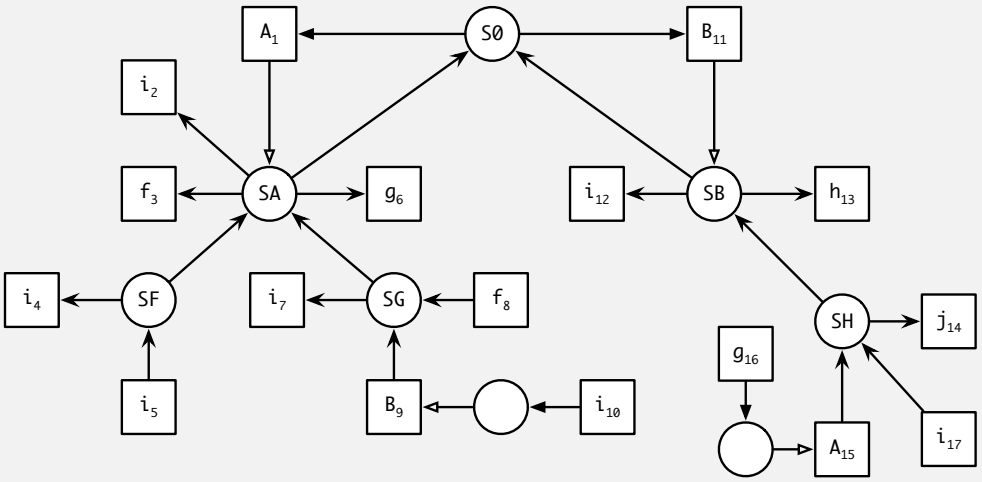
Solution: Path 1: $R.P.D$ from z_1 to z_3
Path 2: $R.I(A_2).D$ to z_1 with path for import A_2 is $R.P.D$ to A_1
The second path is preferred since imports shadow parents.



(c) Given the following Java program, draw its scope graph.Note that the subscripts of identifiers indicate the position of the identifier in the program. Two occurrences x_i and x_j denote different occurrences of the same name. (5)

```
class A1 {
  static int i2;
  static int f3(int i4) {
    return i5;
  }
  static int g6(int i7) {
    return f8(B9.i10);
  }
}
class B11 {
  static int i12;
  static int h13(int j14) {
    return A15.g16(i17);
  }
}
```

Solution:



Remarks
i˙10 is reference in an anonymous scope importing B˙9
g˙16 is reference in an anonymous scope importing A˙15

(d) For each reference in the program list its resolution path and the declaration to which it resolves. (5)

Solution:

- i_5: *R.D* to i_4
- f_8: *R.P.D* to f_3
- B_9: *R.P.P.D* to B_{11}
- i_{10}: $R.I(B_9).D$ through import of B to i_{12}
- A_{15}: *R.P.P.D* to A_1
- g_{16}: $R.I(A_{15}).D$ through import of A to g_6
- i_{17}: *R.P.D* to i_{12}

Question 7: Java Virtual Machine (10 points)

Execute the bytecode instructions of `Main/main()V`, starting with an empty stack. The initial value of local variable 0 is `4242 4303`, pointing to an object of class `Foo`. Show stacks and local variables after each instruction. If you have to invoke a method, execute the bytecode of this method as well. Make clear when stack frames are created and destroyed, and which data is passed between frames.

```
_________ Main/main()V _________
    aload_0
    bipush 4
    iconst_2
    isub
    invokevirtual Foo/f(I)V
________________________________
```

```
_________ Foo/f(I)V _________
    iload_1
    goto l2
l1: iload_1
    ldc -2
    iadd
    dup
    istore_1
l2: ifne l1
    return
_____________________________
```

Question 8: Dataflow Analysis (20 points)

Consider the following intermediate code:

```
 1      c := r3
 2      a := r1
 3      b := r2
 4      d := 0
 5      e := a
 6  l1: d := d + b
 7      e := e - 1
 8      if e > 0 goto l1
 9      r1 := d
10      r3 := c
11      return
```

(a) Construct the control graph for the shown intermediate code on paper. You are allowed to omit the actual code and to use line numbers instead. (2)

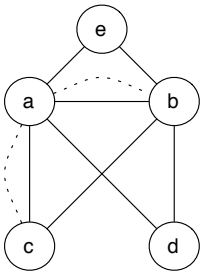
(b) Provide a table with successor nodes, defined variables, and used variables for each node in the control graph. (3)

(c) Calculate live-ins and live-outs for each node in the control graph in a table on paper. Show each round of calculation. (15)

Question 9: Register allocation (15 points)

Liveness analysis of the intermediate code on the left yields the interference graph on the right, which should be coloured with two colours.

```
s: if a = 0 goto r
   e := b / a
   d := e * a
   c := b - d
   b := a
   a := c
   goto s
r: return b
```



(a) Can both move-related edges be removed safely? Why (not)? (1)

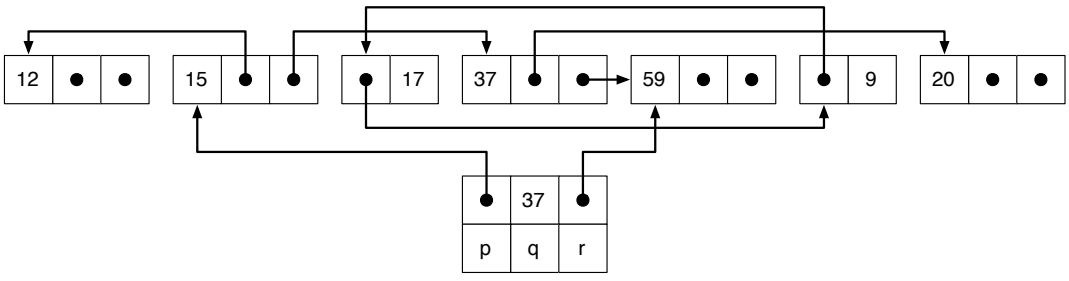
(b) Assume the move-related edges were removed. Why does the next step need to be a spill? (2)

(c) Spill node `a` and continue the graph colouring until you can decide if this spill is an actual one. (7)

(d) Is node `a` an actual spill? (1)

(e) Perform the spill on the intermediate code. (4)

Question 10: Garbage collection (20 points)



(a) What kind of data is stored on the heap? (2)

(b) Perform a copy collection on the given heap data structure. Show the data structure after each copying step (after pointer adjustments). (15)

(c) Explain the key benefit of copy collection over mark-and-sweep garbage collection. (1)

(d) Explain the effect of copy collection on the locality of data. (2)