Examination in Compilers, EDAN65

Department of Computer Science, Lund University

2025–10–28, 14.00-19.00

Note! **Your exam will be marked only if you have completed all six programming lab assignments in advance.**

Start each solution (1, 2, 3, 4) on a separate sheet of paper. Write only on one side of each sheet. Write your *anonymous code* and *personal identifier*[1] on every sheet of paper. Write clearly and legibly. Try to find clear, readable solutions with meaningful names. Unnecessary complexity will result in point reduction.

The following documents may be used during the exam:

- *Reference manual for JastAdd2*
- *x86 Cheat Sheet*

Max points: 60
For grade 3: Min 30
For grade 4: Min 40
For grade 5: Min 50

Good luck!

[1]The *personal identifier* is a short phrase, a code or a brief sentence of your choice. It can be anything, but not something that can reveal your identity. The purpose of this identifier is to make it possible for you to identify your exam in case something goes wrong with the anonymous code on the exam cover (such as if it is confused with another code due to sloppy writing).

1 Lexical analysis

A language has the following token definitions for whitespace, the `on` and `or` keywords, the ++ operator, and identifiers, defined in a `jflex` specification:

```
WS = (" ")+          { return new Token("WS"); }
ON = "on"            { return new Token("ON"); }
OR = "or"            { return new Token("OR"); }
PP = "++"            { return new Token("PP"); }
ID = [a-z][a-z0-9]*  { return new Token("ID", yytext()); }
```

The usual disambiguation rules of rule priority and longest match apply.

a) Draw five finite automata, one for each of the token definitions. Mark the final state of each automaton with the token in question. (5p)

b) Combine the five automata to an NFA by joining their start states, and label each state with a unique number. (1p)

c) Construct a DFA which is equivalent to the NFA you constructed in 1b). Each state in the DFA should be labelled with the set of corresponding states from the NFA. Each final state should be marked with the appropriate token. (5p)

d) To handle errors in the input, an error token is introduced by adding the following rule after all the other rules:

```
ERR = [^]            { return new Token("ERR", yytext()); }
```

Given the string "`on1 or on+++`" as input, what sequence of tokens will be generated? The answer only needs to include the token *kind*, i.e., the first parameter used in the Token constructors. Note that this scanner generates tokens also for whitespace. (2p)

e) Suppose earliest match is used, rather than longest match. What sequence of tokens would then be generated from the string "`on1 or on+++`"? (2p)

2 Context-Free Grammars

a) Consider the following context-free grammar. (`ID` is a predefined token for identifiers and `EOF` is a token representing end of file.)

p_0: `S -> E EOF`
p_1: `E -> E "+" E`
p_2: `E -> E "**" E`
p_3: `E -> "(" E ")"`
p_4: `E -> ID`

The grammar is ambiguous. Prove this by finding a sentence that can be matched with two different parse trees, and draw the two trees. (5p)

b) Construct an unambiguous grammar that is equivalent to the one in 2 a), and where `"**"` is right-associative, `"+"` is left-associative, and `"**"` has higher priority than `"+"`. The grammar should be on canonical form. (5p)

c) Now, consider the following grammar:

p_0: `S -> E EOF`
p_1: `E -> ID`
p_2: `E -> ID "(" L ")"`
p_3: `L ->` ϵ
p_4: `L -> M E`
p_5: `M -> M E ","`
p_6: `M ->` ϵ

Construct the FOLLOW set for the nonterminal `E` for this grammar. Prove that each element belongs to FOLLOW by showing derivations from the start symbol `S`, and marking out the element (e.g., by underscoring it). A derivation should be written in the form

`S` $\Rightarrow$ `...` $\Rightarrow$ `...`

where only one nonterminal is replaced in each derivation step. (5p)

d) Construct the LL(1) parser table for the grammar in 2 c). (Since the grammar is not LL(1), there will be at least one conflict in the table.) (5p)

e) Construct a grammar on EBNF form that is equivalent to the grammar in 2 c). The EBNF grammar should have as few nonterminals as possible. (Recall that EBNF allows the use of alternatives, repetition, optionals, and parentheses.) (5p)

3 Program analysis

A Roman numeral is a sequence consisting of the letters I, V, X, L, C, D, and M, each with the face value 1, 5, 10, 50, 100, 500, and 1000, respectively. To compute the value of a Roman numeral, the face values are added together, except for letters that are in *subtractive position*, in which case the face value is instead subtracted.

For example, the numeral MMXXV means 2025 $(1000 + 1000 + 10 + 10 + 5)$. As an example of a numeral with a letter in subtractive position, consider XIV which means 14 $(10 - 1 + 5)$. Here, I is in subtractive position because the V to the right of it has a greater face value $(5 > 1)$.

Standard Roman numerals have many rules about where subtractive letters may be placed, and how many times certain letters may be repeated. We will instead consider *simple* Roman numerals, where letters can be placed in any order, and repeated any number of times. For simple Roman numerals, a letter is considered to be in subtractive position if *any* letter to the right of it has a greater face value, i.e., not only the neighbor immediately to the right. In simple Roman numerals, there are many ways to write down the same number. For example, the number 42 can be written:

```
XXXXII   // 10 + 10 + 10 + 10 + 1 + 1 = 42
XXXVVII  // 10 + 10 + 10 + 5 + 5 + 1 + 1 = 42
XLII     // -10 + 50 + 1 + 1 = 42
IVIIL    // -1 - 5 - 1 - 1 + 50 = 42
IIIIVLI  // -1 - 1 - 1 - 1 - 5 + 50 + 1 = 42
```

Note that for a numeral like IVIIL, the first I is in subtractive position because it is followed by V, and the three letters in VII are also in subtractive position because each of them has a higher valued letter (L) somewhere to the right of it.

We will represent Roman numerals using the following abstract grammar:

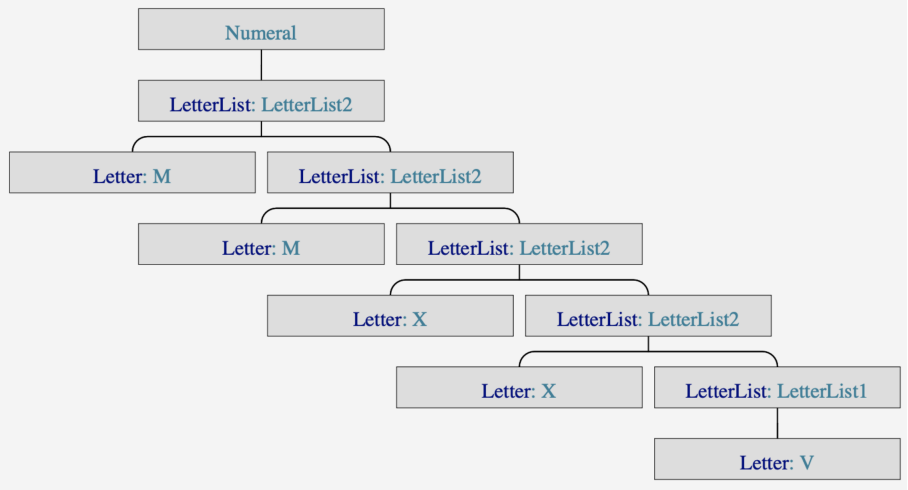
```
Numeral ::= LetterList;
abstract LetterList;
LetterList1 : LetterList ::= Letter;
LetterList2 : LetterList ::= Head:Letter Tail:LetterList;
abstract Letter;
I : Letter;
V : Letter;
X : Letter;
L : Letter;
C : Letter;
D : Letter;
M : Letter;
```

Solve the following problems using reference attribute grammars. In your solutions,

- you must not use the `getParent()` method
- you must not use `instanceof` or the `switch` construct
- you must not introduce any helper methods

For each of the `Letter` types, the attribute `int facevalue()` is already defined, returning the face value. E.g., for the letter V, the face value is 5.

The figure below shows the AST for the numeral MMXXV.



a) Define a boolean attribute `Letter.inSubPos()`. The value of `inSubPos` for a letter l should be *true* if there is another letter with a greater face value somewhere to the right of l, and *false* otherwise. For the example MIVIICL, `inSubPos` will be false for the first letter, true for the next four ones, and false for the two last ones. Introduce additional attributes as needed. (6p)

b) Define an `int` attribute `Numeral.value()` that computes the value of a Roman numeral and that makes use of a collection attribute for the computation. For the example MIVIICL, the value 1142 should be computed (1000 - 1 - 5 - 1 - 1 + 100 + 50). For the collection, you can use the following class:

```
class Counter {
    int count = 0;
    void add(int amount) {
        count += amount;
    }
    int count() {
        return count;
    }
}
```

(4p)

4 Code generation and run-time systems

The program on the next page is written in a Java-like language. The program includes a class `Turtle` for simple turtle graphics, and a subclass `BoxyTurtle` that can also draw boxes. The `main` method creates a new `BoxyTurtle` and calls the `drawBox` method which in turn calls `forward` and `turnleft` a number of times. The `forward` method plots a line on a canvas.

The language is implemented without any optimizations. For method calls, all arguments are pushed in reverse order on the stack (like in the labs). The static link (the `this` pointer), is treated as an implicit first argument.

a) Consider the second invocation of the `forward` method, i.e., when it is called with the argument 200. Draw the runtime situation on the stack and heap when this method is about to execute the call instruction to `plotline`, i.e., after the arguments to `plotline` have been pushed.

   Your drawing should include:

   - method activations with dynamic links, static links (`"this"` pointers), arguments, and local variables, including their values when they are known
   - method names for frames
   - frame pointer and stack pointer
   - objects with fields, including their values

   (7p)

b) Translate the statement `drawnBoxes++;` in the method `drawBox` to x86 code. Use only the instructions on the x86 Cheat Sheet. The code should be consistent with your drawing.

   (For simplicity and readability, you may leave out the characters q, %, and , in the code. For example, you may write `add $8 rax` instead of `addq $8, %rax`.) (3p)

```
class Program {
    public static void main(String[] args) {
        BoxyTurtle t = new BoxyTurtle();
        t.setCanvas(new Canvas());
        t.drawBox(100,200);
    }
}

class Canvas {
    void plotline(int x1, int y1, int x2, int y2) { ... }
}

class Turtle {
    Canvas canvas;
    int x = 0;
    int y = 0;
    int dirx = 1;
    int diry = 0;

    void setCanvas(Canvas c) {
        canvas = c;
    }

    void turnLeft() {
        int temp = dirx;
        dirx = -diry;
        diry = temp;
    }

    void forward(int distance) {
        int newx = x + dirx * distance;
        int newy = y + diry * distance;
        canvas.plotline(x, y, newx, newy);
        x = newx;
        y = newy;
    }
}

class BoxyTurtle extends Turtle {
    int drawnBoxes = 0;
    void drawBox(int width, int height) {
        forward(width);
        turnLeft();
        forward(height);
        turnLeft();
        forward(width);
        turnLeft();
        forward(height);
        turnLeft();
        drawnBoxes++;
    }
}
```