

EDAN65: Compilers, Exercise set E-13

# Problems

Görel Hedin

Revised: 2024-10-15

# Problem E13-1

A language L is described by following context-free grammar:

p1:  $E \rightarrow E "+" E$

p2:  $E \rightarrow E "*" E$

p3:  $E \rightarrow ID$

where E is the start symbol, and ID is a terminal symbol representing an identifier. Prove by writing down a left-most derivation that

$ID "+" ID "*" ID$

belongs to L. For each derivation step, show which production was used.

# Problem E13-2

Consider the following context-free grammar for a textual representation of a graph with labelled nodes and edges. The start symbol is Graph:

```
Graph -> ElementList
ElementList -> Element ElementList
ElementList -> ε
Element -> Node
Element -> Edge
Node -> ID
Edge -> ID "(" ID "->" ID ")"
```

The terminal ID has the following regular expression definition:

ID = [a-z]<sup>+</sup>

Draw the parse tree for the following graph:

a e(a->b)

# Problem E13-3

Consider the following context-free grammar for a textual representation of a graph with labelled nodes and edges. The start symbol is Graph:

p1: Graph  $\rightarrow$  ElementList  
p2: ElementList  $\rightarrow$  Element ElementList  
p3: ElementList  $\rightarrow \epsilon$   
p4: Element  $\rightarrow$  Node  
p5: Element  $\rightarrow$  Edge  
p6: Node  $\rightarrow$  ID  
p7: Edge  $\rightarrow$  ID "(" ID  $\rightarrow$  ID ")"

This grammar is not LL(1). Explain why.

# Problem E13-4

The following grammar contains a common prefix.  
Transform the grammar to an equivalent grammar  
where the common prefix is eliminated.

```
Graph -> ElementList
ElementList -> Element ElementList
ElementList -> ε
Element -> Node
Element -> Edge
Node -> ID
Edge -> ID "(" ID "->" ID ")"
```

# Problem E13-5

The following grammar is left-recursive and therefore not LL(1). Transform the grammar to an equivalent grammar that is LL(1). Argue for that your resulting grammar is LL(1).

```
T -> T "*" F
T -> F
F -> ID
F -> "(" T ")"
```

# Problem E13-6

Consider the following context-free grammar for a textual representation of a graph with labelled nodes and edges. The start symbol is G:

p1:  $G \rightarrow \text{ElemList}$   
p2:  $\text{ElemList} \rightarrow \text{Elem ElemList}$   
p3:  $\text{ElemList} \rightarrow \epsilon$   
p4:  $\text{Elem} \rightarrow \text{Node}$   
p5:  $\text{Elem} \rightarrow \text{Edge}$   
p6:  $\text{Node} \rightarrow \text{ID}$   
p7:  $\text{Edge} \rightarrow \text{ID} "(" \text{ID} "->" \text{ID} ")"$

The terminal ID has the following regular expression definition:

$$\text{ID} = [a-z]^+$$

Show how an LR parser would parsing the following program:

a e(a->b)

Show the stack contents, the remaining input, and the parsing action taken in each step.

# Problem E13-7

Consider the following abstract grammar for a graph of nodes and edges.

```
G ::= Element*;  
abstract Element;  
Node:Element ::= <ID>;  
Edge:Element ::= Src:NodeUse Dst:NodeUse;  
NodeUse ::= <ID>;
```

Suppose there is an attribute

```
Node NodeUse.maybeNode()
```

that refers to the node of the same name as the NodeUse, or to null if there is no such node.

Define a boolean synthesized attribute wellFormed() for Edge nodes, that is true iff both its source and destination nodes exist.



# Problem E13-8

Consider the following abstract grammar for a graph of nodes and edges.

```
G ::= Element*;  
abstract Element;  
Node:Element ::= <ID>;  
Edge:Element ::= Src:NodeUse Dst:NodeUse;  
NodeUse ::= <ID>;
```

Suppose there is an attribute

```
Node NodeUse.maybeNode()
```

that refers to the node of the same name as the NodeUse, or to null if there is no such node.

To represent missing nodes, introduce a new AST class UnknownNode, and create an object of this class as an NTA of the root.

Define a new attribute

```
Node NodeUse.node()
```

that refers to the UnknownNode object instead of to null.

# Problem E13-9

Consider the following abstract grammar for a graph of nodes and edges.

```
G ::= Element*;  
abstract Element;  
Node:Element ::= <ID>;  
Edge:Element ::= Src:NodeUse Dst:NodeUse;  
NodeUse ::= <ID>;
```

Implement an attribute

```
Node NodeUse.maybeNode()  
that refers to the node of the same name as the  
NodeUse, or to null if there is no such node.
```

# Problem E13-10

Consider the following abstract grammar for a graph of nodes and edges.

```
G ::= Element*;  
abstract Element;  
Node:Element ::= <ID>;  
Edge:Element ::= src:NodeUse dst:NodeUse;  
NodeUse ::= <ID>;
```

Define an attribute

```
int G.nbrOfEdges()
```

that counts the number of edges in the graph. Use a collection attribute to compute the attribute. You can use a class Counter with the following implementation:

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

# Problem E13-11

Consider the following abstract grammar for a graph of nodes and edges.

```
G ::= Element*;  
abstract Element;  
Node:Element ::= <ID>;  
Edge:Element ::= src:NodeUse dst:NodeUse;  
NodeUse ::= <ID>;
```

Suppose there is an attribute

```
Node NodeUse.maybeNode()
```

that refers to the node of the same name as the NodeUse, or to null if there is no such node.

If there is an edge  $a \rightarrow b$ , we say that the node  $b$  is a target of  $a$ . Implement a collection attribute `Node.targets()` containing all the target nodes for a given node.

For sets, you may use the Java type `HashSet`.

# Problem E13-12

Consider the following abstract grammar for a graph of nodes and edges.

```
G ::= Element*;  
abstract Element;  
Node:Element ::= <ID>;  
Edge:Element ::= Src:NodeUse Dst:NodeUse;  
NodeUse ::= <ID>;
```

If there is an edge  $a \rightarrow b$ , we say that the node  $b$  is a target of  $a$ . Suppose there is a collection attribute

```
Set<Node> Node.targets()  
containing all the target nodes for a given node.
```

The reachable set of a node is the transitive set of target nodes. Implement the reachable set as a circular attribute. You can use the Java class `HashSet` with operations `add` and `addAll`, for adding one element or a set of elements.

# Problem E13-13

```
class Account {
    int balance = 0;
    void deposit(int amount) {
        balance = balance + amount;
    }
    void withdraw(int amount) {
        if (amount > balance)
            overdraft(amount - balance);
        else
            balance = balance - amount;
    }
    void overdraft(int am) {
        /* PC */
        System.out.println
            ("Overdraft with amount "+am);
    }
}
void test() {
    Account a = new Account();
    a.deposit(100);
    a.withdraw(150);
}
```

Suppose that test() is called. Draw the situation on the stack and heap at /\* PC \*/. Your sketch should include dynamic link, fields, local variables, "this" pointer, and arguments including their values. Arguments should be passed on the stack. Explain the contents of the withdraw activation.

# Problem E13-14

```
class Figure {
    int area() { return 0; }
}
class Rectangle extends Figure {
    int w;
    int h;
    void set(int w, int h) {
        this.w = w;
        this.h = h;
    }
    int area() {
        return w * h;
    }
    ...
}
```

Suppose this language is implemented using virtual tables. Draw a sketch over the memory showing a Rectangle object, its class descriptor, and its code. Your sketch should include fields, class link, virtual table, and methods.

# Problem E13-15

```
class Figure {
    int area() { return 0; }
}
class Rectangle extends Figure {
    int w;
    int h;
    void set(int w, int h) {
        this.w = w;
        this.h = h;
    }
    int area() {
        return w * h;
    }
    ...
}
void m(Figure f) {
    int a;
    a = f.area(); // S
}
```

This language is implemented using virtual tables. Draw the situation on stack and heap at statement S, right before the call to `f.area()` is made. Assume `f` is a `Rectangle` object and include the class descriptor in your sketch. Sketch the code for the statement S. Use x86 instructions according to the assignment 6 cheatsheet. Add comments to the code, explaining what it does.