Examination in Compilers, EDAN65

Department of Computer Science, Lund University

2021–10–29, 08.00-13.00

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 a token definition `INT` for integer literals. An `INT` may consist of digits and underscores, and `0`, `123`, `45_99`, and `066__7____28` are examples of valid `INT`s. However, an `INT` may not start or end with an underscore, so strings like `_54_5` and `67__` are not valid `INT`s.

The language also has a token definition `FLOAT` for float literals. A `FLOAT` is a sequence of characters that contains at least one digit and exactly one decimal point. Unlike `INT`s, a `FLOAT` may not contain any underscores. Examples of valid `FLOAT`s are:

```
05.
.781
34.57
```

a) Construct a regular expression for `INT`. (3p)

b) Construct a DFA for `INT`. The DFA should be minimal (contain as few states as possible). Give the states unique numbers. Mark the final state(s) with `INT`. (3p)

c) Construct a DFA for `FLOAT`. The DFA should be minimal. Give the states unique numbers, and that are different from the numbers you used in the `INT` DFA, except for the start state which may use the same number. Mark the final state(s) with `FLOAT`. (3p)

d) Construct a combined DFA by joining the start states for the `INT` and `FLOAT` DFAs, and eliminating any nondeterminism. Mark each state in the combined DFA with the corresponding state number(s) from the `INT` and `FLOAT` DFAs. Mark each final state in the combined DFA by the appropriate token. The DFA should be minimal. (4p)

e) Consider input to a scanner consisting of the following sequence of characters:

```
375_97_.42_3abc
```

Suppose that `INT` and `FLOAT` are defined as above, and that the scanner uses longest match when scanning. What token (`INT` or `FLOAT`) will be the first it will output when scanning the input, and what lexeme (string) does this token correspond to? Which additional characters will the scanner read before deciding to output this token? (2p)

2 Grammars

Consider the following context-free grammar for statements:

p_0 : `start` $\rightarrow$ `stmt EOF`
p_1 : `stmt` $\rightarrow$ `"{" stmt "}"`
p_2 : `stmt` $\rightarrow$ `stmt stmt`
p_3 : `stmt` $\rightarrow$ `ID "=" NUM ";"`
p_4 : `stmt` $\rightarrow \epsilon$

where `start` is the start symbol and the alphabet used is

```
{ "{", "}", "=", ";", ID, NUM, EOF }
```

a) The grammar is ambiguous. Prove this by constructing two different derivation trees for the same sentence. Try to find the smallest trees that can be used to show an ambiguity. (5p)

b) Construct the FOLLOW set for the nonterminal `stmt`. Prove that each of the elements belongs to FOLLOW by showing derivations from the start symbol. A derivation should be written in the form

`start` $\Rightarrow$ `...` $\Rightarrow$ `...`

where only one nonterminal is replaced in each derivation step.

(5p)

c) Construct an equivalent grammar on EBNF form with as few nonterminals and productions as possible.

(5p)

d) Construct an equivalent grammar on canonical form and that is LL(1).

(5p)

3 Abstract grammars and analysis

The programming language Pyva (made up for this exam) is a bit similar to Python. Functions are defined using the `def` keyword, and can be nested inside each other. Variables are not declared. Instead, when running a program, a variable is created the first time it is assigned a value.

Consider the following Pyva program. Here, the function `main` contains a local function `foo(y,z)`, which in turn contains two local functions `bar(s)` and `baz(t)`.

A function can access not just local variables and parameters, but also variables and parameters in outer functions. For example, the code in `foo` accesses the variable `x` that is a local variable in the `main` function.

```
def main() {
  def foo(y, z) {
    def bar(s) {
      return s + baz(y);
    }
    def baz(t) {
      return t+x+z;
    }
    return x + bar(y);
  }
  x = 1;
  print(foo(2, 3));
}
```

In problems 3 b) and 3 c), you will use reference attribute grammars. Note that you may neither use `instanceof` nor the `getParent()` method in your solutions.

a) Construct an *abstract* grammar for Pyva, using the JastAdd notation. Use the names `Program` for the root, `FuncDecl` for a function declaration, and `FuncCall` for a function call. You may assume that all local function declarations occur before the statements in a function. Your grammar should cover the example above and similar programs.

(5p)

b) When a local variable is created, it is stored in the current frame. To access variables in outer frames (corresponding to outer functions), the code generator or interpreter needs to know how many levels out the variable is. Similarly, for function calls, the code generator needs to know how many levels out the function definition is located. This is needed to compute the *static link* for the called function (i.e., the pointer to the frame of its enclosing function, and which is used when accessing non-local variables).

For this purpose, an `int` attribute `levelsOut` for `FuncCall` will be used. The value should be 0 if the definition of the called function is located zero levels out, i.e., it is in the same function as the function call, like the call to `foo` in `main`. The value should be 1 if the called function is defined one level out from the function

containing the call, like the call to `baz` in `bar`. If the called function is defined two levels out, the value should be 2, and so on. For example, if there had been a call to `foo` inside `bar`, the value of `levelsOut` would be 2 for that call.

Define the attribute `levelsOut` for `FuncCall`. You may assume that name analysis is already done, so that each `FuncCall` has an attribute `FuncDecl decl` that points to the definition of the called function (or `null` if the function is not defined). You can use the value `levelsOut = -1` for calls to undefined functions.

Hint! It may be useful to define an `int` attribute `level` for function declarations that is one more than that of its enclosing function. For example, the outermost functions (like `main`) would have `level=1`, their local functions (like `foo`) would have `level=2`, and so on.

(5p)

c) Define the attribute

```
syn FuncDecl FuncCall.decl();
```

that binds each call to its corresponding function, and that is `null` if no such declaration exists.

(5p)

4 Runtime systems

A compiler for Pyva generates unoptimized code. Arguments are passed on the stack. The static link is implemented as an implicit first argument. The return value is passed in the `rax` register.

a) Draw the situation on the stack for the Pyva program in problem 3 when the execution is inside the `baz` function.

Your drawing should show stack frames, stack pointer, frame pointer, dynamic links, local variables, arguments, and the static links. Include the actual values for arguments and local variables that are known at that point in execution, and mark which frame is which.

(8p)

b) Starting with the frame pointer, explain in English what steps are taken when the function `baz` accesses the variable `x`. (2p)