Examination in Compilers, EDAN65

Department of Computer Science, Lund University

2019–04–24, 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 your *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*

You may also use a dictionary from English to your native language.

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

[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:

```
IF = "if"
IFF = "iff"
ID = [a-z]+
```

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

a) Draw three small DFAs, one for each of the token definitions. Mark the final state of each DFA with the token in question. (4p)

b) Construct a DFA that combines the three small DFA's by joining their start states and eliminating nondeterminism. Mark each final state by the appropriate token. (5p)

c) Give an example of a string where the longest match rule is needed. Suppose we didn't use the longest match rule. What ambiguity would then arise when scanning this string? Explain how longest match resolves the ambiguity. (3p)

d) Give an example of a string where rule priority is needed. Suppose we didn't use rule priority. What ambiguity would then arise when scanning this string? Explain how rule priority resolves the ambiguity. (3p)

2 Context-Free Grammars

Consider the following context-free grammar for statements:

p_0 : `start` → `stmt EOF`
p_1 : `stmt` → `"{" stmt "}"`
p_2 : `stmt` → `stmt stmt`
p_3 : `stmt` → `ID "=" NUM ";"`
p_4 : `stmt` → ϵ

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. (5p)

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

(5p)

c) Transform the grammar to an equivalent grammar that is LL(1) and on canonical form.

(5p)

d) Construct the LL(1) table for the grammar you constructed in 2 c).

(If you didn't manage to solve 2 c), you can instead construct the LL(1) table for the original grammar.)

(5p)

3 Program analysis

Consider the following class declarations in a Java-like object-oriented language. (The example has a static-semantic error, because the `D` class is missing.)

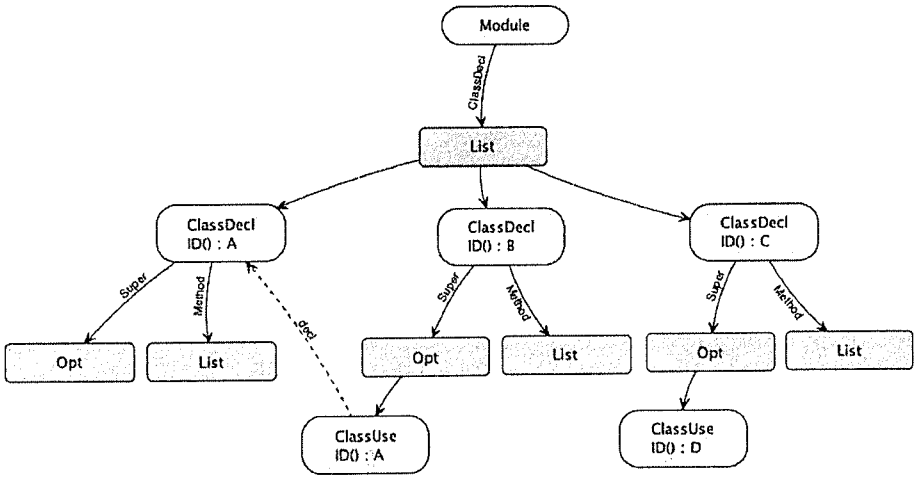
```
class A { ... }
class B extends A { ... }
class C extends D { ... }
```

The following abstract grammar defines part of this language:

```
Module ::= ClassDecl*;
ClassDecl ::= <ID> [Super:ClassUse] MethodDecl*;
ClassUse ::= <ID>;
MethodDecl ::= ...
```

Solve the problems below using reference attribute grammars. Note that you may neither use `instanceof` nor the `getParent()` method.

a) Define an attribute `decl()` for `ClassUse`. The attribute should refer to the referenced `ClassDecl` if there is one, and should otherwise be `null`. This is illustrated in the figure below. Note that the value of `decl` in the use of `D` is not shown, because it is `null`.



(5p)

b) Define an attribute `isCyclic()` for `ClassDecl`, that is true if the class has itself on its superclass chain, and false otherwise.

For example, if we have two classes

```
A class B { ... }
B class A { ... }
```

then `isCyclic` would be `true` for both classes.

Hint! Use a circular attribute.

(5p)

c) Define a collection attribute `subclasses()` that contains a set of references to the immediate subclasses of a class. You may use the standard Java class `HashSet` to represent the sets.

(5p)

Turn page for problem 4

4 Code generation and run-time systems

Consider the following C function:

```
int f(int n, int a, int b) {
    int y;
    if (n>0) y = 3;
    else y = b;
    return a*y;
}
```

A C compiler can use an optimization called *constant propagation* to replace the call `f(1, 4, x)` by the value 12.

a) Suppose that `f` is not a C function, but a method in a Java class `A`:

```
class A {
  int f(int n, int a, int b) {
      ...
  }
}
```

Explain why it is difficult for a Java compiler to apply constant propagation for the call `a.f(1, 4, x)` in the method `m` below.

```
static int m(A a, int x) {
    int result;
    result = a.f(1, 4, x); // Can the call be optimized?
    return result;
}
```

(5p)

b) Consider the following main program that calls `m`.

```
static void main() {
    A a = new A();
    int r = m(a, 7);
    print(r);
}
```

Draw the situation on stack and heap right before the method `f` returns (assuming no optimizations are applied). Local variables and arguments should be stored on the stack, but the return value is assumed to be returned in a register. Your drawing should show the frame pointer, dynamic links, local variables, arguments, static link ("this pointer"), and return addresses (where this follows from the code above).
(5p)