Liberal Syntax

Compiler Construction '16 Final Report

Ireneu Pla   Gabriel Luthier

EPFL

{firtname.lastname}@epfl.ch

1. Introduction

Describe in a few words what you did in the first part of the compiler project (the non-optional labs), and briefly say what problem you want to solve with your extension.

This section should convince us that you have a clear picture of the general architecture of your compiler and that you understand how your extension fits in it.

2. Examples

2.1 Semicolon inference

This extension allow us to write code without any semicolon at the end of the lines. We can still specify it if wanted. It is compulsory if we want to write multiple statements in one line.

```
object SemiColonInference {
    def main() : Unit = {
        println(new Tester().test())
    }
}

class Tester {
    def test() : Int = {
                        var a : Int
                        var b : Int

                        a = 1; b = 2

                        a =
                                a
                                + b
                                * 4

                        return a
    }
}
```

2.2 Methods as infix operators

This extesion allow us to call methods not only in prefix form but also as infix operators. It is only allowed to use this form when methods have one argument.

```
object InfixMethods {
    def main() : Unit = {
        println(new Tester().test())
    }
}

class Tester {
    def test() : Int = {
                var myObject : Object
                myObject = new Object().init()

                return myObject infix_method 6
    }
}

class Object {
        var value : Int

        def init() : Object = {
                value = 14
                return this
        }

        def infix_method(a: Int) : Int = {
                return value + a
        }
}
```

2.3 Expressions as statements

This extension allow to use expressions also when we previously expected a statement.

2.4 Operators as methods

This extension allow us to name methods with operators, such as ||, &&, <, ==, +, −, ∗, / or !.

```
object OperatorsAsMethods {
    def main() : Unit = {
        println(new Tester().test())
    }
}

class Tester {
    def test() : Int = {
                var a: ArbitraryObject
                var b: ArbitraryObject

                a = new ArbitraryObject().init()
                b = new ArbitraryObject().init()

                return a.+(b)
    }
}

class ArbitraryObject {
        def init() : ArbitraryObject = {
                return this;
        }

        def +(b: ArbitraryObject): Int = {
                return 100
        }
}
```

3. Implementation

This is a very important section, you explain to us how you made it work.

3.1 Theoretical Background

If you are using theoretical concepts, explain them first in this subsection. Even if they come from the course (eg. lattices), try to explain the essential points *in your own words*. Cite any reference work you used like this [Appel 2002]. This should convince us that you know the theory behind what you coded.

3.2 Implementation Details

Describe all non-obvious tricks you used. Tell us what you thought was hard and why. If it took you time to figure out the solution to a problem, it probably means it wasn't easy and you should definitely describe the solution in details here. If you used what you think is a cool algorithm for some problem, tell us. Do not however spend time describing trivial things (we what a tree traversal is, for instance).

After reading this section, we should be convinced that you knew what you were doing when you wrote your extension, and that you put some extra consideration for the harder parts.

4. Possible Extensions

If you did not finish what you had planned, explain here what's missing.

In any case, describe how you could further extend your compiler in the direction you chose. This section should convince us that you understand the challenges of writing a good compiler for high-level programming languages.

References

A. W. Appel. *Modern Compiler Implementation in Java*. Cambridge University Press, 2nd edition, 2002.