Mina 1.0

A Hybrid-Partial Evaluator for Scala

http://www.amanj.me/projects/mina

Amanj Sherwany

http://www.amanj.me

Draft of 28th May 2013

Contents

1 Introduction

1.1 What is Mina?

Mina is a Partial Evaluator for Scala. It is a port of Civet for Java, which is an implementation of Hybrid-Partial Evaluator paper (`http://www.cs.utexas.edu/~wcook/Civet/`). "Hybrid partial evaluation (HPE) is a pragmatic approach to partial evaluation that borrows ideas from both online and offline partial evaluation. HPE performs offline-style specialization using an online approach without static binding time analysis. The goal of HPE is to provide a practical and predictable level of optimization for programmers, with an implementation strategy that fits well within existing compilers or interpreters. HPE requires the programmer to specify where partial evaluation should be applied. It provides no termination guarantee and reports errors in situations that violate simple binding time rules, or have incorrect use of side effects in compile-time code."[1]

Mina is a Scala compiler plugin, runs after the **patmat** phase, and partially evaluates Scala source code to a more efficient code, which has exactly the same meaning as the original one.

The plugin name is mina and it is read like */mi:na:/*, which is a Kurdish name of Forget-me-not flower `http://en.wikipedia.org/wiki/Forget-me-not`.

1.2 Partial Evaluation

"In computing, partial evaluation is a technique for several different types of program optimization by specialization. The most straightforward application is to produce new programs which run faster than the originals while being guaranteed to behave in the same way.

A computer program, *prog*, is seen as a mapping of input data into output data:

$$prog : I_{static} \times I_{dynamic} \to O$$

I_{static} the static data, is the part of the input data known at compile time. The partial evaluator transforms $< prog, I_{static} >$ into $prog^* : I_{dynamic} \to O$ by precompiling all static input at compile time. $prog^*$ is called the "residual program" and should run more efficiently than the original program. The act of partial evaluation is said to "residualize" *prog* to $prog^*$."[2]

2 How to use Mina

2.1 Compile-time(`CT`) vs. Runtmie(`RT`)

With Mina you can mark a block of code to be fully evaluated at compile time. If a variable contains a compile time (`CT`) code, then the variable itself is a `CT` variable and will not be present at runtime. Here is a simple of using `CT`:

[1] from Civet `http://www.cs.utexas.edu/~wcook/Civet/`
[2] from Wikipedia `http://en.wikipedia.org/wiki/Partial_evaluation`

```
/**
 * 2 is evaluated at compile time,
 * therefore x is a compile-time variable,
 * i.e. it won't be present at runtime
 */
val x = CT(2)

/**
 * y is a result of adding 5 to x,
 * but x is already a compile-time
 * variable, which means the result
 * of x + 5 will be a compile-time
 * value, which means y is also a
 * compile-time variable
 */
val y = x + 5

/**
 * since y is a compile-time variable,
 * this statement will be evaluated to
 * println(7)
 */
println(y)

/**
 * The above three statements will be
 * evaluated to one statement only:
 *
 * println(7)
 */
```

One can also mark a block of code to be not evaluated at all, simply by surrounding the block with `RT` (...), for example:

```
/**
 * x won't be evaluated, since the
 * right-hand side expression
 * is marked with RT
 */
val x = RT(2)

/**
 * y won't be evaluated too,
 * since x is RT
 */
val y = x + 5

/**
 * The following statement won't be
 * evaluated too, since y is RT
```

```
  */
println(y)
```

Now you should know that you can either tag an (expression/statement/block of statements) is `CT` or `RT`. `CT` makes sure that everything will be fully evaluated at compile time, while `RT` makes sure that everything will be evaluated at runtime. In other words, no `CT` inside `RT` is allowed, and no `RT` inside `CT` is allowed.

But if you do not tag an (expression/statement/block of statements) with neither `CT` nor `RT`, then it will be partially evaluated at compilation-time helping the program to speed-up. In the next sub-sections we will talk about this in detail.

3 Credits