A type inferred programming language

Scala Lite

August 12, 2016

Table of Contents

1 Introduction

Table of Contents

1 Introduction

2 ScalaLite

Table of Contents

1 Introduction

2 ScalaLite

3 Benchmarking

Table of Contents

1. Introduction
2. ScalaLite
3. Benchmarking
4. Implementation

Table of Contents

1. Introduction
2. ScalaLite
3. Benchmarking
4. Implementation
5. Misc

Table of Contents

1. Introduction
2. ScalaLite
3. Benchmarking
4. Implementation
5. Misc

Introduction

Scala−− is a prototype towards to be a full-fledged production-ready functional programming language presently, only support of a small subset of Scala language functionalities. However, it is way faster than JVM-based Scala for both compilation startup time and execution time of the target at runtime leveraging LLVM optimization/analyse. The prototype compiler translates Scala-like source code LLVM IR with OCaml implementation.

Table of Contents

1 Introduction

2 ScalaLite

3 Benchmarking

4 Implementation

5 Misc

Language Features

Language Features

- Machine code ← Assembly ← LLVM-IR ← OCaml-LLVM-binding← OCamlYacc ← OCamlLex
- Basic control flow and scoped varible declaration
- Basic arithmetic
- Similar to Scala syntax defining functions and variables

Issus:

1. TL;DR
2. Functional ?
3. Type inference ...
4. OOP ?

Table of Contents

1 Introduction

2 ScalaLite

3 Benchmarking

4 Implementation

5 Misc

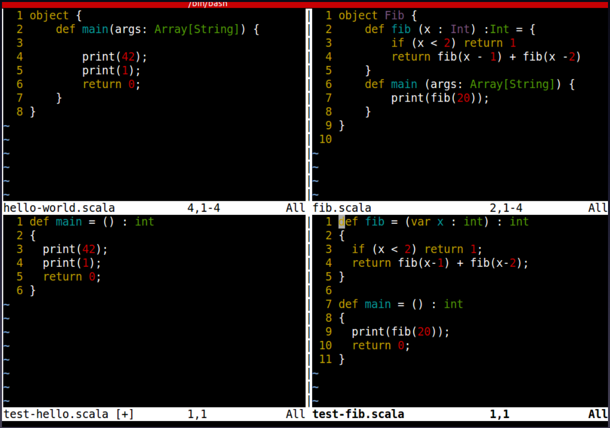
Bencharkmarking

Experiments

- ARM
  - ArchLinux
  - FreeBSD
- amd64
  - Archlinux
  - Ubuntu
  - FreeBSD
  - OS X

Benchmarking

Results



/bin/bash

```
object {
    def main(args: Array[String]) {

        print(42);
        print(1);
        return 0;
    }
}
~
~
~
~
~
~
hello-world.scala              4,1-4           All
```

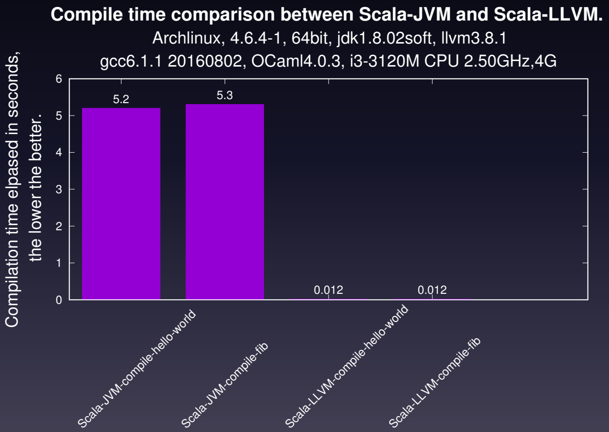
```
object Fib {
    def fib (x : Int) :Int = {
        if (x < 2) return 1
        return fib(x - 1) + fib(x -2)
    }
    def main (args: Array[String]) {
        print(fib(20));
    }
}

~
~
~
~
fib.scala                      2,1-4           All
```

```
def main = () : int
{
  print(42);
  print(1);
  return 0;
}
~
~
~
~
~
~
~
~
test-hello.scala [+]           1,1             All
```

```
def fib = (var x : int) : int
{
  if (x < 2) return 1;
  return fib(x-1) + fib(x-2);
}

def main = () : int
{
  print(fib(20));
  return 0;
}
~
~
~
test-fib.scala                 1,1             All
```

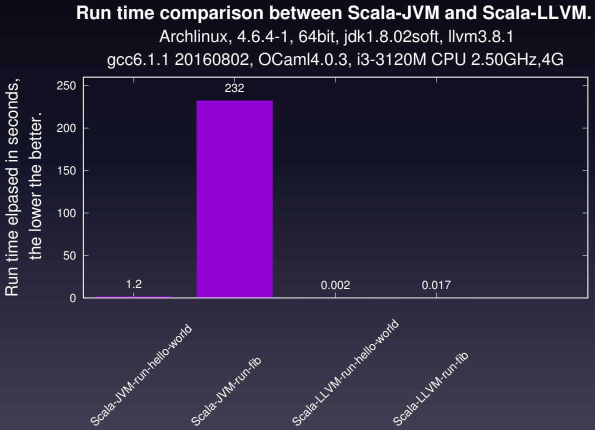
Bencharkmarking

Compile time



Bencharkmarking

Run time



Run time comparison between Scala-JVM and Scala-LLVM.
Archlinux, 4.6.4-1, 64bit, jdk1.8.02soft, llvm3.8.1
gcc6.1.1 20160802, OCaml4.0.3, i3-3120M CPU 2.50GHz,4G

Table of Contents

1. Introduction
2. ScalaLite
3. Benchmarking
4. Implementation
5. Misc

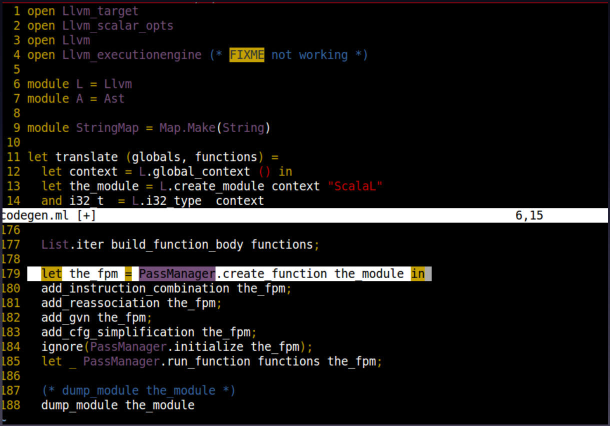
Implementation

	Methods	**Compiler phases**	**Data flow**
Scala-- Compiler front end	ocamllex	Scanner	Token
	ocamlyacc	Parser	Ast
	OCaml	Semantic checker	SAst
	OCaml Hindley-Milner	Type inferrer	TAst
	OCaml LLVM binding	Code generator	LLVM-IR
Scala-- Compiler back end	LLVM toolchain	llc compiler	Assembly
	platform-dependent gcc compiler	assembler	Machaine executable

Table of Contents

1. Introduction
2. ScalaLite
3. Benchmarking
4. Implementation
5. Misc

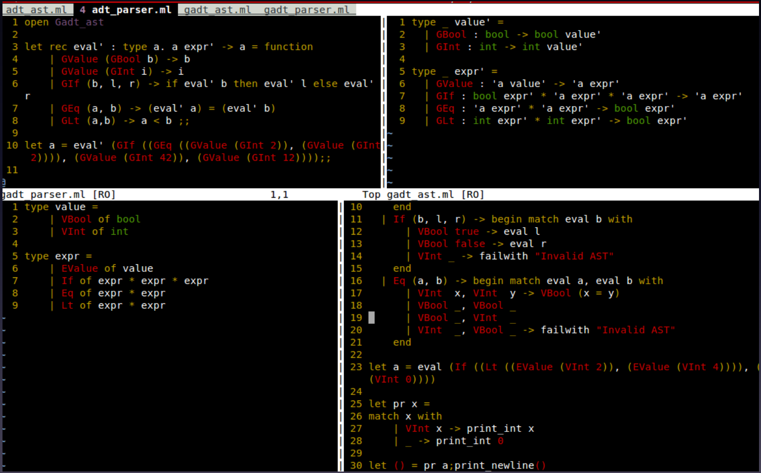
Attempt of Harness advantage of LLVM's optimization power



```
open Llvm_target
open Llvm_scalar_opts
open Llvm
open Llvm_executionengine (* FIXME not working *)

module L = Llvm
module A = Ast

module StringMap = Map.Make(String)

let translate (globals, functions) =
  let context = L.global_context () in
  let the_module = L.create_module context "ScalaL"
  and i32_t  = L.i32_type  context
codegen.ml [+]                                                    6,15

  List.iter build_function_body functions;

  let the_fpm = PassManager.create_function the_module in
  add_instruction_combination the_fpm;
  add_reassociation the_fpm;
  add_gvn the_fpm;
  add_cfg_simplification the_fpm;
  ignore(PassManager.initialize the_fpm);
  let _ PassManager.run_function functions the_fpm;

  (* dump_module the_module *)
  dump_module the_module
```

Some 'other . Cool stuff

GADT



```
open Gadt_ast

let rec eval' : type a. a expr' -> a = function
    | GValue (GBool b) -> b
    | GValue (GInt i) -> i
    | GIf (b, l, r) -> if eval' b then eval' l else eval' r
    | GEq (a, b) -> (eval' a) = (eval' b)
    | GLt (a,b) -> a < b ;;

let a = eval' (GIf ((GEq ((GValue (GInt 2)), (GValue (GInt 2)))), (GValue (GInt 42)), (GValue (GInt 12))));;

```

```
type _ value' =
  | GBool : bool -> bool value'
  | GInt : int -> int value'

type _ expr' =
  | GValue : 'a value' -> 'a expr'
  | GIf : bool expr' * 'a expr' * 'a expr' -> 'a expr'
  | GEq : 'a expr' * 'a expr' -> bool expr'
  | GLt : int expr' * int expr' -> bool expr'
```

```
type value =
    | VBool of bool
    | VInt of int

type expr =
    | EValue of value
    | If of expr * expr * expr
    | Eq of expr * expr
    | Lt of expr * expr
```

```
    end
  | If (b, l, r) -> begin match eval b with
      | VBool true -> eval l
      | VBool false -> eval r
      | VInt _ -> failwith "Invalid AST"
    end
  | Eq (a, b) -> begin match eval a, eval b with
      | VInt  x, VInt  y -> VBool (x = y)
      | VBool _, VBool _
      | VBool _, VInt  _
      | VInt  _, VBool _ -> failwith "Invalid AST"
    end

let a = eval (If ((Lt ((EValue (VInt 2)), (EValue (VInt 4)))), (
(VInt 0))))

let pr x =
match x with
    | VInt x -> print_int x
    | _ -> print_int 0

let () = pr a;print_newline()
```