A type inferred programming language

Scala Lite

August 12, 2016
1 Introduction
Table of Contents

1 Introduction

2 ScalaLite
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>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
</tr>
<tr>
<td>2 ScalaLite</td>
</tr>
<tr>
<td>3 Benchmarking</td>
</tr>
<tr>
<td>4 Implementation</td>
</tr>
<tr>
<td>5 Misc</td>
</tr>
</tbody>
</table>
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

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

Issues:

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

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

Results
Benchmarking

Compile time

Compile 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

Compilation time elapsed in seconds, the lower the better.
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>
<thead>
<tr>
<th>Benchmark</th>
<th>Scala-JVM run <code>hello-world</code></th>
<th>Scala-JVM run <code>fib</code></th>
<th>Scala-LLVM run <code>hello-world</code></th>
<th>Scala-LLVM run <code>fib</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Run time (s)</td>
<td>1.2</td>
<td>0.002</td>
<td>0.017</td>
<td>232</td>
</tr>
</tbody>
</table>
Table of Contents

1 Introduction
2 ScalaLite
3 Benchmarking
4 Implementation
5 Misc
Implementation

ScalaLite Benchmarking Implementation Misc

Implementation

Methods

ocamllex
ocamlyacc
OCaml
OCaml Hindley-Milner
OCaml LLVM binding

Compiler phases

Scanner
Parser
Semantic checker
Type inferrer
Code generator
llc compiler
assembler

Data flow

Token
Ast
S Ast
T Ast
LLVM-IR

Scala-- Compiler front end

Scala-- Compiler back end

LLVM toolchain
platform-dependent gcc compiler

Assembler
Machaine executable
Attempt of Harness advantage of LLVM’s optimization power

```scala
1 open Llvm_target
2 open Llvm_scalar_opts
3 open Llvm
4 open Llvm_executionengine (**FIXME not working**)  
5
6 module L = Llvm
7 module A = Ast
8
9 module StringMap = Map.Make(String)
10
11 let translate (globals, functions) =
12  let context = L.global_context () in
13  let the_module = L.create_module context "ScalaL"
14  and i32 t = L.i32 type context

176 List.iter build_function_body functions;
177
178 let the_fpm = PassManager.create function the_module in
179  add_instruction combination the_fpm;
180  add reassociation the_fpm;
181  add gvn the_fpm;
182  add cfg simplification the_fpm;
183  ignore(PassManager.initialize the_fpm);
184  let _ PassManager.run function functions the_fpm;
185  (* dump module the module *)
186  dump_module the_module
```
Some 'other . Cool stuff

GADT