Intermediate Representation Trees

Dr. Mattox Beckman

Illinois Institute of Technology
Department of Computer Science

Table of Contents

Introduction
Objectives

IR Trees
Definition

Implementation
Getting Started

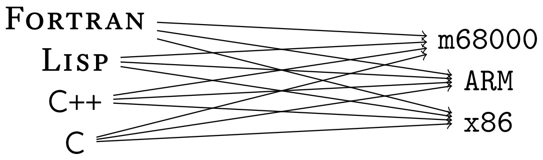
Intermediate Representation Trees

Objectives

- What is an IR tree?
- Why do we need them?
- What do we do with them?

The Problem

- Problem of complexity
  - The initial language we want to compile is huge and complex.
  - The machine code we want to generate is a simple language.
- Problem of Number
  - We have many languages we would like to compile.
  - We have many CPUs we would like to target.



The Solution

- Introduce an intermediate tree.
  - More complex than assembly, easy to translate *to*.
  - Simpler than Tiger, easy to translate *from*.
  - We can also *reuse different back-ends*.

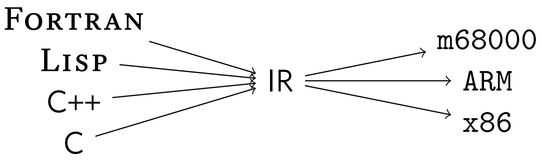


Table of Contents

Introduction
    Objectives

IR Trees
    Definition

Implementation
    Getting Started

Expressions

- We have a type for *expressions:*
  - Constants (integer)
  - Names (a label)
  - Temp (a temporary variable)
  - Binop (a binary operation and two expressions)
  - Mem (an expression)
  - Call (an expression and a list of expressions)
  - ESeq (a statement and an expression)
- The label and variable types could be just strings. But you might want something more complex.

Statements

- We have a type for *statements:*
  - Move (two expressions)
  - Exp (an expression)
  - Jump (an expression and a set of labels)
  - CJump (a relation, two expressions, and a set of labels.)
  - Seq (a list of statements)
  - Label (a label)

Binary Operations

- Plus
- Minus
- Mul
- Div
- And
- Or
- LShift
- RShift
- ARShift
- XOR

In CLOJURE...

```
    {:binop :xor}
```

Relational Operations

- Eq, Ne
- Lt, Gt
- Le, Ge
- ULt, ULe, UGt, Uge

In CLOJURE...

```
    {:relop :ult}
```

Table of Contents

Introduction
Objectives

IR Trees
Definition

Implementation
Getting Started

Time to break ground

- We will create a new CLOJURE project.
- `lein new app tiger`
- To start with IR Trees: create the file `src/tiger/ir.clj`.
- Namespace: `tiger.ir`.
- You can use `core.typed` if you want. I highly recommend it.

Using core.typed

- In your project.clj add the dependency: [org.clojure/core.typed "0.2.77"]
- In your namespace require: (:require [clojure.core.typed :as t])

```
(t/defalias exp (t/U '{:exp ':const .... } ... ))
```

- The expression and statement types are mutually recursive: we may have to fudge this. More details later.

Your Work

- Decide on a representation for each of these types.
- Write Clojure functions that construct these for us.

```
tiger.ir> (seq [(move (temp "t1") (temp "t2")),
                (jump (name "L1") #{"L1"})])
; => {:stm :seq
      :stms [{:stm :move
              :e1   {:exp :temp, :name "t1"}
              :e2   {:exp :temp, :name "t2"}},
            {:stm  :jump
             :dest {:exp :name, :name "L1"}
             :targets #{"L1"}}]}
```

- You will also write a function canonicalize. I'll explain this next time.