Check-In

Review – Runtime Environments

Give an example of a language and its runtime environment

Announcements

Administrivia

University of Kansas | Drew Davidson

EECS 665 COMPILER CONSTRUCTION

Intermediate Representations

Last Time

Lecture Review – Runtime Environments

Runtimes

- Runtime Environments
- Hardware Intuition

<u>You Should Know</u>

- What a runtime environment is
- Basic notions of how we might execute programs
    - OS mediation
    - Virtual machine
    - Accessing memory / registers

Runtimes


COMPILER
LAND
Code Generation
Optimization
Architecture
Intermediate Code
Runtimes
SDD
Parsing
Semantics
Lexical Analysis
Syntactic Definiton
MOLASSES SWAMP
GLOPPY
KING KANDY
CANDY CASTLE
QUEEN FROSTINE
GRAMMA NUTT
PRINCESS LOLLY
LORD LICORICE
LICORICE CASTLE
JOLLY
PLUMPY

Today’s Outline

Lecture Outline – Intermediate Representations

Introduce IRs

- What they are
- How they’re used

Three Address Code (3AC)

- Introduction
- Inventory

Intermediate Representations

Compiler Construction

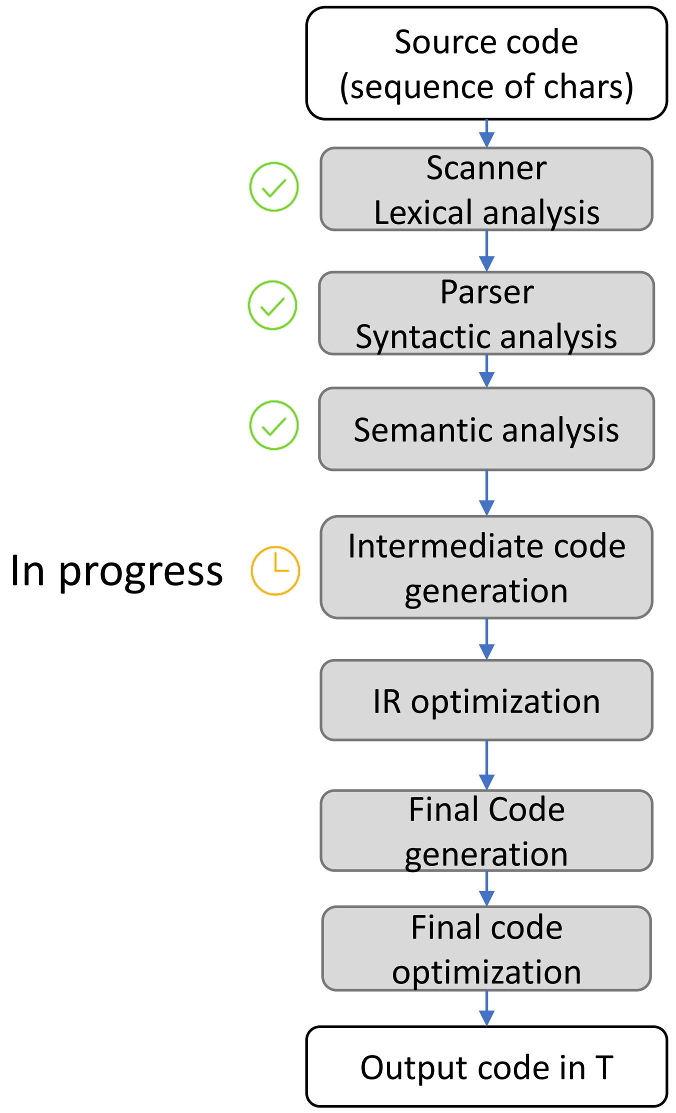
Progress Pics

Done:

- Derived an AST, augmented with types and identifier symbols
- Ensured the program is legal to the best of our abilities

ToDo:

- Get that sucker to run!



IRs: The Big Idea

Intermediate Representations

A big, basic concept

- “Encoding of a program”
- “The output of a compiler frontend and input of a compiler backend”
- “What a compiler knows about a program”
- “A simpler language to which the source language is mapped”

Intermediate Representation Benefits

Introducing IRs

M source languages

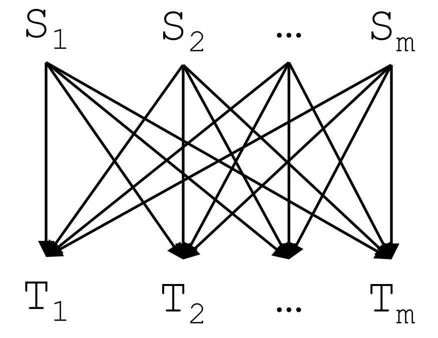
N target languages

Abstraction

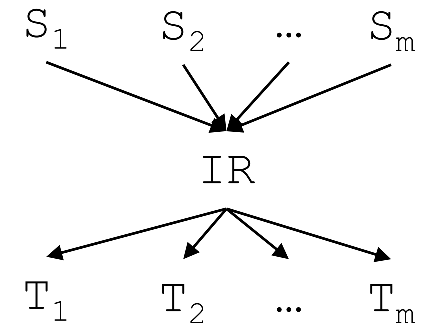
- Decouple compiler frontend from the backend

Analysis

Write M×N compilers



or M+N compiler modules



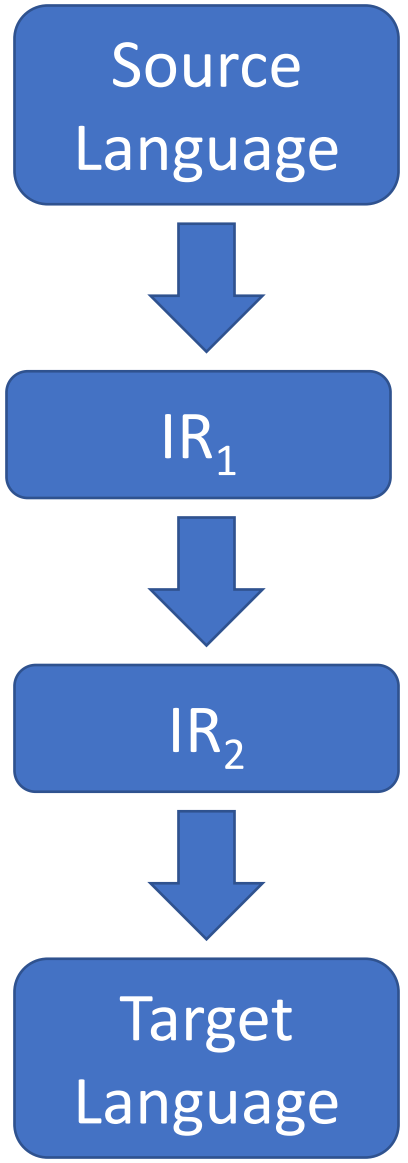
Intermediate Representation Benefits

Introducing IRs

Abstraction

- Decouple compiler frontend from the backend
- Break down source language constructs over several small steps towards target

Analysis



Intermediate Representation Benefits

Abstraction

- Decouple compiler frontend from the backend
- Break down source language constructs over several small steps towards target

Analysis

- Optimize programs

Improve…

- Runtime
- Memory usage
- Power usage
- Security

Intermediate Representation Benefits

Introducing IRs

Abstraction

- Decouple compiler frontend from the backend
- Break down source language constructs over several small steps towards target

Analysis

- Optimize programs

- Predict faults

For example…

- typechecking

But isn't this an analysis on the AST?

Intermediate Representation Benefits

Introducing IRs

Abstraction

- Decoup
  from th
- Break d
  constru
  steps to

Analysis

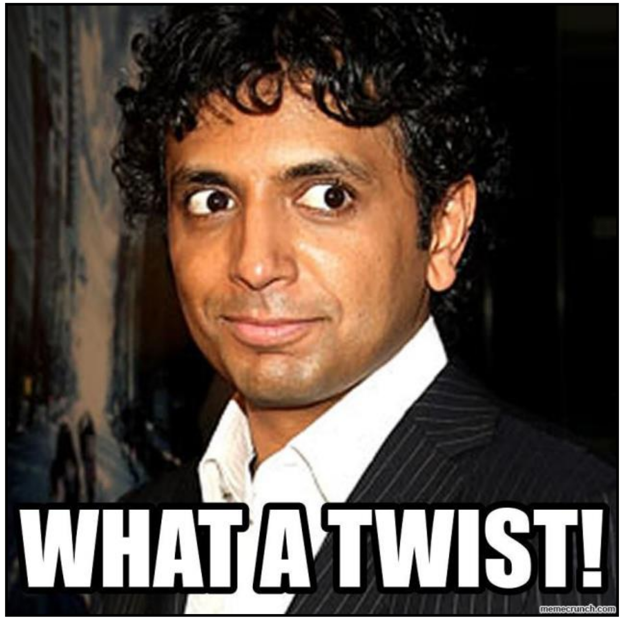
- Optimi

- Predict

For example

ST?

ASTs are an example of an IR!!



M. Night Shyamalan, famous for (ill-considered) plot twists in movies he writes/directs

Classes of IR

Structural

- Abstract-Syntax Tree (AST)
- Abstract Syntax DAG

Linear

- Three-Address Code (3AC)
- Stack machine code

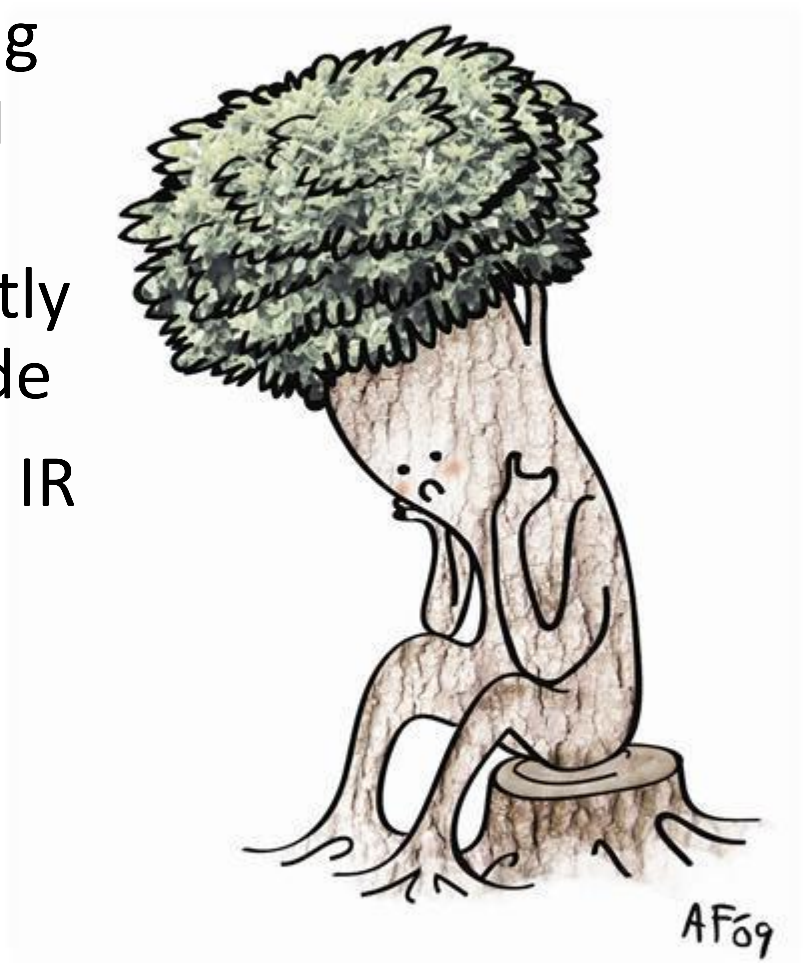
Hybrid

- Control-Flow Graph

Limitations of Trees

Introducing IRs

- AST is great for some things, but not everything
  - Doesn't represent control flow very well
- Compilers *could* go directly from AST to machine code
- Let's consider a different IR



Today's Outline

Lecture Outline – Intermediate Representations

Introduce IRs

- What they are
- How they're used

Three Address Code (3AC)

- Introduction
- Inventory

Intermediate Representations

Introducing 3AC

3AC Description

A Simplified Instruction Set Architecture (ISA)

- A family of pseudocode notations

Like ASTs, there's no canonical 3AC

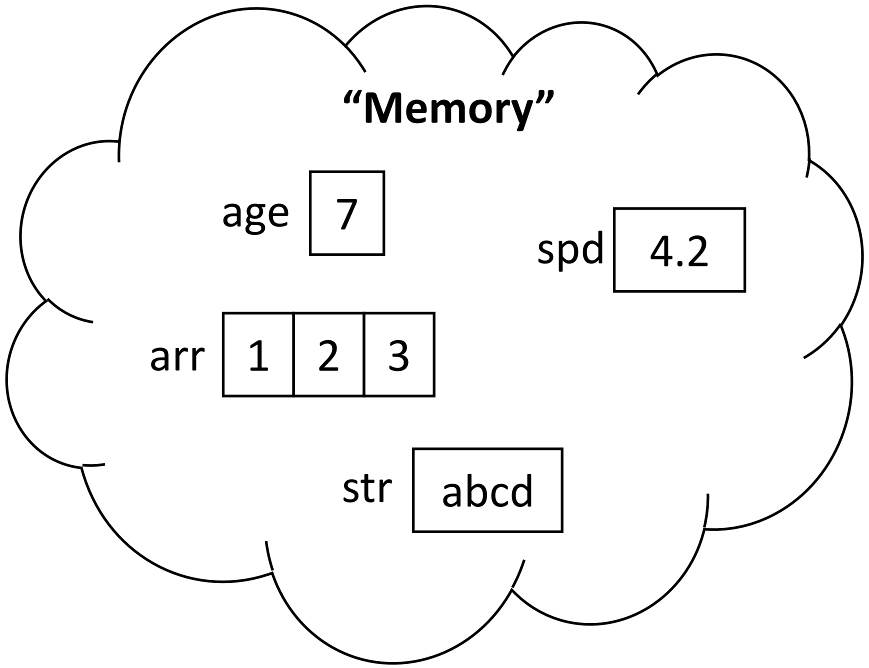
We're more interested in the general idea

Introducing 3AC

3AC Description

A Simplified Instruction Set Architecture (ISA)

- A family of pseudocode notations
- Memory model: infinite “symbolic store”



- Naming a variable adds a location in the store
- We’ll assume that the store can handle scope

Introducing 3AC

3AC Description

A Simplified Instruction Set Architecture (ISA)

- A family of pseudocode notations
- Memory model: infinite "symbolic store"
- Instruction model: linear instructions divided into *procedures*

Discrete code listings

From Variables to Locations (“locs”)

3AC Description

A loc is…

- An address in memory
- A container for a value

Use [ ] around loc to denote value at that location

- [a] is the “value at a”

(sort of like adding a pointer level into every access)

```
int a;
int * b;
a = 1;
*b = 2;
```

```
int * a;
int ** b;
*a = 1;
**b = 2;
```

3AC: What’s With the Name?

3AC Description

Instructions have *at most 3* operands

```
a = b + c * d - e
```

becomes

```
[tmp1] := [c] * [d];
[tmp2] := [b] + [tmp1];
[tmp3] := [tmp2] - [e];
[a] := [tmp3];
```

3AC: Instruction Classes

3AC Description

Data flow

- Assignment
- Math/Logic

Control flow

- Labels
- Jumps

Interprocedural

- Boundaries
- Bodies
- Calls

<opd> := <opd>

Opd stands for “operand”
Literals, variables, etc.

Example:

```
[a]  := 1
[b]  := [a]
```

3AC: Instruction Classes

3AC Description

Data flow

- Assignment
- Math/Logic

Control flow

- Labels
- Jumps

Interprocedural

- Boundaries
- Bodies
- Calls

<opd> := <opd> <opr> <opd>

<opd> := <opr> <opd>

Opd stands for “operand”
Literals, variables, etc.

Opr stands for “operator”
MULT64, DIV64, SUB64, ADD64, etc.

Example:

```
[a] := 1 MULT64 2
[b] := [a] SUB64 4
```

3AC: Instruction Classes

3AC Description

Data flow

- Assignment
- Math/Logic

Control flow

- Labels
- Jumps

Interprocedural

- Boundaries
- Bodies
- Calls

<lbl>: <instr>

Example:

```
Label1: [a] := 1
```

nop

Example:

```
Label1: nop
```

3AC: Instruction Classes

3AC Description

Data flow

- Assignment
- Math/Logic

Control flow

- Labels
- Jumps

Interprocedural

- Boundaries
- Bodies
- Calls

goto <lbl>

Example:

```
Label2: goto Label2
```

ifz <opd> goto <lbl>

Example:

```
        ifz [a] goto Label1
        [a] := 1
Label1: [a] := 2
```

3AC: Instruction Classes

3AC Description

Data flow

- Assignment
- Math/Logic

Control flow

- Labels
- Jumps

Interprocedural

- Boundaries
- Bodies
- Calls

enter <proc>

leave <proc>

Example:

```
enter fn
[global] := 7
leave fn
```

3AC: Instruction Classes

3AC Description

Data flow

- Assignment
- Math/Logic

Control flow

- Labels
- Jumps

Interprocedural

- Boundaries
- Bodies
- Calls

getarg <idx> <opd>
setret <opd>

Example:

```
int fn(int a, int b){
  a = b;
  return 42;
}
```

Example:

```
enter fn
getarg 1, [a]
getarg 2, [b]
[a] := [b]
setret 42
leave fn
```

3AC: Instruction Classes

3AC Description

Data flow

- Assignment
- Math/Logic

Control flow

- Labels
- Jumps

Interprocedural

- Boundaries
- Bodies
- Calls

call <proc>

setarg <int> <opd>

getret <opd>

Example:

```
int fn(int a, int b){
  a = b;
  return 42;
}
int v(){
  int k;
  k = fn(7, 9);
}
```

Example:

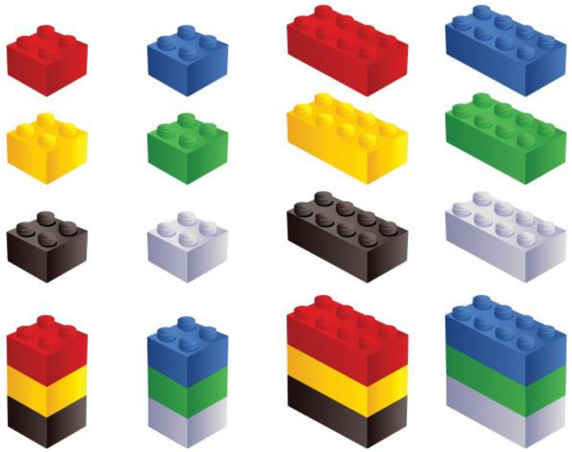
```
enter fn
getarg 1, [a]
getarg 2, [b]
[a] := [b]
setret 42
leave fn
enter v
setarg 1, 7
setarg 2, 9
call fn
getret [k]
leave v
```

That's All we Need!

3AC Description

We can build complex behavior out of these simple building blocks

- One minor loose end to tie up…



Dealing with Scope

3AC Description

```
void fn(){
  int a;
  a = 9;
  if (true){
    int a;
    a = 6;
  }
}
```

```
enter fn
[ a¹]  := 9
 …
[ a²]  := 6
 …
leave fn
```

Name clash?

Only in notation!
These assignment connect to different symbols

We can use superscripts if needed

3AC Data Structures

AST Translation to 3AC – Implementation

- One class per 3AC node type
  - Often referred to as “Quads” – has at most 4 fields (+ label)
  - Each procedure maintains a list of its quads

lbl	dst	src1	opr	src2
L1	[t1]	[a]	SUB 64	2

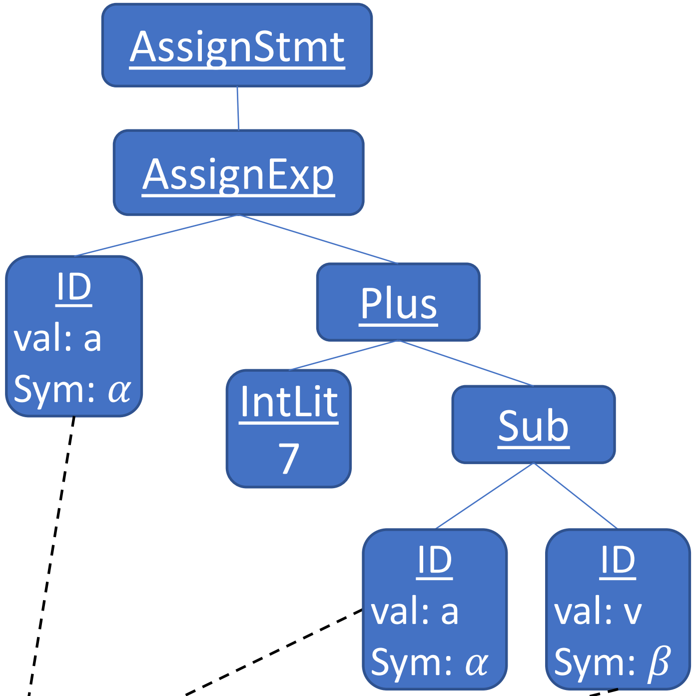
Translation Implementation

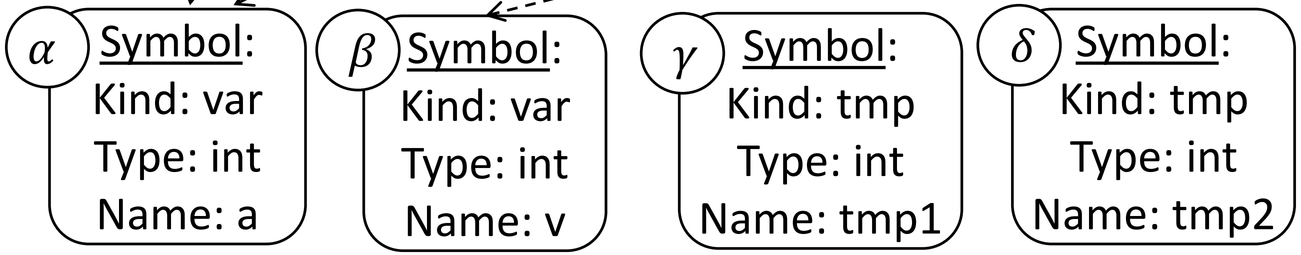
AST Translation to 3AC – Implementation

Propagate context to parent & generate 3AC instruction(s)

```
a = 7 + (a − v)
```

```
[t1] := [a] SUB64 [v]
[t2] := 7 ADD64 [t1]
 [a] := [t2]
```





lbl	dst	src_1	opr	src_2
	t1 (γ)	a (α)	SUB 64	v (β)
	t2 (δ)	7	ADD 64	[t1] (γ)
	[a] (α)	[t2] (δ)		

Next Time
3AC Translation

Translating AST code into 3AC

- A final walk of the AST