CSE 504: Compiler Design

Intermediate Representations
Graphical and Linear IRs

Pradipta De
pradipta.de@sunykorea.ac.kr
Current Topic

- Intermediate Representations
  - Graphical IRs
  - Linear IRs
  - Symbol Table
Intermediate Representations

- Compiler needs representation of all facts derived from the source program
  - Called Intermediate Representations
- During the Backend phase, the IR is the only form that is used – not the source code
- Multiple IR may be generated during the compiler passes
  - Choice of IR is dependent on source language, target machine, and properties of application that the compiler translates
Types of IR

• Graphical
  – Encodes knowledge in a graph form
    • Nodes, edges, lists or trees
    • Ex: parse tree, abstract syntax tree

• Linear
  – Pseudo-code for an abstract machine
  – Simple linear sequence of operations
    • suitable for source code to machine code translation

• Hybrid
  – Combination of graphical and linear techniques
  – Control flow graph
Examples of different IR

Given an array of 4 byte elements stored in row major order – A[1…10, 1…10]

\[
\begin{align*}
\text{subI } r_i,1 & \Rightarrow r_1 \\
\text{multI } r_1,10 & \Rightarrow r_2 \\
\text{subI } r_j,1 & \Rightarrow r_3 \\
\text{add } r_2,r_3 & \Rightarrow r_4 \\
\text{multI } r_4,4 & \Rightarrow r_5 \\
\text{loadI } @A & \Rightarrow r_6 \\
\text{add } r_5,r_6 & \Rightarrow r_7 \\
\text{load } r_7 & \Rightarrow r_{Ai,j}
\end{align*}
\]

ILOC Code
Graphical IR

• Graphical IRs uses edges and nodes, but differ in their level of abstraction
  – How relationships between graph and code is maintained
  – Structure of the graph

• Tree based:
  – Parse tree, Abstract Syntax tree (AST), Directed Acyclic Graph (DAG)

• Flow graphs
  – Control flow graph, Dependence graph, call graph
Tree based Graphical IR

- Tree based IRs encode the syntax of the code in the structure of the tree

- Parse Tree
  - A graphical representation of the derivation
  - Parse tree is large compared to source code as it represents Non terminals
    - Compiler must allocate space for each node → high storage requirement
    - Compiler must traverse each node during compilation → slower compilation

(a) Classic Expression Grammar

(b) Parse Tree for $a \times 2 + a \times 2 \times b$
Abstract Syntax Tree

• A simplified version of the Parse Tree
  – Abstract those nodes that serve no useful purpose in the compilation process

An AST is a contraction of the parse tree that eliminates most nodes for Non-terminals
Directed Acyclic Graph

- AST can be large ➔ high memory requirement
  - Merge the common subtrees in the AST

- A DAG is an AST where the common subtrees are instantiated once with multiple parents
  - A compact representation of AST by avoiding duplication

In the expression: \( a \times 2 + a \times 2 \times b \), the textually identical expression \( a \times 2 \) produce identical values ➔ use a single copy of the subtree

- Assumption: value of \( a \) cannot change between two uses of \( a \)
Graph based IR

- Trees good for representing grammatical structure of the program ... cannot capture other properties of program, like flow of variable value in a program

- More general graph variants for IR
  - Control Flow graph
  - Dependence graph
  - Call graph
Control Flow Graph

• A control flow graph (CFG) models the flow of control between basic blocks in a program
  – A CFG has a node for each basic block, and an edge representing control transfer

• Basic block is a maximal length sequence of straightline or branch-free code
Control Flow Graph

- CFG must be used in conjunction with other IR techniques that can represent the operations within a block.

- Basic blocks can be as short as "a single statement block"
  - Helps in analysis and optimization.
Dependence Graph

• A Dependence graph models the **flow of data value** from point of creation (declaration) to the point of use
  – Nodes represent operations
  – An edge connects two nodes – one that defines the value, and the other that uses the value

```
1  x ← 0
2  i ← 1
3  while (i < 100)
   4    if (a[i] > 0)
   5      then x ← x + a[i]
   6    i ← i + 1
6  print x
```
Call Graph

• Call graph represents runtime transfer of control between procedures
  – A node for each procedure
  – An edge for each distinct procedure call

• Problems of building call graphs during compilation
  – Separate compilation: small subsets of program can be compiled independently \( \rightarrow \) do not know the callee procedure during compilation
  – Procedure valued parameter: the procedure is instantiated on each call \( \rightarrow \) can call a different procedure on each invocation
  – Object oriented programs with inheritance can create ambiguous procedure calls that can be resolved with type information
Linear IRs

• Linear IP represent code being compiled as an ordered sequence of operations
  – An assembly language program is a form of linear code
  – Example: bytecode (form of one address code)

• Forms of Linear IRs:
  – One address code: models behavior of stack machines or accumulator machines
  – Two address code: models operations where one of the operands is always redefined with the result
  – Three address code: takes two operands and produces a result
Stack Machine Code

• A form of one address code
• Assumes the presence of a stack of operands
  – Operations take the operands from stack, and push the results back into stack

Stack Machine code for, a-2*b

```
push 2
push b
multiply
push a
subtract
```
Three Address Code

• Three address code has operations of the form, $i \leftarrow j \ op \ k$, where $op$ is the operator, $j$ and $k$ are the two operands, and $i$ is the result

```
t_1 \leftarrow 2
\text{Three address code for } a-2*b

\begin{align*}
t_2 & \leftarrow b \\
t_3 & \leftarrow t_1 \times t_2 \\
t_4 & \leftarrow a \\
t_5 & \leftarrow t_4 - t_3
\end{align*}
```
Data Structures for Three Address Code

• Three address codes are implemented as a set of quadruples
  – Each quadruple represented with 4 fields: an operator, two operands, and destination

Implementations of Three Address Code

(a) Simple Array

(b) Array of Pointers

(c) Linked List