Lecture 01 - Introduction

THEORY OF COMPILATION

Eran Yahav
Who?

Eran Yahav
Taub 734
Tel: 8294318
yahave@cs.technion.ac.il
Monday 13:30-14:30
http://www.cs.tecnion.ac.il/~yahave
What?

- Understand
  - what is a compiler
  - how does it work
  - techniques that can be re-used in other settings

- What will help us
  - Text books
    - Modern compiler design
    - Compilers: principles, techniques and tools
  - 5 homework assignments

- Will also help
  - Taking a deep breath
  - Focusing on material and not on your grade
What is a Compiler?

- “A compiler is a computer program that transforms source code written in a programming language (source language) into another language (target language). The most common reason for wanting to transform source code is to create an executable program.”

--Wikipedia
What is a Compiler?

source language

- C
- C++
- Pascal
- Java
- Postscript
- TeX
- Perl
- JavaScript
- Python
- Ruby
- Prolog
- Lisp
- Scheme
- ML
- OCaml

Compiler

target language

- IA32
- IA64
- SPARC
- C
- C++
- Pascal
- Java
- Java Bytecode
- ...

"I THINK YOU SHOULD BE MORE EXPLICIT HERE IN STEP TWO."

S. Harris
What is a Compiler?

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

```
MOV R1,2
SAL R1
INC R1
MOV R2,R1
```
Anatomy of a Compiler

Compiler

Source text

Frontend (analysis)

Semantic Representation

Backend (synthesis)

Executable code

int a, b;
a = 2;
b = a*2 + 1;

MOV R1,2
SAL R1
INC R1
MOV R2,R1
Modularity

int a, b;
a = 2;
b = a*2 + 1;

SET R1,2
STORE #0,R1
SHIFT R1,1
STORE #1,R1
ADD R1,1
STORE #2,R1
MOV R1,2
SAL R1
INC R1
MOV R2,R1
SET   R1,2
STORE #0,R1
SHIFT R1,1
STORE #1,R1
ADD R1,1
STORE #2,R1
MOV R1,2
SAL R1
INC R1
MOV R2,R1
Anatomy of a Compiler

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

MOV R1, 2
SAL R1
INC R1
MOV R2, R1
int a, b;
a = 2;
b = a*2 + 1;
Compiler vs. Interpreter

**Source text**

- Frontend (analysis)
- Semantic Representation
- Backend (synthesis)

**Executable code**

- Input
- Execution Engine
- Output
Compiler vs. Interpreter

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

### Syntax Tree

```
3
MOV R1, 8(ebp)
SAL R1
INC R1
MOV R2, R1
```

### Interpretation

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

### Compilation

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

1. **Frontend** (analysis)
2. **Semantic Representation**
3. **Backend** (synthesis)
4. **Execution Engine**
Just-in-time Compiler (Java example)

Just-in-time compilation: bytecode interpreter (in the JVM) compiles program fragments during interpretation to avoid expensive re-interpretation.
Why should you care?

- Every person in this class will build a parser some day
  - Or wish he knew how to build one...
- Useful techniques and algorithms
  - Lexical analysis / parsing
  - Semantic representation
  - ...
  - Register allocation

- Understand programming languages better
- Understand internals of compilers
- Understand (some) details of target architectures
Why should you care?

- Useful formalisms
  - Regular expressions
  - Context-free grammars
  - Attribute grammars
- Data structures
- Algorithms

Source

Compiler

Target

Programming Languages
Software Engineering

Runtime environment
Garbage collection
Architecture
Course Overview

Source text

Compiler

Lexical Analysis
Syntax Analysis
Semantic Analysis
Inter. Rep. (IR)
Code Gen.

Executable code

exe
Journey inside a compiler

```
x = b*b - 4*a*c
```

```
<ID,"x"> <EQ> <ID,"b"> <MULT> <ID,"b"> <MINUS> <INT,4> <MULT> <ID,"a"> <MULT> <ID,"c">
```

Journey inside a compiler

<ID,"x"> <EQ> <ID,"b"> <MULT> <ID,"b"> <MINUS> <INT,4> <MULT> <ID,"a"> <MULT> <ID,"c">
Journey inside a compiler

Abstract Syntax Tree

```
MINUS
  |   |
MUL  MULT
  |   |
'b' 'b'
  |   |
  |   |
'MULT' 'c'
  |   |
  |   |
  |   |
'MULT'
  |   |
  |   |
'4''a'
```

Journey inside a compiler

Annotated Abstract Syntax Tree

```
ULType: int
oc: R1

MINUSType: int
oc: R1

ULType: int
oc: R1

ULType: int
oc: const '4'

ULType: int
oc: sp+16

ULType: int
oc: sp+16

ULType: int
oc: sp+8

ULType: int
oc: sp+24

ULType: int
oc: R2

ULType: int
oc: R2
```

Journey inside a compiler

Intermediate Representation

R2 = 4*a
R1 = b*b
R2 = R2*c
R1 = R1 - R2

Journey inside a compiler

Intermediate Representation

\[ R_2 = 4 \times a \]
\[ R_1 = b \times b \]
\[ R_2 = R_2 \times c \]
\[ R_1 = R_1 - R_2 \]

Assembly Code

\[ \text{MOV} \ R_2, (sp+8) \]
\[ \text{SAL} \ R_2, 2 \]
\[ \text{MOV} \ R_1, (sp+16) \]
\[ \text{MUL} \ R_1, (sp+16) \]
\[ \text{MUL} \ R_2, (sp+24) \]
\[ \text{SUB} \ R_1, R_2 \]
Error Checking

- In every stage...

- Lexical analysis: illegal tokens
- Syntax analysis: illegal syntax
- Semantic analysis: incompatible types, undefined variables, ...

- Every phase tries to recover and proceed with compilation (why?)
  - Divergence is a challenge
Errors in lexical analysis

pi = 3.141.562 → Illegal token

pi = 3oranges → Illegal token

pi = oranges3 → <ID,”pi”>, <EQ>, <ID,”oranges3”>
Error detection: type checking

\[ x = 4 \times a \times \text{"oranges"} \]
The Real Anatomy of a Compiler
Optimizations

- “Optimal code” is out of reach
  - many problems are undecidable or too expensive (NP-complete)
  - Use approximation and/or heuristics

- Loop optimizations: hoisting, unrolling, ...
- Peephole optimizations
- Constant propagation
  - Leverage compile-time information to save work at runtime (pre-computation)
- Dead code elimination
  - space
- ...


Machine code generation

- Register allocation
  - Optimal register assignment is NP-Complete
  - In practice, known heuristics perform well
- Assign variables to memory locations
- Instruction selection
  - Convert IR to actual machine instructions

- Modern architectures
  - Multicores
  - Challenging memory hierarchies
Compiler Construction Toolset

- Lexical analysis generators
  - lex
- Parser generators
  - yacc
- Syntax-directed translators
- Dataflow analysis engines
Summary

- Compiler is a program that translates code from source language to target language
- Compilers play a critical role
  - Bridge from programming languages to the machine
  - Many useful techniques and algorithms
  - Many useful tools (e.g., lexer/parser generators)
- Compiler constructed from modular phases
  - Reusable
  - Different front/back ends
Coming up next

- Lexical analysis