Crash Course on Compilers

Metaphysics → Linguistics → Computation → HPC

Raghuveer S (Raghu)

MSCS, CU Boulder

A bit about me

- MS in Computer Science
- Software Engineer with Hewlett-Packard (Distributed Systems)
- Software Engineer with Synopsys Inc (VLSI)
- **Interest:** Compilers $\cap$ Distributed Systems

Outline

This Presentation

- Understanding Compilers from a Metaphysical and Linguistic Standpoint (15 min).
- Seeing under the hood of a hypothetical compiler (25 min).
- Delving deep into optimizing the compiler from the user's perspective and compiler writer's perspective (10 min).

Outline

Next Presentation

- Superoptimization and Instruction Selection (45 min).

What is a Compiler?

A tool to reason about programs.

More specifically,

- Epistemological Tool
  - What do you know?
  - How do you know that what you know is "True"?

Compilers and Theories of Truth

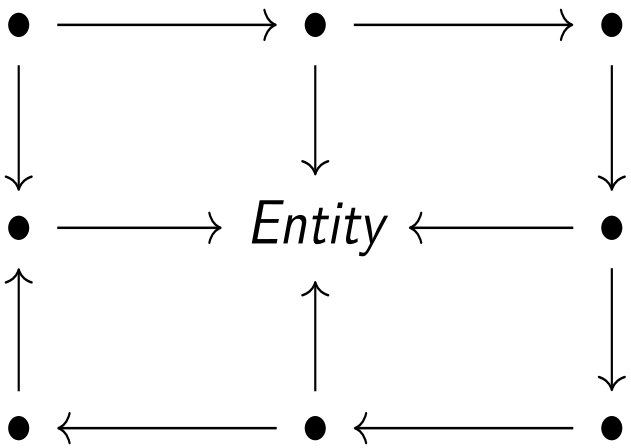
- Correspondence Theory (A form of semantic theory)
- A dog exists vs This dog exists

Other examples:

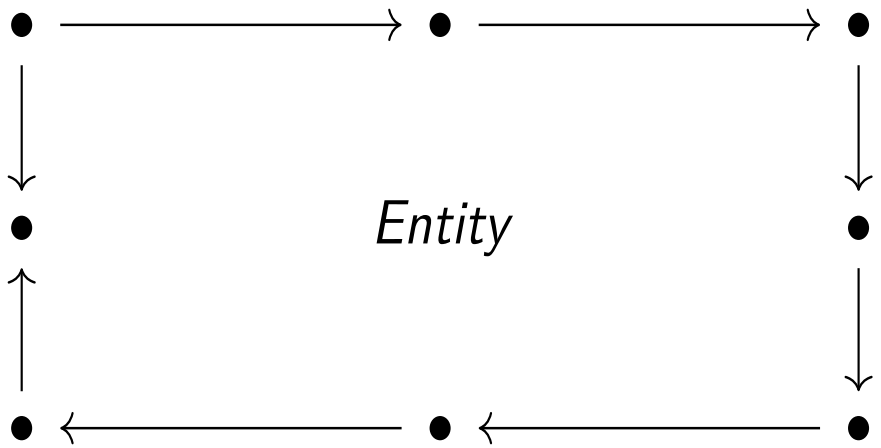
- Identity Theory
- Deflationary Theory

Context and Structure

- Context $\longleftrightarrow$ Structure



Structure



Context + Structure

1

[1] Not related to the term Context in CFG or CSG from Chomsky's Heirarchy.

Structural Dependency

"The guy who fixed the car carefully packed his tools."

Meaning of this statement? Role of "carefully"?

Structural Dependency

- "The guy who fixed the car, carefully packed his tools."
- "The guy who fixed the car carefully, packed his tools."

Role of "carefully"?

Structuralism and Fregian Julius Caesar

Frege's Julius Caesar:

- What number would Julius Caesar be, if he were to be a number?
- 42? 1729? 2520? 17? ...

- 1 and Peano

As computer scientists, we are all structuralists.

Compilers: Syntax and Semantics

Think graphs with nodes and edges.

- Nodes and Edges as Syntax
- Whether a node or an edge can be added as Semantics

Semantics and Sentential Functions

- Satisfaction
- Validity

Sentential Functions

_____, is a three-volume work on the foundations of mathematics, also considered by many as the first book on Type Theory, was written by Alfred North Whitehead and Bertrand Russell.

- What satisfies?
- What makes the satisfaction valid?

One more example on Semantics

- x = 2 [Assignment](This leads to 1 line of generated code when not optimized.)
- x == 2 [Comparison](This leads to over 50 lines of generated code when not optimized.)

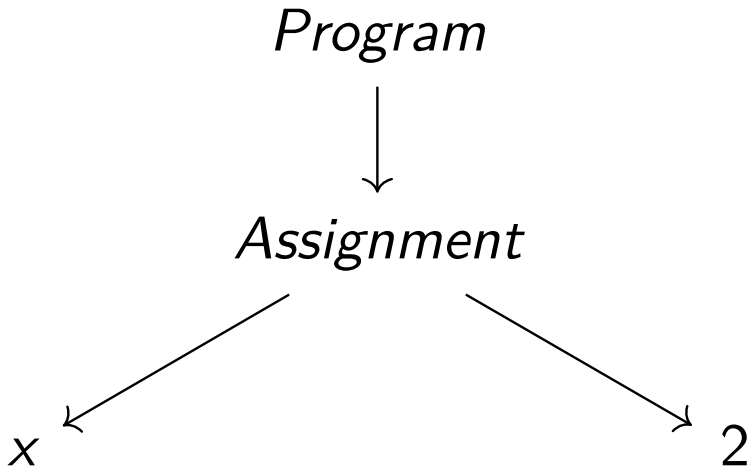
Under the Hood

Now once you understand Syntax and Semantics, all that is left is translating it to a machine understandable concept.

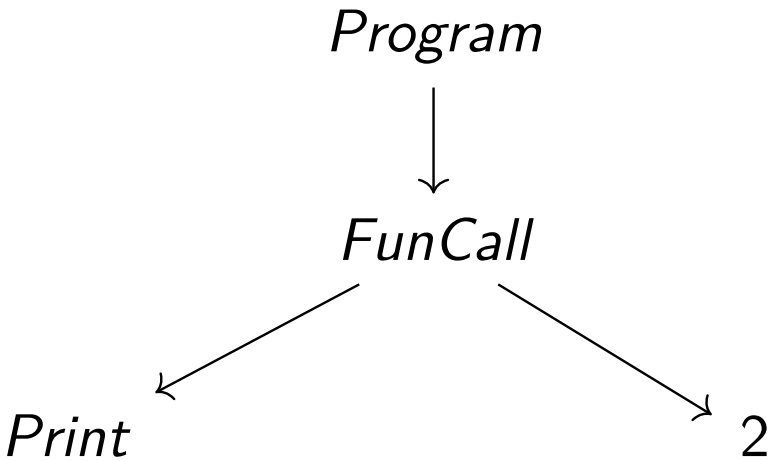
- Lexing
- Parsing
- Flatten
- IR Gen (Optional)
- Register Allocation (Coloring and Spilling)
- Code Generation

Parsing

This is where the semantic analysis starts:



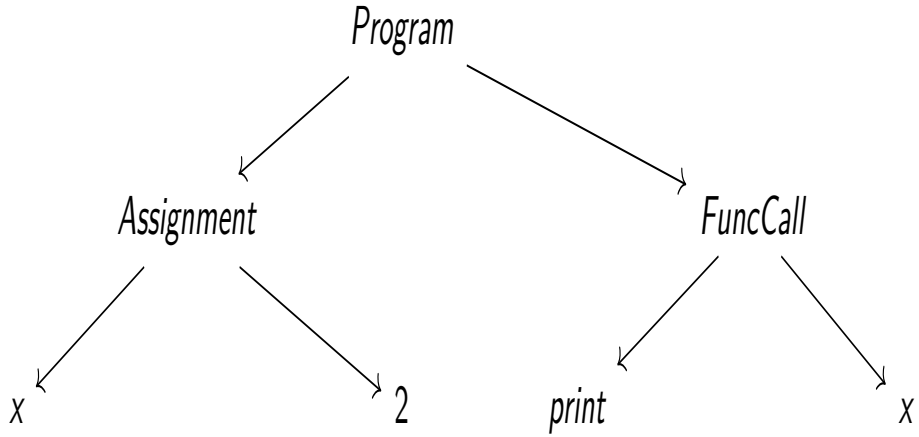
- **Program1:** x = 2 →
- **Program 2:** print 2 →



Parsing

This is where the semantic analysis starts:

- **Program1_2:** x = 2
  print x →



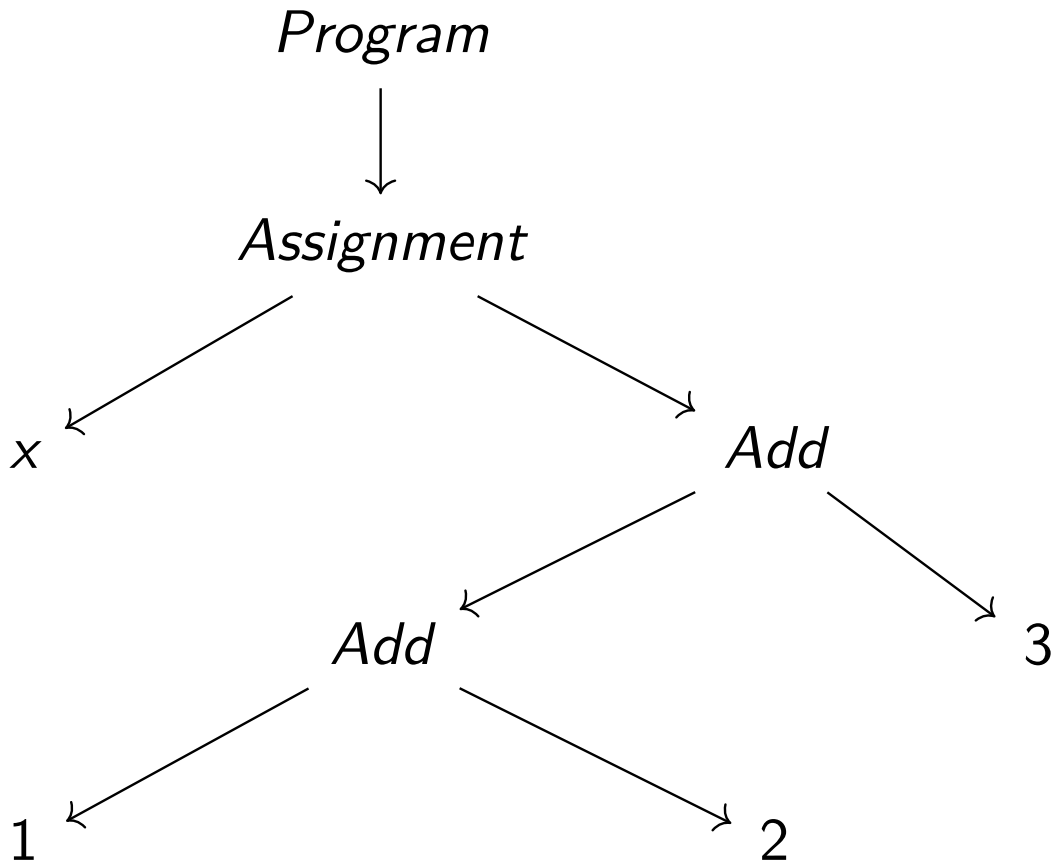
Flattening

```
x = 1 + 2 + 3
```

- How would you go about handling this statement?

Flattening Contd

This is how the parser usually handles it:




```
x = 1 + 2 + 3
      ↓ (#why)
tmp0 = 1 + 2
tmp1 = tmp0 + 3
x = tmp1
```

IR Gen

Let's assume that our target is x86. We'll create an IR that resembles the x86 asm.

This becomes: $\rightarrow$

```
tmp0 = 1 + 2
tmp1 = tmp0 + 3
x = tmp1
```

This:

```
movl $1, tmp0
addl $2, tmp0
movl tmp0, tmp1
addl $3, tmp1
movl tmp1, x
```

Register Allocation

Variables need home and We don't have infinite memory!

Register Allocation

Liveness Analysis

$$\texttt{liveset}(i) = \texttt{liveset}(i+1) - \texttt{write}(i) \cup \texttt{read}(i)$$

```
movl $1, tmp0
addl $2, tmp0
movl tmp0, tmp1
addl $3, tmp1
movl tmp1, x
```

```
{}
movl $1, tmp0
{tmp0}
addl $2, tmp0
{tmp0}
movl tmp0, tmp1
{tmp1}
addl $3, tmp1
{tmp1}
movl tmp1, x
{}
```

Register Allocation

Interference Graph

1. $\{\}$
2. `movl $1, tmp0`
3. $\{tmp0\}$
4. `addl $2, tmp0`
5. $\{tmp0\}$
6. `movl tmp0, tmp1`
7. $\{tmp1\}$
8. `addl $3, tmp1`
9. $\{tmp1\}$
10. `movl tmp1, x`
11. $\{\}$

- **In need of a home:** `tmp0`, `tmp1`, `x`
- `tmp0`, `x`, `tmp1`

Register Allocation

Coloring and Spilling

`tmp0, x, tmp1`

Using graph coloring:

```
tmp0: "eax",
tmp1: "eax",
x: "eax"
```

CodeGen

- `tmp0: "eax", tmp1: "eax", x: "eax"`

```
movl $1, tmp0
addl $2, tmp0
movl tmp0, tmp1
addl $3, tmp1
movl tmp1, x
```

```
movl $1, %eax
addl $2, %eax
# movl %eax, %eax
addl $3, %eax
# movl %eax, %eax
```

Demo

Compiler Explorer (Click Me!)

Optimizations

- User vs Compiler Writer

Optimizations

User: Generic

- Compiler Flags: `-O2`, `-O3`
- Loop Unrolling/Vectorization, Function Inlining, Constant Propagation, etc.

Optimizations

User: Generic

Example 1: Loop Unrolling

```
for (i = 0; i < 3; i++)
{
   do_something;
}
```

```
do_something;
do_something;
do_something;
```

Optimizations

User: Generic

Example 2: Function Inlining

```
int sum(int a, int b)
{
   return a + b;
}
int main()
{
   int y = sum(2, 3);
}
```

```
int main()
{
   int y = 2 + 3;
}
```

Optimizations

User: Specific

- `#pragma omp, -march, compiler flags like "-fno-ira-share-spill-slots", "-fdelete-null-pointer-checks", "-fdelayed-branch", compiler specific attributes like __attribute((always_inline))__` etc.
- Parallelization, Architecture specific optimizations etc.

Optimizations

Compiler Writer

Example 1: `-fdelayed-branch`

- Simple Blocks in a CFG will be re-evaluate to see if some intstructions can be run during the delayed branch slot.

Optimizations

Compiler Writer

Example 1: `-fno-ira-share-spill-slot`

- Register Allocation step will redone by marking a particular variable in the IR as unspillable, so that it always gets a register.
- Useful when you want to reduce the memory usage in a computationally intensive code snippet.

Next Presentation

Stochastic Optimizations and Instruction Selection

QA