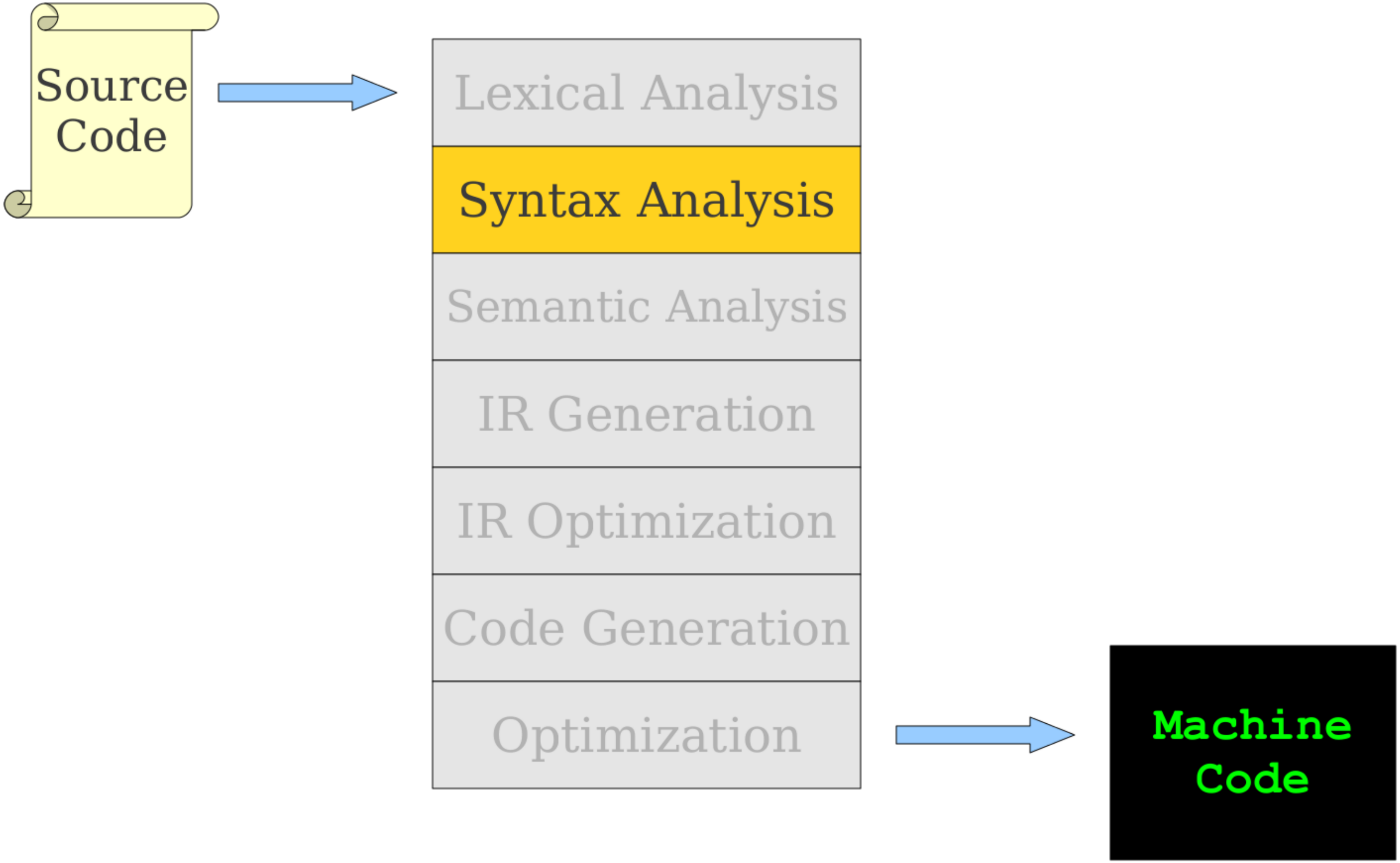
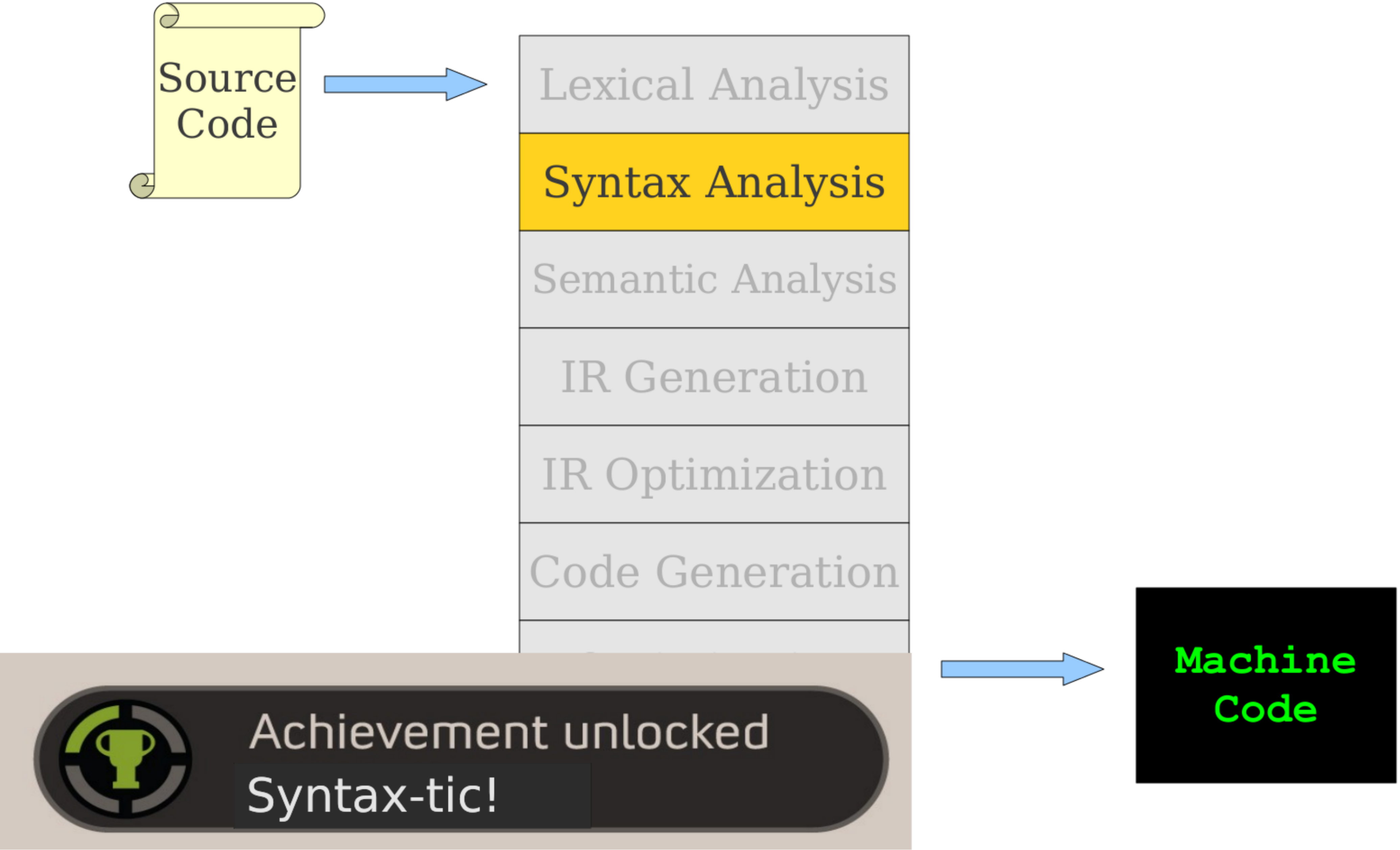
Semantic Analysis

Where We Are


Source Code
Lexical Analysis
Syntax Analysis
Semantic Analysis
IR Generation
IR Optimization
Code Generation
Optimization
Machine Code

Where We Are



Source Code

Lexical Analysis

Syntax Analysis

Semantic Analysis

IR Generation

IR Optimization

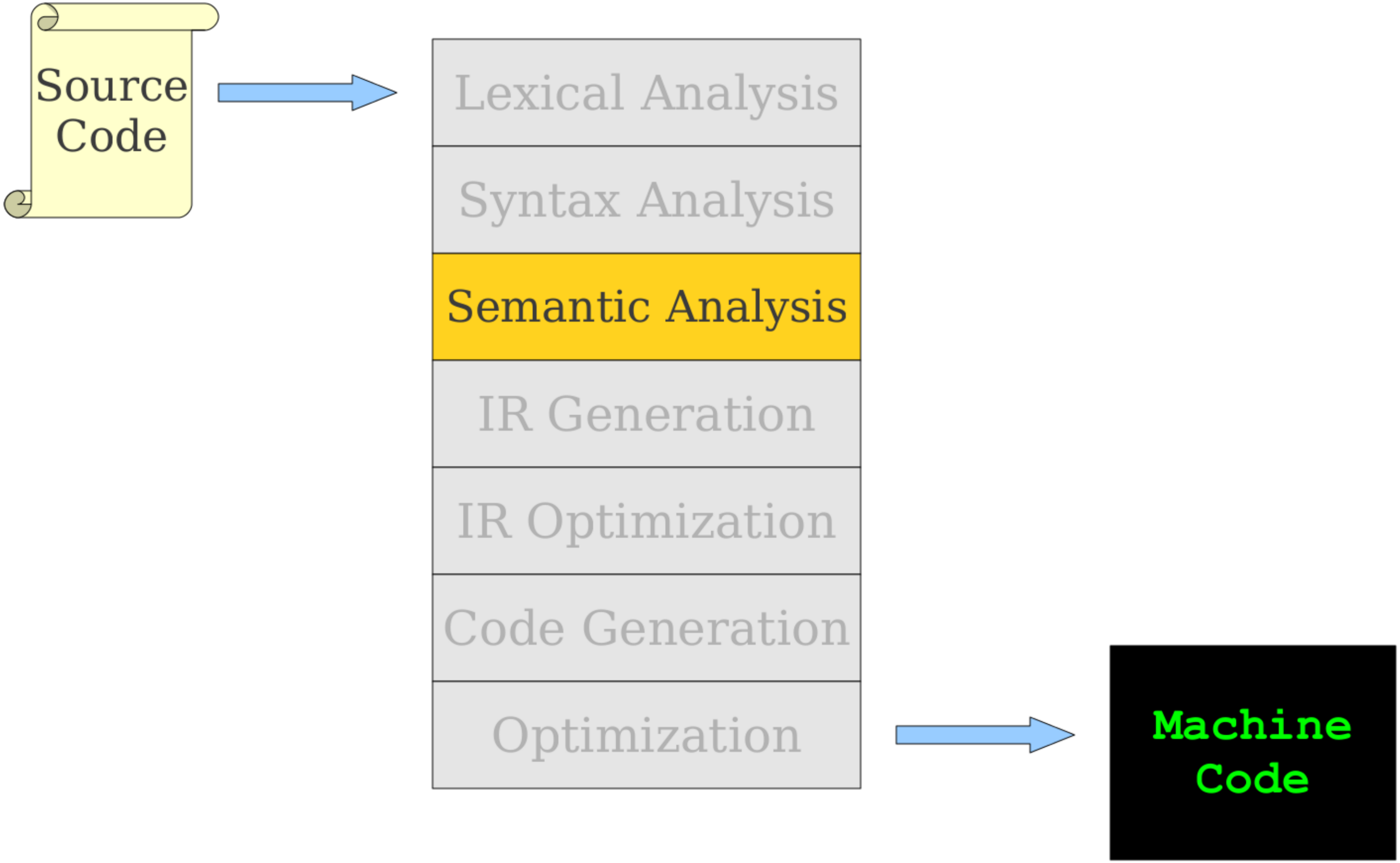
Code Generation

Machine Code

Achievement unlocked

Syntax-tic!

Where We Are



Source Code

Lexical Analysis

Syntax Analysis

Semantic Analysis

IR Generation

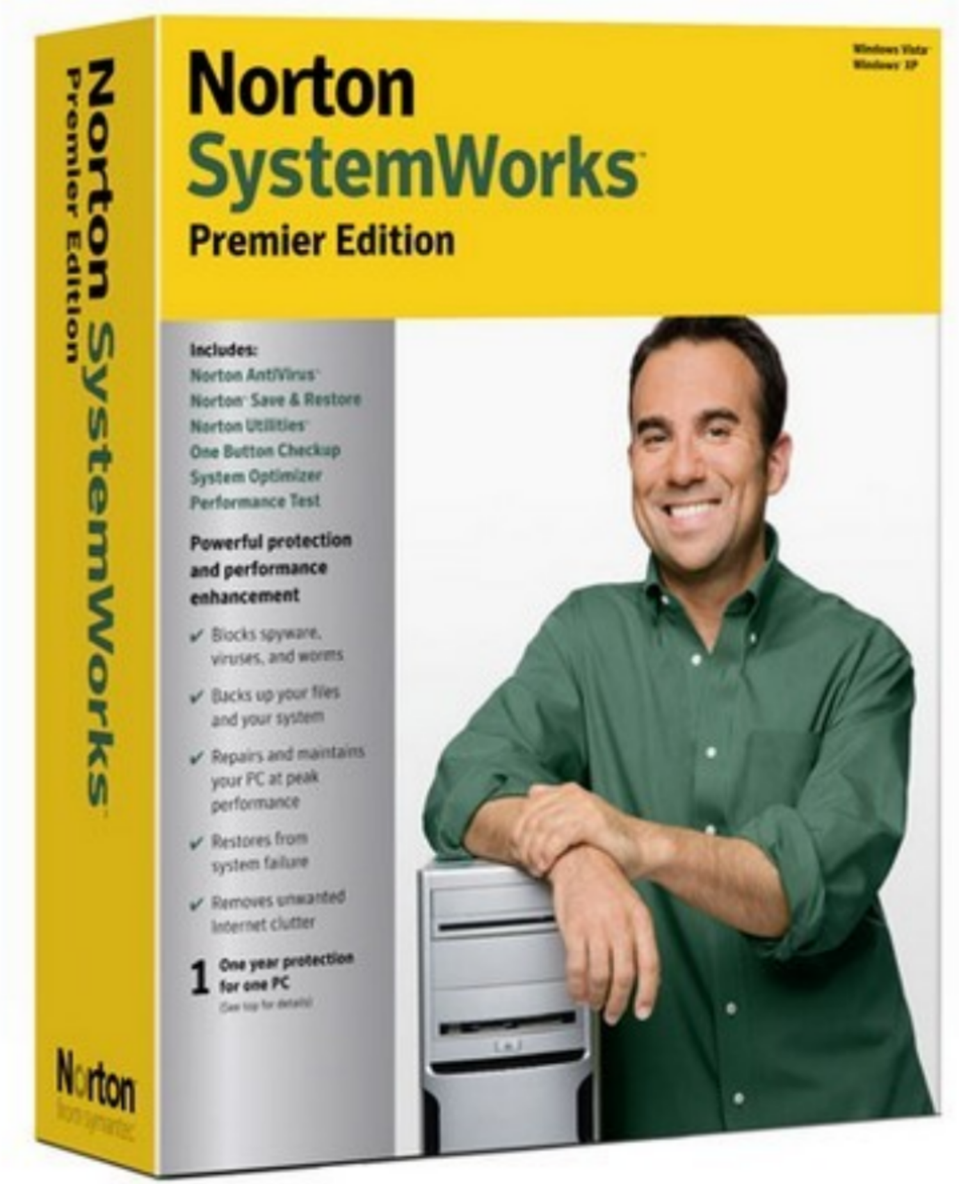
IR Optimization

Code Generation

Optimization

Machine Code

Not Symantec Analysis


Norton
SystemWorks
Premier Edition
Windows Vista
Windows XP
Includes:
Norton AntiVirus
Norton Save & Restore
Norton Utilities
One Button Checkup
System Optimizer
Performance Test
Powerful protection and performance enhancement
✓ Blocks spyware, viruses, and worms
✓ Backs up your files and your system
✓ Repairs and maintains your PC at peak performance
✓ Restores from system failure
✓ Removes unwanted Internet clutter
1 One year protection for one PC
Norton SystemWorks Premier Edition
Norton from symantec

Where We Are

- Program is *lexically* well-formed:
  - Identifiers have valid names.
  - Strings are properly terminated.
  - No stray characters.
- Program is *syntactically* well-formed:
  - Class declarations have the correct structure.
  - Expressions are syntactically valid.
- Does this mean that the program is legal?

A Short Decaf Program

```
class MyClass implements MyInterface {
    string myInteger;

    void doSomething() {
        int[] x = new string;

        x[5] = myInteger * y;
    }
    void doSomething() {

    }
    int fibonacci(int n) {
        return doSomething() + fibonacci(n - 1);
    }
}
```

A Short Decaf Program

```
class MyClass implements MyInterface {
    string myInteger;

    void doSomething() {
        int[] x = new string;

        x[5] = myInteger * y;
    }
    void doSomething() {

    }
    int fibonacci(int n) {
        return doSomething() + fibonacci(n - 1);
    }
}
```

Interface not declared

Wrong type

Can't multiply strings

Variable not declared

Can't redefine functions

Can't add void

No main function

Semantic Analysis

- Ensure that the program has a well-defined **meaning**.
- Verify properties of the program that aren't caught during the earlier phases:
  - Variables are declared before they're used.
  - Expressions have the right types.
  - Arrays can only be instantiated with **`NewArray`**.
  - Classes don't inherit from nonexistent base classes
  - …
- Once we finish semantic analysis, we know that the user's input program is legal.

Challenges in Semantic Analysis

- Reject the largest number of incorrect programs.
- Accept the largest number of correct programs.

Validity versus Correctness

```
int main() {
    string x;
    if (false) {
        x = 137;
    }
}
```

Validity versus Correctness

```
int main() {
    string x;
    if (false) {
        x = 137;
    }
}
```

Safe; can't happen

Validity versus Correctness

```
int Fibonacci(int n) {
    if (n <= 1) return 0;

    return Fibonacci(n - 1) + Fibonacci(n - 2);
}

int main() {
    Print(Fibonacci(40));
}
```

Validity versus Correctness

```
int Fibonacci(int n) {
    if (n <= 1) return 0;

    return Fibonacci(n - 1) + Fibonacci(n - 2);
}

int main() {
    Print(Fibonacci(40));
}
```

Incorrect, should be “return n;“

Challenges in Semantic Analysis

- Reject the largest number of incorrect programs.
- Accept the largest number of correct programs.
- Do so quickly.

Challenges in Semantic Analysis

- Reject the largest number of incorrect programs.
- Accept the largest number of correct programs.
- Do so quickly.

Other Goals of Semantic Analysis

- Gather useful information about program for later phases:
  - Determine what variables are meant by each identifier.
  - Build an internal representation of inheritance hierarchies.
  - Count how many variables are in scope at each point.

Why can't we just do this during parsing?

Limitations of CFGs

- Using CFGs:
    - How would you prevent duplicate class definitions?
    - How would you differentiate variables of one type from variables of another type?
    - How would you ensure classes implement all interface methods?

Limitations of CFGs

- Using CFGs:
  - How would you prevent duplicate class definitions?
  - How would you differentiate variables of one type from variables of another type?
  - How would you ensure classes implement all interface methods?
- For most programming languages, these are *provably impossible*.
  - Use the pumping lemma for context-free languages, or Ogden's lemma.