Compilers

INF-400

Burak Arslan
ext-inf400@burakarslan.com

Galatasaray Üniversitesi

Lecture VIII
2023-12-07

Course website

burakarslan.com/inf400

Semantic Analysis

The compiler so far ...

- Lexical analysis
    - Eliminates invalid tokens using . . .

Semantic Analysis

The compiler so far ...

- Lexical analysis
    - Eliminates invalid tokens using . . .
    - . . . regular expressions

Semantic Analysis

The compiler so far ...

- Lexical analysis
    - Eliminates invalid tokens using . . .
    - . . . regular expressions
- Syntactic analysis
    - Eliminates invalid syntax trees using . . .

Semantic Analysis

The compiler so far ...

- Lexical analysis
  - Eliminates invalid tokens using . . .
  - . . . regular expressions
- Syntactic analysis
  - Eliminates invalid syntax trees using . . .
  - . . . context-free grammars

Semantic Analysis

Next step is **Semantic analysis**:

- Checks that would complicate the grammar too much (KISS)
  - See for yourself! Class stmt vs regular stmt distinction is made in the grammar.
- Checks that can't be modelled by context-free grammars

Semantic Analysis

Stuff like:

- Is a given class/function/variable declared **exactly once** in a given **scope**?
- Are types consistent?
- Are function calls consistent?
- *Do type names start with an uppercase letter whereas other identifiers start with a lowercase letter?*

Semantic Analysis

Scopes

Scoping Rules in Kiraz

Semantic Analysis

Scopes

What is the scope of an identifier?

- It's the part of a program in which that identifier is accessible
- The same identifier may refer to different things in different parts of the program
- Different scopes for same name don't overlap (it's an error otherwise)

Semantic Analysis

Scopes

We need a **symbol table** to keep track of scopes of identifiers.
The following operations mutate the symbol table:

- Class definition (in module scope)
- Method definition (in class scope)
- Attribute definition (in class scope)
- Function definition (in module scope)
- Function argument entry (in func arg scope)
- Variable definition (in regular scope[1])

[1]AKA Function scope

Semantic Analysis

Scopes in Kiraz

A scope, in **kiraz**

- Is wrapped by the `OP_LBRACE` and `OP_RBRACE` tokens.
- Inherits all entries from its parent scope

Semantic Analysis

Scopes in Kiraz

In kiraz, the following code fragment should be rejected with:

```
Semantic Error: Variable 'a' is already defined
```

```
let a = 15;

func f(): Void {
    let a = 5;
}
```

Semantic Analysis

Scopes in Python

Whereas in **Python** it's much more malleable:

```
a = 15

def f():
    a = 5

f(); print(a)
```

What would be the output?

Semantic Analysis

Scopes in Python

Whereas in **Python** it's much more malleable:

```
a = 15

def f():
    global a
    a = 5

f(); print(a)
```

What would be the output?

Semantic Analysis

Scopes in Python

Whereas in **Python** it's much more malleable:

```
a = 15

def f():
    nonlocal a
    a = 5

f(); print(a)
```

What would be the output?

Semantic Analysis

Scopes in Javascript

In **Javascript** it's even crazier:

```
function f() {
    a = 5;
}

a = 15; console.log(a)

f();    console.log(a)
```

What would be the output?

Semantic Analysis

Scopes in Javascript

In **Javascript** it's even crazier:

```
function f() {
    var a = 5;
}

a = 15; console.log(a)

f();    console.log(a)
```

What would be the output?

Semantic Analysis

Scopes in Javascript

In **Javascript** it's even crazier:

```
function f() {
    let a = 5;
}

a = 15; console.log(a)

f();    console.log(a)
```

What would be the output?

Semantic Analysis

Scopes in Javascript

In **Javascript** it's even crazier:

```
(function() {
    {
        var a = 15;
    }
    console.log(a)
})();
```

What would be the output?

Semantic Analysis

Scopes in Javascript

In **Javascript** it's even crazier:

```
(function() {
    {
        let a = 15;
    }
    console.log(a)
})();
```

What would be the output?

Semantic Analysis

Scopes in Kiraz (pt.2)

Back to kiraz...

Semantic Analysis

Scopes in Kiraz (pt.2)

Back to kiraz...

Further scoping rules:

- Functions and Classes don't obey definition order.
- ie. They can be referenced before they are defined.

Semantic Analysis

Scopes in Kiraz (pt.2)

Two options to implement this:

- Require function prototypes / forward declarations like in C/C++
- Use multiple passes for each scoping class

Semantic Analysis

Scopes in Kiraz (pt.2)

Two options to implement this:

- Require function prototypes / forward declarations like in C/C++
- Use multiple passes for each scoping class
  ↰ this is what we are going to do

Semantic Analysis

Scopes in Kiraz (pt.2)

The symbol table is a class with the following interface:

```
void add_symbol(std::string name, Stmt::Ptr);
Stmt::Ptr get_symbol(std::string name) const;

Scope enter_scope(ScopeType, Stmt::Ptr);
void exit_scope();

/* misc. accessors */
```

Semantic Analysis

Scopes in Kiraz (examples)

Let's see how it's supposed to work

Semantic Analysis

Scopes in Kiraz (examples)

```
func f() : R { };
```

Error at 1:16: Return type 'R' of function 'f' is not found

Semantic Analysis

Scopes in Kiraz (examples)

```
func main() : Void { };
```

Semantic Analysis

Scopes in Kiraz (examples)

```
func main() : Void {
    io.print("Hello world!\n");
};
```

Error at 2:14: Identifier 'io' is not found

Semantic Analysis

Scopes in Kiraz (examples)

```
import io;
func main() : Void {
    io.print("Hello world!\n");
};
```

Semantic Analysis

Scopes in Kiraz (examples)

```
import io;
func main() : Integer64 {
    io.print("Hello world!\n");
    return 0;
};
```

Semantic Analysis

Scopes in Kiraz (examples)

```
import io;
class Main {
    func say_hello() : Integer64 {
        io.print("Hello world!\n");
        return 0;
    }
}
```

Semantic Analysis

Scopes in Kiraz (examples)

```
import io;
class Main {
    let hello = "Hello world!\n";
    func say_hello() : Integer64 {
        io.print(hello);
        return 0;
    }
}

func main() : Integer64{
    let hello = "Hello mars!\n";
    io.print(hello);
    return 0;
}
```

Semantic Analysis

Scopes in Kiraz (examples)

```
import io;

func say_hello(a: String) : Void {
  let h = get_hello();
  io.print(h);
}

func get_hello() : String {
  return "Hello, World!\n";
}
```

Semantic Analysis

Scopes in Kiraz (examples)

```
class Count {
    let i = 0;
    func inc() : Count {
      i = i + 1;
      return self;
    }
}
```

Semantic Analysis

Scopes in Kiraz (examples)

. . . and let's see how it's NOT supposed to work

Semantic Analysis

Scopes in Kiraz (examples)

```
import io;
class Main {
    let hello = "Hello world!\n";
    func hello() : Integer64 {
        io.print(hello);
        return 0;
    }
}
```

Error at 7:5: Identifier 'hello' is already in symtab

Semantic Analysis

Scopes in Kiraz (examples)

```
func f(a: String, a: String) : Void {
}
```

Error at 2:1: Identifier 'a' in argument list of function 'f' is already in symtab

Semantic Analysis

Scopes in Kiraz (examples)

```
func f(a: Integer64) : Void {
  let a = 5;
}
```

Error at 2:12: Identifier 'a' is already in symtab

Semantic Analysis

Scopes in Kiraz (examples)

```
class C {
  let hello = "hello";
  func hello() : Void {};
}
```

Error at 3:24: Identifier 'hello' is already in symtab

Semantic Analysis

Scopes in Kiraz (examples)

```
class C {
  let hello = "hello";
  func f() : Void {
    let hello = "world";
  }
}
```

Error at 4:17: Identifier 'hello' is already in symtab

Semantic Analysis

Scope Types

Kiraz has:

- %100 static scoping like eg. C
- Unlike eg. Python

```
a = 5
del a
```

Semantic Analysis

Scope Types

We need scope types to determine implicit identifiers:

- Module
  - All class and func names are in scope anywhere
- Class - has `SelfType`
  - All attribute and method names are in scope anywhere
- Function - doesn't allow `func` or `class` keywords
  - Variable names are made available in order (ie not before being defined)
- Method - same as above, additionally has `self`
  - All attribute and method names are in scope anywhere

Semantic Analysis

Subsymbol Lookup

The following statements can contain other symbols:

- Modules
- Classes

Semantic Analysis

Types

Types in Kiraz

Semantic Analysis

Types

Types:

- Another concept whose definition varies from language to language
- The set of operations that a value can handle are given names called “types”.

Semantic Analysis

Types

A hot topic in any language discussion:

- Primitives vs user-defined types (classes)
- Statically typed (ki) vs dynamically typed (python, js) languages
- It's all integers all the way down (in most (all?) ISA)

Semantic Analysis

Types

Categories of types in Kiraz:

- Primitives vs user-defined types (classes)
- Statically typed vs dynamically typed vs untyped languages
- Operations applied to types are part of the semantics

Semantic Analysis

Types

Same data, but different meanings:

```
double d = 3.14159;
long l = *((long*)&d); // 0x400921F9F01B866E
```

. . . and different operations!

Semantic Analysis

Types

Dynamically typed == just one type?

- True at the implementation level
- Ergonomically – not so much.

Semantic Analysis

Types

Duck typing:

```
def add(a, b):
  return a + b
```

```
double add(double a, double b)
            { return a + b; }
int add(int a, int b)
            { return a + b; }
// etc...
```

Semantic Analysis

Types

Duck typing:

```
def add(a, b):
  return a + b
```

```
template <typename L, typename R>
auto add(L a, R b)
        { return a + b; }
// etc...
```

Run-time error

Compile-time error

Semantic Analysis

Types

Static vs dynamic typing:

- Debate still not settled
- Optional run-time type checking vs compile-time type systems with;
  - Ever-increasing complexity
  - Unsafe casts

Semantic Analysis

Types

Actually, no language is purely static or dynamic:

- Modern python has optional type checking support
- Javascript has Typescript
- C can cast any pointer to each other
- C++ additionally has `std::any`, `std::variant`, templates, etc.
  - Also `boost::variant`, `QVariant`, etc.