Contextual Analysis

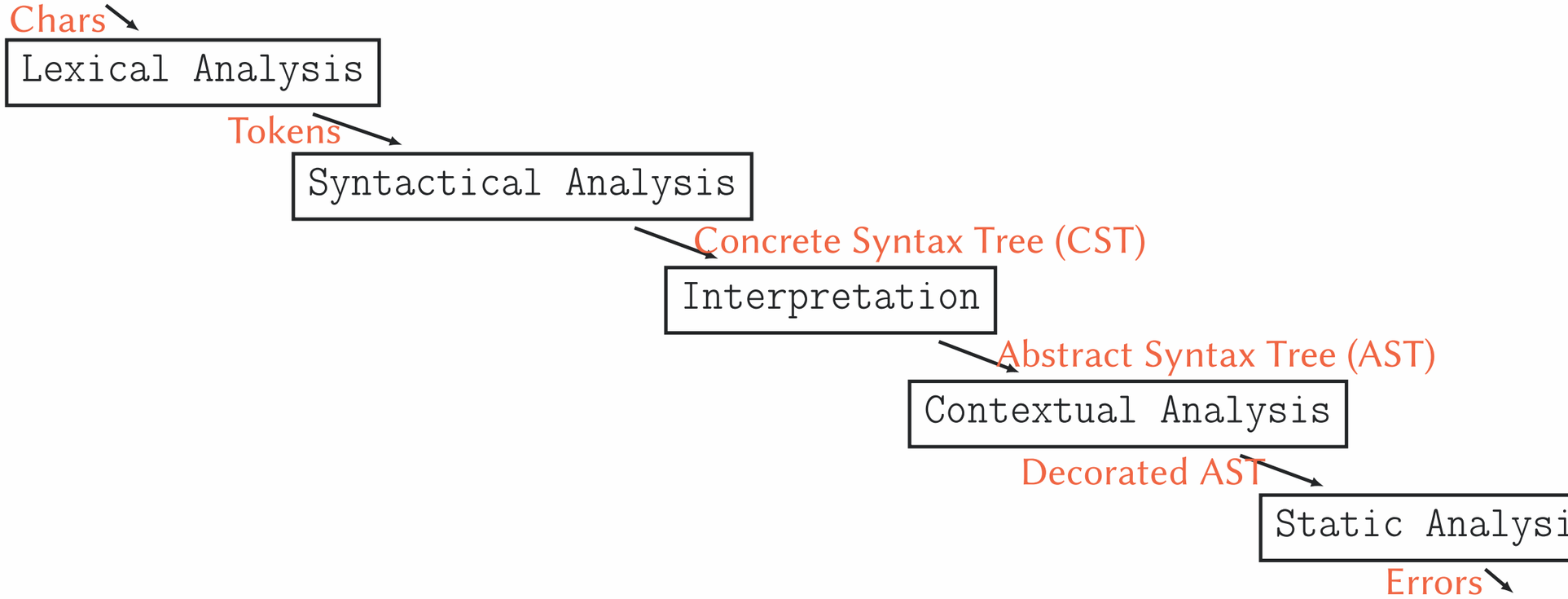
Yegor Bugayenko

Lecture #3 out of 10
80 minutes

The slidedeck was presented by the author in this YouTube Video

- Concrete vs. Abstract
- Identification
- Static Type Checking
- AST Visitor
- Decorated AST
- Control Flow Graph

Code Understanding Pipeline


Chars
Lexical Analysis
Tokens
Syntactical Analysis
Concrete Syntax Tree (CST)
Interpretation
Abstract Syntax Tree (AST)
Contextual Analysis
Decorated AST
Static Analysi
Errors

Chapter #1:

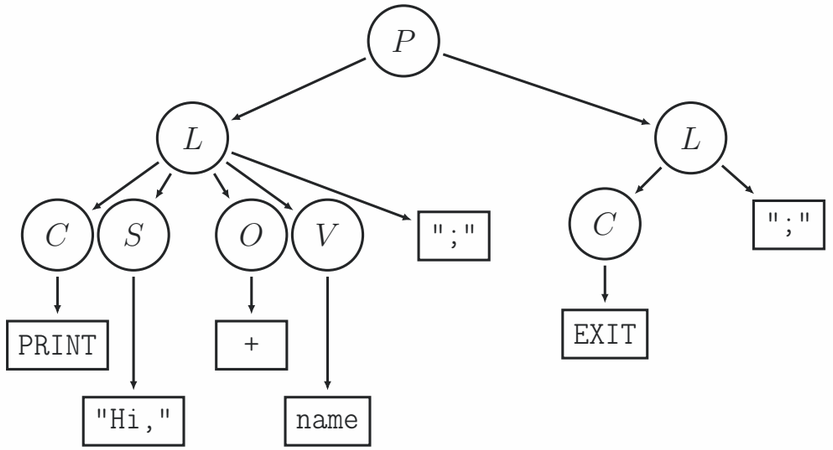
Concrete vs. Abstract

The *concrete syntax* of a programming language is defined by a context free grammar (CFG). The *abstract syntax* of an implementation is the set of trees used to represent programs in the implementation.

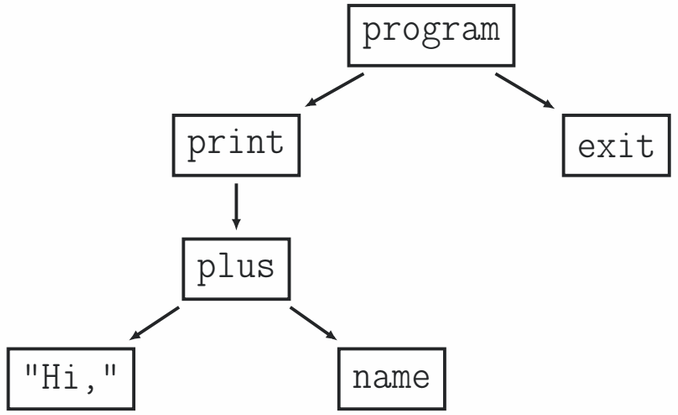
Simple program:

```
1 PRINT "Hi," + name;
2 EXIT;
```

Concrete Syntax Tree:



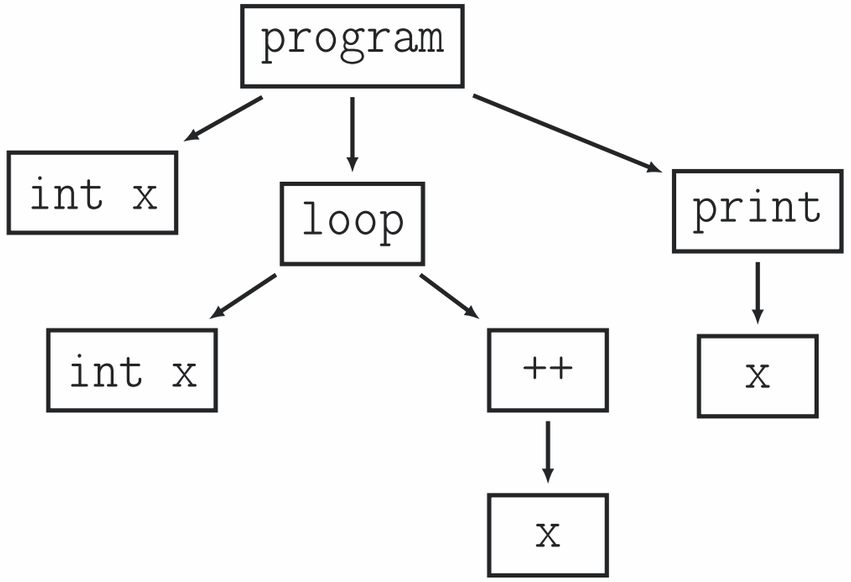
Abstract Syntax Tree:



Chapter #2:

Identification

```
int x;
loop { int x; x++; };
print x;
```



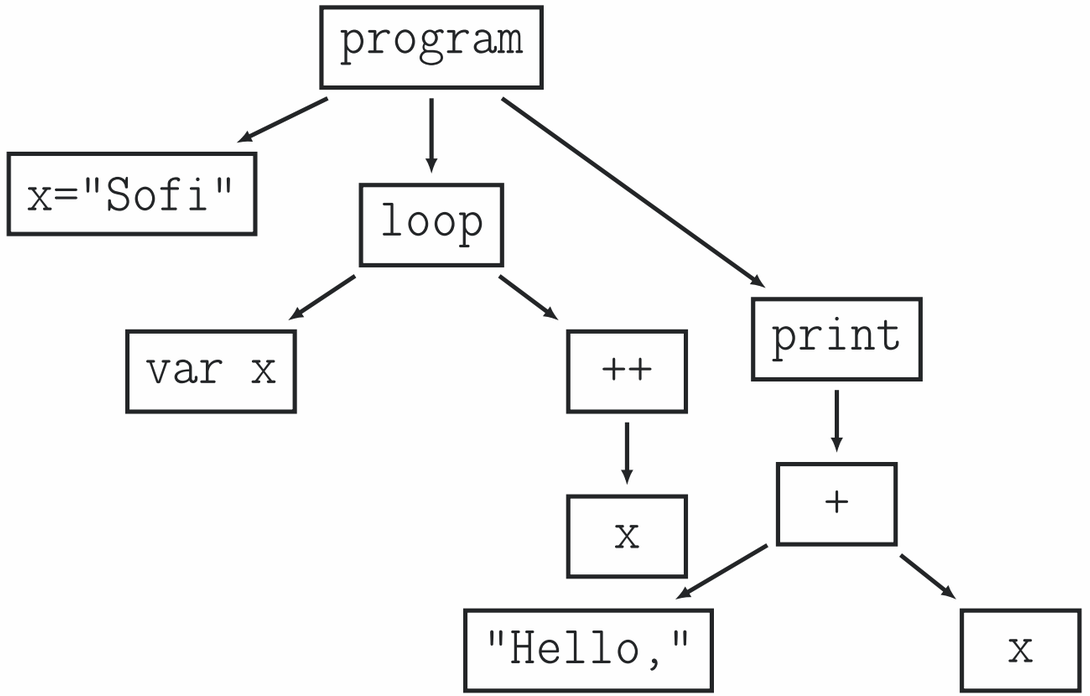
Somehow we must *link* different x to different places, where they are *declared*, maybe with the help of "*Identification Table*," or by attaching attributes to AST nodes, or both. We may want to track their *indentation levels*.

Chapter #3:

Static Type Checking

Dynamically-typed languages perform *type checking* at *runtime*, while statically typed languages perform type checking at *compile time*.

```
var x = "Sofi";
loop { var x; x++; };
print "Hello," + x;
```



Chapter #4:

AST Visitor

ANTLR4 lets us implement the following interface:

```
1 public interface ParseTreeListener {
2   void visitTerminal(TerminalNode var1);
3   void visitErrorNode(ErrorNode var1);
4   void enterEveryRule(ParserRuleContext var1);
5   void exitEveryRule(ParserRuleContext var1);
6 }
```

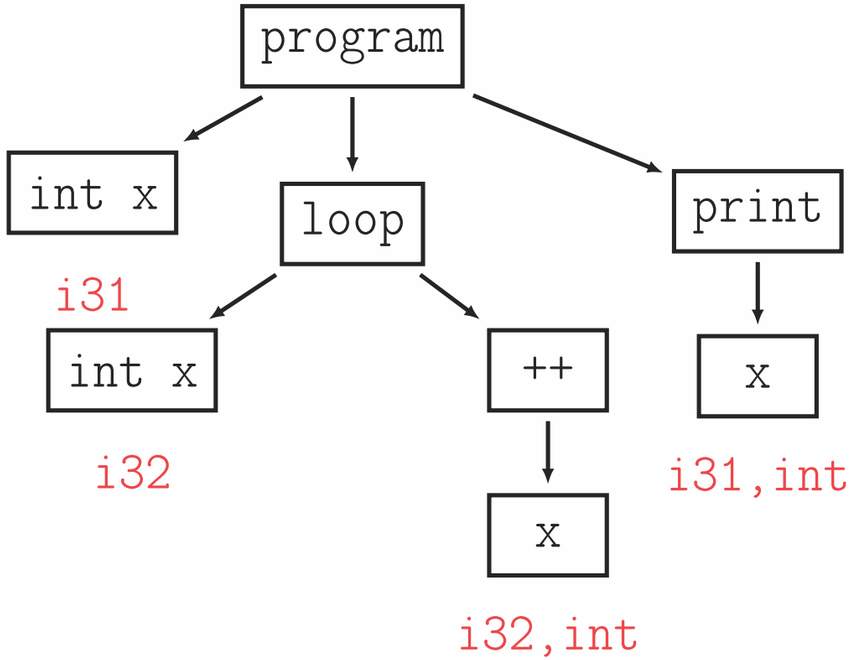
Then:

```
MyLexer lexer = new MyLexer(text); // Lexer
MyParser parser = new MyParser(
  new CommonTokenStream(lexer) // Parser
);
MyListener lsr = new MyListener(); // ParseTreeListener
new ParseTreeWalker().walk(lsr, parser.program());
```

Chapter #5:

Decorated AST

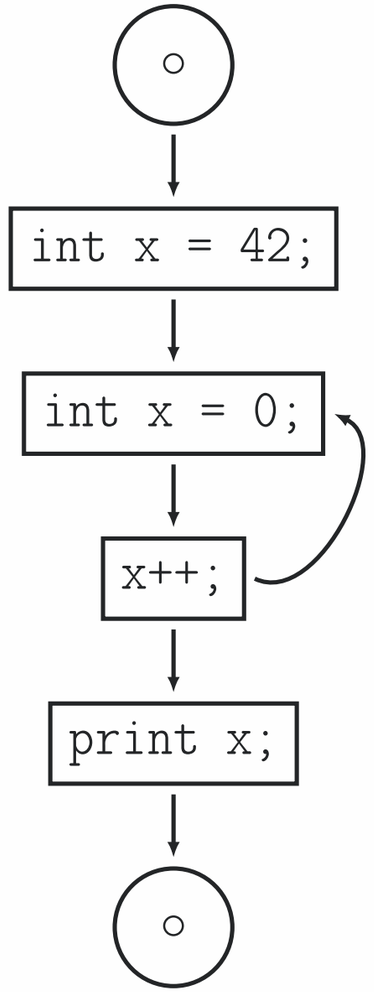
```
int x;
loop { int x; x++; };
print x;
```



Chapter #6:

Control Flow Graph

```
int x = 42;
loop { int x = 0; x++; };
print x;
```



References