Contextual Analysis

AST, Interpretation, CFG

YEGOR BUGAYENKO

Lecture #3 out of 10 90 minutes

All videos are in this YouTube playlist.

All visual and text materials presented in this slidedeck are either originally made by the author or taken from public Internet sources, such as website. Copyright belongs to their respected authors.

Concrete vs. Abstract

Identification

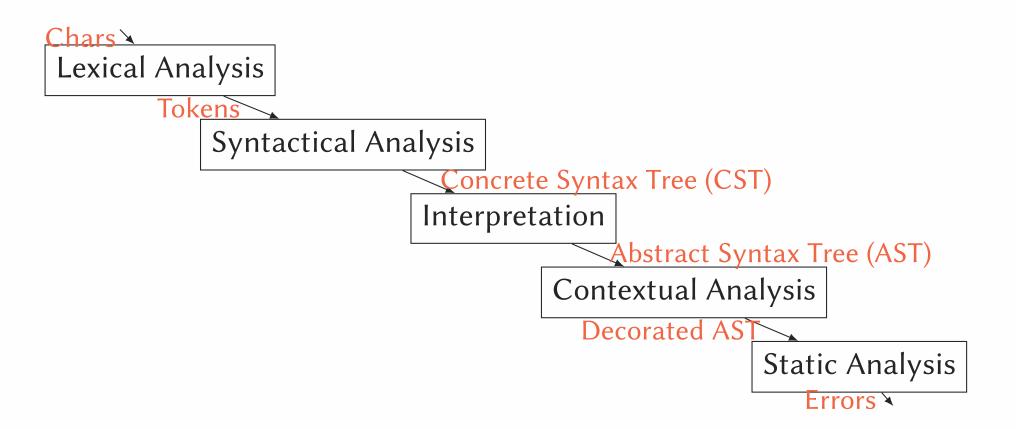
Static Type Checking

AST Visitor

Decorated AST

Control Flow Graph

Code Understanding Pipeline



Chapter #1:

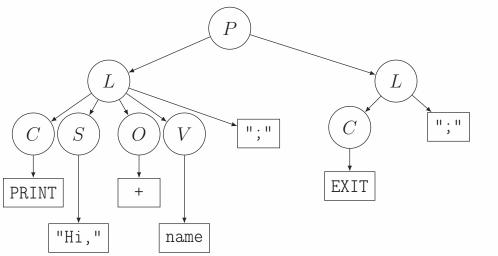
Concrete vs. Abstract

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

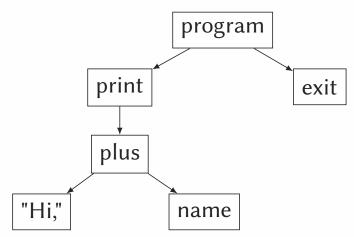
Simple program:

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

Concrete Syntax Tree:



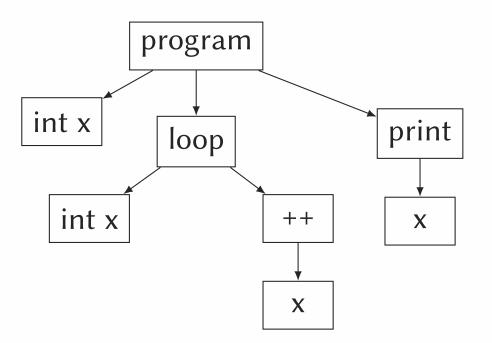
Abstract Syntax Tree:



Chapter #2:

Identification

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



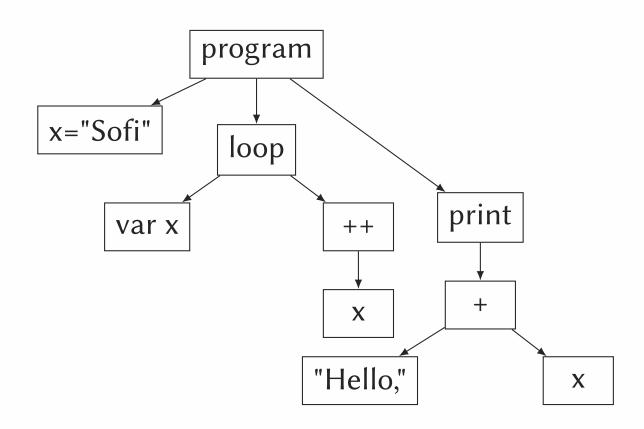
Somehow we must $\underline{\text{link}}$ different x to different places, where they are $\underline{\text{declared}}$, maybe with the help of " $\underline{\text{Identification Table}}$," or by attaching attributes to AST nodes, or both. We may want to track their $\underline{\text{indentation}}$ levels.

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:

```
public interface ParseTreeListener {
  void visitTerminal(TerminalNode var1);
  void visitErrorNode(ErrorNode var1);
  void enterEveryRule(ParserRuleContext var1);
  void exitEveryRule(ParserRuleContext var1);
}
```

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());
```

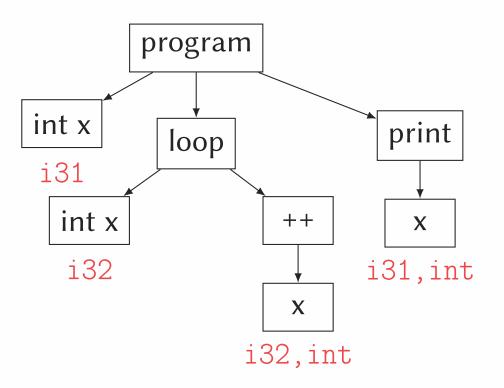
14/17

Chapter #5:

Decorated AST

15/17

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



Chapter #6:

Control Flow Graph

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

