CS:314 Fall 2024

Section **04**

Recitation **6**

Recitation 5 was Exam 1 practice - there is no recitation 5 pptx..

chris.tu@rutgers.edu

Office hours: 2-3pm Thursday CoRE 335

Today's Topics

- Exam 1 Review
- Syntax-Directed Translation

Overview

Syntax-Directed Translation (SDT) is a **compiler design method** to guide programming language translation.

Translation refers to converting a program written in one language (usually high level) to another form (usually machine level)

SDT looks at **parsing** (like LL(1)), which is syntax analysis, and desired **semantic actions**, in order to translate

↳Code Interpretation

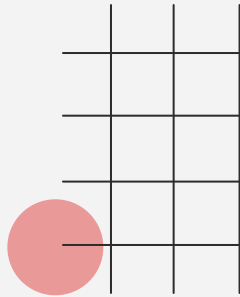
e.g. Code Generation

Type Checking

1: < expr > ::= + < expr > < expr > |
2: < digit >
3: < digit > ::= 0 | 1 | 2 | 3 | … | 9

Originally, we would use a LL(1) recursive descent parser to accept or reject strings that followed our grammar or not.

We could create resultant parse trees that represent a derivation of a expression.



CFG:

1: < expr > ::= + < expr > < expr > |
2: < digit >
3: < digit > ::= 0 | 1 | 2 | 3 | … | 9

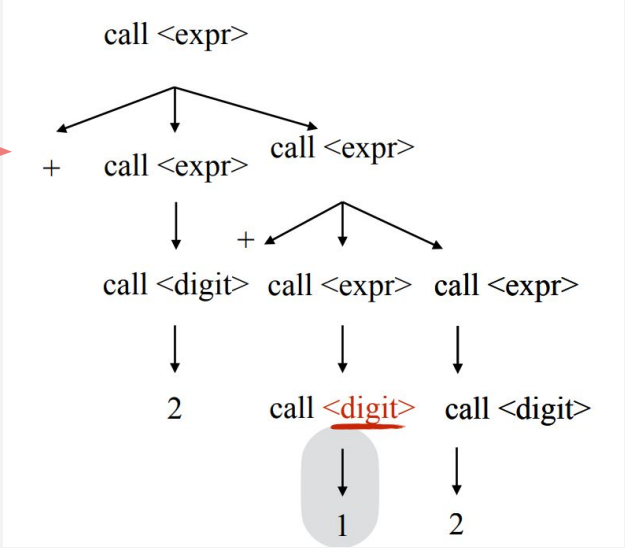
Question, what language is this in?

+ 2 + 1 2

Parse Table:

	+	0…9
< expr >	rule 1	rule 2
< digit >	error	rule 3

Parse Tree:



Originally, we were concerned with the correctness of the input token sequence.

Code Interpretation

Now, we modify the parser to do a **semantic action we want.**
An **Interpreter** runs code directly executes instructions written in a programming language.

expr() is now an int function which returns an int

Once the lookahead is a number 0-9, expr() calls the int function digit()

digit() returns the value of the token as an int ....

Interpreter

```
int expr( ) {
  int val1, val2; // two values
switch token {
    case +:   token := next_token( );
              val1 =  expr( ) ;
              val2 = expr( );
              return val1 + val2;
    case 0..9:
              return digit( );
    …
  } // End switch case
} //End expr()

int digit( ):  // return value of constant
  switch token {
    case 1: token := next_token( ); return 1;
    case 2: token := next_token( ); return 2;
    …
  } // End switch case
}// End digit( )
```

Now, the parser will return the resultant value when we think about evaluating the expression

+ 2 + 1 2

The parser, modified through SDT analysis to handle the semantic action of Code Interpretation, returns 5.

...all the way to case 9:
digit() returns 9, which gets returned by expr() above.

Refer to SDT Lecture Slides to see:

- Type Checking example and
- Code Generation example with Registers