Levels of Program Correctness

19CSE205 : PROGRAM REASONING

Dr. Swaminathan J

Assistant Professor

Department of Computer Science and Engineering

AMRITA VISHWA VIDYAPEETHAM

Jul - Dec 2020

Contents

Classifying correctness

Correctness is a relative term. It indicates absence of errors in programs. Based on the types of errors in a program, correctness can be classifed into following levels.

Lexical correctness	Syntax correctness	Semantic correctness	Logic correctness

1. Lexical correctness refers to well-formedness of individual words in a program.
2. Syntax correctness refers to well-formedness of each statement in a program.
3. Semantic correctness refers to meaningfulness between different part of code or environment.
4. Logic correctness refers to correctness with respect to program's goal/objective.

(The examples are based on C programming language context)

Lexical errors

- An ill-formed word/lexeme
- The compiler catches them

Examples

- *23ab*
- *$?*

Syntax errors

- An ill-formed statement
- The compiler catches them

Examples:

- $a + b = c;$
- $if\ (a == b)\ else\ a = b;$

Semantic errors

- An action out of context
- The compiler may catch them
- Or result in runtime error

Examples:

- *int x; . . . . . .* $x =$ *"hello";*
- *int * p; *p = 5;*
- *FILE * f = fopen("ab.c","w");*
  Note: ab.c may not exist

Role of static analyzers

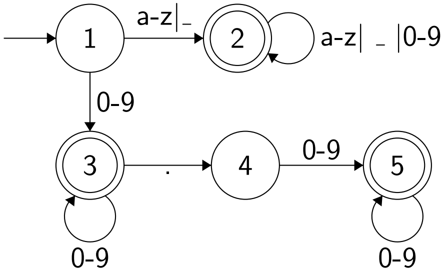
Static analysis refers to the process of analyzing source code to derive variety of useful information. In this case, we are interested in ascertaining the correctness.

- The program is first turned into one or more data structure(s) and analysis is carried out.
- Data structures employed are some form or variants of
  - Stack
  - Tree
  - Graph
  - Dictionary
- Static analyzers are usually automated. It takes the program source as input and spits out the inferences.
- A compiler is a good example of static analyzer.
- We will look briefly at how compilers catch these errors.

Lexical correctness is accomplished by a graph, known as finite state automaton, which attempts to recognize each lexeme of the program, one by one, based on its structure.

- If recognized, the lexeme is classified.
- If not, compiler flags an error.

area = breadth * height / 2;

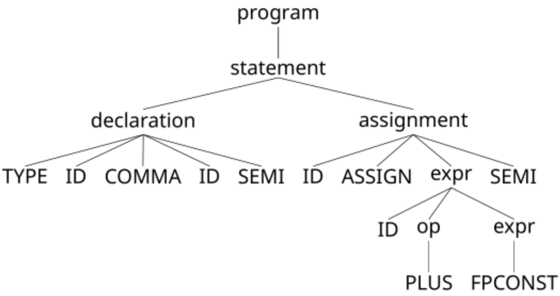


Lexeme	Token
area	IDENTIFIER
=	ASSIGN
breadth	IDENTIFIER
*	MULT
height	IDENTIFIER
/	DIV
2	INT_CONST
;	SEMI

Syntax correctness is accomplished by representing the lexicalized source code in the form of a tree, known as parse tree, and checking if it adheres to syntax specifications of the language.

program	←	statement*
statement	←	declaration
		| assignment
		| . . .
declaration	←	TYPE ID (COMMA ID)* SEMI
assignment	←	ID ASSIGN expr SEMI
expr	←	ID (op expr)*
op	←	PLUS
		| MINUS
		| MULT
		| DIV

```
int x, y;      y = x + 23.65;
```



Semantic correctness

Semantic correctness is accomplished by (i) a tree, known as Abstract Syntax Tree (AST) and (ii) a look-up table, known as Symbol Table. AST is a simplified version of parse tree.

Sample program

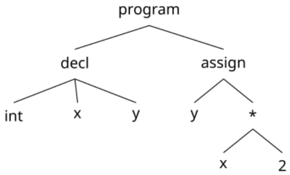
```
int x, y;
y = x * 2;
```

x is not initialized.

So y cannot be computed.

Assume default value or flag error/warning.

Abstract Syntax Tree



Symbol table

var	type	value
x	int	?
y	int	?

Compilers can ascertain only partial semantic correctness.

- Type mismatch
- Undeclared variable
- Uninitialized variable
- Function call & definition signature mismatch

Other errors slip into runtime.

- Division-by-zero
- Memory faults
- File exceptions

Use exception handling feature!

Logic correctness

Logic correctness implies program exhibits "correct" functionality or behavior.

- Errors in program logic does not result in compile time or runtime errors usually.

An example: Computing factorial

```
int factorial(int n) {
    int fact = 1;
    for (int i=2; i<=n; i++)
        fact = fact + i;
    return fact;
}
```

What is the flaw in this logic?

There are million things that could go wrong in program logic!

Focus of this course

Are we interested in lexical correctness?	NO	Compilers are good at this!
Are we interested in syntax correctness?	NO	Compilers are good at this!
Are we interested in semantic correctness?	YES	To a limited extent.
Are we interested in logic correctness?	YES	**Main focus of this course!**