14 Types and Type Checking

Wednesday, April 9, 2025 12:19 PM

Semantics the study of meaning.

e.g. C++

```
float x, y, z;
*x = y[z] << x(z[3.14], x->z);
```

follows the syntax rules.
but
it is meaningless.
because: operators do not match operands.

```
*x = y[z] << x(z[3.14], x->z);
```

- `*x` = deref, x is not a pointer.
- `y[z]` = not an array
- `z[3.14]` = cannot index by float
- `x(...)` = x is not a function
- `x->z` = x is not a pointer to object.

In programming languages, Semantics is concerned about identifiers and their correct use.

- **TYPE CHECKING:**

¿ Are identifiers used in a matter consistent with their definition?

i.e.
- using a variable with appropiate operators
- calling a function with appropiate number and types of parameters
- using membes of compound types correctly

¿ How?

the parser needs to remember the declaration of every named entity.

- THE SYMBOL TABLE:

A table of named entities and their attributes

name	attributes
x	int

examples of symbol table entries:

- variable.
  float y; — y : float
- constants
  const int z = 3 — z : int, const, 3
- type
  struct cell { int row, col; } — cell : struct [int row, int col]
- function
  void foo (int a, int b) { ≡ } — foo : function, void, 2, int a, int b

- Entries are different in different programming languages.

-C++:

```
int z[3];
float foo ( int x, bool y, char& c);
```

Name	attributes
z	int array
foo	function 3 (int, bool, char&)

Python.

```
z = [1,2,3]
def foo ( x, y, c ) :
  . . . .
```

Name.	attributes.
z	[1, 2, 3]
foo	function

Pascal.

```
z : ARRAY [10..12] OF INTEGER;
FUNCTION foo ( x : INTEGER,
               y : BOOL,
               VAR c : char ) : REAL ;
```

name	
z	ARRAY INT 10...12
foo	FUNCTION 3 REAL. INTEGER BOOL VAR char.

VAR c : char) : REAL ;

INTEGER
BOOL
VAR char.

★ Pascal's Enumerated Arrays.

```
Days = {Monday, Tuesday, Wednesday, Thursday, Friday}
z : ARRAY[Monday .. Friday] of INTEGER;

z[Wednesday] := 5;
```

- TYPE CONVERSION:

Most Languages automatically apply some type conversion.

e.g: $z = a + x$ a: float, x: integer.

Most parsers identify the need for conversion and apply it automatically.

int ⟷ float

¿ What conversions are Implicit?

C++ flexible.
Python let it run.... and crash.
Pascal Strict.

E.g.

C++: int ⟷ float ⟷ char ⟷ bool
all sizes all sizes

```
foo (Zombie &c)
{
  ≡
}
≡
foo("hello");
```

```
class Zombie.
{
  ≡
  void bite( ):
  { ≡
  }
  Zombie( String s):
  { ≡
  }
}
```

becomes an implicit

becomes an implicit conversion unless expicit

E.g. Pascal is strict.

C

```
int x = 70;
char c = '!';
          char int.
while ( 'W' - x ) {

  c = '!' + x;
  char char int
  print("%c", c)
  x = x + 1;
}
```

Pascal

```
VAR x : INTEGER = 70;
    c : CHAR = '!';

WHILE  ORD('W') - x > 0  DO

  c := CHR( ORD('!') + x );

END;
```

- TYPE EQUIVALENCE:

a = b

When are two entities of the same type?

1) Name-Type Equivalence

- two entities are of the same type if they are declared using the same type name.

Wolf John; Tiger tony; Wolf Bob:

Just look at text

e.g. C++

private variables they are accesible only to instances of the same class.

class DireWolf : public Wolf
{
}
;

DireWolf Robert:

So protected...

2) Structural Type Equivalence

- two entities are of the same type if they share the same internal structure.

e.g. Imagine C²

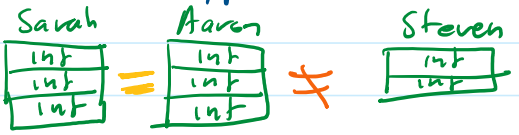
```
struct point
{
  int x,y,z;
}
```

```
struct color
{
  int r,g,b;
}
```

```
struct cell
{
  int row,col;
}
```

Under structural type-equivalence, which are of the same type.

point Sarah
color Aaron
cell Steven.



Sarah = Aaron.

- ⊖ Hard to implement.
  - you need to test structure
  - structure can be nested.

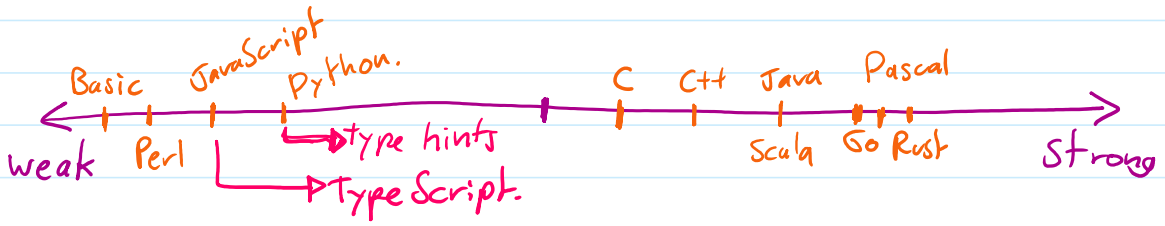
- STRONG vs WEAK TYPING

A characterization of Programming Languages

- "Strongly" typed if:
  - type violations are detected at compile/parse time
  - type conversions are explicit
  - type of a named variable remains fixed.

- General idea:

Detect errors at compile time.
Instead of letting them happen at runtime.


Basic
JavaScript
Python.
C
C++
Java
Pascal
weak
Perl
type hints
TypeScript.
Scala
Go
Rust
Strong