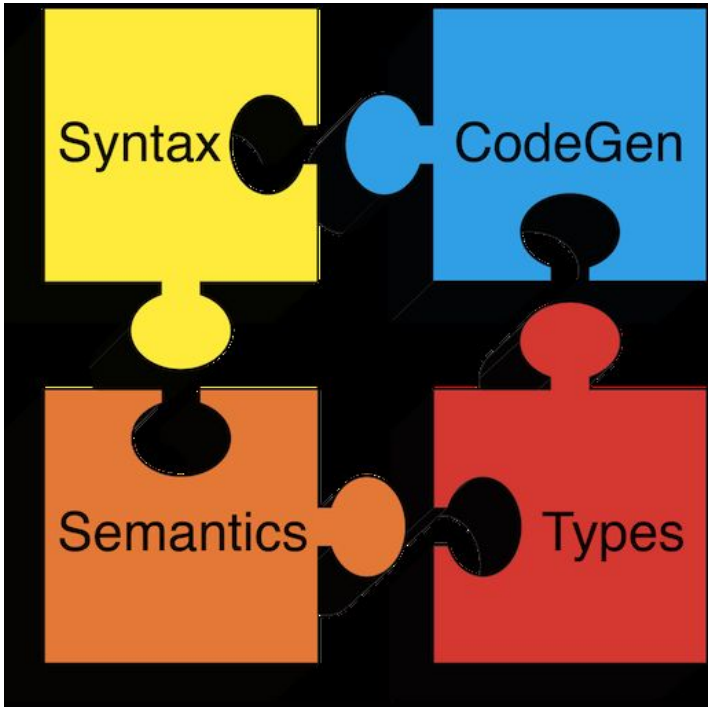
Programming Languages. Part 2. Compilers



Formal Languages

Formal Languages

Def. Let Σ be a set of characters (an *alphabet*).

A *language over* Σ is a set of strings of characters drawn from Σ

Formal Languages

- Alphabet = English characters
- Language = English sentences

- Alphabet = ASCII
- Language = C programs

Meaning function

Meaning function L maps syntax to semantics

L(E) = Meanings

reg expr set of strings

Meaning function

$$\varepsilon = \{ \text{“ ”} \}$$

$$\text{‘c’} = \{ \text{“c”} \}$$

$$A + B = \{ a \mid a \in A \} \cup \{ b \mid a \in B \}$$

$$AB = \{ ab \mid a \in A, b \in B \}$$

$$A^* = \cup A^i, \; i >= 0$$

L : Exp -> Sets Strings

Meaning function

$$L(\varepsilon) = \{ \text{“ ”} \}$$

$$L(\text{‘ c ’}) = \{ \text{“ c ”} \}$$

$$L(A + B) = L(A) \cup L(B)$$

$$L(AB) = \{ab \mid a \text{ from } L(A) \wedge b \text{ from } L(B)$$

$$L(A^*) = \bigcup L(A^i)$$

Meaning function

- Why use a meaning function?

– Makes clear what is syntax, what is semantics.

– Allows us to consider notation as a separate issue

– Because expressions and meanings are not 1-1

Meaning function

0 1 42 107

I IV X XL

Syntax and Semantic

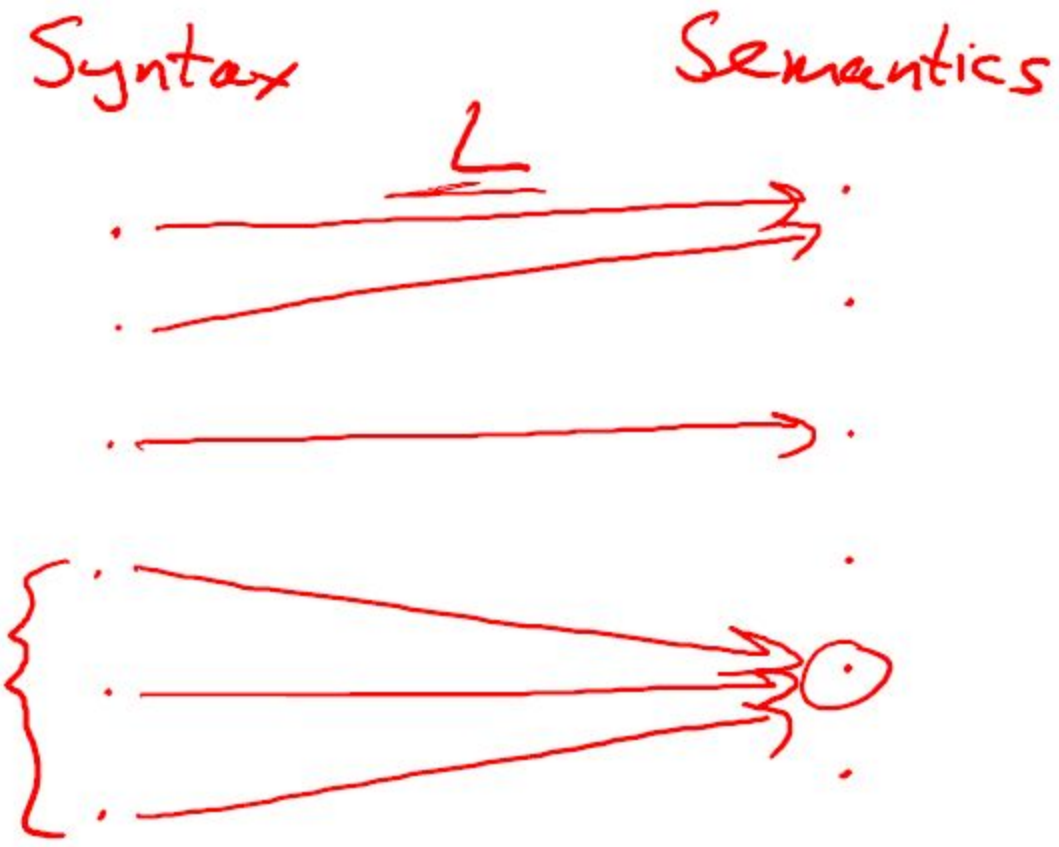
0*

0 + 0*

E + 00*

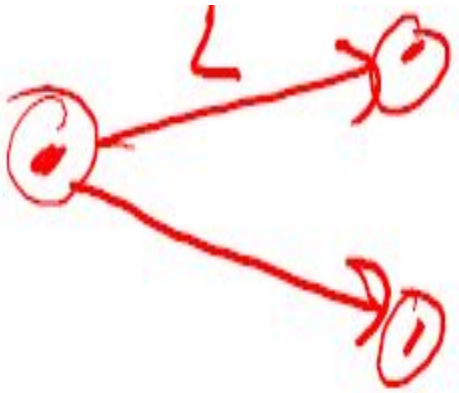
E + 0 + 0*

...



Syntax and Semantic

- Meaning is many to one
  - Never one to many!



Thank you for your attention