CS 314 Lecture 2

January 24, 2019

Syntax and semantics

Syntax and semantics

- apple
- banana
- aodorcuoacedgaduea

Syntax and semantics

- I eat an apple.
- Colorless green ideas sleep furiously.

Syntax and semantics

Variable names:

- abc123
- 123abc
- 24
- while

Syntax

What does a legal program look like?

Syntax

```
if (x > 0) {
    printf("positive");
}
```

```
if x > 0:
    print('positive')
```

```
if x > 0
then print "positive"
else print "not positive"
```

Parsing

How does a program get read?

```
x = 23 + y;
```

characters $\longrightarrow$ $\boxed{\text{lexer}}$ $\longrightarrow$ tokens $\longrightarrow$ $\boxed{\text{parser}}$ $\longrightarrow$ data structure

Parsing

How does a program get read?

```
x = 23 + y;
```

as tokens:

- <VAR, x>
- <ASSIGN>
- <CONST, 23>
- <PLUS>
- <VAR, y>
- <SEMICOLON>

Regular expressions

We define a regular expression with characters and a few operators:

- concatenation: ab means a followed by b
- alternation: $a|b$ means either a or b
- Kleene star: a^* means 0 or more copies of a
- and parentheses for grouping

(and ϵ denotes the empty string)

Regular expressions

- $(0|1)^*$
- $(01)^*$
- $1(0|1)^*1$
- aa^*

Regular expressions

- $(0|1)^*$ – all binary strings (including ϵ)
- $(01)^*$ – ϵ, 01, 0101, 010101, ...
- $1(0|1)^*1$ – all binary strings starting and ending with 1
- aa^* – all strings of a's (excluding ϵ)

Regular expressions

Adding some extra notation:

- `[abcd]` – any of a, b, c, or d
- `[a-d]` – abbreviation for `[abcd]`

Then we can define:

- `[a-zA-Z][a-zA-Z0-9]`*

Perhaps an expression for variables – must start with a letter

Parsing

How do we go from tokens to some data structure?

characters → lexer → tokens → parser → data structure

Context-free grammar

We can define a *grammar* using Backus-Naur form (BNF):

$$
\begin{aligned}
\langle \mathit{expr} \rangle \quad &::= \langle \mathit{expr} \rangle + \langle \mathit{expr} \rangle \\
&\;\;\mid\;\; \langle \mathit{expr} \rangle \text{ - } \langle \mathit{expr} \rangle \\
&\;\;\mid\;\; \langle \mathit{variable} \rangle \\
&\;\;\mid\;\; \langle \mathit{number} \rangle
\end{aligned}
$$

$$\langle \mathit{variable} \rangle ::= \mathrm{a} \mid \mathrm{b} \mid \mathrm{c} \mid \ldots \mid \mathrm{z}$$

$$\langle \mathit{number} \rangle ::= 1 \mid 2 \mid 3 \mid \ldots \mid 9$$

Parsing

Can we now parse something like "2 + 3"?

$$\begin{aligned} \mathit{expr} &\Rightarrow \mathit{expr} + \mathit{expr} \\ &\Rightarrow 2 + \mathit{expr} \\ &\Rightarrow 2 + 3 \end{aligned}$$

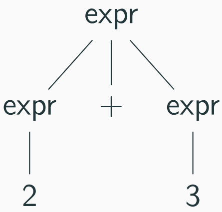
Parsing

Can we now parse something like “9 - 3 - 2”?

$$
\begin{aligned}
\mathit{expr} &\Rightarrow \mathit{expr} - \mathit{expr} \\
&\Rightarrow 9 - \mathit{expr} \\
&\Rightarrow 9 - \mathit{expr} - \mathit{expr} \\
&\Rightarrow 9 - 3 - \mathit{expr} \\
&\Rightarrow 9 - 3 - 2
\end{aligned}
$$

Parse trees

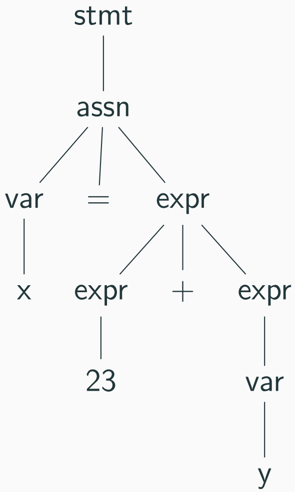
What does the data structure corresponding to "$2 + 3$" look like?



Parsing

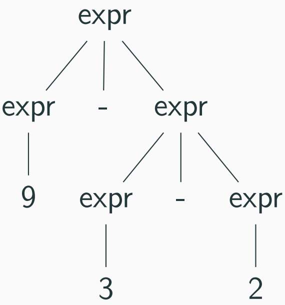
How does a program get read? Going from tokens to a parse tree (assuming a reasonable grammar):

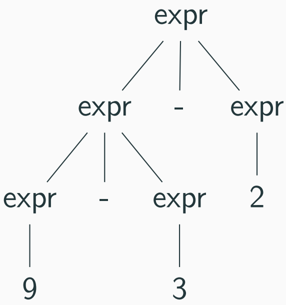
```
x = 23 + y;
```



Parse trees

But the “9 - 3 - 2” example has an issue:





$9 - (3 - 2)$

$(9 - 3) - 2$

Ambiguous grammars

A grammar is ambiguous when, for some example input,

- there's more than one possible parse tree, or
- there's more than one possible derivation (using the same order of expansion)

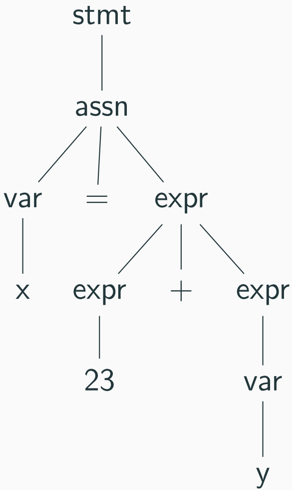
$$\begin{aligned} \mathit{expr} &\Rightarrow \mathit{expr} - \mathit{expr} \\ &\Rightarrow 9 - \mathit{expr} \\ &\Rightarrow 9 - \mathit{expr} - \mathit{expr} \\ &\Rightarrow 9 - 3 - \mathit{expr} \\ &\Rightarrow 9 - 3 - 2 \end{aligned}$$

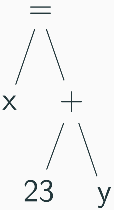
$$\begin{aligned} \mathit{expr} &\Rightarrow \mathit{expr} - \mathit{expr} \\ &\Rightarrow \mathit{expr} - \mathit{expr} - \mathit{expr} \\ &\Rightarrow 9 - \mathit{expr} - \mathit{expr} \\ &\Rightarrow 9 - 3 - \mathit{expr} \\ &\Rightarrow 9 - 3 - 2 \end{aligned}$$

Abstract syntax trees (ASTs)

Parse trees are *concrete*.

But usually we don't care about the full derivation:





Abstract syntax trees (ASTs)

Very different languages may have the same AST!

```
while x <> A[i] do
    i := i + 1
end
```

```
while (x != A[i])
    i = i + 1;
```

Semantics

If syntactically valid, what does the program mean?

Semantics

Not all languages' semantics are as obvious as "if... then...":

$$(\sim R \in R \circ . \times R)\,/R \leftarrow 1 \downarrow \iota R$$

$$\textit{life} \leftarrow \{\uparrow 1\omega \vee . \wedge 3\ 4 = +/,\bar{}1\ 0\ 1 \circ .\theta\ \bar{}1\ 0\ 1 \circ .\phi \subset \omega\}$$

Semantics

Suppose we denote that in state σ, x evaluates to n by $\langle x, \sigma \rangle \to n$.

Then to define the semantics of the '$+$' operator:

$$\frac{\langle x, \sigma \rangle \to n_1 \qquad \langle y, \sigma \rangle \to n_2}{\langle x + y, \sigma \rangle \to n}$$

where n is the sum of n_1 and n_2.

Semantics

Similarly, for "if... then...", we have these two rules:

$$\frac{\langle x, \sigma \rangle \to \text{true} \qquad \langle y, \sigma \rangle \to n_1}{\langle \text{ if } x \text{ then } y \text{ else } z, \sigma \rangle \to n_1}$$

$$\frac{\langle x, \sigma \rangle \to \text{false} \qquad \langle z, \sigma \rangle \to n_2}{\langle \text{ if } x \text{ then } y \text{ else } z, \sigma \rangle \to n_2}$$

Compilers and interpreters

Languages can be executed in a couple of ways:

- compiled
- interpreted