CS 381: Programming Language Fundamentals

Summer 2015

Syntax and Grammars
June 29, 2015

Outline

Syntax and grammars

Representing abstract syntax

Is it abstract or concrete syntax?

Relating abstract syntax to Haskell

What is a language?

Language: a system of communication using "words" in a structured way

Natural language

- used for arbitrary communication
- complex, nuanced, and imprecise

English, Chinese, Hindi, Arabic, Spanish,…

Programming language

- used to describe aspects of computation — i.e. systematic transformation of representation
- programs have a precise **structure** and **meaning**

Haskell, Java, C, Python, SQL, XML, HTML, CSS,…

We use a broad interpretation of "programming language"

Oregon State UNIVERSITY

Syntax vs. semantics

Two main aspects of a **language**:

- **syntax**: the structure of its programs
- **semantics**: the meaning of its programs

Example: well-structured sentences

- **syntax** defines the set of all sentences

How can we define a **syntax**?

1. enumerate all sentences
2. define rules to construct sentences (grammar)

Grammars

Grammars are a **metalanguage** for describing syntax

The language we're defining is called the **object language**

syntactic category

nonterminal symbol

$$
\begin{aligned}
s \in \mathit{Sentence} &::= n\ v\ n \mid s\ \texttt{and}\ s \\
n \in \mathit{Noun} &::= \texttt{cats} \mid \texttt{dogs} \mid \texttt{ducks} \\
v \in \mathit{Verb} &::= \texttt{chase} \mid \texttt{cuddle}
\end{aligned}
$$

terminal symbol

Generating sentences from grammars

How to generate a sentence from a grammar

1. start with a nonterminal ***s***
2. find production rules with ***s*** on the LHS
3. replace ***s*** by one possible RHS case

A sentence is in the language if and only if it can be generated by the grammar!

Animal behavior language

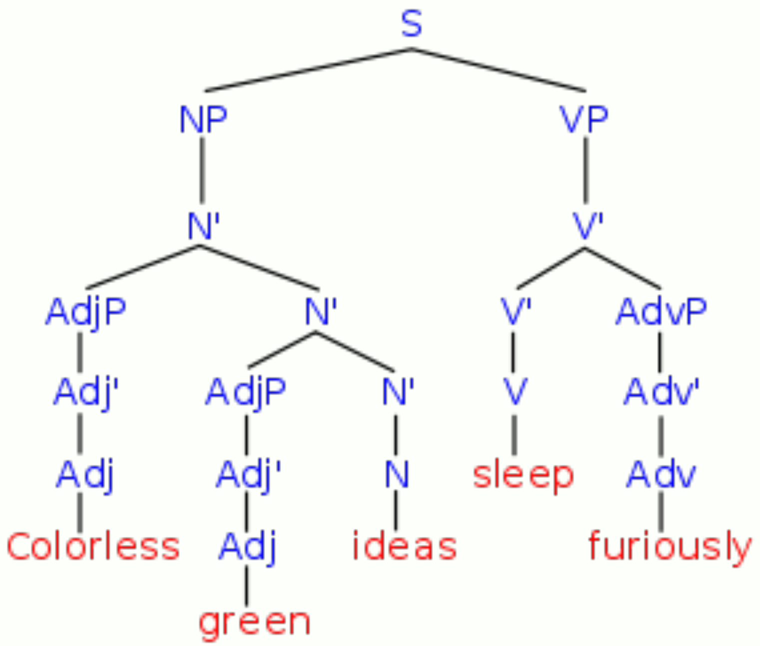
$$
\begin{array}{rcl}
s \in \mathit{Sentence} & ::= & n\ v\ n \mid s\ \texttt{and}\ s \\
n \in \mathit{Noun} & ::= & \texttt{cats} \mid \texttt{dogs} \mid \texttt{ducks} \\
v \in \mathit{Verb} & ::= & \texttt{chase} \mid \texttt{cuddle}
\end{array}
$$

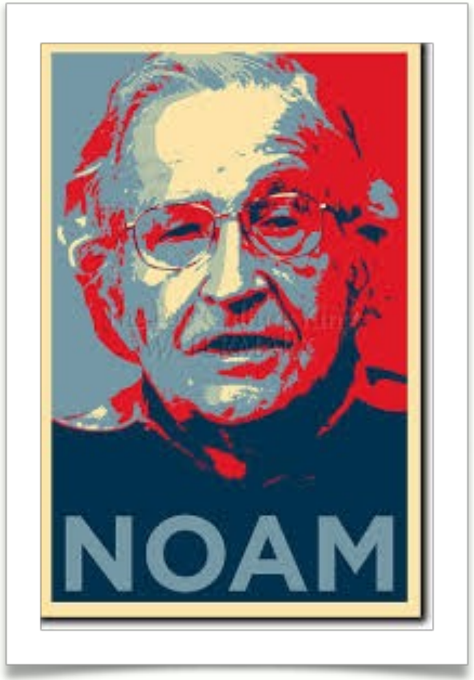
$$
\begin{array}{ll}
 & s \\
\Rightarrow & n\ v\ n \\
\Rightarrow & \texttt{cats}\ v\ n \\
\Rightarrow & \texttt{cats}\ v\ \texttt{ducks} \\
\Rightarrow & \texttt{cats cuddle ducks}
\end{array}
$$

Oregon State UNIVERSITY

Colorless green ideas sleep furiously[1]

Just because a sentence is grammatically correct, doesn't mean it is semantically correct!





[1]Chomsky, Noam. *Syntactic structures*. Walter de Gruyter, 2002.

Derivation order

The order of rule application is *not* fixed

Animal behavior language

$$
\begin{array}{rcll}
s \in \mathit{Sentence} & ::= & n\ v\ n \mid s\ \mathtt{and}\ s & (R1) \\
n \in \mathit{Noun} & ::= & \mathtt{cats} \mid \mathtt{dogs} \mid \mathtt{ducks} & (R2) \\
v \in \mathit{Verb} & ::= & \mathtt{chase} \mid \mathtt{cuddle} & (R3)
\end{array}
$$

$$
\begin{array}{lll}
 & s & \text{R1} \\
\Rightarrow & n\ v\ n & \text{R2} \\
\Rightarrow & \mathtt{cats}\ v\ n & \text{R2} \\
\Rightarrow & \mathtt{cats}\ v\ \mathtt{ducks} & \text{R3} \\
\Rightarrow & \mathtt{cats}\ \mathtt{cuddle}\ \mathtt{ducks} &
\end{array}
$$

$$
\begin{array}{lll}
 & s & \text{R1} \\
\Rightarrow & n\ v\ n & \text{R3} \\
\Rightarrow & n\ \mathtt{cuddle}\ n & \text{R2} \\
\Rightarrow & \mathtt{cats}\ \mathtt{cuddle}\ n & \text{R2} \\
\Rightarrow & \mathtt{cats}\ \mathtt{cuddle}\ \mathtt{ducks} &
\end{array}
$$

Oregon State UNIVERSITY

Exercise: animal behavior language

Animal behavior language

$$
\begin{aligned}
s \in \mathit{Sentence} &::= n\ v\ n \mid s\ \texttt{and}\ s \\
n \in \mathit{Noun} &::= \texttt{cats} \mid \texttt{dogs} \mid \texttt{ducks} \\
v \in \mathit{Verb} &::= \texttt{chase} \mid \texttt{cuddle}
\end{aligned}
$$

Which of the following sentences are well defined in the animal behavior language?

- `cats chase dogs` Yes
- `cats and dogs chase ducks` No
- `dogs cuddle cats and ducks chase dogs` Yes
- `dogs chase cats and cats chase ducks and ducks chase dogs` Yes

Exercise: boolean language

Write a grammar for boolean expressions built from the terms **`true`** and **`false`** and the logical operation **`not`**

Boolean language

$$
\begin{array}{llll}
t \in \mathit{Term} & ::= & \mathbf{true} & (R1) \\
 & | & \mathbf{false} & (R2) \\
 & | & \mathbf{not}\ t & (R3)
\end{array}
$$

Derive the sentence **`not not false`**

$$
\begin{array}{lll}
 & t & \\
\Rightarrow & t\ t\ t & \mathbf{R3} \\
\Rightarrow & t\ \mathbf{not}\ t & \mathbf{R2} \\
\Rightarrow & t\ \mathbf{not}\ \mathbf{false} & \mathbf{R3} \\
\Rightarrow & \mathbf{not}\ \mathbf{not}\ \mathbf{false} &
\end{array}
$$

Outline

Syntax and grammars

Representing abstract syntax

Is it abstract or concrete syntax?

Relating abstract syntax to Haskell

Abstract syntax trees — ASTs

Syntax tree: a structure to *represent* derivations

Derivation: a process of producing a sentence according to the rules of a grammar

Grammar (BNF notation)

$$
\begin{array}{lcl}
t \in \mathit{Term} & ::= & \textbf{true} \\
& | & \textbf{false} \\
& | & \textbf{not}\ t \\
& | & \textbf{if}\ t\ t\ t
\end{array}
$$

Example ASTs

```
                 if             not
true           / | \             |
           true false true      not
                                 |
                               false
```

Language generated by grammar: set of all ASTs

$$
\mathit{Term} = \{\textbf{true}, \textbf{false}\} \cup \left\{ \begin{array}{c} \texttt{not} \\ | \\ t \end{array} \;\middle|\; t \in \mathit{Term} \right\} \cup \left\{ \begin{array}{ccc} & \texttt{if} & \\ / & | & \backslash \\ t_1 & t_2 & t_3 \end{array} \;\middle|\; t_1, t_2, t_3 \in \mathit{Term} \right\}
$$

Programs are trees!

Abstract syntax tree (AST): captures the essential structure of a program

- everything needed to determine its semantics

```
    +
   / \
  2   ×
     / \
    3   4
```

```
       ×
     /   \
    +     +
   / \   / \
  5   6 7    8
```

```
        if
      /  |  \
  true   +   5
        / \
       2   3
```

`2 + 3 x 4`

`(5 + 6) x (7 + 8)`

`if true then (2 + 3) else 5`

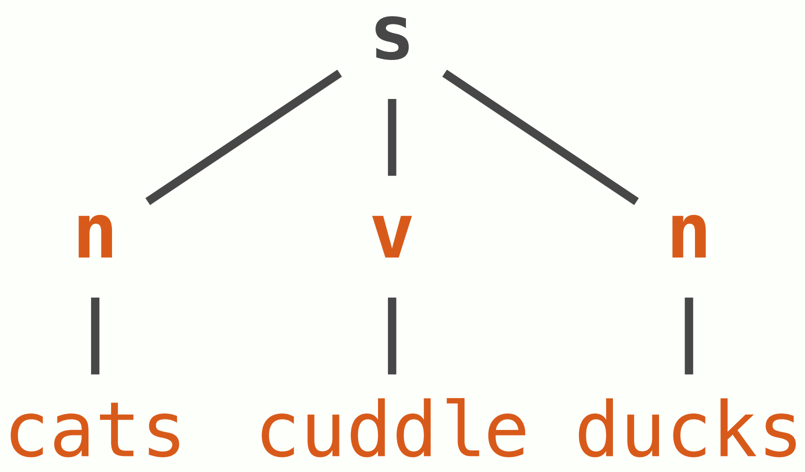
Observations about ASTs

Leaves contain *terminal symbols*

Internal nodes contain ***nonterminal symbols***

Nonterminal in the root node indicates the ***type*** of the syntax tree

Derivation order is *not represented* — which is a good thing, because it is *not important*



Outline

Syntax and grammars

Representing abstract syntax

Is it abstract or concrete syntax?

Relating abstract syntax to Haskell

Abstract syntax vs. concrete syntax

Abstract syntax: captures the **essential structure** of programs

Concrete syntax: describes how programs are written down (**linear representation**)

Abstract grammar

$$t \in \mathit{Term} \ ::= \ \mathtt{true} \mid \mathtt{false} \mid \mathtt{not}\ t \mid \mathtt{if}\ t\ t\ t$$

Concrete grammar

$$t \in \mathit{Term} \ ::= \ \mathtt{true} \mid \mathtt{false} \mid \mathtt{not}\ t \mid \mathtt{if}\ t\ \mathtt{then}\ t\ \mathtt{else}\ t \mid (t)$$

We will focus on **abstract syntax** — always constructing **trees**

- use parentheses to disambiguate linear representations of ASTs

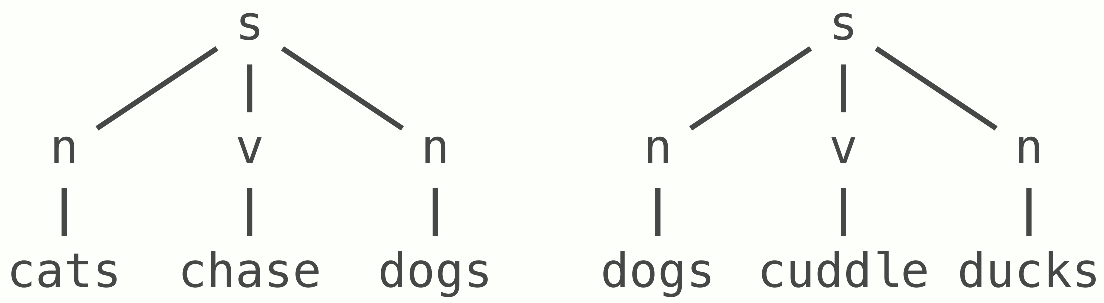
Is it abstract or concrete syntax?

Example: animal behavior language

Animal behavior language

$$
\begin{aligned}
s \in \mathit{Sentence} &::= n\ v\ n \mid s\ \texttt{and}\ s \\
n \in \mathit{Noun} &::= \texttt{cats} \mid \texttt{dogs} \mid \texttt{ducks} \\
v \in \mathit{Verb} &::= \texttt{chase} \mid \texttt{cuddle}
\end{aligned}
$$

abstract syntax → set of *syntax trees*



concrete syntax ↓

set of *sentences/strings* **(linear)**

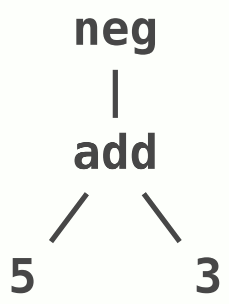
$\mathit{Sentence} = \{$**`cats chase dogs, dogs cuddle ducks,…`**$\}$

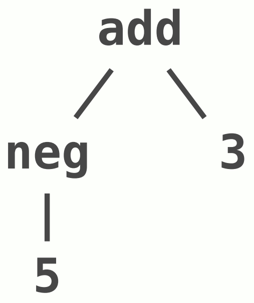
Is it abstract or concrete syntax?

Oregon State UNIVERSITY

Exercise: arithmetic expression language

1. Draw two different ASTs for the expression: `2 + 3 x 4`
2. Draw an AST for the expression: `–5 x (6 + 7)`
3. What are the integer ***results*** of evaluating the following ASTs?





–8

–2

Arithmetic expression language

$$
\begin{array}{rcl}
i \in \mathit{Int} & ::= & 1 \mid 2 \mid \ldots \\
e \in \mathit{Expr} & ::= & \texttt{add}\ e\ e \\
& \mid & \texttt{mul}\ e\ e \\
& \mid & \texttt{neg}\ e \\
& \mid & i
\end{array}
$$

Is it abstract or concrete syntax?

Outline

Syntax and grammars

Representing abstract syntax

Is it abstract or concrete syntax?

Relating abstract syntax to Haskell

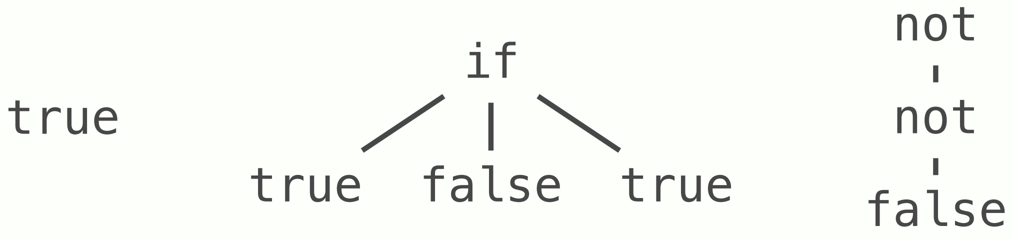
Encoding abstract syntax in Haskell

Abstract grammar

$$
\begin{array}{lcl}
b \in \mathit{Bool} & ::= & \mathbf{true} \mid \mathbf{false} \\
t \in \mathit{Term} & ::= & \mathbf{not}\ t \\
 & \mid & \mathbf{if}\ t\ t\ t \\
 & \mid & b
\end{array}
$$

defines set

Abstract syntax trees



Haskell data type definition

```
data Bool = True | False

data Term = Not Term
          | If Term Term Term
          | Lit Bool
```

defines set

Haskell values

```
Lit True
If (Lit True)
   (Lit False)
   (Lit True)
Not (Not (Lit False))
```

linear encoding

Oregon State University

Translating grammars into Haskell data types

StackLang.hs

Strategy: grammar $\rightarrow$ Haskell

1. For each basic nonterminal, choose a built-in type, e.g. **`Int`**, **`Bool`**
2. For each other nonterminal, define a data type
3. For each production rule, define a data constructor
4. The nonterminals in the production rules determine the arguments to the constructor

Special rule for lists:

- in grammars, $s ::= t^*$ is shorthand for: $s ::= \varepsilon \mid t\ s$ **or** $s ::= \varepsilon \mid t, s$
- can translate any of these to a Haskell list:

```
data Term = ...
type Sentence = [Term]
```

Example: annotated arithmetic expression language

Let.hs

Abstract syntax

$$
\begin{array}{rcll}
n \in \mathit{Nat} & ::= & \texttt{(natural number)} & \\
c \in \mathit{Comm} & ::= & \texttt{(comment string)} & \\
e \in \mathit{Expr} & ::= & \mathbf{neg}\ e & \texttt{negation} \\
& \mid & e\ @\ c & \texttt{comment} \\
& \mid & e + e & \texttt{addition} \\
& \mid & e * e & \texttt{multiplication} \\
& \mid & n & \texttt{literal}
\end{array}
$$

Haskell encoding

```
type Comment = String

data Expr = Neg Expr
          | Annot Comment Expr
          | Add Expr Expr
          | Mul Expr Expr
          | Lit Int
```