Syntax and Grammars
Jul 6, 2017
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

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

We use a broad interpretation of “programming language”
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.

\[
\begin{align*}
    s \in \text{Sentence} & \quad ::= \quad n \, \lor \, n \mid s \, \land \, s \\
    n \in \text{Noun} & \quad ::= \quad \text{cats} \mid \text{dogs} \mid \text{ducks} \\
    v \in \text{Verb} & \quad ::= \quad \text{chase} \mid \text{cuddle}
\end{align*}
\]
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

$S \in \text{Sentence} ::= n \text{ v } n \mid s \text{ and } s$
$n \in \text{Noun} ::= \text{cats } \mid \text{dogs } \mid \text{ducks}$
$v \in \text{Verb} ::= \text{chase } \mid \text{cuddle}$

$s \Rightarrow n \text{ v } n$
$s \Rightarrow \text{cats } v \text{ n}$
$s \Rightarrow \text{cats } v \text{ ducks}$
$s \Rightarrow \text{cats cuddle ducks}$
Colorless green ideas sleep furiously¹

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

Derivation order

The order of rule application is not fixed

Animal behavior language

\[
\begin{align*}
S \in \text{Sentence} & ::= n \lor n \mid s \land s & (R1) \\
n \in \text{Noun} & ::= \text{cats} \mid \text{dogs} \mid \text{ducks} & (R2) \\
v \in \text{Verb} & ::= \text{chase} \mid \text{cuddle} & (R3)
\end{align*}
\]

\[
\begin{align*}
S & \quad R1 \\
\Rightarrow n \lor n & \quad R2 \\
\Rightarrow \text{cats} \lor n & \quad R2 \\
\Rightarrow \text{cats} \lor \text{ducks} & \quad R3 \\
\Rightarrow \text{cats} \lor \text{cuddle ducks} & \quad R3
\end{align*}
\]

\[
\begin{align*}
S & \quad R1 \\
\Rightarrow n \lor n & \quad R2 \\
\Rightarrow n \lor \text{cuddle} n & \quad R2 \\
\Rightarrow \text{cats} \lor \text{cuddle} n & \quad R2 \\
\Rightarrow \text{cats} \lor \text{cuddle ducks} & \quad R2
\end{align*}
\]
Exercise: animal behavior language

Animal behavior language

\[ s \in \text{Sentence} ::= n \; v \; n \mid s \; \text{and} \; s \]
\[ n \in \text{Noun} ::= \text{cats} \mid \text{dogs} \mid \text{ducks} \]
\[ v \in \text{Verb} ::= \text{chase} \mid \text{cuddle} \]

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

\[ t \in \text{Term} ::= \text{true} \quad (R1) \]
\[ | \quad \text{false} \quad (R2) \]
\[ | \quad \text{not } t \quad (R3) \]

Derive the sentence not not false

\[
\begin{align*}
t & \\
\Rightarrow & t \ t \ t \quad \text{R3} \\
\Rightarrow & t \ \text{not } \ t \quad \text{R2} \\
\Rightarrow & t \ \text{not false} \quad \text{R3} \\
\Rightarrow & \text{not not false}
\end{align*}
\]
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)

\[
t \in \text{Term} ::= \text{true} \mid \text{false} \mid \text{not } t \mid \text{if } t \ t \ t
\]

Example ASTs

![Example ASTs](image)

Language generated by grammar: set of all ASTs

\[
\text{Term} = \{\text{true, false}\} \cup \{t \mid t \in \text{Term}\} \cup \{\text{if } t_1 \ t_2 \ t_3 \mid t_1, t_2, t_3 \in \text{Term}\}
\]
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 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
\[ t \in \text{Term} ::= \begin{array}{l}
\text{true} \\
\text{false} \\
\text{not } t \\
\text{if } t \text{ } t \text{ } t
\end{array} \]

Concrete grammar
\[ t \in \text{Term} ::= \begin{array}{l}
\text{true} \\
\text{false} \\
\text{not } t \\
\text{if } t \text{ then } t \text{ else } t \\
(t)
\end{array} \]

We will focus on abstract syntax — always constructing trees
  • use parentheses to disambiguate linear representations of ASTs
Example: animal behavior language

Animal behavior language

\[ s \in \text{Sentence} ::= n \; v \; n \mid s \; \text{and} \; s \]
\[ n \in \text{Noun} ::= \text{cats} \mid \text{dogs} \mid \text{ducks} \]
\[ v \in \text{Verb} ::= \text{chase} \mid \text{cuddle} \]

\[ \text{Sentence} = \{ \text{cats chase dogs}, \text{dogs cuddle ducks}, \ldots \} \]
Exercise: arithmetic expression language

1. Draw two different ASTs for the expression: \(2 + 3 \times 4\)

2. Draw an AST for the expression: \(-5 \times (6 + 7)\)

3. What are the integer results of evaluating the following ASTs?

```
Arithmetic expression language

\[\begin{align*}
i \in Int & ::= 1 \mid 2 \mid \ldots \\
e \in Expr & ::= add \ e \ e \\
 & \mid mul \ e \ e \\
 & \mid neg \ e \\
& \mid i
\end{align*}\]
```

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
\[ b \in \text{Bool} ::= \text{true} \mid \text{false} \]
\[ t \in \text{Term} ::= \text{not} \ t \mid \text{if} \ t \ t \ t \mid b \]

Haskell data type definition
```
data Bool = True | False
```
```
data Term = Not Term |
  | If Term Term Term |
  | Lit Bool
```

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

Abstract syntax trees
```
true
   / \
  true false true
```
```
not
```
```
```
```
Translating grammars into Haskell data types

Strategy: grammar → 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: \( s ::= \epsilon \mid t \cdot s \) or \( s ::= \epsilon \mid t, s \)
• can translate any of these to a Haskell list:

```
data Term = ...
type Sentence = [Term]
```
Example: annotated arithmetic expression language

Abstract syntax

| $n \in Nat$  | ::= (natural number) |
| $c \in Comm$ | ::= (comment string) |
| $e \in Expr$ | ::= $\text{neg } e$  negation |
|             | $\mid e \@ c$    comment |
|             | $\mid e + e$   addition |
|             | $\mid e \times e$ multiplication |
|             | $\mid n$        literal |

Haskell encoding

```haskell
type Comment = String

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

Let.hs
Pretty printing

A **pretty printer** creates a string from a syntax tree.

A **parser** extracts a syntax tree from a string.

\[
\text{cond} ::= \ T \mid \text{not cond} \mid (\text{cond}) \\
\text{stmt} ::= \text{while cond} \{ \text{stmt} \} \mid \text{noop}
\]

\[
\text{data Cond} = \ T \mid \text{Not Cond} \\
\text{data Stmt} = \text{While Cond Stmt} \mid \text{Noop}
\]

while not(not(T)) {
  while T {noop}
}

**pretty printer**

While (Not (Not T))
  (While T Noop)

**parser**
Exercise: statement pretty printer

1. Define a pretty printer for the statement abstract syntax that produces output based on its grammar.

```
data Cond = T | Not Cond
data Stmt = While Cond Stmt | Noop
```

```
cond ::= T | not cond | (cond)
stmt ::= while cond { stmt } | noop
```