

## Levels of Syntax

**Concrete syntax.**

- The **surface structure** of a language.
- Representation of phrases as strings.
- Concerned with readability, ambiguity.

1

## Levels of Syntax

**Abstract syntax.**

- The **deep structure** of a language.
- Representation of phrases as trees (terms).
- Concerned with fundamental structure.

2

## A Bare Bones Start

We'll start by studying  $\mathcal{L}\{\text{num str}\}$ , a very simple language of expressions.

- Natural numbers and strings.
- Variables, simple operations, binding.

3

## Specifying Concrete Syntax

Concrete syntax is an inductively defined set of **strings**.

- Lexical Level
  - Alphabet of **tokens** (e.g., identifiers, numbers).
  - Described by regular expressions or inductive judgements
- Grammatical Level
  - Language phrases made up of tokens
  - Described by context-free grammar notation (aka BNF).

4

## Context-Free Grammars

A **context-free grammar** consists of three things:

1. An alphabet  $\Sigma$  of **terminals**, or **symbols**.
2. A finite set  $\mathcal{N}$  of **non-terminals**, or **categories**.
3. A finite set of **productions** of the form  $A ::= \alpha$ , where  $A \in \mathcal{N}$  and  $\alpha \in (\Sigma \cup \mathcal{N})^*$ .

5

## Lexical Structure

First phase of syntactic processing, called **lexical analysis** or **lexing**, is converting stream of characters to tokens. For  $\mathcal{L}\{\text{num str}\}$  this involves:

1. Identifying numbers and strings
2. Identifying keywords and operators
3. Discarding "white space" (spaces, tabs, comments, etc.)

6

## Lexical Structure for $\mathcal{L}\{\text{num str}\}$

Item  $\text{itm} ::= \text{kwd} \mid \text{id} \mid \text{num} \mid \text{lit} \mid \text{spl}$   
 Keyword  $\text{kwd} ::= \text{l} \cdot \text{e} \cdot \text{t} \cdot \text{e} \mid \text{b} \cdot \text{e} \cdot \text{e} \mid \text{i} \cdot \text{n} \cdot \text{e}$   
 Identifier  $\text{id} ::= \text{ltr}(\text{ltr} \mid \text{dig})^*$   
 Numeral  $\text{num} ::= \text{dig} \text{dig}^*$   
 Literal  $\text{lit} ::= \text{qum}(\text{ltr} \mid \text{dig})^* \text{qum}$   
 Special  $\text{spl} ::= + \mid * \mid ^ \mid ( \mid ) \mid |$

7

## Lexical Structure for $\mathcal{L}\{\text{num str}\}$ (cont.)

Letter  $\text{ltr} ::= \text{a} \mid \text{b} \mid \dots \mid \text{z}$   
 Digit  $\text{dig} ::= 0 \mid 1 \mid \dots \mid 9$   
 Quote  $\text{qum} ::= "$

8

## Grammars as Inductive Definitions

A grammar is a **simultaneous inductive definition** of several sets of strings.

- Each category determines a set of strings.
- Each production determines a rule.

9

## Inductive Definition of Tokens

$\frac{s \text{ str}}{\text{ID}[s] \text{ tok}}$      $\frac{n \text{ nat}}{\text{NUM}[n] \text{ tok}}$      $\frac{s \text{ str}}{\text{LIT}[s] \text{ tok}}$   
 $\overline{\text{LET tok}}$      $\overline{\text{BE tok}}$      $\overline{\text{IN tok}}$   
 $\overline{\text{ADD tok}}$      $\overline{\text{MUL tok}}$      $\overline{\text{CAT tok}}$   
 $\overline{\text{LP tok}}$      $\overline{\text{RP tok}}$      $\overline{\text{VB tok}}$

10

## Lexical Analysis Judgements

$s \text{ inp} \longleftrightarrow t \text{ tokstr}$     Scan input  
 $s \text{ itm} \longleftrightarrow t \text{ tok}$     Scan an item  
 $s \text{ kwd} \longleftrightarrow t \text{ tok}$     Scan a keyword  
 $s \text{ id} \longleftrightarrow t \text{ tok}$     Scan an identifier  
 $s \text{ num} \longleftrightarrow t \text{ tok}$     Scan a number  
 $s \text{ spl} \longleftrightarrow t \text{ tok}$     Scan a symbol  
 $s \text{ lit} \longleftrightarrow t \text{ tok}$     Scan a string literal  
 $s \text{ whs}$     Skip whitespace

11

### Inductive Definition of Lexical Analysis

$$\begin{array}{l} \frac{}{\varepsilon \text{ inp} \longleftrightarrow \varepsilon \text{ tokstr}} \\ \frac{s = s_1 \wedge s_2 \wedge s_3 \quad s_1 \text{ whs} \quad s_2 \text{ itm} \longleftrightarrow t \text{ tok} \quad s_3 \text{ inp} \longleftrightarrow ts \text{ tokstr}}{s \text{ inp} \longleftrightarrow t \cdot ts \text{ tokstr}} \\ \frac{s \text{ kwd} \longleftrightarrow t \text{ tok}}{s \text{ itm} \longleftrightarrow t \text{ tok}} \quad \frac{s \text{ id} \longleftrightarrow t \text{ tok}}{s \text{ itm} \longleftrightarrow t \text{ tok}} \quad \frac{s \text{ num} \longleftrightarrow t \text{ tok}}{s \text{ itm} \longleftrightarrow t \text{ tok}} \\ \frac{s \text{ lit} \longleftrightarrow t \text{ tok}}{s \text{ itm} \longleftrightarrow t \text{ tok}} \quad \frac{s \text{ spl} \longleftrightarrow t \text{ tok}}{s \text{ itm} \longleftrightarrow t \text{ tok}} \\ \frac{s = l \cdot e \cdot t \cdot \varepsilon \text{ str}}{s \text{ kwd} \longleftrightarrow \text{LET tok}} \quad \frac{s = b \cdot e \cdot \varepsilon \text{ str}}{s \text{ kwd} \longleftrightarrow \text{BE tok}} \quad \frac{s = i \cdot n \cdot \varepsilon \text{ str}}{s \text{ kwd} \longleftrightarrow \text{IN tok}} \end{array}$$

12

### Inductive Definition of Lexical Analysis (cont.)

$$\begin{array}{l} \frac{s = s_1 \wedge s_2 \quad s_1 \text{ ltr} \quad s_2 \text{ lord} \quad (\text{and } s \text{ not a keyword})}{s \text{ id} \longleftrightarrow \text{ID}[s] \text{ tok}} \\ \frac{s = s_1 \wedge s_2 \quad s_1 \text{ dig} \quad s_2 \text{ dgs} \quad s \text{ num} \longleftrightarrow n \text{ nat}}{s \text{ num} \longleftrightarrow \text{NUM}[n] \text{ tok}} \\ \frac{s = s_1 \wedge s_2 \wedge s_3 \quad s_1 \text{ qum} \quad s_2 \text{ lord} \quad s_3 \text{ qum}}{s \text{ lit} \longleftrightarrow \text{LIT}[s_2] \text{ tok}} \\ \frac{s = + \cdot \varepsilon \text{ str}}{s \text{ spl} \longleftrightarrow \text{ADD tok}} \quad \frac{s = * \cdot \varepsilon \text{ str}}{s \text{ spl} \longleftrightarrow \text{MUL tok}} \quad \frac{s = \wedge \cdot \varepsilon \text{ str}}{s \text{ spl} \longleftrightarrow \text{CAT tok}} \\ \frac{s = ( \cdot \varepsilon \text{ str}}{s \text{ spl} \longleftrightarrow \text{LP tok}} \quad \frac{s = ) \cdot \varepsilon \text{ str}}{s \text{ spl} \longleftrightarrow \text{RP tok}} \quad \frac{s = | \cdot \varepsilon \text{ str}}{s \text{ spl} \longleftrightarrow \text{VB tok}} \end{array}$$

13

### Grammatical Structure for $\mathcal{L}\{\text{num str}\}$

$$\begin{array}{l} \text{Exp} \quad \text{exp} ::= \text{num} \mid \text{lit} \mid \text{id} \mid \text{LP exp RP} \mid \text{exp ADD exp} \mid \\ \quad \text{exp MUL exp} \mid \text{exp CAT exp} \mid \text{VB exp VB} \\ \quad \text{LET id BE exp IN exp} \\ \text{Number} \quad \text{num} ::= \text{NUM}[n] \quad (n \text{ nat}) \\ \text{String} \quad \text{lit} ::= \text{LIT}[s] \quad (s \text{ str}) \\ \text{Identifier} \quad \text{id} ::= \text{ID}[s] \quad (s \text{ str}) \end{array}$$

14

### Inductive Definition of Grammar for $\mathcal{L}\{\text{num str}\}$

$$\begin{array}{l} \frac{s \text{ num}}{s \text{ exp}} \quad \frac{s \text{ lit}}{s \text{ exp}} \quad \frac{s \text{ id}}{s \text{ exp}} \\ \frac{s_1 \text{ exp} \quad s_2 \text{ exp}}{s_1 \text{ ADD } s_2 \text{ exp}} \quad \frac{s_1 \text{ exp} \quad s_2 \text{ exp}}{s_1 \text{ MUL } s_2 \text{ exp}} \quad \frac{s_1 \text{ exp} \quad s_2 \text{ exp}}{s_1 \text{ CAT } s_2 \text{ exp}} \\ \frac{s \text{ exp}}{\text{VB } s \text{ VB exp}} \quad \frac{s \text{ exp}}{\text{LP } s \text{ RP exp}} \\ \frac{s_1 \text{ id} \quad s_2 \text{ exp} \quad s_3 \text{ exp}}{\text{LET } s_1 \text{ BE } s_2 \text{ IN } s_3 \text{ exp}} \\ \frac{n \text{ nat}}{\text{NUM}[n] \text{ num}} \quad \frac{s \text{ str}}{\text{LIT}[s] \text{ lit}} \quad \frac{s \text{ str}}{\text{ID}[s] \text{ id}} \end{array}$$

15

### Ambiguity

This grammar is **ambiguous**: the same string arises in different ways!

The string  $1+2*3$  may be thought of in two ways:

- $e \rightarrow e_1 * e_2 \rightarrow e_{1,1} + e_{1,2} * e_2 \rightarrow^* 1+2*3.$
- $e \rightarrow e_1 + e_2 \rightarrow e_1 + e_{2,1} * e_{2,2} \rightarrow^* 1+2*3.$

**You cannot tell by looking at the string which derivation was used!**

16

### Resolving Ambiguity

Introduce precedence conventions and parenthesization so that

- Every string has a **unique** derivation.
- User can override defaults by grouping.

The decompositions are unique, so expression evaluation is well-defined.

17

### Resolving Ambiguity

Factor  $fct ::= num \mid lit \mid id \mid LP\ prg\ RP$   
Term  $trm ::= fct \mid fct\ MUL\ trm \mid VB\ fct\ VB$   
Expression  $exp ::= trm \mid trm\ ADD\ exp \mid trm\ CAT\ exp$   
Program  $prg ::= exp \mid LET\ id\ BE\ exp\ IN\ prg$

18

### Resolving Ambiguity

Each well-formed string is derived in exactly one way.

- Heavy “layering” of grammar.
- Introduction of additional notation.

19

### Abstract Syntax

Even once ambiguity is resolved, it's not very practical to use concrete syntax for studying or processing programming languages.

- Must analyze strings according to the grammar to create complicated representation that indirectly reveals meaning

20

### Abstract Syntax

Better idea: Separate the **deep structure** from the **surface syntax**.

- Surface syntax: human-oriented, string-representation.
- Deep structure: machine-oriented, tree (term) representation.

The deep structure reveals meaning (e.g., “this is an addition”) directly, rather than by precedence conventions.

21

### Syntax Analysis, or Parsing

A job of a **parser** is to analyse the surface syntax to determine the deep structure of an expression.

- Take apart the string once and for all.
- Translate to abstract syntax to reveal structure.

The abstract syntax of a language is an inductively-defined set of terms.

22

### Abstract Syntax Tree Signature for $\mathcal{L}_{\{num\ str\}}$

$ar(num[n]) = 0$  ( $n\ nat$ )

$ar(str[s]) = 0$  ( $s\ str$ )

$ar(id[s]) = 0$  ( $s\ str$ )

$ar(plus) = 2$

$ar(times) = 2$

$ar(cat) = 2$

$ar(len) = 1$

$ar(let[s]) = 2$  [identifier-indexed family of operators]

23

### Inductive Definition of Abstract Syntax for $\mathcal{L}\{\text{num str}\}$

$$\frac{n \text{ nat}}{\text{num}[n] \text{ ast}} \quad \frac{s \text{ str}}{\text{str}[s] \text{ ast}} \quad \frac{s \text{ str}}{\text{id}[s] \text{ ast}}$$

$$\frac{a_1 \text{ ast} \quad a_2 \text{ ast}}{\text{plus}(a_1; a_2) \text{ ast}} \quad \frac{a_1 \text{ ast} \quad a_2 \text{ ast}}{\text{times}(a_1; a_2) \text{ ast}}$$

$$\frac{a_1 \text{ ast} \quad a_2 \text{ ast}}{\text{cat}(a_1; a_2) \text{ ast}} \quad \frac{a \text{ ast}}{\text{len}(a) \text{ ast}}$$

$$\frac{s \text{ id} \quad a_1 \text{ ast} \quad a_2 \text{ ast}}{\text{let}[s](a_1; a_2) \text{ ast}}$$

24

### Syntax Analysis, or Parsing

A term wears its structure on its sleeve! There is no ambiguity when decomposing a term into its parts.

- The operators are **injective**, meaning that we can recover the sub-terms from the term.
- String concatenation is **non-injective**: many pairs of strings map to the same string.
- Notice that the abstract syntax is **independent** from the particular grammar used to describe the concrete syntax

25

### Terminology

The terms representing abstract syntax are sometimes called

- **abstract syntax trees**, or **ast**'s, emphasizing the tree structure.
- **parse trees**, reflecting the result of a parse (syntactic analysis).

26

### Parsing

A parser is a function mapping concrete to abstract syntax.

- Use a well-structured grammar (in several senses).
  - At least unambiguous – or parsing won't be a function!
- Analyze and decompose strings using one of several methods.
  - Top-down parsers (recursive descent): §630.
  - Bottom-up parsers (LR parsers): §610.

27

### Parsing Judgements for $\mathcal{L}\{\text{num str}\}$

$s \text{ prg} \longleftrightarrow a \text{ ast}$	Parse as a program
$s \text{ exp} \longleftrightarrow a \text{ ast}$	Parse as an expression
$s \text{ trm} \longleftrightarrow a \text{ ast}$	Parse as a term
$s \text{ fct} \longleftrightarrow a \text{ ast}$	Parse as a factor
$s \text{ num} \longleftrightarrow a \text{ ast}$	Parse as a number
$s \text{ lit} \longleftrightarrow a \text{ ast}$	Parse as a literal
$s \text{ id} \longleftrightarrow a \text{ ast}$	Parse as an identifier

28

### Inductive Definition of Parsing for $\mathcal{L}\{\text{num str}\}$

$$\frac{n \text{ nat}}{\text{NUM}[n] \text{ num} \longleftrightarrow \text{num}[n] \text{ ast}} \quad \frac{s \text{ str}}{\text{LIT}[s] \text{ lit} \longleftrightarrow \text{str}[s] \text{ ast}}$$

$$\frac{s \text{ str}}{\text{ID}[s] \text{ id} \longleftrightarrow \text{id}[s] \text{ ast}}$$

$$\frac{s \text{ num} \longleftrightarrow a \text{ ast}}{s \text{ fct} \longleftrightarrow a \text{ ast}} \quad \frac{s \text{ lit} \longleftrightarrow a \text{ ast}}{s \text{ fct} \longleftrightarrow a \text{ ast}} \quad \frac{s \text{ id} \longleftrightarrow a \text{ ast}}{s \text{ fct} \longleftrightarrow a \text{ ast}}$$

$$\frac{s \text{ prg} \longleftrightarrow a \text{ ast}}{\text{LP s RP fct} \longleftrightarrow a \text{ ast}} \quad \frac{s \text{ fct} \longleftrightarrow a \text{ ast}}{s \text{ trm} \longleftrightarrow a \text{ ast}}$$

$$\frac{s_1 \text{ fct} \longleftrightarrow a_1 \text{ ast} \quad s_2 \text{ trm} \longleftrightarrow a_2 \text{ ast}}{s_1 \text{ MUL } s_2 \text{ trm} \longleftrightarrow \text{times}(a_1; a_2) \text{ ast}} \quad \frac{s \text{ fct} \longleftrightarrow a \text{ ast}}{\text{VB s VB trm} \longleftrightarrow \text{len}(a) \text{ ast}}$$

29

### Inductive Definition of Parsing (cont.)

$$\frac{s \text{ trm} \longleftrightarrow a \text{ ast}}{s \text{ exp} \longleftrightarrow a \text{ ast}} \quad \frac{s_1 \text{ trm} \longleftrightarrow a_1 \text{ ast} \quad s_2 \text{ exp} \longleftrightarrow a_2 \text{ ast}}{s_1 \text{ ADD } s_2 \text{ exp} \longleftrightarrow \text{plus}(a_1; a_2) \text{ ast}}$$

$$\frac{s_1 \text{ trm} \longleftrightarrow a_1 \text{ ast} \quad s_2 \text{ exp} \longleftrightarrow a_2 \text{ ast}}{s_1 \text{ CAT } s_2 \text{ exp} \longleftrightarrow \text{cat}(a_1; a_2) \text{ ast}} \quad \frac{s \text{ exp} \longleftrightarrow a \text{ ast}}{s \text{ prg} \longleftrightarrow a \text{ ast}}$$

$$\frac{s_1 \text{ id} \longleftrightarrow \text{id}[s] \text{ ast} \quad s_2 \text{ exp} \longleftrightarrow a_2 \text{ ast} \quad s_3 \text{ prg} \longleftrightarrow a_3 \text{ ast}}{\text{LET } s_1 \text{ BE } s_2 \text{ IN } s_3 \text{ prg} \longleftrightarrow \text{let}[s](a_2; a_3) \text{ ast}}$$

30

### Inductive Definition of Parsing

A successful parse implies that the token string must have been derived according to an unambiguous grammar and the result is a well-formed abstract syntax tree.

#### Theorem 1 (Parsing Judgements)

If  $s \text{ prg} \longleftrightarrow a \text{ ast}$ , then  $s \text{ prg}$  and  $a \text{ ast}$  and similarly for the other parsing judgements.

**Proof:** The proof proceeds by rule induction on the rules for the inductive definition of parsing. ■

31

### Inductive Definition of Parsing

Moreover, if a string is generated according to the rules of the grammar, then it will parse to a unique abstract syntax tree.

#### Theorem 2 (Unique Parse)

If  $s \text{ prg}$ , then there is a unique  $a$  such that  $s \text{ prg} \longleftrightarrow a \text{ ast}$ , and similarly for the other parsing judgements. That is, the parsing judgements have mode  $(\forall, \exists!)$  over the class of well-formed strings and abstract syntax trees.

**Proof:** The proof proceeds by rule induction on the rules corresponding to the unambiguous grammar for  $\mathcal{L}\{\text{num str}\}$ . ■

32

### Inductive Definition of Parsing

And conversely, an abstract syntax tree can be un-parsed into a string that would parse back to the abstract syntax tree.

#### Theorem 3 (Pretty Printing)

If  $a \text{ ast}$ , then there exists a (not necessarily unique) string  $s$  such that  $s \text{ prg}$  and  $s \text{ prg} \longleftrightarrow a \text{ ast}$ . That is, the parsing judgement has mode  $(\exists, \forall)$ .

**Proof:** The proof proceeds by rule induction on the rules corresponding to the unambiguous grammar for  $\mathcal{L}\{\text{num str}\}$ . ■

33

### Parsing to Abstract Binding Trees

A parser is a function mapping concrete to abstract syntax.

- But abstract syntax trees don't account for scope and binding.
- So we revise parsing to produce abstract binding trees.
- Identifiers are no longer operators, but are translated into variables by the parser.

34

### Abstract Binding Tree Signature for $\mathcal{L}\{\text{num str}\}$

$$\begin{aligned} \text{ar}(\text{num}[n]) &= () \\ \text{ar}(\text{str}[s]) &= () \\ \text{ar}(\text{plus}) &= (0, 0) \\ \text{ar}(\text{times}) &= (0, 0) \\ \text{ar}(\text{cat}) &= (0, 0) \\ \text{ar}(\text{len}) &= (0) \\ \text{ar}(\text{let}) &= (0, 1) \end{aligned}$$

35

### Parsing to Abstract Binding Trees

First approach: revised parsing judgements are parametric hypothetical, rather than categorical. For example:

$$\text{ID}[s_1] \text{ id} \longleftrightarrow x_1 \text{ abt}, \dots, \text{ID}[s_n] \text{ id} \longleftrightarrow x_n \text{ abt} \vdash s \text{ prg} \longleftrightarrow a \text{ abt}$$

Updating hypotheses in parsing rules records associations of bound identifiers and corresponding variables:

$$\frac{\Gamma \vdash s_1 \text{ id} \longleftrightarrow x \text{ abt} \quad \Gamma \vdash s_2 \text{ exp} \longleftrightarrow a_2 \text{ abt} \quad \Gamma, s_1 \text{ id} \longleftrightarrow x \text{ abt} \vdash s_3 \text{ prg} \longleftrightarrow a_3 \text{ abt}}{\Gamma \vdash \text{LET}_{s_1 \text{ BE } s_2 \text{ IN } s_3} \text{ prg} \longleftrightarrow \text{let}(a_2; x.a_3) \text{ abt}}$$

36

### Parsing to Abstract Binding Trees

Unfortunately, this doesn't quite work. For example, if two nested LETs bind the same name  $s$ , we could get:

$$\text{ID}[s] \longleftrightarrow x_1 \text{ abt}, \dots, \text{ID}[s] \longleftrightarrow x_n \text{ abt} \vdash s \text{ prg} \longleftrightarrow a \text{ abt}$$

So we're forced to return to categorical judgements, including management of an explicit symbol table:

$$\frac{s_1 \text{ id} \longleftrightarrow x [\sigma] \quad s_2 \text{ exp} \longleftrightarrow a_2 \text{ abt} [\sigma] \quad \sigma' = \sigma[s_1 \mapsto x] \quad s_3 \text{ prg} \longleftrightarrow a_3 \text{ abt} [\sigma']}{\text{LET}_{s_1 \text{ BE } s_2 \text{ IN } s_3} \text{ prg} \longleftrightarrow \text{let}(a_2; x.a_3) \text{ abt} [\sigma']}$$

37

### Parsing to Abstract Binding Trees

Requires addition of a rule for parsing identifiers; can't rely on reflexivity of hypothetical judgement to do it automatically:

$$\frac{\sigma(\text{ID}[s]) = x}{\text{ID}[s] \text{ id} \longleftrightarrow x [\sigma]}$$

Representing a symbol table as a finite sequence of ordered pairs adds a few more judgements:

$$\begin{array}{ll} \sigma \text{ symlab} & \text{well-formed symbol table} \\ \sigma' = \sigma[\text{ID}[s] \mapsto x] & \text{add new association} \\ \sigma(\text{ID}[s]) = x & \text{lookup identifier} \end{array}$$

38

### Informal Specifications of Syntax

We will usually specify abstract syntax informally. For example:

$$\text{Type } \tau ::= \text{num} \mid \text{str}$$

$$\text{Expr } e ::= x \mid \text{num}[n] \mid \text{str}[s] \mid \text{plus}(e_1; e_2) \mid \text{times}(e_1; e_2) \mid \text{cat}(e_1; e_2) \mid \text{len}(e) \mid \text{let}(e_1; x.e_2)$$

This defines two judgements,  $\tau$  type and  $e$  exp, asserting that two categories of objects are well formed. It also implicitly specifies two sets of operators and their corresponding arities and (binding tree) valences (*i.e.*, signatures).

It is often useful to have a more readable representation, so we usually provide a table matching concrete to abstract syntax. For example:

39

### Tabular Specification of Abstract and Concrete Syntax

Category	Item	Abstract	Concrete
Type	$\tau$	$::= \text{num}$ $\mid \text{str}$	<b>num</b> <b>str</b>
Expr	$e$	$::= x$ $\mid \text{num}[n]$ $\mid \text{str}[s]$ $\mid \text{plus}(e_1; e_2)$ $\mid \text{times}(e_1; e_2)$ $\mid \text{cat}(e_1; e_2)$ $\mid \text{len}(e)$ $\mid \text{let}(e_1; x.e_2)$	$x$ $n$ $"s"$ $e_1 + e_2$ $e_1 * e_2$ $e_1 \wedge e_2$ $ e $ <b>let</b> $x$ <b>be</b> $e_1$ <b>in</b> $e_2$

Implicitly specifies two signatures,  $\Omega_{\text{type}}$  and  $\Omega_{\text{expr}}$ .

40

### Summary

It is useful to separate **concrete** from **abstract** syntax.

- Surface features.
- Deep structure.

Parsers translate from concrete to abstract syntax.

- Using various grammar "tricks".
- Often generated automatically from the grammar.

41