

# Abstract Syntax Trees

# Formal languages

An extensional view of what constitutes a formal language is that it is completely determined by the set of 'words in the dictionary':

Given an alphabet  $\Sigma$ , we call any subset of  $\Sigma^*$  a (formal) **language** over the alphabet  $\Sigma$ .

# Concrete syntax: strings of symbols

- ▶ possibly including symbols to disambiguate the semantics (brackets, white space, *etc*),
- ▶ or that have no semantic content (e.g. syntax for comments).

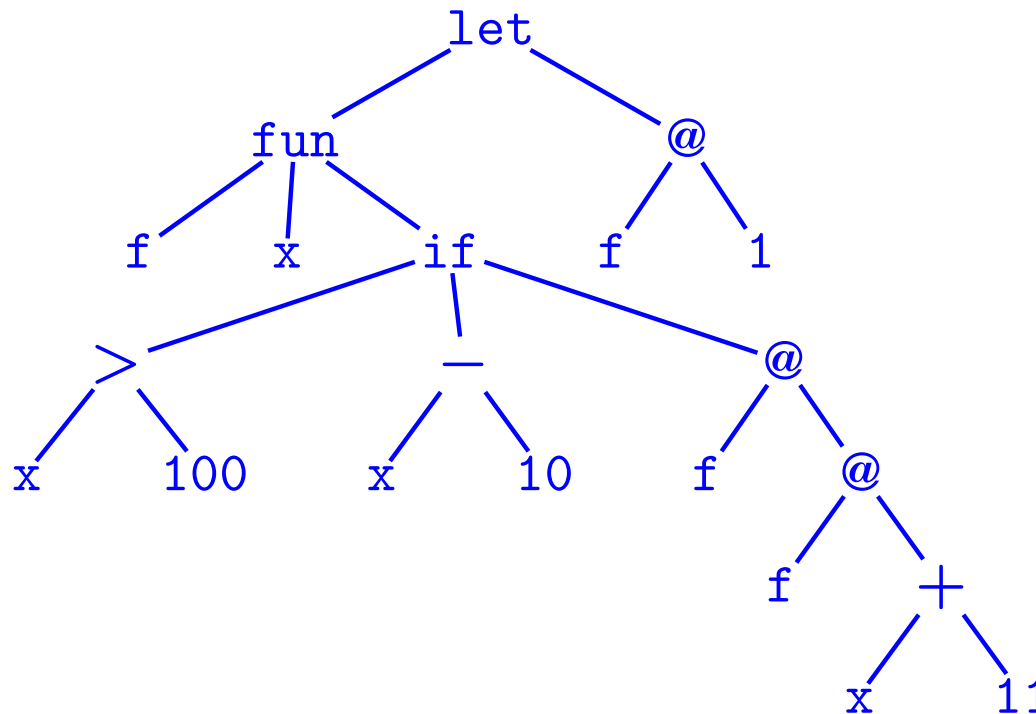
For example, an ML expression:

```
let fun f x =  
  if x > 100 then x - 10  
  else f ( f ( x + 11 ) )  
in f 1 end  
(* value is 99 *)
```

# Abstract syntax: finite rooted trees

- ▶ vertexes with  $n$  children are labelled by **operators** expecting  $n$  arguments ( **$n$ -ary** operators) – in particular leaves are labelled with **0-ary** (nullary) operators (constants, variables, *etc*)
- ▶ label of the root gives the ‘outermost form’ of the whole phrase

E.g. for the ML expression  
on Slide 25:



# Regular expressions (concrete syntax)

over a given alphabet  $\Sigma$ .

Let  $\Sigma'$  be the 4-element set  $\{\epsilon, \emptyset, |, *\}$  (assumed disjoint from  $\Sigma$ )

$$U = (\Sigma \cup \Sigma')^*$$

axioms:  $\frac{}{a}$        $\frac{}{\epsilon}$        $\frac{}{\emptyset}$

rules:  $\frac{r}{(r)}$        $\frac{r \quad s}{r|s}$        $\frac{r \quad s}{rs}$        $\frac{r}{r^*}$

(where  $a \in \Sigma$  and  $r, s \in U$ )

Some derivations of regular expressions  
 (assuming  $a, b \in \Sigma$ )

$\frac{\epsilon \quad \frac{a \quad \frac{b}{b^*}}{ab^*}}{\epsilon   ab^*}$	$\frac{\frac{\epsilon \quad a}{\epsilon   a} \quad \frac{b}{b^*}}{\epsilon   ab^*}$	$\frac{\epsilon \quad \frac{\frac{a \quad b}{ab}}{ab^*}}{\epsilon   ab^*}$
$\frac{\epsilon \quad \frac{\frac{a \quad \frac{b}{b^*}}{a(b^*)}}{(a(b^*))}}{\epsilon   (a(b^*))}$	$\frac{\frac{\epsilon \quad a}{\epsilon   a} \quad \frac{b}{b^*}}{(\epsilon   a)(b^*)}$	$\frac{\epsilon \quad \frac{\frac{\frac{a \quad b}{ab}}{(ab)}}{(ab)^*}}{\epsilon   ((ab)^*)}$

# Regular expressions (abstract syntax)

The 'signature' for regular expression abstract syntax trees (over an alphabet  $\Sigma$ ) consists of

- ▶ binary operators *Union* and *Concat*
- ▶ unary operator *Star*
- ▶ nullary operators (constants) *Null*, *Empty* and *Sym<sub>a</sub>* (one for each  $a \in \Sigma$ ).

~~E.g. can parse concrete syntax  $c|(a(b^*))$  as the abstract syntax tree~~

(delete)

# Regular expressions (abstract syntax)

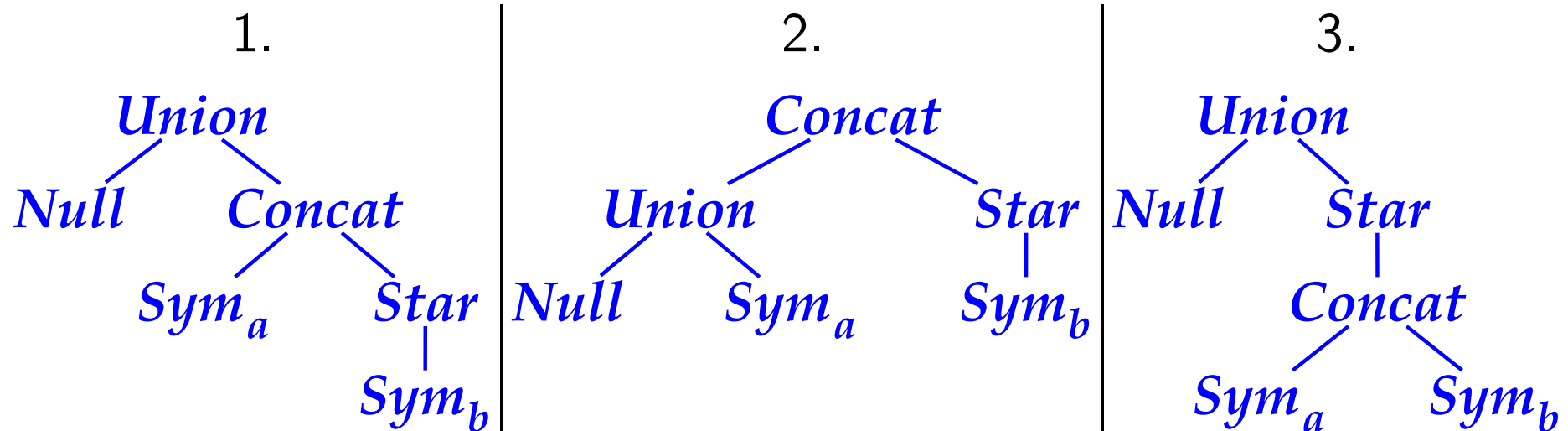
The 'signature' for regular expression abstract syntax trees (over an alphabet  $\Sigma$ ) as an ML datatype declaration:

```
datatype 'a RE = Union of ('a RE) * ('a RE)
                | Concat of ('a RE) * ('a RE)
                | Star of 'a RE
                | Null
                | Empty
                | Sym of 'a
```

(the type `'a RE` is parameterised by a type variable `'a` standing for the alphabet  $\Sigma$ )



# Some abstract syntax trees of regular expressions (assuming $a, b \in \Sigma$ )



(cf. examples on Slide 28)

We will use a textual representation of trees, for example:

1.  $\text{Union}(\text{Null}, \text{Concat}(\text{Sym}_a, \text{Star}(\text{Sym}_b)))$
2.  $\text{Concat}(\text{Union}(\text{Null}, \text{Sym}_a), \text{Star}(\text{Sym}_b))$
3.  $\text{Union}(\text{Null}, \text{Star}(\text{Concat}(\text{Sym}_a, \text{Sym}_b)))$

# Relating concrete and abstract syntax

for regular expressions over an alphabet  $\Sigma$ , via an inductively defined relation  $\sim$  between strings and trees:

$$\overline{a \sim \text{Sym}_a}$$

$$\overline{\epsilon \sim \text{Null}}$$

$$\overline{\emptyset \sim \text{Empty}}$$

$$\frac{r \sim R}{(r) \sim R}$$

$$\frac{r \sim R \quad s \sim S}{r|s \sim \text{Union}(R, S)}$$

$$\frac{r \sim R \quad s \sim S}{rs \sim \text{Concat}(R, S)}$$

$$\frac{r \sim R}{r^* \sim \text{Star}(R)}$$

For example:

$$\epsilon|(a(b^*)) \sim \text{Union}(\text{Null}, \text{Concat}(\text{Sym}_a, \text{Star}(\text{Sym}_b)))$$

$$\epsilon|ab^* \sim \text{Union}(\text{Null}, \text{Concat}(\text{Sym}_a, \text{Star}(\text{Sym}_b)))$$

$$\epsilon|ab^* \sim \text{Concat}(\text{Union}(\text{Null}, \text{Sym}_a), \text{Star}(\text{Sym}_b))$$

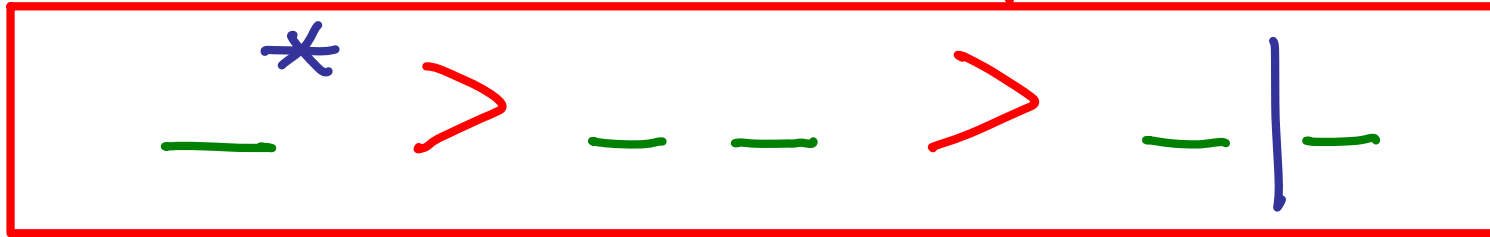
Thus  $\sim$  is a ‘many-many’ relation between strings and trees.

- ▶ **Parsing:** algorithms for producing abstract syntax trees  $\text{parse}(r)$  from concrete syntax  $r$ , satisfying  $r \sim \text{parse}(r)$ .
- ▶ **Pretty printing:** algorithms for producing concrete syntax  $\text{pp}(R)$  from abstract syntax trees  $R$ , satisfying  $\text{pp}(R) \sim R$ .

(See CST IB Compiler construction course.)

[p34]

Regular Expression operator precedence



E.g.  $\epsilon | ab^*$  means

$\epsilon | (a(b^*))$

Union (Null, Concat (Sym<sub>a</sub>, Star (Sym<sub>b</sub>)))

[p34]

Regular Expression **associativity**

**Concatenation**  
**union** } are left associative

E.g.  $\left\{ \begin{array}{l} abc \\ a|b|c \end{array} \right.$  stands for  $(ab)c$   
" " "  $(a|b)|c$

FROM NOW ON WE'LL USE  
CONCRETE SYNTAX OF REGULAR  
EXPRESSIONS TO REFER TO THEIR  
ABSTRACT SYNTAX, RELYING ON  
OPERATOR PRECEDENCE (& ASSOCIATIVITY)  
CONVENTIONS TO AVOID AMBIGUITY

[p34]

Regular expression **associativity**

**concatenation**

**union**

} are left associative

Less important than operator precedence  
because the meaning (**semantics**) of  
these is always associative.

# Matching

Each regular expression  $r$  over an alphabet  $\Sigma$  determines a language  $L(r) \subseteq \Sigma^*$ . The strings  $u$  in  $L(r)$  are by definition the ones that **match**  $r$ , where

- ▶  $u$  matches the regular expression  $a$  (where  $a \in \Sigma$ ) iff  $u = a$
- ▶  $u$  matches the regular expression  $\epsilon$  iff  $u$  is the null string  $\epsilon$
- ▶ no string matches the regular expression  $\emptyset$
- ▶  $u$  matches  $r|s$  iff it either matches  $r$ , or it matches  $s$
- ▶  $u$  matches  $rs$  iff it can be expressed as the concatenation of two strings,  $u = vw$ , with  $v$  matching  $r$  and  $w$  matching  $s$
- ▶  $u$  matches  $r^*$  iff either  $u = \epsilon$ , or  $u$  matches  $r$ , or  $u$  can be expressed as the concatenation of two or more strings, each of which matches  $r$ .



# Inductive definition of matching

$$U = \Sigma^* \times \{\text{regular expressions over } \Sigma\}$$

axioms:

$$\frac{}{(a, a)}$$

$$\frac{}{(\epsilon, \epsilon)}$$

$$\frac{}{(\epsilon, r^*)}$$

abstract syntax trees

rules:

$$\frac{(u, r)}{(u, r|s)}$$

$$\frac{(u, s)}{(u, r|s)}$$

$$\frac{(v, r) \quad (w, s)}{(vw, rs)}$$

$$\frac{(u, r) \quad (v, r^*)}{(uv, r^*)}$$

(No axiom/rule involves the empty regular expression  $\emptyset$  – why?)

# Examples of matching

Assuming  $\Sigma = \{a, b\}$ , then:

- ▶  $a|b$  is matched by each symbol in  $\Sigma$
- ▶  $b(a|b)^*$  is matched by any string in  $\Sigma^*$  that starts with a 'b'
- ▶  $((a|b)(a|b))^*$  is matched by any string of even length in  $\Sigma^*$
- ▶  $(a|b)^*(a|b)^*$  is matched by any string in  $\Sigma^*$
- ▶  $(\varepsilon|a)(\varepsilon|b)|bb$  is matched by just the strings  $\varepsilon$ ,  $a$ ,  $b$ ,  $ab$ , and  $bb$
- ▶  $\emptyset b|a$  is just matched by  $a$

# Some questions

- (a) Is there an algorithm which, given a string  $u$  and a regular expression  $r$ , computes whether or not  $u$  matches  $r$ ?
- (b) In formulating the definition of regular expressions, have we missed out some practically useful notions of pattern?
- (c) Is there an algorithm which, given two regular expressions  $r$  and  $s$ , computes whether or not they are **equivalent**, in the sense that  $L(r)$  and  $L(s)$  are equal sets?
- (d) Is every language (subset of  $\Sigma^*$ ) of the form  $L(r)$  for some  $r$ ?