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 Σ, we call any subset of Σ^* a (formal) **language** over the alphabet Σ.

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).

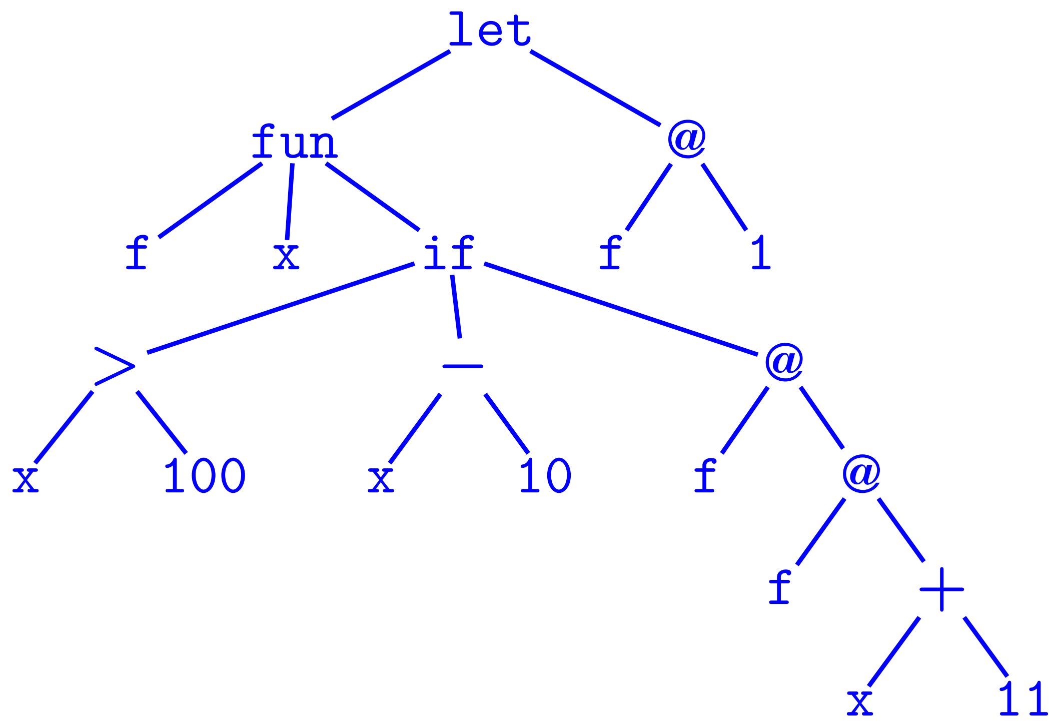
For example, an ML expression:

```
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 $\mathbf{0}$-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 Σ. $\{\varepsilon, \emptyset, |, *, (,)\}$

Let Σ' be the ~~4~~ 6-element set ~~$\{\epsilon, \emptyset, |, *\}$~~ (assumed disjoint from Σ)

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

axioms: $\dfrac{}{a}$ $\quad \dfrac{}{\epsilon}$ $\quad \dfrac{}{\emptyset}$

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

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

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

$$\dfrac{\epsilon \qquad \dfrac{a \qquad \dfrac{b}{b^*}}{ab^*}}{\epsilon|ab^*}$$

$$\dfrac{\dfrac{\epsilon \qquad a}{\epsilon|a} \qquad \dfrac{b}{b^*}}{\epsilon|ab^*}$$

$$\dfrac{\epsilon \qquad \dfrac{\dfrac{a \qquad b}{ab}}{ab^*}}{\epsilon|ab^*}$$

$$\dfrac{\epsilon \qquad \dfrac{\dfrac{a \qquad \dfrac{\dfrac{b}{b^*}}{(b^*)}}{a(b^*)}}{(a(b^*))}}{\epsilon|(a(b^*))}$$

$$\dfrac{\dfrac{\dfrac{\epsilon \qquad a}{\epsilon|a}}{(\epsilon|a)} \qquad \dfrac{\dfrac{b}{b^*}}{(b^*)}}{(\epsilon|a)(b^*)}$$

$$\dfrac{\epsilon \qquad \dfrac{\dfrac{\dfrac{\dfrac{a \qquad b}{ab}}{(ab)}}{(ab)^*}}{((ab)^*)}}{\epsilon|((ab)^*)}$$

Regular expressions (abstract syntax)

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

- binary operators ***Union*** and ***Concat***
- unary operator ***Star***
- nullary operators (constants) ***Null***, ***Empty*** and Sym_a (one for each $a \in \Sigma$).

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

(delete)

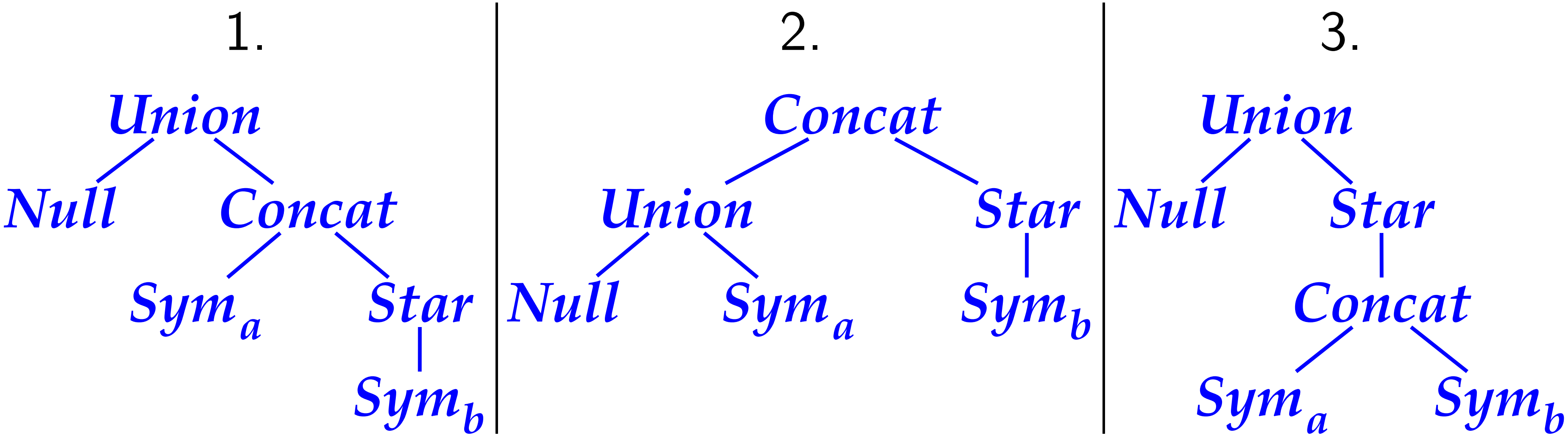
Regular expressions (abstract syntax)

The 'signature' for regular expression abstract syntax trees (over an alphabet $\boldsymbol{\Sigma}$) 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 $\boldsymbol{\Sigma}$)

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



(*cf.* examples on Slide 28)

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

1. $\boldsymbol{Union(Null, Concat(Sym_a, Star(Sym_b)))}$
2. $\boldsymbol{Concat(Union(Null, Sym_a), Star(Sym_b))}$
3. $\boldsymbol{Union(Null, Star(Concat(Sym_a, Sym_b)))}$

Relating concrete and abstract syntax

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

$$\frac{}{a \sim Sym_a} \qquad \frac{}{\epsilon \sim Null} \qquad \frac{}{\emptyset \sim Empty}$$

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

$$\frac{r \sim R \qquad s \sim S}{rs \sim Concat(R,S)} \qquad \frac{r \sim R}{r^* \sim Star(R)}$$

For example:

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

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

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

(See CST IB Compiler construction course.)

[p34]

Regular expression operator precedence

$$\boxed{\_^{*} > \_\ \_ > \_\,|\,\_}$$

E.g. $\varepsilon \,|\, a b^{*}$ means

$$\varepsilon \,|\, (a(b^{*}))$$

Union(Null, Concat(Sym$_a$, Star(Sym$_b$)))

[p34]

Regular expression associativity

concatenation
union
} are left associative

E.g {
abc stands for (ab)c
a|b|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 <u>left</u> associative

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

Matching

Each regular expression r over an alphabet Σ 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 ϵ iff u is the null string ε
- 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 $u = \varepsilon$, 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

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

abstract syntax trees

axioms: $\dfrac{}{(a, a)}$ $\quad \dfrac{}{(\varepsilon, \epsilon)}$ $\quad \dfrac{}{(\varepsilon, r^*)}$

rules:

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

$$\frac{(v, r) \quad (w, s)}{(vw, rs)} \qquad \frac{(u, r) \quad (v, r^*)}{(uv, r^*)}$$

(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 Σ
- $b(a|b)^*$ is matched by any string in Σ^* that starts with a 'b'
- $((a|b)(a|b))^*$ is matched by any string of even length in Σ^*
- $(a|b)^*(a|b)^*$ is matched by any string in Σ^*
- $(\varepsilon|a)(\varepsilon|b)|bb$ is matched by just the strings ε, 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 Σ^*) of the form $L(r)$ for some r?