Example using rule induction

Let I be the subset of $\{a, b\}^*$ inductively defined by the axioms and rules on Slide 17 of the notes.

For $u \in \{a, b\}^*$, let $P(u)$ be the property

$$u \text{ contains the same number of } a \text{ and } b \text{ symbols}$$

We can prove $\forall u \in I.\, P(u)$ by rule induction:

- **base case:** $P(\varepsilon)$ is true (the number of as and bs is zero!)
- **induction steps:** if $P(u)$ and $P(v)$ hold, then clearly so do $P(aub)$, $P(bua)$ and $P(uv)$.

(It's not so easy to show $\forall u \in \{a, b\}^*.\, P(u) \Rightarrow u \in I$ – rule induction for I is not much help for that.)

Example [CST 2009, Paper2, Question 5]

$I \subseteq \{a, b\}^*$ inductively defined by

$$\frac{}{a} \qquad \frac{u}{au} \qquad \frac{u \quad v}{buv}$$

Example [CST 2009, Paper2, Question 5]

$I \subseteq \{a, b\}^*$ inductively defined by

$$\boxed{\quad \frac{}{a}\,0 \qquad \frac{u}{au}\,1 \qquad \frac{u\ v}{buv}\,2 \quad}$$

In this case Rule Induction says:

if (0) $P(a)$

& (1) $\forall u \in I \,.\, P(u) \Rightarrow P(au)$

& (2) $\forall u, v \in I \,.\, P(u) \wedge P(v) \Rightarrow P(buv)$

then $\forall u \in I \,.\, P(u)$

for <u>any</u> predicate $P(u)$

Example [CST 2009, Paper2, Question 5]

$I \subseteq \{a, b\}^*$ inductively defined by

$$\boxed{\frac{}{a}\,0 \qquad \frac{u}{au}\,1 \qquad \frac{u\ v}{buv}\,2}$$

Asked to show

$$u \in I \Rightarrow \#_a(u) > \#_b(u)$$

i.e., that there are more 'a's than 'b's in every string in I

Example [CST 2009, Paper 2, Question 5]

$I \subseteq \{a, b\}^*$ inductively defined by

$$\frac{}{a}\,0 \qquad \frac{u}{au}\,1 \qquad \frac{u \quad v}{buv}\,2$$

Asked to show

$$u \in I \Rightarrow \#_a(u) > \#_b(u)$$

so do so using Rule Induction with

$$P(u) = \#_a(u) > \#_b(u)$$

Example [CST 2009, Paper2, Question 5]

$I \subseteq \{a,b\}^*$ inductively defined by

$$\boxed{\frac{}{a}\,0 \qquad \frac{u}{au}\,1 \qquad \frac{u \quad v}{buv}\,2}$$

$$P(u) = \#_a(u) > \#_b(u)$$

(0) $P(a)$ holds $(1 > 0)$

Example [CST 2009, Paper2, Question 5]

$I \subseteq \{a, b\}^*$ inductively defined by

$$\boxed{\frac{}{a}\;{\color{red}0} \qquad \frac{u}{au}\;{\color{red}1} \qquad \frac{u\;v}{buv}\;{\color{red}2}}$$

$$P(u) = \#_a(u) > \#_b(u)$$

(1) If $P(u)$, then $\#_a(au) = 1 + \#_a(u)$

Example [CST 2009, Paper2, Question 5]

$I \subseteq \{a,b\}^*$ inductively defined by

$$\frac{}{a}\,0 \qquad \frac{u}{au}\,1 \qquad \frac{u\ v}{buv}\,2$$

$$P(u) = \#_a(u) > \#_b(u)$$

(1) If $P(u)$, then $\#_a(au) = 1 + \#_a(u)$

$> \#_a(u) > \#_b(u)$ (because $P(u)$)

$= \#_b(au)$

Example [CST 2009, Paper2, Question 5]

$I \subseteq \{a,b\}^*$ inductively defined by

$$\boxed{\frac{}{a}\,0 \qquad \frac{u}{au}\,1 \qquad \frac{u\ v}{buv}\,2}$$

$$P(u) = \#_a(u) > \#_b(u)$$

(1) If $P(u)$, then $\#_a(au) = 1 + \#_a(u)$

$> \#_a(u) > \#_b(u)$ (because $P(u)$)

$= \#_b(au)$

so $P(au)$ holds as well, and thus $P(u) \Rightarrow P(au)$

Example [CST 2009, Paper2, Question 5]

$I \subseteq \{a,b\}^*$ inductively defined by

$$\frac{}{a}\,0 \qquad \frac{u}{au}\,1 \qquad \frac{u\ v}{buv}\,2$$

$$P(u) = \#_a(u) > \#_b(u)$$

(2) If $P(u) \wedge P(v)$, then $\#_a(buv) = \#_a(u) + \$\#_a(v)$
$\geq ((\#_b(u) + 1) + (\#_b(v) + 1))$ (why?)
$> \#_b(buv)$
so $P(buv)$

Example [CST 2009, Paper2, Question 5]

$I \subseteq \{a,b\}^*$ inductively defined by

$$\frac{}{a}\,0 \qquad \frac{u}{au}\,1 \qquad \frac{u\ v}{buv}\,2$$

$$P(u) = \#_a(u) > \#_b(u)$$

if (0) $P(a)$ ✓

& (1) $\forall u \in I \,.\, P(u) \Rightarrow P(au)$ ✓

& (2) $\forall u,v \in I \,.\, P(u) \wedge P(v) \Rightarrow P(buv)$ ✓

then $\forall u \in I \,.\, P(u)$

so for all $u \in I$, we have $\#_a(u) > \#_b(u)$ $\square$

Example [CST 2009, Paper2, Question 5]

$I \subseteq \{a,b\}^*$ inductively defined by

$$\boxed{\frac{}{a}\,0 \qquad \frac{u}{au}\,1 \qquad \frac{u\ v}{buv}\,2}$$

$$P(u) = \#_a(u) > \#_b(u)$$

although we have $\forall u \in I \,.\, P(u)$

we don't have $\forall u \in \{a,b\}^* \,.\, P(u) \Rightarrow u \in I$

e.g. $P(aab)$ but $aab \notin I$ (Why?)

Deciding membership of an inductively defined subset can be hard!

Deciding membership of an inductively defined subset can be hard!

really, Really hard

e.g. ...

Collatz Conjecture

$$f(n) = \begin{cases} 1 & \text{if } n = 0, 1 \\ f(n/2) & \text{if } n > 1,\ n \text{ even} \\ f(3n+1) & \text{if } n > 1,\ n \text{ odd} \end{cases}$$

Does this define a <u>total</u> function $f : \mathbb{N} \to \mathbb{N}$?

(nobody knows)

Collatz Conjecture

$$f(n) = \begin{cases} 1 & \text{if } n = 0, 1 \\ f(n/2) & \text{if } n > 1, \ \ n \text{ even} \\ f(3n+1) & \text{if } n > 1, \ \ n \text{ odd} \end{cases}$$

Does this define a total function $f : \mathbb{N} \to \mathbb{N}$?

(nobody knows)

(If it does then f is necessarily the unary $\mathbb{1}$ function $n \mapsto 1$)

Collatz Conjecture

$$f(n) = \begin{cases} 1 & \text{if } n = 0, 1 \\ f(n/2) & \text{if } n > 1,\ n \text{ even} \\ f(3n+1) & \text{if } n > 1,\ n \text{ odd} \end{cases}$$

Does this define a <u>total</u> function $f : \mathbb{N} \to \mathbb{N}$?

(nobody knows)

Can reformulate as a problem about inductively defined subsets...

Collatz Conjecture

$$f(n) = \begin{cases} 1 & \text{if } n = 0, 1 \\ f(n/2) & \text{if } n > 1,\ n \text{ even} \\ f(3n+1) & \text{if } n > 1,\ n \text{ odd} \end{cases}$$

Is the subset $I \subseteq \mathbb{N}$ inductively defined by

$$\boxed{\frac{}{0} \qquad \frac{}{1} \qquad \frac{k}{2k} \qquad \frac{6k+4}{2k+1} \qquad (k \geq 1)}$$

equal to the whole of $\mathbb{N}$?

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).

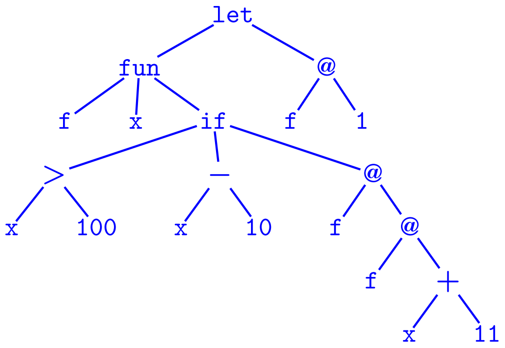
For example, an ML expression:

```
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 41:



Regular Expressions

A regular expression defines a pattern of symbols (and thus a language).

Important to distinguish between the language a particular regular expression defines and the set of possible regular expressions.

We about to look at the second of these.

Regular expressions (concrete syntax)

over a given alphabet Σ.

Let Σ' be the 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 $a \in \Sigma$).

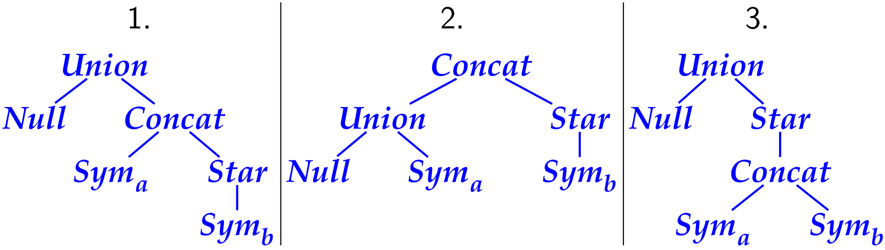
Regular expressions (abstract syntax)

The ‘signature’ for regular expression abstract syntax trees (over an alphabet Σ) 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 Σ)

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



(*cf.* examples a few slides previous)

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

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

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:

$$
\begin{aligned}
\epsilon|(a(b^*)) &\sim Union(Null, Concat(Sym_a, Star(Sym_b))) \\
\epsilon|ab^* &\sim Union(Null, Concat(Sym_a, Star(Sym_b))) \\
\epsilon|ab^* &\sim Concat(Union(Null, Sym_a), Star(Sym_b))
\end{aligned}
$$

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.)

Operator precedence for regular expressions

$$\boxed{\text{Star} > \text{Concat} > \text{Union}}$$

So

$\varepsilon|ab^*$ stands for $\varepsilon|(a(b^*))$

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

Associativity for regular expressions

$$\boxed{\text{Concat \& Union are \underline{left} associative}}$$

So

$$abc \quad \text{stands for} \quad (ab)c$$

$$a|b|c \quad \text{stands for} \quad (a|b)|c$$

From now on, we will rely on operator precedence (& associativity) conventions in the concrete syntax of regular expressions to allow us to map directly to their abstract syntax

associativity less important (in some sense) than precedence because the meaning (semantics) of concatenation and union is always associative

so abc has the same abstract syntax as $(ab)c$, but different abstract syntax from $a(bc)$, but all of these have the same semantics.

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.