

OCaml Syntax

Basic types

Type	Example	Operations
int	3	+ - * /
float	3.14159	+. -. *. /.
string	"Hello world"	^
bool	true, false	&& not
unit	()	

Compound types

Type	Example	Operations
'a * 'b	(5, true)	, fst snd
'a list	[1; 2; 3] or 1::2::3::[]	:: hd tl
'a -> 'b	fun x -> x	

User-defined variant types

'a option = Some of 'a | None

Functions

fun x -> expr

Comparison

= <> < >

Conditional

if 3 > 2 then "X" else "Y"

Local variables

let n = expr in n + 2
 let f x y = expr in f 3 "xyz"
 let rec g x = g (x - 1) in g 3

Match

match opt with
 None -> []
 | Some x -> x::[]

Some utility functions

List.length List.map
 List.fold_left List.fold_right
 String.length
 int_of_string string_of_int
 float_of_string string_of_float

match pr with
 (false, []) -> 7
 | (false, x::xs) -> x
 | (true, _) -> 3

Regular Expressions

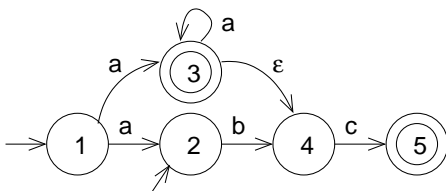
Let Σ be a set of characters (the alphabet), and let $a \in \Sigma$, while P and Q are regular expressions. The table below defines the language of strings $L(R) \subseteq \Sigma^*$ denoted by a regular expression R .

$L(\epsilon)$	$= \{\epsilon\}$	The empty string
$L(a)$	$= \{a\}$	Just the character a
$L(PQ)$	$= L(P) \times L(Q)$	Any string matching P followed by a string matching Q
$L(P^*)$	$= \{\epsilon\} \cup L(P) \times L(P^*)$	A sequence of zero or more strings matching P
$L(P + Q)$	$= L(P) \cup L(Q)$	A string matching either P or Q

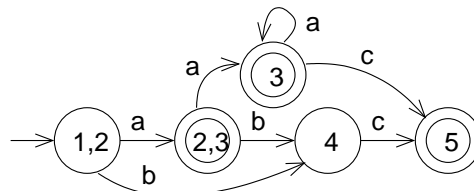
The precedence order is *, then concatenation, then +. For example, $PQ^* + R$ means $(P(Q^*)) + R$.

NFAs & DFAs

This is an NFA:



And this is an equivalent, minimal DFA:



Unification Rules

A **substitution** is a mapping from *variables* to *terms*, $\theta : \text{Variable} \rightarrow \text{Term}$.

A **unifier** is a substitution θ which, when applied to a set of equations E , makes the two sides of each equation equal.

A **most general unifier** for E is a unifier θ s.t. for any other unifier θ' , there exists a substitution S , s.t. $E\theta' = E\theta S$.

Given a set of equations E , we can compute a most general unifier θ as follows. Here t , s_i , and t_i are terms, f and g are term constructors, and α is a variable.

Equation from E	Action
1. $f(s_1, \dots, s_n) = f(t_1, \dots, t_n)$	Decompose: replace by the equations $s_1 = t_1, \dots, s_n = t_n$
2. $f(s_1, \dots, s_n) = g(t_1, \dots, t_m)$	Fail: return <i>not unifiable</i>
3. $t = t$	Delete: remove the equation
4. $t = \alpha$	Orient: replace the equation by $\alpha = t$
4. $\alpha = t, \alpha \notin \text{vars}(t)$	Eliminate: apply $[\alpha \rightarrow t]$ to E and θ ; add $\alpha \rightarrow t$ to θ
5. $\alpha = t, \alpha \in \text{vars}(t), \alpha \neq t$	Fail: return <i>not unifiable</i>

Type Derivation Rules

(UNIT)	$\frac{}{\Gamma \vdash () : \text{unit}}$	$\frac{}{\Gamma \vdash [] : \tau \text{ list}}$	(NIL)
(BOOL)	$\frac{}{\Gamma \vdash b : \text{bool}}$	where $b = \text{true}$ or false	
(INT)	$\frac{}{\Gamma \vdash n : \text{int}}$	where n is an integer	
(FLOAT)	$\frac{}{\Gamma \vdash n : \text{float}}$	where n is a floating point number	
(VAR)	$\frac{}{\Gamma \vdash x : \tau}$	where $\Gamma(x) = \tau$	
(PAIR)	$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : (\tau_1, \tau_2)}$		
(LIST)	$\frac{\Gamma \vdash e_h : \tau \quad \Gamma \vdash e_t : \tau \text{ list}}{\Gamma \vdash e_h :: e_t : \tau \text{ list}}$		
(FUN)	$\frac{\Gamma \cup [x : \tau_x] \vdash e : \tau_e}{\Gamma \vdash \text{fun } x \rightarrow e : \tau_x \rightarrow \tau_e}$		
(APP)	$\frac{\Gamma \vdash e_f : \tau_a \rightarrow \tau_r \quad \Gamma \vdash e_a : \tau_a}{\Gamma \vdash e_f e_a : \tau_r}$		
(LET)	$\frac{\Gamma \vdash e_x : \tau_x \quad \Gamma \cup [x : \tau_x] \vdash e : \tau}{\Gamma \vdash \text{let } x = e_x \text{ in } e : \tau}$		
(LETREC)	$\frac{\Gamma \cup [x : \tau_x] \vdash e_x : \tau_x \quad \Gamma \cup [x : \tau_x] \vdash e : \tau}{\Gamma \vdash \text{let rec } x = e_x \text{ in } e : \tau}$		