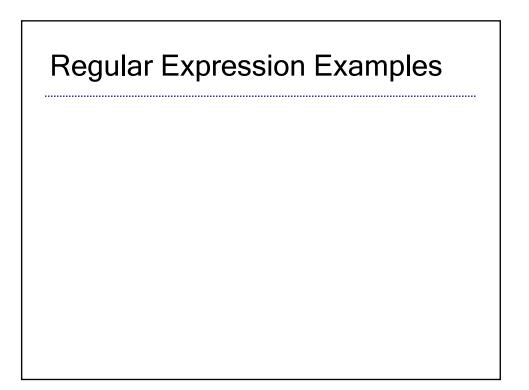
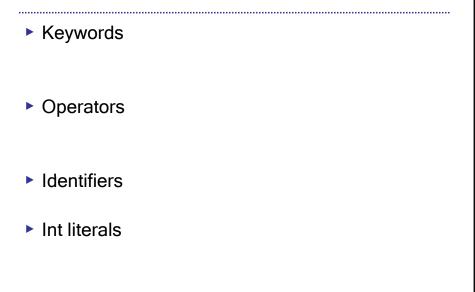


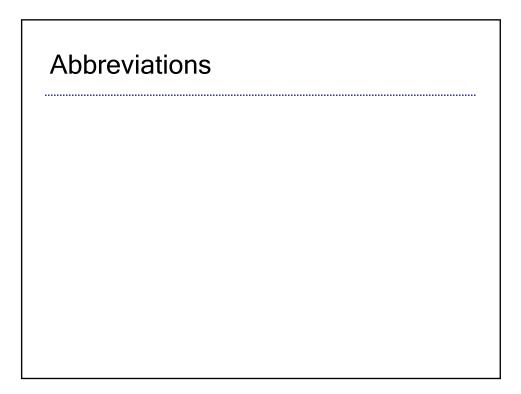
Regular Expressions

- A regular expression is one of
 - ► *ϵ*, aka ""
 - 'a' for any character a
 - $ightharpoonup r_1$ r₂, where r₁ and r₂ are regular expr's
 - $r_1 | r_2$, where r_1 and r_2 are regular expr's
 - r*, where r is a reg expr's
 - ÞØ











Float-point Literal

Regular Expression Example

- New-Style Comments (//)
- Old-Style Comments (/* ... */)

Implementing Reg Expr

Translate RE's to NFA's, then to DFA's

Lexing with Reg Exprs Create one large RE: Then add actions



- Ambiguous cases:
- Two tokens found, one longer
- Two tokens found, the same length

General Input

```
{ header }
let ident = regexp ...
rule entrypoint [arg1... argn] = parse
    regexp { action }
    | ...
    | regexp { action }
and entrypoint [arg1... argn] = parse ...and ...
{ trailer }
```

.....

Ocamllex Input

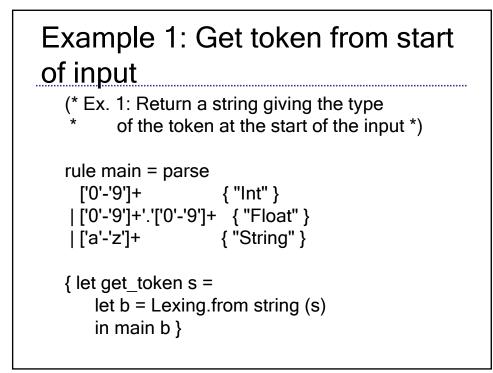
- header and trailer contain arbitrary ocaml code put at top an bottom of <filename>.ml
- let ident = regexp ... Introduces ident for use in later regular expressions

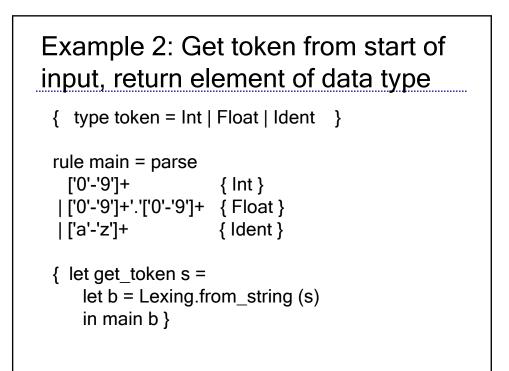
Mechanics

- Put table of regular expressions and corresponding actions (written in ocaml) into a file
 - <filename>.mll
- Call

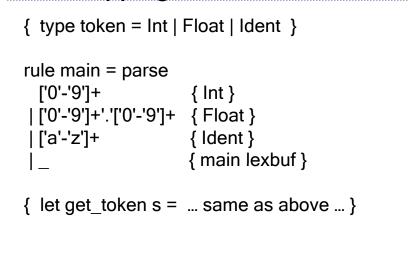
ocamllex <filename>.mll

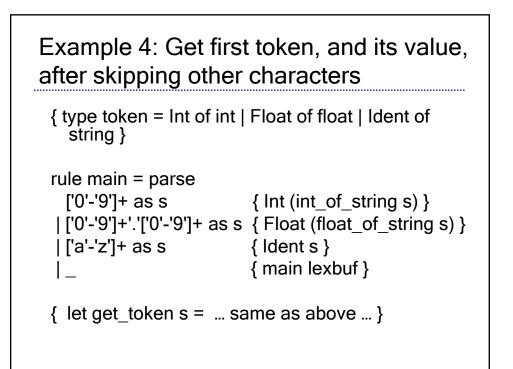
Produces Ocaml code for a lexical analyzer in file <filename>.ml

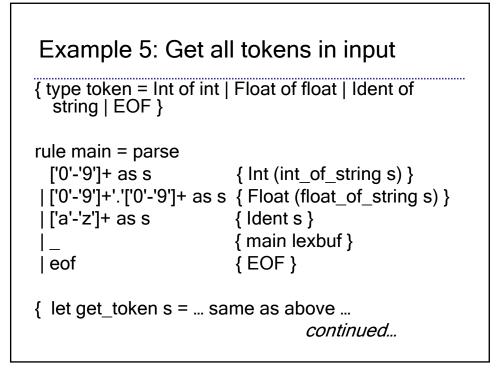


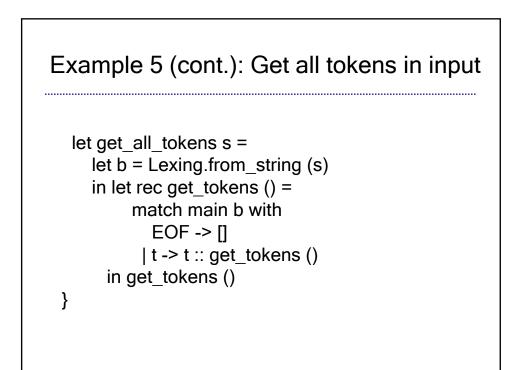


Example 3: Get first token in input, after skipping other characters



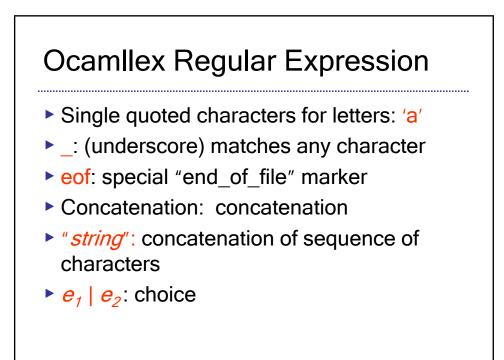






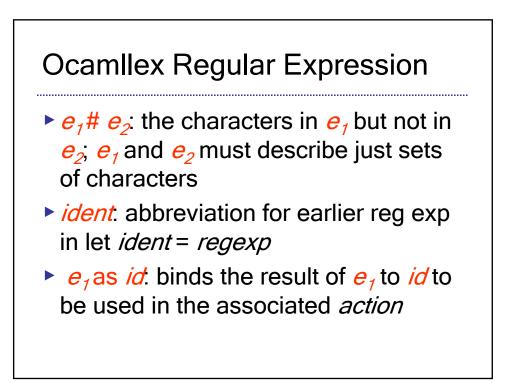
Ocamllex Input

- *<filename>*.ml contains one lexing function per *entrypoint*
 - Name of function is name given for entrypoint
 - Each entry point becomes an Ocaml function that takes n+1 arguments, the extra implicit last argument being of type Lexing.lexbuf
- arg1... argn are for use in action





- [C₁ C₂]: choice of any character between first and second inclusive, as determined by character codes
- [^c₁ c₂]: choice of any character NOT in set
- e*: same as before
- e+: same as e e*
- $e_{?}$: option was e_{1} ϵ



Ocamllex Manual

More details can be found at

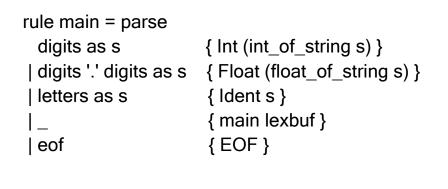
http://caml.inria.fr/pub/docs/manualocaml/manual026.html

Example 6: example 5 using abbreviations

```
{ type token = Int of int | Float of float | Ident of
  string | EOF }
let digit = ['0'-'9']
let digits = digit +
let lower_case = ['a'-'z']
let upper_case = ['A'-'Z']
let letter = upper_case | lower_case
let letters = letter +
```

continued...

Example 6 (cont.): example 5 using abbreviations



C-style comments

let open_comment = "/*"	
let close_comment = "*/"	
rule main = parse	
digits '.' digits as f	<pre>{ Float (float_of_string f) }</pre>
digits as n	{ Int (int_of_string n) }
letters as s	{ Ident s }
continued	

