Lecture 14

Thursday, October 7, 2021 2:01 PM

- Midterm Review + What to expect
- Resolving grammatical ambiguities
  - Shift / reduce parsing
  - Longest match principle
- Algorithms
  - Dynamic programming for regular expressions

    $O(n^3 k)$
  - D.P for context free grammars (CYK)
  - O(n) parsing of regular expressions

    (Finite automata)

① Midterm Review

- Everything in Ocaml is an expression / term
- A few forms:
  - Literals: 3, "abc", true, [2; 8], ...
  - Variables
  - Function applications f a
  - Function abstractions fun x → ____
  - Let expressions let x = ____ in ____
  - Conditionals: if ____ then ____ else ____

"values"

Not "values"

____

Parentheses only for

(a) Resolving precedence

(a) Resolving precedence

"f e x" ⟶ Do you mean

f (e x)

or (f e) x ?

Traditional meaning

(b) Multi argument ADT constructors

Some 3 : int option

Node (3, Leaf, Leaf)

Constructor

Parentheses necessary for multi-argument constructors

```
((List.map (fun x -> x + 1) [1; 2; 3]) : Implicitly
```

List.map : ('a -> 'b) -> ('a list -> 'b list)

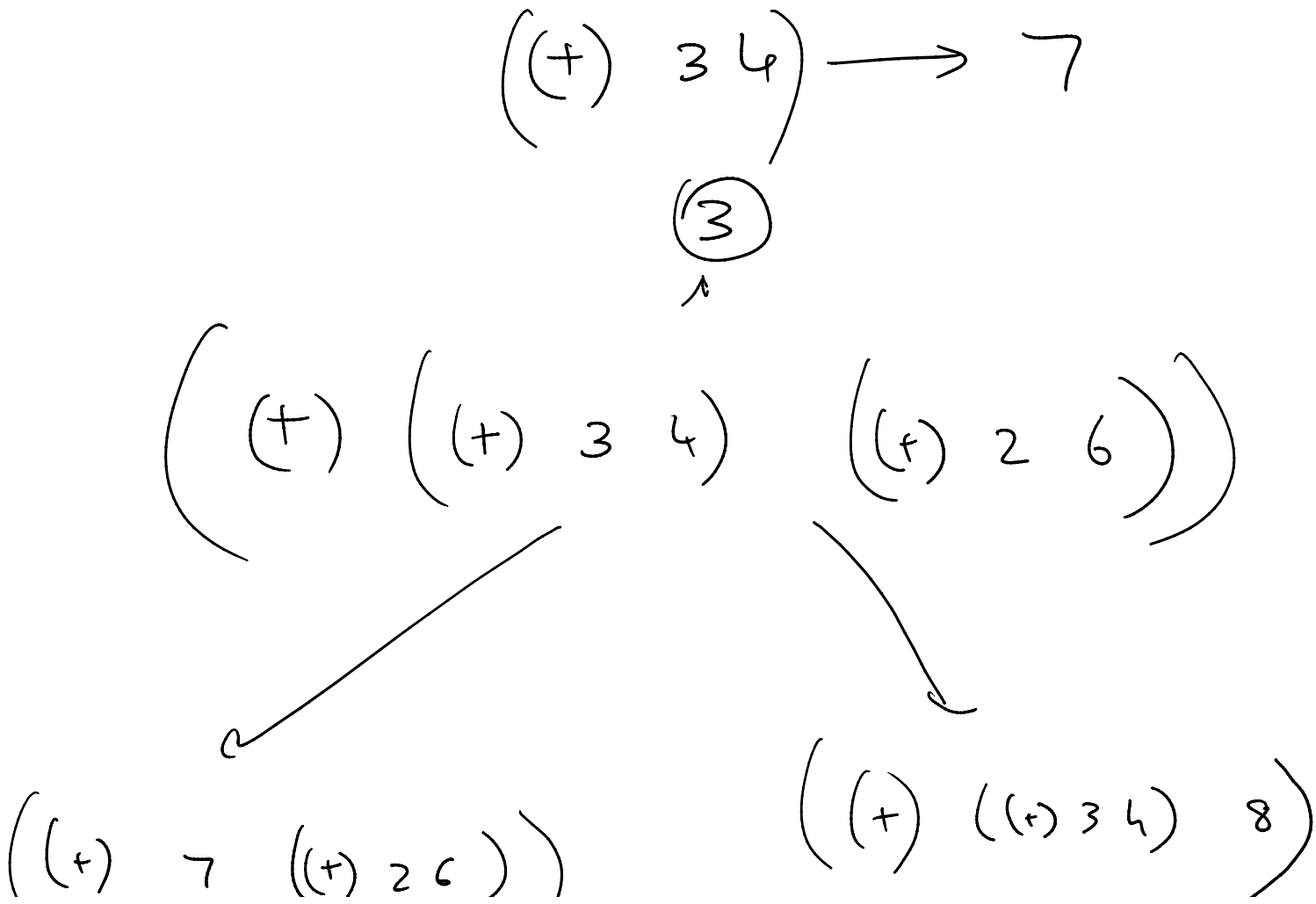
List.map : ('a -> 'b) -> ('a list -> 'b list)

Implicitly

Evaluation rules

Transform expressions to other expressions,

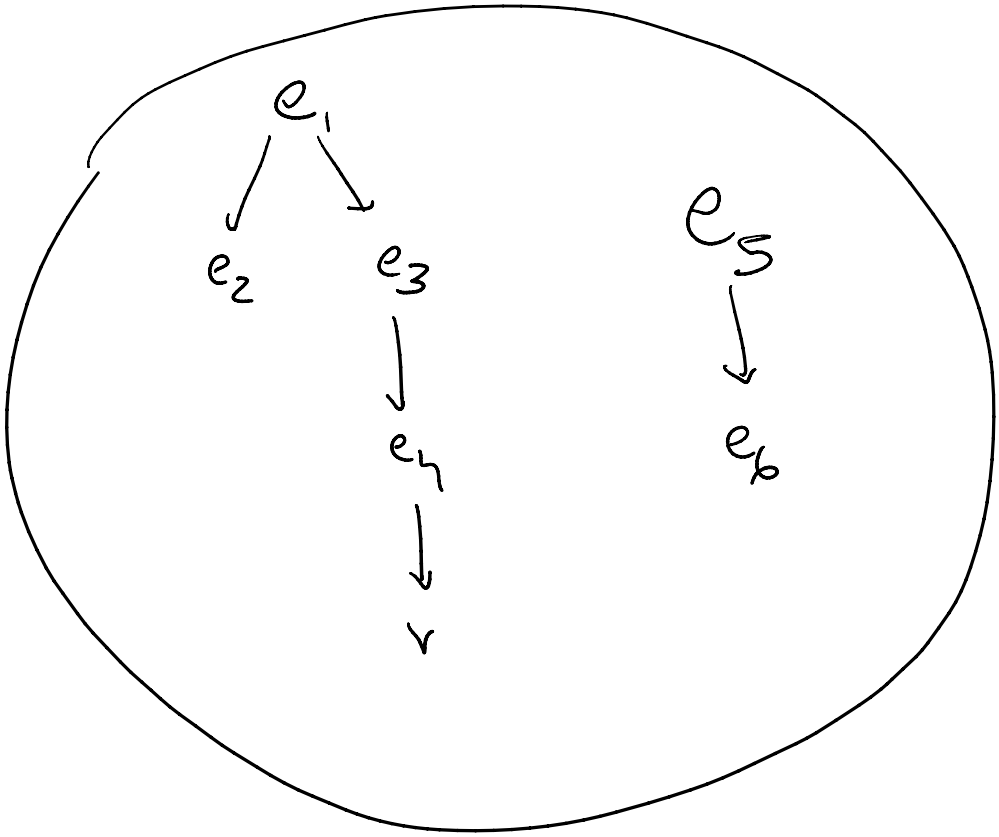
(sometimes)

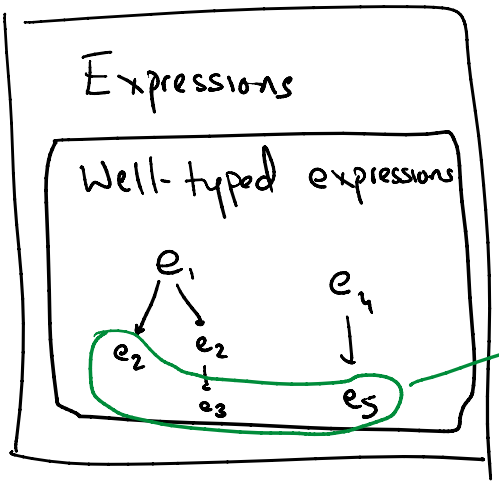


((+) 7 ((+) 2 6))

((+) ((+) 3 4) 8)

Giant forest of expressions





"Values"

Expressions from which no

② Each value has a type

① more stepping can occur

③ Guarantee: If an expression is well-typed:

then it either

(a) steps to another expression of the same type, or

(b) is a value

Midterm Questions 1 — 5

① State type of some expression

② State expressions with same type

③ Show evaluation sequence

④ Scoping rules

} predict result

⑤ Fix

⑤ Fix mistakes in ill-typed expression

Midterm Questions ⑥, ⑦ Design

⑥ Write program that solves problem (Algorithmic)

⑦ Design types for some application

---

⑧ — ⑩: Random topics

⑧ Tail recursion

⑨ Mutable references

⑩ List map & List fold

Resolving Grammatical Ambiguities

Our almost-final calculator

```
"3 - 4 + 5" |> Lexing.from_string |> (top_expr read) |> eval
```

```
"3 - 4 + 5" |> Lexing.from_string |> (top_expr read) |> eval
```

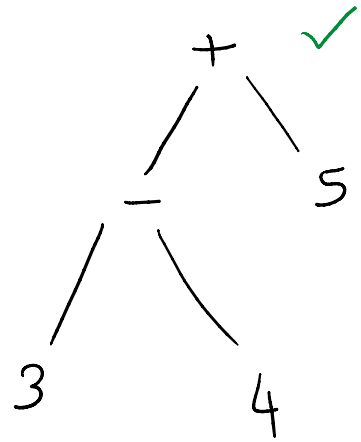
```
"3 - 4 + 5" |> Lexing.from_string |> (top_expr read) |> eval
```

Parser

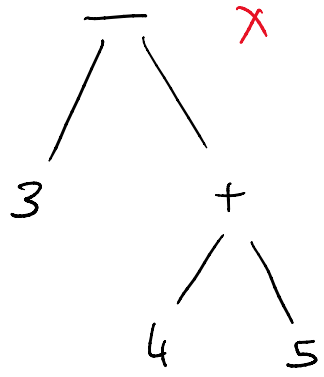
Lexical analyzer

Evaluator

"3 - 4 + 5" ⟶

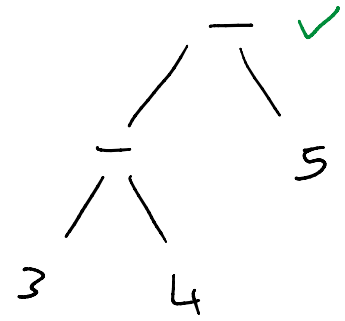


✓

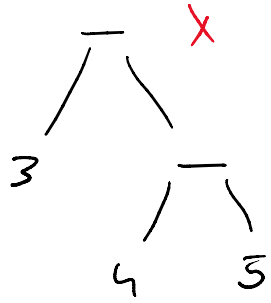


✗

"3 - 4 - 5"



✓



X

Right sides of - & + are always integers

Left sides of - & + are integers