Lecture 14

Thursday, October 8, 2020 4:07 PM

Based on our grammar, the expression 10 - 4 + 3
can be parsed either as

(10 - 4) + 3,

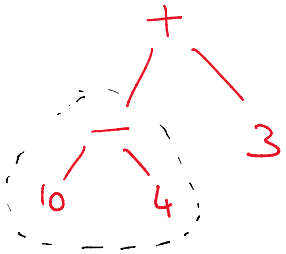
or as

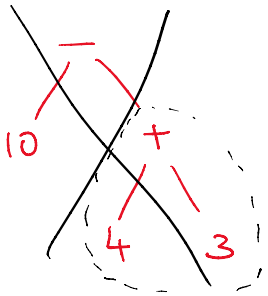
10 - (4 + 3)

What if we say that the right side of a MINUS must be a single number?

Both these parentheses are implicit.

$10 - 4 + 3$





Then, this parse tree is no longer feasible.

- But what about $10 - (4+3)$

expr ::= ...

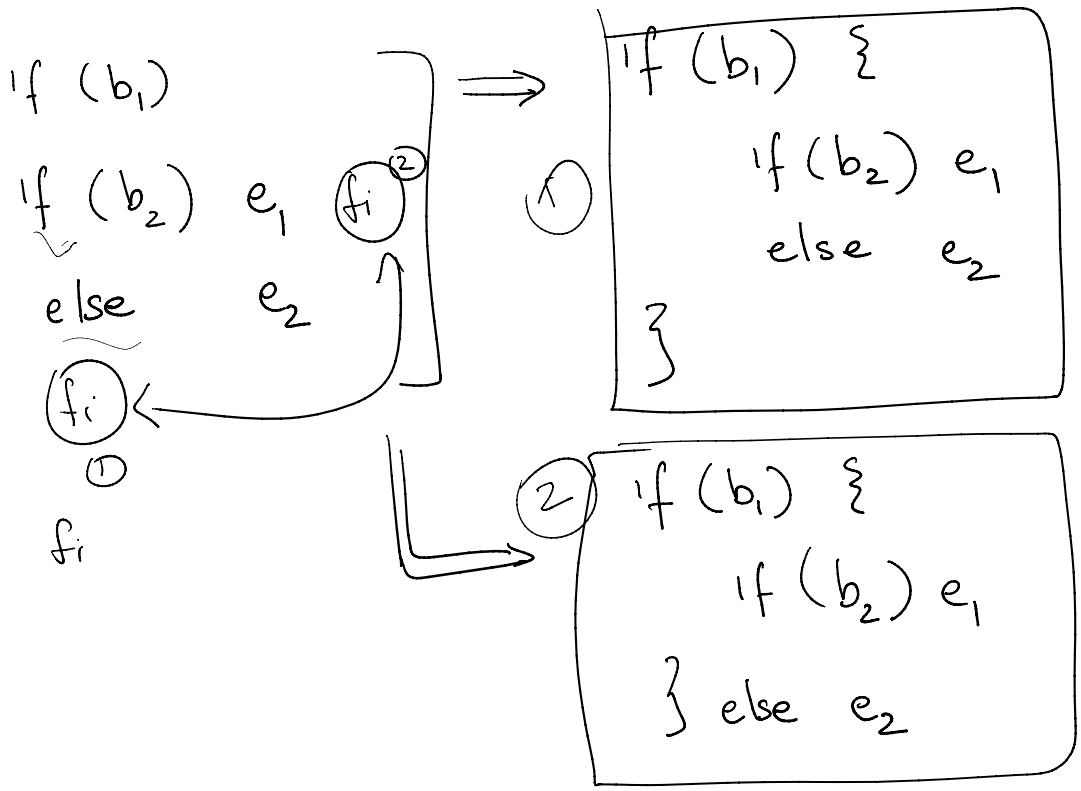
| expr — Int Literal

[| (expr) — (expr)]

| ...

---

if b_1 then if b_2 then e_1 else e_2



---

stmt ::= ...

| if (expr) then stmt fi

| if (expr) then stmt else stmt fi

How Do They Work?

Regular expressions

"c"	$r_1 + r_2$	$r_1 \cdot r_2$	r^*
Concrete string	regular expr r_1 or regular expr r_2	r_1 followed by r_2	Zero or more occurrences of r.

'0' - '9' = '0' + '1' + '2' + ... + '9'.

A string matches r^+ if it can be split into one or more occurrences of r.

$r^+ = \underbrace{r^*}_{\text{zero or more}} + r = r^*$ (zero or one)

X

$r^+ = r^* \cdot r$ ✓

Q1: Does "abc" match $('a' + 'b' + 'c')^*$?

A string matches $('a' + 'b' + 'c')^*$ if it can be broken into zero or more pieces, each of which match $'a' + 'b' + 'c'$.

"abc" = "a" · "b" · "c" . Yes.

Q2: Does "abcd" match $(a + b + c)^*$?

No.

Q3: Does "abc" match $(a+b+c)^* \cdot c^*$?

"abc" = "ab" · "c" = "abc" · ""

"ab": $(a+b+c)^* \cdot c^*$; "abc": $(a+b+c)^*$; "": c^*

$(a+b+c)^*$: "" | c^*

$(a+b+c)^* \cdot c^* = (a+b+c)^*$

Q4: Why does "abc" match $(a+b+c)^*$?

Shouldn't $(a+b+c)^*$ mean that the same letter (a or b or c) is repeated multiple times?

Counterpoint: This would be $a^* + b^* + c^*$

$(a+b+c)^* \neq a^* + b^* + c^*$

Q5: How does one intuitively understand

$(a+b+c)^* \cdot + ?$

$(a+b+c)^* \cdot c^+$?

$(a+b+c)^*$ | c^+

Q6: For example, in C, x5 is a valid variable name., _x5

On the other hand, 5x is not a valid variable name.

alphabet + number + underscore

$(alpha + underscore) \cdot (a+n+u)^*$

Q7: What is a regular expression for email addresses?

person @ example.com

$$\underbrace{(A+N+D+U)}_{\text{alpha number dot underscore}}^{+} \cdot @ \cdot (A+N+D+U)^{+}$$

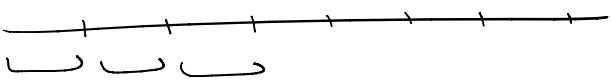
Q8: Given a string s & a regular expression r,
- find if s matches r.

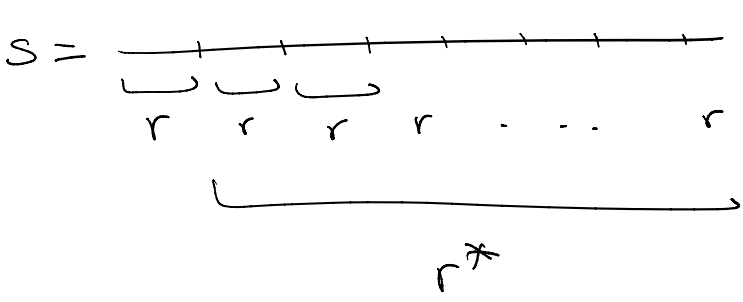
"("	$r_1 + r_2$	$r_1 \cdot r_2$	r^*
Concrete string	regular expr r_1 or regular expr r_2	r_1 followed by r_2	Zero or more occurrences of r.

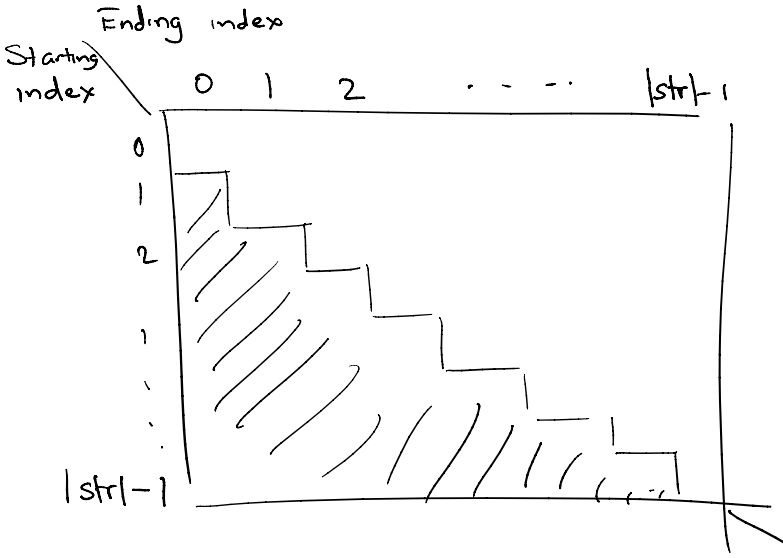
The issue with greedy algorithms:

- How to know if we'll need to backtrack?

---

s =  matches r^*





regex r

All subexpressions of r

r = "str"

If |str| = n & regex r is of size k then $O(n^2k)$ entries in the table

Subexpressions of r.

To fill in table(i j String(s)), (String(s): shape of regex; input string)

check if str[i..j] = s

table(i j $r_1 + r_2$)

= table(i j r_1) || table(i j r_2)

table (i j $r_1 \cdot r_2$)

= Check if $\exists$ k $i \leq k \leq j$

s.t. table (i k r_1) && table (k j r_2)

table (i j r^*)

= Check if $\exists$ k $i \leq k \leq j$

s.t table (i k r) && table (k j r^*)

If |str| = n & regex r is of size k then $O(n^2 k)$ entries in the table

Filling in a single entry might require $O(n)$ time

On the whole, we need $O(n^3 k)$ time to check if a string of size n matches a regex of size k.