

CS311—Computational Structures

Regular Languages and Regular Expressions

Recap

- We have defined two different things:
 - Regular *Languages*
 - the sets \emptyset , Λ , $\{a\}$ and the sets we can make by applying \cup , \cdot and $*$ to other regular languages
 - mathematical object: sets of sequences from the alphabet
 - Regular *Expressions*
 - formulae involving \emptyset , Λ , a , parenthesis and the operations $+$, juxtaposition, and $*$
 - just syntax with no meaning (yet)

The meaning of an RE

- We'll write \mathcal{M} as the meaning function

$$\mathcal{M} [a] = \{a\} \qquad \mathcal{M} [\Lambda] = \{\Lambda\}$$

$$\mathcal{M} [\emptyset] = \{ \}$$

$$\mathcal{M} [pq] = \mathcal{M} [p] \cdot \mathcal{M} [q]$$

$$\mathcal{M} [p+q] = \mathcal{M} [p] \cup \mathcal{M} [q]$$

$$\mathcal{M} [p^*] = \mathcal{M} [p]^*$$

Analogy: the meaning of Numerals

- First, define numerals. In words:

The digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

A decimal numeral is either a digit, or a decimal numeral followed by a digit.

- Or, as a *grammar*

$$D \rightarrow 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$$
$$N \rightarrow D | ND$$

The *meaning* of a Numeral

$$M[0] = 0$$

$$M[1] = 1$$

$$M[2] = 2$$

$$M[3] = 3$$

$$M[4] = 4$$

$$M[5] = 5$$

$$M[6] = 6$$

$$M[7] = 7$$

$$M[8] = 8$$

$$M[9] = 9$$

$$M[ND] = 10 \times M[N] + M[D]$$

Example:

$$M[257] = 10 \times M[25] + M[7]$$

$$M[25] = 10 \times M[2] + M[5]$$

$$= 10 \times 2 + 5$$

$$= 20 + 5 = 25$$

$$M[257] = 10 \times 25 + 7$$

$$= 250 + 7 = 257$$

Example REs

- $a+b+c$
- $(a+b)c$
- $(a+b)(c+d)$
- $a+b+c^*$
- $(a+b)^*$

Properties of REs

Hein §11.1.2

Properties of Regular Expressions

(11.1)

1. (+ properties)

$$R + T = T + R,$$

$$R + \emptyset = \emptyset + R = R,$$

$$R + R = R,$$

$$(R + S) + T = R + (S + T).$$

2. (\cdot properties)

$$R\emptyset = \emptyset R = \emptyset,$$

$$R\Lambda = \Lambda R = R,$$

$$(RS)T = R(ST).$$

3. (Distributive properties)

$$R(S + T) = RS + RT,$$

$$(S + T)R = SR + TR.$$

More properties

(Closure properties)

4. $\emptyset^* = \Lambda^* = \Lambda$.

5. $R^* = R^*R^* = (R^*)^* = R + R^*$,

$$R^* = \Lambda + R^* = (\Lambda + R)^* = (\Lambda + R)R^* = \Lambda + RR^*,$$

$$R^* = (R + \dots + R^k)^* \quad \text{for any } k \geq 1,$$

$$R^* = \Lambda + R + \dots + R^{k-1} + R^k R^* \quad \text{for any } k \geq 1.$$

6. $R^*R = RR^*$.

7. $(R + S)^* = (R^* + S^*)^* = (R^*S^*)^* = (R^*S)^*R^* = R^*(SR^*)^*$.

8. $R(SR)^* = (RS)^*R$.

9. $(R^*S)^* = \Lambda + (R + S)^*S$,

$$(RS^*)^* = \Lambda + R(R + S)^*.$$

- We can *prove* all of these properties of REs by examining the meaning of the REs as sets of strings
- Example:

$$\begin{aligned} \mathcal{M}[R+T] &= \mathcal{M}[R] \cup \mathcal{M}[T] \quad (\text{defn of } +) \\ &= \mathcal{M}[T] \cup \mathcal{M}[R] \quad (\cup \text{ is commutative}) \\ &= \mathcal{M}[T+R] \quad (\text{defn of } +) \end{aligned}$$

- Let's try some others!

Boring!

- This is all pretty mechanical
- An ideal task for a computer program
- How can we represent REs in a programming language?

Regular Expressions in ML

```
datatype RE =  
  Lambda  
| Empty  
| C of string  
| Union of RE * RE  
| Concat of RE * RE  
| Star of RE ;
```

The Regular Expressions

1. Λ , \emptyset , and a are regular expressions for all $a \in A$.
2. If R and S are regular expressions, then the following expressions are also regular: (R) , $R + S$, $R \cdot S$, and R^* .

Not only does this define a datatype, it also defines 6 functions that make values of the type.

```
infixr 5 || ; (* This will be the infix union operation - it will have precedence 5 *)  
infixr 6 ^^ ; (* and this will be the infix concatenation operation, with (higher) precedence 6 *)
```

```
fun Lambda ^^ b = b      (* property 2.2 *)  
  | a ^^ Lambda = a     (* property 2.2 *)  
  | Empty ^^ b = Empty  (* property 2.1 *)  
  | a ^^ Empty = Empty  (* property 2.1 *)  
  | a ^^ b = Concat (a, b);
```

```
fun Empty || b = b      (* property 1.2 *)  
  | a || Empty = a     (* property 1.2 *)  
  | a || b = Union (a, b);
```

Properties of Regular Expressions

1. (+ properties)

$$R + T = T + R,$$

$$R + \emptyset = \emptyset + R = R,$$

$$R + R = R,$$

$$(R + S) + T = R + (S + T).$$

2. (\cdot properties)

$$R\emptyset = \emptyset R = \emptyset,$$

$$R\Lambda = \Lambda R = R,$$

$$(RS)T = R(ST).$$

```
(* String manipulation functions *)
```

```
fun first(str): string = String.extract(str, 0, SOME 1); (* The first character of a string *)
```

```
fun rest(str): string = String.extract(str, 1, NONE); (* the rest of the string; all chars but the first *)
```

```
(* The function string makes an RE from a string. So (string "if") = Concat(C "i", C "f")  
This is for convenience and readability *)
```

```
fun string (str) =  
  case String.size str of  
    0 => Lambda  
  | 1 => C str  
  | _ => Concat (C (first str), string (rest str)); (* _ matches anything, so this is  
the default case *)
```

```
val alpha = C "a" || C "b" || C "c"; (* The same as Union(C "a", Union(C "b", C "c")),  
but more readable *)
```

```
val digit = C "0" || C "1" || C "2";
```

```
val ite = string "if" || string "then" || string "else";
```

```
val ident = alpha ^^ Star (alpha || digit);
```

```
val number = digit ^^ Star digit;
```

```
(* generate r generates a list containing some of the strings in the language represented
   by the RE r. In general, any RE containing Star will generate an infinite language; to
   avoid waiting forever, this implementation of generate limits itself to 0,1,2, and 3
   repetitions on the starred item. *)
```

```
fun generate (Union (l,r)) = generate(l) @ generate(r)
  | generate (Concat (l,r)) = product (generate l) (generate r)
  | generate (Star (r)) = generate Lambda @
    generate r @
    product (generate r) (generate r) @
    product (generate r) (product (generate r) (generate r))
  | generate (C c) = [c]
  | generate Lambda = [""]
  | generate Empty = []
```

```
(* product l1 l2 is the "language product" of the two lists l1 and l2. See Hein page 43. *)
```

```
and product [] ys = []
  | product (x::xs) ys = (map (fn y => x^y) ys ) @ product xs ys;
```

```
val p = C "p";
val q = C "q";
val r = C "r";
val z = C "0";
val i = C "1";
```