

# Lexical Analysis

CMPT 379: Compilers

Instructor: Anoop Sarkar

[anoopsarkar.github.io/compilers-class](https://anoopsarkar.github.io/compilers-class)

# Regular Languages

- The set of regular languages: each element is a regular language
  - $R = \{R_1, R_2, \dots, R_n, \dots\}$
- Each regular language is a formal language, i.e. a set of strings

$$R_1 = \{a\},$$

$$R_2 = \{a^n : n > 0\} = \{a, aa, aaa, \dots\},$$

$$R_3 = \{b\},$$

$$R_4 = \{ba, ab\},$$

$$R_5 = \{b^n : n \geq 0\} = \{\varepsilon, b, bb, bbb, \dots\},$$

...

# Regular Expressions and Regular Languages

- Meaning function  $L(r)$
- $L(r)$  = The *meaning* of regexp  $r$  is the regular language for  $r$ 
  - $L(a^*) = \{a^n : n \geq 0\} = \{\epsilon, a, aa, aaa, \dots\}$
  - $L() = \{\epsilon\}$
  - $L(a) = \{a\}$
  - $L(r_1 | r_2) = L(r_1) \cup L(r_2)$
  - $L(r_1 r_2) = \{xy \mid x \in L(r_1), y \in L(r_2)\}$
  - $L(r_1^2) = \{xy \mid x \in L(r_1), y \in L(r_1)\}$
  - $L(r_1^*) = L(r_1)^0 \cup L(r_1)^1 \cup L(r_1)^2 \cup L(r_1)^3 \dots$

Integer: a non-empty sequence of digits

`digit = (0|1|2|3|4|5|6|7|8|9)`

`{digit}{digit}* ➡ {digit}+`

Identifier: sequence of letters or digits, starting with a letter

```
digit = [0-9]  
letter = [a-zA-Z]
```

```
{letter}({letter}|{digit})*
```

Whitespace: a non-empty sequence of blanks,  
newlines and tabs

$$(\text{" " | "\t" | "\n"})^+$$

# Pattern definition for numbers

`digit = [0-9]`

`digits = [0-9] +`

`opt_frac = ("."{digits})?`

`opt_exp = ((e|E)(\+|\-)?{digits})?`

`num = {digits}{opt_frac}{opt_exp}`

`345, 345.04 , 2e-7, 2e7, 2e+7, 3.14e5`

# Lex regular expressions

Expression	Matches	Example	Using core operators
<code>c</code>	non-operator character <code>c</code>	<code>a</code>	
<code>\c</code>	character <code>c</code> literally	<code>\*</code>	
<code>"s"</code>	string <code>s</code> literally	<code>***</code>	
<code>.</code>	any character but newline	<code>a.*b</code>	
<code>^</code>	beginning of line	<code>^abc</code>	used for matching
<code>\$</code>	end of line	<code>abc\$</code>	used for matching
<code>[s]</code>	any one of characters in string <code>s</code>	<code>[abc]</code>	<code>(a b c)</code>
<code>[^s]</code>	any one character not in string <code>s</code>	<code>[^a]</code>	<code>(b c)</code> $\Sigma = \{a, b, c\}$
<code>r*</code>	zero or more strings matching <code>r</code>	<code>a*</code>	
<code>r+</code>	one or more strings matching <code>r</code>	<code>a+</code>	<code>aa*</code>
<code>r?</code>	zero or one <code>r</code>	<code>a?</code>	<code>(a ε)</code>
<code>r{m,n}</code>	between <code>m</code> and <code>n</code> occurrences of <code>r</code>	<code>a{2,3}</code>	<code>(aa aaa)</code>
<code>r<sub>1</sub>r<sub>2</sub></code>	an <code>r<sub>1</sub></code> followed by an <code>r<sub>2</sub></code>	<code>ab</code>	
<code>r<sub>1</sub>/r<sub>2</sub></code>	an <code>r<sub>1</sub></code> or an <code>r<sub>2</sub></code>	<code>a b</code>	
<code>(r)</code>	same as <code>r</code>	<code>(a b)</code>	
<code>r<sub>1</sub>/r<sub>2</sub></code>	<code>r<sub>1</sub></code> when followed by an <code>r<sub>2</sub></code>	<code>abc/123</code>	<code>r<sub>1</sub>r<sub>2</sub></code> used for matching



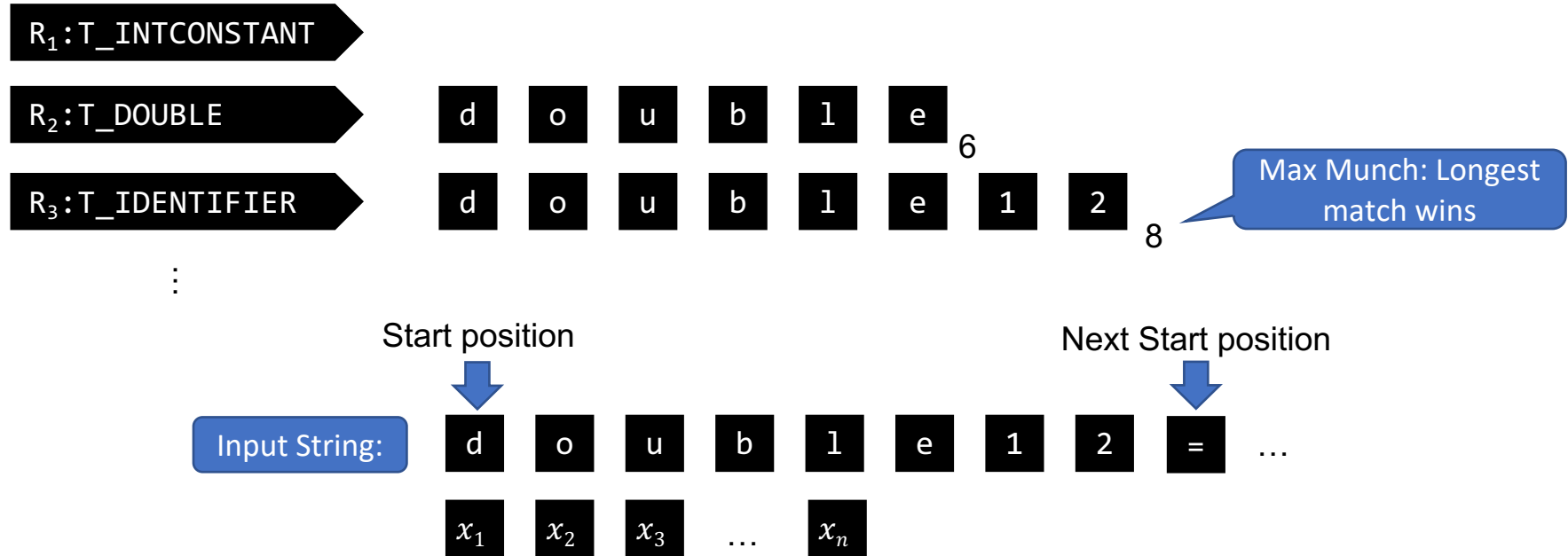
# Regular Expressions for Lexical Analysis

# Regular Expressions for Lexical Analysis

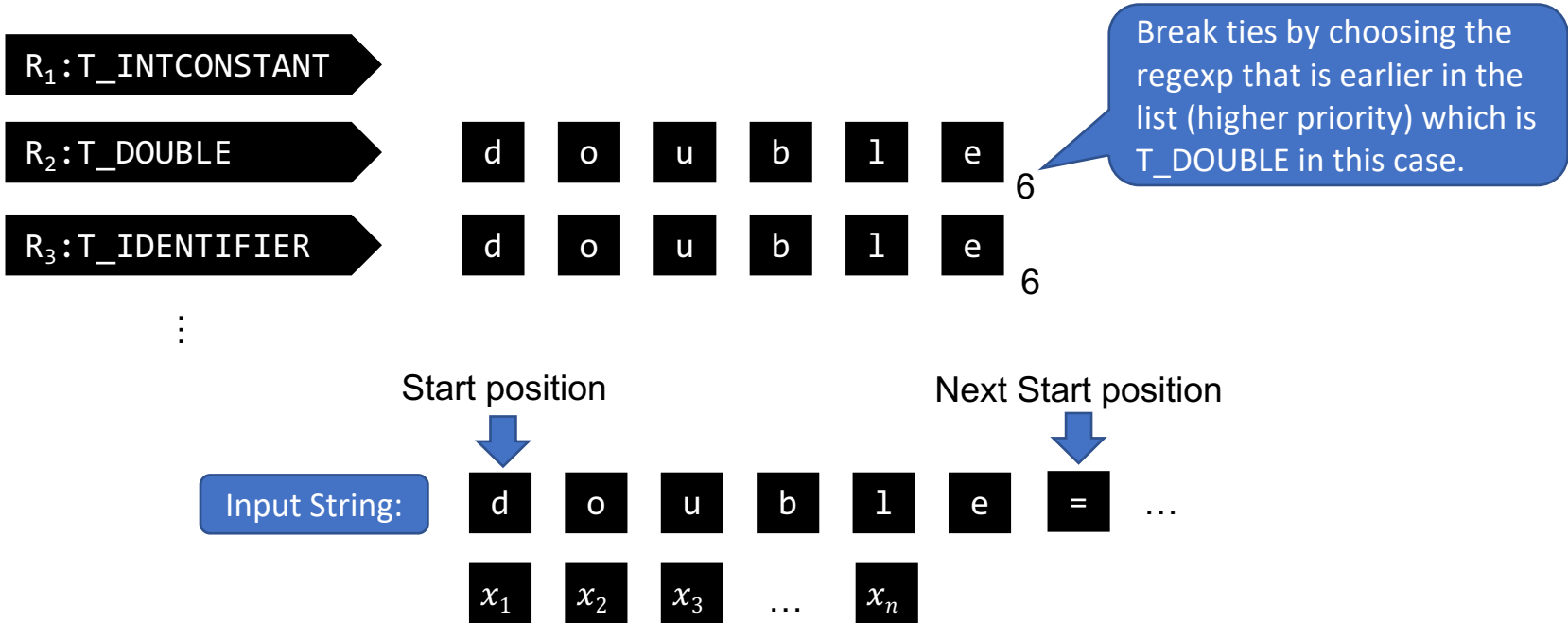
- Write a regexp pattern for each token:
  - $R_1:T\_INTCONSTANT = \{digit\}^+$
  - $R_2:T\_DOUBLE = \text{“double”}$
  - $R_3:T\_IDENTIFIER = \{letter\}(\{letter\}|\{digit\})^+$
  - and so on ...
- Construct an ordered list  $R$  containing all  $t$  regexps.
  - $R = [R_1, R_2, R_3, \dots, R_t]$

The order of regexps is important and provided as part of the lexer definition

# Regular Expressions for Lexical Analysis



# Regular Expressions for Lexical Analysis



# Regular Expressions for Lexical Analysis

$R_1:T\_INTCONSTANT$

$R_2:T\_DOUBLE$

$R_3:T\_IDENTIFIER$

:

What if no regexp matches?

Create a new **Error** regexp that matches any input.

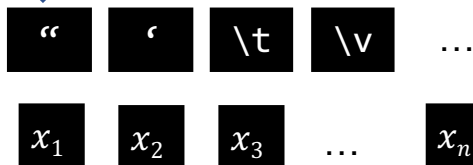
Put the **Error** regexp as the last in the list (the lowest priority).

So when it matches we know there was a lexical analysis error.

Start position



Input String:



# Regular Expressions for Lexical Analysis

$R_1$ :T\_INTCONSTANT

$R_2$ :T\_DOUBLE

$R_3$ :T\_IDENTIFIER

⋮

input:  $x_1, \dots, x_n$

result=list()

$s = 1$

while  $s < n$ :

for all regexps  $R_k$ :

match( $R_k, x_s, \dots, x_n$ ) =  $i_k$

$m, i_m = \max(i_1, \dots, i_t)$

result.append( $(R_m, i_m)$ )

$s = i_m + 1$

return(result, s)

Break ties by choosing  
smallest  $m$  value  
(higher priority regexp)

Input String:

$x_1$

$x_2$

$x_3$

...

$x_n$

# Regexps in Lexical Analysis

- Regular expressions are a concise notation for string patterns
- Use in lexical analysis requires small extensions
  - Maximal munch to handle ambiguous matches
  - Break ties using priority ordering
  - Handle errors
- A good algorithm for lexical analysis will:
  - Require only single pass over the input
  - Few operations per character (lookup table for matching a regexp)