LEX2: Regular Expressions

Lexical Analysis

CMPT 379: Compilers

Instructor: Anoop Sarkar

anoopsarkar.github.io/compilers-class

Regular Languages

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

$$
\begin{aligned}
R_1 &= \{a\},\\
R_2 &= \{a^n : n > 0\} = \{a, aa, aaa, \ldots\},\\
R_3 &= \{b\},\\
R_4 &= \{ba, ab\},\\
R_5 &= \{b^n : n \geq 0\} = \{\varepsilon, b, bb, bbb, \ldots\},\\
&\ldots
\end{aligned}
$$

Regular Expressions and Regular Languages

- Meaning function L(r)
- L(r) = The *meaning* of regexp r is the regular language for r
  - L(a*) = $\{\varepsilon, a, aa, aaa, \ldots\}$
  - L() = $\{\varepsilon\}$
  - 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$_1$*) = $L(r_1)^0 \cup L(r_1)^1 \cup L(r_1)^2 \cup L(r_1)^3 \ \ldots$

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

```
(" "|"\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
c	non-operator character c	a	
\c	character c literally	*	
"*s*"	string s literally	"**"	
.	any character but newline	a.*b	
^	beginning of line	^abc	used for matching
$	end of line	abc$	used for matching
[s]	any one of characters in string s	[abc]	(a|b|c)
[^s]	any one character not in string s	[^a]	(b|c) $\Sigma = \{a, b, c\}$
*r**	zero or more strings matching r	a*	
r+	one or more strings matching r	a+	aa*
r?	zero or one r	a?	$(a\vert\varepsilon)$
r{m,n}	between m and n occurences of r	a{2,3}	(aa|aaa)
r_1r_2	an r_1 followed by an r_2	ab	
$r_1\vert r_2$	an r_1 or an r_2	a|b	
(r)	same as r	(a|b)	
r_1/r_2	r_1 when followed by an r_2	abc/123	r_1r_2 used for matching

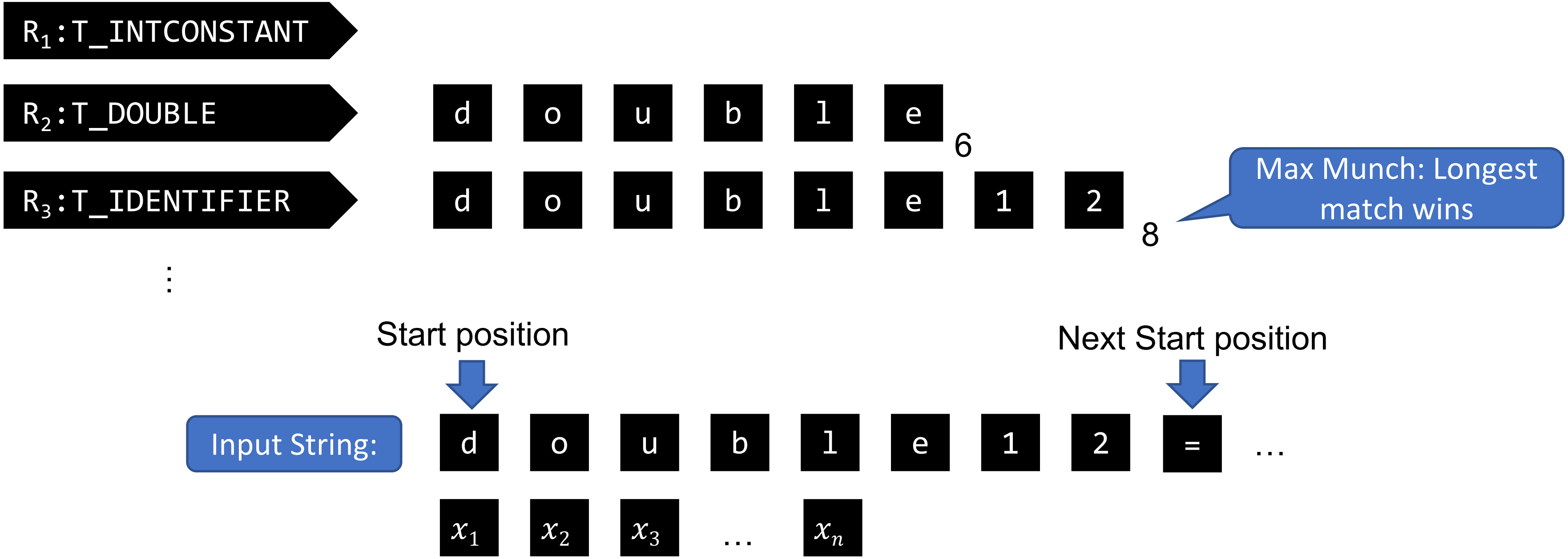
Regular Expressions for Lexical Analysis

Regular Expressions for Lexical Analysis

- Write a regexp pattern for each token:
  - R_1:T_INTCONSTANT = digit+
  - R_2:T_DOUBLE = “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, …, R_t]

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

Regular Expressions for Lexical Analysis



R_1:T_INTCONSTANT

R_2:T_DOUBLE

R_3:T_IDENTIFIER

⋮

d o u b l e 6

d o u b l e 1 2 8

Max Munch: Longest match wins

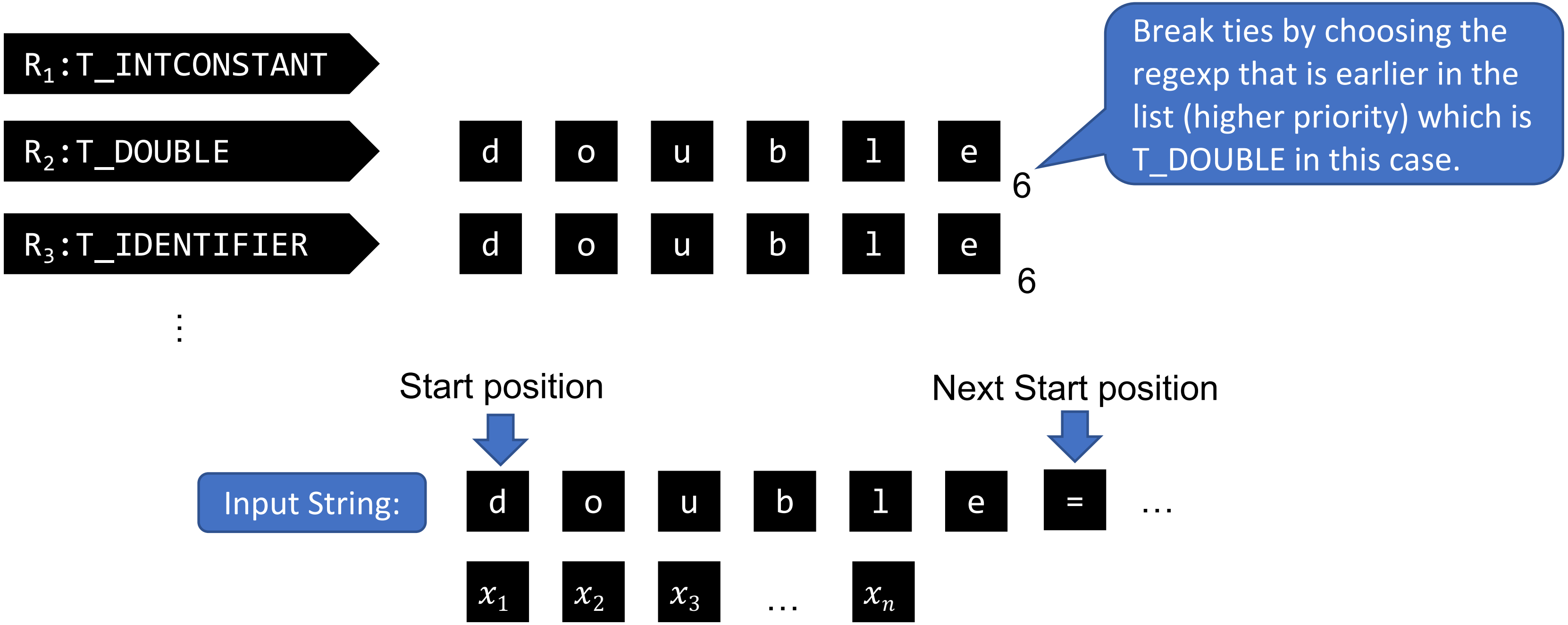
Start position

Next Start position

Input String: d o u b l e 1 2 = …

x_1 x_2 x_3 … x_n

Regular Expressions for Lexical Analysis



R_1:T_INTCONSTANT

R_2:T_DOUBLE d o u b l e 6

R_3:T_IDENTIFIER d o u b l e 6

⋮

Break ties by choosing the regexp that is earlier in the list (higher priority) which is T_DOUBLE in this case.

Start position

Next Start position

Input String: d o u b l e = …

x_1 x_2 x_3 … x_n

Regular Expressions for Lexical Analysis

R_1:T_INTCONSTANT

R_2:T_DOUBLE

R_3:T_IDENTIFIER

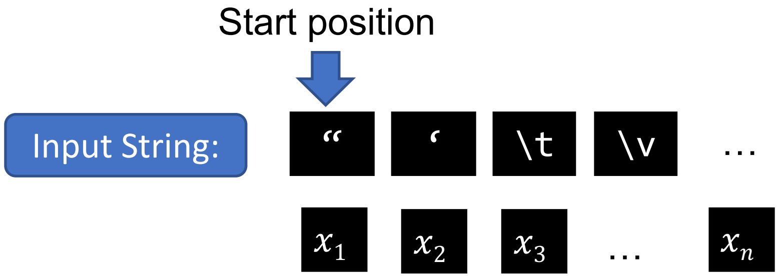
⋮

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:

“ ‘ \t \v …

x_1 x_2 x_3 … x_n

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
  - 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)