Análise Lexical

- Nesta aula → apenas uma ligeira introdução.
- Mais tarde → usaremos teoria de autómatos finitos e linguagens regulares.

- O exemplo desta aula foi retirado do livro "Programming Language Pragmatics" (3rd edition), pág 51.
- Analisador lexico para uma "calculator language".

```
read  A
read  B
sum   := A + B
write sum
write sum / 2
```

exemplo de programa

Tokens:

keyword → "read" ou "write"

assign → :=

plus → +

minus → -

times → *

div → /

lparen → (

rparen →)

number → digit digit* |
digit* (. digit | digit .) digit*

id → letter (letter | digit)*
exceto para read ou write

Expressões Regulares

---

letter → a | b | c | ... | z | A | B | C | ... | Z

digit → 0 | 1 | ... | 9

A definição dos tokens foi feita com expressões regulares.

- mais tarde veremos a def. formal
- por agora:

| ⇝ ou

* ⇝ 0 ou mais

- Exemplos:

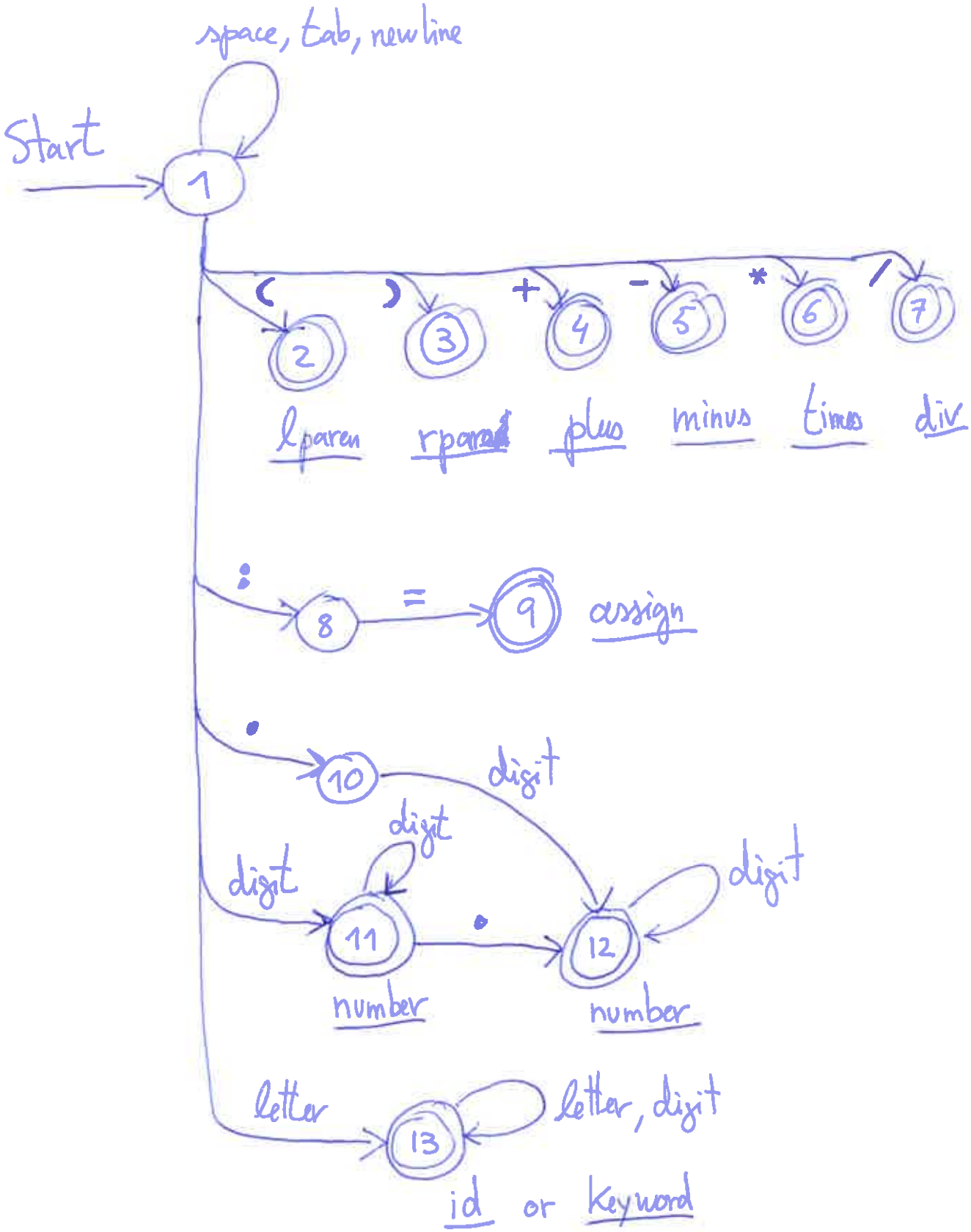
:= é uma expressão regular definida pelo símbolo : seguido do símbolo =

digit digit* é uma expr. regular.

⇝ um digit seguido de 0 ou mais digits (ou seja, 1 ou mais digits)

em que um digit pode ser '0' ou '1' ou '2' ou ... '9'

- Podemos fazer um autómato finito para reconhecer os tokens contidos no input.
- O autómato consome o input caracter a caracter.



- O autómato é determinístico: Dado um estado e um símbolo de input, sabemos qual o estado onde vamos parar.

- Os estados finais (indicados com um duplo círculo) correspondem a um reconhecimento de um token.

- "longest possible rule" → o autómato tenta consumir o máximo de input possível antes de retornar o token.

- Nalguns casos apenas conseguimos reconhecer o token após ter consumido um símbolo/caracter a mais. É o caso dos estados 11, 12, e 13 no autómato anterior.

  → é necessário "desconsumir" esse caracter antes de processar o próximo token.

Esboço de código para um scanner adhoc


```
do {                                                    // state 1
   ch = readchar();
} while ( ch ∈ {' ', '\t', '\n'} )
if ( ch == '(' ) return LPAREN;                         // state 2
else if ( ch == ')' ) return RPAREN;                    // state 3
...
else if ( ch == '/' ) return DIV;                       // state 7
else if ( ch == ':' ) {                                 // state 8
        ch = readchar();
        if ( ch == '=' ) return ASSIGN;                 // state 9
        else return ERROR;
     }
else if ( ch == '.' ) {                                 // state 10
        ch = readchar();
        if ( isdigit(ch) ) {                            // state 12
             do {
                ch = readchar();
             } while ( isdigit(ch) );
             unread(ch);
             return NUMBER;
        }
        else return ERROR;
     }
```

```
else if ( isdigit(ch) ) { // state 11
        :
    }
else if ( isletter(ch) ) { // state 13
        :
    }
else
    return ERROR;
```

Cada vez que este código é executado obtemos mais um token.