Lexer: Tabellenbasierte Implementierung

Carsten Gips (HSBI)

Lexer: Erzeugen eines Token-Stroms aus einem Zeichenstrom

```
/* demo */
a= [5  , 6]     ;
```

Lexer: Erzeugen eines Token-Stroms aus einem Zeichenstrom

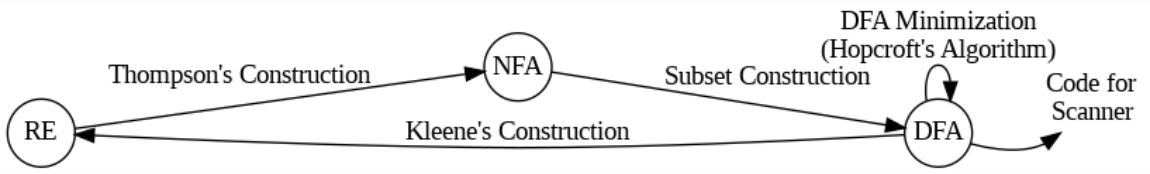
```
/* demo */
a= [5  , 6]     ;
```

```
<ID, "a"> <ASSIGN> <LBRACK> <NUM, 5> <COMMA> <NUM, 6> <RBRACK> <SEMICOL>
```

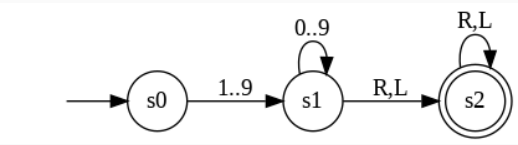
Definition wichtiger Begriffe

- **Token**: Tupel (Tokenname, optional: Wert)
- **Lexeme**: Sequenz von Zeichen im Eingabestrom, die auf ein Tokenpattern matcht und vom Lexer als Instanz dieses Tokens identifiziert wird.
- **Pattern**: Beschreibung der Form eines Lexems

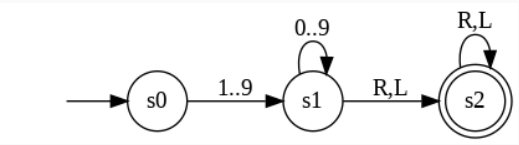
Erkennung mit RE und DFA



Erkennen von Zeichenketten für Strickmuster: “10LRL”



Erkennen von Zeichenketten für Strickmuster: “10LRL”



	0	1,..,9	R, L	*
s0	se	s1	se	se
s1	s1	s1	s2	se
s2	se	se	s2	se
se	se	se	se	se

Tabellenbasierte Implementierung

```
def nextToken():
    state = s0; lexeme = ""; stack = Stack()

    while (state != se):
        consume()       # hole nächstes Zeichen (peek)
        lexeme += peek
        stack.push(state)
        state = TransitionTable[state, peek]

    while (state != s2 and stack.notEmpty()):
        state = stack.pop(); putBack(lexeme.truncate())

    if state == s2: return s2(lexeme)
    else: return invalid()
```

Hinweis: Direkt codierte Implementierung

Wrap-Up

- Zusammenhang DFA, RE und Lexer
- Implementierungsansatz: Tabellenbasiert (DFA-Tabellen)

LICENSE