

# Vorlesung Compilerbau (SoSe2018)

## Teil 7: Shift-Reduce & Präzedenz-Parsing



### Vorlesung Compilerbau: Syntaktische Analyse – Parsing IV Shift-Reduce & Präzedenz-Parsing

Vorlesung des BA-Studiums  
Prof. Johann Christoph Freytag, Ph.D.  
Institut für Informatik, Humboldt-Universität zu Berlin  
SoSe2018

© Prof. J.C. Freytag, Ph.D.

7.1

## Shift-Reduce-Parser

**Bemerkung:** Handle-Pruning wird durch sogenannte SR-Parser (Shift-Reduce-Parser) implementiert

### SR- Parser

- benutzt Kellerspeicher und Eingabepuffer als Datenstrukturen
- Aktionen:
  - initialisiere den Keller mit dem Zeichen \$
  - wiederhole folgende Schritte, bis das oberste Kellerzeichen das Startsymbol ist und das Eingabesymbol das \$-Zeichen
    - a) schiebe (**shift**) solange null oder mehrere Eingabesymbole auf den Keller, bis ein Handle  $\beta$  auf dem Keller liegt
    - b) reduziere (**reduce**) den Handle:
      - falls ein Handle  $A \rightarrow \beta$  auf dem Keller liegt, reduziere wie folgt:
        - i) nimm  $|\beta|$  Symbole aus dem Keller (**pop**)
        - ii) Ersetze diese durch  $A$  auf dem Keller (**push**)

© Prof. J.C. Freytag, Ph.D.

6.2

# Vorlesung Compilerbau (SoSe2018)

## Teil 7: Shift-Reduce & Präzedenz-Parsing

### Beispiel für $x - 2 * y$

```

1 S → E
2 E → E + T
3   | E - T
4   | T
5 T → T * F
6   | T / F
7   | F
8 F → num
9   | id
    
```

| Keller                        | Eingabe       | Aktion     |
|-------------------------------|---------------|------------|
| \$                            | id - num * id | Shift: id  |
| id \$                         | - num * id    | Reduce: 9  |
| <factor> \$                   | - num * id    | Reduce: 7  |
| <term> \$                     | - num * id    | Reduce: 4  |
| <expr> \$                     | - num * id    | Shift: -   |
| - <expr> \$                   | num * id      | Shift: num |
| num - <expr> \$               | * id          | Reduce: 8  |
| <factor> - <expr> \$          | * id          | Reduce: 7  |
| <term> - <expr> \$            | * id          | Shift: *   |
| * <term> - <expr> \$          | id            | Shift: id  |
| id * <term> - <expr> \$       |               | Reduce: 9  |
| <factor> * <term> - <expr> \$ |               | Reduce: 5  |
| <term> - <expr> \$            |               | Reduce: 3  |
| <expr> \$                     |               | Reduce: 1  |
| <goal> \$                     |               | Accept     |

© Prof. J.C. Freytag, Ph.D.

6.3

### Beispiel für $x - 2 * y$

```

1 S → E
2 E → E + T
3   | E - T
4   | T
5 T → T * F
6   | T / F
7   | F
8 F → num
9   | id
    
```


| Keller | Eingabe       | Aktion    |
|--------|---------------|-----------|
| \$     | id - num * id | Shift: id |

© Prof. J.C. Freytag, Ph.D.

7.4

# Vorlesung Compilerbau (SoSe2018)

## Teil 7: Shift-Reduce & Präzedenz-Parsing




### Beispiel für $x - 2 * y$

| Keller | Eingabe       | Aktion    |
|--------|---------------|-----------|
| \$     | id - num * id | Shift: id |
| \$ id  | - num * id    | Reduce: 9 |

- 1 S  $\rightarrow$  E
- 2 E  $\rightarrow$  E + T
- 3     | E - T
- 4     | T
- 5 T  $\rightarrow$  T \* F
- 6     | T / F
- 7     | F
- 8 F  $\rightarrow$  num
- 9     | id

© Prof. J.C. Freytag, Ph.D. 7.5



### Beispiel für $x - 2 * y$


| Keller      | Eingabe       | Aktion    |
|-------------|---------------|-----------|
| \$          | id - num * id | Shift: id |
| \$ id       | - num * id    | Reduce: 9 |
| \$ <factor> | - num * id    | Reduce: 7 |

- 1 S  $\rightarrow$  E
- 2 E  $\rightarrow$  E + T
- 3     | E - T
- 4     | T
- 5 T  $\rightarrow$  T \* F
- 6     | T / F
- 7     | F
- 8 F  $\rightarrow$  num
- 9     | id

© Prof. J.C. Freytag, Ph.D. 7.6

# Vorlesung Compilerbau (SoSe2018)

## Teil 7: Shift-Reduce & Präzedenz-Parsing




### Beispiel für $x - 2 * y$

|   |                            |
|---|----------------------------|
| 1 | $S \rightarrow E$          |
| 2 | $E \rightarrow E + T$      |
| 3 | $E - T$                    |
| 4 | $T$                        |
| 5 | $T \rightarrow T * F$      |
| 6 | $T / F$                    |
| 7 | $F$                        |
| 8 | $F \rightarrow \text{num}$ |
| 9 | $\text{id}$                |

| Keller      | Eingabe  | Aktion    |
|-------------|--|-----------|
| \$          | <span style="color: red;">id</span> - num * id | Shift: id |
| \$ id       | - num * id                                     | Reduce: 9 |
| \$ <factor> | - num * id                                     | Reduce: 7 |
| \$ <term>   | - num * id                                     | Reduce: 4 |

© Prof. J.C. Freytag, Ph.D.
7.7



### Beispiel für $x - 2 * y$

|   |                            |
|---|----------------------------|
| 1 | $S \rightarrow E$          |
| 2 | $E \rightarrow E + T$      |
| 3 | $E - T$                    |
| 4 | $T$                        |
| 5 | $T \rightarrow T * F$      |
| 6 | $T / F$                    |
| 7 | $F$                        |
| 8 | $F \rightarrow \text{num}$ |
| 9 | $\text{id}$                |

| Keller      | Eingabe  | Aktion    |
|-------------|--|-----------|
| \$          | <span style="color: red;">id</span> - num * id | Shift: id |
| \$ id       | - num * id                                     | Reduce: 9 |
| \$ <factor> | - num * id                                     | Reduce: 7 |
| \$ <term>   | - num * id                                     | Reduce: 4 |
| \$ <expr>   | - num * id                                     | Shift: -  |

© Prof. J.C. Freytag, Ph.D.
7.8

# Vorlesung Compilerbau (SoSe2018)

## Teil 7: Shift-Reduce & Präzedenz-Parsing

### Beispiel für $x - 2 * y$

```

1 | S → E
2 | E → E + T
3 |   | E - T
4 |   | T
5 | T → T * F
6 |   | T / F
7 |   | F
8 | F → num
9 |   | id
    
```

| Keller      | Eingabe              | Aktion     |
|-------------|----------------------|------------|
| \$          | <b>id</b> - num * id | Shift: id  |
| \$ id       | - num * id           | Reduce: 9  |
| \$ <factor> | - num * id           | Reduce: 7  |
| \$ <term>   | - num * id           | Reduce: 4  |
| \$ <expr>   | - num * id           | Shift: -   |
| \$ <expr> - | <b>num</b> * id      | Shift: num |

© Prof. J.C. Freytag, Ph.D.

7.9

### Beispiel für $x - 2 * y$

```

1 | S → E
2 | E → E + T
3 |   | E - T
4 |   | T
5 | T → T * F
6 |   | T / F
7 |   | F
8 | F → num
9 |   | id
    
```

| Keller          | Eingabe              | Aktion     |
|-----------------|----------------------|------------|
| \$              | <b>id</b> - num * id | Shift: id  |
| \$ id           | - num * id           | Reduce: 9  |
| \$ <factor>     | - num * id           | Reduce: 7  |
| \$ <term>       | - num * id           | Reduce: 4  |
| \$ <expr>       | - num * id           | Shift: -   |
| \$ <expr> -     | <b>num</b> * id      | Shift: num |
| \$ <expr> - num | <b>*</b> id          | Reduce: 8  |

© Prof. J.C. Freytag, Ph.D.

7.10

# Vorlesung Compilerbau (SoSe2018)

## Teil 7: Shift-Reduce & Präzedenz-Parsing



### Beispiel für $x - 2 * y$

```

1 | S → E
2 | E → E + T
3 |   | E - T
4 |   | T
5 | T → T * F
6 |   | T / F
7 |   | F
8 | F → num
9 |   | id
    
```

| Keller               | Eingabe              | Aktion     |
|----------------------|----------------------|------------|
| \$                   | <b>id</b> - num * id | Shift: id  |
| \$ id                | - num * id           | Reduce: 9  |
| \$ <factor>          | - num * id           | Reduce: 7  |
| \$ <term>            | - num * id           | Reduce: 4  |
| \$ <expr>            | - num * id           | Shift: -   |
| \$ <expr> -          | <b>num</b> * id      | Shift: num |
| \$ <expr> - num      | * id                 | Reduce: 8  |
| \$ <expr> - <factor> | * id                 | Reduce: 7  |

© Prof. J.C. Freytag, Ph.D.

7.11



### Beispiel für $x - 2 * y$

```

1 | S → E
2 | E → E + T
3 |   | E - T
4 |   | T
5 | T → T * F
6 |   | T / F
7 |   | F
8 | F → num
9 |   | id
    
```

| Keller               | Eingabe              | Aktion     |
|----------------------|----------------------|------------|
| \$                   | <b>id</b> - num * id | Shift: id  |
| \$ id                | - num * id           | Reduce: 9  |
| \$ <factor>          | - num * id           | Reduce: 7  |
| \$ <term>            | - num * id           | Reduce: 4  |
| \$ <expr>            | - num * id           | Shift: -   |
| \$ <expr> -          | <b>num</b> * id      | Shift: num |
| \$ <expr> - num      | * id                 | Reduce: 8  |
| \$ <expr> - <factor> | * id                 | Reduce: 7  |
| \$ <expr> - <term>   | * id                 | Shift: *   |

© Prof. J.C. Freytag, Ph.D.

7.12

# Vorlesung Compilerbau (SoSe2018)

## Teil 7: Shift-Reduce & Präzedenz-Parsing



### Beispiel für $x - 2 * y$

```

1 S → E
2 E → E + T
3   | E - T
4   | T
5 T → T * F
6   | T / F
7   | F
8 F → num
9   | id
    
```

| Keller               | Eingabe              | Aktion     |
|----------------------|----------------------|------------|
| \$                   | <b>id</b> - num * id | Shift: id  |
| \$ id                | - num * id           | Reduce: 9  |
| \$ <factor>          | - num * id           | Reduce: 7  |
| \$ <term>            | - num * id           | Reduce: 4  |
| \$ <expr>            | - num * id           | Shift: -   |
| \$ <expr> -          | <b>num</b> * id      | Shift: num |
| \$ <expr> - num      | * id                 | Reduce: 8  |
| \$ <expr> - <factor> | * id                 | Reduce: 7  |
| \$ <expr> - <term>   | * id                 | Shift: *   |
| \$ <expr> - <term> * | <b>id</b>            | Shift: id  |



### Beispiel für $x - 2 * y$

```

1 S → E
2 E → E + T
3   | E - T
4   | T
5 T → T * F
6   | T / F
7   | F
8 F → num
9   | id
    
```

| Keller                  | Eingabe              | Aktion     |
|-------------------------|----------------------|------------|
| \$                      | <b>id</b> - num * id | Shift: id  |
| \$ id                   | - num * id           | Reduce: 9  |
| \$ <factor>             | - num * id           | Reduce: 7  |
| \$ <term>               | - num * id           | Reduce: 4  |
| \$ <expr>               | - num * id           | Shift: -   |
| \$ <expr> -             | <b>num</b> * id      | Shift: num |
| \$ <expr> - num         | * id                 | Reduce: 8  |
| \$ <expr> - <factor>    | * id                 | Reduce: 7  |
| \$ <expr> - <term>      | * id                 | Shift: *   |
| \$ <expr> - <term> *    | <b>id</b>            | Shift: id  |
| \$ <expr> - <term> * id |                      | Reduce: 9  |

# Vorlesung Compilerbau (SoSe2018)

## Teil 7: Shift-Reduce & Präzedenz-Parsing



### Beispiel für $x - 2 * y$

```

1 S → E
2 E → E + T
3   | E - T
4   | T
5 T → T * F
6   | T / F
7   | F
8 F → num
9   | id
    
```

| Keller                        | Eingabe              | Aktion     |
|-------------------------------|----------------------|------------|
| \$                            | <b>id</b> - num * id | Shift: id  |
| \$ id                         | - num * id           | Reduce: 9  |
| \$ <factor>                   | - num * id           | Reduce: 7  |
| \$ <term>                     | - num * id           | Reduce: 4  |
| \$ <expr>                     | - num * id           | Shift: -   |
| \$ <expr> -                   | <b>num</b> * id      | Shift: num |
| \$ <expr> - num               | * id                 | Reduce: 8  |
| \$ <expr> - <factor>          | * id                 | Reduce: 7  |
| \$ <expr> - <term>            | * id                 | Shift: *   |
| \$ <expr> - <term> *          | <b>id</b>            | Shift: id  |
| \$ <expr> - <term> * id       |                      | Reduce: 9  |
| \$ <expr> - <term> * <factor> |                      | Reduce: 5  |

© Prof. J.C. Freytag, Ph.D.

7.15



### Beispiel für $x - 2 * y$

```

1 S → E
2 E → E + T
3   | E - T
4   | T
5 T → T * F
6   | T / F
7   | F
8 F → num
9   | id
    
```

| Keller                        | Eingabe              | Aktion     |
|-------------------------------|----------------------|------------|
| \$                            | <b>id</b> - num * id | Shift: id  |
| \$ id                         | - num * id           | Reduce: 9  |
| \$ <factor>                   | - num * id           | Reduce: 7  |
| \$ <term>                     | - num * id           | Reduce: 4  |
| \$ <expr>                     | - num * id           | Shift: -   |
| \$ <expr> -                   | <b>num</b> * id      | Shift: num |
| \$ <expr> - num               | * id                 | Reduce: 8  |
| \$ <expr> - <factor>          | * id                 | Reduce: 7  |
| \$ <expr> - <term>            | * id                 | Shift: *   |
| \$ <expr> - <term> *          | <b>id</b>            | Shift: id  |
| \$ <expr> - <term> * id       |                      | Reduce: 9  |
| \$ <expr> - <term> * <factor> |                      | Reduce: 5  |
| \$ <expr> - <term>            |                      | Reduce: 3  |

© Prof. J.C. Freytag, Ph.D.

7.16



# Vorlesung Compilerbau (SoSe2018)

## Teil 7: Shift-Reduce & Präzedenz-Parsing

### Beispiel für $x - 2 * y$

```

1 S → E
2 E → E + T
3   | E - T
4   | T
5 T → T * F
6   | T / F
7   | F
8 F → num
9   | id
  
```

| Keller                        | Eingabe              | Aktion     |
|-------------------------------|----------------------|------------|
| \$                            | <b>id</b> - num * id | Shift: id  |
| \$ id                         | - num * id           | Reduce: 9  |
| \$ <factor>                   | - num * id           | Reduce: 7  |
| \$ <term>                     | - num * id           | Reduce: 4  |
| \$ <expr>                     | - num * id           | Shift: -   |
| \$ <expr> -                   | <b>num</b> * id      | Shift: num |
| \$ <expr> - num               | * id                 | Reduce: 8  |
| \$ <expr> - <factor>          | * id                 | Reduce: 7  |
| \$ <expr> - <term>            | * id                 | Shift: *   |
| \$ <expr> - <term> *          | <b>id</b>            | Shift: id  |
| \$ <expr> - <term> * id       |                      | Reduce: 9  |
| \$ <expr> - <term> * <factor> |                      | Reduce: 5  |
| \$ <expr> - <term>            |                      | Reduce: 3  |
| \$ <expr>                     |                      | Reduce: 1  |

© Prof. J.C. Freytag, Ph.D.

7.17

### Beispiel für $x - 2 * y$

```

1 S → E
2 E → E + T
3   | E - T
4   | T
5 T → T * F
6   | T / F
7   | F
8 F → num
9   | id
  
```

| Keller                        | Eingabe              | Aktion     |
|-------------------------------|----------------------|------------|
| \$                            | <b>id</b> - num * id | Shift: id  |
| \$ id                         | - num * id           | Reduce: 9  |
| \$ <factor>                   | - num * id           | Reduce: 7  |
| \$ <term>                     | - num * id           | Reduce: 4  |
| \$ <expr>                     | - num * id           | Shift: -   |
| \$ <expr> -                   | <b>num</b> * id      | Shift: num |
| \$ <expr> - num               | * id                 | Reduce: 8  |
| \$ <expr> - <factor>          | * id                 | Reduce: 7  |
| \$ <expr> - <term>            | * id                 | Shift: *   |
| \$ <expr> - <term> *          | <b>id</b>            | Shift: id  |
| \$ <expr> - <term> * id       |                      | Reduce: 9  |
| \$ <expr> - <term> * <factor> |                      | Reduce: 5  |
| \$ <expr> - <term>            |                      | Reduce: 3  |
| \$ <expr>                     |                      | Reduce: 1  |
| \$ <goal>                     |                      | Accept     |

© Prof. J.C. Freytag, Ph.D.

7.18

# Vorlesung Compilerbau (SoSe2018)

## Teil 7: Shift-Reduce & Präzedenz-Parsing



### Shift-Reduce-Parsing

Operationen eines SR-Parsers (präziser):

- **Shift**  
schiebe nächstes Eingabesymbol auf den Keller
- **Reduce**  
[rechtes Ende eines Handle liegt oben auf dem Keller]
  - finde das linke Ende des Handle im Keller;
  - entferne Handle (rechte Seite einer Regel) aus dem Keller und
  - schiebe das entsprechende Nicht-Terminalsymbol (linke Seite der Regel) auf den Keller
- **Accept**  
beende das Erkennen und signalisiere Erfolg
- **Error**  
Rufe eine Fehlererholungsroutine auf

noch offenes **Problem**: Erkennen einer Handle

© Prof. J.C. Freytag, Ph.D.

7.19



### Konflikte bei einer Shift-Reduce (SR)-Syntaxanalyse

- es existieren kontextfreie Grammatiken, für die shift-Reduce (SR)-Analysen nicht anwendbar sind  
sie gehören nicht zur **LR(k)-Klasse** (Nicht-LR-Grammatik)
- ein SR-Parser kann bei solchen Grammatiken eine Konfiguration erreichen, bei der er
  - nicht entscheiden kann, ob zu schieben oder zu reduzieren ist (**Shift-Reduce-Konflikt**) oder
  - nicht entscheiden kann, welche der verschiedenen Reduktionen anzuwenden ist (**Reduce-Reduce-Konflikt**)
  - obwohl der Keller komplett und das Eingabesymbol bekannt sind

© Prof. J.C. Freytag, Ph.D.

7.20

# Vorlesung Compilerbau (SoSe2018)

## Teil 7: Shift-Reduce & Präzedenz-Parsing

### Nicht-LR-Grammatiken

**Satz:** keine **mehrdeutige** Grammatik kann für irgendein  $k$  eine  $LR(k)$ -Grammatik sein

■ **Beispiel:**

```
stmt → if expr then stmt
      | if expr then stmt else stmt
      | other
```

■ **Konfiguration**

|                       |             |
|-----------------------|-------------|
| Stack                 | Eingabe     |
| ... if expr then stmt | else ... \$ |

**Bemerkung:** können nicht entscheiden, ob "if expr then stmt" ein Handle ist, unabhängig vom weiteren Stack-Inhalt/weitere Eingabe  
 Eingabe-abhängig: → reduce auf **stmt** oder  
 : → shift von **else**

**Lösungsversuch:** Schieben vor Reduzieren (Mehrdeutigkeit bleibt)

© Prof. J.C. Frey

### Nicht-LR-Grammatiken (2)

Grammatik mit Nicht- $LR(k)$ -Eigenschaft: Handle liegt zwar vor, aber Stack-Inhalt und Eingabesymbol lassen keine Auswahl einer Regel zu

**Beispiel:**

- (1)  $stmt \rightarrow id (param\_list)$
- (2)  $stmt \rightarrow expr = expr$
- (3)  $param\_list \rightarrow param\_list, parameter$
- (4)  $param\_list \rightarrow parameter$
- (5)  $parameter \rightarrow id$
- (6)  $expr \rightarrow id (expr\_list)$
- (7)  $expr \rightarrow id$
- (8)  $expr\_list \rightarrow expr\_list, expr$
- (9)  $expr\_list \rightarrow expr$

A(I,J) → Scanner: id (id, id)

A kann Prozedur  
oder Array sein

**Lösung:** bessere Scanner befragen Symboltabelle und liefern procid für Prozedurbezeichner

**Konfiguration**

|             |           |                                   |
|-------------|-----------|-----------------------------------|
| Stack       | Eingabe   | Konflikt bei Regelanwendung       |
| ... id ( id | , id) ... | (5) bei Prozedur<br>(7) bei Array |

© Prof.

## Operator-Grammatiken

- Klasse kontextfreier Grammatiken, die sich *leicht* mit SR-Verfahren parsen lassen
- Besonderheit der sogenannten Operator-Grammatik
  - keine rechte Seite einer Produktion ist  $\epsilon$  oder hat zwei benachbarte Nicht-Terminals

### keine Operator-Grammatik:

$E \rightarrow EAE \mid (E) \mid -E \mid \text{id}$   
 $A \rightarrow + \mid - \mid * \mid / \mid ^$

### Umwandlung ergibt Operator-Grammatik:

$E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid E ^ E \mid$   
 $(E) \mid -E \mid \text{id}$

Grammatik ist mehrdeutig

© Prof. J.C. Freytag, Ph.D.

7.23

## Operator-Präzedenz-Syntaxanalyse

### Vorteil

- leicht zu implementieren

### Nachteil als allgemeine Syntaxanalysetechnik

- Behandlung von Operatoren mit unterschiedlichen Prioritäten erweist sich als schwierig: z.B.: Minus (einstellig & zweistellig)
- deshalb nur für kleine Klasse von Grammatiken anwendbar
- Compiler benutzen dennoch diese Technik für die Analyse von Ausdrücken

© Prof. J.C. Freytag, Ph.D.

7.24

# Vorlesung Compilerbau (SoSe2018)

## Teil 7: Shift-Reduce & Präzedenz-Parsing



### Prioritätsrelationen zwischen Terminalen

Operator-Präzedenz-Verfahren bestimmt Terminal-Relationen zur Handle-Auswahl  
(nicht zu verwechseln mit:  $<$ ,  $=$ ,  $>$ )

| Relation | Semantik                         |
|----------|----------------------------------|
| $a < b$  | a hat niedrigere Priorität als b |
| $a = b$  | a hat gleiche Priorität als b    |
| $a > b$  | a hat höhere Priorität als b     |

**Bem.:** es gibt unterschiedliche Verfahren zur Festlegung der Beziehungen zwischen Terminal-Paaren

© Prof. J.C. Freytag, Ph.D.

7.25



### Verfahren zur Festlegung der Relation von Terminalpaaren

#### ■ intuitives Verfahren

(traditionelle Bezeichnung von Assoziativitäten und Prioritäten von Operatoren)

■ z.B.:  $+$   $<$   $*$  und  $*$   $>$   $+$

Vorgehensweise kann Mehrdeutigkeiten in Grammatiken auflösen und erlaubt den Einsatz von Operator-Präzedenz-Verfahren trotz Problem mit einstelligem Operator Minus

#### ■ alternatives Verfahren

zuerst konstruiert man eindeutige Grammatik (korrekte Grammatik, die Assoziativität und Priorität der Operatoren im Parse-Baum widerspiegelt)

© Prof. J.C. Freytag, Ph.D.

7.26

# Vorlesung Compilerbau (SoSe2018)

## Teil 7: Shift-Reduce & Präzedenz-Parsing



### Anwendung der Operator-Prioritätsrelationen

**Ziel:** Begrenzung des Handle in einer rechtsseitigen Ableitung

- <' (linke Markierung)
- =' (innere Markierung)
- >' (rechte Markierung)
  
- \$ für String-Start-Ende-Markierung  
 $\$ <' b$  und  $b >' \$$  für alle Terminale  $b$

|    | id | +  | *  | \$ |
|----|----|----|----|----|
| id |    | >' | >' | >' |
| +  | <' | >' | <' | >' |
| *  | <' | >' | >' | >' |
| \$ | <' | <' | <' |    |

© Prof. J.C. Freytag, Ph.D.

7.27



### Worterweiterung mit Prioritätsrelationen

**Operator-Grammatik:**

$E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid E \wedge E \mid$   
 $(E) \mid -E \mid id$

**Eingabe-Wort**

id + id \* id

|    | id | +  | *  | \$ |
|----|----|----|----|----|
| id |    | >' | >' | >' |
| +  | <' | >' | <' | >' |
| *  | <' | >' | >' | >' |
| \$ | <' | <' | <' |    |

**Eingabe-Wort mit Prioritätsrelationen**

$\$ <' id' > + <' id >' * <' id >' \$$

© Prof. J.C. Freytag, Ph.D.

7.28

# Vorlesung Compilerbau (SoSe2018)

## Teil 7: Shift-Reduce & Präzedenz-Parsing

### Algorithmus zur Handle-Bestimmung

- analysiere String lexikalisch von links nach rechts bis das erste **>** auftritt
- analysiere rückwärts (nach links) über alle **=** Relationen hinweg, bis ein **<** gefunden wird
- der Handle enthält alles rechts von diesem **<** bis links vom ersten **>**  
alle zwischendurch vorkommenden **Nichtterminale** werden einbezogen !!!

**erstes Handle**

$\$ < ' id > ' + < ' id > ' * < ' id > ' \$$

© Prof. J.C. Freytag, Ph.D.

7.29

### Algorithmus zur Handle-Bestimmung (2)

**erstes Handle**

$\$ < ' id > ' + < ' id > ' * < ' id > ' \$$

Reduce

$E + id * id$

Weglassen  
der Nichtterminale

$+ id * id$

**zweites Handle**

$\$ < ' + < ' id > ' * < ' id > ' \$$

Reduce

...

Weglassen  
der Nichtterminale

...

**drittes Handle**

$\$ < ' + < ' * < ' id > ' \$$

Reduce

...

Weglassen  
der Nichtterminale

$+ *$

**viertes Handle**

$\$ < ' + < ' * > ' \$$

**Fünftes Handle**

$\$ < ' + > ' \$$

© Prof. J.C. Freytag, Ph.D.

7.30

# Vorlesung Compilerbau (SoSe2018)

## Teil 7: Shift-Reduce & Präzedenz-Parsing

### Algorithmus zur Operator-Präzedenz-Syntaxanalyse

```
1. token := erstes Symbol von w$;
2. repeat forever
3.   if (tos() = $ and token = $) then return /* success */
4.   else begin
5.     if (tos() < ' token or tos() = ' token)
6.       then begin
7.         /* shift */
8.         push (token);
9.         token := nexttoken();
10.        end
11.      else
12.        if tos() > ' token
13.          then /* reduce */
14.            repeat
15.              last_token := pop();
16.            until tos() < ' last_token
17.          else FEHLER()
18.        end
19.      end
20.    end
21.  end
22.  end
```

tos() == Rückgabe „top-of-stack“-Symbols

© Prof. J.C. Freytag, Ph.D. 7.31

### Heuristik zur Bestimmung von Prioritätsrelationen

1. hat Operator **op1** höhere Priorität als Operator **op2**:  
 $op1 > op2$  und  $op2 < op1$
2. haben Operatoren **op1** und **op2** gleiche Priorität  
(sie könnten u.U. auch die gleichen sein):  
 $op1 > op2$  und  $op2 > op1$  bei Linksassoziativität  
 $op1 < op2$  und  $op2 < op1$  bei Rechtsassoziativität

#### Beispiel:

- +, - (linksassoziativ):  $+ > +$ ,  $+ > -$ ,  $- > -$ ,  $- > +$
  - ^ (rechtsassoziativ):  $\wedge < \wedge$
- somit garantiert, dass in
- $E - E + E$  das Handle  $E - E$  ausgewählt wird
  - $E \wedge E \wedge E$  das letzte  $E \wedge E$  als Handle ausgewählt wird



# Vorlesung Compilerbau (SoSe2018)

## Teil 7: Shift-Reduce & Präzedenz-Parsing

### Heuristik zur Bestimmung von Prioritätsrelationen (2)

#### 3. zusätzlich für jeden Operator $op$

- $op < ' id$  und  $id > ' op$
- $op < ' ($  und  $( < ' op$
- $) > ' op$  und  $op > ' )$
- $op > ' \$$  und  $\$ < ' op$

somit garantiert, dass in

- $id$  und  $(E)$  auf  $E$  reduziert werden
- und Handlesuche auf  $\$... \$$ -Teilzeichenkette beschränkt bleibt

© Prof. J.C. Freytag, Ph.D.

7.33

### Anwendung der Heuristik

#### Operator-Grammatik:

$E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid E \wedge E \mid (E) \mid -E \mid id$

|    | +  | -  | *  | /  | ^  | id | (  | )  | \$ |
|----|----|----|----|----|----|----|----|----|----|
| +  | >' | >' | <' | <' | <' | <' | <' | >' | >' |
| -  | >' | >' | <' | <' | <' | <' | <' | >' | >' |
| *  | >' | >' | >' | >' | <' | <' | <' | >' | >' |
| /  | >' | >' | >' | >' | <' | <' | <' | >' | >' |
| ^  | >' | >' | >' | >' | <' | <' | <' | >' | >' |
| id | >' | >' | >' | >' | >' |    |    | >' | >' |
| (  | <' | <' | <' | <' | <' | <' | <' | =' |    |
| )  | >' | >' | >' | >' | >' |    |    | >' | >' |
| \$ | <' | <' | <' | <' | <' | <' | <' |    |    |

#### Annahme

1.  $\wedge$  höchste Prior, rechtsass.
2.  $*$ ,  $/$  mittlere Prior, linksass.
3.  $+$ ,  $-$  niedrigste Prior, linksass.
4. Leerzeichen= Fehlereinträge

#### Problem

1. einstelliger Minus-Operator

© Prof. J.C.

7.34

# Vorlesung Compilerbau (SoSe2018)

## Teil 7: Shift-Reduce & Präzedenz-Parsing



### Behandlung einstelliger Operatoren

- ist Operator nur einstellig (z.B.: logische Negation)  
Behandlung nach bisherigem Schema
  - $op < \neg$  , für jeden Operator  $op$  (unabhängig ob einstellig oder zweistellig):
  - $\neg > op$  , falls  $\neg$  höhere Priorität als  $op$
  - $\neg < op$  , falls  $\neg$  kleinere Priorität als  $op$
- analoge Regel für einstellige Postfix-Operatoren
- **Achtung:** Tabelle versagt aber bei  $id^*id$   
Minus als einstelliger Präfix- und zweistelliger Infix-Operator
- **Lösungsvorschlag:** Scanner sollte *unterschiedliche* Zeichen liefern
  - Scanner kann zwar nicht vorausschauen, kann sich aber das vorherige Zeichen merken
  - **Beispiel Fortran:** Minus ist einstellig, falls vorheriges Symbol: Operator, linke Klammer, Komma oder Zuweisung

© Prof. J.C. Freytag, Ph.D.

7.35



### Fehlerbehandlung bei der Operator-Präzedenz-Syntaxanalyse

Stellen, wo Fehler entdeckt werden können:

- a) zwischen dem Terminal-Symbol  $tos$  und der aktuellen Eingabe  $token$  gilt keine Prioritätsregel
- b) obwohl ein Handle gefunden wurde, gibt es keine Produktion (damit kein Handle-Matching)  
→ notwendige Modifikation des Programms zur Fehlererkennung und -diagnose

© Prof. J.C. Freytag, Ph.D.

7.36

# Vorlesung Compilerbau (SoSe2018)

## Teil 7: Shift-Reduce & Präzedenz-Parsing



Fragen??...

