



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

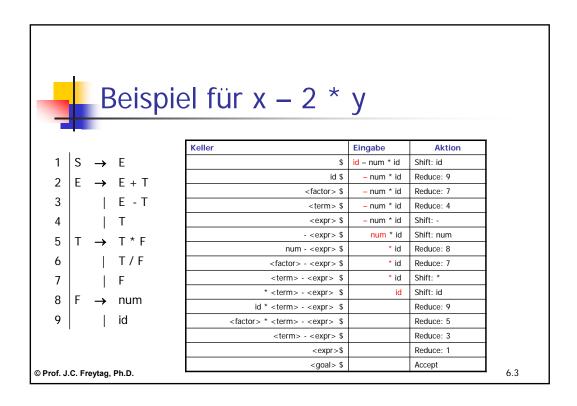
SR-Parser

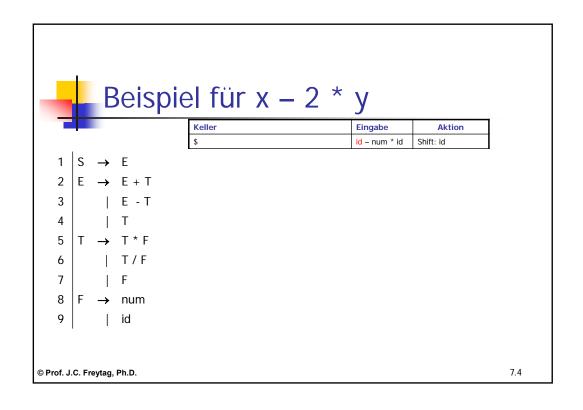
- benutzt Kellerspeicher und Eingabepuffer als Datenstrukturen
- Aktionen:
 - initialisiere den Keller mit dem Zeichen \$
 - wiederhole folgende Schritte, bis das oberste Kellerzeichen das Startsymbol ist und das Eingabesymbol das \$-Zeichen
 - a) schiebe (shift) solange null oder mehrere Eingabesymbole auf den Keller, bis ein Handle β auf dem Keller liegt
 - b) reduziere (reduce) den Handle:

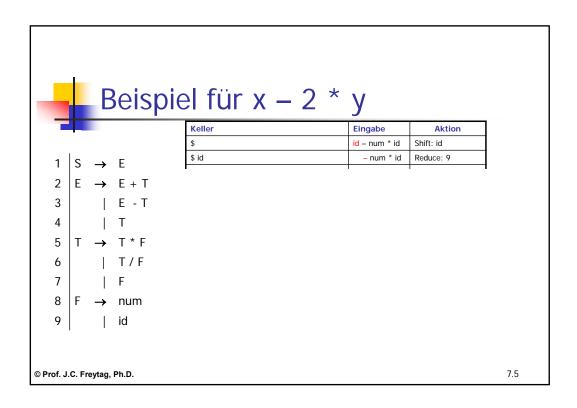
falls ein Handle $A \to \beta$ auf dem Keller liegt, reduziere wie folgt:

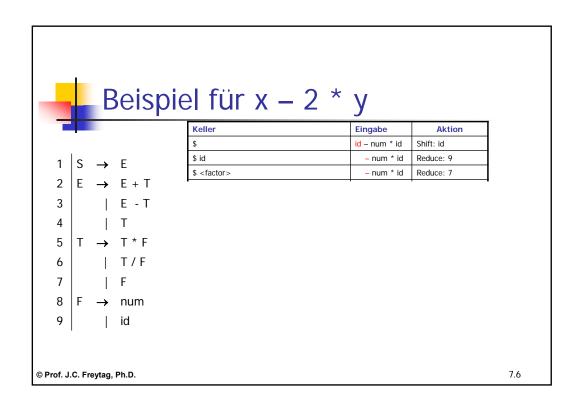
- i) nimm $|\beta|$ Symbole aus dem Keller (pop)
- ii) Ersetze diese durch A auf dem Keller (push)

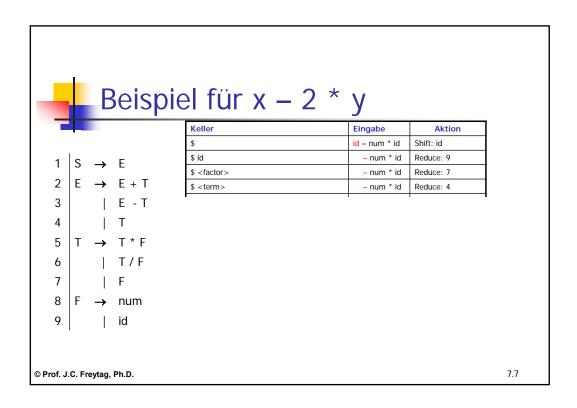
© Prof. J.C. Freytag, Ph.D.

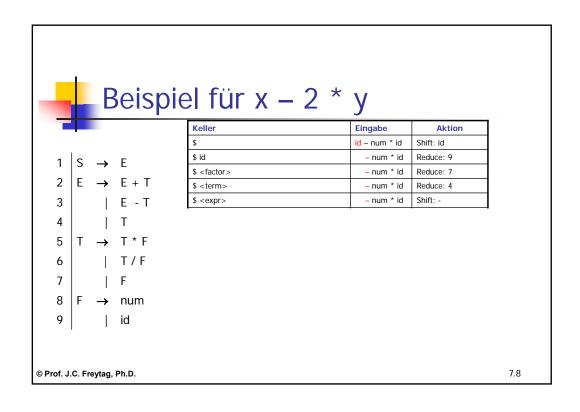


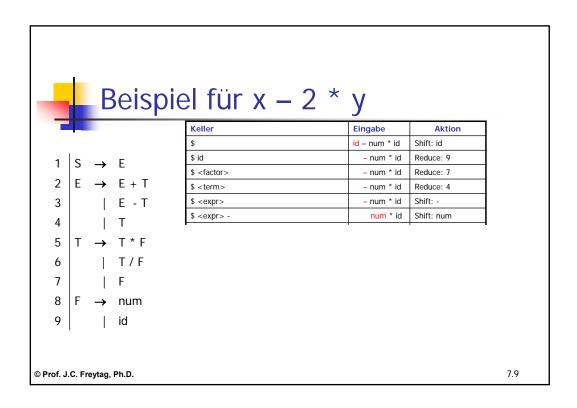


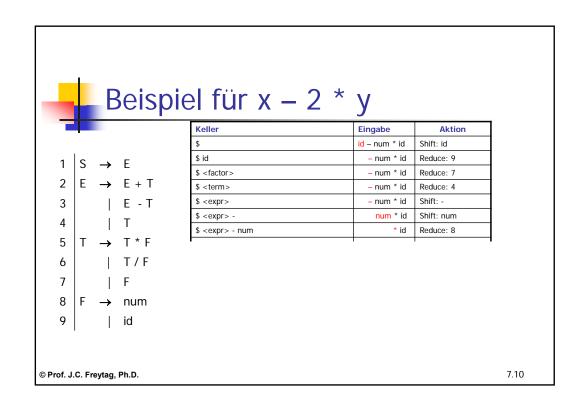


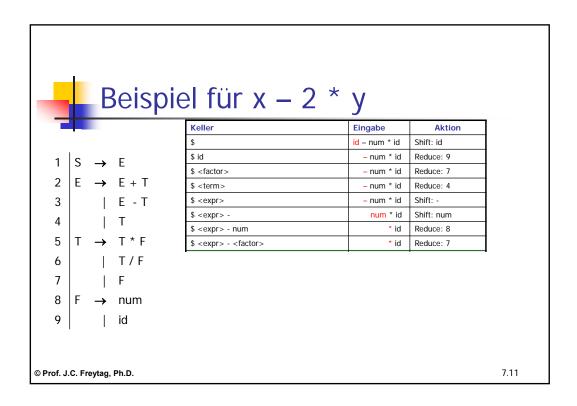


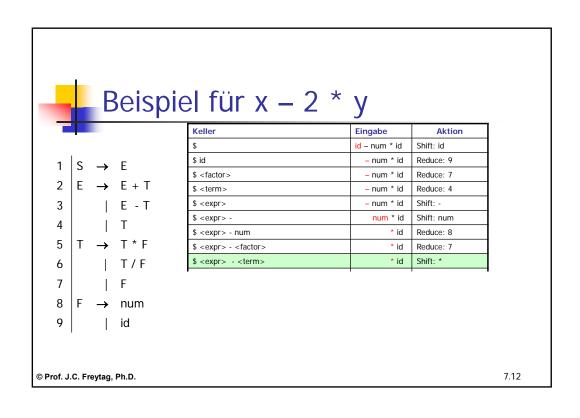


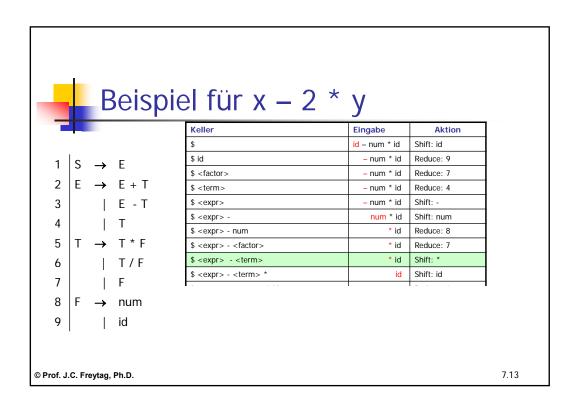


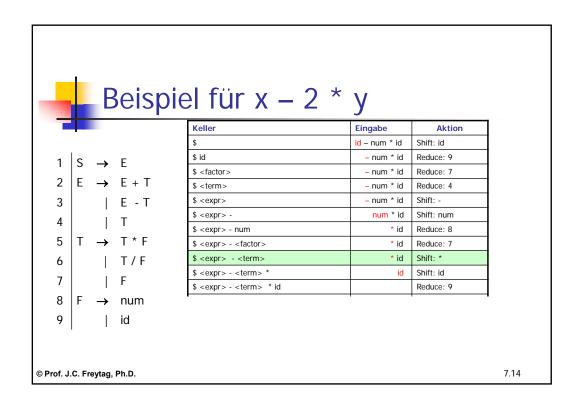


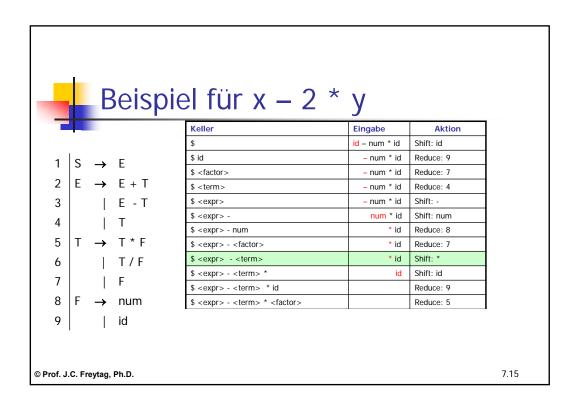














Beispi	el für x – 2 *	у		
	Keller	Eingabe	Aktion	
	\$	id – num * id	Shift: id	
1 S → E	\$ id	– num * id	Reduce: 9	
	\$ <factor></factor>	– num * id	Reduce: 7	
2 E → E+T	\$ <term></term>	– num * id	Reduce: 4	
3 E-T	\$ <expr></expr>	– num * id	Shift: -	
4 T	\$ <expr> -</expr>	num * id	Shift: num	
. ' .	\$ <expr> - num</expr>	* id	Reduce: 8	
5 T → T * F	\$ <expr> - <factor></factor></expr>	* id	Reduce: 7	
6 T/F	\$ <expr> - <term></term></expr>	* id	Shift: *	
7 ' 7 F	\$ <expr> - <term> *</term></expr>	id	Shift: id	
.	\$ <expr> - <term> * id</term></expr>		Reduce: 9	
8 $F \rightarrow \text{num}$	\$ <expr> - <term> * <factor></factor></term></expr>		Reduce: 5	
9 id	\$ <expr> - <term></term></expr>		Reduce: 3	
1 1	\$ <expr></expr>		Reduce: 1	
Prof. J.C. Freytag, Ph.D.	•			7.17

Raisn	iel für x – 2	* v		
Delah		<u>y</u>		_
	Keller	Eingabe	Aktion]
	\$	id – num * id	Shift: id	╛
1 S → E	\$ id	– num * id	Reduce: 9	╛
	\$ <factor></factor>	– num * id	Reduce: 7	╛
$2 \mid E \rightarrow E + T$	\$ <term></term>	– num * id	Reduce: 4	_
3 E-T	\$ <expr></expr>	– num * id	Shift: -	_
4 4 T	\$ <expr> -</expr>	num * id	Shift: num	1
' '	\$ <expr> - num</expr>	* id	Reduce: 8	1
5 T → T * F	\$ <expr> - <factor></factor></expr>	* id	Reduce: 7	
5 T/F	\$ <expr> - <term></term></expr>	* id	Shift: *	
7 . 7 F	\$ <expr> - <term> *</term></expr>	id	Shift: id	1
_ _ ' '	\$ <expr> - <term> * id</term></expr>		Reduce: 9	1
B F → num	\$ <expr> - <term> * <factor></factor></term></expr>		Reduce: 5	_
9 id	\$ <expr> - <term></term></expr>		Reduce: 3	4
1 '	\$ <expr></expr>		Reduce: 1	1
	\$ <goal></goal>		Accept	



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.

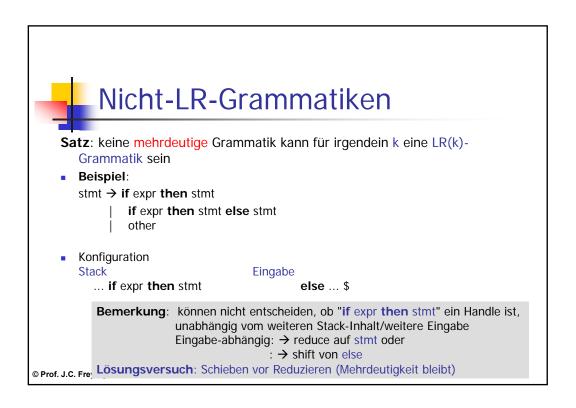
7.19

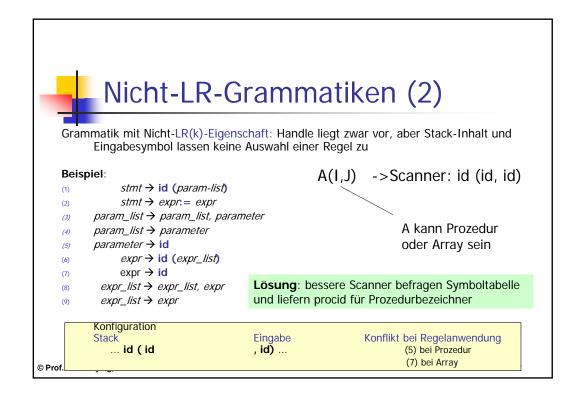


Konflikte bei einer Shift-Reduce (SR)-Syntaxanalyse

- 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.







Operator-Grammatiken

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

keine Operator-Grammatik:

 $E \rightarrow EAE \mid (E) \mid -E \mid id$ $A \rightarrow + \mid - \mid * \mid / \mid \land$ **Umwandlung ergibt Operator-Grammatik:**

 $E \rightarrow E + E \mid E - E \mid E * E \mid E \mid E \mid E \land E \mid$ (E) $\mid -E \mid id$

Grammatik ist mehrdeutig

© Prof. J.C. Freytag, Ph.D.



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



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.



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.



Anwendung der Operator-Prioritätsrelationen

Ziel: Begrenzung des Handle in einer rechtsseitigen Ableitung

- <' (linke Markierung)</p>
- =' (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

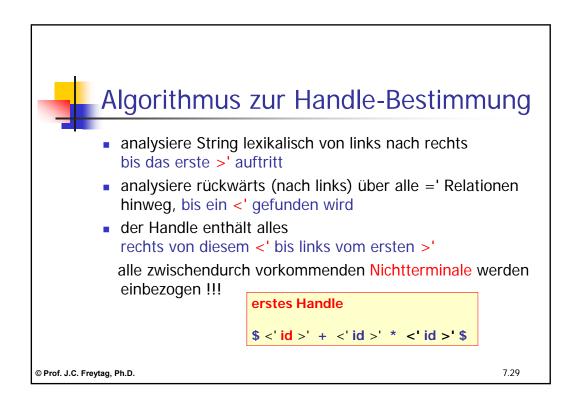
Operator-Grammatik:

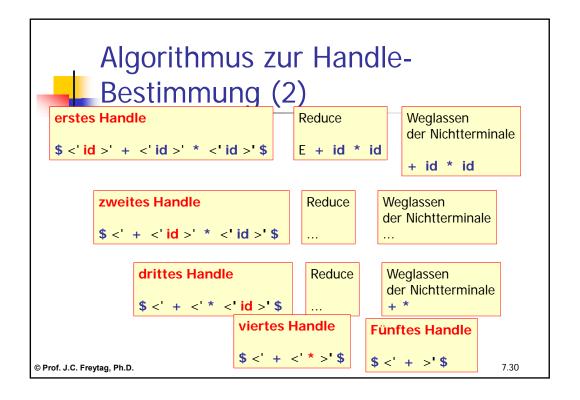
$$E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid E \wedge E \mid$$
(E) | -E | id

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

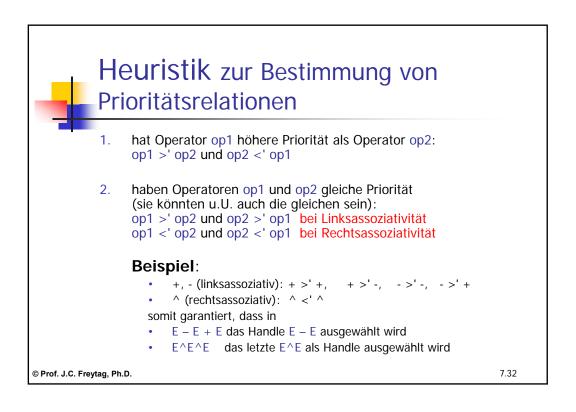
Eingabe-Wort mit Prioritätsrelationen

© Prof. J.C. Freytag, Ph.D.





```
Algorithmus zur Operator-Präzedenz-
            Syntaxanalyse
                  token:= erstes Symbol von w$;
                 repeat forever
                 if (tos() = $ and token= $) then return /* success */
                 else begin
                    if (tos() <
                              token or tos() =' token)
                     then begin
            7.
                           push (token);
            8.
                           token:= nexttoken();
            9.
            10.
                     else
                      if tos() >' token
then /* reduce */
            11.
            12.
                         repeat
            13.
            14.
                             last_token:= pop();
            15.
                          until tos() < last token
                        else FEHLER()
            16.
                                                    tos()== Rückgabe "top-of-stack"-Symbols
                 end
© Prof. J.C. Freytag, Ph.D.
                                                                                     7.31
```





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</pre>

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



© Prof.

Anwendung der Heuristik

Operator-Grammatik:

-		+	_	*	1	٨	id	()	\$
		>'	>'	<'	<'	<'	<'	<'	>'	>'
	+									
	-	>'	>'	<'	<'	<'	<'	<'	>'	>'
	*	>'	>'	>'	>'	<'	<'	<'	>'	>'
	/	>'	>'	^	>'	<'	<'	<'	>'	^
	^	>'	>'	^	>'	<'	<'	<'	>'	>'
	id	>'	>'	>	>'	>'			>'	>'
		<'	<'	'	<'	<'	'	<'	='	
)	>'	>'	^	>'	>'			>'	>'
J.d	\$	<'	<'	<'	<'	<'	<'	<'		

Annahme

- ^ höchste Prior, rechtsass.
- 2. *, / mittlere Prior, linksass.
- . +, niedrigste Prior, linksass.
- 4. Leerzeichen= Fehlereinträge

Problem

1. einstellliger Minus-Operator



Behandlung einstelliger Operatoren

- ist Operator nur einstellig (z.B.: logische Negation) Behandlung nach bisherigem Schema
 - op <' ¬ , für jeden Operator op (unabhängig ob einstellig oder zweistellig):
 - \blacksquare ¬ >' op , falls ¬ höhere Priorität als op
 - ¬ <' op , falls ¬ kleinere Priorität als op
- analoge Regel f
 ür 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.

