Compiler und Programmtransformation

Übung 4

(Parsing: Bottom-Up)

Henning Bordihn

Institut für Informatik und Computational Science

Universität Potsdam

1. Shift-Reduce-Parsing

1) Betrachten Sie die folgende kontextfreie Grammatik (mit Terminalzeichen *a* und *b,* Nichtterminalen *A*, *B* und Startsymbol *A*):

$A \rightarrow AB \mid ab$

$B \rightarrow aba$

Ermitteln Sie einen Shift-Reduce-Parse für die Eingabe *ababa* (so wie auf den Folien 20 bis 29 aus „Parsing: Bottom-Up“), wobei Ihr Parser folgende Strategien zur Konfliktlösung befolgt:

- Shift-Reduce-Konflikt: Reduce hat Vorrang vor Shift, d.h., immer, wenn ein Reduce-Schritt möglich ist, führen Sie diesen aus, sonst führen Sie einen Shift-Schritt aus.
- Reduce-Reduce-Konflikt: Reduce-Schritte werden in der Reihenfolge angewendet, wie sie in der Definition der Grammatik angegeben ist.
- Ist das Ende der Eingabe erreicht und keine Reduktion auf das Startsymbol möglich, so erfolgt ein Backtrack-Schritt bis zum zuletzt ausgeführten Reduce-Schritt, der noch eine andere Aktion erlaubt.

2) Eine kontextfreie Grammatik heißt zyklenfrei, falls sie kein Nichtterminal A enthält, für das $A \Rightarrow^+ A$ gilt. Erklären Sie, weshalb Grammatiken

a) zyklenfrei und

b) frei von ε-Produktionen

sein müssen, damit der obige Shift-Reduce-Parser in jedem Fall erfolgreich angewendet werden kann.

$A \rightarrow AB \mid ab$

$B \rightarrow aba$

Ermitteln Sie einen Shift-Reduce-Parse für die Eingabe *ababa.*

|*ababa*	shift
a | *baba*	shift
ab | *aba*	reduce $A \rightarrow ab$
A | *aba*	shift
Aa | *ba*	shift
Aab | *a*	reduce $A \rightarrow ab$
AA | *a*	shift
AAa |	backtrack
Aab | *a*	shift
Aaba |	reduce $B \rightarrow aba$
AB |	reduce $A \rightarrow AB$
A |	ACCEPT

2) Eine kontextfreie Grammatik heißt zyklenfrei, falls sie kein Nichtterminal A enthält, für das $A \Rightarrow^+ A$ gilt. Erklären Sie, weshalb Grammatiken

a) zyklenfrei und

b) frei von ε-Produktionen

sein müssen, damit der obige Shift-Reduce-Parser in jedem Fall erfolgreich angewendet werden kann.

a) Sonst kann der Parser in eine Endlosschleife geraten (A als Handle wird wegen $A \Rightarrow^+ A$ wieder Topsymbol im Stack.)

b) Sonst kann der Parser durch Reduce-Schritte den Stack unbegrenzt weit aufbauen.

2. LR-Parsing

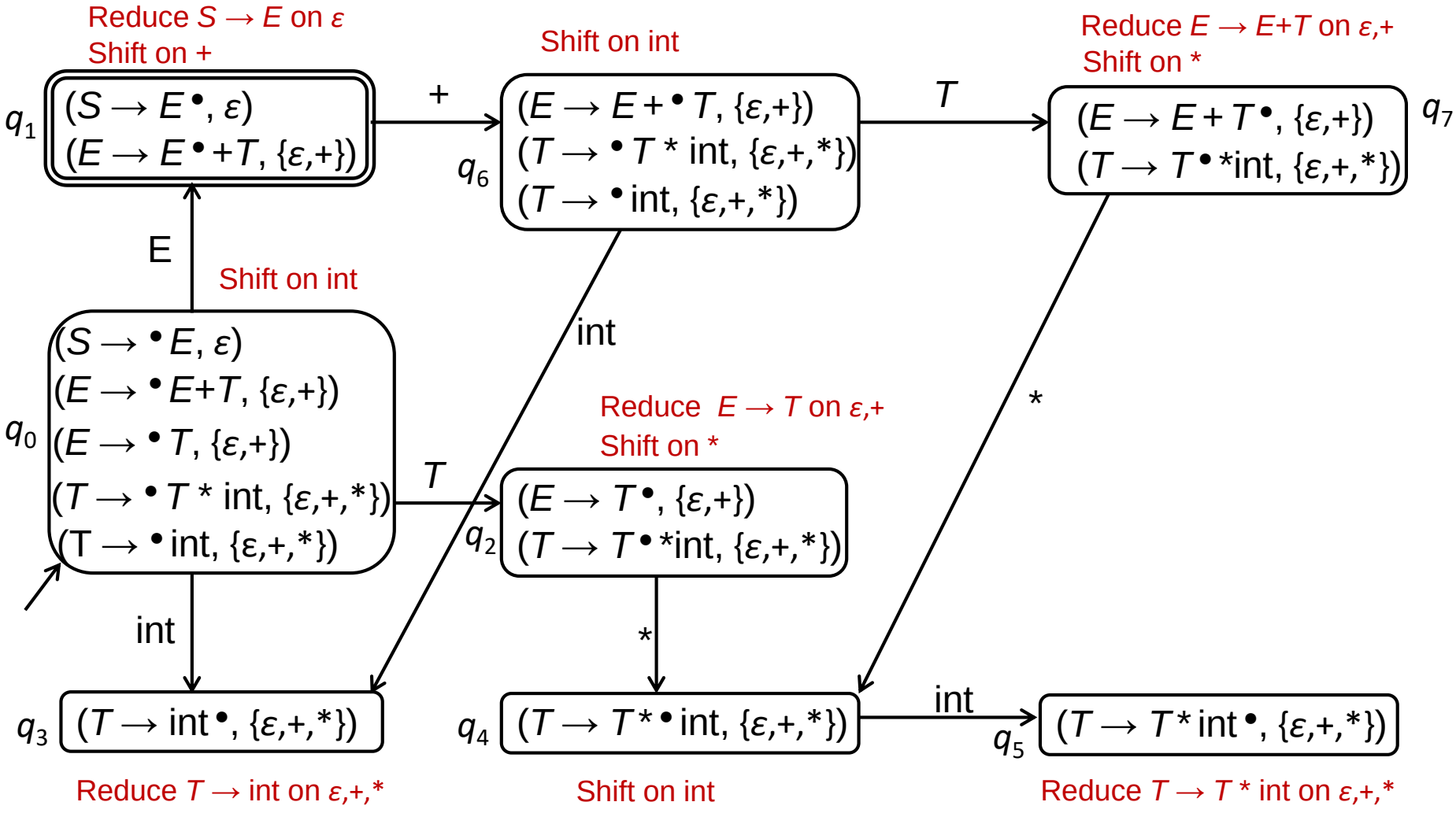
Betrachten Sie erneut den L(1)-Parser aus der Vorlesung.

1) Wenden Sie ihn zum Parsing der folgenden Eingaben an:
   a) int + int * int
   b) int++

2) Betrachten Sie die folgende eindeutige kontextfreie Grammatik:

   $A \rightarrow aAa \mid bAb \mid \varepsilon$

   Handelt es sich um eine LR(1)-Grammatik?
   Wenn nicht, kann der lookahead geeignet vergrößert werden, so dass eine LR(k)-Grammatik für ein geeignetes k > 1 vorliegt?


Reduce $S \rightarrow E$ on ε
Shift on +
q_1
$(S \rightarrow E \bullet, \varepsilon)$
$(E \rightarrow E \bullet + T, \{\varepsilon,+\})$
+
Shift on int
q_6
$(E \rightarrow E + \bullet T, \{\varepsilon,+\})$
$(T \rightarrow \bullet T * \text{int}, \{\varepsilon,+,*\})$
$(T \rightarrow \bullet \text{int}, \{\varepsilon,+,*\})$
T
Reduce $E \rightarrow E+T$ on ε,+
Shift on *
q_7
$(E \rightarrow E + T \bullet, \{\varepsilon,+\})$
$(T \rightarrow T \bullet * \text{int}, \{\varepsilon,+,*\})$
E
Shift on int
q_0
$(S \rightarrow \bullet E, \varepsilon)$
$(E \rightarrow \bullet E+T, \{\varepsilon,+\})$
$(E \rightarrow \bullet T, \{\varepsilon,+\})$
$(T \rightarrow \bullet T * \text{int}, \{\varepsilon,+,*\})$
$(T \rightarrow \bullet \text{int}, \{\varepsilon,+,*\})$
int
T
Reduce $E \rightarrow T$ on ε,+
Shift on *
q_2
$(E \rightarrow T \bullet, \{\varepsilon,+\})$
$(T \rightarrow T \bullet * \text{int}, \{\varepsilon,+,*\})$
*
int
q_3
$(T \rightarrow \text{int} \bullet, \{\varepsilon,+,*\})$
Reduce $T \rightarrow \text{int}$ on ε,+,*
*
q_4
$(T \rightarrow T * \bullet \text{int}, \{\varepsilon,+,*\})$
Shift on int
int
q_5
$(T \rightarrow T * \text{int} \bullet, \{\varepsilon,+,*\})$
Reduce $T \rightarrow T * \text{int}$ on ε,+,*

Beispiel a) LR(1)-Parse

q_0 | int + int * int	Shift
$q_0\ q_3$ | + int * int	Reduce $T \rightarrow$ int
$q_0\ q_2$ | + int * int	Reduce $E \rightarrow T$
$q_0\ q_1$ | + int * int	Shift
$q_0\ q_1\ q_6$| int * int	Shift
$q_0\ q_1\ q_6\ q_3$ | * int	Reduce $T \rightarrow$ int
$q_0\ q_1\ q_6\ q_7$ | * int	Shift
$q_0\ q_1\ q_6\ q_7\ q_4$ | int	Shift
$q_0\ q_1\ q_6\ q_7\ q_4\ q_5$ |	Reduce $T \rightarrow T$ * int
$q_0\ q_1\ q_6\ q_7$ |	Reduce $E \rightarrow E + T$
$q_0\ q_1$ |	**Accept**

Beispiel b) LR(1)-Parse

q_0 | int + +	Shift
$q_0\ q_3$ | + +	Reduce $T \rightarrow$ int
$q_0\ q_2$ | + +	Reduce $E \rightarrow T$
$q_0\ q_1$ | + +	Shift
$q_0\ q_1\ q_6$| +	**ERROR**

2. LR-Parsing

2) Betrachten Sie die folgende eindeutige kontextfreie Grammatik:

$A \rightarrow aAa \mid bAb \mid \varepsilon$

Handelt es sich um eine LR(1)-Grammatik?

Nein. *Intuitiv:* Zum Beispiel bei der Eingabe a^{2n} müsste ε dann auf A reduziert werden, wenn genau die Hälfte der Eingabesymbole, also a^n in den Pushdown geshiftet wurde. Bei hinreichend großem n sind Topsymbol im Keller (Zustand) und der Lookahead in dieser Situation aber identisch mit einer Situation zuvor, in der shift angewendet werden musste.

Wenn nicht, kann der lookahead geeignet vergrößert werden, so dass eine LR(k)-Grammatik für ein geeignetes $k > 1$ vorliegt?

Nein. Siehe oben.