LR-Parsing

- Prinzip: Entwicklung des Parsebaums bottom-up, von links nach rechts (Abb. 2.52)
- Parserkonfiguration: $(s_0 X_1 s_1 \ldots X_m s_m, t_i t_{i+1} \ldots t_n)$

  1. Stack: enthält abwechselnd Zustand s_k und Grammatiksymbol X_{k+1}
     - Zustand: Positionen in einer Menge von rechten Seiten
     - Grammatiksymbol: Link zu einem erzeugten Syntaxbaum
  2. Resteingabe mit Zeichen t_i

- Handle: rechte Seite einer Produktion, die als nächstes reduziert wird
- Verschiedene Techniken (Unterschiede in Erkennung der Handles, nicht in der Reduktion), in aufsteigender Mächtigkeit geordnet:
  - LR(0): sehr einfach, Entscheidung für Reduktion nur aufgrund des Zustands
  - SLR(1): Reduktionsentscheidung aufgrund von FOLLOW-Mengen
  - LALR(1): Zustandskompression von LR(1), Speicherverbrauch von SLR(1)
  - LR(1): berücksichtigt Kontext zur Entscheidung für Handles

Bottom-Up Konstruktion bei LR

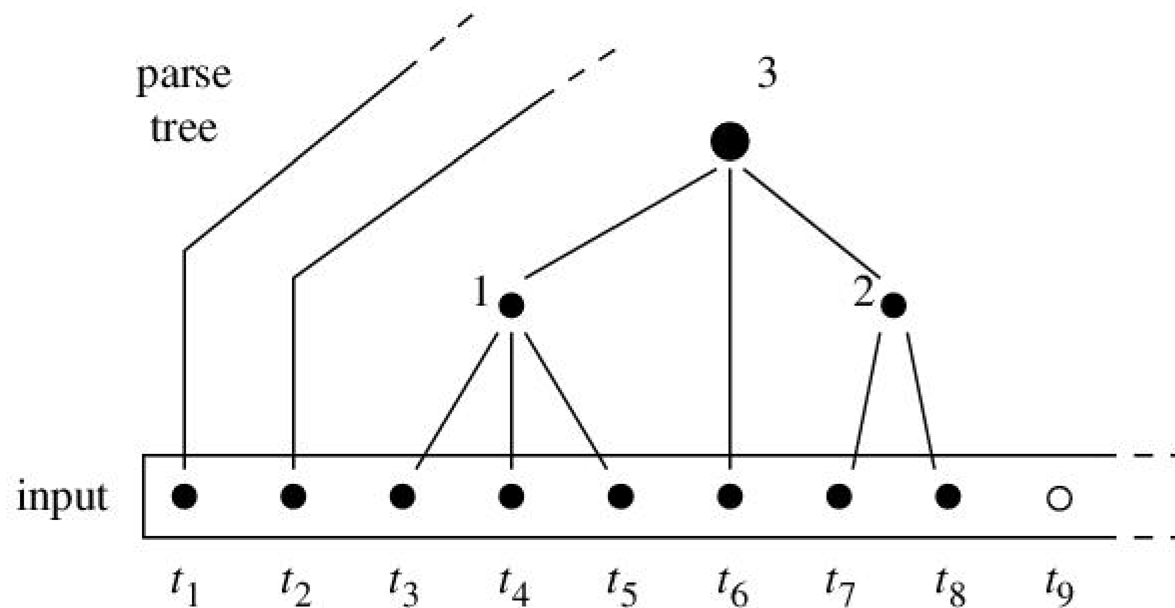

Figure 2.52 A bottom-up parser constructing its first, second, and third nodes.

Beispielgrammatik

```
input → expression EOF
expression → term | expression '+' term
term → IDENTIFIER | '(' expression ')'
```

Figure 2.84 A simple grammar for demonstrating bottom-up parsing.

```
Z → E $
E → T | E + T
T → i | ( E )
```

Figure 2.85 An abbreviated form of the simple grammar for bottom-up parsing.

LR(0)-Itemmengen

Form eines LR-Elements: $\mathbf{N} \rightarrow \alpha \bullet \beta$

α: schon geparst

β: Rest des Handles

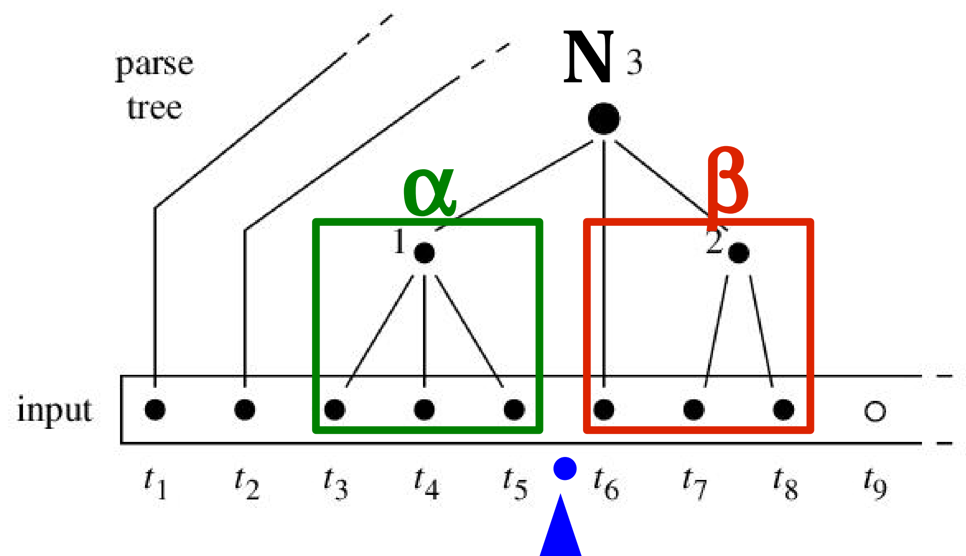

Figure 2.52 A bottom-up parser constructing its first, second, and third nodes.

LR(0)-Itemarten

- *shift-Item*: $\mathbf{N} \rightarrow \alpha \bullet \beta$
  Aktion: schiebe Eingabe und Folgezustand auf den Stack

- *reduce-Item*: $\mathbf{N} \rightarrow \alpha\beta \bullet$
  Aktion: reduziere $\alpha\beta$ auf N (neuer Parsebaumknoten)

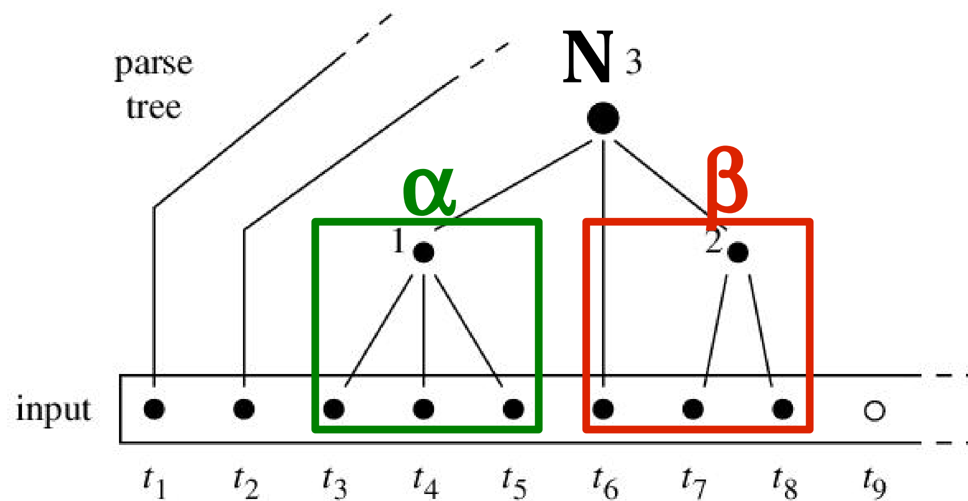

Figure 2.52 A bottom-up parser constructing its first, second, and third nodes.

LR(0)-Itemarten

- *shift-Item*: $\mathbf{N} \rightarrow \alpha \bullet \beta$
  Aktion: schiebe Eingabe und Folgezustand auf den Stack

- *reduce-Item*: $\mathbf{N} \rightarrow \alpha\beta \bullet$
  Aktion: reduziere $\alpha\beta$ auf N (neuer Parsebaumknoten)

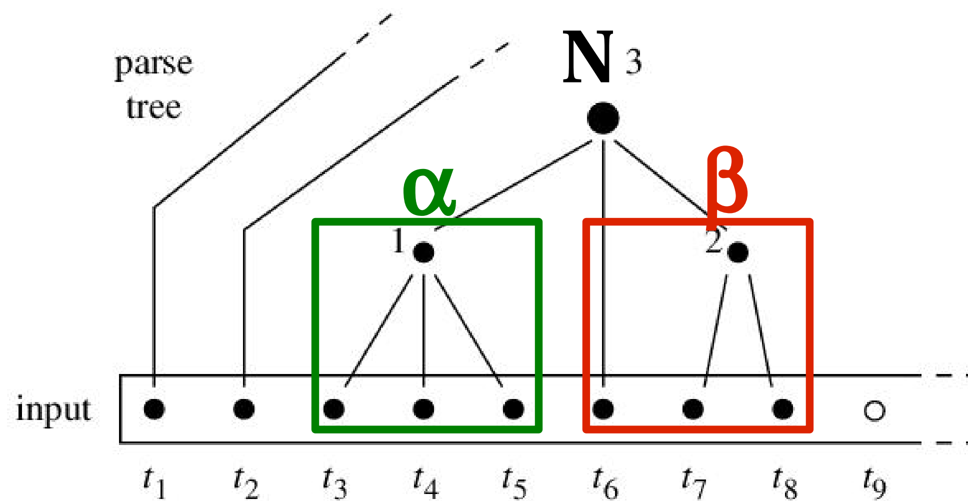

Figure 2.52 A bottom-up parser constructing its first, second, and third nodes.

Berechnung der LR(0)-Itemmengen (a) ε-Hülle

Data definitions:

A set S of LR(0) items.

Initializations:

S is prefilled externally with one or more LR(0) items.

Inference rules:

If S holds an item of the form $P \to \alpha \bullet N\beta$, then for each production rule $N \to \gamma$ in G, S must also contain the item $N \to \bullet\gamma$.

Figure 2.86 ε closure algorithm for LR(0) item sets for a grammar G.

Berechnung der LR(0)-Itemmengen (b) Startmenge

```
FUNCTION Initial item set RETURNING an item set:
    SET New item set TO Empty;

    // Initial contents - obtain from the start symbol:
    FOR EACH production rule S→α for the start symbol S:
        SET New item set TO New item set + item S→•α;

    RETURN ε closure (New item set);
```

Figure 2.87 The routine `Initial item set` for an LR(0) parser.

Berechnung der LR(0)-Itemmengen (c) Zustandsübergang

```
FUNCTION Next item set (Item set, Symbol) RETURNING an item set:
    SET New item set TO Empty;

    // Initial contents - obtain from token moves:
    FOR EACH item N→α•Sβ IN Item set:
        IF S = Symbol:
            SET New item set TO New item set + item N→αS•β;

    RETURN ε closure (New item set);
```

Figure 2.88 The routine `Next item set()` for an LR(0) parser.

Beispiel

Z	→	E	$			
E	→	T	|	E	+	T
T	→	i	|	(	E	)

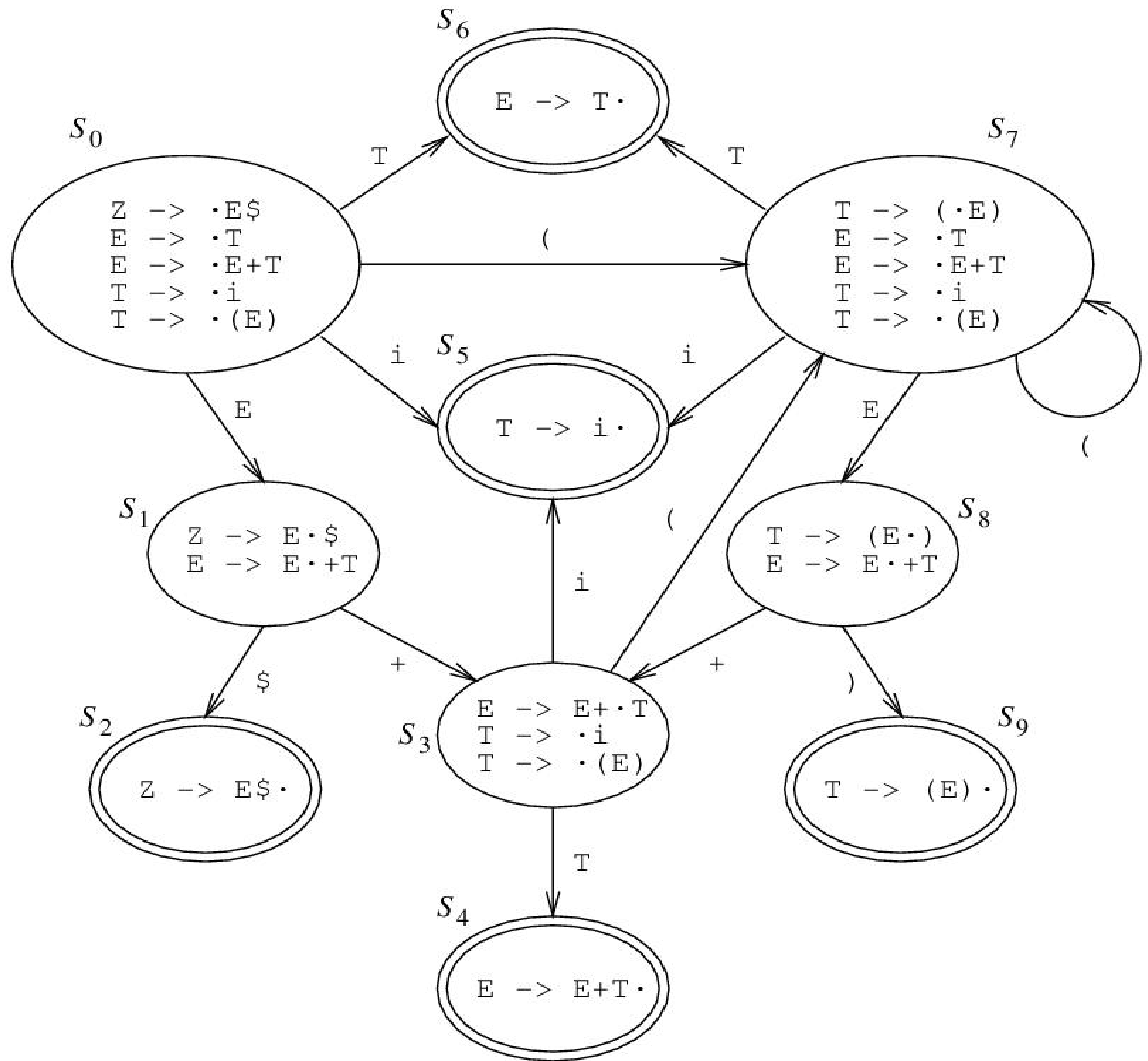

Figure 2.89 Transition diagram for the LR(0) automaton for the grammar of Figure 2.85.

LR(0)-Tabellenstruktur

- Automat: zwei Arten von Items in den Zuständen (Abb. 2.89)
  - Shift: Übernahme des Eingabetokens auf den Stack
  - Reduce: Ersetzung des Handleabschnitts auf dem Stack durch das Nichtterminal auf der linken Seite + semantische Aktion
- Tabelle: zweigeteilt (Abb. 2.90)
  - GOTO: spezifiziert Folgezustand (wie im Automaten)
  - ACTION: spezifiziert, of Shift oder Reduce, bei Reduce auch die Regel
- Operationen (Abb. 2.91, 2.92)
  - Shift
    - push Eingabetoken
    - push Folgezustand aus GOTO[alter Zustand, Eingabetoken]
  - Reduce
    - pop Paare(Zustand,Grammatiksymbol) des Handles (rechte Seite)
    - push Nichtterminal *N* aus ACTION[alter Zustand]
    - push Folgezustand in GOTO[Zustand unter *N*, *N*]

LR(0)-Tabellenstruktur

	|←		GOTO table symbol				→|	ACTION table
state	i	+	(	)	$	E	T	
0	5		7			1	6	shift
1		3			2			shift
2								Z→E$
3	5		7				4	shift
4								E→E+T
5								T→i
6								E→T
7	5		7			8	6	shift
8		3		9				shift
9								T→(E)

Figure 2.90 GOTO and ACTION tables for the LR(0) automaton for the grammar of Figure 2.85.

Beispiel: Parsen von "`i+i`"

Stack	Input	Action
S_0	i + i $	shift
S_0 i S_5	+ i $	reduce by T→i
S_0 T S_6	+ i $	reduce by E→T
S_0 E S_1	+ i $	shift
S_0 E S_1 + S_3	i $	shift
S_0 E S_1 + S_3 i S_5	$	reduce by T→i
S_0 E S_1 + S_3 T S_4	$	reduce by E→E+T
S_0 E S_1	$	shift
S_0 E S_1 $ S_2		reduce by Z→E$
S_0 Z		stop

Figure 2.91 LR(0) parsing of the input `i+i`.

LR(0) Parsing-Algorithmus

```
IMPORT Input token [1..];          // from the lexical analyzer

SET Input token index TO 1;
SET Reduction stack TO Empty stack;
PUSH Start state ON Reduction stack;

WHILE Reduction stack /= {Start state, Start symbol, End state}
    SET State TO Top of Reduction stack;
    SET Action TO Action table [State];
    IF Action = "shift":
        // Do a shift move:
        SET Shifted token TO Input token [Input token index];
        SET Input token index TO Input token index + 1; // shifted
        PUSH Shifted token ON Reduction stack;
        SET New state TO Goto table [State, Shifted token .class];
        PUSH New state ON Reduction stack;      // can be Empty
    ELSE IF Action = ("reduce", N→α):
        // Do a reduction move:
        Pop the symbols of α from Reduction stack;
        SET State TO Top of Reduction stack;    // update State
        PUSH N ON Reduction stack;
        SET New state TO Goto table [State, N];
        PUSH New state ON Reduction stack;      // cannot be Empty
    ELSE Action = Empty:
        ERROR "Error at token ", Input token [Input token index];
```

Figure 2.92 LR(0) parsing with a push-down automaton.

LR(0)-Grammatiken

- Anforderung: deterministischer Parser
  ⇒ Parsetabelle ohne Mehrfacheinträge
- Problem: Konstruktion der ACTION-Tabelle
- Voraussetzungen (keine Mehrfacheinträge)
  - Zustände mit Folgezuständen dürfen keine Reduce-Items haben
    (Verletzung: Shift/Reduce-Konflikt)
  - jeder Zustand darf höchstens ein Reduce-Item enthalten
    (Verletzung: Reduce/Reduce-Konflikt)
- Gegenbeispiele (Verletzung von LR(0))
  - ε-Produktionen (Shift/Reduce-Konflikt), Bsp. dangling else
  - nicht linksfaktorisierte Grammatiken (Shift/Reduce-Konflikt),
    Bsp.: Summenausdrücke mit Arrayindizes
  - Regeln mit gleichen rechten Seiten (Reduce/Reduce-Konflikt),
    Bsp.: Ausdruckssprache mit Zuweisung (**`Z`→`V:=E`**, **`V`→`id`**, **`E`→`id`**)
- Verbesserung: Reduktion abhängig von Vorausschau (FOLLOW-Menge)
  ⇒ zweidimensionale ACTION-Tabelle

SLR(1)-Parsetabelle

Z → E $
E → T | E + T
T → i | (E)

state	i	+	(	)	$
	look-ahead token				
0	shift		shift		
1		shift			shift
2					Z→E$
3	shift		shift		
4		E→E+T		E→E+T	E→E+T
5		T→i		T→i	T→i
6		E→T		E→T	E→T
7	shift		shift		
8		shift		shift	
9		T→(E)		T→(E)	T→(E)

Figure 2.93 ACTION table for the SLR(1) automaton for the grammar of Figure 2.85.

SLR(1)-Tabellenstruktur

- Merkmale der ACTION-Tabelle:
  - Spalten = Eingabetokens
  - Einträge:
    - entweder "Shift" (Nachfolgezustand mittels GOTO-Tabelle)
    - oder eine fest Reduktionsregel für jedes Token in der FOLLOW-Menge des entsprechenden Nichtterminals
- Kombination von GOTO- und ACTION-Tabelle
  - funktioniert nur für Vorausschau auf <u>ein</u> Eingabetoken!
  - Spalten = Grammatiksymbole (wie in der GOTO-Tabelle)
  - Einträge:
    - Eintrag der GOTO-Tabelle bekommt Präfix "s"
    - Eintrag der ACTION-Tabelle wird zu "r" mit Regelnummer
  - Struktur eignet sich hervorragend zur Tabellenkompression!
  - gleiche Größe wie die GOTO-Tabelle, aber dichter besetzt
- Vorausschau nur auf Basis der FOLLOW-Menge, nicht kontextabhängig

Beispiel: SLR(1)

T → i

- Grammatik: Summenausdrücke

```
1: Z → E$
2: E → T
3: E → E+T
4: T → i
5: T → (E)
```

- SLR(1)-Tabelle (Abb. 2.94)
- Erweiterung der Grammatik: um indizierte Variablen

```
6: T → i[E]
```

- Erweiterung der Tabelle
  - zwei neue Spalten für die neuen Token [und]
  - neue Einträge in Zeile 5:

state	i	+	(	)	[	]	$
5		T→i		T→i	shift	T→i	T→i

- Beobachtungen:
  - Shift und Reduce in einer Zeile
  -] $\in$ FOLLOW(T) aber [$\notin$ FOLLOW(T)

Beispiel: SLR(1)

r1: Z → E$
r2: E → T
r3: E → E+T
r4: T → i
r5: T → (E)

state	i	+	(	)	$	E	T
	stack symbol/look-ahead token						
0	s5		s7			s1	s6
1		s3			s2		
2					r1		
3	s5		s7				s4
4		r3		r3	r3		
5		r4		r4	r4		
6		r2		r2	r2		
7	s5		s7			s8	s6
8		s3		s9			
9		r5		r5	r5		

Figure 2.94 ACTION/GOTO table for the SLR(1) automaton for the grammar of Figure 2.85.