

Grundlegende Parsingalgorithmen

Grammatiktransformationen, Komplexität

Kurt Eberle

k.eberle@lingenio.de

(Viele Folien, Teile von Folien, Materialien von **Helmut Schmid's**
Parsing-Kurs WS14 Tübingen, u.a.)

28. Februar 2020

Überblick

Grammatik-Transformationen

Chomsky-Normalform

Greibach-Normalform

Transformationen im Earley-und LC-Parser

Vergleich von Parsing-Algorithmen

Komplexität

Komplexität des CYK-Parsing

Komplexität des Earley-Parsers

Parsing-Komplexität Ergebnisse

Überblick

Grammatik-Transformationen

Chomsky-Normalform

Greibach-Normalform

Transformationen im Earley-und LC-Parser

Vergleich von Parsing-Algorithmen

Komplexität

Komplexität des CYK-Parsing

Komplexität des Earley-Parsers

Parsing-Komplexität Ergebnisse

Grammatik-Transformationen

- ▶ Transformation in Chomsky-Normalform (CNF) für CYK
- ▶ Implizite Transformationen im Earley- und LC-Parser
- ▶ Andere Verfahren: Transformation in Greibach-Normalform (GNF)

Überblick

Grammatik-Transformationen

Chomsky-Normalform

Greibach-Normalform

Transformationen im Earley-und LC-Parser

Vergleich von Parsing-Algorithmen

Komplexität

Komplexität des CYK-Parsing

Komplexität des Earley-Parsers

Parsing-Komplexität Ergebnisse

Chomsky-Normalform

Grammatik $G = (V, \Sigma, P, S)$ ist in CNF gdw G hat die Form

- ▶ $A \rightarrow BC$
- ▶ $A \rightarrow a$
- ▶ ($S \rightarrow \epsilon$; in diesem Fall erscheint S nicht auf der rechten Seite einer Regel)

für $A, B, C \in V, a \in \Sigma$.

Transformation in CNF

1 f.a. $a \in \Sigma$:

- führe neue Symbole X_a ein in V und neue Regeln $X_a \rightarrow a$ in G ;
- ersetze alle $a \in \Sigma$ in allen Regeln in P durch X_a ;

Transformation in CNF

- 1 f.a. $a \in \Sigma$:
 - führe neue Symbole X_a ein in V und neue Regeln $X_a \rightarrow a$ in G ;
 - ersetze alle $a \in \Sigma$ in allen Regeln in P durch X_a ;
- 2 in allen Regeln mit mehr als 2 Elementen auf der rechten Seite:
 - ersetze jeweils ein Par von Symbolen AB durch ein neues V -symbol X_{AB} und
 - füge Regeln $X_{AB} \rightarrow AB$ zu P hinzu;jeweils so lange als notwendig;

Transformation in CNF

- 1 f.a. $a \in \Sigma$:
 - führe neue Symbole X_a ein in V und neue Regeln $X_a \rightarrow a$ in G ;
 - ersetze alle $a \in \Sigma$ in allen Regeln in P durch X_a ;
- 2 in allen Regeln mit mehr als 2 Elementen auf der rechten Seite:
 - ersetze jeweils ein Par von Symbolen AB durch ein neues V -symbol X_{AB} und
 - füge Regeln $X_{AB} \rightarrow AB$ zu P hinzu;jeweils so lange als notwendig;
- 3 falls $S \rightarrow \epsilon \in P$:
 - füge neues Symbol S' hinzu und die Regeln $S' \rightarrow \epsilon, S' \rightarrow S$ in P

Transformation in CNF

- 1 f.a. $a \in \Sigma$:
 - führe neue Symbole X_a ein in V und neue Regeln $X_a \rightarrow a$ in G ;
 - ersetze alle $a \in \Sigma$ in allen Regeln in P durch X_a ;
- 2 in allen Regeln mit mehr als 2 Elementen auf der rechten Seite:
 - ersetze jeweils ein Par von Symbolen AB durch ein neues V -symbol X_{AB} und
 - füge Regeln $X_{AB} \rightarrow AB$ zu P hinzu;
 jeweils so lange als notwendig;
- 3 falls $S \rightarrow \epsilon \in P$:
 - füge neues Symbol S' hinzu und die Regeln $S' \rightarrow \epsilon$, $S' \rightarrow S$ in P
- 4 f.a. $A \rightarrow \epsilon \in P$ ($A \neq S'$) und (a) $C \rightarrow AB$, $C \rightarrow BA$ oder (b) $D \rightarrow A \in P$:
 - tilge $A \rightarrow \epsilon$ und füge $C \rightarrow B$ (im Fall a) und $D \rightarrow \epsilon$ (im Fall b) zu P hinzu;
 - tilge $C \rightarrow AB$ und $C \rightarrow BA$ bzw. $D \rightarrow A$ falls $A \rightarrow \epsilon$ die einzige A -Regel in P war
 jeweils so lange als notwendig;

Transformation in CNF

- 1 f.a. $a \in \Sigma$:
 - führe neue Symbole X_a ein in V und neue Regeln $X_a \rightarrow a$ in G ;
 - ersetze alle $a \in \Sigma$ in allen Regeln in P durch X_a ;
- 2 in allen Regeln mit mehr als 2 Elementen auf der rechten Seite:
 - ersetze jeweils ein Par von Symbolen AB durch ein neues V -symbol X_{AB} und
 - füge Regeln $X_{AB} \rightarrow AB$ zu P hinzu;
 jeweils so lange als notwendig;
- 3 falls $S \rightarrow \epsilon \in P$:
 - füge neues Symbol S' hinzu und die Regeln $S' \rightarrow \epsilon$, $S' \rightarrow S$ in P
- 4 f.a. $A \rightarrow \epsilon \in P$ ($A \neq S'$) und (a) $C \rightarrow AB$, $C \rightarrow BA$ oder (b) $D \rightarrow A \in P$:
 - tilge $A \rightarrow \epsilon$ und füge $C \rightarrow B$ (im Fall a) und $D \rightarrow \epsilon$ (im Fall b) zu P hinzu;
 - tilge $C \rightarrow AB$ und $C \rightarrow BA$ bzw. $D \rightarrow A$ falls $A \rightarrow \epsilon$ die einzige A -Regel in P war
 jeweils so lange als notwendig;
- 5 f.a Regeln $A \rightarrow B$ mit $B \in V$ ('Kettenregeln') und alle $B \rightarrow \omega$ gegeben die bisher stattgefundenen Transformationen:
 - tilge $A \rightarrow B$ und füge $A \rightarrow \omega$ hinzu
 jeweils so lange als notwendig;

Überblick

Grammatik-Transformationen

Chomsky-Normalform

Greibach-Normalform

Transformationen im Earley-und LC-Parser

Vergleich von Parsing-Algorithmen

Komplexität

Komplexität des CYK-Parsing

Komplexität des Earley-Parsers

Parsing-Komplexität Ergebnisse

Greibach-Normalform

Grammatik $G = (V, \Sigma, P, S)$ ist in GNF gdw G ist ϵ -frei und die Regeln von G haben die Form:

- ▶ $A \rightarrow a\alpha$, mit $a \in \Sigma$, $\alpha \in V^*$

Jede CF Grammatik G mit $\epsilon \notin L(G)$ kann in GNF überführt werden (z.B. cf. Hopcroft/Ullman p 103)

Überblick

Grammatik-Transformationen

Chomsky-Normalform

Greibach-Normalform

Transformationen im Earley-und LC-Parser

Vergleich von Parsing-Algorithmen

Komplexität

Komplexität des CYK-Parsing

Komplexität des Earley-Parsers

Parsing-Komplexität Ergebnisse

Implizite Transformationen

Beobachtung: Earley-Parser und LC-Parser führen implizit Grammatiktransformationen durch (die Punktregeln sind neue Regeln!):

$A \rightarrow B_1 \dots B_n$ wird ersetzt durch die Regeln

$$\begin{aligned}
 A &\rightarrow \langle A \rightarrow B_1 \dots B_n \cdot \rangle \\
 \langle A \rightarrow B_1 \dots B_n \cdot \rangle &\rightarrow \langle A \rightarrow B_1 \dots B_{n-1} \cdot B_n \rangle B_n \\
 \langle A \rightarrow B_1 \dots B_{n-1} \cdot B_n \rangle &\rightarrow \langle A \rightarrow B_1 \dots B_{n-2} \cdot B_{n-1} B_n \rangle B_{n-1} \\
 &\dots \\
 \langle A \rightarrow B_1 \cdot B_2 \dots B_n \rangle &\rightarrow B_1 \quad (\text{Left-Corner})
 \end{aligned}$$

oder

$$\begin{aligned}
 &\dots \\
 \langle A \rightarrow B_1 \cdot B_2 \dots B_n \rangle &\rightarrow \langle A \rightarrow \cdot B_1 B_2 \dots B_n \rangle B_1 \quad (\text{Earley}) \\
 \langle A \rightarrow \cdot B_1 B_2 \dots B_n \rangle &\rightarrow \varepsilon \quad (\text{Earley})
 \end{aligned}$$

Nachteil: Redundanz falls die rechten Seiten mehrerer Regeln in derselben Weise beginnen: Kann optimiert werden!

Überblick

Grammatik-Transformationen

Chomsky-Normalform

Greibach-Normalform

Transformationen im Earley-und LC-Parser

Vergleich von Parsing-Algorithmen

Komplexität

Komplexität des CYK-Parsing

Komplexität des Earley-Parsers

Parsing-Komplexität Ergebnisse

Vergleich von Parsing-Algorithmen

TDF: Top-down filter

LA: Look-ahead filter

direction	Type	TDF	LA	Gramm.	Method
top-down	backtr.	–	no	no left rec.	TD with backtr.
top-down	determ.	–	yes	LL(1)	LL(1) parser
top-down	chart	–	no	all	Earley w/o LA
top-down	chart	–	yes	all	Earley with LA

Vergleich von Parsing-Algorithmen

TDF: Top-down filter

LA: Look-ahead filter

direction	Type	TDF	LA	Gramm.	Method
top-down	backtr.	–	no	no left rec.	TD with backtr.
top-down	determ.	–	yes	LL(1)	LL(1) parser
top-down	chart	–	no	all	Earley w/o LA
top-down	chart	–	yes	all	Earley with LA
bottom-up	backtr.	no	no	no ϵ , no cycles	BU with backtr.
bottom-up	determ.	yes	no	LR(0)	LR(0) parser
bottom-up	determ.	yes	yes	LR(1)	LR(1) parser
bottom-up	pseudodet.	yes	yes	all	Tomita parser
bottom-up	chart	no	no	CNF	CYK parser
bottom-up	chart	no	no	(no ϵ)	Left-corner parser
bottom-up	chart	yes	no	(no ϵ)	LC + Reach.
bottom-up	chart	yes	yes	(no ϵ)	LC + Reach. + LA

Überblick

Grammatik-Transformationen

Chomsky-Normalform

Greibach-Normalform

Transformationen im Earley-und LC-Parser

Vergleich von Parsing-Algorithmen

Komplexität

Komplexität des CYK-Parsing

Komplexität des Earley-Parsers

Parsing-Komplexität Ergebnisse

Parsing-Komplexität

- ▶ Alle Kontext-freien Parser liefern äquivalente Ergebnisse.
- ▶ Parsing-Geschwindigkeit und Platzbedarf sind entscheidend.
- ▶ Komplexität:
Wie schnell wächst Runtime (Platzbedarf) mit der Inputlänge im schlimmsten Fall?
- ▶ Dazu Abschätzungen mit *Bachmann/Landau-Symbolen*: 'Big O of ...'

Parsing-Komplexität

- ▶ Bachmann/Landau Symbole: 'Groß O von ...'
- ▶ Die Komplexitätsklasse $O(f(n))$ umfasst alle Algorithmen deren Komplexität $g(n)$ den folgenden Bedingungen genügt:

$$\exists_K \forall_{n>0} : Kf(n) > g(n)$$

Beispiele:

$$O(3n^2 + 5n + \log(n) + 13) = O(n^2)$$

$$O(1,5^{\sqrt{n}} + 7n^{12} - 4n^2) = O(1,5^{\sqrt{n}})$$

$$O(3n^2 * \log(n) + 5n + 13) = O(n^2 * \log(n))$$

$$O(3n^2 + \log(m) + 5n + 13) = O(n^2 + \log(m))$$

Parsing-Komplexität

- ▶ Wir interessieren uns für die Komplexität in Abhängigkeit von der Eingabelänge und (weniger wichtig) von der Größe der Grammatik und dabei für
- ▶ die Space-Komplexität und
- ▶ die Runtime-Komplexität

Überblick

Grammatik-Transformationen

Chomsky-Normalform

Greibach-Normalform

Transformationen im Earley-und LC-Parser

Vergleich von Parsing-Algorithmen

Komplexität

Komplexität des CYK-Parsing

Komplexität des Earley-Parsers

Parsing-Komplexität Ergebnisse

Komplexität des CYK-Parsing

Algorithmus:

Recognizer($x_1 \dots x_n$)

- (1) For $k := 1$ to n
- (2) $t_{k-1,k} := \{A \mid A \rightarrow x_k \in P\}$
- (3) For $i := k - 2$ down to 0
- (4) $t_{ik} := \{\}$
- (5) For $j := i + 1$ to $k - 1$
- (6) For all $A \rightarrow B C \in P$
- (7) If $B \in t_{ij}$ and $C \in t_{jk}$ then
- (8) $t_{ik} := t_{ik} \cup \{A\}$
- (9) If $S \in t_{0n}$ then accept else report error

Space-Komplexität des CYK-Parsing

- ▶ Grammatik $G = (V, \Sigma, P, S)$
- ▶ Jedes Feld der Chart ist eine Tabelle mit einem (Boole'schen) Eintrag für jedes Nicht-Terminal.

⇒

- ▶ Space-Komplexität des Erkenners (ohne den Platz für die Grammatik): $O(n^2 * |V|)$
- ▶ Space-Komplexität des Parsers ist $O(n^3 * |P|)$, da wir bis zu $2 * |P| * n$ viele Pointer in jedes Feld der Chart eintragen müssen die auf die Child-Knoten verweisen.

Runtime-Komplexität des CYK-Parsing

- ▶ Grammatik $G = (V, \Sigma, P, S)$
- ▶ 4 verschachtelte Loops
- ▶ Zeilen (7) and (8) werden höchstens

$$\sum_{k=1}^n \sum_{i=0}^{k-2} \sum_{j=i+1}^{k-1} \sum_{p \in P} 1$$

mal ausgeführt.

- ▶ Einfügen eines Elements in die Chart erfordert konstante Zeit (3-dim. Array). $\Rightarrow O(n^3 * |P|)$ Schritte
- ▶ Initialisierung der Cart (Zeile 4) erfordert $O(n^2 * |V|)$ Schritte.
- ▶ Ergebnis: $O(n^3 * |P|)$

Überblick

Grammatik-Transformationen

Chomsky-Normalform

Greibach-Normalform

Transformationen im Earley-und LC-Parser

Vergleich von Parsing-Algorithmen

Komplexität

Komplexität des CYK-Parsing

Komplexität des Earley-Parsers

Parsing-Komplexität Ergebnisse

Earley-Erkenner

Recognize($x_1, \dots, x_n$)

$t_0 := \{\}$

Predict($S, 0$)

For $j := 1$ to n

$t_j := \{\}$

For all $\langle A \rightarrow \alpha \cdot x_j \beta, i, j-1 \rangle \in t_{j-1}$

Store($A \rightarrow \alpha x_j \cdot \beta, i, j$)

If $\langle S \rightarrow \alpha \cdot, 0, n \rangle \in t_n$ for some α then

accept

else

report error

Aktionen des Earley-Erkenners

Store($A \rightarrow \alpha \cdot \beta, i, k$)

If $\langle A \rightarrow \alpha \cdot \beta, i, k \rangle \notin t_k$ then

$t_k := t_k \cup \{ \langle A \rightarrow \alpha \cdot \beta, i, k \rangle \}$

If $\beta = \varepsilon$ then

Complete(A, i, k)

else if $\beta = B\gamma$ for some B, γ then

If $\langle B \rightarrow \delta \cdot, k, k \rangle \in t_k$ then

Store($A \rightarrow \alpha B \cdot \gamma, i, k$)

else

Predict(B, k)

Aktionen des Earley-Erkenners

Predict(A, i)

If $\text{called}[A, i] = \text{false}$ then

$\text{called}[A, i] := \text{true}$

 For all $A \rightarrow \alpha \in P$

Store($A \rightarrow \cdot \alpha, i, i$)

Complete(A, j, k)

If $\text{called}[A, j, k] = \text{false}$ then

$\text{called}[A, j, k] := \text{true}$

 For all $\langle B \rightarrow \beta \cdot A\gamma, i, j \rangle \in t_j$

Store($B \rightarrow \beta A \cdot \gamma, i, k$)

Space-Komplexität des Earley-Erkenners/-Parsers

- ▶ Im schlimmsten Fall, enthält jedes Feld der Chart alle Punktregeln. Es gibt also höchstens $n^2 * |G|$ Einträge, wobei $|G|$ die Anzahl der Punktregeln ist.

⇒

- ▶ Der Earley-Erkennen benötigt $O(n^2 * |G|)$ viel Platz.
- ▶ Der Earley Parser benötigt $O(n^2 * |G| * n) = O(n^3 * |G|)$ viel Platz.

Runtime-Komplexität des Earley-Parsers

- ▶ Die Store-Funktion (erste Ebene): $O(n^2 * |G|)$ (vgl. space)
- ▶ Die Predict-Funktion fügt im schlimmsten Fall jede Grammatikregel in jedes Feld der Diagonalen: höchstens $(n + 1) * |P|$ Einträge.
- ▶ Die Complete-Funktion bildet im schlimmsten Fall jede mögliche Punktregel für jeden möglichen Teilstring in höchstens n verschiedenen Arten: $|G| * n(n + 1)/2 * n$
- ▶ Zusammen erhalten wir höchstens $O(n^2 * |G|) + O(n * |P|) + O(n^3 * |G|)$, d.h. $O(n^3 * |G|)$ Schritte falls jede Aktion in konstanter Zeit erfolgt.

Komplexität des Earley-Parsers (Besondere Fälle)

Komplexität für unambige Grammatiken

- ▶ Da es keine Ambiguitäten gibt, versucht der Completer nie dieselbe Punktregel ein zweites Mal in dasselbe Feld einzutragen.
- ▶ D.h. der innere Loop der Complete-Funktion wird höchstens $O(n^2 * |G|)$ mal ausgeführt.
- ▶ RuntimeKomplexität also:
$$O(n^2 * |G|) + O(n * |P|) + O(n^2 * |G|) = O(n^2 * |G|)$$

Komplexität des Earley-Parsers (Besondere Fälle)

Komplexität für SLR Grammatiken

- ▶ SLR-Parser prozessieren den Input in linearer Zeit, d.h. in $O(n)$ Schritten.
- ▶ Jeder SLR-Zustand entspricht höchstens N Punktregeln des Earley-Parsers.
- ▶ Also fügt der Earley-Parser höchstens $O(n)$ viele Punktregeln in die Chart ein.
- ▶ Da SLR-Grammatiken immer eindeutig sind, leitet der Completer dieselbe Punktregel für denselben Input-Teilstring nie in einer zweiten Weise ab.
- ▶ Deshalb benötigt der Completer höchstens $O(n)$ Schritte.
⇒ lineare Laufzeit-Komplexität $O(n)$

Überblick

Grammatik-Transformationen

Chomsky-Normalform

Greibach-Normalform

Transformationen im Earley-und LC-Parser

Vergleich von Parsing-Algorithmen

Komplexität

Komplexität des CYK-Parsing

Komplexität des Earley-Parsers

Parsing-Komplexität Ergebnisse

Parsing-Komplexität Ergebnisse I

- ▶ CYK
 - ▶ runtime complexity $O(n^3)$
 - ▶ space complexity $O(n^2)$ (recognizer)
 - ▶ space complexity $O(n^3)$ (parser)
- ▶ Earley
 - ▶ runtime complexity $O(n^3)$
 - ▶ space complexity $O(n^2)$ (recognizer)
 - ▶ space complexity $O(n^3)$ (parser)
 - ▶ runtime complexity for unambiguous grammars $O(n^2)$
 - ▶ runtime complexity for SLR grammars $O(n)$
- ▶ Left-corner
 - ▶ runtime complexity $O(n^3)$
 - ▶ space complexity $O(n^2)$ (recognizer)
 - ▶ space complexity $O(n^3)$ (parser)
 - ▶ runtime complexity for unambiguous grammars $O(n^2)$
 - ▶ runtime complexity for SLR grammars $O(n)$ (with top-down filter)

Parsing-Komplexität Ergebnisse II

- ▶ derivation-oriented parsers (BU, TD)
 - ▶ runtime complexity $O(c^n)$
 - ▶ space complexity $O(n)$
- ▶ table-driven parsers (LL, LR)
 - ▶ runtime complexity $O(n)$
 - ▶ space complexity $O(n)$
- ▶ Tomita
 - ▶ runtime complexity may increase exponentially with the number of rules $\Omega(2^{\sqrt{|P|}})$
- ▶ parse forest output (not binarized)
 - ▶ Runtime complexity is exponential in the length of the rules