Kategorialgrammatiken und Kontextfreie Grammatiken

Kilian Evang, Christian Wurm

Düsseldorf, 27.10.2022

CG und CFG

Definition CG

Eine (applikative) Kategorialgrammatik ist ein Tupel $(Var, \mathtt{Lex}, S, \zeta)$, wobei

1. Var und damit $Cat(Var)$, also die Menge der (atomaren) syntaktischen Kategorien
2. $\mathtt{Lex}$, das Lexikon. Das können Worte sein (eher linguistisch) oder Buchstaben (eher formal-sprachlich). Im letzteren Fall wäre $\mathtt{Lex} = \Sigma$.
3. S ist eine *ausgezeichnete Kategorie*, wie man sie von CFG kennt
4. ζ ist die Typzuweisung.

CFG

Definition CFG

Eine CFG ist ein Tupel $(S, \mathbf{N}, \Sigma, R)$, wobei

1. $\mathbf{N}$ die Menge der Nicht-terminale ist
2. Σ die Menge der Terminale
3. $S \in \mathbf{N}$ ist das Startsymbol
4. $R \subseteq \mathbf{N} \times (\mathbf{N} \cup \Sigma)^*$ ist die Menge der Produktionen/Regeln

Von CG zu CFG

Sei *subcat* die Menge der *Subkategorien* (reflexiv) von $\zeta[\mathtt{Lex}]$.

Das ist also die Menge der Kategorien die vorkommen *können* in einer CG-Ableitung.

Setze $CFG_{CG} = (S, \mathbf{N}_{CG}, \mathtt{Lex}, R_{CG}\}$, wobei

$$\begin{aligned} \mathbf{N}_{CG} &= \mathit{subcat} \\ R_{CG} &= \{(\alpha \to \alpha/\beta\ \beta : \alpha/\beta, \beta \in \mathbf{N}_{CG}\} \\ &\cup \{(\alpha \to \beta\ \beta\backslash\alpha : \beta, \beta\backslash\alpha \in \mathbf{N}_{CG}\} \\ &\cup \{\alpha \to w : \alpha \in \zeta(w)\} \end{aligned}$$

Von CG zu CFG

Es ist ziemlich offensichtlich dass das funktioniert, daraus folgt also:

Lemma
Jede Sprache, die mit einer (applikativen) CG generiert werden kann, kann auch mit einer CFG generiert werden.

Die Gegenrichtung wird uns etwas mehr Arbeit machen.

Greibach Normal Form

Jede CFG kann transformiert werden in eine äquivalente CFG in **Greibach Normalform**, d.h. alle Regeln haben die Form:

$$N \to a\vec{M}, \text{ wobei } N \in \mathbf{N},\ \vec{M} \in \mathbf{N}^*,\ a \in \Sigma$$

Das gute an GNF ist: alle Regeln führen ein Terminal ein. Wir können also jede Regel in eine Zuweisung $\zeta(a)$ übersetzen! Aber wie?

$$(N \to aM_1 ... M_i) \Longrightarrow (N/M_i)/...)/M_1 \in \zeta(a) \qquad (1)$$

Von CFG zu CG

Definition CFG

Eine CFG ist ein Tupel $(S, \mathbf{N}, \Sigma, R)$, wobei

1. $\mathbf{N}$ die Menge der Nicht-terminale ist
2. Σ die Menge der Terminale
3. $S \in \mathbf{N}$ ist das Startsymbol
4. $R \subseteq \mathbf{N} \times (\mathbf{N} \cup \Sigma)^*$ ist die Menge der Produktionen/Regeln

Wir bekommen dann:

- $Var_{CFG} = \mathbf{N}$
- $\mathtt{Lex} = \Sigma$
- $((N/M_i)/...)/M_1 \in \zeta_{CFG}(a)$ gdw. $(N \to aM_1...M_i) \in R$

Von CFG zu CG

Ist das äquivalent? Wir zeigen ein etwas allgemeineres Ergebnis (das ist manchmal einfacher):

$$S \vdash_{CFG} a_1 ... a_i N_1 ... N_m \tag{2}$$

gdw. es gibt $\alpha_1 \in \zeta(a_1), ..., \alpha_i \in \zeta(a_i)$, so dass

$$\alpha_1, ..., \alpha_i, N_1, ..., N_m \vdash_{CG} S \tag{3}$$

(der rein eliminative Kalkül!) Wir zeigen dass mit einer Induktion über i.

Applikative CG: Übung

Zu zeigen:

$$S \vdash_{CFG} a_1 ... a_i N_1 ... N_m \quad \text{gdw.} \quad \alpha_1, ..., \alpha_i, N_1, ..., N_m \vdash_{CG} S$$

Induktionsbasis: $i = 1$.

- $\Rightarrow$ Qua GNF haben wir: $S \rightarrow a_1 N_1 ... N_m$.
- $\Leftrightarrow$ $((S/N_m)/...)/N_1 = \alpha_1 \in \zeta_{CFG}(a_1)$.
- $\Leftrightarrow$ $\alpha_1, N_1, ..., N_m \vdash_{CG} S$.

Geht also in beide Richtungen!

Äquivalenz

Induktionshypothese: Nimm an für ein $i \in \mathbb{N}$ gilt:

$$S \vdash_{CFG} a_1 ... a_i N_1 ... N_m \text{ gdw. } \alpha_1, ..., \alpha_i, N_1, ..., N_m \vdash_{CG} S$$

Außerdem:

$$S \vdash_{CFG} a_1 ... a_i a_{i+1} N_1 ... N_j \tag{4}$$

Dann wurde appliziert eine Regel

$$N_1' \rightarrow a_{i+1} N_1 ... N_k \tag{5}$$

Also gibt es $((N_1'/N_k)/...)/N_1 \in \zeta(a_{i+1})$, und

Äquivalenz

Qua IH gilt:

$$S \vdash_{CFG} a_1 ... a_i N_1' N_{k+1} ... N_m$$

gdw.

$$\alpha_1, ..., \alpha_i, N_1', N_{k+1}, ..., N_m \vdash_{CG} S$$

Also:

$$S \vdash_{CFG} a_1 ... a_i \quad a_{i+1} N_1 ... N_k \quad N_{k+1} ... N_m$$

gdw.

$$\alpha_1, ..., \alpha_i, \quad \alpha_{i+1} N_1, ... N_k \quad , N_{k+1}, ..., N_m \vdash_{CG} S$$

Der Satz von Bar-Hillel, Gaifman und Shamir

Daraus folgt (Spezialfall $m = 0$)

Satz

Eine Sprache L wird generiert von einer CFG genau dann wenn Sie generiert wird von einer applikativen CG.

So sehen also solche Beweise aus!

Der Satz von Buszkowski

Satz

Eine Sprache L wird generiert von einer CFG genau dann wenn Sie generiert wird von einer AB-Grammatik.

Das ist überhaupt nicht trivial:

- wir müssen verhindern, dass Sätze ableitbar werden, die bislang nicht ableitbar waren
- sonst wird unsere Sprache zu groß!
- Der Beweis besteht also darin, ungewollte Inferenzen zu verhindern.

Machen wir aber heute nicht. (Puh!)

Themenwechsel

Neues Thema: **Der AB Kalkül und Typentransparenz**

Der AB-Kalkül

$$(ax) \qquad \alpha \vdash \alpha$$

$$(\mathbf{I}-/) \qquad \frac{\Gamma, \alpha \vdash \beta}{\Gamma \vdash \beta/\alpha} \qquad (\mathbf{I}-\backslash) \qquad \frac{\alpha, \Gamma \vdash \beta}{\Gamma \vdash \alpha\backslash\beta}$$

$$(/-\mathbf{I}) \qquad \frac{\Delta, \beta, \Theta \vdash \gamma \quad \Gamma \vdash \alpha}{\Delta, \beta/\alpha, \Gamma, \Theta \vdash \gamma}$$

$$(\backslash-\mathbf{I}) \qquad \frac{\Delta, \beta, \Theta \vdash \gamma \quad \Gamma \vdash \alpha}{\Delta, \Gamma, \alpha\backslash\beta, \Theta \vdash \gamma}$$

$$(\text{schnitt}) \qquad \frac{\Gamma, \alpha, \Gamma' \vdash \beta \quad \Delta \vdash \alpha}{\Gamma, \Delta, \Gamma' \vdash \beta}$$

AB-Grammatiken

Eine AB Grammatik $G = (Var, \mathtt{Lex}, S, \zeta)$ generiert ein Wort/Satz $\vec{w} = w_1 w_2 ... w_i \in \mathtt{Lex}^+$, falls gilt:

- es gibt einen Beweis im AB-Kalkül von $\alpha_1, ..., \alpha_i \vdash S$, und
- f.a. $j \in \{1, ..., i\}$ gilt: $\alpha_j \in \zeta(w_j)$

Das ist natürlich praktisch und schnell erledigt (wir lassen einfach die Logik für uns arbeiten).

Der Linguist wird aber die Bäumchen vermissen! Also präsentieren wir noch eine andere Variante von AB.

Natürliches Schließen

$$(\mathsf{E}/)\ \frac{\alpha/\beta \quad \beta}{\alpha} \qquad (\mathsf{E}\backslash)\ \frac{\beta \quad \beta\backslash\alpha}{\alpha}$$

$$(\mathsf{I}/)\ \frac{\begin{array}{c}\dots \quad [\beta] \\ \vdots \\ \alpha\end{array}}{\alpha/\beta} \qquad (\mathsf{I}\backslash)\ \frac{\begin{array}{c}[\beta] \quad \dots \\ \vdots \\ \beta\end{array}}{\beta\backslash\alpha}$$

β in den I-Regeln ist hier eine **Annahme**, die wir mit der Regel **auflösen**. Die Annahme muss dabei am linken/rechten Rand des Beweises stehen!

Typentransparenz

In typentheoretischer Semantik hat jedes lexikalische Element einen Typ (oder mehrere). Die einfachste Typentheorie kommt mit zwei Grundtypen aus:

1. e, dem Typ *entity*, der mehr oder weniger alles umfasst
2. t, dem Typ *truth value*, der Menge $\{0, 1\}$

Dann bekommen wir:

- Eigennamen haben den Typ e
- Nomen haben den Typ $e \rightarrow t$ (Mengen!)
- intransitive Verben haben den Typ $e \rightarrow t$ (ebenso!)
- transitive Verben haben den Typ $e \rightarrow (e \rightarrow t)$

Soweit so gut.

Quantoren

Quantoren haben aber überraschende Typen. Betrachten wir die Bedeutung von zwei Sätzen:

(1) Jedes Kind schläft.

(2) Eine Katze miaut.

Was bedeutet das?

(1): $\|kind\| \subseteq \|schlaft\|$

(2): $\|katze\| \cap \|miaut\| \neq \emptyset$

Quantoren

(1): $\|kind\| \subseteq \|schlaft\|$

(2): $\|katze\| \cap \|miaut\| \neq \emptyset$

Sprich: Quantoren sind Funktionen von Mengen zu Mengen zu Wahrheitswerten:

- ein,jeder: $(e \to t) \to ((e \to t) \to t)$

Damit wäre das erledigt, aber es bleibt die Frage nach **Typentransparenz**

Typentransparenz

Wir haben also syntaktische Typen und semantische Typen:

- schläft : $NP\backslash S$: $e \rightarrow t$

Syntaktische Typen sind *spezifischer*, denn semantische Typen geben keine Direktionalität an.

Typentransparenz

Typentransparenz bedeutet: es gibt einen Homomorphismus von syntaktischen zu semantischen Typen.

Typentransparenz ist nicht lebenswichtig. Dennoch: Typen ohne Typentransparenz sind möglich, aber sinnlos.

(In)transparente Typen in AB

Typen ohne Typentransparenz sind möglich, aber sinnlos:

- Transparente Typen sind schön und machen die semantischen Typen redundant/ableitbar
- Syntaktische Applikation entspricht semantischer Applikation – die Interpretation wird zum Selbstläufer

Das Problem dabei ist:

$\Rightarrow$ Syntaktische Köpfe (Funktoren) sind traditionell Verben

$\Rightarrow$ Semantische Köpfe (Funktoren) sind traditionell Quantoren

Was nun?

(In)transparente Typen in AB

Also wie wird das gelöst? Nehmen wir den Homomorphismus:

$$\begin{aligned} h &: S \mapsto t \\ &: NP \mapsto e \qquad (!) \\ &: CN \mapsto e \to t \end{aligned}$$

(In)transparente Typen in AB

$$\texttt{Jedes} := NP/CN \quad (6)$$
$$\texttt{Kind} := CN \quad (7)$$
$$\texttt{schläft} := NP\backslash S \quad (8)$$

Sei $\tau = h \circ \zeta$. Dann haben wir:

$$\tau(\texttt{Jedes}) = (e \to t) \to e) \quad (9)$$
$$\tau(\texttt{Kind}) = e \to T \quad (10)$$
$$\tau(\texttt{schläft}) = e \to t \quad (11)$$

Das sind *nicht* die Typen die wir semantisch brauchen! Aber:

(In)transparente Typen in AB

$$\frac{\dfrac{NP \quad [NP\backslash S]}{S}}{S/(NP\backslash S)} \tag{12}$$

wobei natürlich gilt:

$$h(S/(NP\backslash S) = (e \to t) \to t \tag{13}$$

Wie das sein soll! D.h. wir können den transparenten Typ *ableiten*. Wir können sogar verschiedene Quantorenlesarten ableiten:

(3) Jeder Junge liebt einen Film mit einem Roboter.

(4) Irgendein Depp mäht immer irgendwo.

Quantoren in CG

Dazu später mehr.

Zusammenfassung

Kategorialgrammatiken sind superspannend!