Formale Sprachen

(parallele und regulierte Ersetzung)

SoSe 2013 in Trier

Henning Fernau
Universität Trier
fernau@uni-trier.de

3. Juni 2013

Formale Sprachen Gesamtübersicht

- Organisatorisches / Einführung
  Motivation / Erinnerung / Fragestellungen

- Diskussion verschiedener Sprachklassen:
  gesteuerte Ersetzungsverfahren
  parallele Ersetzungsverfahren

- Algebraische Ansätze:
  abstrakte Sprachfamilien
  formale Potenzreihen

Regulierte Ersetzungsverfahren

—Motivation
—Graphgesteuerte Ersetzungen und Spezialfälle
—Leerheits-, Wort- und Präfixprobleme
—Hierarchische Beziehungen, Universalität
—Umgebungstests

Motivation

Zweck formaler Sprachen: Modellierung (Analyse, Beschreibung) syntaktischer Phänomene aus:

1. Natürlichen Sprachen (Linguistik)

2. Programmiersprachen

3. Biologie (und andere Bereiche)

Beobachtung: Die Welt ist nicht kontextfrei.

Dabei wichtig:

1. Einfachheit der Modelle

2. Natürlichkeit der Modelle

3. Algorithmische Fragestellungen

Linguistische Motivation

John, Mary, David, ...
are
a widower, a widow, a widower, ...
respectively.

Übersetzen wir weibliche Bezeichnungen nach a und männliche nach b sowie "are" und "respectively" nach c, so erhalten wir als Formalisierung solcher Bezeichnungen:

$$\mathrm{L} = \{ww \mid w \in \{\mathrm{a}, \mathrm{b}\}^{+}\{\mathrm{c}\}\}$$

Ist Englisch kontextfrei ?

Trügerisches Argument: $\mathtt{L}$ aus vorigem Beispiel ist nicht kontextfrei.

Wenn Englisch als Sprache $\mathtt{E}$ kontextfrei wäre, so aber auch $\mathtt{E} \cap \mathtt{R}$ für jede reguläre Sprache $\mathtt{R}$.

Mit geeignetem $\mathtt{R}$ könnten wir $\mathtt{L}$ quasi “herausschneiden” aus $\mathtt{E}$.
Daher ist $\mathtt{E}$ nicht kontextfrei.
Alternativ: Definition eines geeigneten a-Transduktors, der $\mathtt{E}$ auf $\mathtt{L}$ abbildet.

Hinweis: Abschlusseigenschaften kann man oft gebrauchen, um Echtheit von Inklusionen zwischen Sprachfamilien nachzuweisen.

Sind Programmiersprachen kontextfrei ?

Betrachte folgendes Fragment:

```
{
     int x;
     y=1
}
```

Unter der Maßgabe, dass eine Variable deklariert sein muss vor ihrer Benutzung, ist obiges Programmstück nur dann richtig, wenn beide vorkommenden Variablen identisch sind.

Wieso sind also Programmiersprachen (mit dieser Eigenschaft) im Allgemeinen nicht kontextfrei ?

Ein logisches Beispiel

Betrachten den Aussagenkalkül mit den logischen Verknüpfungen $\wedge$, $\vee$, $\Longrightarrow$, $\neg$ (sowie Klammern).
Atomare Aussagen werden durch Wörter über $\{a\}$ unär kodiert.
τ sei die Sprache der Theoreme (also hier der Tautologien) in dem skizzierten Kalkül. Bekannt ist: $\tau \in \mathcal{L}_1$.

Betrachte die reguläre Sprache $R = \{(x \vee y) \Longrightarrow z \mid x, y, z \in \{a\}^*\}$.
$\tau \cap R = \{(x \vee x) \Longrightarrow x \mid x \in \{a\}^*\} \notin \mathcal{L}_2$, daher $\tau \notin \mathcal{L}_2$.

Erstaunlicherweise unbekannt:
Ist τ, eingeschränkt auf endliche Menge atomarer Aussagen, kontextfrei ?

Relationen und Funktionen als Sprachen

Die Sprache $\{a_1^n a_2^n \mid n \geq 0\}$ könnte man auch als Unärkodierung der Diagonalen auf den natürlichen Zahlen $\mathbb{N}$ deuten.

Ist allgemeiner R eine k-stellige Relation auf den natürlichen Sprachen, so entspricht dieser Relation die Sprache

$$L_R = \{a_1^{m_1} a_2^{m_2} \cdots a_k^{m_k} \mid (m_1, \ldots, m_k) \in R\}$$

über dem Alphabet $\{a_1, \ldots, a_k\}$.

Interpretieren wir eine k-stellige Funktion $f : \mathbb{N}^k \to \mathbb{N}$ als $(k+1)$-stellige Relation, so liefert f eine Sprache L_f.

Viele scheinbar einfache Relationen und Funktionen liefern nicht kontextfreie Sprachen.

Beispiel: $f(x) = x^2$, $g(x, y) = xy$, aber: $h(x, y) = x + y$. Was ist mit: $\phi(x, y) = \max(x, y)$?

D0L Systeme

werden wir später noch genauer studieren...
Hier schon einmal die Definition:
Ist $h : \Sigma^* \to \Sigma^*$ ein Morphismus und $\omega \in \Sigma^*$ ein Axiom, so ist die vom D0L System $G = (\Sigma, h, \omega)$ beschriebene Sprache:

$$L(G) = \{h^i(\omega) \mid i \in \mathbb{N}\} = \{\omega, h(\omega, h(h(\omega)), \dots\}.$$

Da h Abbildung, ist auch $f_G : \mathbb{N} \to \mathbb{N}, n \mapsto \ell(h^n(\omega))$ eine Abbildung, die *Wachstumsfunktion* von G.

So können z.B. die Fibonacci-Zahlen beschrieben werden, aber auch die Zweierpotenzen (wie ?) .

Siehe auch 2. Hauptthema der Vorlesung: Die Welt ist nicht sequentiell.

Flussdiagramme für (kontextfreie) Gramm. $\rightsquigarrow$ *programmierte* Grammatiken.

- Arbeit auf Satzformen als Zustandsraum
- Operationen: (sequentielles) Anwenden von Regeln
- Abfrage (zur Verzweigung): Anwendbarkeit von Regeln
- Beispiele (Tafel): $L_1 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$, $L_2 = \{a^{2^n} \mid n \in \mathbb{N}\}$.
  Wo liegen die Sprachen in der Chomsky-Hierarchie ?

Ein einheitlicher Rahmen für regulierte Ersetzung

Eine *graphgesteuerte Grammatik* kurz *G Grammatik*, ist ein 8-Tupel
$G = (V_N, V_T, P, S, \Gamma, \Sigma, \Phi, h)$ mit:

- V_N, V_T, P, S sind die Mengen der Nonterminale, Terminale, kontextfreien *Kernregel*n, sowie das Startsymbol;
- Γ ist ein zweigefärbter Digraph, d.h., $\Gamma = (U, E)$, wobei U eine endliche Knotenmenge ist und $E \subseteq U \times \{g, r\} \times U$ ist eine durch g oder r (“grün” oder “rot”) gefärbte Bögenmenge;
- $\Sigma \subseteq U$ sind die *Anfangsknoten*;
- $\Phi \subseteq U$ sind die *Zielknoten*;
- $h : U \to (2^P \setminus \{\emptyset\})$ setzt Knoten mit Regelmengen in Verbindung

Wir betrachten zwei Arten von *Vorkommenstests*:
$(x, u) \Rightarrow (y, v)$ (bzw. $(x, u) \Rightarrow_c (y, v)$) gilt in G mit $(x, u), (y, v) \in (V_N \cup V_T)^* \times U$, wenn entweder, für ein $(u, g, v) \in E$,

$$x = x_1 \alpha x_2, \quad y = x_1 \beta x_2, \quad \alpha \to \beta \in h(u),$$

oder jede (bzw. eine) Regel von $h(u)$ ist nicht auf x anwendbar, dann gilt $y = x$, $(u, r, v) \in E$.
Die durch G erzeugte Sprache ist: $L_{[c]}(G) =$

$$\{x \in V_T^* \mid \exists u \in \Sigma \exists v \in \Phi(S, u) \stackrel{*}{\Rightarrow}_{[c]} (x, v)\}.$$

Übung 1 *In welchem Sinne ist so eine G Grammatik ein Ersetzungssystem?*

Sprachfamilien:

- $\mathcal{L}(\mathrm{G}_{[\mathrm{c}]}, \mathrm{CF,ac})$: der allgemeine Fall. (*mit Vorkommenstest*)
- $\mathcal{L}(\mathrm{G}_{[\mathrm{c}]}, \mathrm{CF})$: nur grüne Bögen (*ohne Vorkommenstest*)
- $\mathcal{L}(\mathrm{G}_{[\mathrm{c}]}, \mathrm{CF,ut})$: Es gibt es einen grünen Bogen von $\mathfrak{u}$ nach ν gdw. es gibt einen roten Bogen von $\mathfrak{u}$ nach ν (*mit unbedingtem Übergang*)

Die Notation CF$-\lambda$ statt CF weist auf das Verbot löschender Regeln hin.

Hinweis: Auch reguläre, kontextsensitive oder uneingeschränkte Kernregeln denkbar! Wie sehen die entsprechenden Sprachfamilien aus ?

Spezialfälle von G Grammatiken:

- **P***rogrammierte Grammatiken*
  - Jeder Knoten enthält genau eine Regel.
  - Jeder Knoten ist Anfangs- und Zielknoten.

  Übliche Schreibweise der Regeln: $(\mathfrak{r} : A \to w, \sigma, \phi)$.

  Sprachfamilien: $\mathcal{L}(\mathrm{P}, \mathrm{CF}, \mathrm{ac})$, $\mathcal{L}(\mathrm{P}, \mathrm{CF}, \mathrm{ut})$, sowie $\mathcal{L}(\mathrm{P}, \mathrm{CF})$.
- Grammatiken mit **r***egulärer Steuerung (***c***ontrol*)
  - Jeder Knoten enthält genau eine Regel.
  - Gibt es einen roten Bogen von u nach v, so dort auch einen grünen.
- Eine **M***atrix-Grammatik* ist eine rC Grammatik mit folgender zusätzlicher Einschränkung:
  - Nur Anfangsknoten dürfen mehr als einen eingehenden grünen Bogen besitzen.
  - Nur Zielknoten dürfen mehr als einen ausgehenden grünen Bogen besitzen.
  - Zwischen jedem Zielknoten und jedem Anfangsknoten gibt es einen grünen Bogen.

- **T***ime-***v***ariant grammars* / *Zeitvariierende Grammatiken*
  - Gibt es einen roten Bogen von $\mathfrak{u}$ nach ν, so dort auch einen grünen.
  - Der durch die grünen Bögen induzierte Graph ist ein Kreis.
  - Es gibt einen Anfangsknoten, aber jeder Knoten ist Zielknoten.

  Übliche Notation: eine periodische Funktion ϕ, die natürliche Zahlen auf Regelmengen abbildet, sodass $\phi(\mathfrak{n})$ den $\mathfrak{n}$-ten Ableitungsschritt beschreibt.

  Hier können wir sinnvoll sowohl die $\Rightarrow$ als auch die $\Rightarrow_{\mathfrak{c}}$ Ableitungsdefinition benutzen. Dies führt zu den Sprachfamilien $\mathcal{L}(\mathsf{TV}_{[\mathfrak{c}]}, \mathsf{CF}, \mathsf{ac})$, $\mathcal{L}(\mathsf{TV}_{[\mathfrak{c}]}, \mathsf{CF}, \mathsf{ut})$ und $\mathcal{L}(\mathsf{TV}_{[\mathfrak{c}]}, \mathsf{CF})$.

Ähnlich kann man betrachten: **M**atrix **s**et (Matrix-Mengen) und **r**egular **s**et **c**ontrol Grammatiken.

Was man so weiß...

Satz 1 *Gilt* $X \in \{G_c, P, M, rC, TV_c\}$, $Y \in \{CF, CF-\lambda\}$, $Z \in \{ac, \lambda\}$, *so:* $\mathcal{L}(X, Y, Z) = \mathcal{L}(G, Y, Z)$.

Satz 2

- $\mathcal{L}(G, CF) \subsetneq \mathcal{L}(G, CF, ac) = \mathcal{L}_0$.
- $\mathcal{L}(G, CF-\lambda) \subsetneq \mathcal{L}(G, CF-\lambda, ac) \subsetneq \mathcal{L}_1$.

Programmierte Grammatiken

Ursprüngliche Deutung der Regel $(p : A \to w, \sigma, \phi)$:

- im Allgemeinen:
  `if` $A \to w$ is applicable `then`
  `begin` apply $A \to w$;
  continue with a rule from σ
  `end`
  `else begin` continue with a rule from ϕ
  `end`.
- $\phi = \emptyset$ (ohne Vorkommenstest):
  `if` $A \to w$ is applicable `then`
  `begin` apply $A \to w$;
  continue with a rule from σ
  `end`.
- $\phi = \sigma$ (mit unbedingtem Übergang):
  `if` $A \to w$ is applicable `then`
  `begin` apply $A \to w$
  `end`;
  continue with a rule from σ.

Programmed Grammars—The Classical Definition

A *(context-free) programmed grammar (with appearance checking)* is given by

$$G = (N, \Sigma, P, S),$$

where

- N is the nonterminal alphabet,
- Σ is the terminal alphabet,
- $S \in N$ is the start symbol and
- P is a finite set of rules of the form

  $$(r : A \to w, \sigma(r), \phi(r)),$$

  where
  - $r : A \to w$ is context-free, i.e., $A \in N$ and $w \in (N \cup \Sigma)^*$,

- r is a label,
- $\sigma(r)$, the *success field* of r, and $\phi(r)$, the *failure field* of r, are two sets of labels of rules from P.

$\Lambda(P)$ is the set of all labels of rules from P.

For $(x_1, r_1), (x_2, r_2) \in (N \cup \Sigma)^* \times \Lambda(P)$, we write $(x_1, r_1) \Rightarrow (x_2, r_2)$ iff either

$$x_1 = yAz,\ x_2 = ywz,\ (r_1 : A \to w, \sigma(r_1), \phi(r_1)) \in P, \text{ and } r_2 \in \sigma(r_1)$$

or $x_1 = x_2$, $(r_1 : A \to w, \sigma(r_1), \phi(r_1)) \in P$, A does not occur in x_1 and $r_2 \in \phi(r_1)$. Let $\overset{*}{\Rightarrow}$ denote the reflexive transitive hull of $\Rightarrow$. The language generated by G is defined as

$$L(G) = \{w \in \Sigma^* \mid (S, r_1) \overset{*}{\Rightarrow} (w, r_2) \text{ for some } r_1, r_2 \in \Lambda(P)\}.$$

$\mathcal{P}$: the family generated by programmed grammars;
$\mathcal{P}_k$: the family generated by programmed grammars with at most k nonterminals

Programmierte Grammatiken—Ein Beispiel

$G = (\{A, B\}, \{a\}, P, A)$, wobei P die folgenden Regeln enthält:

$$\begin{array}{lll} (1 & : A \to BB, & \{1\}, \quad \{2\}), \\ (2 & : B \to A, & \{2\}, \quad \{1, 3\}), \\ (3 & : A \to a, & \{3\}, \quad \{3\}). \end{array}$$

Dann gilt: $L(G) = \{a^{2^n} \mid n \geq 0\}$.

Programmierte Grammatiken mit unbedingtem Übergang—Beispiel

$G_1 = (\{A, B, F\}, \{a\}, P_1, A)$, wobei P die folgenden Regeln enthält:

$$\begin{array}{lll}
(1 & : A \to BB, & \{1,2\}, \quad \{1,2\}), \\
(2 & : A \to F, & \{3\}, \quad \{3\}), \\
(3 & : B \to A, & \{3,4\}, \quad \{3,4\}), \\
(4 & : B \to F, & \{1,5\}, \quad \{1,5\}), \\
(5 & : A \to a, & \{5\}, \quad \{5\}).
\end{array}$$

Dann gilt: $L(G_1) = \{a^{2^n} \mid n \geq 0\}$.

Übung 2 *Versuchen Sie, weitere Ihnen bekannte nicht-kontextfreie Sprachen mit G Grammatiken darzustellen.*

Übung 3 *Welche Sprachklassen ergeben sich, wenn man anstelle von kontext-freien nur reguläre Kernregeln zuließe?*

Anschluss zum vorigen Foliensatz

Übung 4 *Welche Abschlusseigenschaften gelten für die soeben eingeführten Sprachfamilien ?!*
Unter Zugrundelegung der obigen Sätzen bleiben dafür noch fünf interessante Sprachklassen zu untersuchen.
Finden sich Trios, AFLs etc. darunter ?

Auf dem Weg zu Satz 1 (hier nur für den Fall ohne Vorkommenstest)

Lemma 1. Zu jeder graphgesteuerten Grammatik G (ohne VT) gibt es eine äquivalente graphgesteuerte Grammatik G' (ohne VT), in welcher zu jedem Knoten genau eine Regel assoziiert ist.

Konstruktion: Ist ν ein Knoten, zu dem $h(\nu)$ mehr als eine Regel zuordnet, so ersetze ν durch $|h(\nu)|$ Kopien; jede dieser Kopien enthalte genau eine der Regeln von $h(\nu)$, und die Vorgänger- bzw. Nachfolger-Knoten von ν sind gerade die Vorgänger- bzw. Nachfolger-Knoten einer jeden Kopie. Nach endlich vielen solchen Ersetzungsschritten gelangen wir zu einer Grammatik G' mit der geforderten Eigenschaft.

Per Induktion sieht man, dass G' äquivalent zu G ist, und zwar unabhängig davon, ob wir G im Modus $\Rightarrow$ oder im Modus $\Rightarrow_c$ ableiten.

Folgerung: Für graphgesteuerte Grammatiken ohne VT sind die beiden Ableitungsbegriffe $\Rightarrow$ und $\Rightarrow_c$ äquivalent.

Auf dem Weg zu Satz 1 (hier nur für den Fall ohne Vorkommenstest)

Lemma 2. Zu jeder graphgesteuerten Grammatik G (ohne VT) gibt es eine äquivalente Matrixgrammatik G' (ohne VT).

Konstruktionsidee: Die Information, in welchem Knoten sich der Ableitungsprozess gerade befindet, wird in einem Extra-Zeichen in der Satzform gespeichert. Simulierende "Matrizen" haben dann immer die Form, dass zunächst die Knoteninformation umgeschaltet wird und dann die eigentliche Regelanwendung (die dem Knoten assoziiert ist) erfolgt. Daneben gibt es Startmatrizen, die (als einzige) ein neues Startzeichen S' ableiten: $S' \to pS$, wobei S das Startzeichen von G ist und p ein möglicher Anfangsknoten. Ist q ein Endknoten, so gibt es noch die Matrizen $[q \to \lambda, A \to w]$ (übliche Matrizenschreibweise !), wobei $A \to w$ zu q assoziiert ist und w ein Terminalwort ist.

Problem: Zusätzliche Einführung von löschenden Regeln. . . lässt sich umgehen.

Übung 5 *Wie ?*

Näheres und Beispiele an der Tafel.

Programmierte Grammatiken müssen genauso behandelt werden.

Achtung: Lemma 1 alleine genügt nicht.

Warum ?

Regeln könnten nicht als Anfangsregeln gemeint sein oder aber “vorzeitig” terminieren (ohne in einem Endknoten zu sein). Deshalb ist es wieder nötig, die Knoteninformation in der Satzform mitzuschleppen.

Da regulär gesteuerte Grammatiken Matrixgrammatiken verallgemeinern, haben wir (bis auf zeitvariierende Grammatiken) für den Fall ohne VT die Behauptung von Theorem 1 gezeigt.

Übung 6 *Gelingen Ihnen Konstruktionen für den rC- (leicht) bzw. TV- (schwieriger) Fall?*

Auf dem Weg zu Satz 2

Hauschildt und Jansen haben einen hübschen Zusammenhang mit der Theorie der Petrinetze gefunden, der es gestattet zu folgern, dass

$$\{a^{2^n} \mid n \geq 0\} \notin \mathcal{L}(\mathsf{G,CF})$$

gilt. Wir machen im Folgenden schwächere Überlegungen, die aber immer noch uns folgendes folgern lassen:

Folgerung: $\mathcal{L}(\mathsf{G,CF})$ enthält nur rekursive Sprachen.

Exkurs: Petrinetze

Ein *Petrinetz* $\Pi = (G, \phi_0)$ wird beschrieben durch einen bipartiten Graphen $G = (S \cup T, E)$ mit
S: Menge der Stellen
T: Menge der Transitionen,
sowie eine *Belegungsfunktion* $\phi_0 : S \to \mathbb{N}$.

Eine Transition $t \in T$ kann *feuern* oder *schalten* bezüglich einer Belegung ϕ, wenn $\forall s \in S : (s,t) \in E \implies \phi(s) \geq 1$. Dann gilt für die *Nachfolgerbelegung* ϕ':

$$\phi'(s) = \begin{cases} \phi(s) - 1, & \text{falls } (s,t) \in E \wedge (t,s) \notin E \\ \phi(s) + 1, & \text{falls } (t,s) \in E \wedge (s,t) \notin E \\ \phi(s), & \text{sonst} \end{cases}$$

Wir schreiben auch $\phi \vdash_\Pi \phi'$ für diesen Umstand.
Eine Belegung ψ heißt *erreichbar* in Π gdw. $\phi_0 \vdash_\Pi^* \psi$.

Mitteilung: Das Erreichbarkeitsproblem ist entscheidbar. (höchst nicht-trivial)

Lemma 3. Das Leerheitsproblem für graphgesteuerte Grammatiken ist entscheidbar, gdw. es entscheidbar ist für solche graphgesteuerten Grammatiken, die lediglich löschende Regeln als Möglichkeit besitzen, ein Terminalwort abzuleiten.

Satz 3 *Das Leerheitsproblem für graphgesteuerte Grammatiken von Vorkommenstest ist äquivalent zum Erreichbarkeitsproblem für Petrinetze.*

Konstruktion: Die Stellenmenge des Petrinetzes besteht aus der Nichtterminalmenge, disjunkt vereinigt mit der Potenzmenge der Knotenmenge des Steuergraphen.
Transitionen simulieren die Anwendung von Regeln.
Dabei gibt die Steuergraphenknotenmengenstelle an, in welchen Knoten sich die simulierte Grammatik befinden könnte.
Ein Endknoten wird erreicht und gleichzeitig ein Terminalwort abgeleitet gdw. im Petrinetz eine Belegung erreicht wird, deren einzig nicht-null belegte Steuergraphenknotenmengenstelle einen Endzustand enthält und in der alle Nichtterminalstellen auf Null gesetzt sind.

Beschreibungskomplexität—Einige Ergebnisse

Es gibt keine allgemeine obere Schranke auf die Zustandszahl von endlichen Automaten.

Shannon: Zustandskomplexität von Turingmaschinen ist zwei.

Minsky: Registermachinen (zu unterscheiden von denen aus Algorithmen und Datenstrukturen) haben Registerkomplexität von zwei.

Păun: Obere Schranke von 6 für die Nichtterminalkomplexität von Matrixgrammatiken (für **RA**) und daraus Nichtterminalkomplexitätsschranke von 8 für programmierte Grammatiken.

Auf der MCU 2001 konnte gezeigt werden: Obere Schranke von 3 für die Nichtterminalkomplexität von programmierten Grammatiken (und daraus von 4 für Matrixgrammatiken).

Hauptresult (der genannten MCU-Arbeit)

Satz 4 $\mathcal{P}_3 = \mathcal{L}_0$.

Beweis: Simuliere (generative!) Turingmaschine

Eine *Turingmaschine* (mit einseitigem Band) ist gegeben durch:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_f, \#_L, \#_R, \#),$$

wobei:

$\#_L/\#_R \in \Gamma$ linke/rechte Randmarkierung,

$\# \in \Gamma$ Blank-Symbol.

Eine *Konfiguration* von M ist beschrieben durch ein Wort

$$c \in \#_L \tilde{\Gamma}^* \#_R Q \cup \#_L \tilde{\Gamma}^* Q \tilde{\Gamma}^* \#_R$$

mit $\tilde{\Gamma} = \Gamma \setminus \{\#_L, \#_R\}$. $c = \#_L w q v$ bedeutet dabei: Der Kopf der Turingmaschine befindet sich gegenwärtig über dem letzten Zeichen a von $\#_L w$.

Codierungen

Sei $c \in (\Gamma \cup Q)^*$ eine Konfiguration von M, $\beta(c) \in \{0,1\}^*$ bezeichne irgendeine binäre Codierung von c mithilfe von $\gamma = \lceil \log_2(|\Gamma| + |Q|) \rceil$ vielen Bits pro Zeichen von $\Gamma \cup Q$.
Wir deuten die Wörter $\beta(c)$ als natürliche Zahlen.

Annahme: $\beta(\#) = 0^\gamma$, $\beta(\#_L) = 0^{\gamma-1}1$, $\beta(\#_R) = 10^{\gamma-1}$.
Beobachte: $|\beta(c)| = |c|\gamma$.
c kann eindeutig über Alphabet $\{A\}$ codiert werden vermöge $A^{\beta(c)} \neq \lambda$.

Simulationsschleife

Ziel: Simuliere $c \vdash_M c'$ by $(A^{\beta(c)}B, p) \stackrel{*}{\Rightarrow} (A^{\beta(c')}B, p')$;
der Turingmaschinenzustand q von c ist in der Markierung p gespeichert.
Implementation:

1. gehe über eine nichtdeterministisch gewählte Zahl codifizierter Buchstaben;
2. überprüfe, dass das Teilwort aq an der gegenwärtigen Position steht;
3. ersetze aq gemäß δ;
4. kehre zur Standardpräsentation zurück.

Skipping a symbol

$((\mathsf{skip}, q, i, 1):$	$A \to C,$	$\{(\mathsf{skip}, q, i, 2)\},$	$\emptyset)$
$((\mathsf{skip}, q, i, 2):$	$A \to A,$	$\{(\mathsf{skip}, q, i, 3)\},$	$\emptyset)$
$((\mathsf{skip}, q, i, 3):$	$C \to A,$	$\{(\mathsf{skip}, q, i, 4)\},$	$\emptyset)$
$((\mathsf{skip}, q, i, 4):$	$B \to C^2,$	$\{(\mathsf{skip}, q, i, 4)\},$	$\{(\mathsf{skip}, q, i, 5)\})$
$((\mathsf{skip}, q, i, 5):$	$C \to B,$	$\{(\mathsf{skip}, q, i, 5)\},$	$\{(\mathsf{skip}, q, i, 6)\})$
$((\mathsf{skip}, q, i, 6):$	$A \to C,$	$\{(\mathsf{skip}, q, i, 7)\},$	$\{(\mathsf{skip}, q, i, 11)\})$
$((\mathsf{skip}, q, i, 7):$	$B \to B,$	$\{(\mathsf{skip}, q, i, 8)\},$	$\emptyset)$
$((\mathsf{skip}, q, i, 8):$	$C \to \lambda,$	$\{(\mathsf{skip}, q, i, 9)\},$	$\emptyset)$
$((\mathsf{skip}, q, i, 9):$	$A \to C,$	$\{(\mathsf{skip}, q, i, 6)\},$	$\{(\mathsf{skip}, q, i, 10)\})$
$((\mathsf{skip}, q, i, 10):$	$B \to B^2,$	$\{(\mathsf{skip}, q, i, 11)\},$	$\emptyset)$
$((\mathsf{skip}, q, i, 11):$	$C \to A,$	$\{(\mathsf{skip}, q, i, 11)\},$	$\mathsf{exit\text{-}skip}(i))$

Hier ist $\mathsf{exit\text{-}skip}(i)$ gleich $\{(\mathsf{skip}, q, i+1, 1)\}$ falls $i < \gamma$, aber andernfalls gleich $\mathsf{simstart}(q)$.

Simulation von $r = (q, a, X, q', a') \in \delta$, $a \in \tilde{\Gamma}$ und $X = L$.
Betrachte $\beta(aq) = \beta_1 \dots \beta_{2\gamma}$ mit $\beta_i \in \{0,1\}$ und
$\beta(q'a') = \beta'_1 \dots \beta'_{2\gamma}$ mit $\beta'_i \in \{0,1\}$.
Die Prüf-Phase:

$((\text{sim-1}, r, i, 1):$	$A \to C,$	$\{(\text{sim-1}, r, i, 2)\},$	$\emptyset)$
$((\text{sim-1}, r, i, 2):$	$A \to A,$	$\{(\text{sim-1}, r, i, 3)\},$	$\emptyset)$
$((\text{sim-1}, r, i, 3):$	$C \to A,$	$\{(\text{sim-1}, r, i, 4)\},$	$\emptyset)$
$((\text{sim-1}, r, i, 4):$	$A \to C,$	$\{(\text{sim-1}, r, i, 5)\},$	$f_{0,i}(\beta_{2\gamma-i+1}))$
$((\text{sim-1}, r, i, 5):$	$B \to B,$	$\{(\text{sim-1}, r, i, 6)\},$	$\emptyset)$
$((\text{sim-1}, r, i, 6):$	$C \to \lambda,$	$\{(\text{sim-1}, r, i, 7)\},$	$\emptyset)$
$((\text{sim-1}, r, i, 7):$	$A \to C,$	$\{(\text{sim-1}, r, i, 4)\},$	$f_{1,i}(\beta_{2\gamma-i+1}))$
$((\text{sim-1}, r, i, 8):$	$C \to A,$	$\{(\text{sim-1}, r, i, 8)\},$	$\text{cont-sim-1}(i))$

Hierbei sei für $b \in \{0,1\}$

$$f_{j,i}(b) = \begin{cases} \emptyset, & \text{if } j \neq b; \\ \{(\text{sim-1}, r, i, 8)\}, & \text{if } j = b. \end{cases}$$

Die Erzeugungs-Phase:

$((\text{sim-1}, r, i, 9):$	$B \to C^2,$	$\{(\text{sim-1}, r, i, 9)\},$	$\{(\text{sim-1}, r, i, 10)\})$
$((\text{sim-1}, r, i, 10):$	$C \to B,$	$\{(\text{sim-1}, r, i, 10)\},$	$f_i'(\beta_i'))$
$((\text{sim-1}, r, i, 11):$	$B \to B^2,$	$\text{exit-sim-1}(i),$	$\emptyset)$

Hierbei ist:

$$f_i'(b) = \begin{cases} \text{exit-sim-1}(i), & \text{falls } b = 0; \\ \{(\text{sim-1}, r, i, 11)\}, & \text{falls } b = 1; \end{cases}$$

und $\text{exit-sim-1}(i)$ ist gleich $\{(\text{sim-1}, r, i+1, 9)\}$, falls $i < 2\gamma$, andernfalls aber $\{(\text{return}, q', 1)\}$. In jedem Fall gilt $1 \leq i \leq 2\gamma$ sowie $1 \leq j \leq 11$.

Zurück zur Standardpräsentation

$((\text{return}, q, 1) :$	$B \to C,$	$\{(\text{return}, q, 2)\},$	$\emptyset)$
$((\text{return}, q, 2) :$	$B \to B,$	$\{(\text{return}, q, 3)\},$	$\{(\text{return}, q, 10)\})$
$((\text{return}, q, 3) :$	$C \to B,$	$\{(\text{return}, q, 4)\},$	$\emptyset)$
$((\text{return}, q, 4) :$	$A \to C^2,$	$\{(\text{return}, q, 4)\},$	$\{(\text{return}, q, 5)\})$
$((\text{return}, q, 5) :$	$C \to A,$	$\{(\text{return}, q, 5)\},$	$\{(\text{return}, q, 6)\})$
$((\text{return}, q, 6) :$	$B \to \lambda,$	$\{(\text{return}, q, 7)\},$	$\{(\text{return}, q, 9)\})$
$((\text{return}, q, 7) :$	$B \to C,$	$\{(\text{return}, q, 6)\},$	$\{(\text{return}, q, 8)\})$
$((\text{return}, q, 8) :$	$A \to A^2,$	$\{(\text{return}, q, 9)\},$	$\emptyset)$
$((\text{return}, q, 9) :$	$C \to B,$	$\{(\text{return}, q, 9)\},$	$\{(\text{return}, q, 1)\})$
$((\text{return}, q, 10) :$	$C \to B,$	$\text{simstart}(q),$	$\emptyset)$

Ein paar Worte zur Terminierung

- Wann darf eine Simulation terminieren ?

- Wie werden unbeabsichtigte Ableitungen vermieden ?
  Insbesondere gefährlich sind “Quereinstiege”, d.h., Möglichkeiten, falsch zu beginnen (jeder Knoten ist Startknoten).
  Unangenehm sind auch evtl. frühzeitige Abbrüche (jeder Knoten ist final).

Ein paar Worte zur Terminierung aus der Originalarbeit

Termination rules

Firstly, we check in some preparatory steps whether there is at least one A and exactly one B in the string. Then, we continue checking for the occurrence of $\#_R$ at the rightmost position of the simulated Turing tape.

$((\text{term}, \#_R, 0, 0):$	$A \to A,$	$\{(\text{term}, \#_R, 0, 1)\},$	$\emptyset)$
$((\text{term}, \#_R, 0, 1):$	$B \to C,$	$\{(\text{term}, \#_R, 0, 2)\},$	$\emptyset)$
$((\text{term}, \#_R, 0, 2):$	$B \to B,$	$\emptyset,$	$\{(\text{term}, \#_R, 0, 3)\})$
$((\text{term}, \#_R, 0, 3):$	$C \to B,$	$\{(\text{term}, \#_R, 1, 1)\},$	$\emptyset)$

Now, let $\hat{\Sigma} = \Sigma \cup \{\#_L, \#_R, \#, q_f\}$ be the set of symbols admissible in a configuration whose tape contains a terminal string. In addition, for $a \in \hat{\Sigma} \setminus \{\#_L\}$ with $\beta(a) = \beta_1 \dots \beta_\gamma$, $\beta_i \in \{0, 1\}$, and for $1 \leq i < \gamma$, we have:

$$
\begin{array}{|llll|}
\hline
((\mathsf{term}, a, i, 1): & A \to C, & \{(\mathsf{term}, a, i, 2)\}, & f_{0,i}(\beta_{\gamma-i+1})) \\
((\mathsf{term}, a, i, 2): & B \to B, & \{(\mathsf{term}, a, i, 3)\}, & \emptyset) \\
((\mathsf{term}, a, i, 3): & C \to \lambda, & \{(\mathsf{term}, a, i, 4)\}, & \emptyset) \\
((\mathsf{term}, a, i, 4): & A \to C, & \{(\mathsf{term}, a, i, 1)\}, & f_{1,i}(\beta_{\gamma-i+1})) \\
((\mathsf{term}, a, i, 5): & C \to A, & \{(\mathsf{term}, a, i, 5)\}, & \{(\mathsf{term}, a, i+1, 1)\}) \\
\hline
\end{array}
$$

Here, for $b \in \{0, 1\}$,

$$
f_{j,i}(b) = \begin{cases} \emptyset, & \text{if } j \neq b; \\ \{(\mathsf{term}, a, i, 5)\}, & \text{if } j = b. \end{cases}
$$

Similarly, the first bit is finally checked:

$$
\begin{array}{|llll|}
\hline
((\mathsf{term}, a, \gamma, 1): & A \to C, & \{(\mathsf{term}, a, \gamma, 2)\}, & f_{0,i}(\beta_1)) \\
((\mathsf{term}, a, \gamma, 2): & B \to B, & \{(\mathsf{term}, a, \gamma, 3)\}, & \emptyset) \\
((\mathsf{term}, a, \gamma, 3): & C \to \lambda, & \{(\mathsf{term}, a, \gamma, 4)\}, & \emptyset) \\
((\mathsf{term}, a, \gamma, 4): & A \to C, & \{(\mathsf{term}, a, \gamma, 1)\}, & f_{1,i}(\beta_1)) \\
\hline
\end{array}
$$

Then, different things may happen, depending on which tape symbol has been currently read:

$((\mathsf{term}, \#_R, \gamma, 5):$	$C \to A,$	$\{(\mathsf{term}, \#_R, \gamma, 5)\},$	$\{(\mathsf{term}, q_f, 1, 1)\})$
$((\mathsf{term}, q_f, \gamma, 5):$	$C \to A,$	$\{(\mathsf{term}, q_f, \gamma, 5)\},$	$T(\{\#_R\}))$
$((\mathsf{term}, \#, \gamma, 5):$	$C \to A,$	$\{(\mathsf{term}, \#, \gamma, 5)\},$	$T(\{\#_R\}))$
$((\mathsf{term}, \tilde{a}, \gamma, 5):$	$C \to A,$	$\{(\mathsf{term}, \tilde{a}, \gamma, 5)\},$	$\{(\mathsf{term}, \tilde{a}, \gamma, 6)\})$
$((\mathsf{term}, \tilde{a}, \gamma, 6):$	$B \to \tilde{a}B,$	$T(\{\#, \#_R\}),$	$\emptyset)$

where $\tilde{a} \in \Sigma$ and $T(X) = \{(\mathsf{term}, a, 1, 1) \mid a \in \hat{\Sigma} \setminus X\}$ for $X \subset \hat{\Sigma}$.

Finally, we check the codification of the leftmost tape symbol, i.e., $\#_L$, and yield the terminal string if everything was all right up to now.

$((\mathsf{term}, \#_L, 1, 1):$	$A \to C,$	$\{(\mathsf{term}, \#_L, 1, 2)\},$	$\emptyset)$
$((\mathsf{term}, \#_L, 1, 2):$	$B \to \lambda,$	$\{(\mathsf{term}, \#_L, 1, 3)\},$	$\emptyset)$
$((\mathsf{term}, \#_L, 1, 3):$	$C \to \lambda,$	$\{(\mathsf{term}, \#_L, 1, 4)\},$	$\emptyset)$
$((\mathsf{term}, \#_L, 1, 4):$	$A \to A,$	$\emptyset,$	$\{(\mathsf{term}, \#_L, 1, 1)\})$

Ein Beispiel: Angenommen, $\#_L q_0 \#_R \vdash \#_L \#_R q_0 \vdash \#_L q_f a \#_R \vdash \#_L a q_f \#_R$ ist ein terminierender Lauf einer vorgegebenen Turingmaschine. Ferner nehmen wir eine 3-Bit-Codierung an:

$$\begin{array}{lll} \beta(\#) = 000 & \beta(\#_L) = 001 & \beta(\#_R) = 100 \\ \beta(a) = 010 & \beta(q_0) = 011 & \beta(q_f) = 101 \end{array}$$

Unter Verwendung von Binärzahlen als Exponenten liefert die simulierende Grammatik:

$$\begin{array}{lll} A & \Rightarrow & A^{1;011;100}B \qquad \text{(denn } \beta(\#_L q_0 \#_R) = 001011100) \\ & \stackrel{*}{\Rightarrow} & A^{1;100;011}B \qquad \text{(simul. Rechtsbewegung, } \#_L \text{ abtastend)} \\ & \stackrel{*}{\Rightarrow} & A^{1;101;010;100}B \qquad \text{(simul. Linksbewegung, } \#_R \text{ abtastend)} \\ & \stackrel{*}{\Rightarrow} & A^{1;101;010}B^{1;001} \qquad \text{(mit skip)} \\ & \stackrel{*}{\Rightarrow} & A^{1;010;101}B^{1;001} \qquad \text{(simul. Rechtsbewegung, } \#_L \text{ abtastend)} \\ & \stackrel{*}{\Rightarrow} & A^{1;010;101;100}B \qquad \text{(Rückkehr zur Standardcodierung)} \\ & \stackrel{*}{\Rightarrow} & A^{1;010;101}B \qquad \text{(Benutze term für } \#_R) \\ & \stackrel{*}{\Rightarrow} & A^{1;010}B \qquad \text{(Benutze term für } q_f) \\ & \stackrel{*}{\Rightarrow} & A^{1}aB \qquad \text{(Benutze term für } a\text{, d.h., Umrechnen in Terminalsymbole)} \\ & \stackrel{*}{\Rightarrow} & a \qquad \text{(Benutze term für } \#_L) \end{array}$$

Eine Folgerung für Matrixgrammatiken

Folgerung 5 $\mathcal{M}_4 = \mathcal{L}_0$.

Beweisidee: Codiere “Zustände” unär (!) mit einem neuen NT.

Ungewöhnliches zum unbedingten Übergang

Satz 6 *Sei* $L \in \mathcal{L}_0$, $L \subseteq V_T^*$. *Dann gilt:*
Es gibt eine (P,CF,ut) Grammatik $\tilde{G}$, *sodass:*
$w \in L$ *gdw.* $w\$\# \in L(\tilde{G})$ *mit* $\{\$, \#\} \cap V_T = \emptyset$.

Simulationsidee: Gegeben (P,CF,ac) Grammatik $G = (V_N, V_T, P, S)$. führe ein:
(1) Eine Initialisierungsregel $(\text{init} : \tilde{S} \to S\$\sigma, \{p_i^+, p_i^- \mid A_i = S\})$,
(2) Terminierungsregeln $(t_i : B_i \to F, \{t_{i+1}\})$ für $1 \leq i \leq n$ sowie $(t_{n+1} : \sigma \to \#, \{t_{n+1}\})$;
(3) für jede Regel p_i, $1 \leq i \leq m$, simulierende Regeln

$$\begin{array}{lcll} (p_i^- & : & A_i \to F, \{\, q^+, q^- \mid q \in \phi_i \,\}) & , \\ (p_i^+ & : & A_i \to w_i\sigma, \{p_i'\}) & \text{und} \\ (p_i' & : & \sigma \to \lambda, \{q^+, q^- \mid q \in \sigma_i\} \cup \{t_1\}) & . \end{array}$$

□

Folgerungen

- $\mathcal{L}$(P,CF,ut) enthält nicht-rekursive Sprachen.
- $\mathcal{L}$(P,CF$-\lambda$,ut) $\subsetneq$ $\mathcal{L}$(P,CF,ut).
- $\mathcal{L}$(P,CF,ut) $=$ $\mathcal{L}_0$ gdw. $\mathcal{L}$(P,CF,ut) ist abgeschlossen unter Rechtsderivaten. Gölte dieser Abschluss, so wäre er nicht-konstruktiv.

Bemerkung: Das Leerheitsproblem ist entscheidbar für (P,CF,ut) Grammatiken (Rosenkrantz), wie wir unten noch ausführen werden.

Der nicht-löschende Fall lässt sich ähnlich behandeln und liefert:

Satz 7 *Sei* $L \in \mathcal{L}(P,CF{-}\lambda,ac)$, $L \subseteq V_T^*$. *Dann gilt:*
Es gibt eine Konstante $m > 0$ *und (P,CF–*λ*,ut) Grammatik* $\tilde{G}$, *sodass:*
$w \in L$ *gdw.* $w\$\#^m \in L(\tilde{G})$ *mit* $\{\$, \#\} \cap V_T = \emptyset$.

Übung 7 *Beweisen Sie diesen Satz!*

Folgerung 8 $\mathcal{L}(P,CF{-}\lambda,ut) = \mathcal{L}(P,CF{-}\lambda,ac)$ *gdw.*
$\mathcal{L}(P,CF{-}\lambda,ut)$ *ist abgeschlossen unter Rechtsquotient mit regulären Mengen.*

Unbedingter Übergang: Charakterisierungen I

Satz 9 $\mathcal{L}(P,CF[-\lambda],ut)$ *ist gekennzeichnet durch:*

- $\mathcal{L}(M,CF[-\lambda],ut)$,
- $\mathcal{L}(rC,CF[-\lambda],ut)$,
- $\mathcal{L}(TV_c,CF[-\lambda],ut)$,
- $\mathcal{L}(G_c,CF[-\lambda],ut)$.

Hinweis: Lediglich der TV-Fall bereitet Mühe beim Beweis.

Unbedingter Übergang: Charakterisierungen II

Satz 10 $\mathcal{L}(P,CF[-\lambda],ac)$ *ist gekennzeichnet durch:*

- $\mathcal{L}(MS,CF[-\lambda],ut)$,
- $\mathcal{L}(rSC,CF[-\lambda],ut)$,
- $\mathcal{L}(TV,CF[-\lambda],ut)$,
- $\mathcal{L}(G,CF[-\lambda],ut)$.

Beweise gefällig? B.w. für die kompliziertesten Fälle.

Beweis Teil I $\mathcal{L}$(P,CF,ac) $\subseteq$ $\mathcal{L}$(G,CF,ut):

Betrachte $L \in \mathcal{L}$(P,CF,ac), $L \subseteq V_T^*$,
$L = \bigcup_{a \in V_T}(\{a\}V_T^+ \cap L) \cup ((V_T \cup \{\lambda\}) \cap L)$.
Wir konstruieren eine (G,CF,ut) Grammatik $G' = (V_N \cup \{\#, F, S'\}, V_T, P', S', \Gamma, \Sigma, \Phi, h)$ mit
$\Gamma = (U, E)$ sosdass $L(G') = \{a\}V_T^+ \cap L$.
Leicht zu sehen: $\mathcal{L}$(G,CF,ut) ist abgeschlossen unter Vereinigung.

$G = (V_N, V_T, P, S)$ sei eine (P,CF,ac) Grammatik für $\{ w \in V_T^+ \mid aw \in L \}$.

Eine Regel $(p_i : A_i \to w_i, \sigma_i, \phi_i)$ von G bestimmt
zwei Knoten F_i^+, F_i^- mit $h(F_i^+) = \{A_i \to w_i, \# \to F\}$,
$E \cap \{F_i^+\} \times \{g, r\} \times U = \{F_i^+\} \times \{g, r\} \times (\{F_j^{+/-} \mid p_j \in \sigma_i\} \cup \{t\})$,
$h(F_i^-) = \{A_i \to F\}$,
$E \cap \{F_i^-\} \times \{g, r\} \times U = \{F_i^-\} \times \{g, r\} \times \{F_j^{+/-} \mid p_j \in \phi_i\}$.

Der Anfangsknoten I enthält die Regel $S' \to \#S$ mit sowohl grünen als auch roten Bögen nach $\{F_i^+, F_i^- \mid (p_i : S \to w_i, \sigma_i, \phi_i) \in P\}$.

Zielknoten T enthält die Regel $\# \to a$ mit grünen und roten Schlingen.

Beweis Teil II $\mathcal{L}$(G,CF,ut) $\subseteq \mathcal{L}$(TV,CF,ut):

Sei $G = (V_N, V_T, P, S, \Gamma, \Sigma, \Phi, h)$ eine (G,CF,ut) Grammatik mit $\Gamma = (U, E)$;
$U = \{u_1, \ldots, u_n\}$, $u_i = \{A_{i1} \to w_{i1}, \ldots, A_{ir_i} \to w_{ir_i}\}$.

Wir geben eine simulierende (TV,CF,ut)-Grammatik
$(V_N \cup \{X^{(i)} \mid X \in V_N, 1 \leq i \leq n\} \cup \{S', F\} \cup U, P', S', \phi)$ an, wobei ϕ eine Funktion mit Periode $2n+3$ ist: Für $i = 1, \ldots, n$ sei

$$\begin{aligned}
\phi(i) &= \{A_{i\rho} \to A_{i\rho}^{(i)} \mid 1 \leq \rho \leq r_i\} \cup \{u_j \to u_j \mid j \neq i\},\\
\phi(n+i) &= \{A_{j\rho}^{(j)} \to F \mid j \neq i, 1 \leq \rho \leq r_k\} \cup \{u_j \to u_j \mid j \neq i\};\\
\phi(2n+1) &= \{A_{i\rho}^{(i)} \to w_{i\rho} \mid 1 \leq i \leq n, 1 \leq \rho \leq r_i\},\\
\phi(2n+2) &= \{S' \to S u_i \mid u_i \in \Sigma\} \cup \{u_i \to u_j \mid (u_i, g, u_j) \in E\},\\
\phi(2n+3) &= \{u_i \to u_i \mid u_i \in U\} \cup \{u_i \to \lambda \mid u_i \in \Phi\}.
\end{aligned}$$

Kommentare zum Beweis:

Angenommen, die Simulation “befindet sich” in Knoten U_i.
$\phi(i)$ dient zum Markieren einer linken Seite einer Regel von Knoten U_i.
So eine Markierung erfolgt genau dann, wenn eine linke Seite einer Regel aus $h(U_i)$ in der augenblicklichen Satzform vorhanden ist.
$U_j \to U_j$ gestattet das “Überspringen” so einer Markiererei.
$\phi(n+i)$ überprüft, ob nicht mehr als ein Zeichen markiert wurde.
Dann würde diese Regelmenge nämlich gerade nicht übersprungen werden können.
$\phi(2n+1)$ vollzieht den eigentlichen Ableitungsschritt.
$\phi(2n+2)$ geht zum nächsten Zustand über und erledigt auch den Anfang.
$\phi(2n+3)$ gestattet die Terminierung einer Satzform.
Wurde fehlerhafterweise einer nicht-terminalen Satzform die Knoteninformation gelöscht, so würden im nächsten Durchlauf in den ersten n Schritten etliche Nichtterminale markiert werden; selbst wenn nur eines markiert wird, wird in den darauf folgenden n Schritten ein Fehlersymbol eingeführt.

Aufgrund der bekannten Abschlusseigenschaften von $\mathcal{L}(\mathrm{G}, \mathrm{CF}{-}\lambda, \mathrm{ut})$ unter Derivaten and Konkatenation, sowie von $\mathcal{L}(\mathrm{TV}, \mathrm{CF}{-}\lambda, \mathrm{ut})$ unter Vereingung, folgt auch der λ-freie Fall sofort.

Higmans Lemma wurde 1952 wohl erstmals von Higman publiziert, gehört aber vermutlich zu den am häufigsten (wiederholt und vermutlich unabhängig) gefundenen und publizierten Sätzen der Mathematik überhaupt.

Für $u, v \in \Sigma^*$, $u = u_0 u_1 \dots u_n$ definiere: u *teilt* v, kurz $u \mid v$,
gdw. $v \in \Sigma^* u_0 \Sigma^* u_1 \Sigma^* \dots \Sigma^* u_n \Sigma^*$.

Lemma 11 $u|v$ *beschreibt eine Halbordnung auf* Σ*, die sog. Divisionsordnung.*

Satz 12 (Higmans Lemma) *Es sei* Σ *ein endliches Alphabet. Jede Sprache* $L \subseteq \Sigma^*$ *enthält eine endliche Teilsprache* L'*, sodass jedes Wort in* L *einen Teiler in* L' *besitzt.*

Mehr zu Higmans Lemma

Die im Lemma beschriebene Menge L' heißt auch *Higman-Menge* für L.

Beweise für den Satz von Higman sind nicht-konstruktiv. Ist L durch TM gegeben, so benötigt eine Turingmaschine zur Berechnung von L' das Halteproblem als Orakel.

Wir werden hier auf einen Beweis verzichten, empfehlen aber, selbst mal nach einem zu suchen.
Dabei werden Sie auch herausfinden, weshalb die Endlichkeit des Grundalphabets wesentlich ist.

Satz 13 *Sei $\mathcal{G}$ eine Grammatikfamilie mit einem entscheidbaren Leerheitsproblem, welche effektiv gegen Durchschnitt mit regulären Sprachen abgeschlossen sei. Dann gibt es einen Algorithmus, der zu jeder gegebenen Grammatik $G \in \mathcal{G}$ eine Higman-Menge $L' \subseteq L(G)$ konstruiert.*

Beweis. Betrachte $G \in \mathcal{G}$ mit $L(G) \subseteq \Sigma^*$.
Wegen der beiden Voraussetzungen an $\mathcal{G}$ ist $L(G)$ rekursiv. (*)
Für $w \in \Sigma^*$, $I(w) := \{u \in \Sigma^* \mid w \mid u\}$ ist regulär.
$\rightsquigarrow I(\{w_1, \ldots, w_n\}) := I(w_1) \cup \cdots \cup I(w_n)$ ist regulär.
L' ist Higman-Menge gdw. $L(G) \subseteq I(L')$ gdw. $L(G) \cap (\Sigma^* \setminus I(L')) = \emptyset$.
Die letztere Eigenschaft kann algorithmisch überprüft werden, denn $\Sigma^* \setminus I(L')$ ist regulär.
Nimm irgend eine Aufzählung von $L(G)$ (ex. wegen (*)), z.B. $w_0, w_1, \ldots$, und berechne

$L' := \{w_0\}$; $i := 1$;
`while` $L(G) \cap (\Sigma^* \setminus I(L')) \neq \emptyset$ `do begin`
    $L' := L' \cup \{w_i\}$; $i := i + 1$
`end`

Das Lemma von Higman sichert die Terminierung der Prozedur. □

Manchmal ist Higman leicht—ein Exkurs

Eine Higman-Menge kann recht leicht von gewissen Sprachbeschreibungen berechnet werden. Ausgehend von regulären Ausdrücken, d.h.

$$R \in \mathcal{R}(\Sigma) \subseteq (\Sigma \cup \{(,), \cup, *\})^*,$$

kann man folgenden Algorithmus anwenden:

- $L'(R_1) = \{R_1\}$ if $R_1 \in \Sigma^* \subset \mathcal{R}(\Sigma)$.
- If $R_1, R_2 \in \mathcal{R}(\Sigma)$, then
  - $L'((R_1 R_2)) = L'(R_1)L'(R_2)$,
  - $L'((R_1 \cup R_2)) = L'(R_1) \cup L'(R_2)$, and
  - $L'(R_1*) = \{\lambda\}$. $\longleftarrow$!

Linksableitungen gestatten verschiedene Definitionen.

(i) Ersetze stets das linkeste Vorkommen irgendeines Nichtterminals.

(ii) In graphgesteuerten Grammatiken gibt es eine Teilmenge von Nichtterminalen, die in einem Schritt ersetzt werden können; ersetze aus dieser Menge das linkeste Vorkommen.*

(iii) Ersetze bei der gerade anzuwendenen Regel das linkeste Vorkommen ihrer linken Seite.

Übung 8 *Geben Sie die genannten Modi formal an.*

* In der Literatur wird hiervon manchmal subtil abgewichen.

Linksableitungen vom Typ 1

Satz 14 *Selbst für ganz allgemeine G Grammatiken gilt:*
$\mathcal{L}(G, \mathit{CF}, \mathit{ac}, \mathit{left\text{-}1}) = \mathcal{L}(\mathit{CF})$.

Beweis. Sei $G = (V_N, V_T, P, S, \Gamma = (U, E), \Sigma, \Phi, h)$ eine G Grammatik.
Idee: Variante der Tripel-Konstruktion.

$$G' = (V'_N = U \times V_N \times U \cup \{S_0\}, V_T, P', S_0)$$

— $S_0 \to (u, S, u')$, falls $u \in \Sigma, u' \in \Phi$.
Betrachte $A \to w \in h(u)$.
— Falls $(u, r, u'') \in E$ und B ist keine linke Seite irgend einer Regel in u, so sei $(u, B, u') \to (u'', B, u') \in P'$;
— Falls $w \in V_T^*$ und $(u, g, u') \in E$, so füge $(u, A, u') \to w$ in P'.
— Sei $w = x_0 B_1 x_1 B_2 \cdots x_{r-1} B_r x_r$ mit $x_i \in V_T^*$ und $B_i \in V_N$.
Füge $(u_0, A, u_r) \to x_0(\hat{u}_0, B_1, u_1)x_1(u_1, B_2, u_2) \cdots x_{r-1}(u_{r-1}, B_r, u_r)x_r$
zu P' wobei alle $u_i \in U$ und $(u_0, g, \hat{u}_0) \in E$. □

Linksableitungen vom Typ 2

Satz 15 *Selbst für ganz allgemeine G Grammatiken gilt:*
$\mathcal{L}(\mathrm{G}, CF, ac, \textit{left-2}) = \mathcal{L}_0$.
$\mathcal{L}(\mathrm{G}, CF - \lambda, ac, \textit{left-2}) = \mathcal{L}_1$.

Entsprechendes gilt für P und M Grammatiken nicht gemäß unserer Definition.

Übung 9 *Beweisen Sie die Behauptungen durch Angabe geeigneter Konstruktionen.*

Linksableitungen vom Typ 3

Diese sind eine ganz eigene Welt, die sich ähnlich verhält wie “freie Ableitungen” (der “normale” Fall).

Benauer gilt:

Satz 16 *Selbst für ganz allgemeine G Grammatiken gilt:*
Seien $X \in \{$ *G, TV, MS, rSC, P, M, rC* $\}$, $Y \in \{$ *CF, CF–*$\lambda\}$, $Z \in \{\lambda,$ *ac*, *ut*$\}$.
$\mathcal{L}($*P*, Y, Z[, *left-3*]$) = \mathcal{L}($X, Y, Z[, *left-3*]$)$.

Für P und M Grammatiken fällt nach unserer Def. der Links-2 und der Links-3-Fall zusammen.

Hierarchische Beziehungen

Satz 17 (i) $\mathcal{L}(\mathrm{P},\mathrm{CF}) \subsetneq \mathcal{L}(\mathrm{P},\mathrm{CF},\text{left-3}) \subsetneq \mathcal{L}(\mathrm{P},\mathrm{CF},\mathrm{ac}[,\text{left-3}]) = \mathcal{L}(\mathrm{P},\mathrm{CF},\mathrm{ut},\text{left-3}) = \mathcal{L}_0$.

(ii) $\mathcal{L}(\mathrm{P},\mathrm{CF}-\lambda) \subsetneq \mathcal{L}(\mathrm{P},\mathrm{CF}-\lambda,\text{left-3}) \subsetneq \mathcal{L}(\mathrm{P},\mathrm{CF}-\lambda,\mathrm{ac},\text{left-3}) \subsetneq \mathcal{L}_1$.

(iii) $\mathcal{L}(\mathrm{P},\mathrm{CF}-\lambda,\mathrm{ut},\text{left-3}) \subsetneq \mathcal{L}(\mathrm{P},\mathrm{CF}-\lambda,\mathrm{ac},\text{left-3})$

(iv) **Offene Frage:** Ist irgend eine der folgenden Einschlüsse echt ?

- $\mathcal{L}(\mathrm{P},\mathrm{CF}-\lambda,\mathrm{ac}) \subseteq \mathcal{L}(\mathrm{P},\mathrm{CF}-\lambda,\mathrm{ac},\text{left-3})$
- $\mathcal{L}(\mathrm{P},\mathrm{CF}[-\lambda],\mathrm{ut}) \subseteq \mathcal{L}(\mathrm{P},\mathrm{CF}[-\lambda],\mathrm{ac}[,\text{left-3}])$

UT kann manchmal alles !

Satz 18 $\mathcal{L}_0 = \mathcal{L}(P, CF, ut, \textit{left-3})$.

Verallgemeinerung von $\mathcal{L}_0 = \mathcal{L}(\mathrm{P}, \mathrm{CF}, \mathrm{ac}, \text{left-3})$, was zuerst Rosenkrantz bewies und wir mit der NT-Komplexitätsbetrachtung verfeinert gesehen haben.

Beweis. Sei $\mathrm{L} \subseteq \Sigma^*$ aufzählbar mit Higman-Menge L'. D.h., $\mathrm{L} = \bigcup_{w \in \mathrm{L}'} (\mathrm{L} \cap \mathrm{I}(w))$.
$\mathrm{L}(w) := \mathrm{L} \cap \mathrm{I}(w) \in \mathcal{L}(\mathrm{P}, \mathrm{CF}, \mathrm{ac}, \text{left-3})$.

Die übliche Simulation von einem Typ-0-Mechanismus benutzt eine spezielle Codierung, welche am Schluss erst in ein Terminalwort überführt wird. Genau an dieser Stelle greifen wir nun ein: Da wir wissen, dass w jedes Wort von $\mathrm{L}(w)$ teilt, können wir die Konstruktion abändern:
Wir starten nun mit $S \Rightarrow S' S_0 w_1 S_1 \cdots w_n S_n$, wobei S' die schon bekannte Simulation anstößt.
Die Terminierungsroutine wird verändert, sodass die S_i dazu verwendet werden die gewünschten Teile des Terminalwortes aufzufüllen. □

More details of the proof (aus der Originalarbeit)

So, we have a (P,CF,ac,left-3) grammar $G = (V_N, \Sigma, P, S)$ for $L(w)$.
$S \to x$ with $x = S'S_0w_1S_1 \cdots w_nS_n$ is the start rule.
Let $P = \{p_1, \ldots, p_m\}$, and $(p_\mu : A_\mu \to \nu_\mu, \sigma(p_\mu), \phi(p_\mu))$.
We give a (P,CF,ut,left-3) grammar $\tilde{G} = (\tilde{V}_N, \Sigma, \tilde{P}, \tilde{S})$ with $L(\tilde{G}) = L(w)$.
Since $\mathcal{L}(P, CF, ut, left\text{-}3)$ is easily seen to be closed under finite union, this shows that $L \in \mathcal{L}(P, CF, ut, left\text{-}3)$.

If V is an alphabet, $\overline{V} = \{\overline{a} \mid a \in V\}$ is the set of barred symbols. Consider the morphisms $g : V_G^* \to \overline{V_G}^*$ given by $A \mapsto \overline{A}$, if $A \in V_G$, $g' : V_G^* \to (\overline{V_N} \cup \Sigma)^*$ defined by $A \mapsto \overline{A}$, if $A \in V_N$, and $a \mapsto a$, if $a \in \Sigma$, and $h : V_G^* \to V_N^*$, $A \mapsto A$, if $A \in V_N$, and $a \mapsto \lambda$, if $a \in \Sigma$.

Let $\tilde{V}_N = V_N \cup \overline{V_G} \cup \{\tilde{S}, F, E\} \cup E_\Sigma$, where $V_N = \{B_1, \ldots, B_\ell\}$ and $\Sigma = \{a_1, \ldots, a_r\}$, $E_\Sigma = \{E_\rho \mid 1 \leq \rho \leq r\}$.
$\tilde{P}$ contains

- an initialization rule
  $(\text{init} : \tilde{S} \to h(x)Eg'(x), \{p_\mu^- \mid A_\mu = S\})$
  instead of $S \to x$ where $x = S'S_0w_1S_1 \cdots w_nS_n$,

- simulation rules for $\mu = 1, 2, \ldots, m$

$$
\begin{array}{lcl}
(p_\mu^- & : & h(A_\mu) \to F, \{q^+, q^- \mid q \in \phi(p_\mu)\}), \\
(p_\mu^+ & : & h(A_\mu) \to E, \{p'_\mu\}), \\
(p'_\mu & : & E \to h(\nu_\mu), \{p''_\mu\}), \\
(p''_\mu & : & g(A_\mu) \to g(\nu_\mu), \{q^+, q^- \mid q \in \sigma(p_\mu)\} \cup \{t_1\}).
\end{array}
$$

- termination rules

$$
\begin{array}{lcll}
(t_i & : & B_i \to F, \{t_{i+1}\}) & \text{, for } i = 1, 2, \ldots, \ell - 1, \\
(t_\ell & : & B_\ell \to F, \{t_{\ell+1}\}) & \\
(t_{\ell+1} & : & E \to E_1 \cdots E_r, \{T_1\}) & \text{(i.e., no error in simul.)} \\
(T_\rho & : & g(a_\rho) \to E_\rho, \{T'_\rho\}) & \text{, for } a_\rho \in \Sigma, \\
(T'_\rho & : & E_\rho \to \lambda, \{T''_\rho\}) & \text{(if error, then erase all)} \\
(T''_\rho & : & E_\rho \to E_\rho a_\rho, \{T_\rho, T'''_\rho\}) & \\
(T'''_\rho & : & g(a_\rho) \to F, \{T_{(\rho \bmod r)+1}\}) & .\square
\end{array}
$$

In the successful case, the derivation is simulated as follows: if

$$x = x_0 w_1 x_1 ... w_n x_n$$

is a sentential form derived via grammar G, and rule p_μ is to be applied to x, then

$$h(x) E g(x_0) w_1 g(x_1) \cdots w_n g(x_n)$$

represents x in the simulating grammar $\tilde{G}$, where the simulating grammar has to guess beforehand whether to simulate the negative case via p_μ^- or the positive case via the sequence p_μ^+, p_μ', and p_μ''. Especially, at the end, a terminal string x is represented by

$$E g(x_0) w_1 g(x_1) \cdots w_n g(x_n).$$

Now, all barred terminal symbols are converted into their unbarred counterparts.

If the positive case was entered erroneously during a simulation of rule p_μ, i.e., no occurrence of A_μ is present in the current sentential form, the success witness E is erased. In the termination phase, this would lead to the erasure of all nonterminals without introducing terminals as "compensation". Therefore, the shortest word of $L(w)$, namely w, is derived in this way.

Der Satz von Rosenkrantz (1969)

Satz 19 *Die Präfixeigenschaft ist entscheidbar für programmierte Grammatiken mit unbedingtem Übergang (unter Rechts-3-Ableitung).*

Folgerung 20 *Das Leerheitsproblem ist entscheidbar für programmierte Grammatiken mit unbedingtem Übergang.*

Der Beweis beruht auf folgender Aussage (Lemma von Dickson 1913):

Satz 21 *Jede Menge paarweise unvergleichbarer Vektoren natürlicher Zahlen ist endlich.*

(hier angewendet auf *Parikh-Vektoren* $\Psi(w)$ für die Nichtterminalzeichen).

Der Satz von Rosenkrantz: Der Beweis

Seien Grammatik $G = (V_N, V_T, P, Lab, S)$ und Eingabe x gegeben.
O.E.: S taucht auf keiner rechten Regelseite auf.
Daher gibt es "Anfangsregel" mit Label r_0.
Hieraus wird ein endlicher Graph $\Gamma = (V, E)$ konstruiert:
$V = \{(\phi, \nu, r) \mid \phi \in V_G^{\leq |x|}, \nu \in \mathbb{N}^{|V_N|}, r \in Lab\}$.
potentielle Bögen, die von (ϕ, ν, r) nach (ψ, u, q) führen:
$q \in \sigma(r)$, $\phi\xi$ wird durch r in $\psi\eta$ verwandelt mit $\Psi(\xi) = \nu$ und $\Psi(\eta) = u$.
Anfangsknoten $(S, \vec{0}, r_0)$.
Der Graph wird sukzessive aufgebaut.
Wann immer ein neuer Knoten (ψ, u, q) eingeführt werden sollte, wird erst geschaut, ob bereits ein Knoten (ψ, u', q) existiert mit $u' \leq u$. Wenn ja, wird kein neuer Knoten eingeführt, sondern statt dessen dorthin verbunden.
Nach dem Lemma von Dickson ist der so konstruierte Graph endlich.
Ferner gilt: x ist Präfix eines Wortes aus $L(G)$ $\iff$ Es gibt einen Weg von $(S, \vec{0}, r_0)$ nach $(x, \vec{0}, r)$ für ein Label r.

Grammatiken mit wahlfreiem Kontext

Eine *(kontextfreie) Grammatik mit wahlfreiem Kontext (mit Vorkommenstest)* ist gegeben durch

$$G = (N, \Sigma, P, S),$$

wobei

- N: Nichtterminalalphabet,
- Σ: Terminalalphabet,
- $S \in N$: Startsymbol und
- P: endliche Menge von Regeln der Form

  $$(A \to w, Q, R)$$

  mit
  - $A \to w$ ist kontextfrei, d.h., $A \in N$ und $w \in (N \cup \Sigma)^*$,
  - $Q \subseteq N$ ist der *gestattende Kontext* und
  - $R \subseteq N$ ist der *verbietende Kontext*.

Grammatiken mit wahlfreiem Kontext: Ableitungsbegriff

$x \Rightarrow y$ gdw. $\exists x', x'' \in V_G^* \exists (\alpha \to \beta, Q, R) \in P$:

- $x = x'\alpha x'' \wedge y = x'\beta x''$.
- Alle $B \in Q$ kommen in $x'x''$ vor.
- Kein $B \in R$ kommt in $x'x''$ vor.

Wie üblich: $L(G) = \{w \in \Sigma^* \mid S \stackrel{*}{\Rightarrow} w\}$.

Grammatiken mit wahlfreiem Kontext: Beispiele

$G = (\{A, B, C, S, A', B', C'\}, \{a, b, c\}, P, S)$; P enthält folgende Regeln:

$(S \to ABC, \emptyset, \emptyset)$

$(A \to aA', \{B, C\}, \emptyset)$ $(B \to bB', \{C, A'\}, \emptyset)$ $(C \to cC', \{A', B'\}, \emptyset)$

$(A' \to A, \{B', C'\}, \emptyset)$ $(B' \to B, \{C', A\}, \emptyset)$ $(C' \to C, \{A, B\}, \emptyset)$

$(A \to a, \{B, C\}, \emptyset)$ $(B \to b, \{C\}, \emptyset)$ $(C \to c, \emptyset, \emptyset)$

Welche Sprache $L(G)$ wird beschrieben ? (hierzu Beweisskizze an der Tafel)

Übung 10 *Kann man auf die blauen Einschränkungen verzichten, ohne die Sprache zu verändern ?*

Grammatiken mit wahlfreiem Kontext: Beispiele

$G = (\{A, B, C, S\}, \{a\}, P, S)$; P enthält folgende Regeln:

$(S \to AA, \emptyset, \{B, C\})$ $(A \to B, \emptyset, \{S, C\})$ $(B \to S, \emptyset, \{A, C\})$

$(A \to C, \emptyset, \{S, B\})$ $(C \to a, \emptyset, \{S, A, B\})$

Welche Sprache $L(G)$ wird beschrieben ?

Sprachklassen:

– $\mathcal{L}(\mathcal{RC}, \mathsf{CF}[-\lambda], \mathsf{ac})$ allgemein, mit Vorkommenstest

– $\mathcal{L}(\mathcal{RC}, \mathsf{CF}[-\lambda])$ ohne Vorkommenstest (stets $R = \emptyset$)

– $\mathcal{L}(\mathsf{fRC}, \mathsf{CF}[-\lambda])$ (nur) mit verbietendem Vorkommenstest (stets $Q = \emptyset$)

Grammatiken mit wahlfreiem Kontext: Beschreibungsmächtigkeit

Satz 22 $\mathcal{L}(\mathcal{RC}, \mathit{CF}[-\lambda], \mathit{ac}) = \mathcal{L}(\mathcal{P}, \mathit{CF}[-\lambda], \mathit{ac})$.

Satz 23 $\mathcal{L}(\mathit{CF}) \subsetneq \mathcal{L}(\mathcal{RC}, \mathit{CF}[-\lambda]) \subseteq \mathcal{L}(\mathcal{P}, \mathit{CF}[-\lambda]) \subsetneq \mathcal{L}(\mathcal{P}, \mathit{CF}[-\lambda], \mathit{ac})$.

Satz 24 $\mathcal{L}(\mathit{CF}) \subsetneq \mathcal{L}(\mathit{fRC}, \mathit{CF}[-\lambda]) \subsetneq \mathcal{L}(\mathcal{P}, \mathit{CF}[-\lambda], \mathit{ut}) \subseteq \mathcal{L}(\mathcal{P}, \mathit{CF}[-\lambda], \mathit{ac})$.

Grammatiken mit wahlfreiem Kontext: Beweisideen und Hinweise

Satz 22: $\subseteq$ “sequentielles Überprüfen”; $\supseteq$ explizites Mitführen der Zustandsinformation in der Satzform. Vorkommenstest ist auch notwendig, um (nur) einmaliges Anwenden der kontextfreien Regel zu gewährleisten.

Satz 23: Achtung Echheit der einen Inklusion unbekannt

Satz 24: Abschlusseigenschaften:
Lemma: $\mathcal{L}(\mathsf{fRC}, \mathsf{CF}[-\lambda])$ ist gegen [nicht-löschende] Morphismen und gegen Schnitt mit regulären Sprachen abgeschlossen.
Folgerung: fRC-Grammatiken haben entscheidbares Leerheitsproblem.
$\rightsquigarrow \mathcal{H}(\mathcal{L}(\mathsf{fRC}, \mathsf{CF}[-\lambda]))$ ist entscheidbar im Ggs. zu $\mathcal{H}(\mathcal{L}(\mathcal{P}, \mathsf{CF}[-\lambda], \mathsf{ut}))$.

Geordnete Grammatiken

$G = (N, \Sigma, P, S, \prec)$ mit $\prec$ Halbordnung auf P.

$A \to w$ ist nur dann anwendbar, wenn keine Regel $B \to v$ anwendbar ist mit $(A \to w) \prec (B \to v)$.

Beispiel:

$$\begin{array}{llll} 1 : S \to AB & 2 : A \to aA' & 3 : A \to A'' & 4 : A \to F \\ 5 : B \to bB'c & 6 : B \to bc & 7 : B \to F & 8 : A' \to A \\ 9 : A' \to F & 10 : A'' \to a & 11 : A'' \to F & 12 : B' \to B \\ 13 : B'' \to F & & & \end{array}$$

Halbordnung: $2, 3 \prec 13$; $6, 12 \prec 9$; $8, 10 \prec 7$; $5, 6 \prec 4$; $5 \prec 11$
Beobachte: Auf Satzform $a^r A b^r B c^r$ ist nur $A \to A''$ und $A \to aA'$ erfolgreich anwendbar.

Übung 11 *Geordnete Grammatik für* $\{a^{2^n} \mid n \geq 0\}$.

Geordnete Grammatiken: Eigenschaften

Satz 25 $\mathrm{L} \in \mathcal{L}(\mathit{fRC}, \mathit{CF}[-\lambda])$ *gdw.* $\mathrm{L} = \mathrm{L}(\mathrm{G})$ *für eine geordnete Grammatik* G *[ohne* λ*-Regeln].*

Satz 26 $\mathcal{L}(\mathit{fRC}, \mathit{CF}[-\lambda])$ *ist eine AFL, die sogar voll ist, gestattet man löschende Regeln.*

Übung 12 *Führen Sie die Beweise aus; die folgenden beiden Folien mögen Ihnen eine Anregung liefern.*

Beweis der Trio-Eigenschaft (aus Originalarbeit)

Beweis. We show the closure of ordered languages under 1-a-transducer mappings. So, let $G = (V_N, V_T, P, S, \prec)$ be an ordered grammar and $M = (Q, V_T, Y, H, q_0, q_f)$ be a 1-a-transducer.

We construct an ordered grammar $G' = (V'_N, Y, P', S', \prec')$ such that $L(G') = M(L(G))$. Let

$$\begin{aligned}
V'_N &= Q \times (V_N \cup V_T \cup \{L\}) \times Q \cup \{S'\};\\
P'(A \to w) &= \{(q_1, A, q_{n+1}) \to (q_1, w_1, q_2) \cdots (q_n, w_n, q_{n+1}) \mid \\
&\qquad w = w_1 \cdots w_n, |w| = n \geq 1, w_i \in V_N \cup V_T\};\\
P'(A \to \lambda) &= \{(q, A, q') \to (q, L, q')\} \cup \{(q, L, q) \to \lambda \mid q \in Q\};
\end{aligned}$$

$$
\begin{aligned}
P' &= \{S' \to (q_0, S, q_f)\} \\
&\cup \bigcup_{A \to w \in P} P'(A \to w) \\
&\cup \{(q, B, q'') \to (q, B, q')(q', L, q''), (q, B, q'') \to (q, L, q')(q', B, q'') \mid \\
&\qquad q, q', q'' \in Q, B \in V_T \cup V_N \cup \{L\}\} \\
&\cup \{(q, a, q') \to w \mid (q, a, w, q') \in H, a \in V_T\} \\
&\cup \{(q, L, q') \to w \mid (q, \lambda, w, q') \in H\} \\
&\cup \{X \to X \mid X \in Q \times V_N \times Q\}
\end{aligned}
$$

The order $\prec'$ is defined by
(1) $\forall A' \to w' \in P'(A \to w) \forall B' \to v' \in P'(B \to v)\ A \to w \prec B \to v$ iff $A' \to w' \prec' B' \to v'$ and by
(2) $\forall X \in Q \times V_N \times Q \forall (q, a, q') \in Q \times (V_T \cup \{L\}) \times Q$, if $(q, a, q') \to w \in P'$, then $(q, a, q') \to w \prec' X \to X$.

The stepwise simulation of productions of the form $A \to w$ in the original grammar G via productions from $P'(A \to w)$ is obvious, including the obedience of the order. . . . □

Eine kleine Normalform

Lemma 1 *Zu jeder (generierenden) geordneten Grammatik* G *findet sich eine äquivalente generierende geordnete Grammatik* G' *ohne Regeln* $A \to z$, $A \to z'$, *wobei* $A \to z \succ A \to z'$. □

Beweis. Lasse einfach die kleinere der Regeln fort; sie ist nie anwendbar. □

Sprachgeneratoren und Sprachakzeptoren

Wir hatten bereits eingangs gesehen, dass sich auch *akzeptierende* (z.B.) kontextfreie Grammatiken einführen lassen, im Wesentlichen durch Umkehr der Regeln. Zu einer Grammatik G lässt sich so die *duale Grammatik* G^{d} definieren. Der Deutlichkeit notieren wir Regeln für akzeptierende Grammatiken mit $\rightarrow$ und die zugehörige Ableitungsrelation mit $\overset{*}{\Rightarrow}$.

Leicht einzusehen ist damit:

Satz 27 *Eine Sprache* L *wird genau dann von einer Typ-*i*-Grammatik erzeugt (generiert), wenn sie von einer Typ-*i*-Grammatik akzeptiert wird.*

Sprachgeneratoren und Sprachakzeptoren–mit Steuerung

Für Grammatiken mit wahlfreiem Kontext gilt dasselbe Argument (fast!) wieder:

Satz 28 *Eine Sprache* L *wird genau dann von einer Grammatik* G *mit wahlfreiem Kontext erzeugt, wenn das (geeignet definierte) Dual* G^{d} *von* G *die Sprache* L *akzeptiert.*

Übung 13 *Beweisen Sie die letzte Aussage!*
Wie genau muss G^{d} *definiert sein, damit der Beweis funktioniert?*
Gilt die Aussage auch, wenn nur gestattender oder nur verbietender Kontext zugelassen sind ?

Geordnete Grammatiken—akzeptieren mehr ?!

Satz 29 *Zu jeder geordneten Grammatik* G *mit kontextfreien Kernregeln gibt es eine geordnete Grammatik* G' *mit kontextfreien Kernregeln, sodass* $\mathrm{L}(\mathrm{G}) = \mathrm{L}_{acc}(\mathrm{G}')$.

Beweis. Wegen obigen Lemmas können wir voraussetzen, dass die O-Grammatik $\mathrm{G} = (V_\mathrm{N}, V_\mathrm{T}, \mathrm{P}, S, \prec)$ **keine** Regeln $A \to w$, $A \to w'$ enthält
mit $A \to w \succ A \to w'$.
Zu $A \to w \in \mathrm{P}$ sei $X_{A \to w}$ die Menge von Nichtterminalen A'
mit $A' \to w' \succ A \to w$ für irgendein w'.

Wir konstruieren eine akzeptierende O-Grammatik $\mathrm{G}' = (V'_\mathrm{N}, V_\mathrm{T}, \mathrm{P}', S, \prec')$.
Definiere dafür

$$V'_\mathrm{N} = V_\mathrm{N} \cup \{[w, A] \mid A \to w \in \mathrm{P}\} \cup \{\mathrm{F}\}$$

Die Vereinigung sei dabei disjunkt, F sei ein besonderes Fehlerzeichen.
Für jede Regel $A \to w \in \mathrm{P}$ führen wir folgende Regeln und Relationen in P' ein (dadurch werden P' und $\prec'$ vollständig beschrieben).

- $w \to [w, A] \prec' [w, A] \to \mathrm{F}$;
- $[w, A] \to A$ mit $[w, A] \to A \prec' A' \to \mathrm{F}$ für jedwedes $A' \in X_{A \to w}$;
- Gilt $u \to V \in \mathrm{P}'$ mit $V \neq \mathrm{F}$ und $u \neq [w, A]$, füge die Relation $u \to V \prec' [w, A] \to \mathrm{F}$ hinzu.

Der Nachweis der Richtigkeit der Konstruktion verbleibe als Übungsaufgabe. □

Geordnete Grammatiken—akzeptieren mehr ?!

Satz 30 $\mathcal{L}_{acc}(O,CF{-}\lambda) = \mathcal{L}(CS)$.

Beweis. Die Richtung $\subseteq$ geht durch LBA-Simulation (Übungsaufgabe).

Die Richtung $\supseteq$ ist schwieriger. Sei $L \in \mathcal{L}(CS)$, $L \subseteq V^+$. Offenkundig gilt:

$$L = \bigcup_{a,b \in V} \{a\} d_a^l(d_b^r(L)) \{b\} \cup (L \cap (V \cup V^2)).$$

(d^l bzw. d^r bezeichnen hier Links- bzw. Rechtsableitungen.)
Da offenbar $\mathcal{L}_{acc}(O,CF{-}\lambda)$ gegen Vereinigung abgeschlossen ist (Übungsaufgabe), genügt es im Folgenden zu zeigen, dass $\{a\}M\{b\} \in \mathcal{L}_{acc}(O,CF{-}\lambda)$ für bel. $M \in \mathcal{L}(CS)$, $\lambda \notin M$. Es sei $G = (V_N, V_T, P, S)$ eine kontextsensitive Grammatik (ohne λ-Regeln) in Kuroda Normalform für M.
Sei eine eindeutige Markierung r für jede Regel der Form $XU \to YZ$ vergeben (Lab sei die zugehörige Markierungsmenge). Wir konstruieren eine geordnete Grammatik $G' = (V'_N, V'_T, P', S', \prec)$ mit konstextfreien, nicht löschenden Regeln, welche $\{a\}M\{b\}$ akzeptiert. Es sei

$$V'_N = V_N \cup \{A, B, S', F\} \cup \{(A, r), [A, r], (Y, r), [Z, r] \mid r : XU \to YZ \in P\}$$

(disjunkte Vereinigungen). $V'_T = V_T \cup \{a, b\}$. P' enthalte die folgenden Regeln:

1. Einige einfache Start- und Terminierungsregeln sowie direkte Simulationsregeln:
   (a) $a \to A \prec \{y \to F \mid y \in V_{G'} \setminus V'_T\}$;
   (b) $b \to B \prec \{y \to F \mid y \in V_{G'} \setminus (V'_T \cup \{A\})\}$;
   (c) $x \to X \prec \{(A, r) \to F, [A, r] \to F \mid r \in Lab\}$ für kontextfreie Regeln $X \to x \in P$;
   (d) $ASB \to S'$;

2. Für jede "echte" kontextsensitive Regel der Form $r : XU \to YZ \in P$ führen wir folgende Simulationsregeln ein:
   (a) $A \to [A, r] \prec \{[A, s] \to F, (A, s) \to F \mid s \in Lab\}$;
   (b) $Y \to [Y, r] \prec \{A \to F, (A, s) \to F, [A, s'] \to F, [T, s] \to F, (T, s) \to F \mid s \in Lab, s' \in Lab \setminus \{r\}, T \in V_N\}$;
   (c) $Z \to (Z, r) \prec \{A \to F, (A, s) \to F, [A, s'] \to F, [T, s'] \to F, (T, s) \to F \mid s \in Lab, s' \in Lab \setminus \{r\}, T \in V_N\}$;

(d) $[A, r] \to (A, r) \prec \{A \to F, (A, s) \to F, [A, s'] \to F, [T, s'] \to F, (T, s') \to F, z(Z, r) \to F, [Y, r]y \to F \mid s \in Lab, s' \in Lab \setminus \{r\}, T \in V_N, z \in V_{G'} \setminus \{[Y, r]\}, y \in V_{G'} \setminus \{(Z, r)\}\}$; (Die linken und rechten Kontexte erfordern es, linke und rechte Randmarker einzuführen.)

(e) $(A, r) \to A \prec (Z, r) \to U \prec [Y, r] \to X \prec \{A \to F, [A, s] \to F, (A, s') \to F, [T, s'] \to F, (T, s') \to F \mid s \in Lab, s' \in Lab \setminus \{r\}, T \in V_N\}$;

Übung 14 *Beschreiben Sie in Worten, wie eine Simulation erfolgen soll. Wieso kann es keine "falschen" Ableitungen bei der Simulation geben?*

□

Geordnete Grammatiken—akzeptieren mehr ?! Weitere Folgerungen

Korollar 31 $\mathcal{L}(CF) \subset \mathcal{L}(O,CF\text{–}\lambda) \subset \mathcal{L}_{acc}(O,CF\text{–}\lambda)$. □

Korollar 32 $\mathcal{L}(O,CF) \subseteq \mathcal{L}_{acc}(O,CF) = \mathcal{L}(RE)$.

Beweis. Selbstverständlich gilt $\mathcal{L}(\mathrm{RE}) \subseteq \mathcal{L}_{\mathrm{acc}}(\mathrm{O,CF})$.
Bekanntermaßen (Falls unbekannt, Übung!) lässt sich jede aufzählbare Sprache $L \subseteq \Sigma^*$ als homomorphes Bild einer kontextsensitiven Sprache $L' \subseteq V^*$ darstellen, d.h., $L = h(L')$ für einen Morphismus $h : V^* \to \Sigma^*$. O.E.: $\Sigma \cap V = \emptyset$.
Betrachte neue Zeichenmenge $V'' = \{A'' \mid A \in V\}$. Unter Hinzunahme folgender Regeln klappt die Konstruktion aus dem vorigen Satz:

- $w \to A'' \prec \{B \to F \mid B \in V\}$ für $h(A) = w$; (Beachte: $w = \lambda$ ist möglich.)
- $A'' \to A$ für $A \in V$;
- $c \to F$ für gewisse (welche genau?) $c \in \Sigma \cup V''$ □

Geordnete Grammatiken—akzeptieren mehr ?! Weitere Folgerungen

Aus früheren Resultaten (welchen genau?) folgt:

Lemma 2 *Das Leerheitsproblem ist für generierende (O,CF)-Grammatiken entscheidbar*

Korollar 33 $\mathcal{L}(O,CF) \subset \mathcal{L}(REC)$

Beweis. Aus dem vorigen Lemma sowie den (konstruktiven) Abschlusseigenschaften (Schnitt mit regulären Sprachen) der geordneten (erzeugten) Sprachen folgt die Entscheidbarkeit des Wortproblems. □

Korollar 34 $\mathcal{L}(O,CF) \subset \mathcal{L}_{acc}(O,CF)$ □

Mehr Stoff zum Üben

Wir haben in diesem Abschnitt zahlreiche Grammatik-Formalismen kennengelernt.

Untersuchen Sie, wie sich das Konzept der akzeptierenden Grammatiken auf die Sprachfamilien auswirkt.

Wie sieht es aus, wenn Linksableitungsmodi zusätzlich betrachtet werden?
(Letztere Fragestellung liefert sicher auch Stoff für eine Diplom- / Master-Arbeit.)