

Formale Sprachen (parallele und regulierte Ersetzung)

SoSe 2007 in Trier

Henning Fernau

Universität Trier

fernau@informatik.uni-trier.de

Formale Sprachen Gesamtübersicht

- Organisatorisches / Einführung
Motivation / Erinnerung / Fragestellungen
- **Diskussion verschiedener Sprachklassen:**
gesteuerte Ersetzungsverfahren
parallele Ersetzungsverfahren
- Algebraische Ansätze:
abstrakte Sprachfamilien
formale Potenzreihen

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:

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

Ist Englisch kontextfrei ?

Trügerisches Argument: L aus vorigem Beispiel ist nicht kontextfrei.

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

Mit geeignetem R könnten wir L quasi “herausschneiden” aus E.

Daher ist E nicht kontextfrei.

Hinweis: Abschlusseigenschaften kann man oft gebrauchen, um Echtheit von Inklusionen 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 , \implies , \neg (sowie Klammern).

Atomare Aussagen werden durch Wörter über $\{a\}$ unär kodiert.

τ sei die Sprache der Theoreme in dem skizzierten Kalkül. Bekannt ist: $\tau \in \mathcal{L}_1$.

Betrachte die reguläre Sprache $R = \{(x \vee y) \implies z \mid x, y, z \in \{a\}^*\}$.

$\tau \cap R = \{(x \vee x) \implies x \mid x \in \{a\}^*\} \notin \mathcal{L}_2$, daher $\tau \notin \mathcal{L}_2$.

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

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

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

Interpretieren wir eine k -stellige Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ 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

hatten wir beiläufig in der allerersten Vorlesung kennengelernt.

Ist $h : \Sigma^* \rightarrow \Sigma^*$ 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 $f_G : \mathbb{N} \rightarrow \mathbb{N}, n \mapsto \ell(h^n(\omega))$ 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 ?

Im Folgenden teilweise (bewusst) englische Folien (von der MFCS 1996), s. [9].

Unified Framework

A *grammar controlled by a bicoloured digraph* (*graphgesteuerte Grammatik*) [3, 16, 17, 19, 25, 26] or *G grammar* is an 8-tuple $G = (V_N, V_T, P, S, \Gamma, \Sigma, \Phi, h)$ where

- V_N, V_T, P, S define the set of nonterminals, terminals, context-free core rules, and the start symbol;
- Γ is a bicoloured digraph, i.e., $\Gamma = (U, E)$, where U is a finite set of nodes and $E \subseteq U \times \{g, r\} \times U$ is a set of arcs coloured by g or r (“green” or “red”);
- $\Sigma \subseteq U$ are the initial nodes;
- $\Phi \subseteq U$ are the final nodes;
- $h : U \rightarrow (2^P \setminus \{\emptyset\})$ relates nodes with rule sets.

Two different definitions of the *appearance checking mode* (*Vorkommenstest*):

$(x, u) \Rightarrow (y, v)$ ($(x, u) \Rightarrow_c (y, v)$, respectively) holds in G with $(x, u), (y, v) \in (V_N \cup V_T)^* \times U$, if either, for some $(u, g, v) \in E$,

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

or every (one, respectively) rule of $h(u)$ is not applicable to x , $y = x$, $(u, r, v) \in E$. The reflexive transitive closure of \Rightarrow (\Rightarrow_c , respectively) is denoted by \Rightarrow^* (\Rightarrow_c^* , respectively). The corresponding languages generated by G are defined by $L_{[c]}(G) =$

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

Language families:

- $\mathcal{L}(G_{[c]}, CF, ac)$: general case. (*mit Vorkommenstest*)
- $\mathcal{L}(G_{[c]}, CF)$: all arcs are green. (*ohne Vorkommenstest*)
- $\mathcal{L}(G_{[c]}, CF, ut)$: all arcs are both green and red. (*mit unbedingtem Übergang*)

We write $CF-\lambda$ instead of CF if we do not allow erasing core rules.

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

Special cases of G grammars:

- **Programmed grammars** [3, 18, 22, 23]:
 - Every node contains exactly one rule.
 - Every node is both initial and final.Usual notation of rules: $(r : A \rightarrow w, \sigma, \phi)$.
Language families: $\mathcal{L}(P, CF, ac)$, $\mathcal{L}(P, CF, ut)$, and $\mathcal{L}(P, CF)$.
- Grammars with **regular control** [12, 3, 22]:
 - Every node contains exactly one rule.
 - If there is a red arc from node u to node v , then there is also a green arc from node u to node v .
- **Matrix grammars** [1, 3, 22]: an rC grammar obeying the additional restriction:
 - Only the initial nodes are allowed to have more than one in-going green arc.
 - Only final nodes are allowed to have more than one out-going green arc.
 - Between every final node and every initial node, there is a green arc.

- **Time-variant grammars** [3, 20, 21, 22]:
 - If there is a red arc from node u to node v , then there is also a green arc from node u to node v . (The “green graph” is a ring.)
 - There is one initial node, and every node is final.

Usual notation: a periodic function ϕ mapping natural numbers to rule sets such that $\phi(n)$ prescribes the n^{th} derivation step.

Now, we can consider both the \Rightarrow and the \Rightarrow_c derivations, leading to the families of languages $\mathcal{L}(\text{TV}_{[c]}, \text{CF}, \text{ac})$, $\mathcal{L}(\text{TV}_{[c]}, \text{CF}, \text{ut})$, and $\mathcal{L}(\text{TV}_{[c]}, \text{CF})$.

Similar: **Matrix set** and **regular set control grammars**.

Known facts

Theorem 1. If $X \in \{G_c, P, M, rC, TV_c\}$, $Y \in \{CF, CF - \lambda\}$, $Z \in \{ac, \lambda\}$, then $\mathcal{L}(X, Y, Z) = \mathcal{L}(G, Y, Z)$.

Theorem 2.

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

(Sources: [3, 7, 5, 13, 15])

Programmed grammars

Original interpretation of rule $(p : A \rightarrow w, \sigma, \phi)$:

- in general:
if $A \rightarrow w$ is applicable then
begin apply $A \rightarrow w$;
 continue with a rule from σ
end
else begin continue with a rule from ϕ
end.
- $\phi = \emptyset$:
if $A \rightarrow w$ is applicable then
begin apply $A \rightarrow w$;
 continue with a rule from σ
end.
- $\phi = \sigma$:
if $A \rightarrow w$ is applicable then
begin apply $A \rightarrow w$
end;
continue with a rule from σ .

Auf dem Weg zu Theorem 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 v ein Knoten, zu dem $h(v)$ mehr als eine Regel zuordnet, so ersetze v durch $|h(v)|$ Kopien; jede dieser Kopien enthalte genau eine der Regeln von $h(v)$, und die Vorgänger- bzw. Nachfolger-Knoten von v 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 Theorem 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: $S' \rightarrow pS$, wobei S das Startzeichen von G ist und p ein möglicher Anfangsknoten. Ist q ein Endknoten, so gibt es noch die Matrizen $[q \rightarrow \lambda, A \rightarrow w]$ (**übliche Matrizenschreibweise!**), wobei $A \rightarrow w$ zu q assoziiert ist und w ein Terminalwort ist.

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

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.

Auf dem Weg zu Theorem 2

Hauschildt und Jansen haben einen hübschen Zusammenhang mit der Theorie der Petrinetze gefunden [13], der es gestattet zu folgern, dass

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

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

Folgerung: $\mathcal{L}(G,CF)$ 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* $\phi_0 : S \rightarrow \mathbb{N}$.

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: 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.

**nochmal englische Folien, diesmal Beschreibungskomplexität, MCU 2001,
s. [11]**

Descriptive Complexity—Ideas

How concise can a description of an object be?

Typical measures for automata: #states, #rules

Typical measures for grammars: #nonterminals, #rules

Descriptive Complexity—Some Results

No general upper bound on the state complexity of finite automata.

Shannon: Turing machines have state complexity of two (for RE).

Minsky: Register machines have register complexity of two (for RE).

Păun proved an upper bound of **six** on the nonterminal complexity of matrix grammars (for RE), yielding an upper bound of **eight** on the nonterminal complexity of programmed grammars.

Here: upper bound of **three** on the nonterminal complexity of programmed grammars and of **four** on the nonterminal complexity of matrix grammars.

Programmed Grammars—The 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

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

where

- $r : A \rightarrow w$ 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

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

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

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

\mathcal{P} : the family generated by programmed grammars;

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

Programmed Grammars—An Example

$G = (\{A, B\}, \{a\}, P, A)$, where P contains:

- (1 : $A \rightarrow BB$, {1}, {2}),
- (2 : $B \rightarrow A$, {2}, {1, 3}),
- (3 : $A \rightarrow a$, {3}, {3}).

Then, $L(G) = \{a^{2^n} \mid n \geq 0\}$.

Programmed Grammars with Unconditional Transfer—Example

$G_1 = (\{A, B, F\}, \{a\}, P_1, A)$, where P contains:

- (1 : $A \rightarrow BB$, {1, 2}, {1, 2}),
- (2 : $A \rightarrow F$, {3}, {3}),
- (3 : $B \rightarrow A$, {3, 4}, {3, 4}),
- (4 : $B \rightarrow F$, {1, 5}, {1, 5}),
- (5 : $A \rightarrow a$, {5}, {5}).

Then, $L(G_1) = \{a^{2^n} \mid n \geq 0\}$.

Main Result

Theorem 3. $\mathcal{P}_3 = \mathcal{L}_0$.

Proof: Simulate (generative) Turing machine.

A *Turing Machines* (with one one-sided tape) is given by

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

where:

$\#_L/\#_R \in \Gamma$ is the left/right endmarker,

$\# \in \Gamma$ is the blank symbol.

A *configuration* of M is described by a word

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

with $\tilde{\Gamma} = \Gamma \setminus \{\#_L, \#_R\}$. Here $c = \#_L w q v$ means: The head of the Turing machine is currently scanning the last symbol a of $\#_L w$.

Encoding

Given a configuration $c \in (\Gamma \cup Q)^*$ of M , let $\beta(c) \in \{0, 1\}^*$ denote some binary encoding of c using $\gamma = \lceil \log_2(|\Gamma| + |Q|) \rceil$ many bits per symbol from $\Gamma \cup Q$. We interpret strings $\beta(c)$ as natural numbers.

Assume: $\beta(\#) = 0^\gamma$, $\beta(\#_L) = 0^{\gamma-1}1$, $\beta(\#_R) = 10^{\gamma-1}$.

Observe that $|\beta(c)| = |c|\gamma$.

c can be codified uniquely over $\{A\}$ by $A^{\beta(c)} \neq \lambda$.

Simulation loop

Aim: Simulate $c \vdash_{\mathcal{M}} c'$ by $(A^{\beta(c)}B, p) \xRightarrow{*} (A^{\beta(c')}B, p')$;
the Turing machine state q of c is stored in the label p .

Implementation:

1. pass over a nondeterministically chosen number of codified letters;
2. verify that subword αq stands at the current position;
3. replace αq according to δ ;
4. (return to standard presentation).

Skipping a symbol

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

Here, $\text{exit-skip}(i)$ equals $\{(\text{skip}, q, i + 1, 1)\}$ if $i < \gamma$ and $\text{simstart}(q)$, otherwise.

Simulation (of $r = (q, a, X, q', a') \in \delta$, $a \in \tilde{\Gamma}$ and $X = L$).

Let $\beta(aq) = \beta_1 \dots \beta_{2\gamma}$ with $\beta_i \in \{0, 1\}$ and

$\beta(q'a') = \beta'_1 \dots \beta'_{2\gamma}$ with $\beta'_i \in \{0, 1\}$.

The checking phase:

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

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

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

The generating phase:

$((\text{sim-1}, r, i, 9) : B \rightarrow C^2, \{(\text{sim-1}, r, i, 9)\}, \{(\text{sim-1}, r, i, 10)\})$
$((\text{sim-1}, r, i, 10) : C \rightarrow B, \{(\text{sim-1}, r, i, 10)\}, f'_i(\beta'_i))$
$((\text{sim-1}, r, i, 11) : B \rightarrow B^2, \text{exit-sim-1}(i), \emptyset)$

Here,

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

and $\text{exit-sim-1}(i)$ equals $\{(\text{sim-1}, r, i + 1, 9)\}$ if $i < 2\gamma$ and $\{(\text{return}, q', 1)\}$, otherwise. In any case, we have $1 \leq i \leq 2\gamma$ and $1 \leq j \leq 11$.

Returning to standard presentation

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

An Example: Assume that $\#_L q_0 \#_R \vdash \#_L \#_R q_0 \vdash \#_L q_f a \#_R \vdash \#_L a q_f \#_R$ is a terminating run of a given Turing machine. Assume further a three-bit-codification:

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

Taking binary numbers as exponents, the simulating grammar derives:

$$\begin{array}{ll}
 A \Rightarrow A^{1;011;100}B & \text{(since } \beta(\#_L q_0 \#_R) = 001011100\text{)} \\
 \xRightarrow{*} A^{1;100;011}B & \text{(simul. a right move, scanning } \#_L\text{)} \\
 \xRightarrow{*} A^{1;101;010;100}B & \text{(simul. a left move, scanning } \#_R\text{)} \\
 \xRightarrow{*} A^{1;101;010}B^{1;001} & \text{(using skip)} \\
 \xRightarrow{*} A^{1;010;101}B^{1;001} & \text{(simul. a right move, scanning } \#_L\text{)} \\
 \xRightarrow{*} A^{1;010;101;100}B & \text{(return to standard encoding)} \\
 \xRightarrow{*} A^{1;010;101}B & \text{(using term for } \#_R\text{)} \\
 \xRightarrow{*} A^{1;010}B & \text{(using term for } q_f\text{)} \\
 \xRightarrow{*} A^1 a B & \text{(using term for } a, \text{ i.e., conversion)} \\
 \xRightarrow{*} a & \text{(using term for } \#_L\text{)}
 \end{array}$$

Programmed Versus Graph-Controlled

- no explicit start rule
 \rightsquigarrow avoid rule sequences $A \xRightarrow{*} \lambda$ in the simulating grammar (start symbol A)
- no explicit termination rules
 \rightsquigarrow avoid premature termination possibilities

Cf.: Păun and Freund present at MCU a proof that three nonterminals are enough for graph-controlled grammars.

Further Consequences—Leftmost Derivations

Cor. 1. $\mathcal{P}_3^{\ell-3} = \mathcal{L}_0$.

Theorem 4. $\mathcal{P}_4^{\ell-2} = \mathcal{L}_0$.

Further Consequences—Matrix Grammars

Cor. 2. $\mathcal{M}_4 = \mathcal{M}_4^{\ell-3} = \mathcal{L}_0$.

Cor. 3. $\mathcal{M}_5^{\ell-2} = \mathcal{L}_0$.

Leftmost derivations — ideas (SOFSEM 98), s. [10]

- (i) *At each step of a derivation, the leftmost occurrence of a nonterminal is rewritten.*
- (ii) *At each step of a derivation, the leftmost occurrence of a nonterminal which can be rewritten is rewritten.*

In regulated grammars, only certain nonterminal occurrences can be rewritten in a given stage of derivation.
- (iii) *to use each rule in a leftmost manner, i.e., the leftmost appearance of its left-hand member is rewritten.*

Set modes, abbreviated as \Rightarrow_{s_1} and \Rightarrow_{s_2} .

Let $G = (V_N, V_T, P, S, \Gamma, \Sigma, \Phi, h)$ with $\Gamma = (U, E)$ be a G grammar.

Let $(x, V), (x', V_{s_i}) \in (V_N \cup V_T)^* \times 2^U$.

$(x, V) \Rightarrow_{s_i} (x', V_{s_i})$ holds in G , if

- either, for some $v \in V$,

$$x = x_1 \alpha x_2, \quad x' = x_1 \beta x_2, \quad \alpha \rightarrow \beta \in h(v),$$

where $V_{s_i} = \{v' \mid (v, g, v') \in E\}$,

- or no rule of $\bigcup_{v \in V} h(v)$ is applicable to x , $x' = x$,
 - $V_{s_1} = \{v' \mid (v, r, v') \in E\}$ for some $v \in V$,
 - $V_{s_2} = \{v' \mid \exists v \in V (v, r, v') \in E\}$.

$$L_{s_i}(G) = \{x \in V_T^* \mid \exists V \in 2^U, V \cap \Phi \neq \emptyset \text{ and } (S, \Sigma) \xRightarrow{*}_{s_i} (x, V)\}$$

Remarks:

- (i) $L_m(G) = L(G)$ for every $m \in \{c, s_1, s_2\}$ and every G grammar G without appearance checks, since it does not matter whether we choose first the node and then the rule to be applied or whether we choose a rule from the set of all possibilities.
- (ii) $L_s(G) = L(G)$ for every $s \in \{s_1, s_2\}$ and every TV grammar G, since there is always only at most one possible continuation node.
- (iii) $L_m(G) = L(G)$ for every $m \in \{c, s_1, s_2\}$ and every RC grammar G by combining the arguments of the first two points: inside the matrices of the MS grammar, there is (at most) one possible continuation (ii), and between (different) matrices, there are only green arcs (i).

Lemma 4. For $Y \in \{\text{CF}, \text{CF} - \lambda\}$, $Z \in \{\text{ac}, \text{ut}\}$, $s \in \{s_1, s_2\}$:

$$\mathcal{L}(\mathbf{G}, Y, Z) \subseteq \mathcal{L}(\mathbf{G}_s, Y, Z).$$

Lemma 5. For $Y \in \{\text{CF}, \text{CF} - \lambda\}$, $s \in \{s_1, s_2\}$:

$$\mathcal{L}(\mathbf{G}_s, Y, \text{ac}) \subseteq \mathcal{L}(\mathbf{G}, Y, \text{ac}).$$

Theorem 5. For $Y \in \{\text{CF}, \text{CF} - \lambda\}$, $s \in \{s_1, s_2\}$:

$$\mathcal{L}(\mathbf{M}_s, Y, \text{ut}) = \mathcal{L}(\mathbf{M}, Y, \text{ac}).$$

Proof. \subseteq : Exercise.

\supseteq : Consider a P grammar

$$G = (V_N, V_T, P, S, \Gamma = (\mathcal{U}, E), h), \quad V_N = \{A_1, \dots, A_n\}, \quad \mathcal{U} = \{u_1, \dots, u_k\}.$$

Simulating ut M grammar

$$G' = (V'_N, V_T, P', S', \Gamma', h')$$

with $V'_N = V_N \cup \mathcal{U} \cup \{S', F\}$:

- start matrices ($S' \rightarrow uS$) for all $u \in \mathcal{U}$;
- a termination matrix ($A_1 \rightarrow F, \dots, A_n \rightarrow F, u_1 \rightarrow \lambda, \dots, u_k \rightarrow \lambda$);
- trap matrices ($u \rightarrow F$) for every $u \in \mathcal{U}$.
- For every $u \in \mathcal{U}$ and every u' with $(u_i, g, u') \in E$ we define a matrix ($A \rightarrow w, u_1 \rightarrow F, \dots, u_{i-1} \rightarrow F, u_{i+1} \rightarrow F, \dots, u_k \rightarrow F, u_i \rightarrow u'$) for $h(u_i) = \{A \rightarrow w\}$.
- For every $u \in \mathcal{U}$ and every u' with $(u, r, u') \in E$ we define a matrix ($u \rightarrow u', A \rightarrow F$) for $h(u) = \{A \rightarrow w\}$.

(Γ' , P' and h' are implicitly defined.)

Theorem 6. Summary of free case.

Let $Y \in \{\text{CF}, \text{CF} - \lambda\}$, $Z \in \{\lambda, \text{ac}\}$.

- (i) $\mathcal{L}(P, Y, Z) = \mathcal{L}(X_m, Y, \text{ac})$ for $X \in \{\text{G}, \text{M}, \text{MS}, \text{rC}, \text{rSC}, \text{TV}, \text{P}\}$ and $m \in \{\lambda, \text{c}, s_1, s_2\}$.
- (ii) $\mathcal{L}(P, Y, \text{ac}) = \mathcal{L}(X_m, Y, \text{ut}) = \mathcal{L}(X'_s, Y, \text{ut})$ for $X \in \{\text{G}, \text{MS}, \text{rSC}, \text{TV}\}$, $m \in \{\lambda, s_1, s_2\}$, $X' \in \{\text{rC}, \text{M}, \text{P}\}$, and $s \in \{s_1, s_2\}$.
- (iii) $\mathcal{L}(P, Y, \text{ut}) = \mathcal{L}(X_c, Y, \text{ut}) = \mathcal{L}(X'_m, Y, \text{ut})$ for $X \in \{\text{G}, \text{MS}, \text{rSC}, \text{TV}\}$, $X' \in \{\text{rC}, \text{M}, \text{P}\}$, and $m \in \{\lambda, \text{c}\}$.

Theorem 7. Summary of left-1 case.

Let $X \in \{G, TV, P, M[S], r[S]C\}$, $m \in \{\lambda, c, s_1, s_2\}$, $Y \in \{CF, CF-\lambda\}$, $Z \in \{ac, ut, \lambda\}$. Then:

$$\mathcal{L}(X_m, Y, Z, \text{left-1}) = \mathcal{L}(CF).$$

Proof. $G = (V_N, V_T, P, S, \Gamma = (U, E), \Sigma, \Phi, h)$ be a G grammar.

Idea: variant of triple construction.

Here: Restriction to \Rightarrow -mode.

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

— $S_0 \rightarrow (u, S, u')$, if $u \in \Sigma, u' \in \Phi$.

Consider $A \rightarrow w \in h(u)$.

— If $(u, r, u'') \in E$ and A is not left-hand side of any rule in u , $(u, A, u') \rightarrow (u'', A, u') \in P'$;

— If $w \in V_T^*$, then put $(u, A, u) \rightarrow w$ into P' .

— Let $w = x_0 B_1 x_1 B_2 \cdots x_{r-1} B_r x_r$ with

$x_i \in V_T^*$ and $B_i \in V_N$. Put $(u_0, A, u_r) \rightarrow x_0(u_0, B_1, u_1)x_1(u_1, B_2, u_2) \cdots x_{r-1}(u_{r-1}, B_r, u_r)x_r$ into P' where all $u_i \in U$ and $(u_0, g, u_r) \in E$.

Leftmost 2 derivations

The derivation mode \Rightarrow_c :

State: a pair (u, w) , where $u \in U$ and $w \in V_G^*$.

First, we choose a rule $A \rightarrow v \in h(u)$.

So, there is just one “nonterminal which can be rewritten”, namely A .

If possible, replace A 's leftmost occurrence by v and proceed via one of the green arcs leaving node u ; otherwise, leave node u by one of its outgoing red arcs.

The derivation mode \Rightarrow :

State: a pair (u, w) , where $u \in U$ and $w \in V_G^*$.

where $u \in U$ and $w \in V_G^*$.

The set N of “nonterminals which can be rewritten” is the set of left-hand sides of rules in $h(u)$.

If possible, replace the symbol $A \in N$ occurring leftmost in w using some rule $A \rightarrow v \in h(u)$ and proceed via one of the green arcs leaving node u . If no symbol from N is contained in w , we can leave node u by one of its outgoing red arcs.

The derivation modes \Rightarrow_{s_1} and \Rightarrow_{s_2} :

State: a pair (V, w) , where $V \subseteq U$ and $w \in V_G^*$.

The set N of “nonterminals which can be rewritten” is the set of left-hand sides of rules in $\bigcup_{u \in V} h(u)$.

If possible, replace the symbol $A \in N$ occurring leftmost in w using some rule $A \rightarrow v \in h(u)$ (for some $u \in V$) and proceed via one of the green arcs leaving node u . If no symbol from N is contained in w , the set of next nodes is defined via the outgoing red arcs of one (all, respectively) nodes in V .

Theorem 8. Summary of left-2 case.

(not touching left-3...)

Let $X \in \{G, TV, MS, rSC\}$, $X' \in \{P, M, rC\}$, $m \in \{\lambda, s_1, s_2\}$, $s \in \{s_1, s_2\}$, $Y \in \{CF, CF-\lambda\}$, $Z \in \{ac, ut, \lambda\}$. Then,

$$(i) \mathcal{L}(X_m, CF-\lambda, Z, \text{left-2}) = \mathcal{L}(X'_s, CF-\lambda, Z, \text{left-2}) = \mathcal{L}(RC_m, CF-\lambda, [ac], \text{left-2}) = \mathcal{L}_1;$$

$$(ii) \mathcal{L}(X_m, CF, Z, \text{left-2}) = \mathcal{L}(X'_s, CF, Z, \text{left-2}) = \mathcal{L}(RC_m, CF, [ac], \text{left-2}) = \mathcal{L}_0.$$

Proofs necessary for P (textbook), TV, MS, and RC.

Question: What about forbidding RC?

Theorem 9. Summary of left-3 case. Let $X \in \{ G, TV, MS, rSC, P, M, rC \}$, $Y \in \{ CF, CF-\lambda \}$, $Z \in \{ \lambda, ac, ut \}$.

- $\mathcal{L}(X_c, Y, Z, \text{left-2}) = \mathcal{L}(X_c, Y, Z, \text{left-3})$.
- $\mathcal{L}(P, Y, Z[, \text{left-3}]) = \mathcal{L}(X_c, Y, Z[, \text{left-3}])$.

Theorem 10. Summary of left-3 case. Let $X \in \{ G, TV, MS, rSC, M, rC \}$, $X' \in \{ G, TV, MS, rSC \}$, $Y \in \{ CF, CF-\lambda \}$. Then, we have

- $\mathcal{L}(P, Y[, \text{left-3}]) = \mathcal{L}(X, Y[, \text{left-3}])$;
- $\mathcal{L}(P, Y, ac[, \text{left-3}]) = \mathcal{L}(X, Y, ac[, \text{left-3}])$
 $= \mathcal{L}(X', Y, ut[, \text{left-3}])$;
- $\mathcal{L}(P, Y, ut[, \text{left-3}]) = \mathcal{L}(M, Y, ut[, \text{left-3}])$
 $= \mathcal{L}(rC, Y, ut[, \text{left-3}])$.

Theorem 11. Let $X \in \{G, M, MS, rC, rSC, TV, P\}$, $Y \in \{CF, CF - \lambda\}$, $Z \in \{\lambda, ac\}$.

(i) $\mathcal{L}(P, Y, Z[, left-3]) = \mathcal{L}(X_m, Y, Z[, left-3])$ for $m \in \{\lambda, c, s_1, s_2\}$.

(ii) $\mathcal{L}(P, Y, ac[, left-3]) = \mathcal{L}(X_s, Y, ut[, left-3])$ for $s \in \{s_1, s_2\}$.

Hierarchical relations.

- (i) $\mathcal{L}(\mathbf{P}, \mathbf{CF}) \subsetneq \mathcal{L}(\mathbf{P}, \mathbf{CF}, \text{left-3}) \subsetneq \mathcal{L}(\mathbf{P}, \mathbf{CF}, \text{ac}[, \text{left-3}]) = \mathcal{L}(\mathbf{P}, \mathbf{CF}, \text{ut}, \text{left-3}) = \mathcal{L}_0.$
- (ii) $\mathcal{L}(\mathbf{P}, \mathbf{CF} - \lambda) \subsetneq \mathcal{L}(\mathbf{P}, \mathbf{CF} - \lambda, \text{left-3}) \subsetneq \mathcal{L}(\mathbf{P}, \mathbf{CF} - \lambda, \text{ac}, \text{left-3}) \subsetneq \mathcal{L}_1.$
- (iii) $\mathcal{L}(\mathbf{P}, \mathbf{CF} - \lambda, \text{ut}, \text{left-3}) \subsetneq \mathcal{L}(\mathbf{P}, \mathbf{CF} - \lambda, \text{ac}, \text{left-3})$
- (iv) **Open questions:** Are any of the following inclusions strict ?
- $\mathcal{L}(\mathbf{P}, \mathbf{CF} - \lambda, \text{ac}) \subseteq \mathcal{L}(\mathbf{P}, \mathbf{CF} - \lambda, \text{ac}, \text{left-3})$
 - $\mathcal{L}(\mathbf{P}, \mathbf{CF}[-\lambda], \text{ut}) \subseteq \mathcal{L}(\mathbf{P}, \mathbf{CF}[-\lambda], \text{ac}[, \text{left-3}])$
 - $\mathcal{L}(\text{fRC}, \mathbf{CF}[-\lambda]) \subseteq \mathcal{L}(\text{fRC}, \mathbf{CF}[-\lambda], \text{left-3})$

Main Lemma (MFCS slides)

Let $L \in \mathcal{L}_0$, $L \subseteq V_T^*$. Then:

there is a (P,CF,ut) grammar \tilde{G} with:

$w \in L$ iff $w\$ \in L(\tilde{G})$,

where $\{\$, \#\} \cap V_T = \emptyset$.

Simulation idea of given (P,CF,ac) grammar $G = (V_N, V_T, P, S)$:

(1) one initialization production

(init : $\tilde{S} \rightarrow S\$ \sigma, \{p_i^+, p_i^- \mid A_i = S\}$),

(2) termination rules ($t_i : B_i \rightarrow F, \{t_{i+1}\}$) for $1 \leq i \leq n$ and

($t_{n+1} : \sigma \rightarrow \#, \{t_{n+1}\}$), and

(3), for every p_i , $1 \leq i \leq m$, simulating rules

$$\begin{aligned} (p_i^- & : A_i \rightarrow F, \{q^+, q^- \mid q \in \Phi_i\}) & , \\ (p_i^+ & : A_i \rightarrow w_i \sigma, \{p_i'\}) & , \text{ and} \\ (p_i' & : \sigma \rightarrow \lambda, \{q^+, q^- \mid q \in \sigma_i\} \cup \{t_1\}) & . \end{aligned}$$

□

Corollaries

- $\mathcal{L}(P,CF,ut)$ contains non-recursive languages.
- $\mathcal{L}(P,CF-\lambda,ut) \subsetneq \mathcal{L}(P,CF,ut)$.
- $\mathcal{L}(P,CF,ut) = \mathcal{L}_0$ iff
 $\mathcal{L}(P,CF,ut)$ is closed under right derivatives.
 (anyhow, non-effective closure!)

Remark: Emptiness is decidable for (P,CF,ut) grammars [18].

The non-erasing case

Let $L \in \mathcal{L}(P, CF-\lambda, ac)$, $L \subseteq V_T^*$.

Then, there is a constant $m > 0$ and a $(P, CF-\lambda, ut)$ grammar \tilde{G} such that

$w \in L$ iff $w\$ \#^m \in L(\tilde{G})$,

where $\{\$, \#\} \cap V_T = \emptyset$.

Therefore:

$\mathcal{L}(P, CF-\lambda, ut) = \mathcal{L}(P, CF-\lambda, ac)$ iff

$\mathcal{L}(P, CF-\lambda, ut)$ is closed under right derivatives.

Characterizations of UT

$\mathcal{L}(P, CF[-\lambda], ut)$ is characterized by:

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

TV case quite involved.

Characterizations of AC

$\mathcal{L}(P, CF[-\lambda], ac)$ is characterized by:

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

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

Let $L \in \mathcal{L}(P,CF,ac)$, $L \subseteq V_T^*$,
 $L = \bigcup_{a \in V_T} (\{a\}V_T^+ \cap L) \cup ((V_T \cup \{\lambda\}) \cap L)$.

We construct a (G,CF,ut) grammar

$G' = (V_N \cup \{\#\}, F, S', V_T, P', S', \Gamma, \Sigma, \Phi, h)$ with

$\Gamma = (U, E)$ such that $L(G') = \{a\}V_T^+ \cap L$.

Easy: $\mathcal{L}(G,CF,ut)$ is closed under union.

$G = (V_N, V_T, P, S)$ be a (P,CF,ac) grammar generating $\{w \in V_T^+ \mid aw \in L\}$.

A rule $(p_i : A_i \rightarrow w_i, \sigma_i, \phi_i)$ of G determines two production nodes F_i^+, F_i^- with

$h(F_i^+) = \{A_i \rightarrow w_i, \# \rightarrow F\}$,

$E \cap \{F_i^+\} \times \{g, r\} \times U = \{F_i^+\} \times \{g, r\} \times (\{F_j^{+/-} \mid p_j \in \sigma_i\} \cup \{t\})$,

$h(F_i^-) = \{A_i \rightarrow F\}$,

$E \cap \{F_i^-\} \times \{g, r\} \times U = \{F_i^-\} \times \{g, r\} \times \{F_j^{+/-} \mid p_j \in \phi_i\}$.

Initial node I contains the rule $S' \rightarrow \#S$ with green and red arcs leading to $\{F_i^+, F_i^- \mid (p_i : S \rightarrow w_i, \sigma_i, \phi_i) \in P\}$.

Final node T contains the rule $\# \rightarrow a$ with green and red arcs leading to itself.

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

Let $G = (V_N, V_T, P, S, \Gamma, \Sigma, \Phi, h)$ be a (G,CF,ut) grammar with $\Gamma = (U, E)$;

$U = \{U_1, \dots, U_n\}$,

$U_i = \{A_{i1} \rightarrow w_{i1}, \dots, A_{ir_i} \rightarrow w_{ir_i}\}$.

We give a simulating (TV,CF,ut) -grammar

$(V_N \cup \{X^{(i)} \mid X \in V_N, 1 \leq i \leq n\} \cup \{S', F\} \cup U, P', S', \phi)$, where ϕ is a function with period $2n + 3$:
for $i = 1, \dots, n$, let

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

Higman's Lemma (Now: Slides from DLT'97)

Confer *Higman* [14, Theorem 4.4].

Let $u, v \in \Sigma^*$, $u = u_0u_1 \dots u_n$. u divides v , written $u \mid v$, if $v \in \Sigma^*u_0\Sigma^*u_1\Sigma^* \dots \Sigma^*u_n\Sigma^*$. Division is a partial ordering on Σ^* .

Theorem 1 [Higman] *Every set L of words over a finite alphabet has a finite subset L' such that every word in L has a divisor in L' .*

Call L' a *Higman set* for L .

Remark: No effective proof!

Especially, for Turing machines T , calculating L' for $L(T)$ needs the halting problem as oracle.

Theorem 2 *Let \mathcal{G} be a grammar family with a decidable emptiness problem which is effectively closed under intersection with regular sets. Then, there is an algorithm which constructs for every given grammar $G \in \mathcal{G}$ a Higman set $L' \subseteq L(G)$.*

Proof. Let $G \in \mathcal{G}$, where $L(G) \subseteq \Sigma^*$. For $w \in \Sigma^*$, $I(w) := \{u \in \Sigma^* \mid w \mid u\}$ is regular. So,

$$I(\{w_1, \dots, w_n\}) := I(w_1) \cup \dots \cup I(w_n)$$

is regular. L' is Higman set iff $L(G) \subseteq I(L')$ iff $L(G) \cap (\Sigma^* \setminus I(L')) = \emptyset$. The latter property can be tested algorithmically. Take some enumeration of $L(G)$, say w_0, w_1, \dots , and compute

```

L' := {w0}; i := 1;
while L(G) ∩ (Σ* \ I(L')) ≠ ∅ do begin
    L' := L' ∪ {wi}; i := i + 1
end

```

Theorem 1 ensures termination. □

Remark: A Higman set can be computed quite easily from special language descriptions. For regular expressions, i.e.,

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

one can use the following algorithm:

- $L'(R_1) = \{R_1\}$ 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
 - $L'(R_1^*) = \{\lambda\}$. ← !

Main Theorem (DLT 97)

Theorem 3 $\mathcal{L}_0 = \mathcal{L}(P\text{-left-3}, CF, ut)$.

Generalization of $\mathcal{L}_0 = \mathcal{L}(P\text{-left-3}, CF, ac)$, see [18, Theorem 5] and [3, Theorem 1.2.5].

Proof. Let $L \subseteq \Sigma^*$ be r.e. with Higman set L' . So, $L = \bigcup_{w \in L'} (L \cap I(w))$.
 $L(w) := L \cap I(w) \in \mathcal{L}(P\text{-left-3}, CF, ac)$.

The simulation of the type-0 grammar in [3] uses a special coding which is finally translated into the terminal word. Knowing that w divides each word of $L(w)$, we can alter the construction such that we start with $S \Rightarrow S'S_0w_1S_1 \cdots w_nS_n$, where S' starts the simulation as in the proof of [3], while the output routine is changed such that the symbols S_i “fill in” the desired parts of the terminal word.

More details of the proof

So, we have a (P-left-3,CF,ac) grammar $G = (V_N, \Sigma, P, S)$ for $L(w)$.

Let $P = \{p_1, \dots, p_m\}$, and $(p_\mu : A_\mu \rightarrow w_\mu, \sigma(p_\mu), \phi(p_\mu))$.

We give a (P-left-3,CF,ut) grammar $\tilde{G} = (\tilde{V}_N, \Sigma, \tilde{P}, \tilde{S})$ with $L(\tilde{G}) = L(w)$.

Since $\mathcal{L}(P\text{-left-3, CF, ut})$ is easily seen to be closed under finite union, this shows that $L \in \mathcal{L}(P\text{-left-3, CF, ut})$.

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

Let $\tilde{V}_N = V_N \cup \bar{V}_G \cup \{\tilde{S}, F, E\} \cup E_\Sigma$, where $V_N = \{B_1, \dots, B_\ell\}$ and $\Sigma = \{a_1, \dots, a_r\}$, $E_\Sigma = \{E_\rho \mid 1 \leq \rho \leq r\}$.

\tilde{P} contains

- an initialization rule

(init : $\tilde{S} \rightarrow h(x)Eg'(x), \{p_\mu^- \mid A_\mu = S\}$)

instead of $S \rightarrow x$ where $x = S'S_0w_1S_1 \cdots w_nS_n$,

- simulation rules for $\mu = 1, 2, \dots, m$ (cf. [8, 9]):

$(p_\mu^- : h(A_\mu) \rightarrow F, \{q^+, q^- \mid q \in \phi(p_\mu)\})$,

$(p_\mu^+ : h(A_\mu) \rightarrow E, \{p'_\mu\})$,

$(p'_\mu : E \rightarrow h(w_\mu), \{p''_\mu\})$,

$(p''_\mu : g(A_\mu) \rightarrow g(w_\mu), \{q^+, q^- \mid q \in \sigma(p_\mu)\} \cup \{t_1\})$.

- termination rules (different from [9]):

$$\begin{array}{ll}
(\mathbf{t}_i & : \mathbf{B}_i \rightarrow \mathbf{F}, \{\mathbf{t}_{i+1}\}) & , \text{ for } i = 1, 2, \dots, \ell - 1, \\
(\mathbf{t}_\ell & : \mathbf{B}_\ell \rightarrow \mathbf{F}, \{\mathbf{t}_{\ell+1}\}) & , \\
(\mathbf{t}_{\ell+1} & : \mathbf{E} \rightarrow \mathbf{E}_1 \cdots \mathbf{E}_r, \{\mathbf{T}_1\}) & (\text{no error in simul.}), \\
(\mathbf{T}_\rho & : \mathbf{g}(\mathbf{a}_\rho) \rightarrow \mathbf{E}_\rho, \{\mathbf{T}'_\rho\}) & , \text{ for } \mathbf{a}_\rho \in \Sigma, \\
(\mathbf{T}'_\rho & : \mathbf{E}_\rho \rightarrow \lambda, \{\mathbf{T}''_\rho\}) & (\text{if error then erase all}), \\
(\mathbf{T}''_\rho & : \mathbf{E}_\rho \rightarrow \mathbf{E}_\rho \mathbf{a}_\rho, \{\mathbf{T}_\rho, \mathbf{T}'''_\rho\}) & , \\
(\mathbf{T}'''_\rho & : \mathbf{g}(\mathbf{a}_\rho) \rightarrow \mathbf{F}, \{\mathbf{T}_{(\rho \bmod r)+1}\}) & .\square
\end{array}$$

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

$$x = x_0 w_1 x_1 \dots w_n x_n$$

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

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

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

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

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.

Corollaries

- For $L \subseteq \{a\}^*$, the following assertions are equivalent: $L \in \mathcal{L}_0$, $L \in \mathcal{L}(P, CF, ut)$, $L \in \mathcal{L}(klETOL)$ (see *Wätjen* [24]) without left derivations.
- $\mathcal{L}(P,CF,ut)$ and $\mathcal{L}(klETOL)$ contain non-recursive languages. [8, 6, 9]
- $\mathcal{L}(P[-left-3], CF, ut)$ is closed, but not effectively closed, under right derivatives.

Remark: (*Rosenkrantz* [18]) Emptiness is decidable for $(P[-left-3],CF,ut)$ grammars. \implies Recursion

Left-1 and Left-2

Theorem 4 $\mathcal{L}(P\text{-left-1}, CF, ut) =$
 $\mathcal{L}(P\text{-left-1}, CF-\lambda, ut) = \mathcal{L}(CF) \bmod \lambda.$

Theorem 5 • $\mathcal{L}(P\text{-left-2}, CF, ut) = \mathcal{L}_0,$
• $\mathcal{L}(P\text{-left-2}, CF-\lambda, ut) = \mathcal{L}_1 \bmod \lambda.$

(Immediate from known results [3].)

Remark: Except from this slide, language families are not considered to be equal modulo the empty word.

Some Recursion Theory

Let \mathcal{G} be some fixed grammar family, especially $\mathcal{G}_{\text{ut}[\text{,left-3}]}$, $\mathcal{G}_{\text{ac}[\text{,left-3}]}$, or \mathcal{G}_{kl} .
Problems related to main theorem:

- LSW: Given $G \in \mathcal{G}$, determine the **length** of the **shortest word** in $L(G)$, if $L(G) \neq \emptyset$.
- NSW: Given $G \in \mathcal{G}$, determine the **number** of the **shortest words** in $L(G)$, if $L(G) \neq \emptyset$, i.e.,

$$\text{NSW}(G) = |L(G) \cap V^{\text{LSW}}|.$$

- FMS: Given $G \in \mathcal{G}$, determine a **finite minimal set** (Higman set) for $L(G)$.

Obvious: $\text{LSW}, \text{NSW} < \text{FMS}$.

Rice's Theorem \Rightarrow All these problems are undecidable for $\mathcal{G}_{\text{ac}[\text{,left-3}]}$.

Theorem 6 *LSW is recursive for \mathcal{G}_{ut} , $\mathcal{G}_{\text{ut, left-3}}$ and \mathcal{G}_{kl} .*

Proof. \mathcal{G}_{kl} can be effectively transformed into \mathcal{G}_{ut} . So, let $G \in \mathcal{G}_{\text{ut}}$, $L(G) \subseteq V^*$, be given. First, check that $L(G) \neq \emptyset$. Then, the algorithm subsequently (for $n = 0, 1, 2, \dots$) explores (*) whether

$$L_n(G) := L(G) \cap \{w \in \Sigma^* \mid |w| \leq n\} \neq \emptyset.$$

Since $L(G) \neq \emptyset$, this checking procedure will terminate, hence yielding the smallest $n = \text{LSW}(G)$ such that $L_n(G) \neq \emptyset$.

How does the algorithm solve (*)? Consider $G = (V, V_T, P, S) \in \mathcal{G}_{\text{ut}}$. We construct $G' = (V', V_T, P', S) \in \mathcal{G}_{\text{ut}}$ with $L(G') = L_n(G)$.

Let H be a new symbol, $V' = V \cup \{H\}$, and h be a morphism that assigns to $w \in V_G^*$ the word H^m , where m equals the number of occurrences of terminals in w . P' consists of all rules $(r : A \rightarrow wh(w), \sigma(r) \cup \{t_1\})$, where $(r : A \rightarrow w, \sigma(r)) \in P$, and, in addition, of rules $(t_i : H \rightarrow \lambda, \{t_{i+1}\})$ for $i = 1, 2, \dots, n$ and of the rule $(t_{n+1} : H \rightarrow H, \{t_{n+1}\})$.

Same construction for $\mathcal{G}_{\text{ut, left-3}}$ is possible. □

Theorem 7 *NSW and FMS are undecidable for our grammar families. More precisely, for grammars of the type $\mathcal{G}_{\text{ut, left-3}}$, \mathcal{G}_{ut} and \mathcal{G}_{kl} , the problem to decide whether $\text{NSW}(G) \geq 2$ is many-one equivalent to the halting problem K for Turing machines.*

The proof is a combination of techniques developed in Theorem 3 together with the register machine simulations contained in [4, 6].

So, the information put into the grammar constructed in Theorem 3 cannot be retrieved algorithmically from some arbitrary grammar with unconditional transfer.

Summary and Open Problems

- $\mathcal{L}(P\text{-left-3}, CF, ut) = \mathcal{L}(P\text{-left-3}, CF, ac)$, while the strictness of the inclusion $\mathcal{L}(P, CF, ut) \subseteq \mathcal{L}(P, CF, ac)$ is unknown.
- $\mathcal{L}(P\text{-left-3}, CF-\lambda, ut) \subset \mathcal{L}(P\text{-left-3}, CF-\lambda, ac)$, while the strictness of the inclusion $\mathcal{L}(P, CF - \lambda, ut) \subseteq \mathcal{L}(P, CF-\lambda, ac)$ is unknown.
- What is the exact relation between:
 - $\mathcal{L}(P\text{-left-3}, CF-\lambda, ut)$ and $\mathcal{L}(P, CF-\lambda, ac)$ or $\mathcal{L}(P, CF - \lambda, ut)$ (It is known that $\mathcal{L}(P, CF - \lambda, ut)$ is not contained in $\mathcal{L}(P\text{-left-3}, CF-\lambda, ut)$ [18].),

Remark 1: Characterizations of UT obtained in [9] are also valid for left derivations.

Remark 2: Recently, many open problems concerning $\mathcal{L}(P\text{-left-3}, CF)$ have been solved [2].

Der Satz von Rosenkrantz

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

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

Der Beweis beruht auf folgender Aussage (siehe auch Lemma von König):

Satz 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:

$V = \{(\phi, v, r) \mid \phi \in V_G^{\leq |x|}, v, r \in Lab\}$.

potentielle Kanten, die (ϕ, v, r) und (ψ, u, q) verbinden:

$q \in \sigma(r)$, $\phi\xi$ wird durch r in $\psi\eta$ verwandelt mit $\Psi(\xi) = v$ 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 zitiertem Satz ist der so konstruierte Graph endlich.

Ferner gilt: x ist Präfix eines Wortes aus $L(G)$ gdw. Es gibt einen Weg von $(S, \vec{0}, r_0)$ nach $(x, \vec{0}, r)$ für ein Label r .

Literatur

1. S. Ábrahám. Some questions of phrase-structure grammars I. *Comput. Linguistics*, 4:61–70, 1965.
2. J. Dassow, H. Fernau, and Gh. Păun. On the leftmost derivation in matrix grammars. *International Journal of Foundations of Computer Science*, 10(1):61–80, 1999.
3. J. Dassow and Gh. Păun. *Regulated Rewriting in Formal Language Theory*, volume 18 of *EATCS Monographs in Theoretical Computer Science*. Springer, 1989.
4. H. Fernau. Membership for 1-limited ET0L languages is not decidable. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 30(4):191–211, 1994.
5. H. Fernau. A predicate for separating language classes. *EATCS Bulletin*, 56:96–97, June 1995.
6. H. Fernau. Membership for k-limited ET0L languages is not decidable. *Journal of Automata, Languages and Combinatorics*, 1:243–245, 1996.

7. H. Fernau. On grammar and language families. *Fundamenta Informaticae*, 25(1):17–34, 1996.
8. H. Fernau. On unconditional transfer. In W. Penczek and A. Szalas, editors, *MFCS'96*, volume 1113 of *LNCS*, pages 348–359, 1996.
9. H. Fernau. Unconditional transfer in regulated rewriting. *Acta Informatica*, 34:837–857, 1997.
10. H. Fernau. On regulated grammars under leftmost derivation. *GRAMMARS*, 3(1):37–62, 2000.
11. H. Fernau. Nonterminal complexity of programmed grammars. *Theoretical Computer Science*, 296:225–251, 2003.
12. S. Ginsburg and E. H. Spanier. Control sets on grammars. *Mathematical Systems Theory*, 2:159–177, 1968.
13. D. Hauschildt and M. Jantzen. Petri net algorithms in the theory of matrix grammars. *Acta Informatica*, 31:719–728, 1994.

14. G. Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society* (3), 2(7):326–336, 1952.
15. F. Hinz and J. Dassow. An undecidability result for regular languages and its application to regulated rewriting. *EATCS Bulletin*, 38:168–173, 1989.
16. A. Pascu and Gh. Păun. On the planarity of bicolored digraph grammar systems. *Discrete Mathematics*, 25:195–197, 1979.
17. Gh. Păun. Normal forms for bicolored-digraph-grammar systems. *International Journal of Computer Mathematics*, 7:109–118, 1979.
18. D. J. Rosenkrantz. Programmed grammars and classes of formal languages. *Journal of the ACM*, 16(1):107–131, 1969.
19. G. Rozenberg and A. K. Salomaa. Context-free grammars with graph-controlled tables. Technical Report DAIMI PB-43, Institute of Mathematics at the University of Aarhus, Ny Munkegade, DAN–8000 Aarhus C, January 1975.

20. A. K. Salomaa. On grammars with restricted use of productions. *Annales academiae scientiarum Fennicae*, Serie A-454:1–32, 1969.
21. A. K. Salomaa. Periodically time-variant context-free grammars. *Information and Control (now Information and Computation)*, 17:294–311, 1970.
22. A. K. Salomaa. *Formal Languages*. Academic Press, 1973.
23. E. D. Stotskii. Control of the conclusion in formal grammars. *Problemy peredachi informacii; translated: Problems of information transmission*, 7(3):257–270, 1971 (Translation 1973).
24. D. Wätjen. k -limited 0L systems and languages. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 24(6):267–285, 1988.
25. D. Wood. Bicolored digraph grammar systems. *RAIRO Informatique théorique et Applications/Theoretical Informatics and Applications*, 1:45–50, 1973.
26. D. Wood. A note on bicolored digraph grammar systems. *International Journal of Computer Mathematics*, 3:301–308, 1973.