Formale Sprachen

(parallele und regulierte Ersetzung)

SoSe 2008 in Trier

Henning Fernau
Universität Trier
fernau@uni-trier.de

Formale Sprachen Gesamtübersicht

- Organisatorisches / Einführung
  Motivation / Erinnerung / Fragestellungen

- Diskussion verschiedener Sprachklassen:
  gesteuerte Ersetzungsverfahren
  parallele Ersetzungsverfahren

- Algebraische Ansätze:
  abstrakte Sprachfamilien
  formale Potenzreihen

Formale Sprachen — Gesamtübersicht

- Organisatorisches / Einführung
  Motivation / Erinnerung / Fragestellungen

- Diskussion verschiedener Sprachklassen:
  gesteuerte Ersetzungsverfahren
  parallele Ersetzungsverfahren

- Algebraische Ansätze:
  abstrakte Sprachfamilien
  formale Potenzreihen

Parallele Grammatiken

- Lindenmayer-Systeme
- eingeschränkter Parallelismus

Erinnerung: *0L-Systeme*
Fortlassen der “Determinismus-Bedingung” bei D0L
Formal bedeutet dies, dass wir zu jeder linken Seite a eine nicht-leere, endliche Teilmenge $h(a)$ assoziieren.
$h : \Sigma \to 2^{\Sigma^*}$ lässt sich wieder als Morphismus
$h : \Sigma^* \to 2^{\Sigma^*}$ begreifen (Komplexprodukt).

Mit $G = (\Sigma, h, \omega)$ gilt dann:
Ableitungsbegriff: $x \Rightarrow y$ gdw. $y \in h(x)$ (uneingeschränkt parallele Ableitung)

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

Beispiel: Regelmenge $a \to a, a \to aa$, $\omega = a$, $L = \{a^? \mid \ldots\}$
Gibt es äquivalentes D0L-System ?

Adult-Sprachen

Die *Adult-Sprache* eines vorgelegten 0L-Systems $G = (\Sigma, h, \omega)$ ist:

$$A(G) = \{w \in L(G) \mid \forall w' \in \Sigma^* : (w \Rightarrow_G w') \rightsquigarrow w = w'\}.$$

Beispiel: Regelmenge: $S \to aSb$, $S \to \lambda$, $a \to a$, $b \to b$, S Axiom $\rightsquigarrow$ $A = \{a^n b^n \mid n \in \mathbb{N}\}$.

Das kleinste Teilalphabet Σ_A, über dem alle Wörter von $A(G)$ sind, heißt auch *Adult-Alphabet* von G.

Lemma: (Adult-NF) Zu jedem gegebenen 0L-System H kann ein 0L-System G konstruiert werden mit gleicher Adult-Sprache (und somit auch gleichem Adult-Alphabet Σ_A), so dass die einzige Regel für $a \in \Sigma_A$ die Regel $a \to a$ ist.

Adult-Sprachen

Satz: A0L-Sprachen sind genau die kontextfreien Sprachen.

Beweis: Wenn wir ein 0L-System in Adult-NF als kontextfreie Grammatik interpretieren, wird dieselbe Sprache beschrieben.

Umgekehrt können wir eine kontextfreie Grammatik mit lauter produktiven Symbolen als 0L-System lesen, wenn wir noch Regeln $a \rightarrow a$ für die Terminalzeichen einführen. Die Adult-Sprache jenes Systems ist gleich der vorgelegten kontextfreien Sprache.

Folgerung: E0L-Sprachen sind echte Obermenge von A0L-Sprachen.

Systeme mit Tafeln

Biologische Motivation:
Modellierung verschiedener Umweltbedingungen, beispielsweise Tag und Nacht oder langfristige Unterschiede in den Wachstumsbedingungen (Jahreszeiten).

Formal bedeutet dies: Ein *T0L-System* ist ein Quadrupel $G = (\Sigma, H, \omega)$ mit $H = \{h_1, \ldots, h_t\}$, sodass für jedes h_i gilt: $G_i = (\Sigma, h_i, \omega)$ ist 0L-System.
Ableitungsbegriff: $x \Rightarrow y$ gdw. $y \in h_i(x)$ für ein $1 \leq i \leq t$.

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

Also: $L(G) \supseteq \bigcup_{i=1}^{t} L(G_i)$; im Allg. strikt !

Varianten: ET0L, CT0L, AT0L

Normalformen für ET0L-Systeme

Lemma: Zu jedem ET0L-System kann man ein äquivalentes ET0L-System konstruieren, bei dem Terminalzeichen nur trivial ersetzt werden können, nämlich durch Regeln der Form $a \to a$.

Beweis: Hausaufgabe !

Der *Synchronisationsgrad* einer ET0L-Sprache ist die kleinste Anzahl Tafeln, mit ein ET0L-System jene Sprache beschreiben kann.

Satz: Der Synchronisationsgrad jeder ET0L-Sprache ist maximal zwei.

Normalformen für ET0L-Systeme

Satz: Der Synchronisationsgrad jeder ET0L-Sprache ist maximal zwei.

Beweisidee: Es sei L durch ein ET0L-System $G = (\Sigma, H, \omega, \Delta)$ beschrieben in der NF obigen Lemmas. $H = \{h_1, \ldots, h_t\}$ seien die Tafeln.
Es sei $[A, i]$ für $A \in \Sigma \setminus \Delta$ und $1 \leq i \leq t$ ein neues Alphabet.
Für $a \in \Delta$ seien $[a, i]$ alternative Schreibungen für a.

$$\Sigma' = \{[A, i] \mid A \in \Sigma, 1 \leq i \leq m\}.$$

Für ein Wort $x = \xi_1 \ldots \xi_m$ sei

$$[x, i] := [\xi_1, i] \ldots [\xi_m, i].$$

$G' = (\Sigma', H', [\omega, 1], \Delta)$ enthält zwei Tafeln:

- eine Simulationstafel mit $[A, i] \to [w, i]$ gdw. $A \to w \in h_i$ für $1 \leq i \leq t$ und
- eine Verteilertafel mit $[A, i] \to [A, (i \bmod n) + 1]$ für alle $1 \leq i \leq t$ und $A \in \Sigma$.

Frage: Wo geht genau das Lemma ein ?

Zusammenhänge der Sprachfamilien

Satz: $\mathcal{L}(\mathsf{ET0L}) = \mathcal{L}(\mathsf{AT0L})$. Tafeln “bringen” also viel für Adult-Sprachen.

Satz: $\mathcal{L}(\mathsf{EPT0L}) \subsetneq \mathcal{L}(\mathsf{O}, \mathsf{CF} - \lambda)$.

Hinweis: Simulation durch Sequentialisierung (Wie genau ?)

Aus einer kombinatorischen Eigenschaft von ET0L-Systemen folgt:

$L = \{(ab^m)^n c^n \mid m \geq n \geq 1\} \notin \mathcal{L}(\mathsf{ET0L})$.

Wie lässt sich die Sprache durch eine geordnete Grammatik beschreiben ?

Mehr zum Beweis (1)

Let $L \subseteq \Delta^*$ be an ET0L language. Let L be decomposed as

$$L = \bigcup_{a \in \Delta} aL_a \cup L_F \quad \text{where} \quad L_a = \{w \in \Delta^+ \mid aw \in L\}$$

is basically the left derivative of L under a and L_F is some finite language. As we already know, each L_a is in fact an EPT0L language.

We are going to show that each language aL_a is an ordered language, which, together with the closue of ordered languages under union proves the desired result.

Let L_a by described by an ET0L system $G_a = (\Sigma, H, \omega, \Delta)$ with $H = \{h_1, \ldots, h_t\}$. Let Σ' be an alphabet of primed symbols from Σ. We interpret $'$ also as homomorphism, which means that $' : A \mapsto A'$. Let S, F, A be three new symbols, the start symbol, the failure symbol and a symbol which will finally generate the leftmost a of aL_a. The nonterminal alphabet of the simulating ordered grammar $G'_a = (N, P, S, \Delta, <)$ is given by

$$N := \Sigma' \cup \{S, F, A\} \cup ((\Sigma \cup \{A\}) \times \{k \mid 1 \leq k \leq t\}).$$

The last bunch of symbols keep track of the currently simulated table.

Mehr zum Beweis (2)

We now describe the simulating rules together with their use in the simulation.

$$S \to (A, k)\omega' \quad \text{for } 1 \le k \le t$$

is a set of start rules. A simulation of table k ($1 \le k \le t$) is done by sequentialisation, requiring—as usual—a marking and a real application phase:

marking: $B' \to (B, k) \quad \left\langle \begin{array}{ll} (A, s) \to F & \text{for } 1 \le s \le t, s \ne k \\ A \to F \end{array} \right.$

the actual application of table h_k: $(B, k) \to w' \quad \left\langle \begin{array}{ll} (A, s) \to F & \text{for } 1 \le s \le t, s \ne k \\ A \to F \end{array} \right.$

dispatcher rule: $(A, k) \to (A, s) \quad < \quad (B, r) \to F \quad \text{for } 1 \le k, r, s \le t, B \in \Sigma.$

termination rules: $(A, k) \to A \quad \left\langle \begin{array}{ll} (B, \ell) \to F & \text{for } B \in \Sigma, 1 \le \ell \le t \\ B' \to F & \text{for } B \in \Sigma \setminus \Delta \end{array} \right.$

$b' \to b \quad < \quad (A, k) \to F \quad \text{for } b \in \Delta, 1 \le k \le t$

$A \to a \quad < \quad b' \to F \quad \text{for } b \in \Delta$

Partieller Parallelismus Formal sehen alle Systeme aus wie T0L systeme $G = (\Sigma, H, \omega)$, $H = \{h_1, \ldots, h_t\}$. Redefiniere "$\Rightarrow$":

Bharat (T0B) Systeme $x \Rightarrow y$ gdw. $\exists 1 \leq i \leq t\ \exists a \in \Sigma$: alle Vorkommen von a in x werden durch ein Wort in $h_i(a)$ ersetzt, um y from x zu erhalten;

k-limitierte (kIT0L) Systeme $x \Rightarrow y$ gdw. $\exists 1 \leq i \leq t\ \forall a \in \Sigma$: $\min\{|x|_a, k\}$ Vorkommen von a in x werden durch ein Wort in $h_i(a)$ ersetzt, um y aus x zu erhalten; ($|x|_a$ ist die Anzahl der Vorkommen von a in x.)

uniformly k-limitierte (ukIT0L) systems $x \Rightarrow y$ gdw. $\exists 1 \leq i \leq t$: $\min\{|x|, k\}$ Symbole in x werden gemäß h_i ersetzt, um y aus x zu erhalten.

Beispiele

$$G = (\{a, b\}, \{\{a \to aa, b \to ab\}\}, abb)$$

0L System $abb \Rightarrow aaabab \Rightarrow a^6aba^2ab.$

0B System $abb \Rightarrow aabb \Rightarrow aaabab \Rightarrow a^3abaab$

1I0L System $abb \Rightarrow aaabb \Rightarrow aaaabab \Rightarrow a^4ba^3b$

2I0L System $abb \Rightarrow aaabab \Rightarrow a^5abaab$

u2I0L System $abb \Rightarrow aabab \Rightarrow a^4bab \Rightarrow a^6bab$

Was ist $L(G)$ in jedem der Fälle ?

Limitierte Lindenmayer Systeme

Wir zeigen im Folgenden drei Hauptresultate:

A. Satz: Jede kIET0L Sprache ist eine 1IET0L Sprache.

B. Satz: (a) Jede kIET0L Sprache ist eine programmierte Sprache mit unbedingtem Übergang.
(b) Jede (P,CF,ut)-Sprache ist eine 1IET0L Sprache.
Entsprechendes gilt auch, wenn man löschende Regeln verbietet.

C. Satz: Es gibt nicht-rekursive kIDT0L Sprachen

Beweis zu Satz A.

Beweis: Idea: Sequentialisierung durch Markierung bei Simulierung von G durch G'.
Für jedes Symbol A von G führe markiertes Symbol $A[i,j]$ mit $1 \leq i,j \leq k$ ein.

Markierungstafel M_i, $1 \leq i \leq k$, mit folgenden Regeln:
$A \to A[i,i]$ und $A[i-1,j] \to A[i,j]$ für jedes Zeichen A und $j < i$;
alle anderen (markierten) Zeichen leiten das Fehlerzeichen F ab.

Ersetze jede Original-Tafel h durch eine simulierende Tafel h' mit Regel $A[k,j] \to w$, falls $A \to w \in h$, sowie $A \to A$. (Alle anderen Zeichen leiten F ab.)

Funktioniert das wirklich so?
Prüfe: $L(G) \subseteq L(G')$? & $L(G') \subseteq L(G)$?
Beh.: Fängt die Simulation mit M_1 an, dann folgen $M_2, M_3, \ldots, M_k, h'$.

Zwei häufige Techniken in diesem Gebiet:
Sequentialisierung der Regelanwendungen sowie
explizite Zustandsinformation.

Beweis zu Satz B.

Beweis: Zur Simulation von 1IET0L-Systemen mit programmierten Grammatiken mit unbedingtem Übergang führen wir zu jeder Tafel $\mathfrak{h}$ eine Simulationsschleife ein. Diese ermöglicht es, für jedes Zeichen $\mathfrak{a}$ eine Regel $\mathfrak{a} \to w$ von $\mathfrak{h}$ mithilfe von $\mathfrak{a}' \to w$ zu simulieren.
Außerdem gibt es eine Strich-Schleife in der für jedes $\mathfrak{a}$ eine Regel $\mathfrak{a} \to \mathfrak{a}'$ benutzt wird.
Unter Verwendung einer Normalform für 1IET0L-Systeme
(Wätjen: ausschließlich "Nichtterminal"-Symbole werden "nicht-konstant" ersetzt)
zeigt dies die Sequentialierungskonstruktion von Dassow.
Beobachte: Simulation von allg. kIET0L-Systemen durch progr. Grammatiken folgt mit Satz A.
Für die Gegenrichtung benutze explizite Zustandsinformation, gespeichert in Zustandssymbol.
Ist $(\mathfrak{p} : A \to w, \gamma)$ programmierte Regel, so führe eine Tafel ein mit einzigen Nicht-Fehler-Regeln $\mathfrak{p} \to \mathfrak{q}$ für jedes $\mathfrak{q} \in \gamma$ und $A \to w$.
In einer Terminierungstafel wird das Zustandssymbol gelöscht, falls ein Terminalwort (sonst) abgeleitet wurde.

Beweis zu Satz C.

Beweis: Problem: Wahl des “richtigen” Berechnungsmodells.

Hier: Variante von Registermaschinen.

Each register is capable of storing a non-negative integer.

Registers are labelled by positive integers.

The input is stored in the first register.

The output is expected to be found in the second register.

Beispiel: An example of a register machine program (RMP):

$$
\begin{array}{lcl}
L_1 & : & a_1 \\
L_2 & : & a_2 \\
L_3 & : & JZ_1 L_7 \\
L_4 & : & s_1 \\
L_5 & : & a_2 \\
L_6 & : & JNZ_1 L_4 \\
L_7 & : & \text{END}
\end{array}
$$

This example computes $x + 2$ for each input x; if the start were at L_3, the identity function would be computed.

RMP consists of a sequence of labelled commands

- for incrementing and decrementing* register numbered i (by the commands a_i and s_i, respectively),
- for jumping if register numbered i is zero or not (by JZ_i and JNZ_i, respectively),
- for indicating the END of an RMP.

*In RMP, a modified decrement is used: if a register containing zero is decremented, it will contain also zero afterwards.

If an RMP uses at most r registers and ℓ labels, we call it (r, ℓ)-RMP.

Let $k \geq 1$. A *kIT0L machine* is given by

$$M = (\Sigma, \{h_1, \ldots, h_t\}, \{\sigma, x, y, R\}, k),$$

where Σ and $\{h_1, \ldots, h_t\}$ are the total alphabet and the set of tables, respectively.
σ, x, y, R are special symbols in Σ.
We say that M computes the function $f : \mathbb{N} \multimap \mathbb{N}$ iff the corresponding kIT0L system $G_{M,n} = (\Sigma, \{h_1, \ldots, h_t\}, x^{kn}R\sigma, k)$ with axiom $x^{kn}R\sigma$ generates a word of the form $y^{km}\sigma$ if and only if $m = f(n)$.
Especially, there is at most one word in $\{y\}^*\{\sigma\} \cap L(G_{M,n})$.

Lemma: For any computable function $f : \mathbb{N} \multimap \mathbb{N}$ and any $k \geq 1$, there exists a kIT0L machine computing f.

Beweis: $f : \mathbb{N} \multimap \mathbb{N}$ can be described by an (r, ℓ)-RMP P. We describe a simulating kIT0L machine $M = (\Sigma, H, \{\sigma, x, y, R\})$ with

$$\Sigma = \{\sigma, F, R, S, x = A_1, \ldots, A_r, y, C_1, \ldots, C_r\} \cup L \cup L'.$$

F is a failure symbol; we list only productions which do not lead to F.
$L = \{L_1, \ldots, L_\ell\}$ is the set of labels controlling the simulation of the program P.
L' is a set of primed version of the labels in L.
We can assume that every jump statement in P is actually a sequence of two complementary statements: JZ_i lab_1 and JNZ_i lab_2 which carry the same register index i. This avoids implicit jumps.

The set of tables H consists of

- one initialization table h_I containing $\sigma \to \sigma$, $x \to x$ and $R \to C_1 \cdots C_r L_1$;
- two termination tables
  h_T with $\sigma \to \sigma$, $A_i \to \lambda$ for $i \neq 2$, $A_2 \to y$, $L_\ell \to S$, $S \to S$, $y \to y$, $C_i \to C_i$ $(i \leq r)$;

and h'_T with $\sigma \to \sigma$, $C_i \to \lambda$ for $i \leq r$, $S \to \lambda$, and $y \to y$.
This way, the result is transferred from register 2 into a sequence of y's.

- for any label L_j, there corresponds one or two simulation tables.
- $L_s : a_i$: $L_s \to L_{s+1}$, $C_i \to A_i^k C_i$, $C_j \to C_j$ for $j \neq i$, $A_j \to A_j$ for $1 \leq j \leq r$, $\sigma \to \sigma$.
- $L_s : s_i$: $L_s \to L_{s+1}$, $A_i \to \lambda$, $C_j \to C_j$ for $1 \leq j \leq r$, $A_j \to A_j$ for $j \neq i$, $\sigma \to \sigma$.
- $L_s : JZ_i L_t$: $L_s \to L_t$, $C_j \to C_j$ for $1 \leq j \leq r$, $A_j \to A_j$ for $j \neq i$, $\sigma \to \sigma$.
- $L_s : JNZ_i L_t$: There are two simulating tables here:
  - t_1: $L_s \to L'_s$, $A_i \to \sigma A_i$, $C_j \to C_j$ for $1 \leq j \leq r$, $A_j \to A_j$ for $j \neq i$, $\sigma \to \sigma$.
  - t_2: $L'_s \to L_t$, $\sigma \to \lambda$, $C_j \to C_j$ and $A_j \to A_j$ for $1 \leq j \leq r$.

Observe: we do not (cannot ?) bother about the sequence in which the symbols occur in a string derived via M. Nevertheless, the correctness of the construction is easily seen observing the special role of σ as a *success witness*.

Mithilfe des vorigen Lemmas folgt Satz C. (Übung) !

Satz: Für jedes $k \geq 1$ gilt $\mathcal{L}(O, CF) \subsetneq \mathcal{L}(k\text{IET0L})$.

Beweis: (Sketch of crucial simulation)
Alphabet of kIET0L system G' that simulates ordered grammar $G = (N, P, S, \Delta, <)$:

$$\Sigma = N \cup \Delta \cup \Delta' \cup \{\sigma, \chi, \alpha, F\}.$$

Priming is seen as a morphism keeping fixed symbols others than terminals.
Axiom of G': σS, σ indicates simulation mode.
Each rule $p : A \to w$ of G is simulated by two tables:

1. h_p containing $A \to \alpha^k w'$, $A \to A$, as well as $B \to F$ for each left-hand side B of any rule greater than $A \to w$ in G.
   Moreover, h_p contains rules $X \to X$ for the other nonterminals and rules $b' \to b'$ for the terminals b.
   $\sigma \to \chi$ introduces the checking mode marker;
   all other symbols are sent to the failure symbol F.

2. h'_p having only the following non-failure rules:
   $\chi \to \sigma$ and $\alpha \to \lambda$, as well as $X' \to X'$ for any nonterminal and terminal X of G.

$\mathfrak{h}'_{\mathrm{p}}$ checks that when applying $\mathfrak{h}_{\mathrm{p}}$, rule $A \to \alpha^{k} w'$ was applied at most once.

Termination table $\mathfrak{t}$ contains non-failure rules $\mathfrak{b}' \to \mathfrak{b}'$ and $\mathfrak{b} \to \mathfrak{b}$ for terminals $\mathfrak{b}$ and $\sigma \to \lambda$ to stop the simulation.

Explain: Purpose of primed terminals?!
Why simulation of ordered rule in two tables?!
Unfortunately, the above simulation does not carry over to the case of disallowing erasing rules in general. Why?

Uniform limitierte Systeme

Satz: (K. Salomaa / Wätjen) Die Sprachfamilien der k-uniform limitierten E0L-Systeme bilden eine unendliche Hierarchie:

$$\mathcal{L}(\mathrm{CF}) = \mathcal{L}(1\mathrm{ulE0L}) \subsetneq \mathcal{L}(2\mathrm{ulE0L}) \subsetneq \mathcal{L}(3\mathrm{ulE0L}) \subsetneq \cdots .$$

Satz: Für alle $k \geq 1$ gilt:

$$\begin{aligned} \mathcal{L}(k\mathrm{ulET0L}) &\subseteq \mathcal{L}(\mathrm{P}, \mathrm{CF}) \\ \mathcal{L}(k\mathrm{ulEPT0L}) &\subseteq \mathcal{L}(\mathrm{P}, \mathrm{CF} - \lambda) \end{aligned}$$

Beweis: (Idea contains (possibly) non-algorithmic elements in the erasing case!)
Given a kulET0L system $G = (\Sigma, H, \omega, \Delta)$, we can compute the finite set

$$L' := L(G) \cap \{w \in \Sigma^* \mid |w| \leq k\}.$$

Consider now a slight variant of uniform k-limitation—*exact uniform k-limitation*: $x \Rightarrow_{ex} y$ if y is obtained from x by replacing exactly k symbols in x. If $L_{ex}(H)$ denotes the language generated in this way by a kulET0L system H, then it is not hard to see that

$$L(G) = L_{ex}(G) \cup \bigcup_{w \in L'} L_{ex}(G[w])$$

where $G[w]$ is the system with axiom w (instead of ω).

Each *exact* uniformly limited system can be simulated by a programmed grammar without appearance checking by sequentialisation. $\mathcal{L}(P, CF)$ is closed under union, $\rightsquigarrow$ ✓.

Open: (1) Can L' be computed algorithmically?
(2) Furthermore, the strictness of the inclusions in the preceding theorem is unknown.

Aus geeigneten kombinatorischen Ergebnissen folgt:

Satz:

- Für alle $k \geq 1$ gilt: $\mathcal{L}(\text{kuIE0L}) \subsetneq \mathcal{L}(\text{kuIET0L})$ und
- für alle $k \geq 1$, $\mathcal{L}(\text{kIE0L}) \subsetneq \mathcal{L}(\text{kIET0L})$.

Ähnliche Ergebnisse gelten auch für propagierende und für deterministische Systeme.