Lernalgorithmen

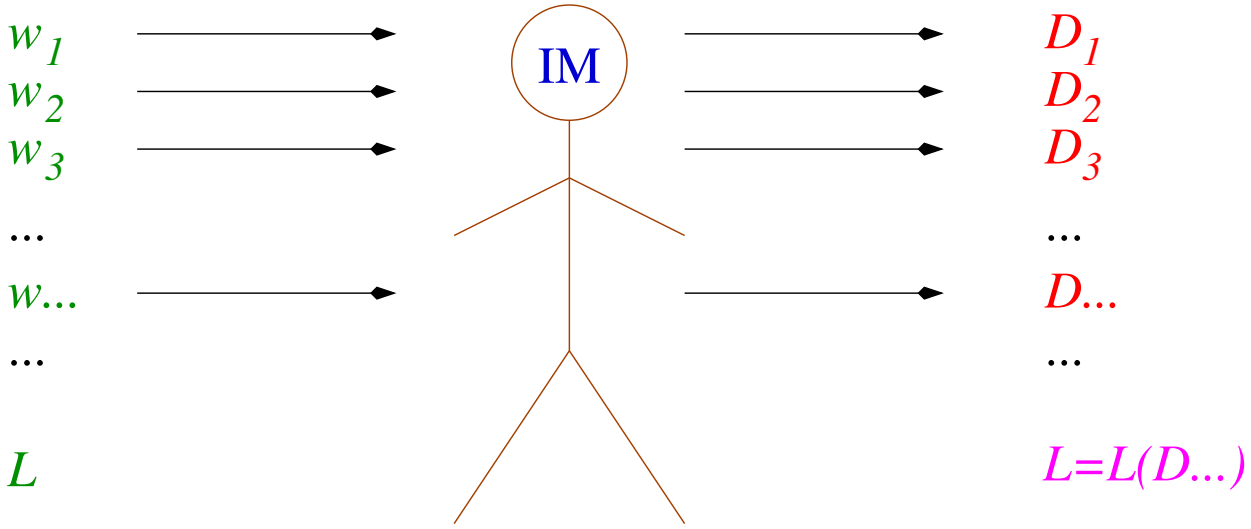
SoSe 2010 in Trier

Henning Fernau
Universität Trier
fernau@uni-trier.de

Lernalgorithmen Gesamtübersicht

0. Einführung
1. Identifikation (aus positiven Beispielen)
2. Zur Identifikation regulärer Sprachen, mit XML-Anwendung
3. HMM — Hidden Markov Models
4. Lernen mittels Anfragen & zur Roboterorientierung
5. Lernen mit negativen Beispielen
6. PAC-Lernen

Golds Lernmodell des Textlernens



Grammatikerzeugung für strukturierte Dokumente

als mögliches Anwendungsszenario:

1. Ahonens Ansatz für SGML
2. Zum direkten Textlernen regulärer Ausdrücke
3. Das MDL-Prinzip als alternatives Lernmodell

Grammatikerzeugung für strukturierte Dokumente

Ahonens Anwendungsfall: Automatische Inferenz grammatischer Struktur in Dokumenten.
Man betrachte folgendes Beispielfragment eines finnischen Wörterbuchs in SGML (Standard Generalized Markup Language)

⟨Entry⟩⟨Headword⟩kaame/a⟨/Headword⟩⟨Inflection⟩15⟨/Inflection⟩
⟨Sense⟩ kammottava, kamala, kauhea, karmea, hirveä ⟨/Sense⟩.
⟨Example_block⟩⟨Example⟩Kaamea onnettomuus, verityö. ⟨/Example⟩
⟨Example⟩ Tuliaseet tekivät kaameaa jälkeä. ⟨/Example⟩
⟨Example⟩ Kertoa kaameita kummitusjuttuja.⟨/Example⟩⟨/Example_block⟩
⟨Sense_structure⟩⟨Technical_field⟩Ark.⟨/Technical_field⟩
⟨Example_block⟩⟨Example⟩Kaamea hattu.⟨/Example⟩
⟨Example⟩ On kaamean kylmä.⟨/Example⟩ ⟨/Example_block⟩
⟨/Sense_structure⟩ ⟨/Entry⟩
⟨Entry⟩⟨Headword⟩kaameasti⟨/Headword⟩ ⟨Example_block⟩
⟨Example⟩Sireenit ulvoivat kaameasti.⟨/Example⟩⟨/Example_block⟩
⟨/Entry⟩
⟨Entry⟩⟨Headword⟩kaameus⟨/Headword⟩⟨Inflection⟩40⟨/Inflection⟩
⟨Example_block⟩⟨Example⟩ Sodan kaameus.⟨/Example⟩⟨/Example_block⟩
⟨/Entry⟩

SGML kurzgefasst

SGML ist eine *Markierungssprache* (markup language).
Ihre konkrete Spezifizierung erfolgt in einer sogenannten
Dokument-Typ-Definition (DTD).
Die in einem Dokument eines gewissen Typs zulässigen logischen Gliederungsteile heißen *Elemente*.
Elemente können in der Form erweiterter geklammmerter kontextfreier Grammatiken spezifiziert werden.

Inferenzaufgabe: Induktion rechter Seiten, also regulärartiger Ausdrücke, für jedes Element.

Ein DTD-Beispiel:

⟨!–	ELEMENTS	CONTENT	–⟩
⟨!ELEMENT	Entry	(Headword, Inflection?, Sense?, Example_block, Sense_structure?)	⟩
⟨!ELEMENT	Headword	(#PCDATA)	⟩
⟨!ELEMENT	Inflection	(#PCDATA)	⟩
⟨!ELEMENT	Sense	(#PCDATA)	⟩
⟨!ELEMENT	Example_block	(Example)*	⟩
⟨!ELEMENT	Sense_structure	(Technical_field, Example_block)	⟩

#PCDATA steht für “parsed character data”.

Modellgruppe in SGML (Definition) ähnlich zu regulären Ausdrücken

1. Ist e ein Elementname, so ist (e) Modellgruppe.

2. Sind A, B Modellgruppen, so auch:

   (A, B) , ist der *Sequenzkonnektor*, d.h. A wird von B gefolgt.

   $(A\&B)$ & ist der *Undkonnektor*, d.h. A wird von B gefolgt oder umgekehrt.

   $(A|B)$ | ist der *Oderkonnektor*, d.h. As oder Bs Spezifikation ist erfüllt.

   $(A?)$ *Optionales Element:* As Spezifikation wird erfüllt oder übergangen.

   $(A+)$ *wiederholbares Element:* As Spezifikation wird mind. einmal erfüllt.

   $(A*)$ *wiederholbares optionales Element*

Mögliche Anwendungen automatisch erzeugter DTDs

1. (Nachträgliche) Analyse “gewachsener”, “historischer” Dokumente, wo explizite DTD-Grammatiken (vermutlich) nie existiert haben oder aber nicht zugreifbar sind.

2. Werkzeug zur Unterstützung des DTD-Entwurfs für neue Dokumente.

3. Hilfe bei der Umstellung von nicht-SGML-konformen Textverarbeitungssystemen auf SGML-Standard.

4. Bei großen Dokumenten können für verschiedene Besuchergruppen unterschiedlich rigide DTD-Sichten (Views) entworfen werden.

5. Intelligente Zusammenstellung von einem individuellen Dokument aus mehreren existierenden. Dazu ist möglichst gute online-erstellte DTD vonnöten.

Eigenschaften bzw. Forderungen aus speziellen Anwendungen

1. Schnelligkeit der Algorithmen.
2. Lesbarkeit der DTDs für Menschen bzw. für Maschinen.
3. Batch- oder Dialogbetrieb
4. Vollständige Lösung erforderlich?
5. Möglichst reichhaltige Struktur in DTD abzubilden?
6. Kann das (SGML-) Dokument geändert werden, um “bessere” DTD zu ermöglichen?

Ahonens Lösungsvorschlag

Ahonen schlägt sogenannte (k, h)-kontextuelle Sprachen, eine Erweiterung der k-reversiblen Sprachen, vor, die ihres Erachtens gut für den skizzierten Anwendungsfall geeignet sind. Wir wollen hierauf nicht im Detail hier eingehen.

Problem: Konversion von erhaltenem DEA in lesbaren regulären Ausdruck !

Nähere Informationen in: H. Ahonen, H. Mannila, and E. Nikunen. Generating grammars for SGML tagged text lacking DTD. *Mathematical and Computer Modelling*, 26:1–13, 1997.

Hauptproblem des Ansatzes: “Aufblähen” der Ausdrücke durch Umweg über endliche Automaten.

XML in a Nutshell

- Teilmenge von SGML (Standard Generalized Markup Language, ISO Standard 8879)
- erweitert HTML
- Wohlgeformte XML Dokumente
- Valide XML Dokumente

Extensible Markup Language (XML) 1.0,
Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen, 10 February 1998.
Available at http://www.w3.org/TR/REC-xml

Buchhandlungen: ein einfaches XML Beispiel

```
<book>
   <author><last-name>Abiteboul</last-name></author>
   <author><last-name>Vercoustre</last-name></author>
   <title>Research and Advanced Technology for Digital Libraries. ...</title>
   <price>56.24 Euros</price>
</book> <book>
   <author><last-name>Thalheim</last-name></author>
   <title>Entity-Relationship Modeling. ...</title>
   <price>50.10 Euros</price>
</book>
```

wohlgeformt: richtige Tag-Klammerstruktur

valide / gültig: bzgl. Dokumenttypdefinition (DTD), einer besonderen Form einer kontextfreien Grammatik.

Grammatische Inferenz / Grammatik-Induktion (GI)

Ziel: Ermittele DTDs (oder Grammatiken, Automaten, ...) in einer "Standard-form" aus gegebenen Beispielen (vielleicht mit weiteren Informationen)

Hauptproblem: Wann sollte ein Inferenzalgorithmus anfangen zu "lernen", also zu "verallgemeinern" (und nicht nur zu reproduzieren)?
(Darin steckt auch die eigentliche "Intelligenz.")

Anwendungsszenarien für das Lernen von DTDs

- XML Dokumente könnten keine “gute” DTD besitzen.

- XML Dokument-Schreiber haben möglicherweise mangelnde Fertigkeiten im Entwurf von Grammatiken (DTDs)
  $\leadsto$ Aufgabe: Inferenz von DTDs aus wohlgeformten Beispieldokumenten
  Hierzu gibt es auch kommerzielle Tools.
  Wir werden auf einen Ansatz im Folgenden näher eingehen.

Warum DTD-Inferenz ? eine “offizielle” Begründung

1998 schrieb Tim Bray `www.xml.com/axml/notes/Any1.html`:

Suppose you’re given an existing well-formed XML document and you want to build a DTD for it. One way to do this is as follows:

1. Make a list of all the element types that actually appear in the document, and build a simple DTD which declares each and every one of them as ANY. Now you’ve got a DTD (not a very useful one) and a valid document.

2. Pick one of the elements, and work out how it’s actually used in the document. Design a rule, and replace the ANY declaration with a . . . content declaration. This, of course, is *the tricky part*, particularly in a large document.

3. Repeat step 2, working through the elements one by one, until you have a useful DTD.

XML Grammatiken

Die abstrakte DTD

$$
\begin{array}{l}
< !\texttt{DOCTYPE}\ a\ [\\
\qquad\qquad < !\texttt{ELEMENT}\ a \quad ((a|b),(a|b)) > \\
\qquad\qquad < !\texttt{ELEMENT}\ b \quad (b)* > \\
\qquad] >
\end{array}
$$

kann als eine sog. *XML Grammatik* (Berstel/Boasson 2002) notiert werden:

$$
\begin{array}{lcl}
X_a & \rightarrow & a(X_a|X_b)(X_a|X_b)\bar{a} \\
X_b & \rightarrow & b(X_b)^*\bar{b}
\end{array}
$$

mit Axiom (Startzeichen) X_a.

Überstriche vermitteln zwischen öffnenden und schließenden Klammern.
Die rechten Seiten entsprechen regulären Ausdrücken.

Zurück zum Buch...

Nonterminale für “Tags”:
X_b entspricht `<book>` und `</book>`,
X_a entspricht `<author>` und `</author>`,
X_n entspricht `<last-name>` und `</last-name>`,
X_t entspricht `<title>` und `</title>`,
X_p entspricht `<price>` und `</price>`.

Die Beispielmenge für den Lerner wäre also: $I_+^b = \{aatp, atp\}$.
Ein möglicher Ausdruck: $S^b = a^*tp$.
$\rightsquigarrow$ DTD-Vorschlag: `<!ELEMENT book (author*,title,price) >`

Versuch eines direkten Lerners für reguläre Ausdrücke (F. ALT 2005)

Grund-Blöcke des Verfahrens

Beispieleingaben

$$\mathtt{abab}, \mathtt{aabb}, \mathtt{abaa}, \mathtt{abc}$$

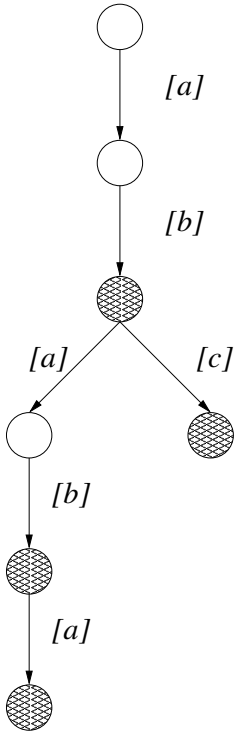
werden umgeformt in Wörter mit *Blockbuchstaben*:

$$[\mathtt{a}][\mathtt{b}][\mathtt{a}][\mathtt{bb}], [\mathtt{aa}][\mathtt{bb}], [\mathtt{a}][\mathtt{b}][\mathtt{a}][\mathtt{b}][\mathtt{a}], [\mathtt{a}][\mathtt{b}][\mathtt{c}]$$

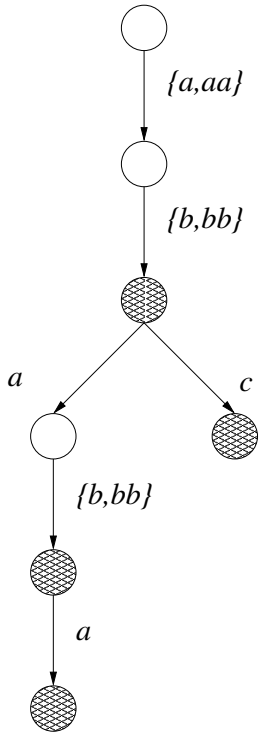
Blockbuchstaben werden dann linksbündig geschrieben (aligniert):

[a] [b] [a] [bb]
[aa] [bb]
[a] [b] [a] [b] [a]
[a] [b] [c]

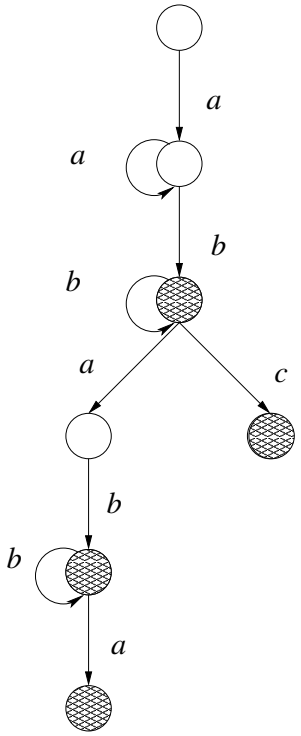
Der sich ergebenene Präfixakzeptor



Hieraus: ein **einfacher NEA** für $(a|aa)(b|bb)(\lambda|a(b|bb)(\lambda|a)|c)$.



Wir brauchen Kreise: aa^* anstelle von a, aa etc.



Sprache: $aa^*(bb^*(\lambda|a(bb^*(\lambda|a))|c))$

Die Sprachklasse $\mathcal{SL}$, DEA-Definition

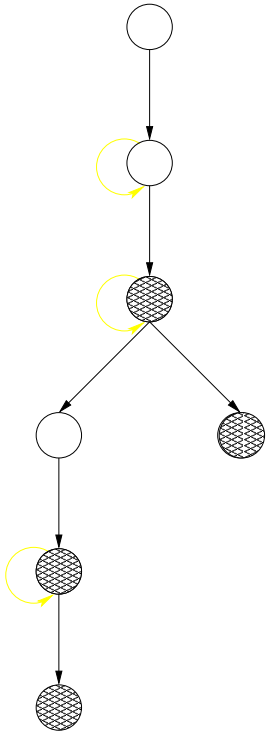
Ein DEA heiße *einfach kreisend* (*simple looping* $\mathcal{SL}$) gdw. folgende drei Bedingungen gelten:

- Entfernt man alle Kreise und Kantenbeschriftungen aus dem Automatengraphen, so ist der sich ergebene *Skelettgraph* ein gerichteter Baum ohne Mehrfachkanten und Schlingen.

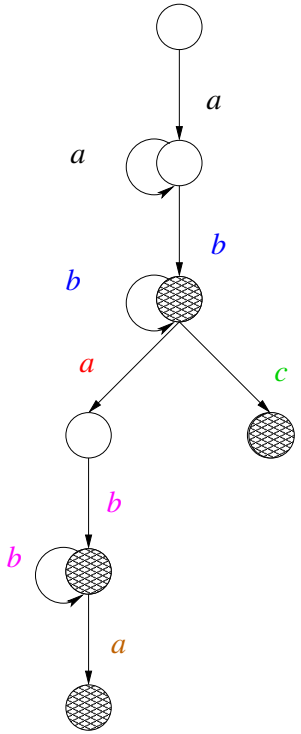
- Alle Nicht-Schlingenkanten, zu denen ein gewisser Knoten inzident sind, tragen paarweise unterschiedliche Beschriftungen.

- Ist a Schlingenbeschriftung am Knoten s, so ist a ebenfalls die Beschriftung der notwendig existierenden zweiten, nach s zeigenden Kante.

Ein Skelettgraph ohne Schlingen.



Kantenbeschriftungsbedingungen für $\mathcal{SL}$



Der Lernalgorithmus

1. Aligniere die Blockbuchstabensprache der Beispielmenge I_+ linksbündig.
2. Konstruiere hieraus Präfixakzeptor für die vereinfachte Blocksprache.

"Vereinfacht" soll heißen: Kein Unterschied z.B. zwischen $[a]$ und $[aa]$.

3. Konstruiere hieraus NEA durch Ersetzen der Blöcke durch die Wörter, die sie repräsentieren (können).
4. Ersetze Kante von u nach v, die mit Sprache $\{a^i \mid i \in I\}$ (mit $|I| > 1$) beschriftet ist, durch
   (a) Kante von u nach v, die mit a beschriftet ist.
   (b) Kante von v nach v, die mit a beschriftet ist.
5. Liefere den so erhaltenenen DEA zurück.

Eine charakteristische Menge für einfach kreisende DEA A

Assoziiere zu jedem Endzustand s das Wort w_s, gelesen als Beschriftung des eindeutig bestimmten Pfades vom Anfangszustand s_0 von A nach s (im Skelettgraphen).
Alle diese w_s bilden die Menge L_F; die Menge aller Zustände, durch die Wörter w_s führen, ist die Gesamtzustandsmenge S von A.
Für Nicht-Endzustände s bezeichne w_s irgendein Wort aus L_f, welches durch s führt.
u_s beschreibt den Präfix von w_s, der von s_0 nach s führt. v_s ist gegeben durch $w_s = u_s v_s$.
Hat A eine mit a beschriftete Schlinge bei Zustand s, setze $w_s^\circ = u_s a v_s$. L° sammle all solche Wörter w_s° auf.

Lemma. $\chi(L) = L_F \cup L^\circ$ ist eine charakteristische Menge von L.

Bem. $\chi = \{ab, abc, abab, ababa, aab, abb, ababb\}$ in unserem Beispiel

Abschließendes zum Lernen von regulären Ausdrücken

Aus einem gegebenen einfach kreisenden DEA kann man einen äquivalenten regulären Ausdruck einfach ablesen:

Beispiel: $\mathrm{a}\mathrm{a}^*(\mathrm{b}\mathrm{b}^*(\lambda|\mathrm{a}(\mathrm{b}\mathrm{b}^*(\lambda|\mathrm{a}))|\mathrm{c}))$ (s.o.)

Satz: $\mathcal{SL}$ ist textlernbar.

Hinweis: Man kann $\mathcal{SL}$ leicht modifizieren, um weitere textlernbare Sprachfamilien zu erhalten.

Anwendung des MDL Prinzips auf die DTD Inferenz

genauer: Inferenz von regulären Ausdrücken.

Garofalakis et al. 2003 XTRACT; Witten et al. 1994.

Die Beschreibung besteht aus zwei Teilen:

1. der Länge der Theorie (in Bits) und

2. der Länge der Daten (in Bits), kodiert mit Hilfe der gewählten Theorie.

Bsp.: $\{ab, abb, abbb, abbbb, abbbbb\}$ bei Theorie ab^+ codierbar durch $\{1, 10, 11, 100, 101\}$.

MDL Trade-off gemäß Solomonoff (1997):

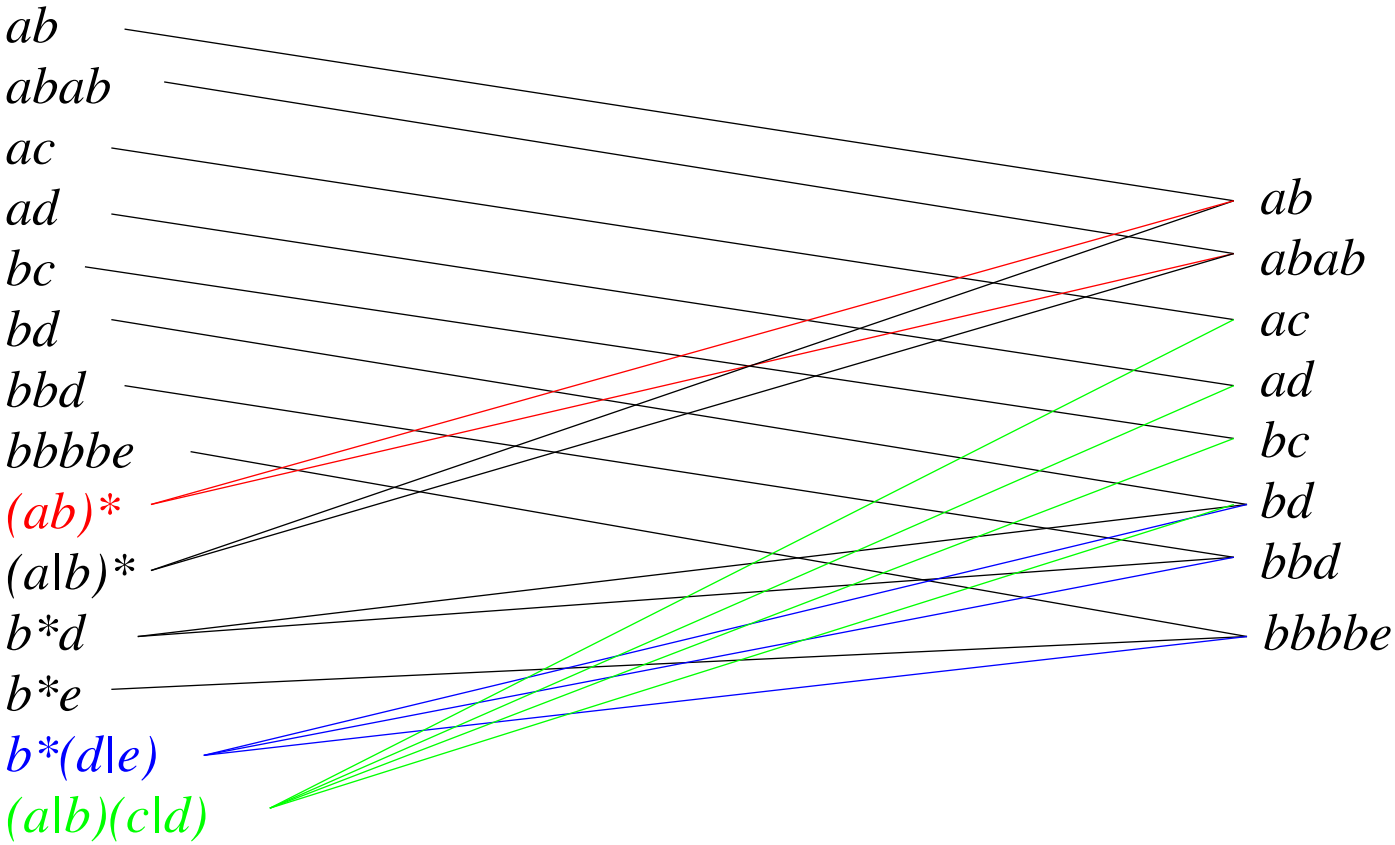
“I was trying to find an algorithm for the discovery of the ‘best’ grammar for a given set of acceptable sentences. One of the things I sought was: Given a set of positive cases of acceptable sentences and several grammars, any of which is able to generate all the sentences, what goodness of fit criterion should be used?

It is clear that the ad hoc grammar, that lists all of the sentences in the corpus, fits perfectly.

The promiscuous grammar, that accepts any conceivable sentence, also fits perfectly.

The first grammar has a long description; the second has a short description. It seemed that some grammar half-way between these, was ‘correct’—but what criterion should be used?”

Ein abstraktes Beispiel



MDL als ein kombinatorisches Problem

MDL-OPTIMAL-CODING

Gegeben:

- Eine Menge $R = \{r_1, \ldots, r_n\}$ regulärer Ausdrücke (über Grundalphabet Σ),
- eine Menge von Wörtern $S = \{s_1, \ldots, s_m\} \subset \Sigma^+$,
- und eine positive ganze Zahl k.

Frage: Gibt es eine Teilmenge R' von R, sodass

$$\sum_{r \in R'} |c(r)| + \sum_{s \in S} |c(s|R')| \leq k \quad ?$$

Thm.: MDL-OPTIMAL-CODING ist NP-vollständig.

Mehr zur Kodierung mit regulären Ausdrücken

Es seien ein Wort s und ein regulärer Ausdruck r gegeben mit $s \in L(r)$; definiere $c(s|r)$:

- $c(s|s) = \lambda$ (das leere Wort),
- $c(s|r_1 \cup \cdots \cup r_m) = b(i)c(s|r_i)$, falls $s \in L(r_i)$,
- $c(\lambda|r^*) = c(\lambda|r?) = 0$,
- $c(s|r?) = 1c(r)$ falls $s \neq \lambda$,
- $c(s_1 \ldots s_k|r^*) = c(s_1 \ldots s_k|r^+) = 1^{\log_2 k}0b(k)c(s_1|r) \ldots c(s_k|r)$ falls $k > 0$ und $s_i \in L(r)$,
- $c(s_1 \ldots s_k|r_1 \ldots r_k) = c(s_1|r_1) \ldots c(s_k|r_k)$ für $k > 1$ falls $s_i \in L(r_i)$.

Beispiel-Kodierungen

$$
\begin{aligned}
c(ab|(ab)^*) &= 101 \\
c(abab|(ab)^*) &= 11010 \\
c(ac|(a \cup b)(c \cup d)) &= 00 \quad \text{ähnlich } ad, bc, bd \\
c(bd|b^*(d \cup e)) &= 1010 \\
c(bbd|b^*(d \cup e)) &= 110100 \\
c(bbbbe|b^*(d \cup e)) &= 11101001
\end{aligned}
$$

MDL als kombinatorisches Problem MDL-OPTIMAL-CODING

Als Parameter wählen wir eine Obergrenze auf die Länge der Codierung.

Satz: MDL-OPTIMAL-CODING ist NP-vollständig. (F. ICGI 2004)

Satz: MDL-OPTIMAL-CODING ist in *FPT*. (Fellows&F AAIM 2008)

Beachte: Genauer gilt:
MDL-OPTIMAL-CODING mit $\mathfrak{n}$ Beispielen kann in Zeit $\mathcal{O}^*(2^{\mathfrak{n}})$ gelöst werden.

Hinweis: Ähnliche Probleme in vielen Bereichen der Datenkompression.

Hinweis: Es gibt regelmäßig Spezialvorlesungen über parameterisierte Algorithmen und über Datenkompression.