Parsingkonzept

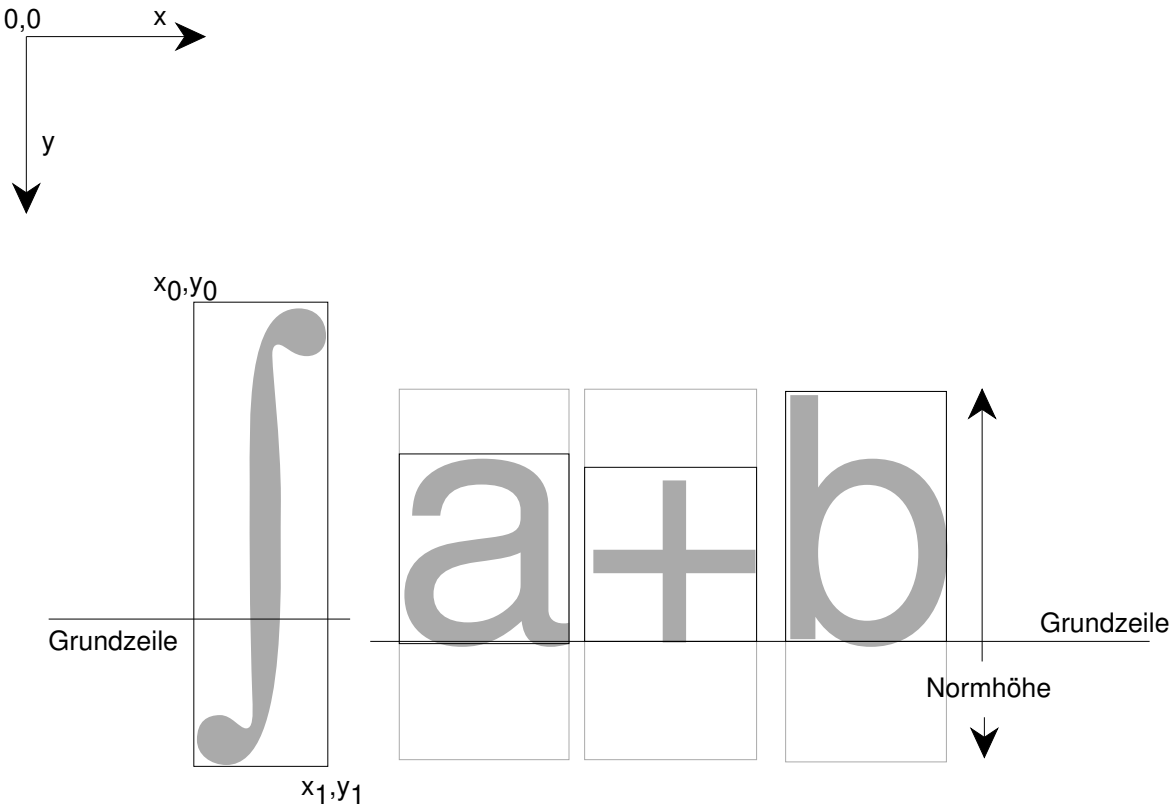
für

unscharf 2-dimensional attributierte Grammatiken

Frank Weigel

Ausgangssituation im Projekt

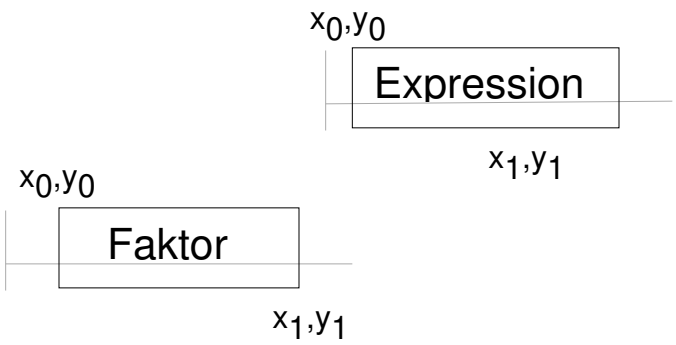
- Nach der zeichenweisen Erkennung der handschriftlichen Eingabe liegen die erkannten Zeichen sowie Informationen zu ihrer räumlichen Lage in Form umfassender Rechtecke vor.



- Zur Beschreibung der erlaubten Eingabesprache dient eine attributierte kontextfreie Grammatik.
  Verwendete Attribute sind:
  - die Koordinaten **x0,y0,x1,y1** des umfassenden Rechtecks
  - die daraus (und aus Zusatzwissen über das Alphabet) berechneten Attribute **Grundzeile** und **Normhöhe.**
    Das Attribut Normhöhe soll den Größenvergleich von Zeichen mit und ohne Ober-/Unterlänge ermöglichen.

  Die Attributierung einer Regel beschreibt, welche räumlichen Anordnungen für die Komponenten dieser Regel zulässig sind.

Als Beispiel die Regel für die Exponentiation:

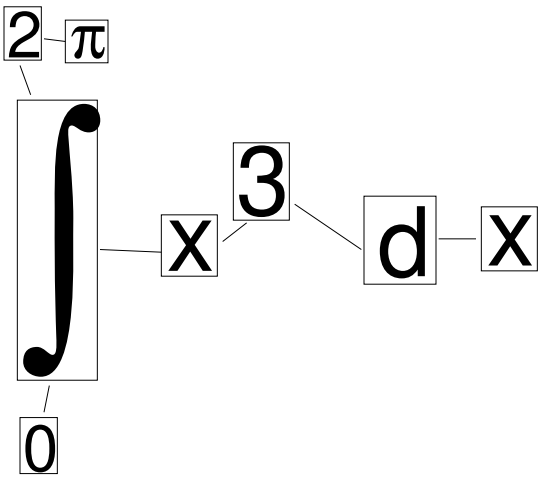


mit den zugehörigen Kontextbedingungen:

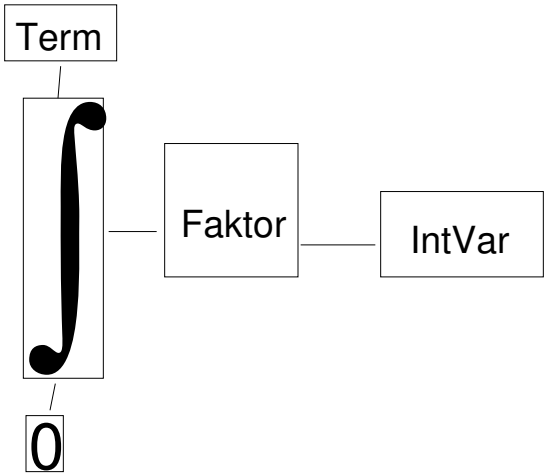
x0(Expression)	≥	x1(faktor) - X_OVERLAP_TOL
x0(Expression)	≤	x1(faktor) + X_DIST_TOL
Grundzeile(Expression)	≤	Grundzeile(Faktor) - 1/3*Normhöhe(Faktor) + Y_OVERLAP_TOL
Grundzeile(Expression)	≥	Grundzeile(Faktor) - 1/3*Normhöhe(Faktor) - Y_DIST_TOL

Bisheriges Parsingkonzept

Das bisherige Parsingkonzept bestimmt in einem ersten Schritt anhand der Schreibreihenfolge einen sog. Grundgraphen, dessen Kanten ausgewählte Nachbarschaften repräsentieren.



In einem zweiten Schritt wird dann entlang der Kanten dieses Grundgraphen eine Graphreduktion durchgeführt, indem die Umgebung noch bestehender Kanten daraufhin getestet wird, ob sie die Kontextbedingungen einer der Regeln der Grammatik erfüllt. Trifft dies zu, wird der entsprechende Teilgraph durch die linke Seite der Regel ersetzt.



Fehler des alten/Zielsetzungen des neuen Parsers:

- die bisherige Formulierung der Kontextbedingungen einer Regel zieht scharfe Grenzen zwischen erlaubten und unerlaubten Anordnungen der Komponenten einer Formel. Das entspricht nicht der menschlichen Anschauung.

- fehlende oder falsch gesetzte Kanten im Grundgraphen verhindern eine Erkennung der Formel, obwohl die eingegebenen Zeichen alle Kontextbedingungen der attributierten KfG erfüllen.
  Ziel ist daher die Abkehr von der Graphreduktion als Grundlage des Parsing. Alleine die Kontextbedingungen sollten entscheiden, ob eine Formel gültig ist oder nicht. Zusätzlich erlaubt dies eine Verwendung des Parsers für Zeichenmengen, bei denen die zeitliche Reihenfolge unbekannt ist (z.B. gescannte Vorlagen)

- die bislang existierende Version des Parsers macht einmal durchgeführte Reduktionen des Graphen nicht rückgängig. D.h. es kann eine syntaktisch mögliche und nach den Kontextbedingungen korrekte, im Formelzusammenhang aber falsche Reduktion durchgeführt werden, die eine erfolgreiche Reduktion der Gesamtformel unmöglich macht. Und dies, obwohl alle notwendigen Kanten im Grundgraphen existieren und obwohl die eingegebenen Zeichen alle Kontextbedingungen der attributierten KfG erfüllen könnten.

- leichte Änder-/Erweiterbarkeit der Grammatik

(strikte Trennung von Parsing-Algorithmus und Grammatik)

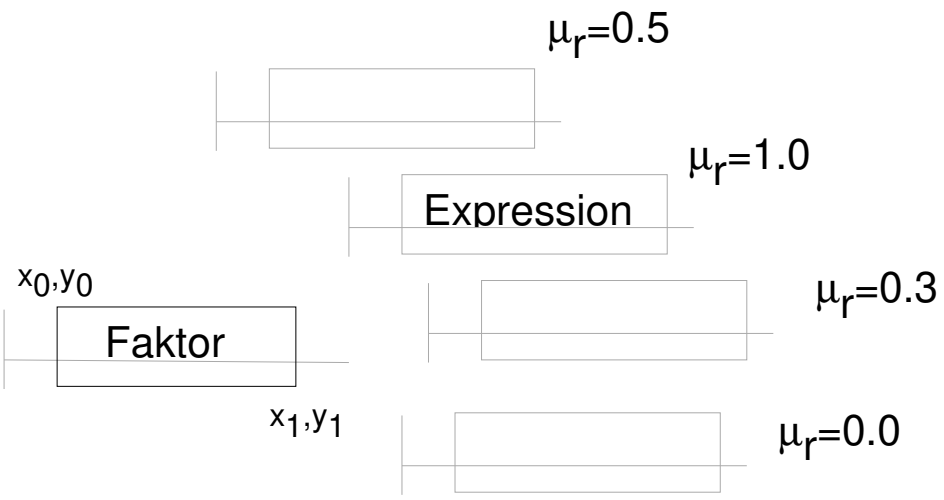
<u>Änderungen in der Attributierung der Grammatik</u>

- Die Attribute eines Ableitungsbaumes werden um das Attribut **Güte** $q \in [0..1]$ erweitert.

- statt der Kontextbedingungen in Form von Ungleichungen, existiert zu jeder Regel

  r: $N_0 \leftarrow N_1 \ldots N_n$ $\qquad\qquad$ $N_0 \in N,\ N_1 \ldots N_n \in N \cup \Sigma$

  eine Funktion μ_r: $A^n \rightarrow [0..1]$ (A ist die Menge aller möglichen Attributwerte, also z.B. $Z^6 \times [0..1]$ für umfassendes Rechteck, Grundzeile, Normhöhe und Güte), die angibt "wie gut" eine Konfiguration von n gegebenen Attributwerten (also u.A. die räumliche Anordnung) zu einer Regel paßt.



- wird eine solche Regel r angewandt, d.h. werden Ableitungsbäume $t_1,\ldots,t_n$ durch einen neuen Baum t_0 ersetzt, so berechnet sich die Güte des neuen Baumes zu

  $q(t_0) = (1+\varepsilon) * \min(\mu_r(ATTR(t_1),...,ATTR(t_n)), q(t_1), ..., q(t_n))$

  Der Korrekturfaktor $(1+\varepsilon)$ soll die durch die Minimumbildung verursachte kontinuierliche Güteabnahme ausgleichen, damit hohe Ableitungsbäume nicht benachteiligt werden

Bottom-Up Parsing-Strategie, allgemeine Formulierung:

AB bezeichnet die Menge der aktuell erzeugten Ableitungsbäume

(S1) $AB = \emptyset$
berechne die Attribute für die Zeichen der Eingabe

wiederhole

(S2) wähle Regel $r = N_0 \leftarrow N_1 \ldots N_n$

(S3) finde Menge von n disjunkten Ableitungsbäumen $\{t_1,\ldots,t_n\} \subseteq AB$, die syntaktisch auf r passen, d.h.
$TYP(t_i) = N_i \quad \forall i, 1 \leq i \leq n$

(S4) berechne $\mu_r(ATTR(t_1),\ldots,ATTR(t_n))$

(S5) falls $\mu_r > \mu_{threshold}(=0)$, dann erzeuge einen neuen Ableitungsbaum t_0 vom Typ N_0 und füge ihn zu AB hinzu

(S6) bis kein neuer Test mehr möglich ist

Alle Ableitungsbäume t_{end}, für die
$TYP(t_{end}) = S$ (Startsymbol der Grammatik) und
$|BLATTWORT(t_{end})| = |Eingabe|$
sind gültige Interpretationen der Formel und werden ausgegeben.

Nachteile dieser allgemeinen Strategie:

- hoher Suchaufwand zur Bestimmung der n disjunkten Ableitungsbäume $\{t_1,\ldots,t_n\}$ in Schritt (S3)
- Buchführung über die bereits getesteten Kombinationen ist kostenintensiv
- die Strategie ist nicht zielgerichtet

  Die *nichtdeterministische* Version des Parsers, die nur die notwendigen Reduktionen durchführt, stellt die anzustrebende ideale Parsing-Strategie dar.

Minimale Forderungen für einen "fast" idealen Parser sind daher:

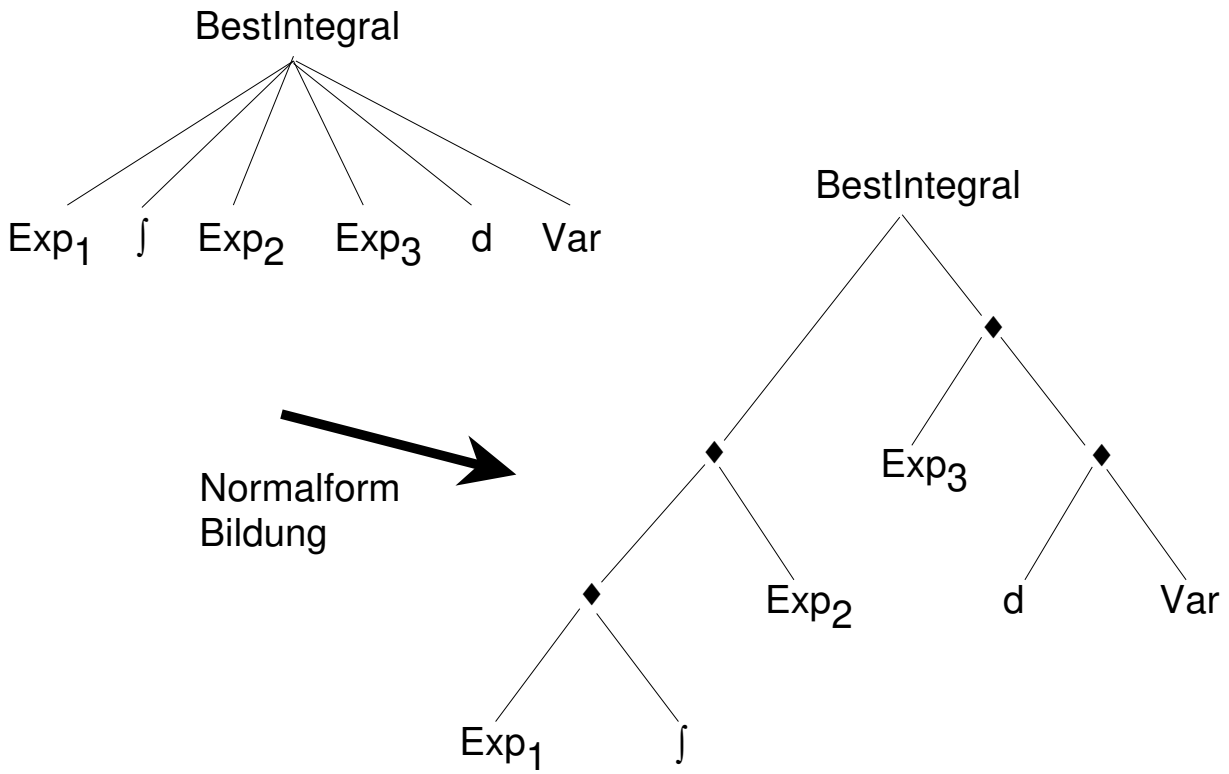
- Reduktion der Komplexität von Suche und Buchführung
- schnelle Antwortzeit für ideale Schreibweisen, d.h. für $q(t_{end}) >> 0$
- längere Antwortzeiten für Problemfälle, dh. für $q(t_{end}) << 1$

<u>erste Vereinfachung:</u>

Grammatik in Chomsky-Normalform bringen, d.h. es sind nur noch Regeln der Form

$A \leftarrow x \qquad A \in N,\ x \in \Sigma$
$A \leftarrow BC \qquad A,B,C \in N$

zugelassen.



Dies setzt eine entsprechende Umformulierung der Funktion $\mu_r()$ voraus.

Vorteile:

- die unären Regeln A ← x können im Vorverarbeitungsschritt (S1) des allgemeinen Algorithmus behandelt werden, d.h. alle Terminale werden zu (evtl. mehreren) Ableitungsbäumen der Höhe 1 umgewandelt und zu AB hinzugefügt.

- die zentrale Schleife des Parsing hat nur noch mit binären Regeln der Form A←BC zu tun. Dadurch reduziert sich die Suche in Schritt (S3) auf die Suche von Paaren.

- da alle Regeln 2-stellig sind, können für ein einmal gefundenes Paar von disjunkten Ableitungsbäumen alle Regeln getestet werden.

Modifizierter Algorithmus:

(S1') AB = ∅
berechne anhand der unären Regeln die Ableitungsbäume für die Zeichen der Eingabe

wiederhole

(S2') finde Paar disjunkter Ableitungsbäume $(t_1,t_2) \subseteq AB$

wiederhole für alle binären Regeln r

(S3') falls Regel
r: $N_0 \leftarrow N_1\ N_2$
syntaktisch auf (t_1,t_2) paßt, d.h.
$TYP(t_i) = N_i,\ i=1,2$
dann

(S4') berechne $\mu_r(ATTR(t_1),ATTR(t_2))$

(S5') falls $\mu_r > \mu_{threshold}(=0)$, dann erzeuge einen neuen Ableitungsbaum t_0 vom Typ N_0 und füge ihn zu AB hinzu

(S6') bis kein neuer Test mehr möglich ist

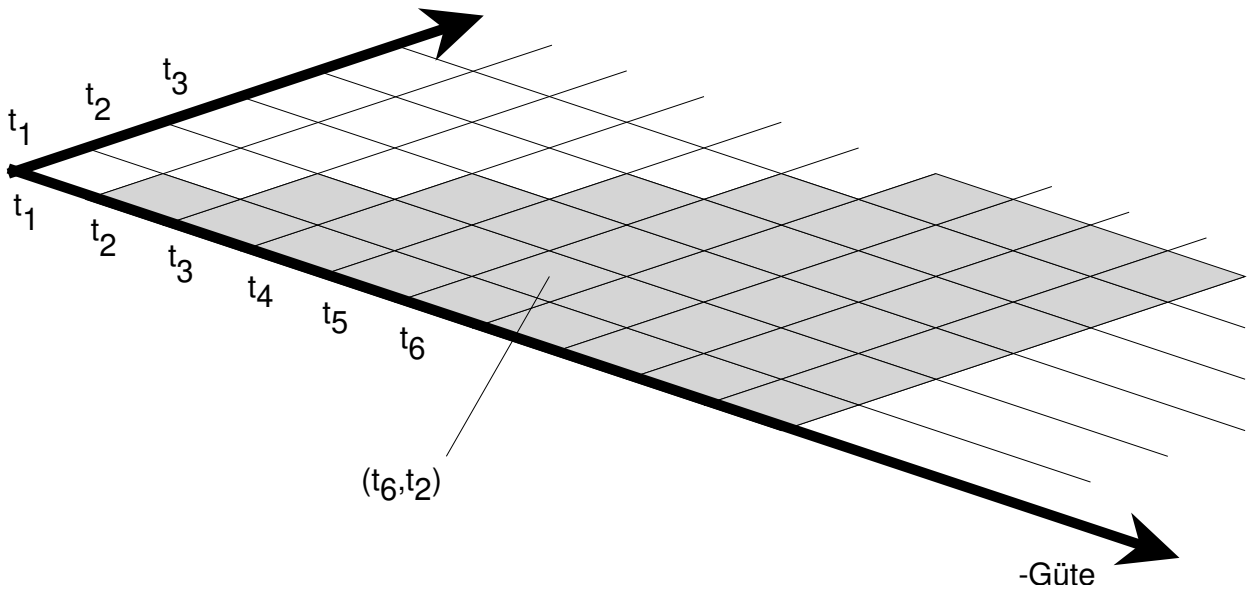
Einführung einer Suchstreategie:

Um "ideale" Schreibweisen zu bevorzugen, betrachte von allen gerade denkbaren Paaren zunächst dasjenige, das die höchste Eingangsgüte hat. Mit der Eingangsgüte eines Paares (t_i,t_j) ist dabei min($q(t_i)$, $q(t_j)$) gemeint.

Dazu:

- definiere mit Hilfe der Güte eine lineare Ordnung auf den Ableitungsbäumen und halte die Bäume absteigend sortiert
- zur Vermeidung der symmetrischen Fälle untersuche nur Paare (t_i,t_j) mit $q(t_i) < q(t_j)$

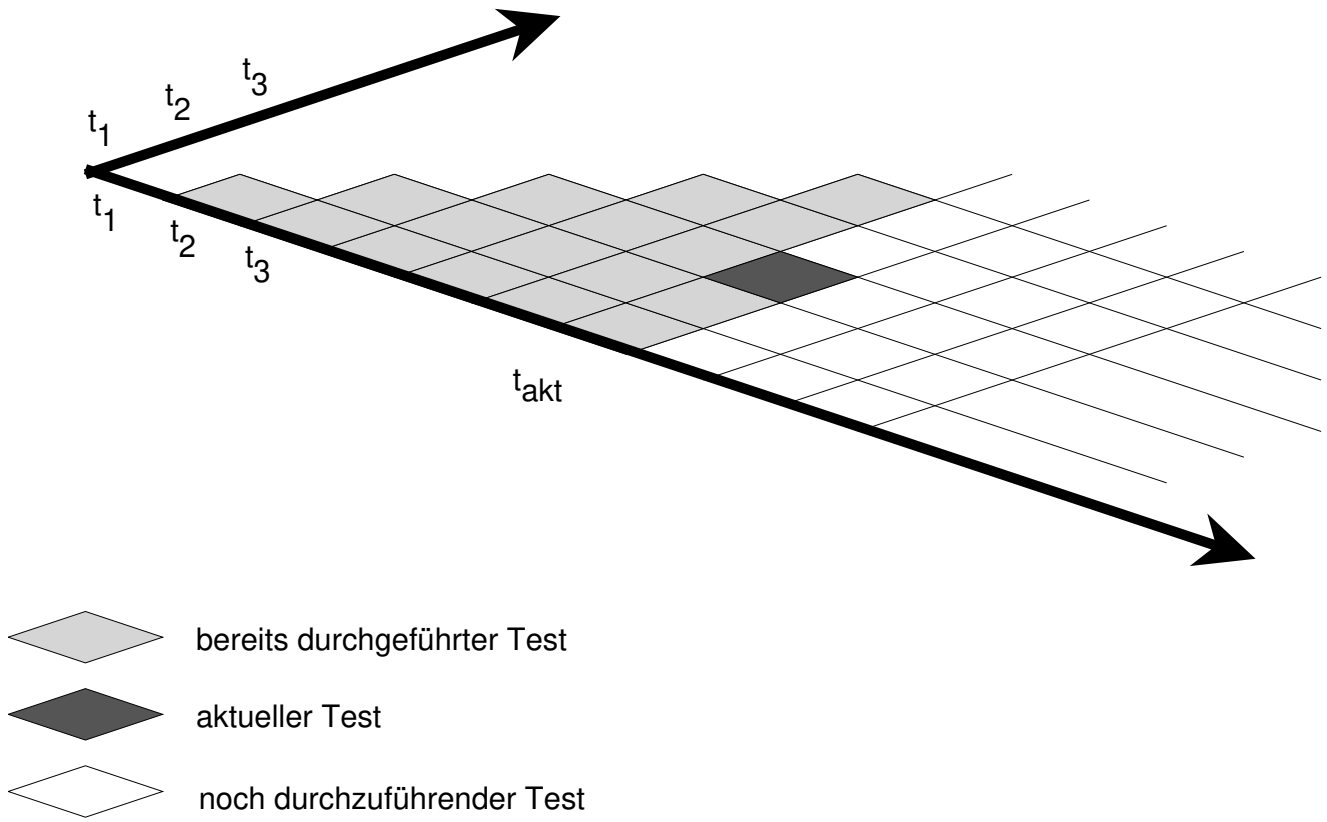
Ordnet man die zu testenden Paare demgemäß in der Ebene an, so ergibt sich folgendes Bild:



Die zuvor formulierte Suchstrategie entspricht folgender Invariante im Zeitpunkt der Betrachtung von t_{akt}:

> für alle Bäume t_i mit $q(t_i) > q(t_{akt})$ sind die Tests für alle Paare (t_i,t_j) mit $q(t_j) > q(t_i)$ bereits durchgeführt.

Im Bild veranschaulicht ergibt sich folgende Situation (getestet wird gerade das Paar (t_{akt},t_3):



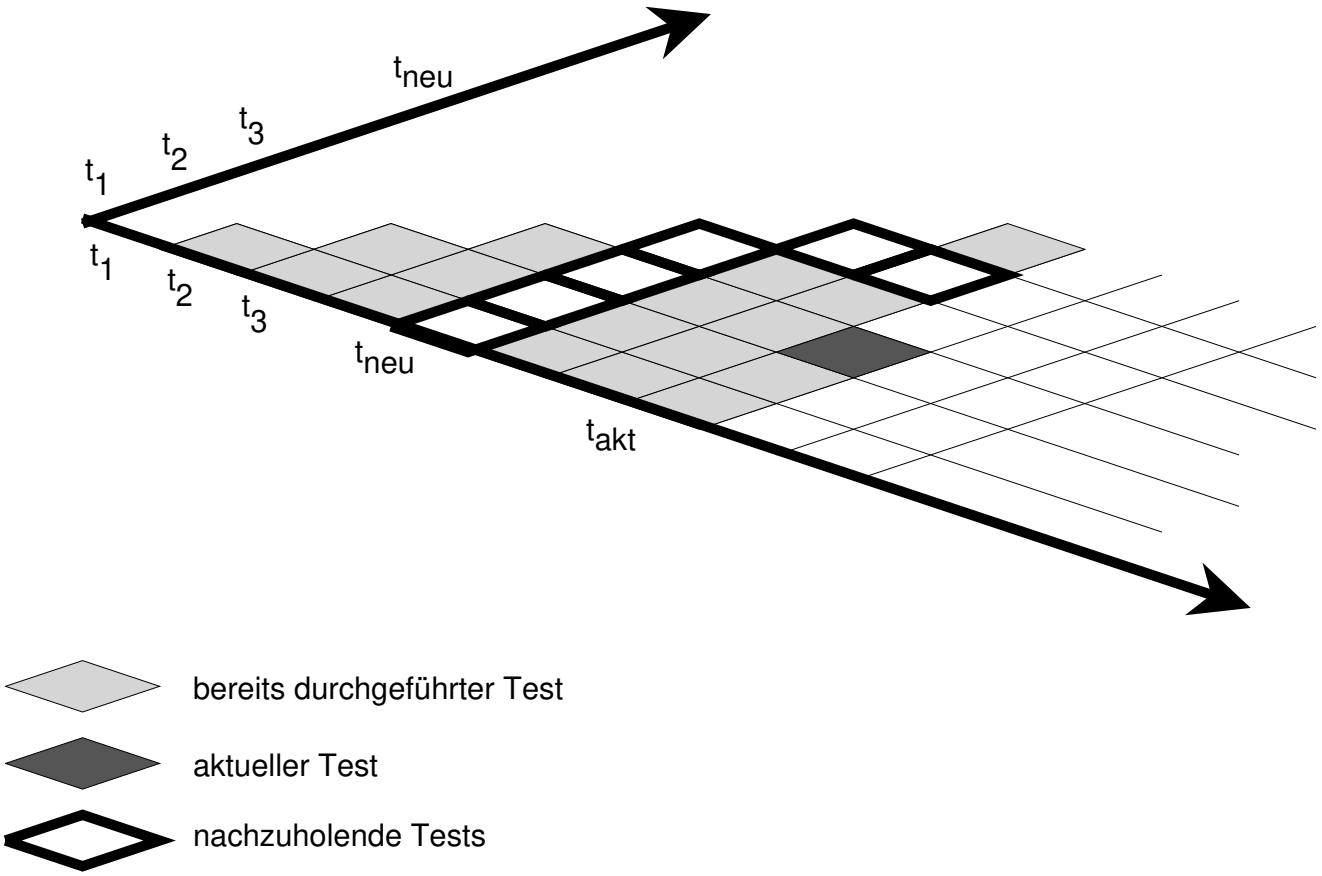
Die Schwierigkeit besteht darin, die Invariante beim Einfügen eines neuen Baumes aufrecht zu erhalten. Zwei Fälle sind dabei zu unterscheiden:

1. $q(t_{neu}) < q(t_{akt})$

   t_{neu} wird im Sinne obiger Ordnung "hinter" t_{akt} eingefügt, die Invariante wird dadurch nicht verletzt

2. $q(t_{neu}) > q(t_{akt})$

   t_{neu} wird "vor" t_{akt} eingefügt und die Invariante ist verletzt für alle Paare, die t_{neu} enthalten, wie folgendes Bild zeigt:



D.h. wird ein solcher Baum t_{neu} erzeugt, so müssen die entsprechenden Tests nachgeholt werden (ein Vorgang, der sich u.U. rekursiv wiederholt)

Weitere Suchvorgaben:

Es erscheint unnötig, bereits zu Anfang des Parsing komplexere Regeln (etwa Matrizen-Regeln) zu betrachten, die ohnehin noch nicht erfüllt werden können.
Außerdem erscheint es sinnvoll, für Ableitungsbäume hoher Güte bereits komplexere Regeln zu betrachten als für Bäume niedrigerer Güte.

Allen Regeln wird beim Entwurf der Grammatik daher ein (willkürlicher, vom Entwerfenden zu bestimmender) Wert

$$level_r \in N$$

zugewiesen, der die Komplexität der Regel angeben soll (höhere Werte entsprechen höherer Komplexität).

Desweiteren wird das Parsing zeitlich in Abschnitte $a_1 \ldots a_m$ unterteilt. Kennzeichnend für einen Abschnitt a ist eine Funktion

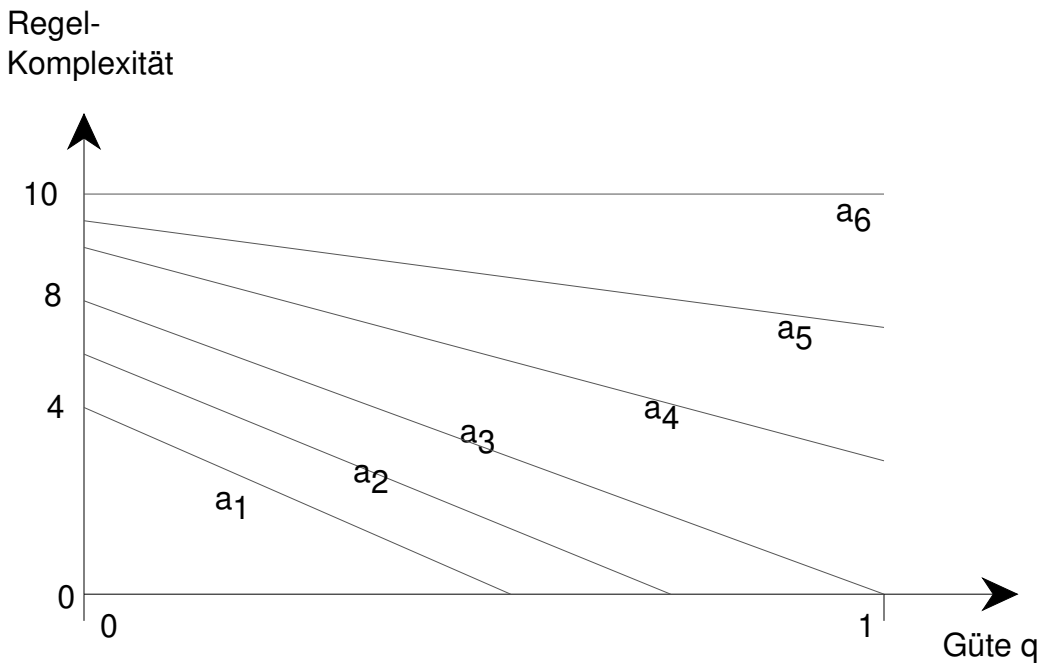
$$\text{maxlevel}_a : [0..1] \rightarrow \{ \text{level}_r, r \in R \}$$

die für ein zu testendes Baumpaar (t_i,t_j) die maximale Komplexität der zu testenden Regeln angibt, d.h. im Abschnitt a werden für ein Paar von Ableitungsbäumen (t_i,t_j) nur die Regeln

$$R_{a,t_i,t_j} = \{ r \in R / \text{level}_r \leq \text{maxlevel}_a(\min(q(t_i), q(t_j))) \}$$
$$= \{ r \in R / \text{level}_r \leq \text{maxlevel}_a(q(t_i)) \} \text{ wegen der Anordnung der Paare}$$

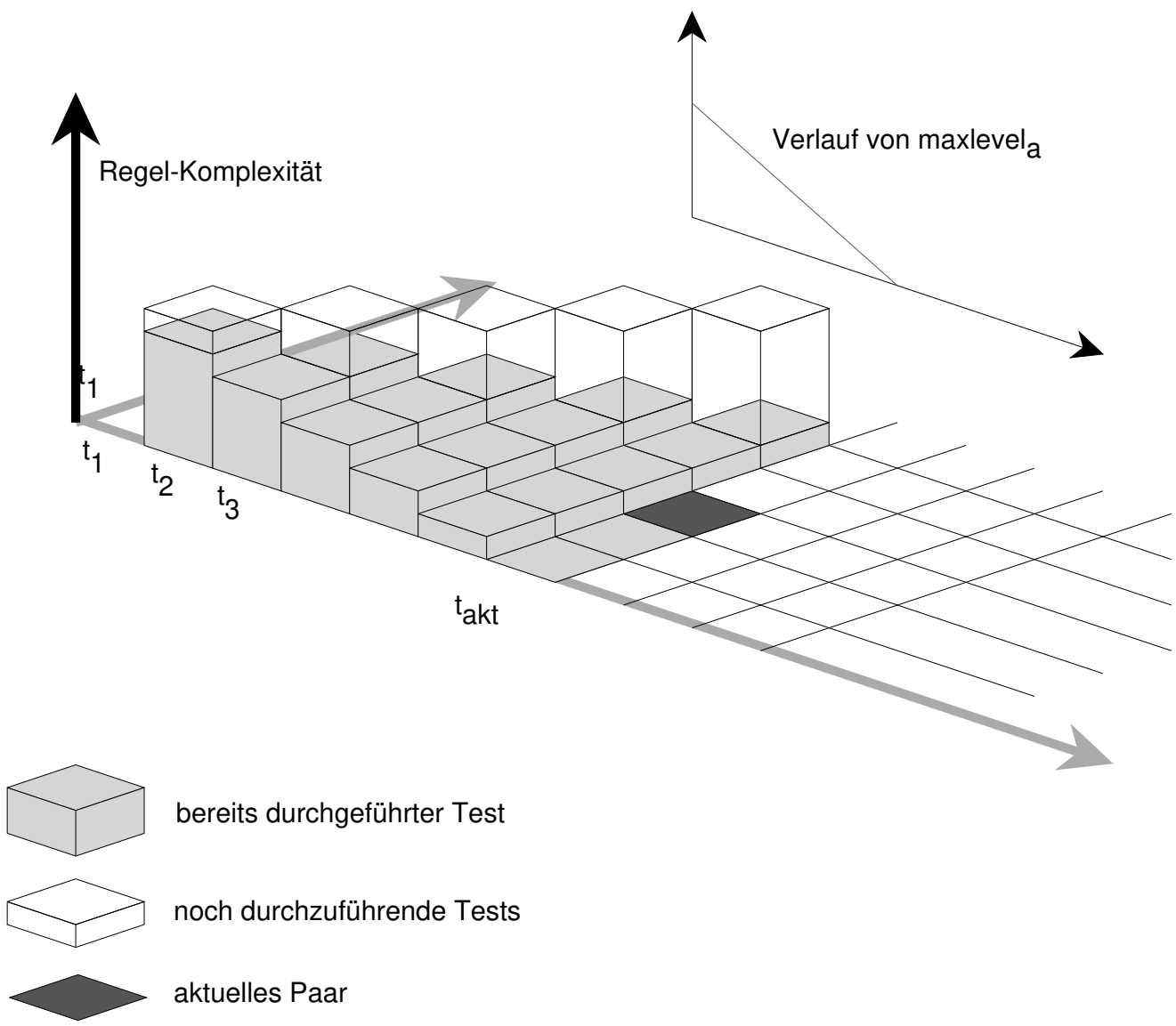
untersucht.

Angestrebter Verlauf der abschnittsbeschreibenden Funktionen maxlevel_a:



Im ersten Abschnitt a_1 werden z.B. nur für Ableitungsbäume hoher Güte Regeln mit zugeordnetem Level 4 angewandt. Für Regeln niedriger Güte (< 0.5) wird dagegen noch gar keine Regel angewandt. Erst bei Erreichen des Abschnittes a_6 werden alle Reglen für auf alle Bäume angewandt.

Betrachtet man wieder die bereits durchgeführten Tests und führt die Regelkomplexität als dritte Dimension ein, so ergibt sich folgendes Bild (es sind nur die durchgeführten Tests vollständig eingezeichnet):



Damit ergibt sich abschließend folgender Algorithmus:

(S1'') AB = ∅
berechne anhand der unären Regeln die Ableitungsbäume für die Zeichen der Eingabe

wiederhole für alle Abschnitte a

(S2.0'') Beginne suche beim Baum 2-höchster Güte

wiederhole

(S2'') finde nächstes Paar disjunkter Ableitungsbäume $(t_1,t_2) \subseteq AB$

wiederhole für alle binären Regeln r

(S3'') falls Regel r: $N_0 \leftarrow N_1\ N_2$ syntaktisch auf (t_1,t_2) paßt, d.h.
$TYP(t_i) = N_i,\ i=1,2,$
Regel r noch nicht mit (t_1,t_2) getestet wurde (*)
und $level_r \leq maxlevel_a(\ q(t_1)\ )$
dann

(S4'') berechne $\mu_r(ATTR(t_1),ATTR(t_2))$

(S5'') falls $\mu_r > \mu_{threshold}(=0)$, dann erzeuge einen neuen Ableitungsbaum t_{neu} vom Typ N_0 und füge ihn zu AB hinzu

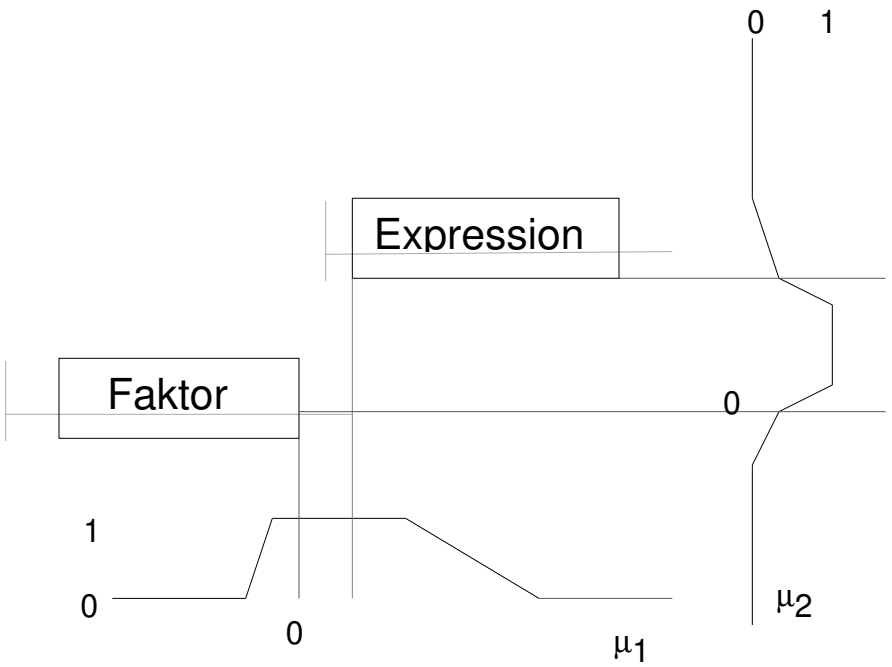
(S6'') falls $q(t_{neu}) > q(t_1)$ dann
hole Tests nach durch rekursiven Aufruf der Schleife um (S2'') .. (S6''), wobei die Suche auf die Bäume $t_{neu}..t_1$ beschränkt bleibt

(S7'') bis kein neuer Test mehr möglich ist

Integration des Parsers ins Gesamtsystem

- Realisierung des Generatorkonzepts
  - schon jetzt mehrere Grammatiken möglich
  - evtl. Grammatik dynamisch ladbar

- Editor zur graphischen Eingabe der Grammatik und der Funktionen μ_r



- Editor erzeugt automatisch Chomsky-Normalform der Regeln
- μ_r ist für jede Regel graphisch eingebbar. Jedoch ist die Fkt. dabei starken Einschränkungen unterworfen. So kann μ_r z.B. nur aus 2-stelligen, abschnittsweise linearen Teilfunktionen zusammengesetzt werden.
- Der Attributsatz der Grammatik ist momentan nicht erweiterbar...
- eine neue Version des Editors, die einige dieser Einschränkungen aufhebt, wird zur Zeit am Lehrstuhl entwickelt.