Technische Universität Dresden
Fakultät Informatik
Institut für Theoretische Informatik
Lehrstuhl Grundlagen der Programmierung

Bachelorarbeit

Eingabeprodukt eines linearen nichtlöschenden Baumübersetzers mit einer regulären gewichteten Baumsprache

Oleksiy Pitkin

eingereicht am: 30. September 2013

Verantwortlicher Hochschullehrer: Prof. Dr.-Ing. habil. Heiko Vogler

Betreuer: Dipl.-Inf. Matthias Büchse

Inhaltsverzeichnis

1 Einleitung **1**

2 Definitionen **5**
2.1 Grammatik 6
2.2 Transducer-Formalismen 8

3 Input Product **11**

4 Implementierung **15**
4.1 Datentypen 15
4.1.1 Baum-zu-Wort-Übersetzer 15
4.1.2 Grammatik 16
4.1.3 Hypergraph 17
4.2 Funktionen 18
4.2.1 Hilfsfunktionen 18
4.2.2 Wahrscheinlichkeit eines Terminalbaumes 19
4.2.3 Wahrscheinlichkeit der input product Regeln 20
4.2.4 Konstruktion der Hyperkanten für das input product Baum-zu-Wort-Übersetzer 20
4.2.5 Menge der Hyperkanten und Gewichtsvektor 22
4.2.6 Input Product 23

5 Zusammenfassung **25**

Literaturverzeichnis **27**

1 Einleitung

Das Forschungsgebiet der natürlichen Sprachverarbeitung (natural language processing, kurz NLP) beschäftigt sich mit unterschiedlichen Themen rund um die Sprache. Ein Teilgebiet der NLP ist das syntaxbasierte maschinelle Übersetzen, welches auf der Basis der baumbasierten Modelle funktioniert. Als eine mögliche Grundlage für so ein baumbasiertes Modell gelten Baumübersetzer, insbesondere lineare nichtlöschende Baum-zu-Wort-Übersetzer (tree-to-string Transducer).

Solche Baum-zu-Wort-Übersetzer betrachten die syntaktische Struktur des Satzes und übersetzen diesen Satz gleichzeitig in die andere Sprache. Um solch ein baumbasiertes Übersetzungsmodell zu beeinflussen wurde das Eingabeprodukt (input product) entwickelt. Das input product ist ein effektives Verfahren um die Wahrscheinlichkeit der Regeln eines Baum-zu-Wort-Übersetzers zu verändern. Dafür wird eine reguläre Baumgrammatik benötigt, mit der man den Eingabebaum des Baum-zu-Wort-Übersetzers ableitet und dadurch eine Wahrscheinlichkeit bekommt, mit der man die ursprüngliche Regel verändert.

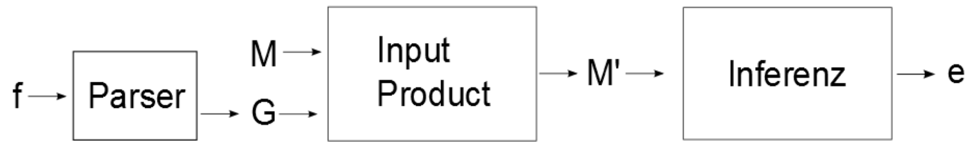
Bei einem gegebenen Baum-zu-Wort-Übersetzer M und einer regulären Bamgrammatik G, wird das input product als eine Abbildung definiert, die das Gewicht

$$[\![G]\!](\xi) \cdot [\![M]\!](\xi, w)$$

für jeden Eingabebaum ξ berechnet, wobei $[\![M]\!](\xi, w)$ die gewichtete Übersetzung des Baum-zu-Wort-Übersetzers M ist und $[\![G]\!](\xi)$ eine Abbildung ist, die dem Baum ξ eine Wahrscheinlichkeit zuordnet. Mit anderen Worten soll der konstruierte Baum-zu-Wort-Übersetzer M' das Gewicht $[\![M]\!](\xi, w)$ des Übersetzers M mit dem Gewicht $[\![G]\!](\xi)$, welches man mithilfe der Grammatik berechnet, skalieren.

Das input product hat mehrere mögliche Verwendungszwecke. Mithilfe des input products ist man in der Lage, die Regeln des Baum-zu-Wort-Übersetzers zu verändern, beschränken oder sogar auch das Gewicht ganz auf Null setzen.

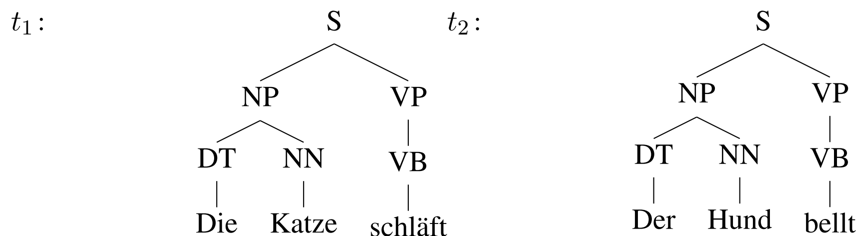
Beispiel der Anwendung des input products beim Übersetzen:



Es wird ein Satz f an den Parser übergeben. Der Parser berechnet aus dem Satz f die Grammatik G. Das input product wird aus der Grammatik und dem Baum-zu-Wort-Übersetzer konstruiert. Als Ausgabe wird ein Satz e gewählt, so dass die beste Ableitung von dem input product Baum-zu-Wort-Übersetzer eine Übersetzung nach e beschreibt.

Beispiel 1.1 Der Baum-zu-Wort-Übersetzer M enthält folgende Bäume:

t_1: t_2:



Es wird ein Satz $f =$ "Die Katze schläft" an den Parser übergeben und aus dem Satz berechnet der Parser die Grammatik G.

Zum Beispiel weist die Grammatik G dem Baum t_1 eine Wahrscheinlichkeit von 0.3 zu.

$$[\![G]\!](t_1) = 0.3.$$

Die Wahrscheinlichkeit von dem Baum t_2 wird aber 0.

$$[\![G]\!](t_2) = 0.$$

Der Baum-zu-Wort-Übersetzer M beschreibt folgende Übersetzungen:

$$[\![M]\!](t_1, \text{the cat sleeps}) = 0.2,$$
$$[\![M]\!](t_1, \text{the cat is sleeping}) = 0.1,$$
$$[\![M]\!](t_2, \text{the dog barks}) = 0.9.$$

Es wird das input product M' aus der Grammatik und dem Baum-zu-Wort-Übersetzer konstruiert:

$$[\![M']\!](t_1, \text{the cat sleeps}) = 0.3 \cdot 0.2 = 0.06,$$

$$[\![M']\!](t_1, \text{the cat is sleeping}) = 0.3 \cdot 0.1 = 0.03,$$
$$[\![M']\!](t_2, \text{the dog barks}) = 0.$$

So wird als Ausgabe der Satz “the cat sleeps” geliefert. Durch die Einwirkung des input products auf den Baum-zu-Wort-Übersetzer in diesem Beispiel wird dem Baum t_2 eine Wahrscheinlichkeit von 0 zugewiesen, obwohl der Baum mit einer sehr hohen Wahrscheinlichkeit übersetzt wird. $\square$

2 Definitionen

In diesem Abschnitt werden alle für das Thema notwendigen und grundlegenden Definitionen aufgestellt.

Ein *Alphabet* Σ ist eine endliche, nichtleere Menge von Symbolen.

Die Menge der *Wörter* über Σ wird durch Σ^* bezeichnet.

Die Menge der *Bäume* U_Σ ist die kleinste Menge von Wörtern U über $\Sigma \cup \{(,),,\}$ so dass, wenn $\sigma \in \Sigma$ und $\xi_1, \ldots, \xi_k \in U$ dann $\sigma(\xi_1, \ldots, \xi_k) \in U$.

Die Menge der *Positionen* eines Baumes d ist die Menge:

$$pos(d) = \{\epsilon\} \cup \{iv \mid 1 \leq i \leq k, v \in pos(d_i)\},$$

wenn $d = \sigma(d_1, \ldots, d_k), k \geq 0$.

Ein *Rangalphabet* ist ein Tupel (Σ, rk), wobei Σ ein Alphabet ist und $rk\colon \Sigma \to \mathbb{N}$ eine Abbildung ist, die jedem Element aus Σ einen Rang zuordnet.

Das *Label* von ξ an v wird als $\xi(v)$ bezeichnet, wobei $v \in pos(\xi)$ und $\xi \in U_\Sigma$.

Der *Rang* von ξ an v wird durch $rk(v)$ bezeichnet, wobei $v \in pos(\xi)$ und $\xi \in U_\Sigma$.

Die Menge der *Rangbäume* T_Σ ist definiert durch:

$$T_\Sigma = \{\xi \in U_\Sigma \mid \forall v \in pos(\xi)\colon rk(v) = rk(\xi(v))\}.$$

Eine *gewichtete Baumsprache* ist eine Abbildung $\varphi\colon U_\Sigma \to \mathbb{R}_{\geq 0}$.

Eine *gewichtete Baum-zu-Wort-Übersetzung* (kurz gewichtete Übersetzung) ist eine Abbildung $\tau\colon U_\Sigma \times \Delta^* \to \mathbb{R}_{\geq 0}$.

2.1 Grammatik

Definition 2.1 Eine *probabilistische reguläre Baumgrammatik (in Normalform)* ist ein Tupel $G = (P, \Gamma, p_0, R, wt)$, wobei

- P eine endliche Menge ist, deren Elemente wir *Zustände* nennen,
- Γ ein Alphabet ist, dessen Elemente wir *Terminalsymbole* nennen,
- $p_0 \in P$ ein Zustand ist, genannt *Initialzustand* oder *Startzustand*,
- R eine endliche Menge von *Regeln* der Form

  $$p \to \sigma(p_1, \dots, p_k)$$

  ist, wobei $p, p_1, \dots, p_k \in P$, $\sigma \in \Gamma$. Der Rang der Regel ist k.
- $wt\colon R \to \mathbb{R}_{\geq 0}$ eine Abbildung ist, die jeder Regel ihr *Gewicht* zuordnet.

Im Folgenden sei $G = (P, \Gamma, p_0, R, wt)$ eine probabilistische reguläre Baumgrammatik.

Jetzt definieren wir die *Semantik* von G.

Eine *Ableitung* d von G ist ein Baum in T_R, wobei

- jeder Knoten mit einer Regel $r \in R$ beschriftet ist,
- Wenn $v \in pos(d)$ ein Knoten mit der Regel $p \to \sigma(p_1, \dots, p_k)$ ist, dann hat dieser Knoten k Kindknoten, deren linken Seiten jeweils $p_1, \dots, p_k$ sind.

Die *Menge aller Ableitungen* von G bezeichnen wir mit D_G^p. Die Regel an der Wurzel hat den Zustand p auf der linken Seite.

Wir definieren $\pi\colon T_R \to U_\Sigma$ als die Abbildung, die aus einer Ableitung einen Terminalbaum wie folgt bildet. Den Terminalbaum erhält man durch das Ersetzen jeder Regel in der Ableitung durch das in ihrer rechten Seite enthaltene Terminalsymbol.

Mit $D_G^p(t)$ bezeichnen wir die *Menge der Ableitungen*, deren Bild unter π der Baum $t \in U_\Sigma$ ist und bei denen die Regel an der Wurzel den Zustand p auf der linken Seite hat.

Das *Gewicht einer Ableitung* $wt(d)$ für $d \in D_G^p$ ist das Produkt aller Gewichte der in der Ableitung vorkommenden Regeln, d.h.,

$$wt(d) = \prod_{v \in \mathrm{pos}(d)} wt(d(v)).$$

Die *Baumsprache* $L(G)$ ist die Menge aller Terminalbäume, also

$$L(G) = \{\pi(d) \mid d \in D_G^p\}.$$

Für jedes $p \in P$ definieren für die Abbildung $[\![G]\!]^p\colon U_\Sigma \to \mathbb{R}_{\geq 0}$ durch:

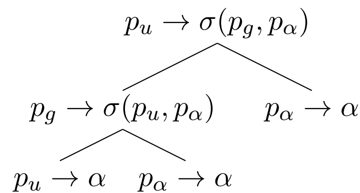
$$[\![G]\!]^p(t) = \sum_{d \in D_G^p(t)} wt(d).$$

Falls $p = p_0$, dann lassen wir das hochgestellte p weg und nennen die Abbildung die *Semantik von G*. □

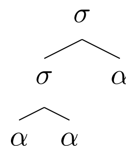
Beispiel 2.2 Sei $G = (P, \Sigma, p_u, R_G, wt_G)$ mit $P = \{p_g, p_u, p_\alpha\}$, $\Sigma = \{\sigma, \alpha\}$ und R_G, wt_G wie folgt:

$$\begin{array}{lc} p_1\colon p_g \to \sigma(p_u, p_\alpha) & 1.0 \\ p_2\colon p_u \to \alpha & 0.6 \\ p_3\colon p_u \to \sigma(p_g, p_\alpha) & 0.4 \\ p_4\colon p_\alpha \to \alpha & 1.0. \end{array}$$

Eine Ableitung von G ist:



Der korrespondierende Terminalbaum ist:



Die Wahrscheinlichkeit der Ableitung wird wie folgt berechnet:

$$\begin{aligned} wt_G(d) &= wt_G(p_3) \cdot wt_G(p_1) \cdot wt_G(p_4) \cdot wt_G(p_2) \cdot wt_G(p_4) \\ &= 0.4 \cdot 1.0 \cdot 1.0 \cdot 0.6 \cdot 1.0 \\ &= 0.24. \end{aligned}$$

Für den Terminalbaum gibt es nur eine mögliche Ableitung. Die Wahrscheinlichkeit des Terminalbaumes ist demzufolge gleich der Wahrscheinlichkeit der Ableitung. □

2.2 Transducer-Formalismen

Im Folgenden sei $X_k = \{x_1, \ldots, x_k\}$.

Definition 2.3 Ein *Baum-zu-Wort-Übersetzer* ist ein Tupel

$$M = (Q, \Sigma, \Delta, q_0, R, wt),$$

wobei

- Q eine endliche Menge ist, deren Elemente wir *Zustände* nennen,
- Σ und Δ Eingabe- und Ausgabealphabete sind,
- $q_0 \in Q$ ein Zustand ist, genannt *Initialzustand* oder *Startzustand*,
- R eine endliche Menge von *Regeln* der Form

  $$q \to \langle \xi, w \rangle (q_1, \ldots, q_k)$$

  ist, wobei $q, q_1, \ldots, q_k \in Q$, $\xi \in U_\Sigma(X_k)$ und $w \in (\Delta \cup X_k)^*$, und jede Variable aus X_k kommt genau einmal in ξ sowie in w vor,
- $wt \colon R \to \mathbb{R}_{\geq 0}$ eine Abbildung ist, die jeder Regel ihr *Gewicht* zuordnet.

Ein Baum-zu-Wort-Übersetzer ist *proper*, wenn

$$\sum\nolimits_{\rho \in R_q} wt(\rho) = 1,$$

wobei für jedes $q \in Q$,

$$R_q = \{q \to \langle \xi, w \rangle (q_1, \ldots, q_k) \mid q \to \langle \xi, w \rangle (q_1, \ldots, q_k) \in R\}$$

gilt. □

Im Folgenden sei $M = (Q, \Sigma, \Delta, q_0, R, p)$ ein Baum-zu-Wort-Übersetzer.

Jetzt definieren wir die Semantik von M. Dafür nutzen wir den Ansatz der Bimorphismen [AD76].

Definition 2.4 Die *center tree PRTG D_M zu M* ist die PRTG $(P, \Gamma, p_0, R_G, wt)$, wobei

$$\Gamma = \{\langle \xi, w\rangle^{(k)} \mid \exists q, q_1, \ldots, q_k, \xi, w \colon q \to \langle \xi, w\rangle(q_1, \ldots, q_k) \in R\} .$$

Wir definieren die Abbildungen $h_1 \colon T_\Gamma \to U_\Sigma$ und $h_2 \colon T_\Gamma \to \Delta^*$ wie folgt. Für jeden Baum $\langle \xi, w\rangle(\zeta_1, \ldots, \zeta_k) \in T_\Gamma$ gelte:

$$h_1(\langle \xi, w\rangle(\zeta_1, \ldots, \zeta_k)) = \xi[x_i/h_1(\zeta_i)],$$
$$h_2(\langle \xi, w\rangle(\zeta_1, \ldots, \zeta_k)) = w[x_i/h_2(\zeta_i)],$$

wobei $[x_i/h_1(\zeta_i)]$ die Ersetzung der Variable x_i durch die Anwendung der Abbildung h_1 auf ζ_i ist. Analog gilt das für die Abbildung h_2.

Die Semantik $[\![M]\!]$ von M ist definiert durch

$$[\![M]\!] \colon U_\Sigma \times \Delta^* \to \mathbb{R}_{\geq 0} \cup \{\infty\} ,$$
$$[\![M]\!](\xi, w) = \textstyle\sum_{\zeta \in T_\Gamma \colon h_1(\zeta) = \xi, h_2(\zeta) = w} [\![D_M]\!](\zeta) . \qquad \square$$

Wenn die Indexmenge der Summe unendlich ist, z.B., wenn eine Regel

$$q \to \langle x_1, x_1\rangle(q)$$

existiert, dann kann die Summe divergieren. Ob dies passiert, hängt von der Abbildung wt ab. Wenn diese proper ist, divergiert die Summe nicht. Im Folgenden gehen wir dann davon aus, dass die Summe nicht divergiert und notieren daher auch:

$$[\![M]\!] \colon U_\Sigma \times \Delta^* \to \mathbb{R}_{\geq 0},$$

d.h. $[\![M]\!]$ ist eine Baum-zu-Wort-Übersetzung.

Beispiel 2.5 Sei $M = (Q, \Sigma, \Delta, q, R_M, wt_M)$ mit $Q = \{q\}$, $\Sigma = \{\sigma, \alpha\}$, $\Delta = \{a\}$ und R_M, wt_M wie folgt:

$$r_1 \colon q \to (\beta_1, x_1 a)(q) \qquad 0.5,$$
$$r_2 \colon q \to (\beta_2, a)() \qquad 0.5,$$

wobei $\beta_1 = \sigma(x_1, \alpha)$ und $\beta_2 = \alpha$.

Wir betrachten folgenden center tree t:

$$\begin{array}{c}\beta_1\\ |\\ \beta_1\\ |\\ \beta_2\end{array}$$

Die Berechnung von h_1 läuft folgendermaßen ab:

$$h_1(t) = \begin{array}{c}\sigma\\ \diagup\;\diagdown\\ x_1 \quad \alpha\end{array} [x_1/h_1(\beta_1(\beta_2))]$$

$$h_1\left(\begin{array}{c}\beta_1\\ |\\ \beta_1\end{array}\right) = \begin{array}{c}\sigma\\ \diagup\;\diagdown\\ x_1 \quad \alpha\end{array} [x_1/h_1(\beta_2)]$$

$$h_1(\beta_2) = \alpha$$

Dann wird $h_1(\beta_2) = \alpha$ in die Variable x_1 der vorherigen Regel eingesetzt und man erhält den Baum $\begin{array}{c}\sigma\\ \diagup\;\diagdown\\ \alpha \quad \alpha\end{array}$

Analog geht man mit $h_1\left(\begin{array}{c}\beta_1\\ |\\ \beta_1\end{array}\right)$ vor und erhält den kompletten Terminalbaum: $\begin{array}{c}\sigma\\ \diagup\;\diagdown\\ \begin{array}{c}\sigma\\ \diagup\;\diagdown\\ \alpha \quad \alpha\end{array} \quad \alpha\end{array}$

Die Berechnung von h_2 läuft folgendermaßen ab:

$$h_2(t) = x_1 a[x_1/h_2(\beta_1(\beta_2))]$$

$$h_2\left(\begin{array}{c}\beta_1\\ |\\ \beta_1\end{array}\right) = x_1 a[x_1/h_1(\beta_2)]$$

$$h_2(\beta_2) = a$$

Genauso wie bei der Abbildung h_1 werden alle Variablen ersetzt und man erhält das Wort aaa. □

3 Input Product

Die gewichtete Übersetzung eines Baum-zu-Wort-Übersetzers ist eine Abbildung $\tau\colon U_\Sigma \times \Delta^* \to \mathbb{R}_{\geq 0}$. Das Eingabeprodukt (input product) von τ und einer gewichteten Baumsprache $\varphi\colon U_\Sigma \to \mathbb{R}_{\geq 0}$ ist eine gewichtete Übersetzung $(\varphi \triangleleft \tau)$ mit :

$$(\varphi \triangleleft \tau)(\xi, w) = \varphi(\xi) \cdot \tau(\xi, w)$$

für jedes $\xi \in U_\Sigma$ und $w \in \Delta^*$[Mal10].

Theorem 3.1 *Sei $M = (Q, \Sigma, \Delta, q_0, R_M, wt_M)$ ein Baum-zu-Wort-Übersetzer und $G = (P, \Gamma, p_0, R_G, wt_G)$ eine PRTG. Daraus lässt sich ein Baum-zu-Wort-Übersetzer konstruieren, so dass $[\![M']\!] = [\![G]\!] \triangleleft [\![M]\!]$.*

Definition 3.2 Konstruktion des input products:

- Das input product ist ein Baum-zu-Wort-Übersetzer

$$M' = (Q', \Sigma, \Delta, (q_0, p_0), R', wt)$$

- Die neue Menge der Zustände Q' ist das kartesische Produkt der Zustände des Baum-zu-Worts-Übersetzers und der Grammatik, d.h. $Q' = Q \times P$.

- Die Menge der Regeln von M' ist:

$$R' = \{(q,p) \to \langle \xi, w\rangle((q_1,p_1), \ldots, (q_k,p_k)) \mid q \to \langle \xi, w\rangle(q_1, \ldots, q_k) \in R_M, \\ p, p_1, \ldots, p_k \in P\}$$

- Sei $(r, p, p_1, \ldots, p_k) \in R'$ eine Abkürzung für:

$$(q,p) \to \langle \xi, w\rangle((q_1,p_1), \ldots, (q_k,p_k)).$$

- Für jede Regel $(r, p, p_1, \ldots, p_k) \in R'$ wird eine neue Grammatik $G_{p,p_1,\ldots,p_k}$ erzeugt, deren Regelmenge folgendermaßen definiert ist:

  $$R_{p,p_1,\ldots,p_k} = R_G \cup \{p_1 \to x_1, \ldots, p_k \to x_k\}.$$

  Jede der Regeln $\{p_1 \to x_1, \ldots, p_k \to x_k\}$ ist mit der Wahrscheinlichkeit 1 gewichtet. Startzustand der Grammatik $G_{p,p_1,\ldots,p_k}$ ist p.

- Das Gewicht für die Regel $(r, p, p_1, \ldots, p_k) \in R'$:

  $$wt(r, p, p_1, \ldots, p_k) = wt(r) \cdot [\![G_{p,p_1,\ldots,p_k}]\!]^p(\xi). \qquad \square$$

Man kann zeigen, dass $[\![M']\!] = [\![G]\!] \triangleleft [\![M]\!]$ gilt.

Beispiel 3.3 Für die Konstruktion von M' wird ein Baum-zu-Wort-Übersetzer und eine Grammatik gebraucht.

Der Baum-zu-Wort-Übersetzer aus dem Beispiel 2.5 mit der Regelmenge:

$$R_M = \{\ r_1\colon q \to (\ \sigma(x_1, \alpha)\ ,\ x_1 a\ )(q), \quad r_2\colon q \to (\alpha, a)()\ \}$$

wird mit der Grammatik G aus dem Beispiel 2.2 geschnitten.

Die Regelmenge der Grammatik ist:

$$R_G = \{p_1\colon p_g \to \sigma(p_u, p_\alpha),\ p_2\colon p_u \to \alpha,\ p_3\colon p_u \to \sigma(p_g, p_\alpha),\ p_4\colon p_\alpha \to \alpha\}$$

Zuerst werden alle mögliche Regeln aus r_1 konstruiert:

- (r_1, p_g, p_u): $(q, p_g) \to (\ \sigma(x_1, \alpha)\ ,\ x_1 a\ )((q, p_u))\}$

  Die Regel $p_u \to x_1$ mit der Wahrscheinlichkeit 1 wird in die Grammatik eingefügt.

  $wt(r_1, p_g, p_u) = wt_M(r_1) \cdot [\![G_{p_g,p_u}]\!]^{p_g}(\xi)$

  Aus dem Zustand p_g muss der Baum ξ gebildet werden. Für den Baum gibt es nur eine mögliche Ableitung:

  $$p_g \to \sigma(p_u, p_\alpha)\left(p_u \to x_1,\ p_\alpha \to \alpha\right)$$

$wt_G(d) = wt_G(p_1) \cdot wt_G(p_4) \cdot 1 = 1 \cdot 1 \cdot 1 = 1$

$wt(r_1, p_g, p_u) = wt_M(r_1) \cdot (wt_G(p_1) \cdot wt_G(p_4)) = 0.5 \cdot 1 = 0.5$

- (r_1, p_g, p_g): $(q, p_g) \to (\sigma(x_1, \alpha), x_1 a)((q, p_g))$

  Die Regel $p_g \to x_1$ mit der Wahrscheinlichkeit 1 wird in die Grammatik eingefügt.

  $wt(r_1, p_g, p_g) = wt_M(r_1) \cdot [\![G_{p_g, p_g}]\!]^{p_g}(\xi)$

  Aus dem Zustand p_g muss der Baum ξ gebildet werden. Da es keine Regel $p_u \to x_1$ gibt, kann man den Baum ξ nicht konstruieren und das Gewicht der Ableitung wird 0.

  $$p_g \to \sigma(p_u, p_\alpha)$$
  $$p_u \to \ldots \quad p_\alpha \to \alpha$$

  $wt(d) = 0$

  $wt(r_1, p_g, p_g) = wt_M(r_3) \cdot 0 = 0$

Analog konstruiert man alle weiteren Regeln. Daraus ergibt sich das input product Baum-zu-Wort-Übersetzer M'

$M' = (Q', \Sigma, \Delta, (q, p_u), R', wt)$

$Q' = \{(q, p_u), (q, p_g), (q, p_\alpha))\}$

$R' = \{r'_1\colon (q, p_u) \to (\sigma(\mathrm{x}_1, \alpha), x_1 a)((q, p_g)),$

$r'_2\colon (q, p_u) \to (\alpha, a)(),$

$r'_3\colon (q, p_g) \to (\sigma(\mathrm{x}_1, \alpha), x_1 a)((q, p_u))$

$r'_4\colon (q, p_\alpha) \to (\alpha, a)()\}.$

Gewichte der Regeln sind:

$wt(r'_1) = wt_M(r_1) \cdot (wt_G(p_3) \cdot wt_G(p_4)) = 0.5 \cdot 0.4 \cdot 1 = 0.2$

$wt(r'_2) = wt_M(r_2) \cdot wt_G(p_2) = 0.5 \cdot 0.6 = 0.3$

$$wt(r'_3) = wt_M(r_1) \cdot (wt_G(p_1) \cdot wt_G(p_4)) = 0.5 \cdot 1 \cdot 1 = 0.5$$
$$wt(r'_4) = wt_M(r_2) \cdot wt_G(p_4) = 0.6 \cdot 1 = 0.6$$

□

4 Implementierung

Die Konstruktion des input products ist in der funktionalen Sprache Haskell implementiert. Zunächst werden alle Datentypen, die bei der Konstruktion eine Rolle spielen, vorgestellt. Danach wird die Konstruktion und alle Funktionen erklärt.

4.1 Datentypen

`StrictIntPair` ist ein Integer-Paar. Die beiden Komponenten werden jeweils mit `fst` und `snd` beschriftet.

```
data StrictIntPair
  = SIP
    { _fst :: !Int
    , _snd :: !Int
    } deriving (Eq, Ord, Show)
```

4.1.1 Baum-zu-Wort-Übersetzer

Der Datentyp für den Baum-zu-Wort-Übersetzer heißt `XRS`.

```
data XRS
  = XRS
    { irtg :: IRTG Int
    , weights :: VU.Vector Double
    }
```

Er besteht aus einem anderen Datentyp `IRTG`, welcher die Regelmenge des Baum-zu-Wort-Übersetzers repräsentiert und einem Double-Vektor `weights`, in dem die Gewichte der Regeln gespeichert sind.

Der Datentyp `IRTG` enthält vier Komponenten, die den Baum-zu-Wort-Übersetzer beschreiben.

```
data IRTG i
  = IRTG
    { rtg :: Hypergraph StrictIntPair i
    , initial :: Int
    , h1 :: V.Vector (T.Tree NTT)
    , h2 :: V.Vector (V.Vector NTT)
    }
```

Die Regelmenge wird mithilfe des Hypergraphen Funktion `rtg` beschrieben. Jede Hyperkante dieses Graphs hat ein `StrictIntPair` als Label und eine ganze Zahl (`i` = `Int`) als Identifikationsnummer.

Der Startzustand des Baum-zu-Wort-Übersetzers `initial` ist eine ganze Zahl.

Außerdem werden zwei Vektoren `h1` und `h2` gebraucht, um die Mengen der Bäume und der Wörter zu speichern. In den Labels der Hyperkanten wird ein Idex auf `h1` und `h2` gespeichert.

4.1.2 Grammatik

Der Datentyp der Baumgrammatik ist ähnlich dem Datentyp `XRS` aufgebaut. Er besteht aus einem Datentyp `Grammar` und einem Gewichtsvektor `gweight`, in dem die Gewichte der Regeln gespeichert sind.

```
data WGrammar
  = WGrammar
    { grammar :: Grammar Int
    , gweight :: VU.Vector Double
    }

data Grammar i
  = Grammar
    { rules :: Hypergraph Int Int
    , ginit :: Int
    }
```

Die Grammatik enthält wiederum einen Hypergraphen `rules`, der die Regelmenge der Grammatik beschreibt.

Startzustand der Grammatik ist eine ganze Zahl `ginit`.

4.1.3 Hypergraph

Für die Repräsentation der Regeln der Grammatik und des Baum-zu-Wort-Übersetzers und der Grammatik ist der Datentyp `Hypergraph` zuständig.

```
data Hypergraph l i
  = Hypergraph
    { nodes :: Int
    , edges :: [Hyperedge l i]
    }
    deriving Show
```

Alle Knoten des Hypergraphen sind ganze Zahlen.

Die Beziehungen zwischen den Knoten werden mit den Hyperkanten hergestellt.

`nodes` ist die Anzahl der Zustände im Intervall von $0 \ldots nodes - 1$.

`edges` ist die Menge der Hyperkanten.

Der Datentyp einer Hyperkante `Hyperedge` ist folgendermaßen definiert:

```
data Hyperedge l i
    = Hyperedge
    { to :: !Int
    , from :: !(VU.Vector Int)
    , label :: !l
    , ident :: !i
    }
```

Hyperkanten des Baum-zu-Wort-Übersetzers

Alle Regeln sind der Form: $q \to \langle \xi, w \rangle (q_1, \ldots, q_k)$

Der Zustand auf der linken Seite der Regel q ist eine ganze Zahl `to`.

Die Zustände $q_1, \ldots, q_k$ werden in dem Vektor `from` gespeichert.

Label der Hyperkante bei dem Baum-zu-Wort-Übersetzer ist ein `StrictIntPair`. In der ersten Komponente von dem `StrictIntPair` ist der Index auf den Baum ξ und in der zweiten Komponente ist der Index auf das Wort w gespeichert.

Hyperkanten der Grammatik

Alle Regeln sind der Form $p \rightarrow \sigma(p_1, \ldots, p_k)$

Der Zustand auf der linken Seite der Regel p ist eine ganze Zahl `to`.

Die Zustände $(p_1, \ldots, p_k)$ werden in dem Vektor `from` gespeichert.

Label der Hyperkante bei der Grammatik ist eine ganze Zahl.

4.2 Funktionen

4.2.1 Hilfsfunktionen

Die Zustände des input product Baum-zu-Wort-Übersetzers sind Paare (q, p), wobei q Zustand des Baum-zu-Wort-Übersetzers und p Zustand der Grammatik ist. Die Knoten des Hypergraphs sind aber ganze Zahlen und deswegen muss für jedes Zustandspaar eine Zahl zugewiesen werden. Da die Anzahl der Elemente beiden Zustandsmengen immer ≥ 0 ist, kann man den Zustand des input product Baum-zu-Wort-Übersetzers als eine ganze Zahl darstellen: $q \cdot |P| + p$, wobei $|P|$ die Anzahl der Elemente der Zustandsmenge von der Grammatik ist.

`stateid` liefert für zwei bestimme Zustände `pstate`, `qstate` und die Anzahl der Elemente einer Zustandsmenge `allpstates` den Zustand des input product Baum-zu-Wort-Übersetzers.

```
stateid :: Int → Int → Int → Int
stateid pstate qstate allpstates = pstate ∗ allpstates + qstate
```

`vectorid` überführt einen Vektor der Zustände der Grammatik in eine Liste von Zuständen des input product Baum-zu-Wort-Übersetzers.

```
vectorid :: V.Vector Int → Int → Int → [Int]
vectorid vec qstate allpstates = V.toList $ V.map (+qstate∗allpstates) vec
```

`alllists` erzeugt eine Liste von Listen von möglichen Kombinationen. Die Eingabe ist die ganze Zahl, die bestimmt wie viele Elemente in einer Kombination vorkommen und eine Liste der Elemente.

Beispiel: alllists 2 [’a’, ’b’] = [’aa’ , ’ab’, ’ba’, ’bb’]

```
alllists :: Int → [t] → [[t]]
alllists 0 _ = [[]]
alllists n xs = [ x : xs' | x ← xs, xs' ← alllists (n - 1) xs ]
```

`allvectors` erzeugt eine Liste von allen möglichen Vektoren.

```
allvectors :: Int → [Int] → [V.Vector Int]
allvectors n xs = map V.fromList $ alllists n xs
```

`bstar` liefert die Liste der Hyperkanten eines Hypergraphen.

```
bstar :: Hypergraph l i → Int → [Hyperedge l i]
bstar h v = [ e | e ← edges h, to e == v ]
```

4.2.2 Wahrscheinlichkeit eines Terminalbaumes

Da es nicht besonders sinnvoll ist, zuerst alle Ableitungen zu suchen und dann sie aufzusummieren, um die Wahrscheinlichkeit eines Baumes zu berechnen, kann man die Wahrscheinlichkeit auf eine andere Weise ausrechnen.

Das folgende Lemma liefert eine effiziente rekursive Berechnungsvorschrift für die Berechnung der Wahrscheinlichkeit.

Lemma 4.1 *Es gilt:*

$$[\![G]\!]^p(t) = \sum\nolimits_{p_1 \ldots p_k,\, r \in R,\, r = p \to \sigma(p_1 \ldots p_k)} (wt(r) \cdot [\![G]\!]^{p_1}(t_1) \cdot \ldots [\![G]\!]^{p_k}(t_k)),$$

wobei $t = \sigma(t_1, \ldots, t_k)$

Beweis. $[\![G]\!]^p(t) = \sum_{d \in D_G^p(t)} wt(d) = \sum_{d \in D_G^p(t)} wt(d(\epsilon)) \cdot wt(d|_1) \cdot \ldots \cdot wt(d|_k)$

$= \sum_{r \in R;\, d_1 \in D_G^{p_1}(t_1), \ldots, d_k \in D_G^{p_k}(t_k);\, r = q_0 \to \sigma(p_1, \ldots, p_k)} wt(r) \cdot wt(d_1) \cdot \ldots \cdot wt(d_k)$

$= \sum_{r \in R;\, r = q_0 \to \sigma(p_1, \ldots, p_k)} \sum_{d_1 \in D_G^{p_1}(t_1)} \cdots \sum_{d_k \in D_G^{p_k}(t_k)} wt(r) \cdot wt(d_1) \cdot \ldots \cdot wt(d_k)$

$= \sum_{r \in R;\, r = q_0 \to \sigma(p_1, \ldots, p_k)} wt(r) \cdot (\sum_{d_1 \in D_G^{p_1}(t_1)} \cdots \sum_{d_k \in D_G^{p_k}(t_k)} wt(d_1) \cdot \ldots \cdot wt(d_k))$

$= \sum_{r \in R;\, r = q_0 \to \sigma(p_1, \ldots, p_k)} wt(r) \cdot (\sum_{d_1 \in D_G^{p_1}(t_1)} wt(d_1) \cdot (\ldots \cdot (\sum_{d_k \in D_G^{p_k}(t_k))} wt(d_k)))$

$= \sum_{r \in R;\, r = q_0 \to \sigma(p_1, \ldots, p_k)} wt(r) \cdot (\sum_{d_1 \in D_G^{p_1}(t_1)} wt(d_1)) \cdot \ldots \cdot (\sum_{d_k \in D_G^{p_k}(t_k)} wt(d_k))$

$= \sum_{p_1 \ldots p_k,\, r \in R,\, r = p \to \sigma(p_1 \ldots p_k)} (wt(r) \cdot [\![G]\!]^{p_1}(t_1) \cdot \ldots [\![G]\!]^{p_k}(t_k)).$ ■

Haskell-Funktion

`treeprob` berechnet die Wahrscheinlichkeit eines Terminalbaums. Für die Eingabe der Funktion wird ein Hypergraph `h`, ein Gewichtsvektor `w`, ein Zustand aus dem der Baum konstruiert werden soll `p`, ein Baum `t` und ein Zustandsvektor `pv` benötigt.

```
treeprob
  :: Hypergraph Int Int
  → VU.Vector Double
  → Int
  → T.Tree NTT
  → V.Vector Int
  → Double
treeprob h w p t pv
  = case T.rootLabel t of
      NT i → if p == pv V.! i then 1.0 else 0.0
      T i → sum [ (w VU.! ident e) ∗ product [ treeprob h w v' t' pv
                         | (v', t') ← zip (from e) (T.subForest t) ]
                         | e ← bstar h p, label e == i ]
```

4.2.3 Wahrscheinlichkeit der input product Regeln

`newruleprob` berechnet die Wahrscheinlichkeit einer Regel des input product Baum-zu-Wort-Übersetzers. Es wird ein Double-Wert zurückgegeben. Die Wahrscheinlichkeit ist das Produkt des Resultats der Funktion `treeprob` und dem Wert aus dem Gewichtsvektor.

```
newruleprob
  :: Hyperedge l Int
  → VU.Vector Double
  → Double
  → Double
newruleprob he whe tprob
  = whe VU.! ident he ∗ tprob
```

4.2.4 Konstruktion der Hyperkanten für das input product Baum-zu-Wort-Übersetzer

Die Menge der Hyperkanten wird duch mehrere Funktionen gebildet.

`allhyperedgesrules` iteriert über die Liste der Hyperkanten und liefert die Liste aus den Paaren von Hyperkanten und ihren Wahrscheinlichkeiten. Für jede Hyperkante wird die Funktion `allstatesrules` angewendet.

```
allhyperedgesrules
  :: [Hyperedge StrictIntPair Int]
  → VU.Vector Double
  → [Int]
  → Hypergraph Int Int
  → VU.Vector Double
  → V.Vector (T.Tree NTT)
  →[(Hyperedge StrictIntPair Int, Double)]
allhyperedgesrules hedges whe allgrstateslist hgr wgr trees
  = concatMap (allstatesrules whe hgr wgr trees allgrstateslist) hedges
```

`allstatesrules` iteriert über die Liste von Zuständen der Baumgrammatik und liefert die Liste aus den Paaren von Hyperkanten und ihren Wahrscheinlichkeiten. Für jeden Zustand wird die Funktion `createvector` angewendet.

```
allstatesrules
  :: VU.Vector Double
  → Hypergraph Int Int
  → VU.Vector Double
  → V.Vector (T.Tree NTT)
  → [Int]
  → Hyperedge StrictIntPair Int
  →[(Hyperedge StrictIntPair Int, Double)]
allstatesrules whe hgr wgr trees allgrstateslist he
    = concatMap (createvector he whe  allgrstateslist hgr wgr trees)
                allgrstateslist
```

`createvector` erzeugt für eine bestimmte Hyperkante die Liste von möglichen Vektoren, die eine Regel für das input product bilden können.

```
createvector
  :: Hyperedge StrictIntPair Int
  → VU.Vector Double
  → [Int]
  → Hypergraph Int Int
  → VU.Vector Double
  → V.Vector (T.Tree NTT)
  → Int
  →[(Hyperedge StrictIntPair Int, Double)]
createvector he whe allgrstateslist hgr wgr trees stategr
```

```
  = let veclist = allvectors (length $ from $ he) allgrstateslist
    in concatMap (createrule he whe (length  allgrstateslist)
                   stategr hgr wgr trees) veclist
```

`createrule` ist die Hauptfunktion. Die Funktionen iteriert über die Liste der möglichen Vektoren, berechnet die Wahrscheinlichkeit des Terminalbaums und bildet Hyperkanten mit ihren Wahrscheinlichkeiten, wenn die Wahrscheinlichkeit des Terminalbaums größer als null ist.

```
createrule
  :: Hyperedge StrictIntPair Int
  → VU.Vector Double
  → Int
  → Int
  → Hypergraph Int Int
  → VU.Vector Double
  → V.Vector (T.Tree NTT)
  → V.Vector Int
  → [(Hyperedge StrictIntPair Int, Double)]
createrule he whe allgrstates stategr hgr wgr trees x
  = case tprob of
      0 → []
      _ → [((mkHyperedge (stateid  (to he)  (stategr) allgrstates)
                         (vectorid x  (to he)  allgrstates)
                         (label he)
                         (ident he)),
                         (newruleprob he whe tprob))]
     where tprob = treeprob hgr wgr stategr
                           (trees V.!(_fst $ label $ he)) x
```

4.2.5 Menge der Hyperkanten und Gewichtsvektor

Nachdem die Funktion `allhyperedgesrules` eine Liste der Hyperkanten mit ihren Wahrscheinlichkeiten liefert, muss jeder Hyperkante noch eine ID zugewiesen werden. Die Funktion `changeidedges` nimmt das Ergebnis und erstellt eine Liste von Hyperkanten für das input product.

```
changeidedges
  :: [(Hyperedge StrictIntPair Int, Double)]
  → [Hyperedge StrictIntPair Int]
changeidedges xs =
   let m = (M.fromList (zip (L.nub  (snd (unzip xs))) [0..]))
```

```
  in [ e {ident = m M.! d}  | (e, d) ← xs ]
```

Genauso nimmt die Funktion `hevector` das Ergebnis von der Funktion `allhyperedgesrules` und überführt es in einen Gewichtsvektor für das input product.

```
hevector
  :: [(Hyperedge StrictIntPair Int, Double)]
  → VU.Vector Double
hevector xs = let m = (L.nub  (snd (unzip xs))) in VU.fromList m
```

4.2.6 Input Product

Die Funktion nimmt einen Baum-zu-Wort-Übersetzer, eine Grammatik als Eingabe und berechnet das input product. Als Ausgabe kommt ein neuer Baum-zu-Wort-Übersetzer.

```
inputprod :: XRS → WGrammar → XRS
inputprod xrs xgrammar
  = let hyperedgespairs
          = (allhyperedgesrules (edges $ rtg $ irtg $ xrs)
             (weights xrs)
             (flip take [0..] $ nodes $ rules $ grammar $ xgrammar)
             (rules $ grammar $ xgrammar)
             (gweight xgrammar)
             (h1 $ irtg $ xrs))
        unzipedpairs = (L.nub(snd (unzip hyperedgespairs)))


    in XRS { irtg =
              IRTG {
                     rtg = mkHypergraph (changeidedges
                                         hyperedgespairs
                                         unzipedpairs),
                     initial = (stateid (ginit $ grammar $ xgrammar)
                                (initial $ irtg $xrs)
                                (nodes $ rules $ grammar $ xgrammar)),
                     h1 = (h1 $ irtg $ xrs) ,
                     h2 = (h2 $ irtg $ xrs)
                   } ,
              weights = (hevector unzipedpairs)
           }
```

5 Zusammenfassung

Hauptsächlich beschäftigt sich diese Bachelorarbeit mit der Anwendung, Definition und Implementierung von dem Eingabeprodukt eines Baum-zu-Wort-Übersetzers und einer regulären Baumgrammatik.

In der Einleitung wurde die Anwendung des input products angesprochen und an einem Beispiel erklärt. In dem nächsten Kapitel wurden alle für das Thema wichtige Begriffe, der Baum-zu-Wort-Übersetzer und die Grammatik formal definiert. Danach wurde die Konstruktion des input products definiert und an einem Beispiel vorgeführt. Schließlich wurde das input product und alle dazu notwendigen Funktionen in der funktionalen Sprache Haskell implementiert. In der Arbeit wurde die Funktionsweise des Programms dokumentiert und erklärt. Außerdem wurde das input product in das Programm Vanda-Studio integriert.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit selbstständig und nur unter Zuhilfenahme der angegebenen Literatur verfasst habe.

Dresden, den 30.09.2013 Unterschrift

Literaturverzeichnis

[AD76] André Arnold and Max Dauchet. Bi-transduction de forêts. In *Proc. 3rd Int. Coll. Automata, Languages and Programming*, pages 74–86. Edinburgh University Press, 1976.

[Mal10] Andreas Maletti. Input and output products for weighted extended top-down tree transducers. In Yuan Gao, Hanlin Lu, Shinnosuke Seki, and Sheng Yu, editors, *Proc. 14th Int. Conf. Developments in Language Theory*, volume 6224 of LNCS, pages 316–327. Springer, 2010.