Die Verarbeitung von Parallelismus-Constraints

Katrin Erk*

Programming Systems Lab, Universität des Saarlandes, Saarbrücken.
www.ps.uni-sb.de/~erk

Zusammenfassung Parallelismus-Constraints sind partielle Beschreibungen von Bäumen. Wir verwenden sie als Repräsentationsformalismus in der unterspezifizierten natürlichsprachlichen Semantik. Parallelismus-Constraints sind gleichmächtig wie Kontext-Unifikation, deren Entscheidbarkeit ein bekanntes offenes Problem ist.
Dieser Text beschreibt ein Semi-Entscheidungs-Verfahren für Parallelismus-Constraints und eine erste Implementierung. Anders als alle bekannten Verfahren für Kontext-Unifikation terminiert diese Prozedur für Dominanz-Constraints, eine für die linguistische Anwendung wichtige Teilklasse.

1 Einleitung

Parallelismus-Constraints sind partielle Beschreibungen von Bäumen. Wir verwenden sie als Beschreibungsformalismus in der natürlichsprachlichen Semantik [8, 14, 7]. Ein bekanntes Problem in der Semantik ist das gehäufte Auftreten von Mehrdeutigkeiten (von denen die meisten dem menschlichen Leser gar nicht auffallen). Will man alle Lesarten eines Satzes aufzählen, so hat man mit der kombinatorischen Explosion zu kämpfen. Eine bekannte Lösung für dies Problem ist Unterspezifikation: Alle Lesarten werden in einer einzigen kompakten Repräsentation dargestellt. Parallelismus-Constraints als partielle Baumbeschreibungen sind hierzu hervorragend geeignet.

In diesem Text konzentrieren wir uns auf eine Art von Mehrdeutigkeit, nämlich Skopusambiguität, sowie ein weiteres Phänomen, die Ellipse, die auf interessante Weise mit Skopusambiguitäten interagiert. Vielleicht *der* prototypische Satz, der Skopusambiguität illustriert, ist

(1) Every man loves a woman.

Dieser Satz hat zwei Lesarten, die sich im *Skopus* der beiden Nominalphrasen „every man“ und „a woman“ unterscheiden. In Logik erster Stufe ausgedrückt, lassen sich diese zwei Lesarten so beschreiben:

(2) $\forall x.(man(x) \rightarrow \exists y.(woman(y) \wedge loves(x, y)))$ und
(3) $\exists y.(woman(y) \wedge \forall x.(man(x) \rightarrow loves(x, y)))$.

* Katrin Erk wird durch das DFG-Graduiertenkolleg „Kognitionswissenschaft – Empirie, Modellbildung, Implementation“ der Universität des Saarlandes unterstützt.

(Der entsprechende deutsche Satz „Jeder Mann liebt eine Frau“ ist nicht mehrdeutig. Im Englischen ist das Phänomen der Skopusambiguität zwar verbreiteter als im Deutschen, aber es kommt auch hier vor, z.B. in „Eine Fuge hat jeder Pianist in seinem Repertoire“ [24].) Man kann beide Lesarten unterspezifiziert beschreiben durch einen einzigen Parallelismus-Constraint, der beide Anordnungen der Teilformeln zulässt. Wir geben diesen Constraint in Abschnitt 3 an, nachdem wir Parallelismus-Constraints formal eingeführt haben.

Ein elliptischer Satz ist zum Beispiel

(4) John sleeps, and so does Bill.

Dieser Satz bedeutet soviel wie „John sleeps, and Bill sleeps“. In einer Ellipse wird etwas ausgelassen, Material, das sonst doppelt vorkäme. Wenn nun in einem elliptischen Satz zusätzlich noch eine Skopusambiguität vorliegt, kommt es zu interessanten Effekten [11]:

(5) Every linguist attends a workshop, and every computer scientist does, too.

Dieser Satz hat drei Lesarten. Zum einen könnten alle Linguisten und Informatiker gemeinsam einen Workshop besuchen (Lesart a). Zum anderen könnte es sein, dass alle Linguisten zu einem gemeinsamen Workshop fahren, während die Informatiker alle gemeinsam an einem anderen Workshop teilnehmen (Lesart b). Drittens könnte jeder einzelne, ob Linguist oder Informatiker, zu seinem individuellen Workshop fahren (Lesart c). Man kann (5) aber nicht so verstehen, dass alle Linguisten denselben Workshop besuchen, während die Informatiker auf verschiedene Workshops fahren: Entscheidet man sich im ersten Teilsatz, „a workshop“ weiten Skopus zu gewähren, dann muss auch im zweiten Teilsatz „a workshop“ weiten Skopus haben.

Ein Parallelismus-Constraint beschreibt einen Baum aus einer internen Perspektive, in Form von Beziehungen zwischen Knoten. Die interessantesten dieser Beziehungen sind Dominanz und Parallelismus. Ein *Dominanz-Literal* $X \triangleleft^* Y$ zwischen Variablen X und Y besagt, dass der Baumknoten, den X bezeichnet (kurz: X-Knoten) ein Vorfahr des Y-Knotens ist (dabei dürfen die beiden Knoten auch gleich sein). In der Computerlinguistik sind Dominanzen seit langem bekannt und werden vielseitig genutzt [18, 32, 1, 21, 10]. Wir verwenden sie für Skopusambiguitäten, ähnlich wie [27, 21, 4, 13]. Ein *Parallelismus-Literal* $A/B \sim C/D$, wie in Abb. 1 skizziert, sagt aus, dass die Baumstruktur zwischen dem A- und dem B-Knoten isomorph ist zu der zwischen dem C- und dem D-Knoten. Mit diesem Konstrukt kann man die Semantik von Ellipsen beschreiben. Desweiteren gibt es noch Constraints, um auszudrücken, daß ein Baumknoten gelabelt ist, Vater oder Bruder eines anderen Knotens ist, sowie dass zwei Knoten ungleich sind oder in disjunkten Positionen liegen.

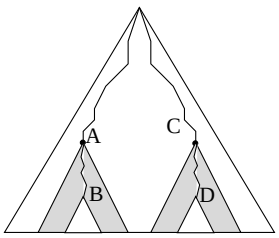


Abbildung 1. Parallelismus $A/B \sim C/D$

Wir stellen in diesem Text ein Semi-Entscheidungs-Verfahren für Parallelismus-Constraints vor. Für die linguistische Anwendung bedeutet das Verfahren, dass bei Skopus-Ambiguitäten alle Lesarten aufgezählt werden können und dass bei elliptischen Sätzen die Semantik „komplettiert“ wird, d.h. das ausgelassene Material wird eingefügt. Parallelismus-Constraints sind gleichmächtig wie Kontext-Unifikation [5, 29], eine Variante der linearen Unifikation 2. Ordnung [16, 25]. (Für einen Beweis der Gleichmächtigkeit siehe [22].) Die Entscheidbarkeit der Kontext-Unifikation ist ein bekanntes offenes Problem [28, 30]. Insofern war es nicht unser Ziel, ein terminierendes Verfahren für Parallelismus-Constraints zu entwerfen, zumal in der linguistischen Anwendung nur relativ einfache Fälle von Parallelismus aufzutreten scheinen.

Gliederung des Textes. Im folgenden Abschnitt beschreiben wir Syntax und Semantik von Parallelismus-Constraints. In Abschnitt 3 kehren wir zu den Beispielsätzen (1), (4) und (5) zurück; wir geben Constraints an, die unterspezifiziert ihre Semantik repräsentieren. Die Abschnitte 4 und 5 stellen zunächst einen Algorithmus vor, der die Erfüllbarkeit von Dominanz-Constraints testet, dann darauf aufbauend eine erweiterte Prozedur für Parallelismus-Constraints. Abschnitt 6 schließlich beschreibt eine Implementierung im Rahmen des *CHORUS Demo Systems*.

2 Semantik und Syntax

Sei eine Signatur Σ von Funktionssymbolen $f, g, a, b \ldots$ gegeben. Jedes Funktionssymbol f habe eine Stelligkeit $\mathsf{ar}(f) \geq 0$. Wir nehmen an, dass Σ mindestens 2 Funktionssymbole enthält, darunter eine Konstante und ein mindestens zweistelliges Symbol.

Ein (endlicher) *Baum* τ ist ein Grundterm über Σ, zum Beispiel $f(g(a, a))$. Einen *Knoten* eines Baumes kann man mit seinem *Pfad* von der Wurzel aus identifizieren. Dieser Pfad ist ein Wort über $\mathbb{N}$ (der Menge der natürlichen Zahlen ausschließlich 0). Wir schreiben ε für den leeren Pfad (die Wurzel) und $\pi_1\pi_2$ für die Konkatenation von π_1 und π_2. Ein Pfad π ist ein Präfix eines Pfades π', wenn es einen (ggf. leeren) Pfad π'' gibt mit $\pi\pi'' = \pi'$. Der Knoten πi ist das i-te Kind des Knotens bzw. Pfades π.

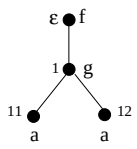


Abbildung 2. $f(g(a, a))$

Einen Baum kann man eindeutig beschreiben durch eine *Baum-Domäne* (die Menge seiner Pfade) und eine *Labeling-Funktion*. Eine Baum-Domäne ist eine nichtleere, unter Präfix-Bildung abgeschlossene Menge von Pfaden, für die gelten muss, dass $\pi i \in D \Longrightarrow \forall j < i.\pi j \in D$. Eine Labeling-Funktion ist eine Funktion $L : D \to \Sigma$, für die gilt: $\forall \pi \in D, k \in \mathbb{N}.\big(\pi k \in D \iff k \leq \mathsf{ar}(L(\pi))\big)$. Wir schreiben D_τ für die Domäne eines Baumes τ und L_τ für seine Labeling-Funktion. Für den Baum $\tau = f(g(a, a))$ in Abb. 2 ist zum Beispiel $D_\tau = \{\epsilon, 1, 11, 12\}$, $L_\tau(\epsilon) = f$, $L_\tau(1) = g$ und $L_\tau(11) = a = L_\tau(12)$.

Definition 1. *Die* Baumstruktur $\mathcal{M}^\tau$ *für einen Baum* τ *ist eine Struktur erster Ordnung mit Universum* D_τ*. Sie umfasst eine Labeling-Relation* $:f^\tau \subseteq D_\tau^{\mathsf{ar}(f)+1}$

für jedes $f \in \Sigma$:

$$:f^\tau = \{(\pi, \pi 1, \ldots, \pi n) \mid L_\tau(\pi) = f, \mathsf{ar}(f) = n\}$$

Wir schreiben für $(\pi, \pi_1, \ldots, \pi_n) \in :f^\tau$ auch $\mathcal{M}^\tau \models \pi{:}f(\pi_1, \ldots, \pi_n)$. Diese Relation besagt, dass der Knoten π von τ das Label f trägt und π_i als i-tes Kind hat (für $1 \leq i \leq n$).

Man kann eine Baumstruktur $\mathcal{M}^\tau$ um zusätzliche Relationen erweitern, die durch die Labeling-Relation schon vollständig festgelegt sind: Die *Dominanz-Relation* $\pi \triangleleft^* \pi'$ ist die Präfix-Relation zwischen Pfaden. Wir schreiben außerdem $\pi \triangleleft^+ \pi'$, falls $\pi \triangleleft^* \pi'$ gilt und $\pi \neq \pi'$ ist. *Disjunktheit* $\pi \perp \pi'$ gilt, falls weder $\pi \triangleleft^* \pi'$ noch $\pi' \triangleleft^* \pi$ gilt.

Für die *Parallelismus-Relation* brauchen wir noch etwas zusätzliche Notation. Falls $\pi_1 \triangleleft^* \pi_2$ in $\mathcal{M}^\tau$ gilt, dann sei $\mathsf{betw}_\tau(\pi_1, \pi_2)$ die Menge aller Knoten *zwischen* π_1 und π_2:

$$\mathsf{betw}(\pi_1, \pi_2) = \{\pi \in D_\tau \mid \pi_1 \triangleleft^* \pi, \text{ aber nicht } \pi_2 \triangleleft^+ \pi\}$$

In einer Parallelismus-Relation $\pi_1/\pi_2 \sim \pi_3/\pi_4$ heißen π_1 und π_3 die *Wurzeln*, π_2 und π_4 die *Kronen* der zwei parallelen *Kontexte* $\mathsf{betw}_\tau(\pi_1, \pi_2)$ und $\mathsf{betw}_\tau(\pi_3, \pi_4)$. Die Kronen spielen insofern eine Sonderrolle, als sie selbst zwar zu den parallelen Kontexten gehören, ihre Labels aber nicht.

Definition 2. Parallelismus $\pi_1/\pi_2 \sim \pi_3/\pi_4$ *gilt in* $\mathcal{M}^\tau$ *genau dann, wenn* $\pi_1 \triangleleft^* \pi_2$ *und* $\pi_3 \triangleleft^* \pi_4$ *gelten und eine* Korrespondenzfunktion $c : \mathsf{betw}_\tau(\pi_1, \pi_2) \to \mathsf{betw}_\tau(\pi_3, \pi_4)$ *existiert, eine bijektive Funktion mit* $c(\pi_1) = \pi_3$ *und* $c(\pi_2) = \pi_4$, *die die Baumstruktur von* $\mathcal{M}^\tau$ *erhält: Für alle* $\pi \in \mathsf{betw}_\tau(\pi_1, \pi_2) - \{\pi_2\}$ *und* $f \in \Sigma$ *mit* $n = \mathsf{ar}(f)$ *muss gelten, dass*

$$\mathcal{M}^\tau \models \pi{:}f(\pi 1, \ldots, \pi n) \quad \Longleftrightarrow \quad \mathcal{M}^\tau \models c(\pi){:}f(c(\pi 1), \ldots, c(\pi n))$$

Das erzwingt schon, dass $c(\pi_1 \pi) = \pi_3 \pi$ ist für alle $\pi_1 \pi \in \mathsf{betw}_\tau(\pi_1, \pi_2)$.

Nach der Semantik nun zur Syntax: Wir nehmen eine Menge $\mathcal{V}$ von (Knoten-)Variablen $A, B, C, D, X, Y, Z, U, V, W \ldots$ an. Ein *Parallelismus-Constraint* φ ist eine Konjunktion von *atomaren Constraints* oder *Literalen* für Parallelismus, Dominanz, Labeling und Disjunktheit. Ein *Dominanz-Constraint* ist ein Constraint ohne Parallelismus-Literale. Ein Parallelismus-Constraint hat folgende abstrakte Syntax:

$$\begin{aligned} \varphi, \psi ::= {} & A/B \sim C/D \mid X \triangleleft^* Y \mid X{:}f(X_1, \ldots, X_n) \qquad (\mathsf{ar}(f) = n) \\ & \mid X \perp Y \mid X \neq Y \mid \mathbf{false} \mid \varphi \ \wedge \ \psi \end{aligned}$$

Abkürzungen: $X{=}Y$ für $X \triangleleft^* Y \wedge Y \triangleleft^* X$ und $X \triangleleft^+ Y$ für $X \triangleleft^* Y \wedge X \neq Y$

Der Einfachheit halber betrachten wir Parallelismus-, Ungleichheits- und Disjunktheits-Literale als symmetrisch.

Wir interpretieren Parallelismus-Constraints über der Klasse von Baumstrukturen in der üblichen Tarskischen Art. Wir schreiben $\mathcal{V}(\varphi)$ für die Variablen im

Constraint φ. Falls ein Paar $(\mathcal{M}^\tau, \alpha)$ aus einer Baumstruktur $\mathcal{M}^\tau$ und einer Variablenbelegung $\alpha : \mathcal{G} \to D_\tau$ (für irgendeine Menge $\mathcal{G} \supseteq \mathcal{V}(\varphi)$) φ erfüllt, so schreiben wir $(\mathcal{M}^\tau, \alpha) \models \varphi.$ und sagen, $(\mathcal{M}^\tau, \alpha)$ ist eine Lösung für φ. φ heißt *erfüllbar*, falls es eine Lösung besitzt.

Wir stellen Constraints oft als Graphen dar, wobei die Knoten die Variablen des Constraints repräsentieren. Eine gelabelte Variable wird mit ihren Kindern durch durchgezogene Linien verbunden, eine gepunktete Linie repräsentiert Dominanz. Zum Beispiel steht der Graph in Abb. 3 für den Constraint $X{:}f(X_1, X_2) \wedge X_1 \triangleleft^* Y \wedge X_2 \triangleleft^* Y$. Da Bäume nicht nach oben verzweigen, ist dieser Constraint unerfüllbar.

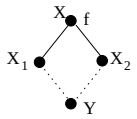


Abbildung 3. Ein unerfüllbarer Constraint

Atomaren Parallelismus stellen wir textuell und graphisch dar, letzteres in Form von eckigen Klammern wie in Abb. 4. (Dieser Constraint codiert übrigens das String-Unifikations-Problem [17] $gx = xg$; die zwei parallelen Kontexte repräsentieren die zwei Auftreten des x.) Literale für Ungleichheit und Disjunktheit werden ausschließlich textuell annotiert.

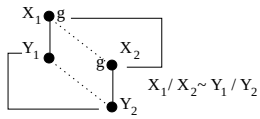


Abbildung 4. String-Unifikation

3 Unterspezifizierte Semantik

Wie in Abschnitt 1 gesagt, ist der Satz

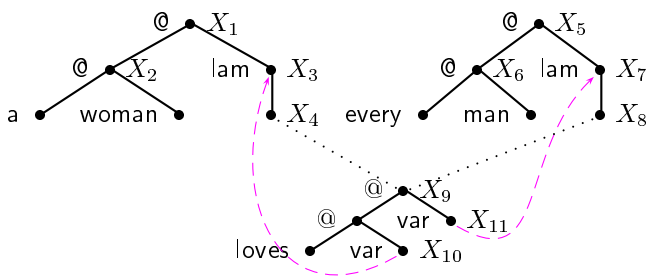
(1) Every man loves a woman.

ambig. Die zwei möglichen Lesarten sind

(2) $\forall x.(man(x) \to \exists y.(woman(y) \wedge loves(x, y)))$ und
(3) $\exists y.(woman(y) \wedge \forall x.(man(x) \to loves(x, y)))$.

Beide Formeln bestehen aus den Teilen $\forall x.(man(x) \to \_)$, $\exists y.(woman(y) \wedge \_)$ und $loves(x, y)$, diese Teilformeln sind nur in unterschiedlicher Reihenfolge zusammengesetzt. Wir beschreiben beide Lesarten unterspezifiziert durch folgenden Constraint:

(6)



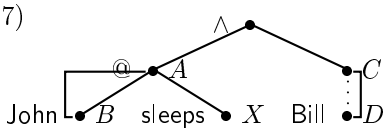
Zunächst einige Worte zur Darstellung: Einer langen Tradition der Computerlinguistik seit Montague [19] folgend, verwenden wir den einfach getypten

Lambda-Kalkül zur Beschreibung der Semantik eines Satzes. Damit sind die Bäume, die als Lösungen des Constraints in Frage kommen, ebenfalls Lambda-Terme. Wir verwenden das Knotenlabel lam für λ, var für Variablen, und @ für die funktionale Applikation. Wir identifizieren eine Variable mit dem λ, das sie bindet, nicht durch einen gleichen Namen („$\lambda x. \ldots x \ldots$“), sondern über einen *Lambda-Link*, der von der Baumposition der Variablen nach oben zum bindenden Lambda-Knoten zeigt (im Bild als gestrichelte Pfeile dargestellt). Für eine eingehende Behandlung von Lambda-Links siehe [7]. Dass wir darüber hinaus generalisierte Quantoren [2] verwenden, hat den Nebeneffekt, dass es die Formeln und somit auch die Graphen lesbarer macht; so vereinfacht sich (2) etwa zu $\mathsf{every\ man}\Big(\lambda x\ (\mathsf{a\ woman})\big(\lambda y\ (love\ y)x\big)\Big)$.

Wieso repräsentiert (6) nun genau die Lesarten (2) und (3)? Parallelismus-Constraints beschreiben Bäume, und in einem Baum können Knoten an disjunkten Positionen keinen gemeinsamen Nachfolger besitzen. Das heißt, wenn $(\mathcal{M}^\tau, \alpha)$ eine Lösung des Constraints in (6) ist, dann liegen $\alpha(X_1)$ und $\alpha(X_5)$ nicht an disjunkten Positionen, es gilt also entweder $\alpha(X_1) \triangleleft^* \alpha(X_5)$ oder $\alpha(X_5) \triangleleft^* \alpha(X_1)$. Genauere Betrachtung ergibt, dass die zwei gelabelten Graph-Fragmente für „every man“ und „a woman“ einander nicht überlappen können. Es muss also entweder $\alpha(X_4) \triangleleft^* \alpha(X_5)$ oder $\alpha(X_8) \triangleleft^* \alpha(X_1)$ gelten – ersteres entspricht Lesart (3), letzteres Lesart (2).

Die Semantik des elliptischen Satzes (4), „John sleeps, and so does Bill,“ kann man mit Hilfe eines Parallelismus-Literals ganz einfach so darstellen:

(7)



$A/B \sim C/D$

Wenn nun $(\mathcal{M}^\tau, \alpha)$ eine Lösung dieses Constraints ist, dann muss der Baum-Kontext zwischen $\alpha(A)$ und $\alpha(B)$ dieselbe Struktur aufweisen wie der zwischen $\alpha(C)$ und $\alpha(D)$ – also muss τ dem Term

$$and(sleeps(john),\ sleeps(bill))$$

entsprechen.

Genauso kann man bei Satz (5), „Every linguist attends a workshop, and every computer scientist does, too“ vorgehen. Der Constraint (8) für diesen Satz ist in Abb. 5 dargestellt. Wie kann eine Lösung $(\mathcal{M}^\tau, \alpha)$ dieses Constraints aussehen? Wie im Fall von (6) können $\alpha(X_1)$ und $\alpha(X_3)$ nicht disjunkt liegen, weil sie beide $\alpha(X_5)$ dominieren, ebenso $\alpha(X_1)$ und $\alpha(X_0)$. Und wieder können die gelabelten Graph-Fragmente sich nicht überlappen. Für das „a workshop“-Fragment ergeben sich insgesamt drei mögliche Positionen: Entweder es liegt oberhalb von $\alpha(X_0)$. Das entspricht Lesart a. Oder es liegt zwischen $\alpha(A)$ und $\alpha(X_3)$. Dann gilt $\alpha(X_2) \triangleleft^* \alpha(X_3)$ in $\mathcal{M}^\tau$. Nun erzwingt die Parallelismus-Relation, dass der Kontext zwischen $\alpha(A)$ und $\alpha(B)$ dieselbe Struktur hat wie der zwischen $\alpha(C)$ und $\alpha(D)$. Also gibt es zwischen $\alpha(C)$ und $\alpha(D)$ auch ein „a workshop“-Baum-Fragment, und zwar muss es wegen der Strukturgleichheit oberhalb vom „every

(8)

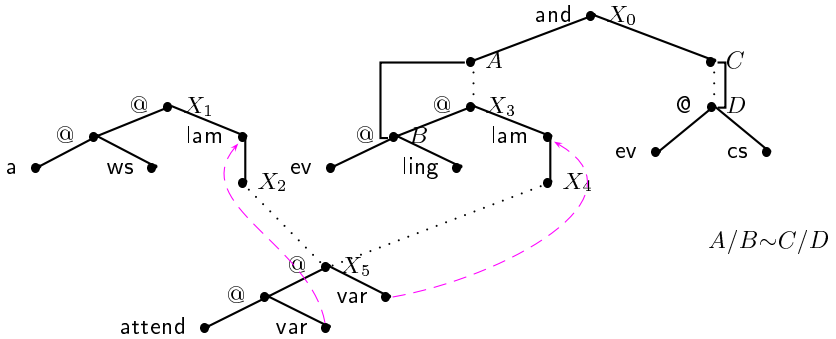


Abbildung 5. „Every linguist attends a workshop, and every computer scientist does, too.“

computer scientist“-Fragment liegen. Also ergibt sich hier Lesart b. Die dritte mögliche Position des „a workshop“-Fragments, zwischen $\alpha(X_3)$ und $\alpha(X_4)$, ergibt Lesart c. Andere Möglichkeiten gibt es nicht.

4 Ein Algorithmus für Dominanzconstraints

Wir stellen in diesem Abschnitt zunächst einen Algorithmus für Dominanz-Constraints (der von [6] subsumiert wird) vor, den wir im folgenden Abschnitt zu einer Prozedur für Parallelismus-Constraints erweitern. Dominanz-Constraints sind in der Linguistik seit langem bekannt und werden vielseitig genutzt [18, 32, 1, 21, 10]. Ihr Erfüllbarkeitsproblem ist NP-vollständig [15], aber es gibt dafür schnelle Lösungsalgorithmen auf Constraint-Basis [6].

Die Idee des Constraint-Lösens ist, beim Durchlaufen eines Suchbaumes Sackgassen möglichst früh zu erkennen. Dazu werden in jedem Knoten des Suchbaumes zuerst *Propagierungsschritte* (Inferenzen) ausgeführt, bevor eine Fallunterscheidung, ein *Distributionsschritt*, unternommen wird. Die Verfahren, die wir vorstellen, *saturieren* einen Constraint: Ein Saturierungsverfahren ist Menge S von Regeln der Form $\varphi \to \bigvee_{i=1}^{n} \varphi_i$. Jede Regel beschreibt eine Bedingung, unter der ein Constraint um weitere Konjunkte erweitert werden kann. Falls $n = 1$ ist, liegt eine *Propagierungs-Regel* vor, sonst eine *Distributions-Regel*.

Der Lesbarkeit halber identifizieren wir im Folgenden einen Constraint mit der Menge seiner Literale. Wir nehmen an, dass zu jedem Constraint $\rho \in S$ eine Anwendungsbedingung $C_\rho(\varphi)$ gegeben ist, die entscheidet, ob ρ auf φ angewendet werden kann. Ein *Saturierungs-Schritt* $\to_S$ besteht aus einer einzelnen

Propagierungsregeln:

(D.Clash.Ineq) $X{=}Y \ \wedge \ X{\neq}Y \rightarrow \mathbf{false}$

(D.Clash.Disj) $X \perp X \rightarrow \mathbf{false}$

(D.Dom.Refl) $\varphi \rightarrow X \triangleleft^* X$ — falls $X \in \mathcal{V}(\varphi)$

(D.Dom.Trans) $X \triangleleft^* Y \ \wedge \ Y \triangleleft^* Z \rightarrow X \triangleleft^* Z$

(D.Eq.Decom) $X{:}f(X_1, \ldots, X_n) \ \wedge \ Y{:}f(Y_1, \ldots, Y_n) \ \wedge \ X{=}Y \rightarrow \bigwedge_{i=1}^{n} X_i{=}Y_i$

(D.Lab.Ineq) $X{:}f(\ldots) \ \wedge \ Y{:}g(\ldots) \rightarrow X{\neq}Y$ — für $f \neq g$

(D.Lab.Disj) $X{:}f(\ldots X_i, \ldots, X_j, \ldots) \rightarrow X_i \perp X_j$ — für $1 \leq i < j \leq n$

(D.Prop.Disj) $X \perp Y \ \wedge \ X \triangleleft^* X' \ \wedge \ Y \triangleleft^* Y' \rightarrow Y' \perp X'$

(D.Lab.Dom) $X{:}f(\ldots, Y, \ldots) \rightarrow X \triangleleft^+ Y$

Distributionsregeln:

(D.Distr.Child) $X \triangleleft^* Y \ \wedge \ X{:}f(X_1, \ldots, X_n) \rightarrow Y{=}X \ \vee \ \bigvee_{i=1}^{n} X_i \triangleleft^* Y$

(D.Distr.NotDisj) $X \triangleleft^* Z \ \wedge \ Y \triangleleft^* Z \rightarrow X \triangleleft^* Y \ \vee \ Y \triangleleft^* X$

Abbildung 6. Regelschemata für Algorithmus D: Dominanz-Constraints

Regelanwendung:

$$\frac{\psi \subseteq \varphi \quad \rho \in S}{\varphi \rightarrow_S \varphi \wedge \psi_i} \text{ falls } C_\rho(\varphi), \text{ und } \rho \text{ ist } \psi \rightarrow \bigvee_{i=1}^{n} \psi_i.$$

Zunächst reicht es aus, festzusetzen: $C_{\psi \rightarrow \vee_{i=1}^{n} \psi_i}$ sei wahr, wenn $\psi_i \not\subseteq \varphi$ ist für $1 \leq i \leq n$. Wir nennen einen Constraint S-*saturiert*, wenn er bezüglich $\rightarrow_S$ nicht mehr reduzierbar ist, und *clash-frei*, wenn er **false** nicht enthält.

Abbildung 6 zeigt Regelschemata, deren (unendliche) Menge von Instanzen den Algorithmus D für Dominanz-Constraints bilden. Der Algorithmus, der korrekt und vollständig ist, findet sämtliche *clash*-freie Saturierungen des Eingabe-Constraints. Die Propagierungsschritte erzwingen „Baumförmigkeit“, z.B. besagt (A.Dom.Trans), dass Dominanz transitiv ist, und (A.Eq.Decom) ist ein Dekompositionsschritt.

Es gibt nur zwei Situationen, in denen distribuiert werden muss. In der Situation in Abb. 7 wird (D.Distr.NotDisj) aktiv: Hier muss entweder $X \triangleleft^* Y$ oder $Y \triangleleft^* X$ gelten. Dies ist exakt die Situation, die in Constraints für Skopus-ambige Sätze auftritt.

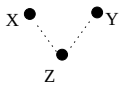


Abbildung 7. Skopus-Situation

Abb. 8 zeigt einen Fall, in dem (D.Distr.Child) greift: Es muss entweder $Y{=}X$ oder $X_1 \triangleleft^* Y$ oder $X_2 \triangleleft^* Y$ gelten. Mit $Y{=}X$ ergibt sich

hier ein unerfüllbarer Constraint, da Y mit g gelabelt ist. Die anderen beiden Möglichkeiten führen zu erfüllbaren Constraints.

Beispiel 3. Sehen wir uns den unerfüllbaren Constraint

$$X{:}f(X_1, X_2) \wedge X_1 \triangleleft^* Y \wedge X_2 \triangleleft^* Y$$

an: Nach (D.Lab.Disj) können wir $X_1 \perp X_2$ dazunehmen. Dann wird aber (D.Prop.Disj) anwendbar und wir erhalten das neue Konjunkt $Y \perp Y$, da sowohl $X_1 \triangleleft^* Y$ als auch $X_2 \triangleleft^* Y$ vorhanden ist. Aus $Y \perp Y$ folgert (D.Clash.Disj) nun **false**.

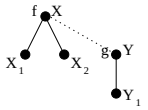
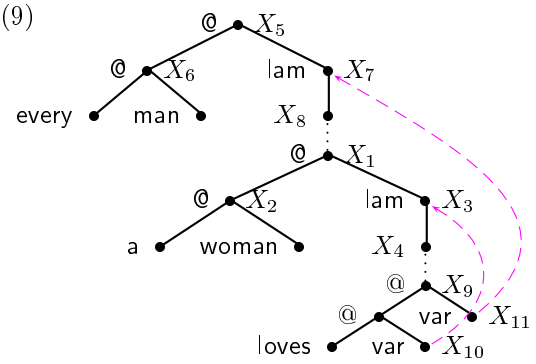


Abbildung 8. Überlappung?

Beispiel 4. Mit Algorithmus D können wir Constraint (6), die Semantik von „every man loves a woman," lösen.

X_4, X_8 und X_9 bilden ein „Dreieck" wie in Abb. 7, damit ist (D.Distr.Not Disj) anwendbar: Es muss entweder $X_4 \triangleleft^* X_8$ oder $X_8 \triangleleft^* X_4$ dazugenommen werden. Verfolgen wir den Fall $X_8 \triangleleft^* X_4$ weiter. Der Algorithmus muss jetzt nur noch feststellen, dass man das „every man"- und das „a woman"-Fragment nicht überlappen kann. Zunächst findet sich ein weiteres „Dreieck," nämlich $X_8 \triangleleft^* X_4 \wedge X_3 \triangleleft^* X_4$, auf das wir wieder (D.Distr.NotDisj) anwenden. Der Fall $X_3 \triangleleft^* X_8$ führt zu einem *Clash*. Verfolgt man den Fall $X_8 \triangleleft^* X_3$ weiter, erreicht man nach einigen weiteren Rechenschritten eine *clash*-freie Saturation:

(9)



Dieser Constraint ist fast baumförmig – er enthält noch „bewegliche", nur durch Dominanzen verbundene Teile. Aber es ist intendiert, dass der Algorithmus hier nicht X_8 und X_1 bzw. X_4 und X_9 identifiziert, da für die Behandlung anderer Phänomene [14] eventuell an solchen „beweglichen" Stellen noch Constraints eingefügt werden müssen.

Nachdem die Skopus-Lesart mit der ersten Anwendung von (D.Distr.NotDisj) entschieden ist, muss Algorithmus D distribuieren, um festzustellen, dass man das „every man"- nicht mit dem „a woman"-Fragment überlappen kann. Aber das Distribuieren ist hier nicht unbedingt notwendig: Ein um weitere Saturierungsregeln erweiterter Algorithmus in [6] kommt zu demselben Schluss rein durch Propagieren.

5 Eine Prozedur für Parallelismus-Constraints

Wir ergänzen Algorithmus D aus dem vorigen Abschnitt um weitere Saturierungsregeln zu einer Semi-Entscheidungs-Prozedur für Parallelismus-Constraints. Dadurch erben wir für die neue Prozedur die gute Verarbeitung von Dominanz-Constraints. Statt mit Algorithmus D kann man die Parallelismus-Regeln auch mit dem erwähnten erweiterten Dominanz-Constraint-Algorithmus aus [6] kombinieren.

Bei der Verarbeitung von Parallelismus geht es darum, die Korrespondenz-Funktionen für alle Parallelismus-Literale des Eingabe-Constraints zu berechnen. Dazu verwenden wir *Pfadgleichheits-Literale*, um das Problem mit möglichst viel Propagieren und möglichst wenig Distribution zu lösen.

Wir definieren die Menge $\mathsf{betw}_\varphi(A, B)$ von Variablen *zwischen* A und B analog zu $\mathsf{betw}_\tau(\pi_1, \pi_2)$: Falls $X_1 \triangleleft^* X_2 \in \varphi$ ist, dann ist

$$\mathsf{betw}_\varphi(A, B) := \{X \in \mathcal{V}(\varphi) \mid A \triangleleft^* X \in \varphi \text{ und entweder } X \triangleleft^* B \in \varphi \text{ oder } X \perp B \in \varphi\}.$$

Für jede Variable $X \in \mathsf{betw}_\varphi(A, B)$ muss die Prozedur nun eine *Korrespondierende* im Kontext $\mathsf{betw}_\varphi(C, D)$ finden. Dazu müssen unter Umständen lokale Variablen eingeführt werden. Im Folgenden betrachten wir einen Constraint φ immer zusammen mit einer Menge $\mathcal{G} \subseteq \mathcal{V}$ von *globalen* Variablen; alle anderen Variablen sind *lokal*. Ist φ eine Eingabe für die Prozedur, dann sei immer $\mathcal{V}(\varphi) \subseteq \mathcal{V}$. Für einen Constraint φ mit lokalen Variablen ist ein Paar $(\mathcal{M}^\tau, \alpha)$ eine Lösung, falls $(\mathcal{M}^\tau, \alpha) \models \exists\big(\mathcal{V}(\varphi) - \mathcal{G}\big).\varphi$ gilt.

Um Korrespondenz zu notieren, verwenden wir Pfadgleichheits-Literale, Hilfsconstraints, die im Lauf der Berechnung eingeführt werden, aber nicht zur Constraint-Sprache gehören. Ein Pfadgleichheits-Literal $\mathrm{p}\left(\begin{smallmatrix} A & C \\ X & Y \end{smallmatrix}\right)$ sagt aus, dass X unter A zu Y unter C korrespondiert. Genauer gesagt: Eine Pfadgleichheits-Relation $\mathrm{p}\left(\begin{smallmatrix} \pi_1 & \pi_3 \\ \pi_2 & \pi_4 \end{smallmatrix}\right)$ gilt in einer Baumstruktur $\mathcal{M}^\tau$ genau dann, wenn es einen Pfad π gibt, so dass erstens $\pi_2 = \pi_1\pi$ und $\pi_4 = \pi_3\pi$ ist und zweitens für jedes $\pi' \triangleleft^+ \pi$ gilt, dass $L_\tau(\pi_1\pi') = L_\tau(\pi_3\pi')$ ist.

Abbildung 9 zeigt die Schemata der Regelmengen P und N, die Korrespondenzen berechnen, und Abbildung 12 zeigt die Schemata der Menge T für interagierende Parallelismus-Literale. $D \cup P \cup N \cup T$, abgekürzt $DPNT$,[1] bildet eine korrekte und vollständige Semi-Entscheidungs-Prozedur für Parallelismus-Constraints [9].

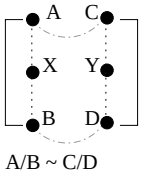


Abbildung 10. Korrespondenz

Gegeben ein Parallelismus-Literal $A/B \sim C/D$, stellt Regelschema (P.Root) fest, dass die beiden Wurzel-Variablen A und C zueinander korrespondieren, ebenso die beiden Kronen-Variablen B und D. Wie findet man nun Korrespondierende für die restlichen Variablen im

[1] Entsprechende Abkürzungen verwenden wir auch für andere Regelmengen-Kombinationen.

Propagierungsregeln:

(P.Root)	$A/B{\sim}C/D \to \mathrm{p}\binom{A\ C}{A\ C} \ \wedge\ \mathrm{p}\binom{A\ C}{B\ D}$
(P.Copy.Dom)	$U_1 R U_2 \wedge \bigwedge_{i=1}^{2} \mathrm{p}\binom{A\ C}{U_i\ V_i} \to V_1 R V_2$ — für $R \in \{\triangleleft^*, \perp, \neq\}$, $A/B{\sim}C/D \in \varphi$, $U_1, U_2 \in \mathsf{betw}_\varphi(A, B)$
(P.Copy.Lab)	$U_0{:}f(U_1, \ldots, U_n) \ \wedge\ \bigwedge_{i=0}^{n} \mathrm{p}\binom{A\ C}{U_i\ V_i} \to V_0{:}f(V_1, \ldots, V_n)$ — für $A/B{\sim}C/D \in \varphi$, $U_0, \ldots, U_n \in \mathsf{betw}_\varphi(A, B)$
(P.Path.Sym)	$\mathrm{p}\binom{X\ Y}{U\ V} \to \mathrm{p}\binom{Y\ X}{V\ U}$
(P.Path.Dom)	$\mathrm{p}\binom{X\ Y}{U\ V} \to X \triangleleft^* U \ \wedge\ Y \triangleleft^* V$
(P.Path.Eq.1)	$\mathrm{p}\binom{X_1\ X_3}{X_2\ X_4} \ \wedge\ \bigwedge_{i=1}^{4} X_i{=}Y_i \to \mathrm{p}\binom{Y_1\ Y_3}{Y_2\ Y_4}$
(P.Path.Eq.2)	$\mathrm{p}\binom{X\ X}{U\ V} \to U{=}V$

Distributionsregeln:

(P.Distr.Crown)	$A \triangleleft^* X \to X \triangleleft^* B \vee X \perp B \vee B \triangleleft^+ X$ — für $A/B{\sim}C/D \in \varphi$
(P.Distr.Project)	$\varphi \to X{=}Y \ \vee\ X{\neq}Y$ — für $X, Y \in \mathcal{V}(\varphi)$

Einführung neuer Variablen:

(N.New)	$\varphi \to \mathrm{p}\binom{A\ C}{X\ X'}$ — für $A/B{\sim}C/D \in \varphi$ und $X \in \mathsf{betw}_\varphi(A, B)$; X' ist neu und lokal

Abbildung 9. Schemata der Regelmengen P und N für Parallelismus

Parallelismus-Kontext? Sehen wir uns den Constraint in Abb. 10 an. X liegt im Kontext zwischen A und B, Y im Kontext zwischen C und D. Aber sie werden von A bzw. C nur dominiert, über ihre genaue Position im Kontext ist nichts bekannt. Deshalb wäre es vorschnell, X einfach in Korrespondenz zu Y zu setzen, solange nichts vorliegt, was uns zu diesem Schritt zwänge. Diese Einsicht setzt (N.New) so um: Für ein Parallelismus-Literal $A/B{\sim}C/D$ und eine Variable $X \in \mathsf{betw}_\varphi(A, B)$ wird eine Korrespondenz $\mathrm{p}\binom{A\ C}{X\ X'}$ ausgesagt für eine *neue lokale* Variable X'. Dabei heißt „neu“, dass $X' \notin \mathcal{V}(\varphi)$, und „lokal“, dass $X' \notin \mathcal{G}$ ist. Falls sich herausstellen sollte, dass die Struktur des Constraints eine Korrespondenz zwischen X und Y erzwingt, dann wird $Y{=}X'$ schon geschlussfolgert durch eine Kombination der anderen Regeln. (N.New) braucht nur dann angewendet zu werden, wenn X für *dies* Parallelismus-Literal noch keine Korrespondierende besitzt. Das schlägt sich in einer abgeänderten Anwendungsbedingung für alle Regeln des Schemas (N.New) nieder:

$$C_{\varphi' \to \mathrm{p}\binom{A\ C}{X\ X}'}(\varphi) \text{ gilt gdw. } X' \notin \mathcal{V}(\varphi) \cup \mathcal{G} \text{ und } \forall Y \in \mathcal{V}.\mathrm{p}\binom{A\ C}{X\ Y} \notin \varphi.$$

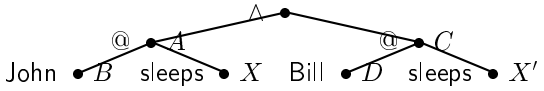
Für ein Parallelismus-Literal $A/B{\sim}C/D \in \varphi$ kopieren (P.Copy.Dom) und (P.Copy.Lab) Dominanz-, Disjunktheits-, Ungleichheits- und Labeling-Literale von $\mathsf{betw}_\varphi(A,B)$ nach $\mathsf{betw}_\varphi(C,D)$ und umgekehrt. Die Seitenbedingung in (P.Copy.Lab), die die Position von U_0 einschränkt, stellt sicher, dass die Label der Kronen B und D nicht kopiert werden.

Darüber hinaus enthält P zwei Distributions-Schemata: (P.Distr.Crown) entscheidet für Variablen, die potentiell in einem Parallelismus-Kontext sein könnten (d.h. $A{\lhd^*}X$), ob sie tatsächlich im Kontext liegen ($X{\lhd^*}B \vee X{\perp}B$) oder nicht ($B{\lhd^+}X$). (P.Distr.Project) rät, ob zwei Variablen identifiziert werden sollten oder nicht. In der Praxis möchte man diese Regel nicht unbedingt anwenden; sie ist zu teuer.

Schon mit den bisher behandelten Regeln in *DPN* können wir die verbleibenden Beispiele aus Abschnitt 3 berechnen. Wir zeigen, wie das geht, bevor wir uns der Regelmenge T für interagierende Parallelismus-Literale zuwenden.

Beispiel 5. Der Constraint (7) stellt die unterspezifizierte Semantik des Satzes „John sleeps, and so does Bill“ dar. Dieser Constraint kann mit *DPN* wie folgt saturiert werden: Mit (P.Root) erhalten wir zunächst $\mathrm{p}(\begin{smallmatrix}A & C\\ A & C\end{smallmatrix}) \wedge \mathrm{p}(\begin{smallmatrix}A & C\\ B & D\end{smallmatrix})$. Zu X führen wir mit (N.New) eine neue, lokale Variable X' ein sowie den Constraint $\mathrm{p}(\begin{smallmatrix}A & C\\ X & X'\end{smallmatrix})$. (P.Copy.Lab) ergänzt $C{:}@(D,X') \wedge X' : \mathit{sleeps}$. Als Graph sieht das so aus:

(10) $A/B{\sim}C/D$



Beispiel 6. Der Constraint im letzten Beispiel war sehr einfach und schnell zu saturieren. Das ist typisch für Constraints, die elliptische Sätze beschreiben. Es lassen sich aber auch komplexere Beispiele konstruieren, etwa das in Abb. 11. Hier ist $C \in \mathsf{betw}_\varphi(A,B)$, da sich die Kontexte überlappen. Die Anwendung von (N.New) auf C ergibt $\mathrm{p}(\begin{smallmatrix}A & C\\ C & C'\end{smallmatrix})$ für eine neue, lokale Variable C'. Nach (P.Copy.Dom) gilt $C{\lhd^*}C'{\lhd^*}D$. Damit haben wir ein „Dreieck“: $C'{\lhd^*}D \wedge B{\lhd^*}D$, und (D.Distr.NotDisj) kommt zum Einsatz. Im Fall $B{\lhd^*}C'$ bleibt nur noch eine Korrespondierende für $B \in \mathsf{betw}_\varphi(C,D)$ zu finden. Für $C'{\lhd^*}B$ dagegen müssen wir zusätzlich für $C' \in \mathsf{betw}_\varphi(A,B)$ eine Korrespondierende in $\mathsf{betw}_\varphi(C,D)$ finden – der oben beschriebene Ablauf wiederholt sich.

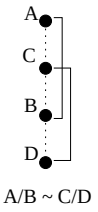


Abbildung 11. Selbst-Überlappung

Beispiel 7. Der Constraint in (8) beschreibt die unterspezifizierte Semantik des Satzes „Every linguist attends a workshop, and every computer scientist does, too.“ Zunächst kann Algorithmus D desambiguieren wie in Bsp. 4. Da gibt es zum einen das Dreieck $X_1{\lhd^*}X_5 \wedge X_3{\lhd^*}X_5$, zum anderen $X_0{\lhd^*}X_5 \wedge X_1{\lhd^*}X_5$. Insgesamt errechnet D drei mögliche Positionen der gelabelten Graph-Fragmente:

(T.Trans.H) $\mathrm{p}\binom{X\ Y}{U\ V} \wedge \mathrm{p}\binom{Y\ Z}{V\ W} \rightarrow \mathrm{p}\binom{X\ Z}{U\ W}$

(T.Trans.V) $\mathrm{p}\binom{X_1\ Y_1}{X_2\ Y_2} \wedge \mathrm{p}\binom{X_2\ Y_2}{X_3\ Y_3} \rightarrow \mathrm{p}\binom{X_1\ Y_1}{X_3\ Y_3}$

(T.Diff.1) $\mathrm{p}\binom{X_1\ Y_1}{X_2\ Y_2} \wedge \mathrm{p}\binom{X_1\ Y_1}{X_3\ Y_3} \rightarrow \mathrm{p}\binom{X_2\ Y_2}{X_3\ Y_3}$ — falls $X_2 \triangleleft^* X_3$, $Y_2 \triangleleft^* Y_3 \in \varphi$

(T.Diff.2) $\mathrm{p}\binom{X_1\ Y_1}{X_3\ Y_3} \wedge \mathrm{p}\binom{X_2\ Y_2}{X_3\ Y_3} \rightarrow \mathrm{p}\binom{X_1\ Y_1}{X_2\ Y_2}$ — falls $X_1 \triangleleft^* X_2$, $Y_1 \triangleleft^* Y_2 \in \varphi$

Abbildung 12. T propagiert Transitivität von Pfadgleichheits-Literalen

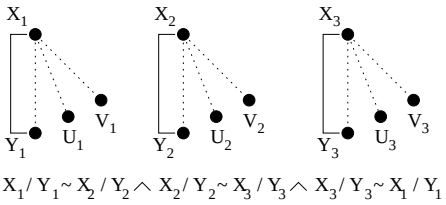


Abbildung 13. Hier ist T vonnöten

Entweder $X_1 \triangleleft^* X_0$ oder $A \triangleleft^* X_1 \wedge X_2 \triangleleft^* X_3$ oder $X_3 \triangleleft^* X_1 \wedge X_2 \triangleleft^* X_5$. In den letzteren beiden Fällen liegt das „a workshop"-Fragment in $\mathsf{betw}_\varphi(A, B)$. Dann sorgen (P.Copy.Dom) und (P.Copy.Lab) dafür, dass die beiden parallelen Kontexte strukturgleich sind: Ist etwa $\mathrm{p}\binom{A\ C}{X_1\ Y_1}$ und $\mathrm{p}\binom{A\ C}{X_3\ Y_3}$ und $X_3 \triangleleft^* X_1$, so erzwingt (P.Copy.Dom), dass auch $Y_3 \triangleleft^* Y_1$ dem Constraint hinzugefügt wird. Also ergeben sich genau drei Saturierungen des Constraints, die den drei Lesarten entsprechen. Die saturierten Constraints für Lesarten b und c zeigt der Bildschirmausschnitt in Abb. 15.

Interagierende Parallelismus-Literale

Die Propagierungs-Regeln der Menge T (Abb. 12) sorgen dafür, dass die Korrespondenz-Funktionen „überlappender" Parallelismus-Kontexte in der richtigen Weise interagieren. (T.Trans.H) beschreibt die horizontale Transitivität von Pfadgleichheits-Literalen, während (T.Trans.V), (T.Diff.1) und (T.Diff.2) die vertikale Transitivität behandeln.

Sehen wir uns ein Beispiel an, wo T die korrekte Interaktion von Korrespondenz-Funktionen sicherstellt: den Constraint in Abb. 13. Dort gilt $X_i \triangleleft^* U_i$ und $X_i \triangleleft^* V_i$ für $1 \leq i \leq 3$, insofern ist (P.Distr.Crown) anwendbar. Nehmen wir an, dass jeweils $U_i \perp Y_i$ und $V_i \perp Y_i$ gewählt werden. Nehmen wir außerdem an, dass (P.Distr.Project) $U_1 \neq V_1$ gewählt hat. Mit den Regeln in P werden nun wie üblich die Korrespondenz-Funktionen der drei Parallelismus-Literale berechnet. Nehmen wir an, dass zufällig durch (P.Distr.Project) die neuen Variablen so mit alten, globalen Variablen gleichgesetzt werden, dass folgen-

de Pfadgleichheiten gelten: $\mathrm{p}(\begin{smallmatrix}X_1 & X_2\\ U_1 & U_2\end{smallmatrix})$, $\mathrm{p}(\begin{smallmatrix}X_1 & X_2\\ V_1 & V_2\end{smallmatrix})$ für die Korrespondenz-Funktion von $X_1/Y_1 \sim X_2/Y_2$; $\mathrm{p}(\begin{smallmatrix}X_2 & X_3\\ U_2 & U_3\end{smallmatrix})$, $\mathrm{p}(\begin{smallmatrix}X_2 & X_3\\ V_2 & V_3\end{smallmatrix})$ für die Korrespondenz-Funktion von $X_2/Y_2 \sim X_3/Y_3$; und (nun kommen wir zum zentralen Punkt) $\mathrm{p}(\begin{smallmatrix}X_3 & X_1\\ V_3 & U_1\end{smallmatrix})$, $\mathrm{p}(\begin{smallmatrix}X_3 & X_1\\ V_3 & U_1\end{smallmatrix})$ für die Korrespondenz-Funktion von $X_3/Y_3 \sim X_1/Y_1$. Nun ist der Constraint unerfüllbar: Wenn eine Baumstruktur diesen Constraint erfüllte, dann müsste darin der Pfad vom X_1-Knoten zum U_1-Knoten derselbe sein wie der Pfad vom X_1- zum V_1-Knoten – aber der Constraint enthält $U_1 \neq V_1$. Nur mit T kann eine solche Konstellation erkannt werden, in diesem Fall mit (T.Trans.H): Aus $\mathrm{p}(\begin{smallmatrix}X_1 & X_2\\ U_1 & U_2\end{smallmatrix})$ und $\mathrm{p}(\begin{smallmatrix}X_2 & X_3\\ U_2 & U_3\end{smallmatrix})$ schließen wir $\mathrm{p}(\begin{smallmatrix}X_1 & X_3\\ U_1 & U_3\end{smallmatrix})$, was wir weiter mit $\mathrm{p}(\begin{smallmatrix}X_3 & X_1\\ U_3 & V_1\end{smallmatrix})$ kombinieren zu $\mathrm{p}(\begin{smallmatrix}X_1 & X_1\\ U_1 & V_1\end{smallmatrix})$. Nach (P.Path.Eq.2) ergibt das $U_1 = V_1$ und somit einen *clash* durch (D.Clash.Ineq).

Die Regeln in T operieren also unabhängig davon, zu welchem Parallelismus-Literal ein Pfadgleichheits-Constraint gehört, und erschließen dadurch neue Pfadgleichheiten, die über die Korrespondenzfunktionen der einzelnen Parallelismen hinausgehen. Das ist der Vorteil darin, Korrespondenzen in Form von Pfadgleichheits-Literalen zu notieren und nicht direkt in Form syntaktischer Korrespondenz-Funktionen.

6 Das CHORUS Demo System

In einer ersten Implementierung von *DPNT* wurden die Saturierungs-Regeln direkt umgesetzt, ohne Optimierungen. Dabei werden die Regeln in folgender Reihenfolge angewendet:

- Ein Constraint wird jeweils unter allen Propagierungsregeln in *DPT* saturiert, bevor eine Fallentscheidung getroffen wird.
- Erst wenn der Constraint unter *DPT* (bis auf (P.Distr.Project)) saturiert ist, wird einmal (N.New) angewendet.
- (P.Distr.Project) wird gar nicht benutzt.

Bislang haben wir nur wenige Constraints gefunden, bei denen (P.Distr.Project) zur Saturierung nötig ist. Dabei sind jeweils mehrfach selbstüberlappende Parallelismus-Literale im Spiel, und es handelt sich nicht um linguistisch relevante Constraints. Dagegen gibt es durchaus Fälle von mehreren interagierenden Parallelismus-Literalen. So enthält der Constraint für

(11) John revised a paper before the teacher did, and so did Bill.

zwei geschachtelte Parallelismus-Literale: Der innere Parallelismus beschreibt „John revised a paper before the teacher revised a paper“. Beide Kontexte dieses inneren Parallelismus sind im linken Kontext des äußeren Parallelismus enthalten. Der äußere Parallelismus beschreibt „John revised ... and Bill revised. ...“

Es ist nicht von vornherein offensichtlich, was berechnungsmäßig teurer ist, die Fallunterscheidung mit (P.Distr.Project) oder die Propagierung mit T, die eine große Menge vierstelliger Constraints inferiert. Aber unserer Intuition nach sollte eine so allgemeine und damit so mächtige Distributionsregel wie

(P.Distr.Project) für die Beispiele aus der Linguistik nicht nötig sein. Hier sind allerdings weitere Untersuchungen vonnöten.

Über die in diesem Text beschriebene Constraint-Sprache hinaus kann die Implementierung noch mit Anaphern umgehen. Das sind textliche Bezüge: In „John visits his mother“ kann sich „his“ z.B. auf „John“ beziehen. Man kann anaphorische Bezüge ähnlich wie Lambda-Links als Querkanten durch den Graphen realisieren. Auch bei Anaphern ergeben sich interessante Interaktionen mit Ellipsen [7].

Diese erste Implementierung von *DPNT* ist Teil des *CHORUS Demo Systems* [3], das in Mozart Oz [31, 20] geschrieben ist. Spannend am *CHORUS Demo System* ist, dass es nicht nur für einen gegebenen Constraint eine Saturierung erzeugt, sondern auch den Constraint, der die unterspezifizierte Semantik eines Satzes repräsentiert, herleiten kann: Man kann sowohl Constraints als auch natürlichsprachliche Sätze eingeben. Letztere werden mittels einer HPSG-Grammatik [26] syntaktisch analysiert. Aus dem Analyse-Ergebnis wird die unterspezifizierte Semantik des Satzes automatisch konstruiert.

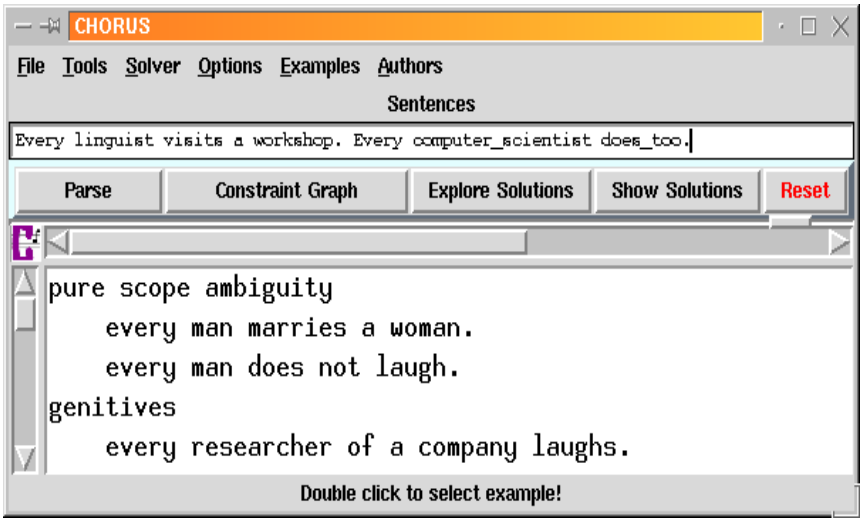


Abbildung 14. Das CHORUS-Demo-Programm

Abbildung 14 zeigt das Hauptfenster der CHORUS-Demo. Der obere Bereich des Fensters enthält das Material, das analysiert werden soll. Der untere Bereich des Fensters bietet eine Reihe von analysierbaren Sätzen zur Auswahl. Momentan ist das Lexikon und damit auch die Menge der verfügbaren Sätze noch klein. Mit der Schaltfläche „Constraint Graph“ kann man sich die unterspezifizierte Semantik des aktuellen Satzes graphisch anzeigen lassen. Wählt man „Explore Solutions“, so werden sämtliche Saturierungen dieses Constraints berechnet. Abbildung 15 zeigt zwei der drei Saturierungen, die für Constraint (8), die Se-

mantik von „Every linguist visits a workshop. Every computer scientist does, too,“ berechnet werden. Im linken saturierten Constraint hat „a workshop“ weiten Skopus — das ist Lesart b —, in der rechts angezeigten Saturierung engen Skopus (Lesart c). Die dritte Lesart wurde aus Gründen der Lesbarkeit unterdrückt.

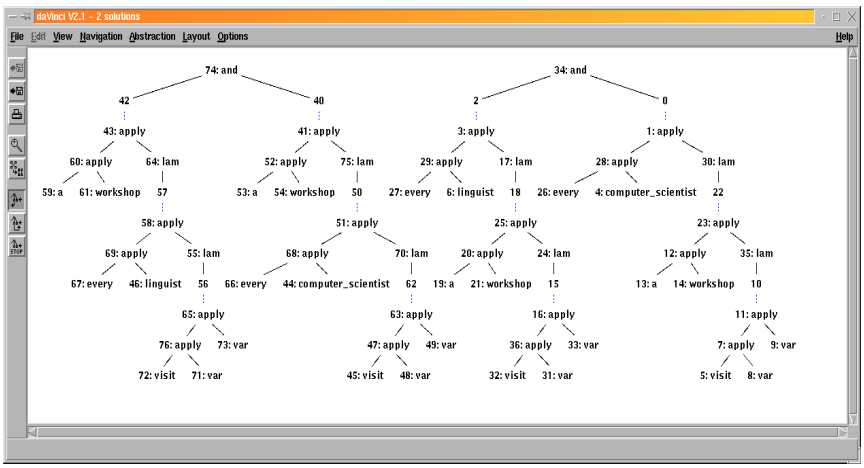


Abbildung 15. Zwei der drei Lösungen für Constraint (8)

Ein früherer Ansatz [23] verwendete Kontext-Unifikation zur Darstellung von unterspezifizierter Semantik. Dieses Verfahren konnte gut mit Ellipsen umgehen, hatte bei Skopusambiguitäten aber sehr schnell Probleme mit der kombinatorischen Explosion. Das liegt daran, dass es in der Kontext-Unifikation kein Äquivalent der Dominanz gibt; dadurch mussten Skopusambiguitäten mit einem eigentlich zu mächtigen Konstrukt beschrieben werden. Bessere Ergebnisse lieferte ein unvollständiges Verfahren, das aber immer noch Probleme hatte mit Sätzen mit mehr als 2 skopustragenden Elementen. Außerdem wurde nie genau nachgewiesen, für welche Fälle das unvollständige Verfahren ausreicht und für welche nicht. Tabelle 1 vergleicht die vollständige Kontext-Unifikations-Prozedur (CU), die unvollständige (CU unvollst.) und das hier vorgestellte Parallelismus-Constraint-Verfahren. Verwendet wurden die Sätze „Every man loves a woman“ (1) sowie

(12) Peter likes Mary. John does, too.
(13) Every researcher of a company saw most samples.

7 Die nächsten Ziele

Sowohl in theoretischer als auch in praktischer Hinsicht sind noch viele Fragen offen. Ein wichtiger Punkt ist, dass offenbar in der linguistischen Anwen-

	Laufzeiten		
Beispiel	CU	CU unvollst.	*DPNT*
(1)	40 sec.	1 sec	1 sec
(12)	2+ h	1 sec	4 sec
(13)	k.A.	15 sec	4 sec

Tabelle 1. Laufzeitvergleich des hier vorgestellten Verfahrens mit Vorgängersystemen

dung nicht die volle Mächtigkeit von Parallelismus-Constraints gebraucht wird. Damit stellt sich die Frage, wie sich das linguistisch relevante Fragment von Parallelismus-Constraints formal beschreiben lässt und ob sich zeigen lässt, dass der Parallelismus-Constraint-Löser für dies Fragment immer terminiert. Diese Frage ist von offensichtlichem praktischem, aber auch von theoretischem Interesse, da dies Fragment gleichzeitig ein neues entscheidbares Fragment der Kontext-Unifikation definieren könnte. Weitere theoretische Fragen ergeben sich aus dem Beweis der Korrektheit und Vollständigkeit der Prozedur [9]. Interessant ist, dass es sich hier um eine Art Unifikation auf flachen Relationen, nicht wie üblich auf Termen handelt. Eine andere spannende Frage ist die nach der linguistischen Abdeckung: Es gibt noch viel mehr und kompliziertere elliptische Konstruktionen als die, die wir hier vorgestellt haben. Können wir sie mit Parallelismus-Constraints beschreiben?

8 Danksagungen

Herzlichen Dank an Joachim Niehren, mit dem zusammen ich die hier berichteten Ergebnisse erarbeitet habe, sowie an Tobias Müller für seine vielen hilfreichen Kommentare zu diesem Text.

Literatur

1. R. Backofen, J. Rogers, and K. Vijay-Shanker. A first-order axiomatization of the theory of finite trees. *J. Logic, Language, and Information*, 4:5–39, 1995.
2. J. Barwise and R. Cooper. Generalized quantifiers and natural language. *Linguistics and Philosophy*, 4:159–219, 1981.
3. M. Bodirsky, M. Egg, R. Fuchss, A. Koller, J. Niehren, S. Pado, K. Striegnitz, and S. Thater. The CHORUS demo system. `www.coli.uni-sb.de/cl/projects/chorus/demo.html`, 1999.
4. J. Bos. Predicate logic unplugged. In *10th Amsterdam Colloquium*, pages 133–143, 1996.
5. H. Comon. Completion of rewrite systems with membership constraints. In *Coll. on Automata, Languages and Programming*, volume 623 of *LNCS*, 1992.
6. D. Duchier and J. Niehren. Dominance constraints with set operators. Submitted. Available at `www.ps.uni-sb.de/Papers/abstracts/dombool.html`, 2000.

7. M. Egg, A. Koller, and J. Niehren. The constraint language for lambda structures. Technical report, Universität des Saarlandes, Programming Systems Lab, 2000. Submitted. Available at `http://www.ps.uni-sb.de/Papers/abstracts/clls2000.html`.
8. M. Egg, J. Niehren, P. Ruhrberg, and F. Xu. Constraints over Lambda-Structures in Semantic Underspecification. In *Proc. COLING/ACL'98*, Montreal, 1998.
9. K. Erk and J. Niehren. Parallelism constraints. In *International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science, Norwich, U.K., 2000. Springer-Verlag, Berlin.
10. C. Gardent and B. Webber. Describing discourse semantics. In *Proc. 4th TAG+ Workshop*, Philadelphia, 1998. University of Pennsylvania.
11. P. Hirschbühler. VP deletion and across the board quantifier scope. In J. Pustejovsky and P. Sells, editors, *NELS 12*, Univ. of Massachusetts, 1982.
12. A. Koller. Evaluating context unification for semantic underspecification. In *Proc. Third ESSLLI Student Session*, pages 188–199, 1998.
13. A. Koller and J. Niehren. Scope underspecification and processing. Lecture Notes, ESSLLI '99, Utrecht, 1999. `www.coli.uni-sb.de/~koller/papers/esslli99.html`.
14. A. Koller, J. Niehren, and K. Striegnitz. Relaxing underspecified semantic representations for reinterpretation. In *Proc. Sixth Meeting on Mathematics of Language*, 1999.
15. A. Koller, J. Niehren, and R. Treinen. Dominance constraints: Algorithms and complexity. In *Proc. Third Conf. on Logical Aspects of Computational Linguistics*, Grenoble, 1998.
16. J. Lévy. Linear second order unification. In *7th Int. Conference on Rewriting Techniques and Applications*, volume 1103 of *LNCS*, pages 332–346, 1996.
17. G. Makanin. The problem of solvability of equations in a free semigroup. *Soviet Akad. Nauk SSSR*, 223(2), 1977.
18. M. P. Marcus, D. Hindle, and M. M. Fleck. D-theory: Talking about talking about trees. In *Proc. 21st ACL*, pages 129–136, 1983.
19. R. Montague. The proper treatment of quantification in ordinary English. In R. Thomason, editor, *Formal Philosophy. Selected Papers of Richard Montague*. Yale University Press, New Haven, 1974.
20. Mozart Consortium. The Mozart Programming System web pages. `www.mozart-oz.org/`., 1999.
21. R. Muskens. Order-Independence and Underspecification. In *Ellipsis, Underspecification, Events and More in Dynamic Semantics*. DYANA Deliverable R.2.2.C, 1995.
22. J. Niehren and A. Koller. Dominance Constraints in Context Unification. In *Proc. Third Conf. on Logical Aspects of Computational Linguistics*, Grenoble, 1998. To appear in LNCS.
23. J. Niehren, M. Pinkal, and P. Ruhrberg. A uniform approach to underspecification and parallelism. In *Proc. ACL'97*, pages 410–417, Madrid, 1997.
24. J. Pafel. Skopus und logische Struktur. Studien zum Quantorenskopus im Deutschen. Arbeitspapiere des SFB 340 Nr. 129, Univ. Stuttgart/Univ. Tübingen/IBM Deutschland, Oktober 1998.
25. M. Pinkal. Radical underspecification. In *Proc. 10th Amsterdam Colloquium*, pages 587–606, 1996.
26. C. Pollard and I. Sag. *Head-driven Phrase Structure Grammar*. CSLI and University of Chicago Press, 1994.

27. U. Reyle. Dealing with ambiguities by underspecification: construction, representation, and deduction. *Journal of Semantics*, 10:123–179, 1993.
28. Decidability of context unification. The RTA list of open problems, number 90, `www.lri.fr/~rtaloop/`, 1998.
29. M. Schmidt-Schauß. Unification of stratified second-order terms. Technical Report 12/94, J. W. Goethe Universität, Frankfurt, 1994.
30. M. Schmidt-Schauß and K. Schulz. On the exponent of periodicity of minimal solutions of context equations. In *International Conf. on Rewriting Techniques and Applications*, volume 1379 of *LNCS*, 1998.
31. G. Smolka. The definition of kernel Oz. Dfki oz documentation series, German Research Center for Artificial Intelligence (DFKI), Saarbrücken, 1994.
32. K. Vijay-Shanker. Using descriptions of trees in a tree adjoining grammar. *Computational Linguistics*, 18:481–518, 1992.