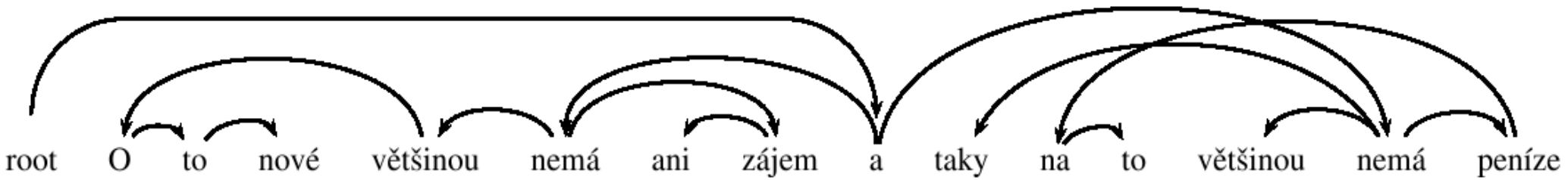
Non-Projective Dependency Parsing

- Jeśli jest „non-projective”, pozwala wygodnie parsować języki z wolnym szykiem zdań
- „Drzewa zależności” (dependency trees) często odzwierciedlają semantykę



He is mostly not even interested in the new things and in most cases, he has no money for it either.

...and beyond

Non-Projectivity

- Parsowanie jest trudne: jeśli nie wprowadzamy ograniczenia „projectivity”, odpadają parsery gramatyk z produkcjami (chart-parsers).
- „A dependency graph is projective, if the yields of its nodes are intervals.”
- „yeld” = zbiór wierzchołków poddrzewa
- „Projective grammars” są równoważne CFG z wyróżnionymi termianalami „głowami”

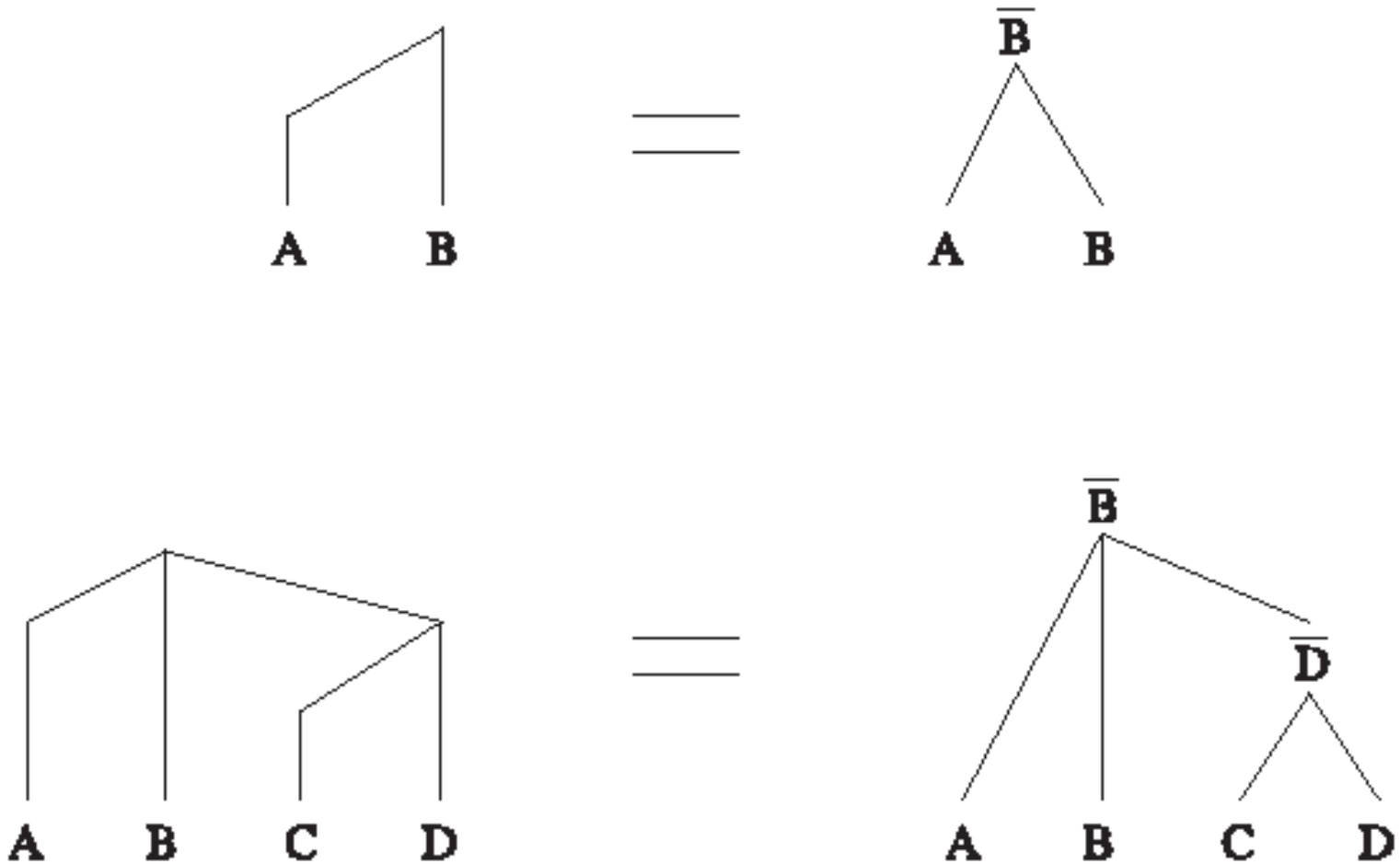
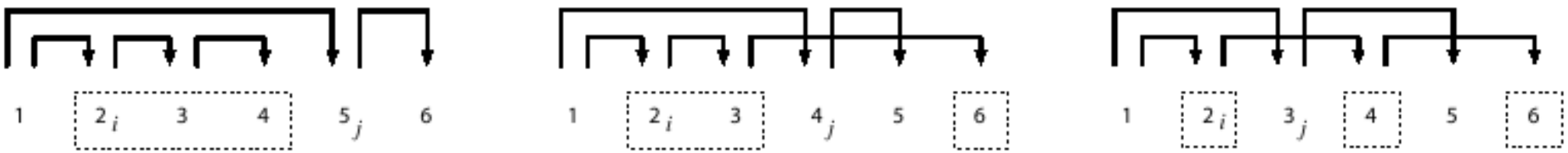


Figure 4: Equivalent dependency and constituency trees.

Mild Non-Projectivity

- Charakter i ilość „non-projectivity” można stopniować
- Nas będą jednak interesować drzewa bez ograniczeń



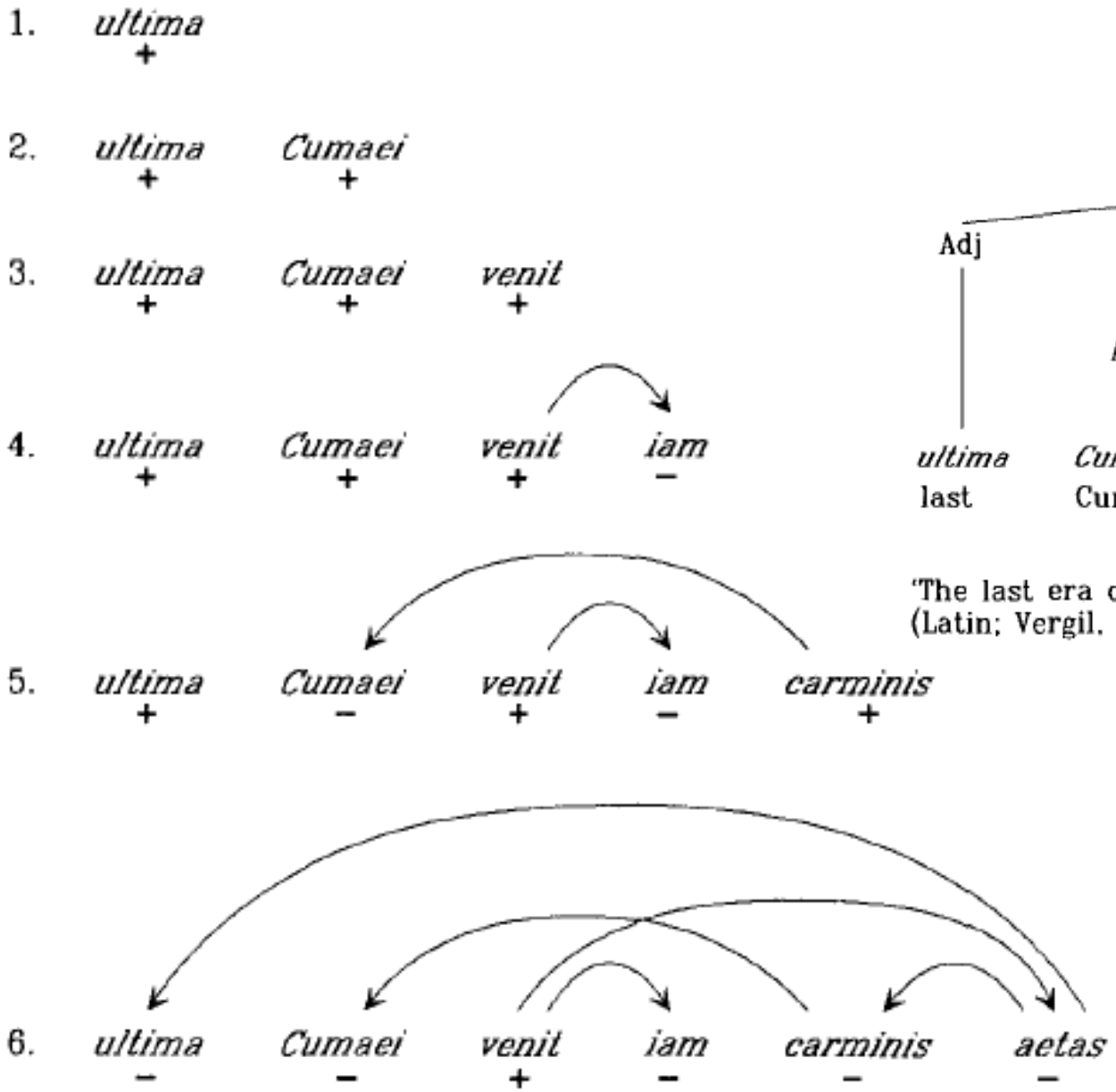
(a) $gd = 0$, $ed = 0$, $wn+$ (b) $gd = 1$, $ed = 1$, $wn+$ (c) $gd = 2$, $ed = 1$, $wn-$

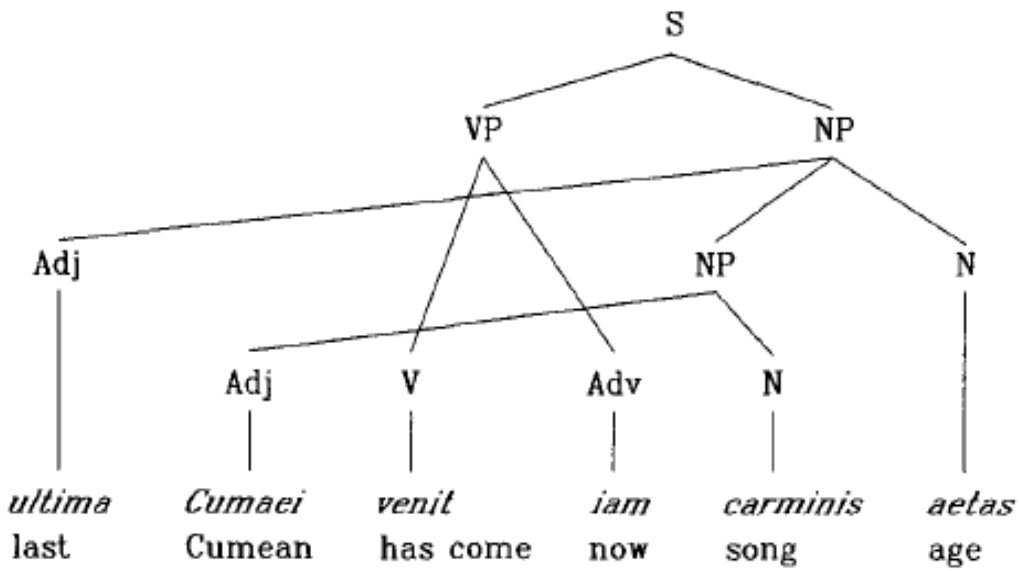
Figure 3: Gap degree, edge degree, and well-nestedness

Backtracing Parser

- Dodając kolejne słowo:
  - przeszukaj listę „niezależnych” słów w dotychczasowym wejściu, jeśli któreś unifikuje z pewną pozycją argumentową nowego słowa, podepnij je i uaktualnij więz unifikacyjny
  - przeszukaj listę „otwartych pozycji argumentowych” w dotychczasowym wejściu, jeśli któraś unifikuje z typem nowego słowa, podepnij słowo w tą pozycję
- Każde podpięcie jest punktem nawrotu.
- Nawracamy, jeśli po przeczytaniu całego zdania nie mamy drzewa rozbioru.

Backtracing Parser (Covington)





'The last era of the Cumean song has now arrived'
(Latin; Vergil, Eclogues IV.4)

$$\begin{bmatrix} \textit{category:noun} \\ \textit{person:X} \\ \textit{number:Y} \\ \textit{case:nominative} \end{bmatrix} \leftarrow \begin{bmatrix} \textit{category:verb} \\ \textit{person:X} \\ \textit{number:Y} \end{bmatrix}$$

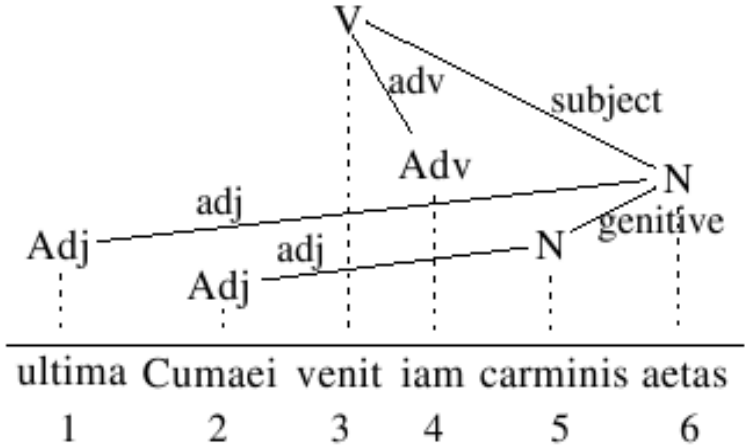
Backtracing Parser (Covington)

Covington, (a za nim niektórzy inni autorzy), uważa, że złożoność algorytmu z nawrotami jest O(n^3), ale wydaje się jasne, że jest wykładnicza. (Prawdopodobnie Covington robi założenie o lokalności, o czym dalej.) Niektórzy autorzy wiążą problem pełnego „non-projective dependency parsing” z LCFRS, które są NP-zupełne.

„Set-Constraint Dependency Parser”

- System Duchiera: w pełni „non-projective”
- słowa mają pięć głównych atrybutów:
  - *string:* fonetyka
  - *cat:* kategoria
  - *agr:* wektor cech dla związku zgody
  - *comps:* argumenty obowiązkowe
  - *mods:* argumenty opcjonalne
- argumenty obowiązkowe występują dokładnie raz, opcjonalne dowolną ilość razy

$$\begin{bmatrix} \text{string} & \text{loves} \\ \text{cat} & \text{V} \\ \text{agr} & \langle \texttt{sing 3 nom} \rangle \\ \text{comps} & \left\{ \texttt{subject, object} \right\} \end{bmatrix}$$



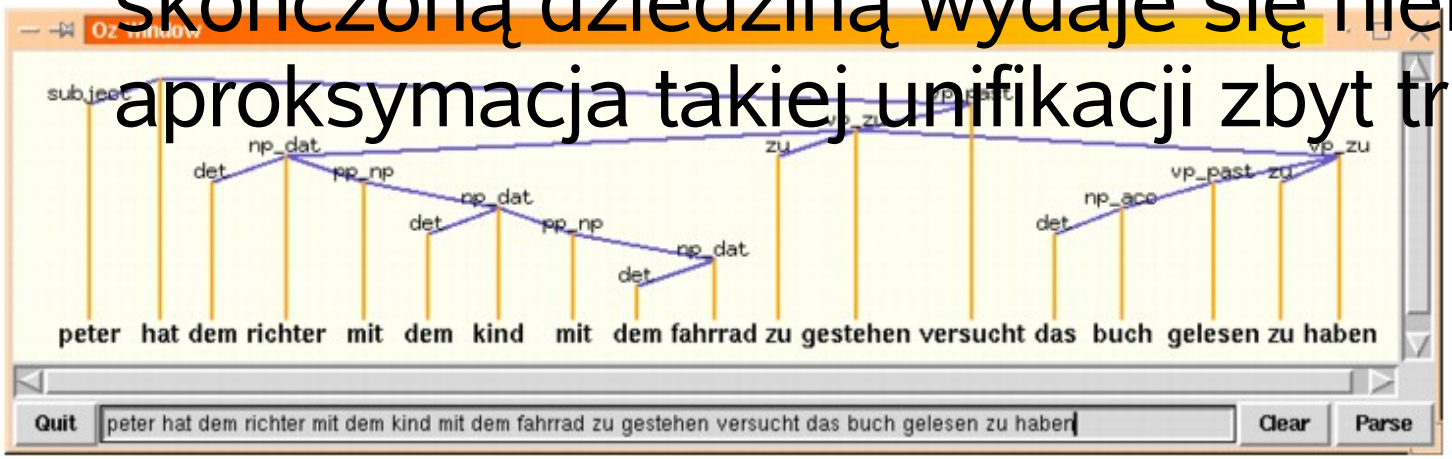
$$1\begin{bmatrix}\text{string} & \text{ultima}\\ \text{cat} & \text{Adj}\\ \text{agr} & \langle \texttt{fem sing 3 nom}\rangle\\ \text{comps} & \{\}\\ \text{mods} & \{\}\end{bmatrix}\quad 2\begin{bmatrix}\text{string} & \text{Cumaei}\\ \text{cat} & \text{Adj}\\ \text{agr} & \langle \texttt{fem sing 3 gen}\rangle\\ \text{comps} & \{\}\\ \text{mods} & \{\}\end{bmatrix}$$

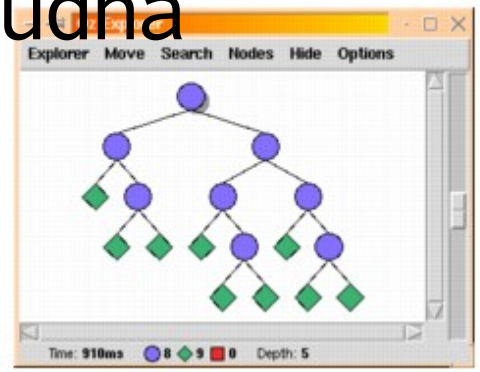
$$3\begin{bmatrix}\text{string} & \text{venit}\\ \text{cat} & \text{V}\\ \text{agr} & \langle \texttt{fem sing 3 nom}\rangle\\ \text{comps} & \{\texttt{subject}\}\\ \text{mods} & \{\texttt{adv}\}\end{bmatrix}\quad 4\begin{bmatrix}\text{string} & \text{iam}\\ \text{cat} & \text{Adv}\\ \text{agr} & \langle\rangle\\ \text{comps} & \{\}\\ \text{mods} & \{\}\end{bmatrix}$$

$$5\begin{bmatrix}\text{string} & \text{carminis}\\ \text{cat} & \text{N}\\ \text{agr} & \langle \texttt{fem sing 3 gen}\rangle\\ \text{comps} & \{\}\\ \text{mods} & \{\texttt{adj}\}\end{bmatrix}\quad 6\begin{bmatrix}\text{string} & \text{aetas}\\ \text{cat} & \text{N}\\ \text{agr} & \langle \texttt{fem sing 3 nom}\rangle\\ \text{comps} & \{\}\\ \text{mods} & \{\texttt{genitive,adj}\}\end{bmatrix}$$

„Set-Constraint Dependency Parser”

- Działa pięknie!
- zbiory reprezentują przetwarzanie wielu drzew-kandydatów jednocześnie
- ale trudno modyfikować „linguistic framework”
- zaprogramowanie unifikacji dowolnych struktur atrybutowych jako więzów na zbiorach nad skończoną dziedziną wydaje się niemożliwe, a aproksymacja takiej unifikacji zbyt trudna





Maximum Spanning Tree

Chu-Liu-Edmonds(G, s)

Graph $G = (V, E)$

Edge weight function $s : E \to \mathbb{R}$

1. Let $M = \{(x^*, x) : x \in V, x^* = \arg\max_{x'} s(x', x)\}$
2. Let $G_M = (V, M)$
3. If G_M has no cycles, then it is an MST: return G_M
4. Otherwise, find a cycle C in G_M
5. Let $G_C = \text{contract}(G, C, s)$
6. Let $\boldsymbol{y} = \text{Chu-Liu-Edmonds}(G_C, s)$
7. Find a vertex $x \in C$ s. t. $(x', x) \in \boldsymbol{y}, (x'', x) \in C$
8. return $\boldsymbol{y} \cup C - \{(x'', x)\}$

contract$(G = (V, E), C, s)$

1. Let G_C be the subgraph of G excluding nodes in C
2. Add a node c to G_C representing cycle C
3. For $x \in V - C : \exists_{x' \in C} (x', x) \in E$
   Add edge (c, x) to G_C with
   $s(c, x) = \max_{x' \in C} s(x', x)$
4. For $x \in V - C : \exists_{x' \in C} (x, x') \in E$
   Add edge (x, c) to G_C with
   $s(x, c) = \max_{x' \in C} [s(x, x') - s(a(x'), x') + s(C)]$
   where $a(v)$ is the predecessor of v in C
   and $s(C) = \sum_{v \in C} s(a(v), v)$
5. return G_C

Maximum Spanning Tree Parsing

- Algorytm Chu i Liu (1965) Edmons (1967) naiwnie zaimplementowany działa w czasie O(n^3), a zaimplementowany przez Tarjana (1977) w czasie O(n^2) dla grafów gęstych.
- s(i, j) jest liczbą oznaczającą „jakość” podporządkowywania j przez i, więc algorytm działa dla gramatyk probabilistycznych które można „podszlifować” na korpusie.
- Niestety...

MST i lokalność zależności

- Algorytm jest poprawny (znajduje MST) tylko wtedy, gdy wartość s(i, j) jest lokalna (jest funkcją tylko (i, j), nie zależy od reszty zbudowanego drzewa).
- Ale to wyklucza nie tylko gramatyki unifikacyjne, ale nawet normalny wymóg, że (niektóre) role mają być pełnione przez dokładnie jeden argument!
- Nadzieja: MST z dobrze wyszkoloną gramatyką daje 84% skuteczności na czeskim „tree-banku”

Tree Correcting Parsing

- Chcemy gramatykę unifikacyjną, więc:
  - Uruchamiamy MST z $s(i,k, j) = u(i,k, j) + h(i,k, j)$
  - $u(i,k, j) = 0$ jeśli typ słowa j unifikuje z typem k-tego argumentu słowa i, w przec. wyp. $u(i,k, j) = -\infty$
  - $h(i, j)$ jest heurystyką uwzględniającą odległość słowa i-tego od j-tego, czy są oddzielone interpunkcją, wymagania pozycyjne słowa i dla argumentu k, być może wyniki analizy statystycznej
  - zapamiętujemy j w zbiorze dla k-tego argumentu i, dla wygrywającej pary (i, k)

Tree Correcting Parsing

- generujemy „świeże zmienne” dla typów w wystąpieniach słów w (zbiorach w) węzłach drzewa zwróconego przez MST
- dla argumentów które mają wielu kandydatów, wybieramy tego kandydata którego typ unifikuje z typem argumentu przy uwzględnieniu zgromadzonego dotąd więzu unifikacyjnego, uaktualniamy więz unifikacyjny
- pozostałe drzewa dopasowujemy (uaktualniając więz unifikacyjny) w wolne miejsca, w kolejności zgodnej z h(i,k, j)

Tree Correcting Parsing

Prosty algorytm dla „Tree Correcting Parsing” warto rozszerzyć o „backtracing” lub iteracyjne poprawianie (ale jak?)

Logika liniowa a dependency parsing

- Zazwyczaj nie wiąże się metod logiki liniowej z pojęciem „dependency parsing”, ale dają one wszystkie możliwości nieunifikacyjnego „non-projective dependency parsing” plus...
- pozwalają wymuszać obecność dziur określonego typu w podporządkowywanych poddrzewach.
- Np. leksem typu (P -> Q) -> R buduje węzeł R podporządkowujący poddrzewo o korzeniu Q, w którym brakuje wierzchołka P

Logika liniowa a CG

- Logika liniowa i logika Lambeka są logikami *podstrukturalnymi*: wnioskowanie zależy od struktury, w jaką są ułożone przesłanki.
- W przypadku logiki Lambeka modus ponens obejmuje tylko przyległe przesłanki w ciągu, usuwając je z dalszego wnioskowania
- W logice liniowej ograniczenie jest osłabione: przesłanki stanowią multizbiór, można użyć dowolnej przesłanki ale tylko raz
- nie ograniczamy się do rachunku sekwentów, ale konstruujemy term dowodu (izomorfizm C-H)

Modyfikatory

W „dependency grammars” zazwyczaj uważa się, że frazy modyfikujące są podporządkowane modyfikowanym (jako argumenty nieobligatoryjne; w CG jest na odwrót). Fraza modyfikująca to fraza typu T -> T.

Przypadek „bez dziur”

fraza podporządkowywana

fraza podporządkowująca

$$(206) \qquad \frac{\psi : A : S_1 \qquad \delta : A \rightarrow B : S_2}{\delta(\psi) : B : S_1 \cup S_2}\,\mathrm{E} \qquad \text{provided } S_1 \cap S_2 = \emptyset$$

typ frazy

zbiór indeksów słów we frazie

Przechowujemy dwie tablice: „atomową” (dla wyrażeń f : A : S, gdzie A jest atomem) indeksowaną atomami A, i „nie-atomową” (dla wyrażeń f : A->B : S, gdzie A jest atomem) indeksowaną argumentami A, oraz stos wyrażeń do przetworzenia, zainicjowany przez w : T : {i}.
Przetwarzamy wyrażenie K ze stosu: jeśli jest atomowe, odkładamy na stos jego kombinacje z wyrazami odpowiedniej pozycji tablicy „nie-atomowej”, w p.p. z tablicy „atomowej”. Na koniec wkładamy K do odpowiedniej tablicy i przetwarzamy kolejny element ze stosu.

Przypadek ogólny „wyższego rzędu”

$$((((A^+ \to B^-)^- \to C^+)^+ \to D^-)^- \to E^+)^+$$

- algorytm generuje postać „eta-rozwiniętą”
- dodajemy zmienne dla wszystkich pozytywnych podtermów typów
- pamiętamy, które zmienne trzeba gdzie zużyć

$m : (D \to C \to B) \to A$	$m : B_{[2,3]} \to A : \{1\}$ $v_2 : D : \{2\}$ $v_3 : C : \{3\}$
$m : (D \to E) \to (C \to B) \to A$	$m : E_{[2]} \to B_{[3]} \to A : \{1\}$ $v_2 : D : \{2\}$ $v_3 : C : \{3\}$

(212) $$\frac{\psi : A : S_1 \qquad \delta : A_L \to B : S_2}{\delta(\lambda v_{i_1}, \ldots, \lambda v_{i_n}.\psi) : B : S_1 \cup S_2}\,\mathrm{E}$$ provided $S_1 \cap S_2 = \emptyset$ and $L \subseteq S_1$ and $L = [i_1, \ldots, i_n]$

Zastosowania: scope ambiguities

$$\dfrac{\dfrac{see : \mathbf{2}^e \to \mathbf{3}^e \to \mathbf{1}^t \qquad some(woman) : (\mathbf{3}^e \to \mathbf{1}^t) \to \mathbf{1}^t}{\lambda x.some(woman, \lambda y.see(x,y)) : \mathbf{2}^e \to \mathbf{1}^t}\,\text{COMP} \qquad every(man)}{every(man, \lambda x.some(woman, \lambda y.see(x,y))) : \mathbf{1}^t}\,\text{APP}$$

$$\dfrac{\dfrac{\dfrac{see : \mathbf{2}^e \to \mathbf{3}^e \to \mathbf{1}^t}{\lambda y \lambda x.see(x,y) : \mathbf{3}^e \to \mathbf{2}^e \to \mathbf{1}^t}\,\text{EXCH} \qquad every(man) : (\mathbf{2}^e \to \mathbf{1}^t) \to \mathbf{1}^t}{\lambda y.every(man, \lambda x.see(x,y)) : \mathbf{3}^e \to \mathbf{1}^t}\,\text{COMP} \qquad some(woman)}{some(woman, \lambda y.every(man, \lambda x.see(x,y))) : \mathbf{1}^t}\,\text{APP}$$

Figure 3.4: Two derivations for “Every man saw some woman.”

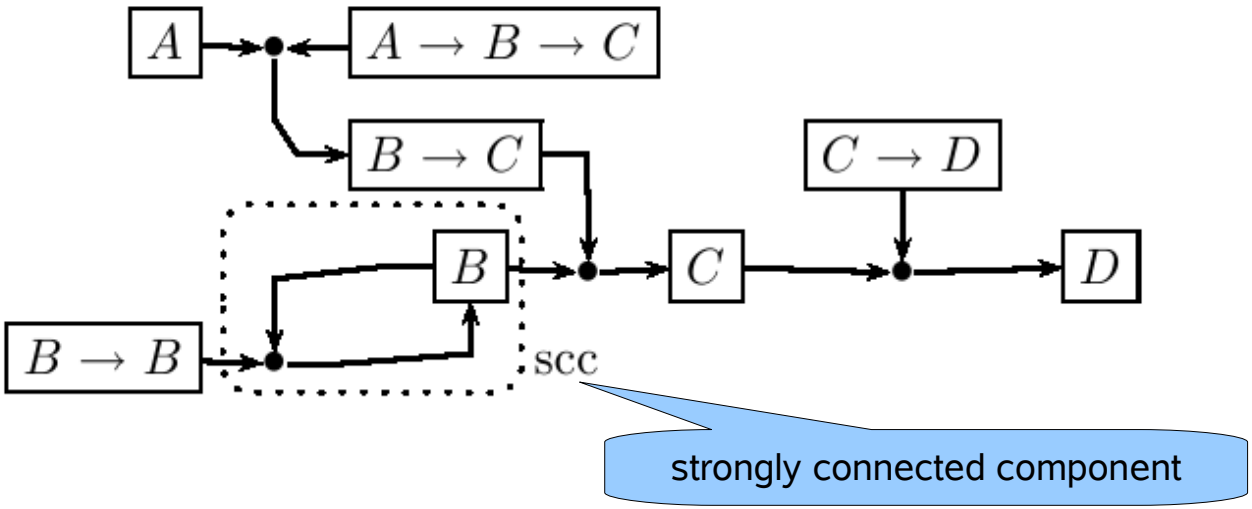
Logiki liniowe pozwalają naturalnie wyrażać niejednoznaczności semantyczne.

Optymalizacja konstrukcji dowodu

- spakowanie razem wyrażeń mających ten sam span i typ
- graf kategorii pozwala uniknąć „ślepych uliczek” w parsowaniu oraz maksymalnie spakować wyrażenia

(234) Premises: A, B, $B \to B$, $A \to B \to C$, $C \to D$

Scalamy silnie spójne podgrafy i sortujemy topologicznie uzyskany DAG; w algorytmie parsującym przechodzimy do wierzchołków „na prawo” tylko jeśli wykorzystano już indeksy wszystkich wierzchołków „na lewo”.
(To tylko, błędny jeszcze, szkic wstępny: szczegóły w pracy Iddo Leva.)



Labelled Deduction Systems

- Ogólne LDSy pozwalają na „proceduralną” manipulację inferencją
- Współczesnym rozwinięciem koncepcji LDSów z mechanizmami proceduralnymi (ale też z tree adjoining, unifikacją, ...) jest *Dynamic Syntax*
- LDSy pozwalają na jednolitą implementację systemów parsująco - planujących / wnioskujących, jak SHRDLU czy inteligentne interfejsy baz danych

John likes Mary, whom he admires.

For $l_1 : f_1, \ldots, l_n : f_n$ labels l_i and formulas $f_i, 1 \leq i \leq n$ projected from words $w_i, \ldots, w_n$:

prove $\{ l_1 : f_1, \ldots, l_j : f_j \} \vdash \alpha : t$

s_1 :

1.	$John : e$	use_me_last
2.	$like : e \rightarrow e \rightarrow t$	
3.	CHOOSE $s_1 = t_i, t_i \subset t_{now}$	
4.	$Mary : e$	use_me_first
12.	$like(Mary) : e \rightarrow t$	
13.	$like(John, Mary) : t$	

exit $\alpha = like(John, Mary)$

LINK $v = Mary$

prove $\{ l_k : f_k, \ldots, l_n : f_n \} \vdash \beta[v/\text{Mary}] : t$

s_2 :

5.	$\mathbf{u}_{he} : e$	$\Theta\mathbf{u}_{he} \notin s_2$, male($\Theta\mathbf{u}_{he}$), use_me_last
6.	CHOOSE $\mathbf{u}_{he} = John$	
7.	$admire : e \rightarrow e \rightarrow t$	
8.	CHOOSE $s_2 = t_i, t_i \subset t_{now}$	
9.	$v : e$	use_me_first
10.	$admire(v) : e \rightarrow t$	
11.	$admire(John, v) : t$	

exit $\beta[v/Mary] = admire(John, Mary)$

Figure 1: *Utterance Interpretation in the L&R model*

Na zachętę: abdukcja

CUSTOMER:	*I want to hire a car.*
SALESMAN:	*When for?*
CUSTOMER:	*Today.*
SALESMAN:	*Would you like to hire this Sierra?*
CUSTOMER:	*I don't drive big cars.*
SALESMAN:	*The small cars are reserved for members only.*
CUSTOMER:	*Can I join?*
SALESMAN:	*It will take a week to process the papers.*
	Would you like to reconsider?
CUSTOMER:	*Big cars are not easy for me to drive. I can't control them.*
SALESMAN:	*But the Sierra has power-assisted steering!*
CUSTOMER:	*So?*
SALESMAN:	*Power-assisted steering makes a car easy to control.*

Źródła

- „Non-projective Dependency Parsing using Spanning Tree Algorithms”, Ryan McDonald, Fernando Pereira, Kiril Ribarov, Jan Hajic 2005
- „Mildly Non-Projective Dependency Structures”, Joakim Nivre, Marco Kuhlmann 2006
- „A Fundamental Algorithm for Dependency Parsing”, Michael A. Covington 2000

- „Mildly Non-Projective Dependency Structures”, Joakim Nivre, Marco Kuhlmann 2006

- „An Introduction to Dependency Grammar”, Ralph Debusmann

Źródła

- „Parsing Discontinuous Constituents in Dependency Grammar.” Technical Correspondence to CL, Michael A. Covington 1990
- „Axiomatizing dependency parsing using set constraints”, Denys Duchier 1999
- „Packed Computation of Exact Meaning Representations”, Iddo Lev 2007
- „Labelled Abduction and Relevance Reasoning”, Dov Gabbay, Ruth Kempson, Jeremy Pitt 1994