社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report
2005－NL－167 (9)
2005－SLP－56 (9)
2005／5／27

統語森に対応する圧縮共有型依存構造「依存森」について

平川 秀樹
(株) 東芝 研究開発センター 知識メディアラボラトリ
email: hideki.hirakawa@toshiba.co.jp

選好依存文法 (PDG) は，自然言語の形態素，構文，意味解析を統合的に行う枠組みであり，各レベルの種々の曖昧性を統合的に効率良く保持し，各レベルの知識により優先度を設定し，全体解釈として最適な解を計算する．本稿では，選好依存文法で用いられるヘッド付き統語森，依存森といった言語解釈を統合保持するデータ構造とその構築手法について述べ，文の句構造を圧縮共有する統語森と依存構造を圧縮共有する依存森との対応関係において完全性と健全性が成立することを示す．

Dependency Forest: Packed Shared Dependency Structure Corresponding to Parse Forest

Hideki Hirakawa
Knowledge Media Laboratory
TOSHIBA Corporate Research & Development Center
1, Komukai-Toshiba-cho, Saiwaiku, Kawasaki, 212-8582, Japan

Preference Dependency Grammar(PDG) is a framework for the morphological, syntactic and semantic analysis for natural language sentences. PDG gives packed shared data structures to hold the various ambiguities in each levels of sentence analysis with preferences scores and a method for calculating the most plausible interpretation for a sentence. This paper proposes the packed shared data structures, such as the Head-added Parse Forest, the Dependency Forest adopted in PDG, and shows the completeness and the soundness of the mapping between the Parse Forest and the Dependency Forest.

1 はじめに

自然言語文解析における最大の課題は，形態素，構文，意味，文脈などの各種レベルにおける曖昧性の中から，いかにして正しい解釈を認識するかという点にある．形態素，構文等の各レベルにおいて組合せ的な数の解釈が存在するのみならず，一般に，各種レベルの知識は「ある解釈の優先性 (preference) に関する場合が多い」「異なったレベルにおける知識の干渉がある」などの性質を持っており，組合せ爆発とともにそれを適切に扱うことが重要である．このためには，

(a) 各レベルの解釈を効率良く保持 [共有構造の課題]
(b) 各種知識による優先度を設定 [優先度設定の課題]
(c) 最適な解釈を探索 [最適解導出の課題]

という3つの処理が必要であると考えている[1]．

文献2), 3) では，「意味係り受けグラフ」を提案し，係り受けの多義（構文的多義）と係り受け意味関係の多義を効率良く保持する手法を提案した．この解析手法では，英語等に比べて品詞多義が殆どないこと，「係り受け文法」という修飾語が被修飾語の左に位置する体系であることなど，日本語固有の特徴に依存した制約が入っていた．現在，こうした制約を取り除いた汎用の新しい文解析の枠組みとして，形態素，構文，意味の3レベルに渡り (a)〜(c) の処理を行なう枠組みである選好依存文法 (PDG: Preference Dependency Grammar) を構築中である．このうち，本稿では (a) の共有構造の課題に対するアプローチについて述べる．以下，共有データ構造の要件，従来技術とその課題について述べ，PDG の共有データ構造であるヘッド付き統語森 (HPF:Head-added Parse Forest)，依存森 (DF:Dependency Forest) について説明する．

2 共有データ構造の要件と従来技術

2.1 共有データ構造の要件

自然言語解析の解釈を保持する共有データ構造には次の性質が必要であると考えられる．

(a) 共有データ構造レベルで組合せ爆発が起こらない
(b) 各種レベルの曖昧性を過不足なく表現できる
(c) 各種レベルの知識の総合評価のベースとなりうる

(a) は，実システムを構築する際に特に重要な課題である．一般に解釈の組合せ展開を行うと直ぐに扱いが困難になり，また，計算時間的にも不十分となる．(b) は，多レベル (PDG では，形態素，構文，意味の3レベル) の知識を扱う場合に，各レベルの曖昧性を全て過不足なく表現できること，すなわち共有構造そのものに由来する解釈の枝刈り (あるべき解釈の欠落) や解釈の過生成 (あるべきでない解釈の生成) が起こらないという性質である．この性質を保持した上でシステム構築上有効な枝刈りを導入できることは重要な好ましい性質である．(c) は，具体的には，それぞれのレベルでの知識の記述が行いやすいこと，それぞれのレベルでの制約的適用と優先的適用が融合できること[3] などがあげられる．

2.2 句構造と依存構造の併用

文解析を精度良く行なうためには様々な知識を利用する必要がある．従来，文の構造を記述する代表的枠組みとして句構造と依存構造がある．句構造は，品詞への抽象化により語や句の順序に関する知識の記述に優れており，依存構造は語の間の種々の依存関係に関する知識の記述に優れている．それぞれの表現レベルでの制約知識・選好知識の記述を自然な形で可能とするため，PDG では，句構造形式の共有データ構造 (ヘッド付き統語森) と依存構造形式の共有データ構造 (依存森) をそれぞれ関連付けて組み込んでいる．これは句構造 (C-構造) と機能構造 (F-構

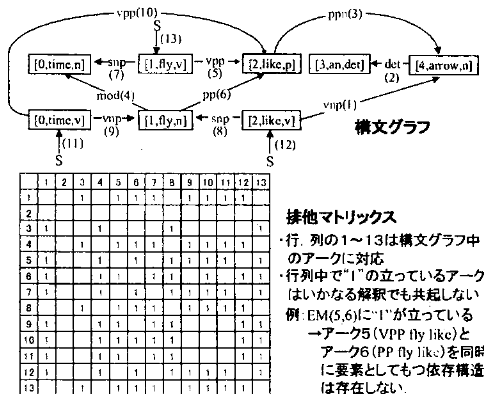


図 1: 例文に対する構文グラフと排他マトリックス

造) という2つの構文レベルの表現を持つ LFG[4] において, SUBJECT,OBJECT など構文的機能に関する制約が F-構造で記述され, 文法の記述性を高めているのと類似している[*2]. なお, Early 法, Chart 法といった文脈自由文法の解析アルゴリズムを用いて依存文法を直接解析して依存構造を求める手法も提案されている[5] が, 句構造を作らない点で本手法とは異なっている.

2.3 従来技術とその問題点

文脈自由文法で入力文を解析し文の可能な解釈全体を得る手法は広く知られており, 例えば, 富田により, グラフスタックを用いた構文解析手法と共に文の句構造解釈 (構文木) 全体を効率的に保持する圧縮共有統語森 (Packed Shared Parse Forest) (以下, 統語森と記述) が提案されている[6].

Seo は, 文の句構造解釈全体と対応する依存構造全体を効率良く保持する方法として構文グラフ (Syntactic Graph) を提案した[7),8]. 構文グラフは, PDG の共有データ構造として有望であったが, 大きな問題もありそのまま採用することはできないことが判明した. 以下, 構文グラフとその問題について説明する.

2.3.1 構文グラフ/排他マトリックス

構文グラフは, 単語間の依存関係をベースに文の可能な解釈を圧縮共有する枠組みである. 本稿では, 単語と品詞の組を語品詞 (WPP:Word Pos Pair) と呼ぶ[*3]が, 構文グラフは, 語品詞に対応するノードとノード間の依存関係を表現する名前付きアークで構成される有向グラフであり, 排他マトリックス (EM:Exclusion Matrix) と呼ばれるデータと組になって, 入力文に含まれる依存構造 (DS:Dependency Structure) の集合 (文の解釈の集合) を表現する. 構文グラフは, 3次組 (Triple) と呼ばれる, アーク名とその両端のノード (語品詞, 表層位置などを持つ) の組からなる集合で表現される. 図 1 は, "Time flies like an arrow" に対する構文グラフ/排他マトリックスである. アークの括弧中の番号はその ID である. 1つのノードに入る複数のアークは修飾の曖昧性を表現している. S は, 開始記号に相当する.[*4]

排他マトリックスは, 構文グラフを構成するアークを行・列とし, アーク間の共起制約を記述するために導入されている. 排他マトリックスの i 行 j 列が 1 である場合には, i 番目と j 番目のアークは, いかなる解釈 (依存木) においても共起しない.

構文グラフ/排他マトリックスは, 統語森に基づいたデータ構造から生成される. PDG でも同じデータ構造を用いており, これをヘッド付き統語森と呼ぶ. ヘッド付き統語森の詳細は, 3 章で述べる.

2.3.2 構文グラフ/排他マトリックスの問題点

文献 7) では, 統語森と構文グラフ/排他マトリックス間の完全性 (completeness), 健全性 (soundness) について言及している. 完全性は, 「圧縮共有統語森中の1つの構文木が存在した時に, 構文グラフ/排他マトリックス中にそれに対応する依存木が存在する」という性質であり, 健全性は, 「構文グラフ/排他マトリックス 中の1つの依存木が存在した時に, 統語森中にそれに対応する構文木が存在する」という性質である. 構文グラフ/排他マトリックスの完全性が成立することは示されているが, 健全性については保証されていない. 排他マトリックスは, 3次組の間の共起関係を規定しているが, ある1つの構文木に対応する依存木が存在した場合, その依存木に含まれる3次組の間の共起排他制約を排他マトリックスから除外するという方法で構成される. 排他マトリックスは, 全ての依存構造の解釈を規定するため, この除外された共起排他制約が他の全ての解釈 (依存木) において制約として必要でない場合にのみ健全性が保証されることになる. 付録1に構文グラフで健全性が破綻する例を示し, 本稿で提案する方式との差を示す.

3 PDG における共有データ構造

PDG では, 前節で述べた従来手法の問題を解決するデータ保持方式として, 文脈自由文法の構文構造の保存方式としてヘッド付き統語森を採用し, 依存構造の保存方式として依存森 (DF:Dependency Forest) を提案する.

3.1 ヘッド付き統語森

ヘッド付き統語森は統語森の一種であり, 適用された書き換え規則に対応する弧 (edge) から構成され, 次の条件を満足する構文木を圧縮共有する.

(a) 句の非終端記号 (カテゴリ) が同じ
(b) 句の被覆する単語範囲が同じ
(c) 句ヘッドとなる主構成素 (語品詞) が同じ

(a),(b) の2つが統語森の共有条件である[9]. ヘッド付き統語森中の構文木は, 統語森中の構文木と対応が取れる. PDG における弧とヘッド付き統語森の具体例は構築アルゴリズムと共に 4 章で述べる.

3.2 依存森

依存森は, 依存グラフ (DG:Dependency Graph) と共起マトリックス (CM:Co-occurrence Matrix) より成る. 以下, 依存森の構造と依存木について説明する.

3.2.1 依存グラフと共起マトリックス

図 2 は, "Time flies like an arrow" に対する依存森の例である. 依存グラフは, 語品詞に対応するノードと1つのルートノード root ならびに, ノード間の

[*2] LFG では, 1つの文解釈である C-構造とそれから作られる F-構造の制約関係などを規定しているが, 文に対する可能な C-構造全体と F-構造全体の扱いについては, 特に規定していない点が PDG との基本的違いである.

[*3] 単語 time は"time /v","time/n" 等の語品詞を持つ.

[*4] 本稿では, アークの方向は, 係り受け解析に従って記述する, 構文グラフとは向きが逆であるが, 本質的な差はない.

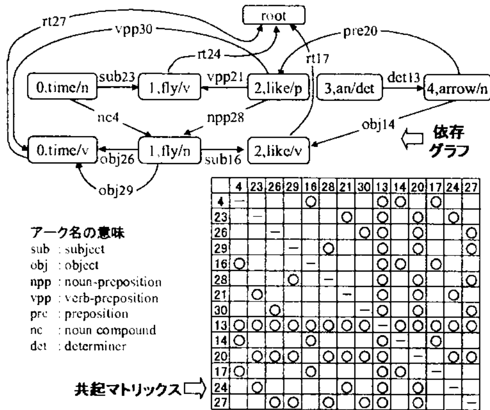


図 2: "Time flies like an arrow" に対する初期依存森

依存関係を表現する名前付きアークより構成される．依存グラフは，root をルートノードとする有効グラフであり，実際には，アークとその両端のノードの組からなる依存片 (dependency piece)*5の集合として表現する．アークは，アーク名と ID を有す．アークの元側を依存ノード (dependent あるいは modifier)，先側を支配ノード (governor あるいは modificand) と呼ぶ．また，ノードには，表層位置などの情報も含まれている．アークの数を依存森のサイズと呼ぶ．また，依存グラフの部分集合で木構造を成す依存片集合が依存木であり，文や句の解釈を表現する．依存グラフは複数の依存木を含む共有データ構造となっている．

共起マトリックスは，アーク ID で示されたアーク集合を行と列に取り，アーク間の共起関係を規定する．共起マトリックス $CM(i,j)$ が○の場合に限り，アーク i と j は 1 つの依存木（解釈）において共起可能であるという制約を表現する．共起関係は双方向関係であり，CM は対称行列となる．

3.2.2 整依存木

依存森中の依存木のうち，次の条件 (a)〜(c) 全体を整依存木条件と呼び，これら全てを満たす依存木を整依存木 (Well-formed Dependency Tree) と呼ぶ．

[整依存木条件]

(a) 表層位置が同じノードは存在しない (語品詞単一解釈条件)
(b) 入力文の単語と，入力文に対応する依存木のノードの間に 1 対 1 対応が取れる (被覆条件)
(c) 共起マトリックスにおいて共起関係が成立する (整共起条件)

(a)(b) の２つを纏めて整被覆条件と呼び，これを満たす木を整被覆木 (Well-covered Dependency Tree) と呼ぶ．また，(c) を満たす依存木を整共起依存木 (Well-cooccuered Dependency Tree) と呼ぶ．整依存木の集合が入力文に対する解釈の集合となる．図 2 の依存森では，「時は矢のように過ぎる」，「時ハエは矢を好む」，「矢のようにハエを計れ」，「矢のようなハエを計れ」に対応する 4 つの整依存木が存在する．

*5 依存片とアークは 1 対 1 対応するので，特に区別の必要がない場合は依存片をアークとも記述する．

3.2.3 初期依存森と縮退依存森

複数の解釈に対するアークの共有の度合いによって，同じ依存木の集合を表すサイズの異なった複数の依存森を構成可能である．詳細は後述するが，PDG では，初期依存森 (Initial Dependency Forest) と，それから変換して得られる縮退依存森 (Reduced Dependency Forest) の 2 種を扱う．それぞれ，初期依存グラフと初期共起マトリックス，ならびに，縮退依存グラフと縮退共起マトリックスよりなる．単に依存森と呼んだ場合は，通常後者を示す．図 2 の初期依存グラフは，図 1 の構文グラフと比較すると，"fly/n" と "time/v" の間のアーク数が異なっている．対応する縮退依存森については後述する．

4 ヘッド付き統語森と依存森の生成

PDG では，形態素解析，構文解析，ヘッド付き統語森・初期依存グラフの生成，縮退依存森導出の順で解析が進む．本稿では，形態素解析処理については省略し，構文解析以降について述べる．

4.1 文法規則

PDG において文法規則は，可能な句構造の定義と，句構造から依存構造へのマッピングとを規定する拡張文脈自由文法 (extended CFG) で記述される．

文法規則は，次の形式をしている．

$$y/Y \rightarrow x_1/X_1, .., x_n/X_n \ \ [arc(arcname_1, X_i, X_j), \ldots arc(arcname_{n-1}, X_k, X_l)] \ (n>0, 0<i,j,k,l\leq n)$$

[例] $vp/V \rightarrow v/V, np/NP, pp/PP$: $[arc(obj, NP, V), arc(vpp, PP, V)]$

規則は，":" で区切られた書き換え規則部と構造構築部よりなる．書き換え規則の左辺の "y/Y" 及び構成素 (constituent) "x_i/X_i" は，"構文カテゴリ/構造変数" を表す．Y は句ヘッド (head of phrase) と呼ばれ，主構成素 (head constituent) に相当し，規則ボディ (rule body) である "$X_1..X_n$" のいずれかと同一となる．構造構築部は，"arc(アーク名，構造変数 1，構造変数 2)" という形式の依存片の集合であり，構造変数には，書き換え規則部の構成要素の句ヘッドとなる語品詞が束縛される．例は，V をヘッドとし，obj アークで NP が，vpp アークで PP が接続する依存構造を示している．規則中の部分依存構造は，次の部分依存構造条件を満足する整部分依存構造である．

(a) 主構成素の句ヘッド Y をルートとする木構造である．(非主構成素は，他の構成素と依存関係 (アーク) を持つ)
(b) 規則ボディの構成素の句ヘッドは，部分依存構造をなす木構造の構造変数と 1 対 1 対応が取れる

例文 "Time flies like an arrow" を解析するための文

```
### Grammar Rules
s/VP   → np/NP, vp/VP          : [arc(sub, NP, VP)]
s/VP   → vp/VP                 : []
np/N   → det/DET, n/N          : [arc(det, DET, N)]
np/N   → n/N                   : []
np/N2  → n/N1, n/N2            : [arc(nc, N1, N2)]
np/NP  → np/NP, pp/PP          : [arc(npp, PP, NP)]
vp/V   → v/V                   : []
vp/V   → v/V, pp/PP            : [arc(vpp, PP, V)]
vp/V   → v/V, np/NP            : [arc(obj, NP, V)]
vp/V   → v/V, np/NP, pp/PP     : [arc(obj, NP, V), arc(vpp, PP, V)]
pp/P   → pre/P, np/NP          : [arc(pre, NP, P)]
### Lexicon
word(n, [time]). word(v, [time]). word(n, [flies]). word(v, [flies]).
word(pre, [like]). word(v, [like]). word(det, [an]). word(n, [arrow]).
```

図 3: 例文を解析する文法規則と辞書

弧の構造:8 次組 <ID,FP,TP,C,PH,FCSL,RCS,DSL>で表現

①ID:弧 ID, ②FP(From Position):弧の始点, ③TP(To Position):弧の終点, ④C(Category):規則ヘッド頂点(対応するCFG規則のヘッドカテゴリ), ⑤PH(Phrase Head):句ヘッド(主構成素となるノード), ⑥FCSL(Found Constituent Sequence List):既存構成素列のリスト(CFG規則の構成素(弧の ID で表現)のリストのリスト), ⑦RCS(Remaining Constituent Sequence):残り構成素列(弧の終端に必要な構成素列), ⑧DSL(Data Structure List):依存構造リスト(部分依存構造のリスト)

部分依存構造: $[ARC_1, ..., ARC_n]$ (アーク(依存片)リスト($n \geq 0$))

アークの構造 :arc(REL-AID, DEP-DP, GOV-GP)

①REL(Relation):関係名, ②AID(Arc ID):アーク(依存片)の ID, ③DEP(Dependant):依存ノード, ④DP(Dependant Position) : DEP の開始位置, ⑤GOV(Governor):支配ノード, ⑥GP(Governor Position):GOV の開始位置

弧の例:

規則: np/N → det/DET, n/N : [arc(det,DET,N)]

弧1:<45, 3, 3, np, $2, [], [det/$1, n/$2], [[arc(det, $1, $2)]]>

弧2:<70, 3, 4, np, $2, [[63]], [n/$2],
[[arc(det, [an]-det-3, $2)]]>

弧3:<84, 3, 5, np, [arrow]-n-4, [[63, 75]], [],
[[arc(det-2, [an]-det-3, [arrow]-n-4)]]>

規則: vp/V→v/V, np/NP, pp/PP :[arc(obj, NP, V), arc(vpp, PP, V)]
vp/V→v/V, np/NP :[arc(obj, NP, V)]

弧4:<210, 0, 5, vp, [time]-v-0, [[110, 125, 203], [110, 212]], [],
[[arc(obj-26, [flies]-n-1, [time]-v-0),
arc(vpp-30, [like]-pre-2, [time]-v-0)],
[arc(obj-29, [flies]-n-1, [time]-v-0)]]

辞書: word(n, [arrow])

弧5:<181, 4, 5, n, [arrow]-n-4, [[lex([arrow]-n)]], [], [[]]>

図 4: 弧の構成と例

法規則と辞書を図 3 に示す.

4.2 構文解析

PDG の構文解析は, 文脈自由文法の解析アルゴリズムに,「句ヘッドが同じ」という共有条件の追加と, 文法規則中の依存構造の構築処理の追加により基本的に実現できる. 本稿では, Bottom-up Chart Parsing のアルゴリズムをベースに実現している. 図 4 に弧の構成と例を, 図 5 にアルゴリズムを示す.

4.2.1 弧の構成と例

図 4 に示すように, 弧は 8 次組で構成されており, 通常の Chart Parsing の弧に比べて, 句ヘッド (PH) と依存構造リスト (DSL) が追加されている. また, 既存構成素列にあたる部分は, 複数の既存構成素列を保持できるリスト (FCSL) となっており, 構造が共有される. FCSL と DSL は同じ長さのリストであり, それぞれの要素が対応関係を持つ. この要素の組を構成素列・部分依存構造ペア (constituent sequence/depenedenty structure pair) と呼び, CSDS ペアと略記する. 図 4 の弧 1 〜 3 は, 名詞句規則に対応する弧が解析が進むにつれて生成されてゆく例である. 弧 3 は, "an arrow" を np として解釈し, 部分依存構造として"arc(det-2,[an]-det-3,[arrow]-n-4)" を持つ不活性弧 (RCS が"[]") である. 弧 4 は, 複数の解釈を纏めた弧の例である. FCSL の 2 要素と DSL の 2 要素がそれぞれ対応し CSDS ペアを構成している. CSDS ペアは, ([110,125,203],[obj-26,vpp-30]),([110,212],[obj-29]) のように記述する. また, 弧 5 のような, 辞書引きにより生成される不活性弧は語彙弧 (lexical edge) と呼ぶ.

4.2.2 構文解析アルゴリズム

図 5 の基本構成は, Agenda を用いた一般的な Chart Parsing アルゴリズム[10] であり, 先頭から順次入力単語を語彙弧化して Agenda に追加処理する処理 ((a),(b)) と Agenda が空になるまで Agenda 中の不活性弧に対して文法規則ならびに Chart 中の活性弧から可能な弧を生成・展開する処理 ((e),(f)) より成る. Agenda 中の弧は, Chart 中の弧と共有可能かどうかが判定され ((c),(j)), 共有可能な場合はマージされ ((d)), 圧縮共有が行なわれる. 基本的に一般的なアルゴリズムであり, 詳細な説明は省略するが, 次に PDG 特有の依存構造の構築の部分について説明する.

本アルゴリズムは, 弧を生成しながら弧の部分依存構造を構築する. これは句ヘッド (ノード) と依存構造中の構造変数を束縛することで実現される. この変数束縛は, 不活性弧と文法規則から新しい弧が生成される時点 ((g)), ならびに, 不活性弧と Chart 中の活性弧により新しい弧が生成される時点 ((h)) に, bind_var により不活性弧の句ヘッドが他方の弧の残り構成素列の先頭の構造変数に束縛されることで行なわれる. さらに, この変数束縛により依存ノードと支配ノードが確定されたアークに対して, add_arcid によりユニークなアーク ID が付与される ((i)). 図 4 の弧 2 の変数$2 にノード"[arrow]-n-4" が束縛されると弧 3 になる. 本アルゴリズムでは, 弧もアークもユニークな ID が付与されるため, アークに対して, それを生成した弧, CSDS ペア (構成素列, 部分依存構造) は 1 つだけ対応することが保証される.

それぞれの弧は弧 ID で関連付けられており, 弧からその下位の弧 (構成要素に対応) を順次辿れる. 図 4 の弧 3 では, 弧#84(弧 ID が 84 の弧) は, "$np \rightarrow det, n$" の規則から生成された弧であり, 既存構成素列 [63,75] は, 弧#63 が文法規則中の構成素 det に, 弧#75 が構成素 n に対応することを示している. また, 弧 4 のように複数の既存構成素列を持つ弧は, merge_csds((d)) により生成される.

PDG では, ルートノードを 1 つにするために規則ヘッドを $root$ とする "$root \rightarrow S$"(S はスタートシンボル) に対応する特殊な規則を導入している[*6]. 構文解析に成功した場合には, Chart にはこの規則に対応する, 句ヘッドが $root$ で文全体を被覆する不活性弧が 1 つ存在する. これをルート弧 (root edge) と呼ぶ.

4.3 ヘッド付き統語森・初期依存森の生成

構文解析後の Chart は, 活性弧, 不活性弧より成る. この集合に対して, 不活性弧 e から辿れる弧の集合を $hpf(e)$ と記述する. ルート弧を e_r とした時, ヘッド付き統語森は $hpf(e_r)$ となる. ルート弧から到達できない不活性弧も存在するため, $hpf(e_r)$ は不活性弧全体の部分集合となる. 初期依存グラフは, ヘッド付き統語森中のアークの集合であり, $hpf(e_r)$ と同時に求められる. また, 初期共起マトリックスも同時に求められる. 図 6 にヘッド付き統語森・初期依存森を求めるアルゴリズム, また, 図 7 に, 図 3 の文法を用いて例文を構文解析した結果得られるヘッド付き統語森を示す. ヘッド付き統語森を構成する全ての弧は不活性弧であるため, 残り構成素列 ([]) は省略している.

図 6 のアルゴリズムは, try_edge,try_FCSL,try_CS の 3 つの関数を再起的に呼びながら, それぞれが支配

[*6] この規則は, PDG の完全性・健全性を保証するだけであれば本質的には不要であるが, 後の処理の都合上導入している.

```
Sent:入力単語リスト, Grammar:文法規則集合
Chart := {}; Agenda := {}; /* 初期化 */
for(Pstn:=0; Pstn < length(Sent); Pstn++) {          /* (a) */
 Agenda := get_lex_edges(Pstn,Sent);  /* 語彙弧生成 (b) */
 while(Agenda != {}){ /* 空になるまで弧の拡張展開を繰返し */
   A_Edge := pop(Agenda); /* Agenda から1つ弧を取出す */
   /* A_Edge とマージ可能な弧を Chart から取り出す */
   C_Edge := mergable_edge(A_Edge);                  /* (c) */
   if( C_Edge != new ) {  /* マージ可能な弧 C_Edge が存在 */
     MrgdEdge = merge_csds(A_Edge,C_Edge);           /* (d) */
     /* マージで変化があれば,C_Edge を MrgdEdge に更新 */
     if(MrgdEdge != C_Edge) { update_chart(MrgdEdge); } }
   else if( C_Edge == new ) {    /* マージ不可→新規の弧 */
     push(A_Edge,Chart);         /* 弧 A_Edge を Chart に登録 */
     /* 文法から生成される新規の弧を Chart/Agenda に登録    */
     add_new_edge_by_searching_grammar(A_Edge);  /* (e) */
  /* 既存の弧から生成される新規の弧を Chart/Agenda に登録  */
     add_new_edge_by_existing_edge(A_Edge); /* (f) */} } }

add_new_edge_by_searching_grammar(Edge) {
 Edge = <ID, FP, TP, C, PH, FCSL, RCS, DSL>;
 foreach Rule in Grammar {   /* 文法中の各規則を順次処理 */
   /* 文法規則の要素の取出し.規則ボディの長さは m(m>0)      */
   Rule = <Y/Yv, [X1/X1v,X2/X2v,...,Xm/Xmv], RDS>;
   if( X1 == C ) { /* カテゴリが一致 */
     /* 変数束縛処理: 変数 X1v を句ヘッド PH に束縛 (g)      */
     RDS := bind_var(X1v,PH,RDS);
     NewYv := bind_var(X1v,PH,Yv);
     NewEdge:=<new_id(),FP,TP,Y,NewYv,[[ID]],
              [X2/X2v,...,Xm/Xmv],[RDS]>;
     if( NewEdge が不活性弧 ) { push(NewEdge,Agenda); }
     else { push(NewEdge,Chart); } } } }

add_new_edge_by_existing_edge(Edge) {
 Edge = <ID, FP, TP, C, PH, FCSL, RCS, DSL>;
 foreach Edge_C in Chart { /* Chart 中の弧を順次処理 */
   Edge_C = <IDc, FPc, TPc, Cc, PHc, FCSLc, RCSc, DSLc>;
   RCSc = [X1/X1v, X2/X2v,...,Xm/Xmv];  /* RCSc 内容(m>0) */
   /* 組合せ拡張の条件: ①Edge_C が活性弧                    */
   /* ②RCSc の最初のカテゴリ X1 が Edge のカテゴリ C と等しい */
   /* ③Edge_C に Edge が隣接(Edge_C の終点==Edge の始点)    */
   if( RCSc != [] && X1 == C && FP == TPc ) {
     /* === Edge_C と Edge から新規の弧 NewEdge を生成 ===  */
     NewFCSL := add_id_to_FCSL(ID,FCSLc);
     NewDSL := bind_var(X1v,PH,DSLc);    /* 変数束縛 (h) */
     NewDSL := add_arcid(NewDSL);     /* アーク ID 付与 (i) */
     NewEdge:=<new_id(),FPc,TP,Cc,PHc,NewFCSL,
              [X2/X2v,...,Xm/Xmv],NewDSL>;
     if( NewEdge が不活性弧 ) { push(NewEdge,Agenda); }
     else { push(NewEdge,Chart); } } } }

mergable_edge(A_Edge) {
 foreach C_Edge in Chart {
  if(A_Edge と C_Edge の"句範囲(FP,TP)","規則ヘッド頂点(C)",
     "句ヘッド(PH)","残り構成素列(RCS)"が等しい)      /* (j) */
    { return(C_Edge); }}
 return(new); }

-------- アルゴリズムの説明で利用する関数・記法 ---------
D = S      : データ構造 D の要素を表現 S で分解して参照.
X ∪ Y      : 集合 X と Y の和集合. length(L): リスト L の長さ.
new_id()   : 呼ばれる毎にユニークな ID(1,2,3..)を返す.
push(E,X)  : 集合 X に要素 E を追加.
pop(X)     : 集合 X から要素を1つ取り去り,その要素を返す.
add_arcid(DSL): DSL 中のアークで依存ノード,支配ノード共に確
  定したアークに対して新 ID を付与. ex. DSL=[[arc(subj,
   [i]-n-0,[love]-v-1),arc(obj,$3,[love]-v-1)]]の時,
   add_arcid(DSL)→[[arc(subj-23,[i]-n-0,[love]-v-1),
                     arc(obj,$3,[love]-v-1)]]
add_id_to_FCSL(ID,FCSL): FCSL 中の各要素に弧 ID を追加した
ものを返す. ex. ID=29, FCSL=[[23,12],[11]] の時,
   add_id_to_FCSL(ID,FCSL) → [[12,23,29],[11,29]]
bind_var(Var,Val,DS): DS 中の変数 Var を値 Val で置換した構造を
   返す. ex. Var=$1,Val=a(x),DS=[b($1),c($2)]の時,
   bind_var(Var,Val,DS) → [b(a(x)),c($2)]
update_chart(E): 弧 E と同じ ID の Chart 中の弧を弧 E に更新
get_lex_edges(FP,S): 単語リスト S の位置 FP から辞書引きし語彙
  弧集合を生成. ex. FP=1,S=[I,love,you]の時,例えば次を返す.
    {<10,2,3,v,[love]-v-2,[[lex([love]-v)]],[],[[]]>,
     <11,2,3,n,[love]-n-2,[[lex([love]-n)]],[],[[]]>}
merge_csds(E1,E2)  : 弧 E1 の CSDS ペアを弧 E2 に追加した弧を返す
 ex.E1=<8,3,5,s,$1,[[17]],[vp/$1],[[arc(obj-3,[i]-n-2,$1)]]>,
    E2=<9,3,5,s,$1,[[13]],[vp/$1],[[arc(subj-5,[i]-n-2,$1)]]>
 の時,E1 の CSDS ペア([17],[arc(obj-3,[i]-n-2,$1)])を E2 に追加
  merge_csds(E1,E2)→<9,3,5,s,$1,[[13],[17]],[vp/$1],
      [[arc(subj-5,[i]-n-2,$1)],[arc(obj-3,[i]-n-2,$1)]]>
 注:E1 中の CSDS ペアが E2 中に存在する場合は,追加されない.
 E2=<7,3,5,s,$1,[[13]],[vp/$1],[[arc(obj-5,[i]-n-2,$1)]]>
 の時,merge_csds(E1,E2)→E2
```

図 5: PDG ボトムアップチャートパージングアルゴリズム

する弧を深さ優先で重複を避けてトラバースする構成となっている ((d),(h),(j)). それぞれの関数の実行後は,その引数 (弧,構成素列リスト,構成素列) に対するヘッド付き統語森 HPF,依存グラフ DG,共起マトリックス CM の要素や値が追加設定されている.

全体に対する処理は,図 6 の (a) の"try_edge(ルート弧)" である.try_edge では,(b) で既に実行済みか否かを判定し,実行済みの場合は,TER に記録済みのアーク集合を取り出して返す.TER への登録を行なうのは (g) である.HPF に弧が追加されるのは,(c) と (e) においてである.(f) にあるように,弧 E が支配するアークは,E の DSL 中のアークと FCSL の支配するアークの和集合である.

try_FCSL は,複数の CSDS ペアを処理し,try_CS は,その中の 1 つの CS を処理する.(i) にあるように,FCSL の支配するアーク集合は,その要素である CS が支配するアークの和集合である.また,(k) にあるように,CS の支配するアーク集合は,その要素である弧が支配するアークの和集合である.

図 7 の弧#210 を例に処理の具体例を次に示す.

```
$try_edge(弧\#210)$                                    (c1)
| $try_FCSL([[110,125,203],[110,212]])$                 (c2)
| | $try_CS([110,125,203],[obj-26,vpp-30])$             (c3)
| | | $try_edge(弧\#110)$->{}                   (c4)->(r4)
| | |          :
| | | $try_edge(弧\#125)$->{}
| | |          :
| | | $try_edge(弧\#203)$->{pre-20,det-13}
| | +-> {pre-20,det-13}                                 (r3)
| | $try_CS([110,212],[obj-29]$                         (c5)
| | | $try_edge(弧\#110)$->{}                   (c6)->(r6)
| | |          :
| | +-> {npp-28,pre-20,det-13}                          (r5)
| +->{pre-20,det-13,obj-26,vpp-30,npp-28,obj-29}  (r2)
+->{pre-20,det-13,obj-26,vpp-30,npp-28,obj-29}    (r1)
```

(c#) は関数の呼び出し,(r#) はその結果 (支配するアーク集合) を示す.(c1)〜(c4) は,図 6 (j),(d),(h) の再起呼び出しである.弧#110 が支配する弧は存在しないため,(c4) は (r4) の" {}" を返す.(c3) の処理が終了し (r3) を得ると,(c2) の 2 番目の CSDS ペア ([110,212],[obj-29]) に対する処理 (c5) が行なわれる.(c6) で再度" try_edge (弧#110)" が実行されるが,この時は,図 6(b) で TER に保存された計算結果を検索して返す.最終的に (r1) が得られる.

ここで,共起マトリックスの生成処理について説明する.共起マトリックスは,1 つの構文木に同時に含

```
/* ヘッド付き統語森 HPF, 依存グラフ DG, C マトリックス CM */
HPF := {}; DG := {}; CM := {}; /* 初期化 */
VE  := {};   /* Visited Edge: 既にトライした弧の ID の集合 */
TER := {};            /* 弧 ID とその支配するアークを記録保持 */
try_edge(ルート弧);   /* ルート弧から HPF, DG, CM を計算 (a) */
exit;  /* HPF, DG, CM が生成されている */

% 弧 E から, HPF, DG, CM を計算, E が支配するアーク集合を返す
try_edge(E) {
 E = <ID, FP, TP, C, PH, FCSL, RCS, DSL>;  /* 弧の要素の取出し */
 if(ID∈VE) { return(get(ID, TER)); }   /*処理済の弧 (b) */
 else if( E が語彙弧 ) {             /* 語彙弧 E を登録 (c) */
   push(E, HPF); put(ID, {}, TER); return({}); }
 A_FCSL :=try_FCSL(FCSL, DSL); /* CSDS ペア集合を処理 (d) */
 push(E, HPF);                       /* 弧 E を HPF へ追加 (e) */
 A_Edge := arcs(DSL) ∪ A_FCSL; /* E の支配するアーク (f) */
 put(ID, A_Edge, TER); /* 弧 E が支配のアーク集合を記録 (g) */
 return(A_Edge); }

% FCSL から HPF, DG, CM を計算, FCSL が支配するアーク集合を返す
try_FCSL(FCSL, DSL) {
 A_FCSL := {}; /* 初期化 */
 foreach (CS, DS) in (FCSL, DSL) { /* CSDS ペア(CS, DS)取出 */
   A_CS := try_CS(CS);        /* 構成素列 CS 以下を処理 (h) */
   set_CM(DS, DS);    /* CM 処理(1): DS 中のアーク間の共起 */
   set_CM(DS, A_CS); /* CM 処理(2): DS と CS 間の共起        */
   A_FCSL := A_FCSL ∪ A_CS; }  /* arcs(FCSL)の計算 (i) */
 return(A_FCSL); }    /* FCSL の支配するアークの集合を返す */

% CS から, HPF, DG, CM を計算, CS が支配するアーク集合を返す
try_CS(CS) {
 A_CS := {}; /* CS の支配するアークの集合の初期化 */
 foreach E in CS {
   A_Edge := try_Edge(E);     /* 弧 E 以下をトラバース (j) */
   set_CM(A_Edge, A_CS); /* CM 処理(3): CS の要素間の共起 */
   A_CS := A_CS ∪ A_Edge; }        /* arcs(CS)の計算 (k) */
 return(A_CS); }         /* CS の支配するアークの集合を返す */

--- アルゴリズムの説明で利用する関数・記法 ---
put(I, E, X): 集合 X に要素 E をインデックス I で追加
get(I, X)   : 集合 X にあるインデックス I の要素を返す
set_CM(A1, A2) : アークリストまたは集合 A1, A2 の要素間の CM の
  値を○にセット. ex. A1={a1, a2}, A2={a3, a4}, a1～a4 のアー
  ク ID が id1～id4 の時, set_CM(A1, A2)は, CM の(id1, id3), (id1,
  id4), (id2, id3), (id2, id4)を○にセット(=CMに追加)
arcs(X)     : データ構造 X に含まれるアークの集合を返す.
```

図 6: ヘッド付き統語森・初期依存森を求めるアルゴリズム

まれるアークの間に共起可能性を設定するよう，次の条件を満たす時に設定される.

(CM1) 1 つの部分依存構造 DS 中のアークは共起する

(CM2) CSDS ペア (CS,DS) において，CS が支配するアークは，DS 中のアークと共起する

(CM3) 1 つの構成素列 CS が支配するアーク間には共起関係がある

これはそれぞれ図 6 の CM 処理 (1)〜(3) に対応している．弧#210 の例では，try_FCSL の処理で CSDS ペア ([110,125,203],[obj-26,vpp-30]) に対して CM 処理 (1) で (CM1) すなわち set_CM([obj-26,vpp-30],[obj-26,vpp-30]) が実施される．また，CM 処理 (2) では，A_CS は図 8(r3) となり，(CM2) すなわち set_CM([obj-26,vpp-30],[pre-20,det-13]) が実施される．また，try_CS([110,125,203]) の処理において，CM 処理 (3) により (CM3) すなわち弧#110,#125,#203 が支配するアーク間の共起関係の CM へのセットが試みられる.

例題に対するアルゴリズムの出力は，図 7 のヘッド付き統語森ならびに図 2 の初期依存森となる．弧

```
Time flies like an  arrow
0    1     2    3   4      5
<200, 0, 5, root, [root]-x-root, [[199], [206], [211]],
         [[arc(root-17, [like]-v-2, [root]-x-root)],
          [arc(root-24, [flies]-v-1, [root]-x-root)],
          [arc(root-27, [time]-v-0, [root]-x-root)]]>
<199, 0, 5, s,   [like]-v-2,    [[136, 196]],
         [[arc(sub-16, [flies]-n-1, [like]-v-2)]]>
<136, 0, 2, np,  [flies]-n-1,   [[101, 123]],
         [[arc(nc-4, [time]-n-0, [flies]-n-1)]]>
<101, 0, 1, n,   [time]-n-0,    [[lex([time]-n)]],  [[]]>
<123, 1, 2, n,   [flies]-n-1,   [[lex([flies]-n)]], [[]]>
<196, 2, 5, vp,  [like]-v-2,    [[159, 190]],
         [[arc(obj-14, [arrow]-n-4, [like]-v-2)]]>
<159, 2, 3, v,   [like]-v-2,    [[lex([like]-v)]],  [[]]>
<190, 3, 5, np,  [arrow]-n-4,   [[178, 181]],
         [[arc(det-13, [an]-det-3, [arrow]-n-4)]]>
<178, 3, 4, det, [an]-det-3,    [[lex([an]-det)]],  [[]]>
<181, 4, 5, n,   [arrow]-n-4,   [[lex([arrow]-n)]], [[]]>
<206, 0, 5, s,   [flies]-v-1,   [[103, 204]],
         [[arc(sub-23, [time]-n-0, [flies]-v-1)]]>
<103, 0, 1, np,  [time]-n-0,    [[101]],            [[]]>
<204, 1, 5, vp,  [flies]-v-1,   [[141, 203]],
         [[arc(vpp-21, [like]-pre-2, [flies]-v-1)]]>
<141, 1, 2, v,   [flies]-v-1,   [[lex([flies]-v)]], [[]]>
<203, 2, 5, pp,  [like]-pre-2, [[156, 190]],
         [[arc(pre-20, [arrow]-n-4, [like]-pre-2)]]>
<156, 2, 3, pre, [like]-pre-2, [[lex([like]-pre)]], [[]]>
<211, 0, 5, s,   [time]-v-0,    [[210]],            [[]]>
<210, 0, 5, vp,  [time]-v-0,    [[110, 125, 203], [110, 212]],
         [[arc(obj-26, [flies]-n-1, [time]-v-0),
            arc(vpp-30, [like]-pre-2, [time]-v-0)],
          [arc(obj-29, [flies]-n-1, [time]-v-0)]]>
         [[arc(npp-28, [like]-pre-2, [flies]-n-1)]]>
<110, 0, 1, v,   [time]-v-0,    [[lex([time]-v)]],  [[]]>
<125, 1, 2, np,  [flies]-n-1,   [[123]],            [[]]>
<212, 1, 5, np,  [flies]-n-1,   [[125, 203]],
         [[arc(npp-28, [like]-pre-2, [flies]-n-1)]]>
```

図 7: 例文に対するヘッド付き統語森

#211,#206,#199 は，同じ非終端記号 s と句の範囲 (0 から 5) を持つが，句ヘッドとなるノードが異なるため共有されておらず，統語森とは異なっている.

4.4 縮退依存森の生成

初期依存森には，図 2 における obj26 と obj29 のようにアーク ID 以外は同一のアークが存在することがあり，これを同値アーク (equivalent arc) と呼ぶ．縮退依存森は，初期依存森を縮退することで得られる．依存森の縮退とは，依存森に内在する依存木の増減・変更なしに複数の同値アークを 1 つにマージする操作であり，結果として依存森のサイズは小さくなる.

4.4.1 同値アークのマージ操作と縮退条件

初期依存グラフの全同値アークを縮退させると構文グラフと同じ構造になり，健全性を保てなくなる．以下，健全性を保つための縮退条件を示す.

依存グラフ DG 中の同値アーク X, Y ($same(X,Y)$ と記述する) に対するマージ操作を次のように定義する.

(1) 依存グラフ DG より，Y を削除する

(2) アーク I ($I \in DG, I \neq X, I \neq Y, CM(Y.I) = ○$) に対して，$set_CM(X, I)$ を行なう

図 8 にマージの例を共起マトリックスの形式で模式的に示す．マージの前後で X のアーク ID は変化しないが，CM が変化する．以下では，マージ後のアーク X を $mrg(X, Y)$ と記述してこの差を表現する.

新たな整依存木 (解) の増加は，同値アークのマー

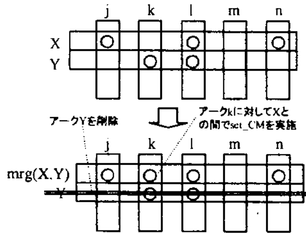


図 8: 同値アークペア (X,Y) のマージ操作の例

ジにより新たにアーク間に共起関係が許されることに起因する．マージにより $CM(U,V)$ が○に変化することをアークペア (U,V) の許諾と呼び，次が成立する．

「アークペア (U,V) の許諾により新たな整依存木が増加する場合，その依存木は U,V を要素として含んでいる.」 $-$ (A)

これは，もし U,V の許諾により U と V の両方を同時に含まない新しい依存木が生成されると仮定するとその依存木は許諾の前でも存在してしまうことになることから明らかである．ここで，$same(X,Y)$ に対して $uniq,diff$ を次のように定義する．

$$uniq(X,Y) = \{I|CM(X,I) = \bigcirc, CM(Y,I)\neq\bigcirc, I\in DG\}$$

$$diff(X,Y) = \{(I,J)|I\in uniq(X,Y), J\in uniq(Y,X)\}$$

$diff(X,Y)$ の要素であるアーク $A,B(A\neq B)$ が X と Y のマージにより新たに共起関係が追加される可能性のあるアークペアとなる．上記の例では，$uniq(X,Y) = \{j,n\}, uniq(Y,X) = \{k\}$, $diff(X,Y) = \{(j,k),(n,k)\}$ となる．(A) より，「$same(X,Y),(A,B)\in diff(X,Y)$ に対して (A,B) を許諾しても解釈が増えない」(縮退条件と呼ぶ) ことが保証されていれば X と Y のマージによる縮退が可能となる．できるだけ効率的に縮退条件を判定するため，アークペア (A,B) を 3 つの場合に分けて縮退条件を詳細化する．

(a) A,B の共起関係が既存
(b) A,B の依存ノードの始点が同一
(c) A,B が同値アークを介して共起
(d) (a),(b),(c) 以外

(a) は $CM(A,B)$ が○で既に共起している場合であり縮退条件を満たす．(b) の場合 ($same_position(A,B)$ と記述する) も整依存木の整被覆条件により A,B は常に排他関係となり縮退条件を満たす．(c) は，$CM(A,X)$ が○,$CM(B,X)$ が " " として，「$same(B,C)$ かつ $CM(C,X) = \bigcirc$ なるアーク C が存在する」または「$same(A,C)$ かつ $CM(C,Y) = \bigcirc$ なるアーク C が存在する」場合であり，$cooc_same(A,B)$ と記述する．この場合も整依存木の条件により同値アークは排他関係となり，同値アークのどちらが選ばれても依存木は同じであり縮退条件を満たす．(d) に相当するアークペアを「不確定アークペア」と呼び，以下，不確定アークペアの縮退条件を詳細化する．

図 9(a) に示すように，同値アーク $same(X,Y)$ と不確定アークペア $(A_0,B_0)\in diff(X,Y)$, $CM(A_0,B_0)\neq\bigcirc$ を考える．図 9(b) に X,Y をマージした場合を示す．このマージで $mrg(X,Y)$ は，A_0,B_0 と共に共起可能となるが，$CM(A_0,B_0)\neq\bigcirc$ のため，このマージを実施しても，A_0,B_0 を共に含む解は生成されない．しかし，別のマージ処理により，A_0,B_0 を共起可能とする事態 (例えば,$same(A_0,A_1)$ のマージにより $CM(A_0,B_0)$ が○になる) が起こると，A_0,B_0 を要素として含む新たな解が増える可能性が生じる．このように，X,Y のマージ直後では解の増加は生じないが，それ以降のマージにより解の増加を起こす可能性のあるアークペアが生じる．この状態にあるアークペアを $CM(A_0,B_0) = ☆$ で表現し，この設定操作を星マーキング，(A_0,B_0) を星アークペアと呼ぶ．$same(A_0,A_1)$ のマージのように，○を☆に重ね合わせるようなマージを危険なマージと呼ぶ．また，星アークペアでない不確定アークペアは縮退条件を満足する．以下，星アークペアの許諾時の縮退条件について説明する．

今，アーク集合 S から得られる整依存木の集合を $dt(S)$、アーク U ならびに U と共起するアーク全体の集合を $co(U)$、アーク U を要素として持つ整依存木の集合を $dt_with_arcs(U)$ と記述する．依存木 $t\in dt_with_arcs(U)$ は，$co(U)$ の要素から構成される．さらに，U と V を共に含む整依存木の集合 $dt_with_arcs(U,V)$ 中の依存木は，$co(U)\cap co(V)$ の要素から構成される．

図 9(b) に示すような危険なマージ (A_0 と A_1 のマージ) において，星アークペア (A_0,B_0) に関する縮退条件は，前述の (A) より，A_0,B_0 を含む木がマージ後に存在しないことである．この条件は，一般にアーク L,M を含む解は $dt(co(L)\cap co(M)))$ に含まれるので，図 9(c) に示すように，A_0,A_1 のマージ前の星アークペアを含む木の集合 $dt(co(A_0)\cap co(B_0))) + dt(co(A_1)\cap co(B_0)))$ とマージ後の星アークペアを含む木の集合 $dt(co(mrg(A_0,A_1))\cap co(B_0)))$ が同一であることである[*7]．今，第一項の $dt(co(A_0)\cap co(B_0)))$ は，共起条件より空であるため，第二項と左辺が同じであれば良い．これを星アークペア (A_0,B_0) の許諾

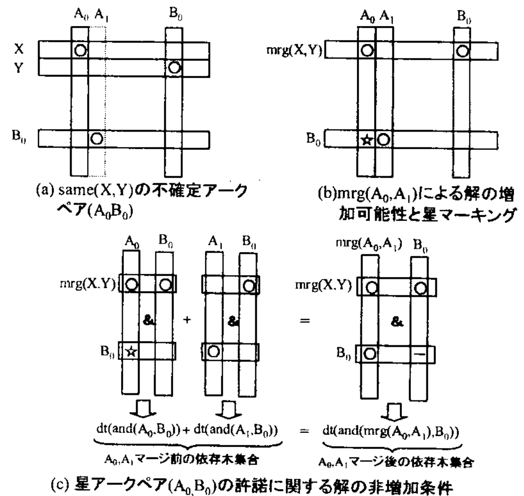


図 9: 同値アークの縮退条件の説明図

*7 依存木の同一性判定時には同値アークは同一アークとして扱う

条件と呼ぶ．全ての星アークペアに関して許諾条件を満たす同値アークはマージ可能である．

4.4.2 依存森の縮退アルゴリズム

図 10 に依存森の縮退アルゴリズムを示す．依存グラフ中にある同値アークペアを順次取り出し，*mergeable* でマージ可能性をチェックし，マージ可能なペアである場合には，星アークペア情報 (☆) の設定を行った後，マージ処理を実行し，共起マトリックスと依存グラフの縮退を行なう．マージ可能性を判定する *mergeable* では，☆を○に重ねる際に星アークペアの許諾条件を判定し，許諾不可の場合はマージしない．*co,mrg* など各種の集合演算は AND/OR などの論理演算で実現できる．縮退条件 $dt(co(A_1)\cap co(B_0))$ == $dt(co(mrg(A_0,A_1))\cap co(B_0)))$ は，関数 $same_{dts}$ で判定する．この関数の詳細は省略するが，$co(mrg(A_0,A_1))\cap co(B_0)$ に対して被覆条件を満足するよう全解探索を実施しながら，同時に $co(A_1)\cap co(B_0)$ に同等の木が存在しないかをチェックすることで実現している．与えられたアーク集合に対する全解探索は文献[3] の暫定解を求めるアルゴリズムと同様の方法で容易に実現可能である．

"Tokyo taxi driver call center" に対するアルゴリズムの動作例を付録 1 の後半に示す．また，"Time flies like an arrow." の例文では，図 2 の依存森において，同値アーク 26,29 がマージ可能であり，29 を 26 にマージした縮退共起マトリックスとなる．縮退依存グラフは，初期依存グラフからアーク 29 を削除したグラフとなる．この例の場合，依存森と構文グラフ/排他マトリックスは同等となる．

マージ処理により生成される依存森は，マージを試みる同値アークの順番により異なる場合があり一意に決るという訳ではない．例えば，付録の例では，複数の縮退依存森が存在する．最小の依存森の構成法などについては今後の課題とする．

```
/***** 依存森の縮退アルゴリズム *****/
DG,CM: 縮退対象の依存森（依存グラフ，共起マトリックス）
while(未処理の同値アーク X,Y ペアを 1 つ取り出す) {
  /* X,Y のマージ可能性チェック */
  if(mergeable(X,Y)==false) { next; }
  /* X,Y のマージにより生じる不確定アークペアを処理 */
  foreach (A,B) in diff(X,Y) {
    if(CM(A,B)==○ ||              /* 既に共起関係が存在 */
       same_position(A,B) ||          /* 同位置にある */
       cooc_same(A,B))        /* 同値アークを介して共起 */
      { next; }
    else /* 星アークペアとして設定 */
      { CM(A,B)="☆"; } }
  /* X,Y のマージ実行（依存森の縮退） */
  mrg(X,Y) を生成し Y を削除する. }

/***** same(X,Y) のマージ可能性をテスト *****/
mergeable(X,Y) {
 foreach B in DG {
  if(CM(X,B)==○ && CM(Y,B)==☆) { CRCL=X; STR=Y; }
  elseif(CM(X,B)==☆ && CM(Y,B)==○) { CRCL=Y; STR=X; }
  else { next; }
  /* 解の増大チェック */
  MRG = mrg(CRCL,STR);
  /* MRG と B を要素として含む整依存木集合 DT1 */
  DT1=dt(co(MRG)∩co(B))∩dt_with_arcs(MRG,B);
  /* STR のみにあるアークを含むかのチェック */
  NEW=arcs(DT1)∩uniq(STR,CRCL);
  if(NEW={}) { next; } /* B に対する解の増加なし */
  return(false); } /* マージ不可 */
 return(true); } /* マージ可能 */
```

図 10: 依存森の縮退アルゴリズム

4.5 依存森の完全性と健全性

依存森は，ヘッド付き統語森との間で完全性と健全性が成立する．付録 2 に初期依存森の完全性と健全性の証明を示す．統語森とヘッド付き統語森は対応関係があり，縮退依存森は初期依存森と同じ依存木の集合を保持しているため，統語森と依存森に完全性と健全性が成立すると言える．

5 おわりに

本論文では，PDG における圧縮共有データ構造について述べた．圧縮共有された句構造解釈（統語森）を圧縮共有された依存構造（依存森）に対応関係を持って変換でき，それぞれのレベルでの言語知識の適応が可能である点が最大の特徴である．今後は，依存森計算のより効率的な処理方法等について検討を進めると共に優先度設定の課題，最適解導出の課題に対する PDG のアプローチについて報告してゆく予定である．

参考文献

[1] 平川秀樹，天野真家，"構文／意味優先規則による日本語解析"，人工知能学会，第 3 回全国大会論文集，1989.

[2] 平川秀樹，天野真家，"日本語解析における最適解探索"，情報処理学会，自然言語処理研究会 74 − 2, 1989.

[3] 平川秀樹，"最適解探索に基づく日本語意味係り受け解析"，情報処理学会論文誌,Vol.43,No.03, 2002.

[4] R. Kaplan, "The Formal Architecture of Lexical-Functional Grammar", Journal of Information Science and Engineering 5, 305-322, 1989.

[5] P. Mertens, "Parsing Dependency Grammar using ALE", Proceedings COLING 2002, vol. 2, p. 653-659, 2002

[6] M. Tomita, "An efficient augmented context-free parsing algorithm", Computational Linguistics, Vol.13, 1987.

[7] J. Seo and R. F. Simmons, "A Syntactic Graphs: A Representation for the Union of All Ambiguous Parse Trees", Computational Linguistics, Vol.15, 1989.

[8] H. C. Rim, J. Seo, and R. F. Simmons, "Transforming Syntactic Graphs into Semantic Graphs", Technical Report AI90-127, Artificial Intelligence Lab, Texas at Austin, 1990.

[9] M. Schiehlen, "Semantic Construction from Parse Forests", In Proceedings of the 16th International Conference on Computational Linguistics (COLING'96), 1996.

[10] T. Winograd, "Language as a Cognitive Process: Volume 1 Syntax", Addison-Wesley Publishing, 1983.

付録１：依存グラフの問題と依存森での処理例

次の文法規則を与えて “Tokyo taxi driver call center” をスタートシンボルを np として解析する場合を考える．

```
### Grammar Rules
np/NP  → npc/NP : []
npc/Nb → np1/NP1,n/Na,n/Nb:[arc(nj,NP1,Nb),arc(nc,Na,Nb)]
npc/Na → np2/NP2,n/Na     :[arc(nc,NP2,Na)]
npc/Na → np3/NP3,n/Na     :[arc(nc,NP3,Na)]
np1/Nc → n/Na,n/Nb,n/Nc   :[arc(nc,Na,Nb),arc(nc,Nb,Nc)]
np2/Nd → n/Na,n/Nb,n/Nc,n/Nd :
         [arc(nj,Na,Nc),arc(nc,Nb,Nc),arc(nc,Nc,Nd)]
np3/Nd → n/Na,n/Nb,n/Nc,n/Nd] :
         [arc(nc,Na,Nb),arc(nj,Nb,Nd),arc(nc,Nc,Nd)]
### Lexicon
word(n,[Tokyo]). word(n,[taxi]). word(n,[driver]).
word(n,[call]).  word(n,[center]).
```

この例文では，図 11 の (a),(b),(c) の 3 つの依存木が解として存在する．依存木の $np1, np2, np3$ の箱は，句構造と依存構造の対応を分かりやすく示すために補助的に入れている．(a) では，nc-1,nc-2, (b) では，nc-2,nc-3, (c) では，nc-3,nc-1 の間で共起関係が成立するため，図の構文グラフ/排他マトリックスにおいて，それぞれ対応する排他マトリックスは “ ” となっている．このため，(d) の依存木も依存グラフ/排他マトリックスに存在するが，これに対応する構文木は存在せず，健全性が損なわれている．

同じ例文に対する初期依存森を図 12(a) に示す．図のマトリックスでは，同値アークのまとまりを 2 重線で示しており，初期依存森には (1,8),(2,6),(7,10),(20,21,22) の 4 種の同値アークが存在する．図 10 のアルゴリズムに従って縮退処理が行なわれる．最初の同値アーク (1,8) は，危険なマージを含まないので $mergeable(1,8) == true$ となり，不確定アークの情報設定が行なわれる．$diff(1,8)$ は，$\{2,19,20\}$ と $\{9,10,22\}$ の組み合わせとなる．最初のアークペア (2,9) は，$same_position(2,9)$ であるため $CM(2,9)$ には特に何もセットされない．一方，アークペア (2,10) は，星アークペアと判定され，$CM(2,10)$ に☆がセットされる．図 12(b) は，同値アーク 8 がアーク 5 にマージされた結果である．以降順次同値アークペアが処理される．

図 12(c) は，最終的に得られる縮退依存森であり，同値アークペア (7,10) を持つ．この依存森に対する縮退アルゴリズムの動作を示す．同値アーク $same(7,10)$ の CM を見ると，アーク 2 に対して，$CM(2,7) ==$ ○ かつ $CM(2,10) ==$ ☆ のため，危険なマージであり，関数 $mergeable$ で解の増大チェックが行なわれる．この場合，$CRCL = 7, STR = 10, B = 2$ であり，$MRG = mrg(CRCL, STR) = 7$(但し，CM はアーク 7 と 10 をマージした値) となる．また，$co(MRG) = \{1,2,5,7,9,20,23\}$,$co(B) = \{1,2,5,7,19,20,23\}$，$co(MRG) \cap co(B) = \{1,2,5,7,20,23\}$，$uniq(STR, CRCL) = uniq(7,10) = \{2,5,7\}$ である．$dt(co(MRG) \cap co(B))$ は，アーク $MRG = 7$ と $B = 2$ を含む整依存木 $\{1,2,7,20,23\}$ を含み，$dt(co(MRG) \cap co(B)) \cap uniq(STR, CRCL) = \{2\}$ となるため，マージ不可と判定される．この依存木は構文グラフで過生成される木である．

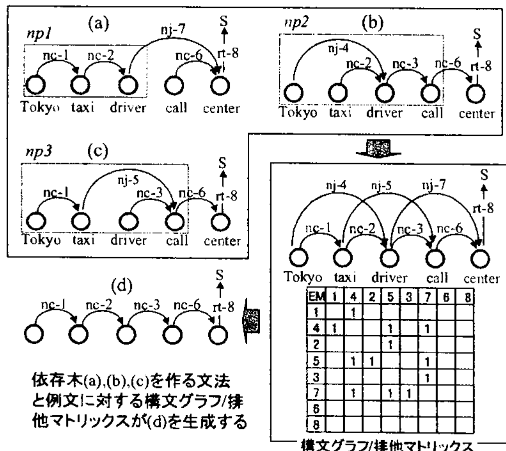


図 11: 依存木と構文グラフ/排他マトリックス

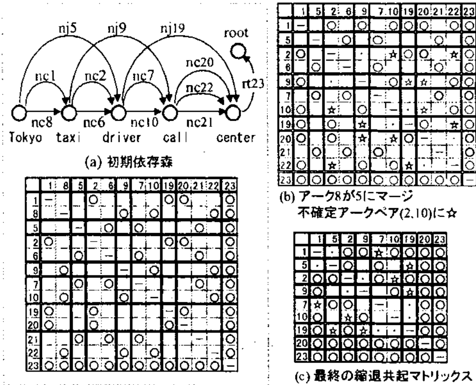


図 12: 例文に対する初期依存森と縮退処理

最終の縮退依存グラフは (a) の初期依存グラフからアーク 8,6,21,22 を取り除いたものとなる．この依存森は，3 つの依存構造のみを保持しており，健全性が保たれている．

付録２：初期依存森の完全性と健全性の証明

単語数 w の文に対するヘッド付き統語森 PF(以下簡略して統語森と記す)，依存森 DF(依存グラフ DG, 共起マトリックス CM より構成) を想定する．構文木 PT は，入力文を被覆し，アーク数 $w-1$ の 1 つの依存構造を生成する．整依存木 DT は整依存木条件 (整共起条件と整被覆条件) を満足する．

まず，各種の前提や用語を説明する．PF, DF の構成アルゴリズムより，あるアーク $a(a \in DT)$ に対して，対応する部分依存構造 $ds(a)$, 弧 $e(a)$ が統語森に 1 つだけ存在する．$e(a)$ は，図 4 の構成を持つが，分かり易さのため $FCSL$ と DSL を CSDS ペアの列 $csds_1, .., csds_m$ で表現し，次のように記述する．

$< e_{id}\ csds_1 | \cdots | csds(a) | \cdots | csds_m >$

ここで，e_{id} は，弧 ID である．アーク a_i に対応する CSDS ペアを $csds(a_i)$ あるいは簡略化して $csds_i$ と記述する．$FCSL$ や DSL の要素を示す場合は，$cs(a_i)$ あるいは cs_i, $ds(a_i)$ あるいは ds_i と記述する．

【完全性】

完全性が成立するとは，「統語森中の構文木 PT に対応する整依存木が依存森に存在する」ことである．

(1) 弧に含まれる構成素列 cs を 1 つ選択し，cs 中の弧を辿るという操作をルート弧 e_r から行なうことにより統語森 PF 中の構文木 PT を得る．この際，選択された構成素列 cs に対応する部分依存構造 ds の集合を $ds(PT)$ と記述する．部分依存構造 ds は，部分依存構造条件を満足するため，依存構造 $ds(PT)$ は木となる．これを依存木 DT とする．

(2) DT は，依存グラフの構築アルゴリズムから，DG に含まれる．また，DT と PT の含むノードは部分依存構造条件より 1 対 1 の対応関係があり，PT が文を被覆するので，DT は整被覆木である．

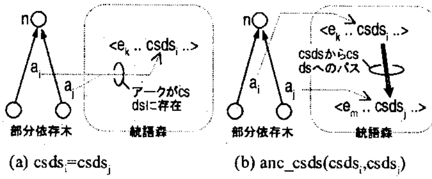
(3) DT 中の全アークに関し，共起マトリックスの構築アルゴリズムより共起関係が成立し DT は整共起依存木となる．

よって，統語森中の構文木 PT に対応する整被覆整共起依存木 DT が依存森に存在する．

【健全性】

健全性が成立するとは，「依存森中の整依存木 DT が与えられた時に，それに対応する構文木が統語森に存在する」ことである．

まず，以下で用いる用語の定義を行う．アーク a の支配ノードを $gov(a)$, 依存ノードを $dep(a)$ と記述し，親子関係にあるという．親ノードを同一とするアーク a_i, a_j を $bro_arc(a_i, a_j)$ と記述し，兄弟アークと呼ぶ．また，上下に連結しているアーク a_i ,a_j を $par_arc(a_i, a_j)$ と記述し，



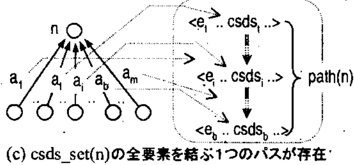


図 13: 依存木と統語森の構造的対応関係 (1)

それぞれ親アーク，子アークと呼ぶ．

PF は，ルート弧 e_r をルートとし語彙弧をリーフとする非循環有向グラフ (DAG) であり，その経路をパスと呼ぶ．PF 上で，弧 e から CSDS ペア $csds$ に到達するパスが存在ことを $anc_edge(e, csds)$ と記述する．また，ある CSDS ペア $csds_i$ から $csds_j$ へ至るパスが存在する場合に $anc_csds(csds_i, csds_j)$ が成立するという．また，ある $csds_i, csds_j$ に対して，$anc_edge(e_u, csds_i), anc_edge(e_v, csds_j)$ かつ $e_u, e_v \in cs_c$ となる CSDS ペア $csds_c$ を結合 CSDS (conjoinning CSDS) と呼び $conj(csds_i, csds_j)$ と記述する．

[1] 兄弟アークと CSDS ペアの関係

DT 中のノード n の下の兄弟アーク a_i, a_j と対応する PF 中の CSDS ペア $csds_i, csds_j$ の関係を考える．a_i, a_j は共起条件を満足するので，共起マトリックスの構成アルゴリズムより，$csds_i, csds_j$ は，次の (r1)〜(r3) のいずれかの関係を満足している必要がある．

(r1) $csds_i = csds_j$
(r2) $anc_csds(csds_i, csds_j)$ 又は $anc_csds(csds_j, csds_i)$
(r3) $conj(csds_i, csds_j)$ が存在する

ここで，(r3) の場合を考える．CDDS ペア $conj(csds_i, csds_j)$ の cs 中に $anc_edge(e_u, csds_i)$, $anc_edge(e_v, csds_j)$ なる e_u, e_v がそれぞれ存在する．e_u, e_v の被覆する入力文の始点，終点をそれぞれ $(f_u, t_u), (f_v, t_v)$ とすると $f_u < t_u \leq f_v < t_v$ または $f_v < t_v \leq f_u < t_u$ のいずれかでなければならない．一方，a_i, a_j の共通の支配ノード n の位置 $position(n)$ は，$f_u \leq position(n) < t_u$ かつ $f_v \leq position(n) < t_v$ でなければならず，矛盾が生じる．よって，(r3) は成立せず，$csds_i$, $csds_j$ に対して，(r1) または (r2) が成立する (図１３ (a),(b))．ここで，(r1) または (r2) が成立することを $anc_or_eq_csds(csds_i, csds_j)$ と記述する．$bro_arc(a_i, a_j)$ であれば，$anc_or_eq_csds(csds_i, csds_j)$ が成立する．今，ノード n の下のアークに対応する CSDS ペアの集合 $\{csds(a_i)|gov(a_i) = n\}$ を $csds_set(n)$ と記述する．$csds_set(n)$ の全要素間に $anc_or_eq_csds$ の関係が成立すること，統語森は非循環有向グラフであることから，次が成立する．

「$csds_set(n)$ 中の全 CSDS ペアを経由するパスが PF 中に存在する.」— (A)

このパスを $path(n)$ と記述する．図 13(c) に示すように，パスのトップをトップ CSDS ペア $csds_t$，ボトムをボトム CSDS ペア $csds_b$ と呼ぶ．$path(n)$ の要素が 1 つの場合は，$csds_t = csds_b$ である．

[2] 親子アークと CSDS ペアの関係

DT 中の親子アーク $par_arc(a_i a_j)$ を考える．共起条件より，前出の (r1)〜(r3) のいずれかが成立するが，$dep(a_i) = gov(a_j)$ であるため，兄弟アークの場合と同様の議論により，(r3) は成立しない．また，(r2) では $anc_csds(csds_j, csds_i)$ を仮定すると，文法規則の部分依存構造条件による $ds(a_j)$ と $ds(a_i)$ の位置関係より，$dep(a_i) \neq gov(a_j)$ である．これは，前提 $par_arc(a_i, a_j)$ と矛盾し，$anc_csds(csds_j, csds_i)$ はありえず，次が成立する．

「DT 中の $par_arc(a_i, a_j)$ に対して $csds_i = csds_j$ または $anc_csds(csds_i, csds_j)$ が成立する」 – (B)

[3] 整依存木と依存森の構造的関係

図 14 を用いて DT を構成する全アークと依存森中の CSDS ペアの間に成り立つ構造的関係を説明する．(A) より，DT 中のノード n_u, n_v に対する $csds_set(n_u), csds_set(n_v)$ は，それぞれ I, II に示すような $path(n_u), path(n_v)$ を持つ．(B) より，$path(n_u), path(n_v)$ は，PF 中において図 14(a) または (b) のいずれかの構成となり，連接する．

「DT 中の親子ノード n_u, n_v に対する $csds_set(n_u)$, $csds_set(n_v)$ が PF 中で構成するパス $path(n_u), path(n_v)$ は連接関係にある」 – (C)

一方，DT を成すアークに対応する CSDS ペアの集合 $CSDS_{DT}$ に対して一般に次が成立する．

「$CSDS_{DT}$ 中の任意の要素 $csds_k, csds_l, csds_u, csds_v$ 13 のループ構造が存在することは無い」 – (D)

これは，$csds_u$ と $csds_v$ が cs_k の中の異なった要素 (弧) に支配されることから，それぞれの句範囲の共通部分がないことが言え，$csds_l$ が存在しえないことから説明できる．

[4] DT に対応する PT の存在

DT 中のノードに対するパスの集合は，(C) と (D) より PF 上で連接して木の構造を成す．このトップ CSDS を $csds_t$ とする．構文木は，PF のルート弧から，弧の構成素列すなわち CSDS ペアを 1 つ選択しながら，全てリーフの弧に辿りつくまで下位の弧を選択してゆくことで得られる．PF の定義よりルート弧 e_r に対して $anc_edge(e_r, csds_t)$ が成立し，$csds_t$ を含むよう CSDS ペアを選択してゆくことができる．$csds_t$ 以下では，上記連接したパス集合に沿って CSDS ペアを選択してゆくことができる．こうして構文木 PT を構成することができる．DT は，整被覆条件より全ての入力位置に対するノードを 1 つづつ含んでおり，木 PT もそれらを含むため文全体をカバーしている．以上より，DT に対応する PT が少なくとも 1 つ PF に存在すると

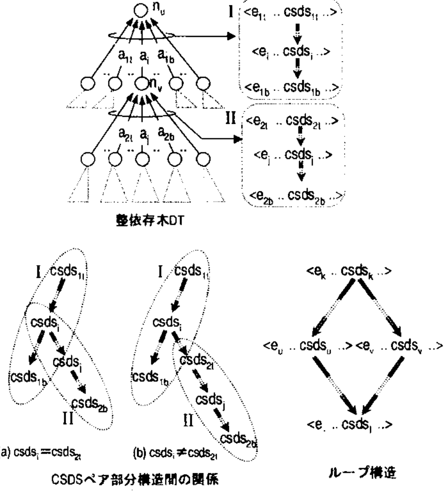


図 14: 依存木と統語森の構造的対応関係 (2)