自 然 言 語 処 理 81－4
（1991. 1. 17）

チャート構造上での意味処理メカニズム

田村直之[1] M. J. W. Bos[1] 吉見毅彦[2] 西田収[2] J. Jelinek[3] 村上温夫[1]

1. 神戸大学大学院自然科学研究科 2. シャープ株式会社 3. シェフィールド大学

本稿では，木の集合を表現したデータ構造上で，優先性 (preference) を効率良く評価するための新しい手法を提案する．木の集合を表すデータ構造としてはチャート構造 (正確には，富田の packed shared forest [14]) を用い，最も優先性の高い木を効率良く選ぶためにストリームの遅延評価を用いた．

この手法は，共照応性解析 [6, 5] に基づく文脈処理システムのために設計したものであるが，次のように得点づけを行なう様々なシステムに対して応用可能である．

(1) 葉節点には，辞書の語彙情報に基づいた得点 (通常は 0 点) が与えられるものとする；

(2) 中間の節点の得点は，その子節点の得点の合計に，その非葉節点が満たす構造上の条件によって動的に変化する付加的な得点 (負でもよい) を加えたものとする．

A SEMANTIC PROCESSING MECHANISM WORKING ON CHART STRUCTURES

N. Tamura[1] M. J. W. Bos[1] T. Yoshimi[2] O. Nishida[2] J. Jelinek[3] H. Murakami[1]

1. The Graduate School of Science and Technology, Kobe University
2. SHARP Corporation
3. Centre for Japanese Studies, The University of Sheffield

This paper proposes a new method enabling the efficient evaluation of preference on a data structure representing a set of trees. Tomita's packed shared forest [14] is used as the data structure. Lazy (or demand-driven) evaluation technique for streams is used for efficient evaluation of preference.

Our method is originally designed for the analysis of correferentiality. But we believe our approach can be adopted for various systems which do scoring as follows : score of a leaf node of a tree is acquired from the lexical entry for the word sense (usually 0); score of a non-leaf node is the sum of the scores of its child nodes and the additional score (possibly minus) whose value varies depending on the structural conditions.

1 はじめに

優先性 (preference) は, 構文解析あるいはある種の意味処理において曖昧性を解消する上で, 単純ではあるが有効な手段であることは良く知られている [2] [3] [8] [10] [11] [16].

しかし, 優先性を有効に利用するためには, 以下のような問題点があげられる.

(1) 優先性をどうやって決めるか. すくなくとも優先性に関する理論が必要である. 著者らの一人は, 文脈処理に関する理論を発表している [6, 5]. さらにそれを実用的なものにするためには, 大規模な実験が必要であろう.

(2) 異なる観点から与えられた優先性をどうやって統合するか. 構文, 意味, 文脈等の異なった観点から与えられた得点を統合する方法を定義する必要がある. これについても大規模な実験が必要であろう.

(3) 最良の解釈をどうやって効率的に選び出すか. 全ての木について別々に評価していたのでは, 非常に効率が悪い. 特に, 処理の対象が一文ごとでなくテキスト全体であるようなシステム (たとえば, TWINTRAN [4]) の場合, 各文ごとに多くの解析木が存在するから, テキスト全体での可能な解析木の数は膨大になる.

本稿では, (3) の問題を解決するための方法を示す [13]. すなわち, 可能な解析木中から, 最良の木を効率的に選び出すためのメカニズムを提案する. 主なアイデアは次の二つである.

(1) 木の集合を表すためデータ構造を導入する;

(2) 得点のついたチャート構造から得点の高い木を効率よく選ぶために, 遅延評価 (lazy evaluation) の技術を用いる.

木の集合を表すデータ構造としては, チャート構造 (正確には, 冨田の packed shared forest [14]) を用いる. チャート構造はチャートパーザ [7] によって作られる. ただし, 本稿では簡単のため, AND-OR グラフを説明に使用する. チャート構造を意味処理に用いるアイデアは, Alshawi 他によって既に発表されている [1].

評価の回数を減らすために, ストリームでの遅延評価の技法を用いる. 与えられた AND-OR グラフにおいて, 各枝にはストリームを対応させ, 各節点にはストリーム演算を対応させる. 大まかに言うと, ストリームの各要素は対応する枝の下の部分木を表し, 得点の降順に整列されている. ストリームの要素は葉節点から根節点へ上向きに流れていく. 各 OR 節点では, 小節点からのストリームが併合 (merge), 整列される. 各 AND 節点では, 子節点からのストリームが結合 (combine) される. 得点は全ての子節点と付加ストリーム (additional stream) の得点の和で与えられる. 付加ストリームの得点は結合される部分木の構造により変化してよい. 根節点に最初に到着した要素が, 最も優先性の高い木を表している. ストリームの流れの制御は遅延評価により行なわれる. すなわちストリームの要素の生成は, それが実際に必要とされるまで遅延される. AND-OR グラフの探索アルゴリズムである AO* アルゴリズム [9] をこの問題に用いることはできない. なぜなら, AO* アルゴリズムでは, 構造上の条件によって変化するような付加的な得点を扱うことができないからである.

2 AND-OR グラフと OR 子供リスト表現

ここでは, 本論文の手法が適用可能となるデータ構造を定義する.

AND-OR グラフは, ただ一つの根節点を持つ閉路のない有向グラフである. グラフ上の各節点は AND 節点か OR 節点のいずれかである (各節点は自然数でラベル付けされているものとする). 節点 v と節点 w が v から w へ向かう枝で結ばれているとき, v を w の親節点, w を v の子節点という. 親節点を持たない節点を根節点, 子節点を持たない節点を葉節点と呼ぶ. また, 複数個の親節点を持つ節点を被共有節点 (shared node) と呼ぶ. 一般に葉節点は AND 節点でも OR 節点でもよいが, 本稿では簡単のため葉節点はすべて AND 節点とする.

AND-OR グラフは, 木の集合を表しているとみなせる (厳密には, 被共有節点があるので木ではないが). G を AND-OR グラフとし, G に含まれる木の全体集合を $T(G)$ で表すと, $T(G)$ は次の規則にしたがって構成される木 T を含む.

(1) G の根節点は T に含まれる;

(2) T に含まれる節点 v が G の AND 節点ならば，v のすべての子節点は T に含まれる；

(3) T に含まれる節点 v が G の OR 節点ならば，v の子節点のいずれか一つだけが T に含まれる．

各 OR 節点においてどの子節点を選ぶかを規定すれば，一つの木 T を特定できる．AND-OR グラフ G 上の OR 節点を $v_1, v_2, \ldots, v_n$ (ただし，$v_1 < v_2 < \cdots < v_n$) とし，各 v_i での全ての子節点の集合を O_i とする．次の規則により生成されるリスト $[A_1, A_2, \ldots, A_n]$ を $T(G)$ の要素 T の**有効** (valid) 表現という．

(1) $v_i \in T$, $w \in O_i$, かつ $w \in T$ のとき，$A_i = w$；

(2) $v_i \notin T$ のとき，$A_i = 0$

AND-OR グラフに含まれている木の集合の部分集合を表現するために，**OR 子供リスト** (OR child list) と呼ぶ表現を導入する．OR 子供リストは，O_i の部分集合 S_i を要素とするリスト $[S_1, S_2, \ldots, S_n]$ である．OR 子供リストから次の規則により生成されるリスト $[A_1, A_2, \ldots, A_n]$ のすべてが $T(G)$ 上の木の有効表現になっているとき，その OR 子供リストは**有効**であるという．

(1) $S_i \neq \emptyset$ ならば ($\emptyset$ は空集合)，S_i の要素の一つを A_i とする；

(2) $S_i = \emptyset$ ならば，$A_i = 0$ とする．

以後，本稿では有効な OR 子供リストだけを考慮する．また，S_i が O_i に一致する場合，S_i のすべての要素を列挙するかわりに下線 "_" を省略記法として用いる．

二つの OR 子供リストの共通集合は簡単に求めることができる．OR 子供リストの各要素ごとの共通集合を計算すればよい．$P = [S_1, S_2, \ldots, S_n]$ と $Q = [U_1, U_2, \ldots, U_n]$ を OR 子供リストとするとき，共通集合 $P \cap Q$ は，$S_i \cap U_i$ を第 i 要素とする OR 子供リストである．ただし，ある i について，S_i と U_i の少なくとも一方は空でなく，かつ共通集合 $S_i \cap U_i$ が空となるとき，$P \cap Q$ は空であるとする．このとき $P \cap Q$ は，いかなる木も含んでいない．$P \cap Q$ が空であるとき P と Q は**相容れない** (inconsistent) といい，そうでないときは**相容れる** (consistent) という．

3 優先性と得点付け

ここでは，本論文の手法が適用可能となる得点付けの条件を述べる．

次のような計算過程により解析木に優先順位を与えるものと仮定する (Hobbs らも同様の定義を行っている [3])．木の葉節点の得点は 単語の意味を表す語義見出しから与えられる (通常は 0 点)．非葉節点の得点は，その子節点の得点の合計に，その非葉節点が満たす構造上の条件によって動的に変化する付加的な得点 (負でもよい) を加えたものとする．木全体としての得点は，その根節点での得点である．その得点に従って木に優先順位を付け，最高点を得た木を最適解釈として選択する．

得点が加えられるための条件や得点の値は，AND 節点に付随している．詳細は 5節で述べる．

4 ストリーム・プログラミング

ストリームは，関数型プログラミング言語や Concurrent Prolog, PARLOG, GHC などの並列論理プログラミング言語に関連して良く知られている．本稿では，プログラムの記述には GHC [15] (厳密にいえば，flat GHC) を用いるが，多少の修正を加えるだけで逐次型 Prolog 処理系上でも実現できる．Prolog-II や NU-Prolog, ESP のような中断機構 (freeze 述語や wait 宣言など) を持つ Prolog の方言を用いて本稿のプログラムを書き換えることは非常にたやすい．また，通常の Prolog 処理系上で実現することも可能である．

GHC における遅延評価を実現するために，文献 [12] で提案された有界バッファ (bounded buffer) の技法を用いる．ストリーム要素の生成は，それがプログラムの他の部分から実際に要求されるまで中断される．

4.1 順序付きストリーム

我々の目的は，与えられた AND-OR グラフから，その優先性 (得点) が高い順に木を選び出すことである．これを実現するために，**順序付きストリーム** (ordered stream) を利用する．

順序付きストリーム

$$[a_1 : P_1, a_2 : P_2, \ldots, a_m : P_m]$$

は, OR 子供リスト P_i とそれが持つ得点 a_i との対 (コロン ':' で区切る) を要素に持つ. ストリームの要素は次の条件を満たすものとする.

(1) $a_1 \geq a_2 \geq \cdots \geq a_m$;

(2) 任意の $i \neq j$ に対して, $P_i \cap P_j = \emptyset$.

以後, ストリームの要素間の大小関係はその得点の大小関係とする. たとえば, ストリーム要素 $a : P$ と $b : Q$ について, 得点 a が得点 b より大きいとき, $a : P$ は $b : Q$ より大きいと言う.

4.2 順序付きストリームの遅延併合

併合 (merge) 操作は, マージソートのプログラムと同様にして, 二つの順序付きストリームから新たに順序付きストリームを生成する. $[X_1, X_2, \ldots, X_m]$ と $[Y_1, Y_2, \ldots, Y_n]$ を入力ストリームとするとき, これらを併合して得られるストリーム $[Z_1, Z_2, \ldots, Z_{m+n}]$ は, 次の条件を満たすストリームである.

(1) 任意の X_i に対して, $X_i = Z_h$ を満たす h が存在する;

(2) 任意の Y_j に対して, $Y_j = Z_h$ を満たす h が存在する;

(3) 任意の Z_h に対して, $Z_h = X_i$ を満たす i, あるいは $Z_h = Y_j$ を満たす j が存在する;

(4) $Z_1 \geq Z_2 \geq \cdots \geq Z_{m+n}$.

図 1 に GHC で記述した遅延併合プログラムを示す. 述語 `merge(Xs,Ys,Zs)` は, 順序付きストリーム `Xs`, `Ys` を併合したストリーム `Zs` を返す. アトム `eos` は, ストリームの終わりを示す記号である.

第三引数が他のプログラムによってリストに束縛するまで, `merge` の実行は中断される. 第三引数が束縛されると, `Xs` と `Ys` もリストに束縛される. 第一要素 `X` と `Y` がなんらかの値に束縛されるまで, 述語 `lazy_merge` で再び実行が中断される.

4.3 順序付きストリームの遅延結合

結合 (combine) 操作は, 与えられた二つの順序付きストリームから, それらのストリームの要素のすべての相容れる組み合わせからなる順序付きストリームを新

```
merge(Xs,Ys,[Z|Zs]) :- true |
  Xs=[X|Xs1], Ys=[Y|Ys1],
  lazy_merge(X,Xs1,Y,Ys1,Z,Zs).

lazy_merge(X,Xs,Y,Ys,Z,Zs) :-
  X=eos, Y=eos | Z=eos.
lazy_merge(X,Xs,Y,Ys,Z,Zs) :-
  X\=eos, Y=eos | Z=X, Zs=Xs.
lazy_merge(X,Xs,Y,Ys,Z,Zs) :-
  X=eos, Y\=eos | Z=Y, Zs=Ys.
lazy_merge(X,Xs,Y,Ys,Z,Zs) :-
  X=(A:P), Y=(B:Q), A>=B |
  Z=X, merge([Y|Ys],Xs,Zs).
lazy_merge(X,Xs,Y,Ys,Z,Zs) :-
  X=(A:P), Y=(B:Q), A<B |
  Z=Y, merge(Ys,[X|Xs],Zs).
```

図 1: 順序付きの併合

たに生成する. 生成されるストリームの要素は, 二つの入力ストリームの要素が持つ得点の合計点と, 入力ストリームの要素の OR 子供リストの共通集合との対である. $[X_1, X_2, \ldots, X_m]$ と $[Y_1, Y_2, \ldots, Y_n]$ を二つの順序付きストリームとするとき, それらを結合したストリーム $[Z_1, Z_2, \ldots, Z_k]$ は次の条件を満たすストリームである.

(1) 任意の相容れる X_i と Y_j の組に対し, $X_i + Y_j = Z_h$ を満たす h が存在する;

(2) 任意の Z_h に対し, $Z_h = X_i + Y_j$ を満たす X_i と Y_j が存在する;

(3) $Z_1 \geq Z_2 \geq \cdots \geq Z_k$.

ただし $X + Y$ は, X が $a : P$ であり Y が $b : Q$ であるとき, $a + b : P \cap Q$ と定義する.

図 2 に GHC で記述した遅延結合プログラムの第一版を示す. 述語 `sum(X,Y,Z)` は, 和 `X+Y` を `Z` に返す. ただし, `X` と `Y` が相容れないときは, `Z` の OR 子供リストには, `empty` という特別な記号が返ってくるものとする.

`combine` の実行は, 第三引数が他のプログラムによってリストに束縛されるまで中断される. 第三引数が束縛されると, `Xs` と `Ys` もリストに束縛される. `lazy_combine` の最初の節は, 第一引数が空のストリームのとき空のストリームを返し, 二番目の節は第二引数が空のストリームのとき空のストリームを返す. 三番目の節は, 二つの

```
combine(Xs,Ys,[Z|Zs]) :- true |
  Xs=[X|Xs1], Ys=[Y|Ys1],
  lazy_combine(X,Xs1,Y,Ys1,Z,Zs).

lazy_combine(X,Xs,Y,Ys,Z,Zs) :-
  X=eos | Z=eos.
lazy_combine(X,Xs,Y,Ys,Z,Zs) :-
  Y=eos | Z=eos.
lazy_combine(X,Xs,Y,Ys,Z,Zs) :-
  X\=eos, Y\=eos | sum(X,Y,Z0),
  lazy_combine2(X,Xs,Y,Ys,Z0,Z,Zs).

lazy_combine2(X,Xs,Y,Ys,Z0,Z,Zs) :-
  Z0=(B:Q), Q\=empty | Z=Z0,
  combine_rest(X,Xs,Y,Ys,Zs).
lazy_combine2(X,Xs,Y,Ys,Z0,Z,Zs) :-
  Z0=(B:Q), Q=empty |
  combine_rest(X,Xs,Y,Ys,[Z|Zs]).

combine_rest(X,Xs,Y,Ys,Zs) :- true |
  combine([X,eos],Ys,Zs1),
  combine([Y|Ys],Xs,Zs2),
  merge(Zs1,Zs2,Zs).
```

図 2: 順序付きの結合 (第一版)

ストリームのそれぞれの第一要素がそれぞれXとYである場合を処理する. XとYの和はZ0に求められ, XとYが相容れるかどうかはlazy_combine2でテストされる. lazy_combine2の最初の節は, Z0のOR子供リストがemptyでないとき, すなわちXとYが相容れるときに実行される. 和Z0は, 出力ストリームの第一要素Zとして返され, 残りの組み合わせは, プログラムcombine_restを用いて順序付きストリームZsとして返される. 二つのストリームの要素をバランス良く利用するため, combine(Xs,[Y|Ys],Zs2)ではなく, 第一引数と第二引数を交換してcombine([Y|Ys],Xs,Zs2)としている. lazy_combine2の二番目の節は, XとYが相容れない場合のためのものである.

このプログラムには, 相容れる組み合わせを見つけるまで停止しないという問題点がある. XとYが相容れない場合, lazy_combine2の二番目の節が実行され, 要求はこのレベルに停止せず更に低いレベルに渡されてしまう. つまり, combine_restの中の述語merge(Zs1,Zs2,Zs)において, Zsはすでにlazy_combine2の二番目の節のリストと束縛されている. したがって, 併合プログラムは中断されず, 即座にZs1とZs2がリストと

```
combine(Xs,Ys,Zs) :- true |
  merge_combine([eos],Xs,Ys,Zs).

merge_combine(Ws,Xs,Ys,[Z|Zs]) :- true |
  Ws=[W|Ws1], Xs=[X|Xs1], Ys=[Y|Ys1],
  lazy_merge_combine(W,Ws1,X,Xs1,Y,Ys1,Z,Zs).

lazy_merge_combine(W,Ws,X,Xs,Y,Ys,Z,Zs) :-
  X=eos | Z=W, Zs=Ws.
lazy_merge_combine(W,Ws,X,Xs,Y,Ys,Z,Zs) :-
  Y=eos | Z=W, Zs=Ws.
lazy_merge_combine(W,Ws,X,Xs,Y,Ys,Z,Zs) :-
  X\=eos, Y\=eos | sum(X,Y,Z0),
  lazy_merge_combine2(W,Ws,X,Xs,Y,Ys,Z0,Z,Zs).

lazy_merge_combine2(W,Ws,X,Xs,Y,Ys,Z0,Z,Zs) :-
  W=eos, Z0=(B:Q), Q\=empty | Z=Z0,
  merge_combine_rest([eos],X,Xs,Y,Ys,Zs).
lazy_merge_combine2(W,Ws,X,Xs,Y,Ys,Z0,Z,Zs) :-
  W=eos, Z0=(B:Q), Q=empty |
  merge_combine_rest([eos],X,Xs,Y,Ys,[Z|Zs]).
lazy_merge_combine2(W,Ws,X,Xs,Y,Ys,Z0,Z,Zs) :-
  W=(A:P), Z0=(B:Q), A>=B | Z=W,
  merge_combine(Ws,[X|Xs],[Y|Ys],Zs).
lazy_merge_combine2(W,Ws,X,Xs,Y,Ys,Z0,Z,Zs) :-
  W=(A:P), Z0=(B:Q), A<B, Q\=empty | Z=Z0,
  merge_combine_rest([W|Ws],X,Xs,Y,Ys,Zs).
lazy_merge_combine2(W,Ws,X,Xs,Y,Ys,Z0,Z,Zs) :-
  W=(A:P), Z0=(B:Q), A<B, Q=empty |
  merge_combine_rest([W|Ws],X,Xs,Y,Ys,[Z|Zs]).

merge_combine_rest(Ws,X,Xs,Y,Ys,Zs) :- true |
  merge_combine(Ws,[X,eos],Ys,Zs1),
  merge_combine(Zs1,[Y|Ys],Xs,Zs).
```

図 3: 順序付きの結合 (改訂版)

束縛される. これにより, combine([X,eos],Ys,Zs1)とcombine([Y|Ys],Xs,Zs2)の実行が再開される. このように, combineは相容れる組み合わせを見つけるまで再帰的に実行を継続する.

しかし, 図3に示すプログラムは, より遅延的に実行される. merge_combine(Ws,Xs,Ys,Zs)の論理的意味は, combine(Xs,Ys,Zs1), merge(Ws,Zs1,Zs)に等しい.

lazy_merge_combine2の三番目の節で, Wの得点がXとYの得点の合計より小さくないときは, たとえXとYが相容れない組み合わせであっても, Wが一番目の要素として返される. そのため, 要求が停止する可能性は第一版のプログラムより高くなる.

5 優先性に対する遅延評価

この節では, AND-OR グラフから順序付きストリームを生成する方法を示す.

与えられた AND-OR グラフにおいて, 枝を順序付きストリームに, 節点をストリーム操作に対応させる. ストリームの要素は, 葉節点から根節点へ向かって上方へ流れる. 葉節点における初期得点, および非葉節点における付加的な得点は, それらに与えられた**付加ストリーム** (additional stream) とよぶストリームから得られるものとする. 付加ストリームは, 得点と OR 子供リストの対を要素とする順序付きストリームであり, OR 子供リストはその得点が加えられるための構造上の条件を示す. ひとつの節点に, 複数個の付加ストリームがあってもよい.

葉節点で付加ストリームを結合したストリームを生成し, 親節点に向かって上方に流す. 非葉節点が AND 節点ならば, 4.3節で述べた combine プログラムを用いて子節点からの全てのストリームを結合したストリームを生成し, さらに付加ストリームがあれば, それと結合する. OR 節点の場合は, 4.2節で述べた merge プログラムを用いて, 全ての子節点からのストリームを併合したストリームを生成し, さらに付加ストリームがあれば, それと結合する.

根節点に到達したストリームの最初の要素が, 最も優先順位の高い木を表している.

6 例

次の二つの文からなる文章を英文に翻訳する問題を考える.

(1) 医者が秘書を呼んだ.

(2) 二人の医者の秘書達が彼女の部屋に来た.

問題は, 図 4に示す構文解析結果 (便宜上葉節点には英訳語が割り当ててある) の AND-OR グラフの中から最も優先順位の高い木を選択することである. それぞれの文には 16 通りの解析木があり, 文章全体として 256 通りの解析木を表している. 図中, 節点 4, 5, 7, 26, 27 が OR 節点である.

おそらく最も望ましい木の一つは次の英文で表される解釈を表すものであろう.

- The doctor called the secretaries.
- The two secretaries of the doctor came to her room.

これは, OR 子供リスト [{12}, {17}, {18}, {31}, {35}] で表現される. 望ましいと考えられる理由は, 第一文の「秘書」は第二文から複数と判断できることから, 「彼女」は単数の「医者」を指し, 「二人の」は「医者」ではなく「秘書達」を修飾すると考えられるからである.

解析木を優先順位に従って順に抽出するために, 各節点に対して表 1に示す付加ストリームを与える. 表に記述されていない葉節点には, その節点に対する得点 0 とその節点が木に含まれる条件を表す OR 子供リストとの対を要素とする付加ストリームが与えられているとする. 例えば, 葉節点 10 は付加ストリーム [0 : [{10}, _, _, _, _]] を持つ. 一般に, 各節点は複数個の付加ストリームを持つことができるが, この例では各節点は高々一つの付加ストリームしか持っていない.

表 2に各節点において得られたストリームを示す.

7 結論

本稿では, 木の集合を表すデータ構造とその構造上で優先性を効率的に評価する新しい手法を提案した. 本手法は, 並列論理プログラミング言語に関連して広く知られている遅延評価技法を基礎にしているが, 逐次型のシステムにも適用できる.

提案した手法は元来, 共照応性解析 [6, 5] に基づく文脈処理システムのために設計したものであるが,

(1) 個々の解析木に得点を付け,

(2) その中から最も得点の大きい木を選択する

といった処理を行う様々なシステムに対して利用可能と考える.

謝辞

シャープ株式会社の, 大崎副所長, 鈴木課長, 塩谷係長に対し, この研究を行なう上での御助力に感謝いたします.

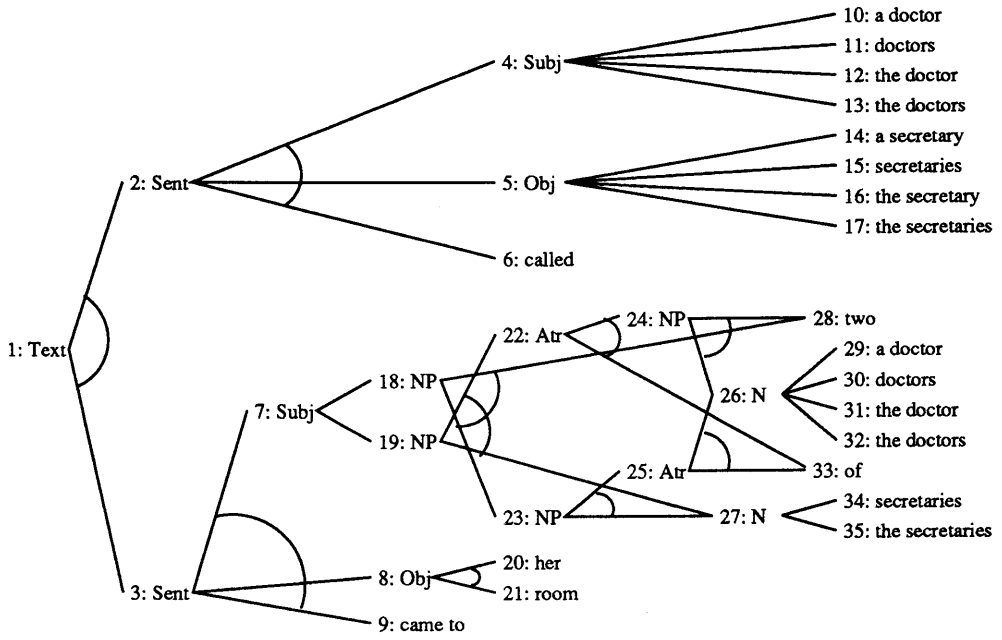

図 4: 例文から得られる AND-OR グラフ

参考文献

[1] H. Alshawi, D. M. Carter, J. van Eijck, R. C. Moore, D. B. Moran, and S. G. Pulman. Overview of the Core Language Engine. In *Proc. of the International Conference on Fifth Generation Computer Systems 1988*, pages 1108–1115, ICOT, Nov. 1988.

[2] D. Fass and Y. Wilks. Preference semantics, ill-formedness, and metaphor. *American Journal of Computational Linguistics*, 13(3-4):251–259, 1987.

[3] Jerry R. Hobbs and John Bear. Two principles of parse preference. In *Proc. of COLING-90 Vol. 3*, pages 162–167, Aug. 1990.

[4] J. Jelinek, G. Wilcock, O. Nishida, T. Yoshimi, M. J. W. Bos, N. Tamura, and H. Murakami. Japanese-to-English Project PROTRAN & TWINTRAN. In *Proc. of COLING-90 Vol. 1*, pages 50–52, Aug. 1990.

[5] Jiri Jelinek. Construct classes. *Prague Studies in Mathematical Linguistics*, 2, 1966.

[6] Jiri Jelinek. A linguistic aspect of transformation rules. *Acta Universitatis Carolinae - Philologica I, Slavica Pragensia*, VII:81–86, 1965.

[7] Martin Kay. Algorithm schemata and data structures in syntactic processing. In Barbara J. Grosz, Karen S. Jones, and Bonnie L. Webber, editors, *Readings in Natural Language Processing*, pages 35–70, Morgan Kaufmann Pub., 1986.

[8] Katashi Nagao. Dependency analyzer: a knowledge-based approach to structural disambiguation. In *Proc. of COLING-90 Vol. 2*, pages 282–287, Aug. 1990.

[9] Nils J. Nilsson. *Principles of artificial intelligence*. Tioga Publishing Co., 1980.

[10] D. Petitpierre, S. Krauwer, D. Arnold, and G. B. Varile. A model of preference. In *Proc. of third conference of the European chapter of the Association for Computational Linguistics*, pages 134–139, 1987.

[11] Lenhart K. Schubert. On parsing preferences. In *Proc. of COLING-84*, pages 247–250, 1984.

[12] Akikazu Takeuchi and Koichi Furukawa. Bounded buffer communication in Concurrent Prolog. In

Ehud Shapiro, editor, *Concurrent Prolog Collected Papers Volume 1*, chapter 18, pages 464–475, The MIT Press, 1987.

[13] N. Tamura, M.J.W. Bos, H. Murakami, O. Nishida, T. Yoshimi, and J. Jelinek. Lazy evaluation of preference on a packed shared forest without unpacking. In *Proc. of 3rd International Workshop on Natural Language Understanding and Logic Programming*, Jan 1991. (to appear).

[14] Masaru Tomita. *An efficient context-free parsing algorithm for natural languages and its applications.* PhD thesis, CMU, May 1985.

[15] Kazunori Ueda. Guarded Horn Clauses. In Ehud Shapiro, editor, *Concurrent Prolog Collected Papers Volume 1*, chapter 4, pages 140–156, The MIT Press, 1987.

[16] Y. Wilks, X. Huang, and D. Fass. Syntax, preference and right attachment. In *Proc. of IJCAI-85*, pages 779–784, Aug. 1985.

表 1: 例文に対する付加ストリーム

Node	Additional stream
18	$[5 : [\_, \_, \{18\}, \_, \_]]$
24	$[5 : [\_, \_, \{19\}, \{30, 32\}, \_],$
	$-5 : [\_, \_, \{19\}, \{29, 31\}, \_]]$
31	$[2 : [\{10, 12\}, \_, \_, \{31\}, \_],$
	$0 : [\{11, 13\}, \_, \_, \{31\}, \_]]$
32	$[2 : [\{11, 13\}, \_, \_, \{32\}, \_],$
	$0 : [\{10, 12\}, \_, \_, \{32\}, \_]]$
35	$[2 : [\_, \{15, 17\}, \_, \_, \{35\}],$
	$0 : [\_, \{14, 16\}, \_, \_, \{35\}]]$
20	$[2 : [\{10, 12\}, \_, \_, \{29, 31\}, \_],$
	$1 : [\{10, 12\}, \_, \_, \{30, 32\}, \_],$
	$1 : [\{11, 13\}, \_, \_, \{29, 31\}, \_],$
	$1 : [\{11, 13\}, \{14, 16\}, \_, \{30, 32\}, \_],$
	$0 : [\{11, 13\}, \{15, 17\}, \_, \{30, 32\}, \_]]$

表 2: 例文から得られるストリーム

Node	Generated stream
2	$[0 : [\{10\}, \{14\}, \_, \_, \_],$
	$0 : [\{10\}, \{15\}, \_, \_, \_],$
	$0 : [\{10\}, \{16\}, \_, \_, \_], \ldots]$
24	$[7 : [\{11, 13\}, \_, \{19\}, \{32\}, \_],$
	$5 : [\_, \_, \{19\}, \{30\}, \_],$
	$5 : [\{10, 12\}, \_, \{19\}, \{32\}, \_],$
	$-3 : [\{10, 12\}, \_, \{19\}, \{31\}, \_], \ldots]$
23	$[4 : [\{10, 12\}, \{15, 17\}, \{18\}, \{31\}, \{35\}],$
	$4 : [\{11, 13\}, \{15, 17\}, \{18\}, \{32\}, \{35\}],$
	$2 : [\{11, 13\}, \{15, 17\}, \{18\}, \{32\}, \{35\}],$
	$2 : [\{11, 13\}, \_, \{18\}, \{32\}, \{34\}], \ldots]$
18	$[9 : [\{10, 12\}, \{15, 17\}, \{18\}, \{31\}, \{35\}],$
	$9 : [\{11, 13\}, \{15, 17\}, \{18\}, \{32\}, \{35\}],$
	$7 : [\{10, 12\}, \{14, 16\}, \{18\}, \{32\}, \{35\}],$
	$7 : [\{10, 12\}, \_, \{18\}, \{32\}, \{34\}], \ldots]$
19	$[9 : [\{11, 13\}, \{15, 17\}, \{19\}, \{32\}, \{35\}],$
	$9 : [\{11, 13\}, \{14, 16\}, \{19\}, \{32\}, \{35\}],$
	$7 : [\{10, 12\}, \_, \{19\}, \{32\}, \{34\}],$
	$7 : [\_, \{15, 17\}, \{19\}, \{30\}, \{35\}], \ldots]$
7	$[9 : [\{10, 12\}, \{15, 17\}, \{18\}, \{31\}, \{35\}],$
	$9 : [\{11, 13\}, \{15, 17\}, \{19\}, \{32\}, \{35\}],$
	$9 : [\{11, 13\}, \{15, 17\}, \{18\}, \{32\}, \{35\}],$
	$7 : [\{11, 13\}, \{14, 16\}, \{19\}, \{32\}, \{35\}], \ldots]$
3	$[11 : [\{10, 12\}, \{15, 17\}, \{18\}, \{31\}, \{35\}],$
	$9 : [\{10, 12\}, \{14, 16\}, \{18\}, \{31\}, \{35\}],$
	$9 : [\{10, 12\}, \_, \{18\}, \{31\}, \{34\}],$
	$9 : [\{10, 12\}, \{15, 17\}, \{18\}, \{29\}, \{35\}], \ldots]$
1	$[11 : [\{10\}, \{15\}, \{18\}, \{31\}, \{35\}],$
	$11 : [\{10\}, \{17\}, \{18\}, \{31\}, \{35\}],$
	$11 : [\{12\}, \{15\}, \{18\}, \{31\}, \{35\}],$
	$11 : [\{12\}, \{17\}, \{18\}, \{31\}, \{35\}],$
	$9 : [\{10\}, \{14\}, \{18\}, \{31\}, \{35\}],$
	$9 : [\{10\}, \{14\}, \{18\}, \{31\}, \{34\}], \ldots]$