概念情報処理

国立国語研究所報告83　研究報告集6（1985）

概念情報処理

田中卓史

はじめに

国立国語研究所では電子計算機を利用して言語の研究を行ってきた。電子計算機は主として日本語のデータを並べ換え索引や語彙表を作るための，あるいは各種の集計を行うための単なる道具として使われることが多かった。近年，漢字入出力機器の開発が進み計算機の性能が向上して，これまで行ってきた日本語の字面上の操作が容易になったので，次の段階として日本語の意味内容にまで立ち入った高次の言語処理へと進みたいと考えている。しかし，日本語の意味処理はどこから始めるのが良いか，どういう方向へ進めば良いかという見通しは，これまでの日本語処理の研究方法を単に延長しても，何も得ることができない。

言語は本来，頭脳内に存在していた情報が外界に表われたものである。漠然と意味と呼んできたものは，頭脳内に形成された情報の系から生じている。自然言語の電子計算機処理において，生ずる意味上の諸問題を根本から解決するために，この情報の系を明らかにすることがどうしても必要になる。この情報の系を明らかにする試みとして，少しずつではあるが，情報処理の立場から言語理解や推論・思考の過程のモデル化を行っている。モデルは電子計算機上に実現することで，計算機の動きとしてモデルの妥当性を確認することができる。モデル化と計算機実験の繰返しは，自然科学の分野において広く行われてきた研究の方法であるが，言語研究においては比較的新しい試みであり，言語研究を飛躍的に発展させるための新しい電子計算機の利用方法である。このような方法で研究を進めることによって，モデルの到達する最終の状態は，人と類似の言語能力を持った言語ロボットである。言語ロボットは人の言語活動のメタの立場からの完全な記述により構成されるので，言語研究の究極の姿であると言えよう。

この報告は日本語の意味処理の進むべき方向を見定めるために行っている基礎的な研究に関するもので，計算機による言語理解の基礎をなす概念の構造とその計算機内での表現方法，文の意味表示としての概念依存構造の理論，

概念構造上での推論，演繹システムを用いた物語の理解などについて述べる。これらの研究はすべて概念の操作が中心的な役割を果すので，概念情報処理と呼ぶことにしている。すべてのシステムは LISP（プログラミング言語）の上に作られている。LISP は概念情報処理を行う上で大変都合のよい性質を持っており，LISP によって初めて概念情報処理が可能になったと言っても過言ではない。それで，最初に LISP 自体を我々の情報の系のモデルとして眺めることから始めよう。

1. LISP と概念構造

1.1 概念間の関係とプロパティリスト

LISP[5],[10],[11],[25]は1960年代の初めに M. I. T. の J. McCarthy らにより開発されたプログラミング言語である。LISP が他のプログラミング言語と大きく異なる点は LISP が用いる計算機内のデータ構造（リスト構造）である。一般のプログラミング言語がプログラムやデータなどの情報の格納にメモリ内の連続した領域を使用するのに対して，LISP ではメモリをセルと呼ぶ単位に分け，セル間をポインタで結んで情報を蓄えている。メモリ内の連続した領域を情報の容器として用いる普通の言語では，蓄えるべき個々の情報に対して常に容器の大きさに注意を払うことが必要になる。一方，LISP では領域の概念を消し去ることができているので，蓄えられる情報はその存在を示す名前だけがあれば良いことになる。情報の格納に際し，容器の大きさを考えずにすむことは，自然言語のような可変長データを取り扱う上で大変便利なことである。しかし LISP が概念情報処理に適している理由は，単に可変長データの取り扱いの問題だけではなくて，LISP 自体をある程度，我々の持つ情報の系のモデルとして眺めることができること，および LISP 関数を次々に定義することによって，より高次の言語を定義し，より高次のモデルを自由に組み上げることができることなどである。

今，「果物」，「りんご」，「色」，「赤」，「黄」，「形」，「丸」，「皮」，「芯」などの語を考えよう。これらの語に対応して頭脳内に情報の単位が存在すると

考え，仮に概念と呼ぶことにする。それぞれの語に対応する概念を“KUDAMONO”，“RINGO”，“IRO”，“AKA”，“KIIRO”，KATACHI”，“MARU”，“KAWA”，“SHHIN” などの LISP アトム（LISP における情報の単位）で表すことにする。語の間に成り立つ上位・下位，対象・属性，全体・部分などの意味的な関係は，それぞれの語に対応する概念間に成り立つ関係，即ち，頭脳内の情報の構造として実在する関係であると考える。これらの概念間の関係は各々の LISP アトムにプロパティリストを付加することによって，LISP の基本機能の内部で簡単に実現することができる。プロパティ・リストは LISP に準備された辞書のメカニズムである。プロパティ・リストは LISP の基本関数 PUTPROP を用いて作成する。式 (1) を評価する* と LISP アトム RINGO にはアトリビュート（属性）IRO，バリュー（属性値）AKA よりなるプロパティ・リストが加えられる。りんごの色に赤のほか，黄や緑もあることにしたければ，より大きな情報の単位であるリスト** “(AKA KIIRO MIDORI)” を用いることができる。式 (2) を評価すると「りんご」の属性「色」に対して，属性値「赤」，「黄」，「緑」を与えることができる（図1）。プロパティ・リストを検索するために，基本関数 GET が準備されている。式 (3) を評価すると値としてリスト “(AKA KIIRO MIDORI)” が得られる。

```
(PUTPROP 'RINGO 'AKA 'IRO)                    (1)
(PUTPROP 'RINGO '(AKA KIIRO MIDORI) 'IRO)     (2)
(GET 'RINGO 'IRO)                             (3)
```

---

* LISP では LISP 関数が他の言語のプログラムに相当する。関数の評価（値を求める操作）がプログラムの実行に相当する。LISP 関数は一般に関数名と引数の並びを括弧でくくったリストの形をしている。LISP 関数は評価を受ける際に副作用を起こすものがある。関数 PUTPROP は副作用として，アトムにプロパティ・リストを付加する。

** リストはアトムやリストの並びを括弧でくくったもので，式 (1) や式 (2) などもまたリストである。

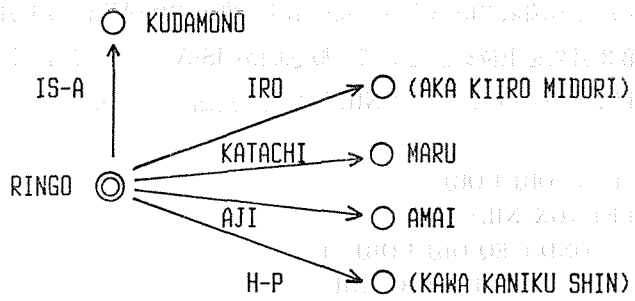


図1　プロパティリストにより表された概念相互の関係

1.2　上位・下位概念の関係

「果物」や「りんご」のような「もの」の概念は上位・下位概念の関係によって階層構造をなす。この構造はプロパティ・リスト上に“IS-A”(上位概念)および“EXTENT”(外延)をアトリビュートして用いることにより，二通りに蓄えることができる(図2)。概念情報処理においては，ある概念(例えば「ポチ」)が他の概念(例えば「動物」)の下位概念であるかどうか調べることが常に必要になる。今，上位概念「動物」の方から外延を調べて行くと，「ポチ」が「魚」であるか，「鳥」であるか，「猫」であるか，……調べることになって極めて能率が悪くなる。一方下位概念「ポチ」の方から“IS-A”のリンクをたどると「ポチ」→「犬」→「哺乳類」→「動物」と数回の試行で目標の上位概念「動物」に到達することになる。式(4)に“IS-A”

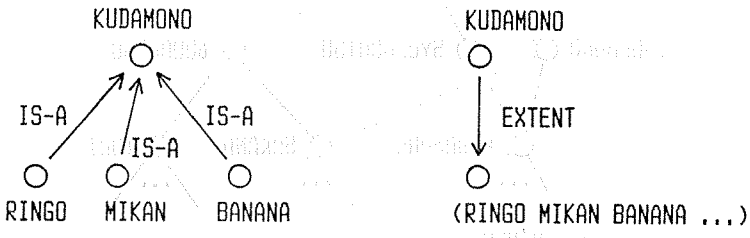


図2　二通りの上位・下位概念の表現

リンクをたどる関数“IS-A”の定義を示す。関数“IS-A”は第1引数を下位概念，第2引数を上位概念として，概念間がIS-Aリンクで結ばれているときに“T”，そうでないとき“NIL”を返す述語として働く。

```
(DE IS-A (OBJ UOBJ)
  (LET (GX NIL)
    (COND ((EQ OBJ UOBJ) T)
          ((NULL OBJ) NIL)
          (T (SETQ GX (GET OBJ 'IS-A))
             (COND ((ATOM GX) (IS-A GX UOBJ))
                   (T (MAPOR '(LAMBDA (X) (IS-A X UOBJ))
                             GX]                    (4)
```

上位・下位概念の関係による階層構造が木構造をなしておれば，各概念に対して“IS-A”の値は一意的に定まるが，図3に示すように部分的に木構造をくずして「果物」の上位概念として「食べ物」も加えたいようなことが起る。これは単に概念のアストリビュート“IS-A”の値をリスト“(SHOKUBUTSU TABEMONO)”の形も許すことにすればよい。一方，下位概念を示す“EXTENT”（外延）のリストにも，「食べ物」を追加することが必要になるが，外延を概念の分類という見方からすると具合が悪くなる。今，概念「もの」が下位概念として「動物」，「植物」，「鉱物」のリストを

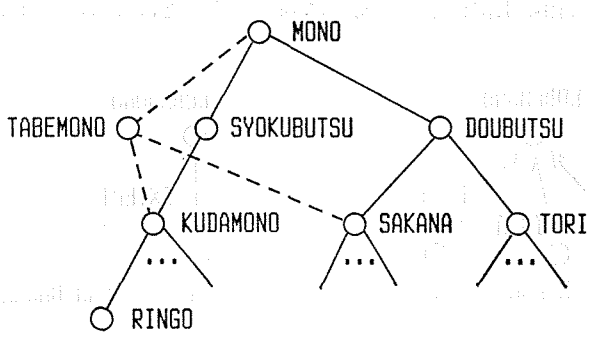


図3　異なった見方による上位・下位概念

持っている場合に，異なった見方から得られる下位概念「食べ物」を同列に加えると，それぞれの外延が排他的でなくなる。そこで，次のリスト (5) を "EXTENT" の値として用いると，下位概念のグループ化が行える。

((DOUBUTSU SYOKUBUTSU KOUBUTSU) (TABEMONO NOMIMONO...) ...) (5)

1.3 属性・属性値，属性の継承

概念「りんご」に対して属性「色」,「形」,「味」および属性値「赤」,「丸」,「甘い」などの概念相互の関係をプロパティ・リストを用いて与えた。プロパティ・リストを検索する関数 GET は対象となる概念とその属性を引数として取るので，対象となる概念があらかじめどのような属性を取り得るか知っておくことが必要になる。そこで「りんご」の持つ属性「色」,「形」,「味」などを一つのリストにまとめ，プロパティ・リストの一つの要素に加えておくことが考えられる。しかし，これらは果物の外延のすべてが取り得る属性と考えられるので，個々の果物にではなく概念「果物」のプロパティリスト

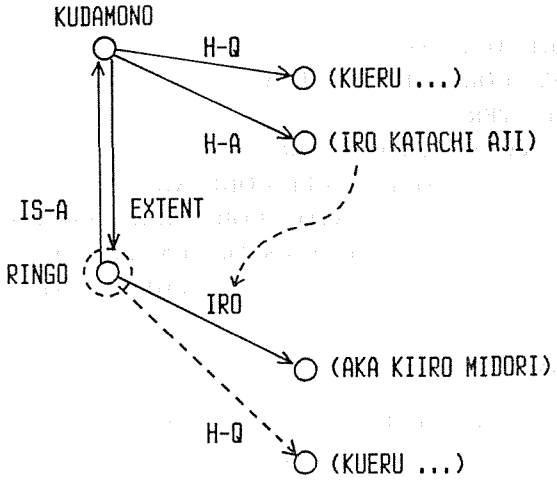


図 4　属性の統合と属性の継承

に特別な名前のアトリビュート "H-A" (has-attribute) を用いて加えることにする(図4)。

概念「果物」に対して「食える」や,「鳥」に対して「飛ぶ」といった「こと」の概念も「もの」の概念に関する一つの属性と考えることができる。しかし,「色」や「形」のような属性と異なって属性値をとらない。そこで,これらの属性はリストにまとめて,特別の名前 "H-Q" (has-quality) をアトリビュートとして与えプロパティ・リストに加える。

一般に上位概念の持つ属性は下位概念も継承することができる。即ち,概念「果物」の属性 H-Q の値「食える」は下位概念の「りんご」も取ることができる。しかし,概念「鳥」のもつ属性「飛ぶ」は例外的に下位概念の「だちょう」や「ペンギン」に継承されると具合が悪くなる。そこで,上位概念の属性と下位概念の属性が矛盾を起す場合には,下位概念に付加された属性を優先するという規則が必要になる。

式 (6) は関数 "HGET" の定義を示す。HGET は指定された概念とそのすべての上位概念を検索し,継承されるものも含めて,指定されたアトリビュートの値を導く。

```
(DE HGET (OBJ ATR)
    (LET (UOBJ (GET OBJ 'IS-A))
      (FILTER
        (CONS (GET OBJ ATR)
              (COND ((NULL UOBJ) NIL)
                    ((ATOM UOBJ) (HGET UOBJ ATR))
                    (T (MAPCAR '(LAMBDA (X)
                                  (HGET X ATR)) UOBJ)]  (6)
```

1.4 全体・部分の関係

概念「果物」に対して,果物の「皮」,「果肉」,「芯」などの概念や「自動車」に対する「エンジン」,「ボディー」,「タイヤ」などの概念は全体と部分の関係にある。そこで,部分を構成する概念を一つのリストにまとめ,アト

リビュートとして特別な名前“H-P”(has-part)を与えて，全体概念のプロパティリストに加える。概念「自動車」に関して，H-Pで導かれる概念（エンジン，ボディーなど）は，H-Aで導かれる概念（色，形など）と同様に外延に対するアトリビュートとしても使える（図5)。例えば「自動車」の外延「マツダ・ファミリア」には「エンジン」，「ボディー」，「タイヤ」などをアトリビュートとして使える。さらに，アトリビュートの値の中にアトリビュート・バリューの組の繰返しを許すと，概念関係は図5に示されるような多

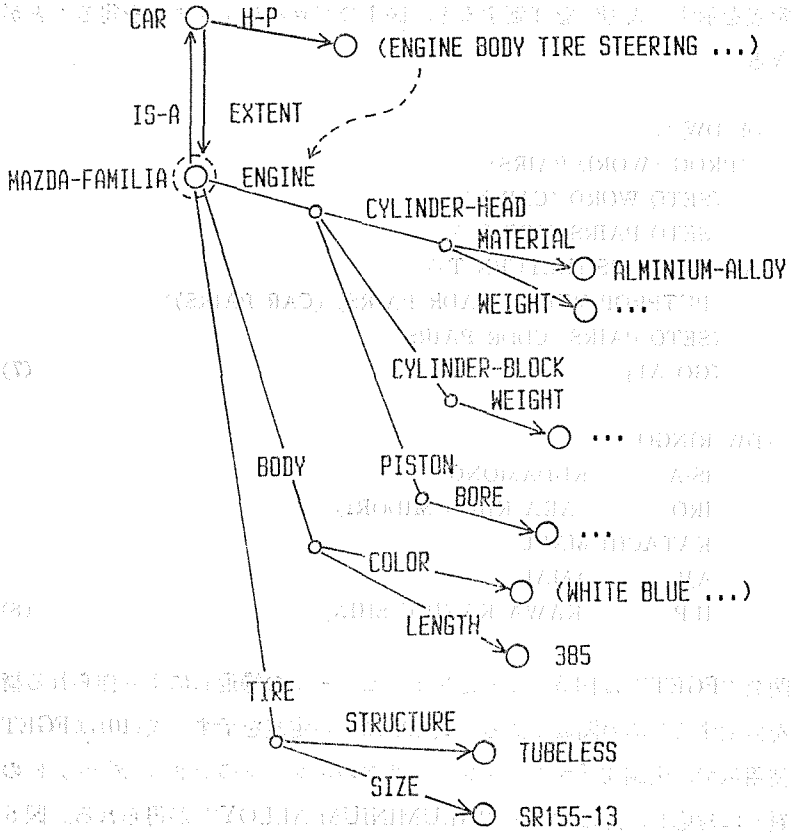


図5 全体・部分の関係

重の全体・部分構造となる。これは「マツダファミリアのエンジンのシリンダヘッドの材質」のような全体・部分の繰返しにより作られる名詞句に対応して，LISP 内に情報の構造が作られたことを示す。全体・部分の繰返しによりできる情報の構造は，具象的なものに限らずにより抽象的な関係にも応用することができる。全く同じ取り扱いにより「象の足の裏の面積」，「日本の首相の出身県の県庁所在地の人口」などの名詞句に対応して，LISP 内に情報の構造を作ることができる。

関数 "DW" は多くのプロパティを一度に登録できる関数である。式 (7) に定義を示す。式 (8) を評価すると，図1のプロパティリストを得ることができる。

```
(DF DW (X)
  (PROG (WORD PAIRS)
    (SETQ WORD (CAR X))
    (SETQ PAIRS (CDR X))
 A1 (OR PAIRS (RETURN T))
    (PUTPROP WORD (CADR PAIRS) (CAR PAIRS))
    (SETQ PAIRS (CDDR PAIRS))
    (GO A1]                                          (7)
```

```
(DW RINGO
    IS-A        KUDAMONO
    IRO         (AKA KIIRO MIDORI)
    KATACHI MARU
    AJI         AMAI
    H-P         (KAWA KANIKU SHIN))                  (8)
```

関数 "FGET" は図5のようなアトリビュートの繰返しにより作られる構造を検索するための関数である。式 (9) にその定義を示す。式(10)はFGETの使用例で，名詞句「マツダ・ファミリアのエンジンのシリンダヘッドの材質」に対応し，評価すると "ALUMINIUM-ALLOY" が得られる。図5のような構造は関数 "DW" を用いて作り出すこともできるが，次の関数

"FPUT" を用いると，名詞句に対応する形で全体・部分の繰返し構造の作成・更新を行うことができる。関数 "FPUT" は式(11)，(12)，(13)により定義される。式(14)を評価すると「マツダ・ファミリアのボディーの幅」が154(cm) であるという情報が，図5に新しい構造として追加される。

```
(DF FGET (X)
  (PROG (OB VAL ATS)
    (SETQ OB (EVAL (CAR X)))
    (SETQ VAL (GET OB '-FM-))
    (SETQ ATS (MAPCAR 'EVAL (CDR X)))
 A1 (OR ATS (RETURN VAL))
    (SETQ VAL (CDR (ASSOC (POP ATS) VAL)))
    (GO A1)))                                          (9)

(FGET 'MAZDA-FAMILIA 'ENGINE 'SYLINDER-HEAD 'MATERIAL)
                                                       (10)

(DF FPUT (X)
    (PUTPROP (EVAL (CAR X))
             (HMARGE (GET (EVAL (CAR X)) '-FM-)
                     (MAPCAR 'EVAL (CDR X)))
             '-FM-))                                   (11)

(DE HMERGE (X Y)
  (COND ((NULL X) (HLIST Y))
        ((NOT (EQ (CAR Y) (CAAR X)))
         (CONS (CAR X) (HMERGE (CDR X) Y)))
        (T (COND ((NULL (CDDR Y)) (CDR X)))
                 ((ATOM (CADAR X))
                  (APPEND (HLIST Y) (CDR X)))
                 (T (CONS (CONS (CAR Y)
                                (HMERGE (CDAR X)
                                        (CDR Y)))
                          (CDR X)))))))                (12)
```

```
(DE HLIST (X)
  (COND ((NULL (CDDR X)) (LIST (CONS (CAR X) (CADR X))))
        (T (LIST (CONS (CAR X) (HLIST (CDR X))))))))      (13)

(FPUT 'MAZDA-FAMILIA 'BODY 'WIDTH  154)                   (14)
```

2. 概念依存構造の理論

2.1 概念のカテゴリ

前章では「もの」の概念に着目して，概念間に成り立つ上位・下位概念の関係，属性・属性値の関係，部分・全体の関係など，概念が形造る静的な構造を見てきた。一方，文や文章が表わす意味内容のように，より動的な情報はどのような概念の構造をなすのであろうか。Roger Schank[33]は人の行為に関する記述を分析して，文の意味内容を格文法よりもさらに深層の構造で表現する概念依存構造の理論を発展させた。概念依存構造の理論では次の概念カテゴリーを用いる。

PP (Picture Producer)：具象的な「もの」の概念
ACT (Action)：11種の基本的な「こと」の概念
LOC (Location)：「ところ」の概念
T (Time)：「とき」の概念
AA (Action Aider)：ACT をモディファイする概念，ほぼ副詞に対応する
PA (Attribute of Object)：「もの」の状態と状態の値，属性と属性値の概念を含む

概念カテゴリ ACT は基本的な「こと」の概念を表すもので，次の11個が設定され，四つにグループ化されている。

（1） 動作に関する ACT

PROPEL： 物に力を加えること
MOVE： 体の一部を動かすこと

INGEST： 体の内側に食物など取り込むこと

EXPEL： 体の外側に排出すること

GRASP： 物をつかむこと

（2） 動作の結果に着目した ACT

PTRANS： 物の存在位置が変わること

ATRANS： 所有権など，物の抽象的な位置関係が変わること

（3） 精神的な行為に関する ACT

MTRANS： 情報の移動が起ること

MBUILD： 情報の生成が起ること

（4） MTRAS の道具格となる ACT

SPEAK： 音を出すこと

ATTEND： 視覚，聴覚などの感覚器管を集中させること

概念カテゴリＰＡは，「もの」の状態・状態値（属性・属性値を含む）を示し，STATE (VALUE) の形をしたものと STATE だけのものがある。

（1） 人の状態に関し，値は相対的な尺度が与えられるもの。

HEALTH （−10〜10）	CONSCIOUSNESS （0〜10）
FEAR （−10〜0）	HUNGER （−10〜0）
ANGER （−10〜0）	DISGUST （−10〜0）
MENTAL-STATE （−10〜10）	SURPRISE （0〜10）
PHYSICAL-STATE（−10〜10）	

（2） 絶対的な尺度をもつもの

LENGTH	MASS
COLOR	SPEED
LIGHT-INTENSITY	

（3） 対象間の関係を示し，値はとらないもの。

CONTROL	PROXIMITY
POSSESION	LOCATION
OWNERSHIP	PHYSICAL-CONTACT

CONTAIN

概念構造上の時間Tと他の補助的様相として，次のものが設定されている。

p：	past	k：	continuous
f：	future	∞：	time less
ts：	start of transition	nil：	present
ft：	end of transition	／：	negation
c：	conditional	？：	interogative

2.2 概念依存構造のシンタクス

概念依存構造の理論では，入力文の意味表現として Conceptualization が作られる。Conceptualization は，次の概念依存構造のシンタクスに基づいて作られる。なお，シンタクス中の矢印の方向は概念の依存する方向を示している。

(1) PP⟺ACT　動作主 PP がアクション ACT を行う。

(2) PP⟺PA　PP に状態（属性）が付加される。

(3) ACT ←O— PP　アクション ACT の作用する対象を示す。

(4) ACT ←D— LOC, LOC　アクション ACT の作用する方向を示す。

(5) ACT ←R— PP, PP　アクション ACT の供与者受領者を示す。

(6) ACT ←‖↕‖　アクション ACT が対象格として別の Conceptualization を取る。

(7) ACT ←I— ‖↕‖　アクション ACT が道具格として別の Conceptualization をとる。

(8) PP ↕ ⟷　PP ↕ ⟺　具象的「もの」PP が他の Conceptualization により詳しく述べられる。

(9) T ⟨⟹⟩ Conceptualization に時間が指定される。

(10) LOC ⇓ ⟨⟹⟩ Conceptualization に場所が指定される。

(11) ⟨⟹⟩ PP ⟨⟹ r → ある Conceptualization に結果として，PP の状態変化が依存することを示す。

(12) ⟨⟹⟩ R ⟨⟹⟩ ある Conceptualization に他の Conceptualization が理由として依存することを示す。

(13) ⟨⟹⟩ E ⟨⟹⟩ ⟨⟹ → E ⟨⟹⟩ ある状態，または状態変化の Conceptualization に，それを可能にした Conceptualization が依存する。

(14) PP⟨⟹⟩PP 一つの PP が他の PP に等しいか，インスタンスになっていることを示す。

(15) ACT ↑ AA アクション ACT が AA でモディファイされることを示す。

2.3 概念依存構造の例

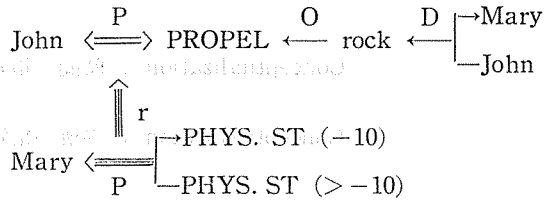
入力文に対してどのような Conceptualization が得られるかを，幾つかの例で示そう。

(1) John gave the book to Mary.

John ⟨P⟹ ATRANS ⟵O OWNERSHIP : book ⟵R ┌→Mary / └—John

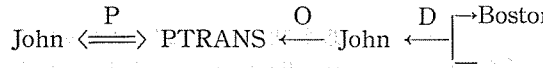
"give" から抽象的なもの「本の所有権」の移動を表す，プリミティブ・アクション ATRANS の構造が作られる。

(2) John killed Mary by throwing a rock at her.

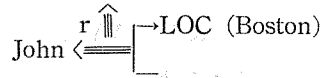


"kill" は人の身体の状態 PHYS. ST の変化を，"throw" は物に力を加えるアクション PROPEL の構造を作り出す。全体として「John が石に力を加えへ Mary と推進させたことが，結果として Mary の身体の状態を −10（死）へと変化させた」という構造になっている。

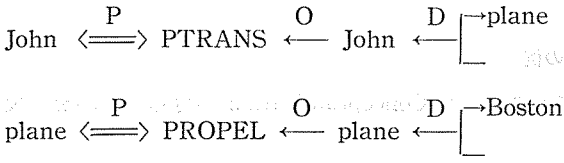
（3） John went to Boston.



"go" は物理的な物の移動の概念 PTRANS の構造を作り出す。"John arrived in Boston." も "John flew to Boston." も "The plane took John to Boston." も基本的に同じ PTRANS の構造となり，結果格として，



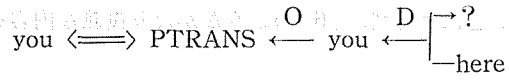
あるいは道具格として



などの構造が付加される。Conceptualization により，文法的あるいは語彙的に全く異なる文でも，同一のあるいは類似の意味内容を持つ文は同一のあるいは類似の概念依存構造へと変換される。一方，文法的，語彙的に似かよった文でも，意味内容的に異なる文は異なった構造が作られることになる。また Conceptualization では "fly" のように動詞の中に暗黙に埋め込まれている情報 "airplane" も陽な形に取り出して表現することになる。このよ

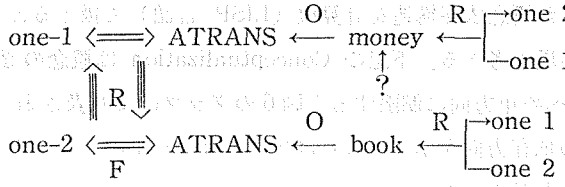
うに入力文の表わす意味内容を限られた ACT や STATE の組合わせとして表現することにより，概念構造上の推論規則として，推論や思考過程の形式化を行える可能性がでてくる。

(4) Where are you going?



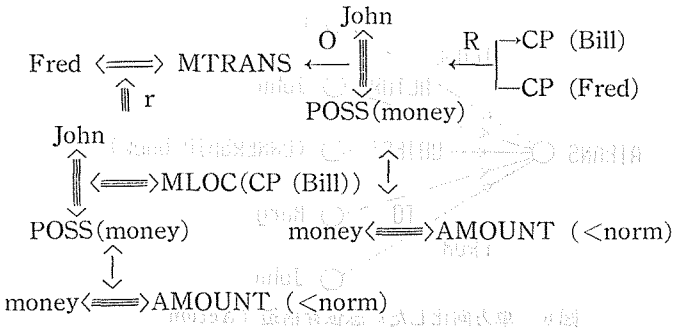
概念の単位に "?" を許すことにすれば，疑問文に対応する構造が作られる。

(5) How much does the book cost?



"how much" は売買の構造をを作り出す Key-word として働いている。

(6) Bill became aware of John's poverty when Fred told him about it.



Schank は概念の存在する場所（メモリ）として，次の三つを設定している。

CP： Conceptual Processor

IM： Intermediate Memory

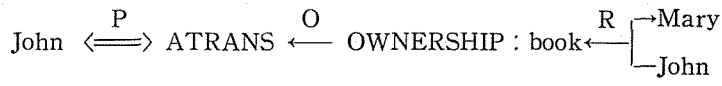
LTM：Long Term Memory

CP は現在の意識下にある Conceptualization が蓄えられる。LTM は個人のもつすべての概念が蓄えられている。IM は CP と LTM の中間的なメモリを想定している。CP, IM, LTM 自体を対象化して Conceptualization の表現の中に取り込むことによって，(6)のようなかなり複雑な内容を表わすことができる。

3. 概念依存構造を生成するパーサ

3.1 リストによる概念依存構造の表現

グラフで表された概念依存構造を計算機（LISP 言語）で扱えるようにするためのリスト表現を考える。下記の Conceptualization は概念の依存方向を ATRANS への単方向に制限すると図6のグラフにより表される。矢印の方向は概念の依存方向を示す。これはアトリビュートバリューのペアリスト(15)で表すことができる。



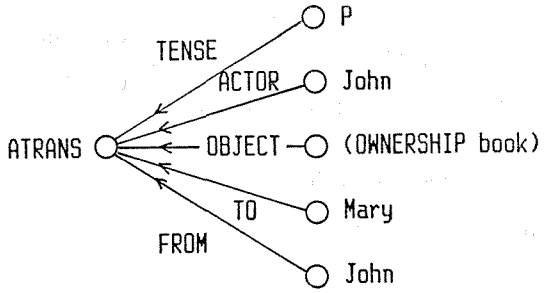


図6　単方向化した概念依存構造（Action）

```
(ATRANS (TENSE p)
        (ACTOR John)
        (OBJECT (OWNERSHIP book))
        (TO Mary)
```

(FROM John)) (15)

John が Mary に本をやった結果，Mary の MENTAL-STATE が happy になったとすれば，左下の構造が付加される。これを多少変形して，次のグラフに変える（図7）。図7はリスト（16）で表され，式（15）の ATRANS のペアリストの一部に加わることになる。

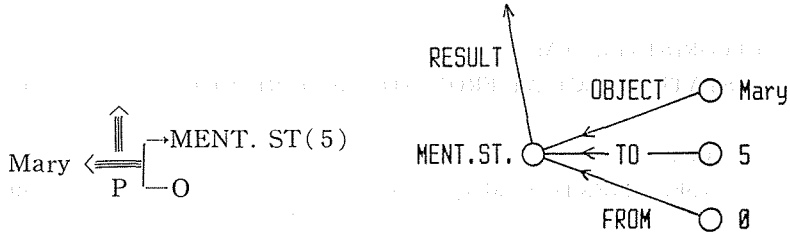


図7　単方向化した概念依存構造（State）

```
(RESULT (MENT-ST (OBJECT Mary)
                 (TO      5)
                 (FROM    0)
                 (TENSE   P)))                    (16)
```

UCILISP (UCI)[25] および TLISP (YALE) ではこのような形の情報をレコード・タイプ付きのデータとして，より便利に扱うことができる。プリミティブ・アクション ACT に着目すると Conceptualization はいずれも ACTion, ACTOR, OBJECT, FROM, TO, INST, CAUSE, RESULT を要素として持つので，このようなデータの組をアクション・フレームと呼び，AFM で表すことにする。次の関数（17）を評価すると AFM という名のレコード・タイプが定義される。同様に状態や状態変化の Conceptualization を表すステート・フレーム SFM と定義する（18）。一度レコード・タイプを定義しておくと，式(19)および(20)のような形で概念構造を扱うことがで

きるようになる。式(19)を評価すると，Mary の精神状態が happy に変化したことを表す構造が CD2 という名の LISP アトムにセットされる。式(20)を評価すると John が Mary に本をやったこと，およびその結果として CD2 の内容が導かれることを示す構造が CD1 という LISP アトムにセットされる。

```
(RECORD-TYPE AFM
  (ACT ACTOR OBJECT FROM TO INST CAUSE RESULT))     (17)

(RECORD-TYPE SFM
  (STATE OBJECT AT FROM TO CAUSE RESULT))           (18)

(:= CD2
    (SFM 'MENT-ST 'Mary ( ) 0 5 ( ) ( ))            (19)

(:= CD1
    (AFM 'ATRANS 'John '(OWNERSHIP book)
         'John 'Mary ( ) ( ) CD2))                  (20)
```

レコード・タイプを定義すると，次のようなタイプ付きデータを操作する便利な関数が自動的に定義される。いずれも左側の式を評価すると，右側の値が得られる。さらに便利なことには，式(21)を評価すると，すでにある構造 CD1 の OBJECT の項を book から ball に変えることができる。

```
(ACT : AFM     CD1)   −>   ATRANS
(ACTOR : AFM   CD1)   −>   John
(OBJECT : AFM  CD1)   −>   book
(RESULT : AFM  CD1)   −>   [content of CD2]
(STATE : SFM   CD2)   −>   MENT-ST
(OBJECT : SFM  CD2)   −>   Mary
(TO : SFM      CD2)   −>   5
...
```

(:= (OBJECT : AFM CD1) 'ball) (21)

3.2 Tiny-ELI

従来の文法パーサ (Syntactic Parser) が入力文から文法情報を利用して構文解析木を作り出すのに対して，概念パーサ (Conceptual Parser) は入力から概念依存構造を作り出す。文法パーサでは構文上のあいさつを取り除くために語の意味情報を補助的に用いることが多かったが，概念パーサでは文法的な知識と意味内容に関する知識との間で取扱い上の差が少なく，むしろ文法の方が補助的な役割を演じている。

概念パーサの主要なメカニズムは Expectation と呼ばれている。パーサは語を読むごとに，述べられる意味内容と次に来る語の予測を行いながら概念依存構造を作り上げてゆく。過程は決定論的に行われ文法パーサが行うような可能な構造の枚挙は行わない。ここで述べる概念パーサ Tiny-ELI は Christopher Riesbeck による McELI[36] を参考にして作成したものである。ELI (English Language Interpreter) はインタプリタの名前が示すようにプログラム自身には概念依存構造を作り上げるための実質的な情報をほとんど含んでいない。パージングに必要な情報はすべて各々の語に付加されたパケットと呼ばれるリストの中に含まれている。このパケットはリクエストと呼ばれる小さなプログラムの集合よりなっている。Tiny-ELI は入力文を一語ずつ読んで，その語に付加されたパケットをスタックへ蓄え，スタック中のリクエストを実行することによって Expectation を実現し，概念依存構造を作り上げる。

パケットを構成する一つのリクエストは次の TEST, ASSIGN, NEXT-PACKET のトリプルよりなっている(22)。パケット内の一つのリクエストはその語の特定の出現環境における意味を定めている。どのリクエストが適用されるかについては，各々のリクエストの TEST の項が定める。TEST の項が満足されると ASSIGN の項が現時点で決定できる変数（文法的主語，目的語や主概念構造の Action, Actor, など）の割当てを行い， NEXT-

PACKET の項が次に来るべき語とそれにより作られる概念構造の予測を行う。

```
((TEST expression)
 (ASSIGN variable-1 expression-1 variable-2 expression-2 ...)
 (NEXT-PACKET request-1 request-2 ...))                    (22)
```

式(23), (24), (25), (26) に Tiny-ELI の LISP プログラムを示す。メインプログラムに相当する関数 PARSE は1個の入力文を引数として取ると，各語に対して関数 PROCESS-WORD を作用させる。PROCESS-WORD は処理の対象としている語のパケット（リクエストの集合）をスタック *STACK* へ蓄える。スタックへ蓄える際にパケットの境界が取り除かれてリクエストが単位となる。即ち，スタックはリクエストの集合となる。次にスタック内のリクエストは，最も新しく蓄えられたリクエストを最優先として順に古い方へ，TEST の項の条件を満たすものが発火してプロダクションの動作を起す。

関数 EVAL-PACKET は各リクエストの TEST の項を評価して，条件を満たしておれば ASSIGN の項を実行し，各変数に値をセットして，そのリクエストをスタックから取り除く。NEXT-PACKET の項を持っておれば，その項に書かれた潜在的リクエストを評価の対象となるアクティブなリクエストへと昇格させる。リクエストで TEST の項を持たないものはすでに条件を満たしていると解釈してただちに実行する。

```
\^ Tiny-ELI

(DE PARSE (SENTENCE)
  (PROG( )
    (SETQ *STACK* NIL)(SETQ *SYN-OBJ* NIL)
    (SETQ *SYN-SBJ* NIL)
    (SETQ *PART-OF-SPEECH* NIL) (SETQ *PREDICATES* NIL)
    (SETQ *SYN-TO* NIL)(SETQ *SYN-FROM* NIL)
```

```
    (SETQ *SYNWITH* NIL)
    (MSG "〈INPUT〉: " SENTENCE)
    (SETQ SENTENCE (CONS '*START* SENTENCE))
    (MAPC 'PROCESS-WORD SENTENCE)
    (MSG T T "〈FINAL-EXPRESSION〉: ")
    (SPRINT *CONCEPT*)
    (RETURN T)))                                          (23)

(DE PROCESS-WORD (*WORD*)†
  (APPENDPUSH *STACK* (GET *WORD* 'WD-DEF))
  (RUN-STACK))                                            (24)

(DE RUN-STACK ( )††
  (AND *DBG* (MSG T T "〈PROCESSING〉: " *WORD*))
  (SETQ *STACK* (MAPPENDCAR 'EVAL-PACKET *STACK*)))       (25)

(DE EVAL-PACKET (PACKET)†††
  (MAPOR '(LAMBDA (X)
            (SELECTQ (CAR X)
                     (TEST (AND (NULL (EVAL (CADR X)))
                                (LIST PACKET)))
                     (ASSIGN (MSG T T "〈TRG-PAC〉: ")
                             (SPRINT PACKET)
                             (MAPC 'EVAL (CDR X)) NIL)
                     (NEXT-PACKET (CDR X))
                     NIL))
          PACKET))                                        (26)
```

3.3 Tiny-ELIの動作

Tiny-ELI の本体は語の定義を参照し，語に付加されたパケットをスタックへ蓄え，スタック内のリクエストを評価する動作を定めているだけで，実

---

† APPENDPUSH：スタックへの PUSH を CONS の代りに APPEND でやる。
†† MAPPENDCAR：MAPCAR の APPEND 版。
†††MAPOR：リストの要素に関数を APPLY して OR をとる。

際のパージングに必要な情報はすべてリクエストの中に含まれる。Tiny-ELI は最初これから入力される文の性質をおおまかに定めるダミーの語 *START* を入力文の先頭に付加して処理を開始する。*START* のパケットは式(27)に示すようにこれから入力される文が平叙文としての文法的な性格を持つことを予測させるための情報である。即ち，最初に名詞句が来て，次に動詞が来，さらに名詞句が続くであろうという予測を示している。

```
(DW *START*
    WD-DEF (((TEST (EQ *PART-OF-SPEECH* 'NOUN-PHRASE))
             (ASSIGN (:= *SYN-SBJ* *CD-FORM*))
             (NEXT-PACKET
              ((TEST (EQ *PART-OF-SPEECH* 'VERB))
               (NEXT-PACKET
                ((TEST(EQ *PART-OF-SPEECH* 'NOUN-PHRASE))
                 (ASSIGN (:= *SYN-OBJ* *CD-FORM*))]      (27)
```

Tiny-ELI の入力文として，9個の語（John, Mary, Boston, medicine, gun, plane, the, a, took）の組合せから構成される文を考えよう。英文としての自然さは別として，次のような文が得られる。

S 1 : (JOHN TOOK A PLANE TO BOSTON *PD*)
S 2 : (JOHN TOOK THE PLANE WITH MARY *PD*)
S 3 : (JOHN TOOK THE PLANE WITH A GUN *PD*)
S 4 : (JOHN TOOK A GUN *PD*)
S 5 : (JOHN TOOK THE GUN TO BOSTON *PD*)
S 6 : (JOHN TOOK MARY TO BOSTON *PD*)
S 7 : (JOHN TOOK MARY WITH THE GUN *PD*)
S 8 : (JOHN TOOK MEDICINE *PD*)
S 9 : (JOHN TOOK MEDICINE TO BOSTON *PD*)
S10 : (JOHN TOOK MEDICINE TO MARY *PD*)
S11 : (JOHN TOOK BOSTON *PD*)
S12 : (THE PLANE TOOK JOHN TO BOSTON *PD*)
S13 : (THE PLANE TOOK THE GUN TO BOSTON *PD*)

いずれも"took"を動詞として用いており，文法的には類似の構造をしているが，Conceptualization はかなり異なったものとなる。道具格や結果格などを除いた Conceptualization のトップレベルを考えると次のようになる。

C1 : John ⟺ PTRANS $\overset{O}{\leftarrow}$ John—[→(plane(To Boston))]

C2 : John ⟺ PTRANS $\overset{O}{\leftarrow}$ (& John Mary)—[→plane]

C3 : John⟺ATRANS$\overset{O}{\leftarrow}$(OWNERSHIP plane)—[→John]

C4 : John ⟺ GRASP $\overset{O}{\leftarrow}$ gun

C5 : John ⟺ PTRANS $\overset{O}{\leftarrow}$ gun—[→Boston]

C6 : John ⟺PTRANS $\overset{O}{\leftarrow}$ Mary—[→Boston]

C7 : John ⟺ DO

C8 : John⟺INGEST$\overset{O}{\leftarrow}$*medicine* —[→(INSIDE-OF John)]

C9 : John ⟺ PTRANS $\overset{O}{\leftarrow}$ medicine—[→Boston]

C10 : John ⟺ PTRANS $\overset{O}{\leftarrow}$ *medicine* —[→Mary]

C11 : John ⟺ ATRANS $\overset{O}{\leftarrow}$ Boston

C12 : plane ⟺ PTRANS $\overset{O}{\leftarrow}$ John—[→Boston]

C13 : plane ⟺ PTRANS $\overset{O}{\leftarrow}$ gun—[→Boston]

入力文 S1～S13 の Conceptualization を作り出す上で，“took”に付加されるパケットが主要な役割を果す。次の“took”の定義において，第3行目の ASSIGN は TEST の項を持たないのでただちに実行され，現在の品詞が動詞であることを示す。第4行の TEST は文法的な文の主語 *SYN-SBJ* にセットされた語が“PERSON”の下位概念となるときに条件が満たされて，6行目以後の NEXT-PACKET がアクティブなリクエストとなる。6行目以後の TEST の項は文法的対象 *SYN-OBJ* が設定され，しかもそれが VEHICLE の下位概念となっているときに条件が満たされる。入力文が S1 の場合，このリクエストが発火して，主概念 *CONCEPT* に PTRANS のフレームがセットされる。次いで NEXT-PACKET により TO や WITH などの前置詞句が来ることを予測するリクエストがアクティブとなる。

```
(DW TOOK
  WD-DEF
    (((ASSIGN ( : = *PART-OF-SPEECH* 'VERB)))
     ((TEST (AND *SYN-SBJ* (IS-A *SYN-SBJ* 'VEHICLE)))
      (ASSIGN
       ( : = *CONCEPT* (FRAME 'PTRANS *SYN-SBJ*( )( )( )( )
        ( ))))
       (NEXT-PACKET
        ((TEST *SYN-OBJ*)
         (ASSIGN ( : = (OBJECT : FRAME *CONCEPT*)
                       *SYN-OBJ*)))
        ((TEST *SYN-TO*)
         (ASSIGN ( : = (TO : FRAME *CONCEPT*) *SYN-TO*)))
        ((TEST *SYN-FROM*)
         (ASSIGN ( : = (FROM : FRAME *CONCEPT*)
                       *SYN-FROM*)))))
      ((TEST (AND *SYN-SBJ* (IS-A *SYN-SBJ* 'PERSON)))
       (NEXT-PACKET
        ((TEST (AND *SYN-OBJ* (IS-A *SYN-OBJ* 'VEHICLE)))
```

```
(ASSIGN
 ( : = *CONCEPT*
      (FRAME 'PTRANS *SYN-SBJ* *SYN-SBJ* *SYN-OBJ*
        ( )( )( ))))
(NEXT-PACKET
 ((TEST *SYN-TO*)
  (ASSIGN
   ( : = (TO : FRAME *CONCEPT*)
        (C-MODIFY (TO : FRAME *CONCEPT*) 'TO SYN-
          TO*))))
((TEST (AND * SYN-WITH* (IS-A *SYN-WITH* 'PERSON)))
 (ASSIGN
  ( : = (RESULT : FRAME *CONCEPT*)
       (FRAME 'PTRANS *SYN-SBJ*  *SYN-WITH*
                 (TO : FRAME *CONCEPT*)( )( )( )))))
   ...
```

4. 推論と知識の形式化

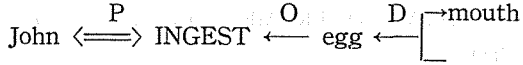
4.1 プリミティブ・アクションに関する推論

「昨日，John が Mary をなぐった。」というような文を聞いた時に，「Mary はなにか John を怒らせるようなことをやったのだろう。」とか「Mary はけがをしたのではないか。」といった事柄を推論することができる。このような推論は文章の理解のメカニズムを考える上で，行間の情報を補い，文と文との間の意味的な関係を与えるために重要になる。このような推論を可能にするには「人1が人2をなぐる。」ときは「人1が人2を怒らせた。」ことが原因になるとか「人2はけがをする。」ことが多いと言った知識を形式化することが必要になる。ここでは Conceptualization により文の意味内容が表されるので，これらの原因あるいは結果を求める推論は一つの Conceptualization から他の Conceptualization を導く操作となる。Conceptualization では，11種のプリミティブ・アクション ACT が主要な役割を演ずるので，これら ACT に関してどのような推論が行えるかを考え

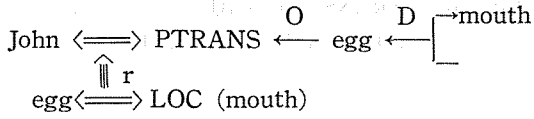
よう。

（1） PHYSICAL-ACT に関する推論

INGEST, EXPEL, GRASP, PROPEL, MOVE などの PYSICAL-ACT に対して PTRANS が常に関係しており，PTRANS の作る構造を推論できる。例えば，「John が卵を食べた」を示す次の Conceptualization に対して，

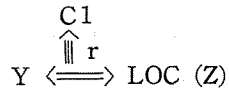


「John が口の中に卵を移動させた。」という事実が行われたことを推論できる。

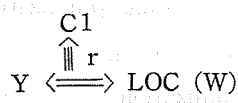


一方，PTRANS の作る Conceptualization C1 に対して，次のように物の存在する場所の変化 (i)，(ii) やアクターの意図 (iii)，(iv) などを推論することができる。

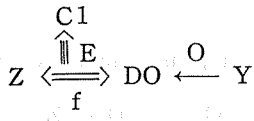
（i） C1 の結果としてYはZの所に存在する。



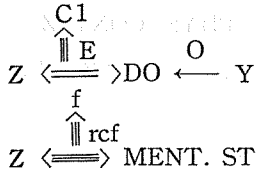
（ii） C1 の結果Yはもはや Wの所にない。



(iii) Zが人であり，Zが C1 を望んでたなら，またはZがXに等しければ，Zは多分，人がYを使って普通にやるようなことをやるであろう。



(iv) (iii)に関し人は通常，自分の MENTAL-STATE が正の方向に向かうような行動をとる。



一方，個々の ACT に関して固有の推論が存在する。例えば INGEEST であれば

C2 に対して，前述の PTRANS のほかに次の推論 (i)，(ii)，(iii)，(iv) が行える。

(i) Yはもはや，もとの形をとどめなくなる。

(ii) Yが食べ物なら，Xの体の栄養になる。

(iii) Yが食べ物でなければ，Xは病気になる。

(iv) XがYをうまいと思えば，Xは満足する。

個々の ACT には，次のような対象に強く依存する推論 (i)，(ii)，(iii) が行える。

(i) Yが酒であれば，Xは酔うであろう。

(ii) Yがチョコレートであれば，Xの歯は悪くなるであろう。

(iii) Yが薬なら，Xは病気をしていたのであろう。およびXは回復するであろう。

これらの推論を行うための知識は対象に関する依存度が高いので，プリミティブ・アクションを中心に構成する知識とは別に，対象を中心として構成

する知識としてまとめる方が良いように思える。

PROPEL に関して次のような推論 (i), (ii), (iii), (iv) が行える。

(i) Yが固定した物でなければ, PTRANS を推論できる。

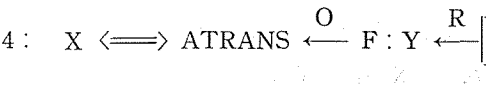
(ii) C3 の結果として Y ⟺ PHYS. CONT(Z) が生じておれば, ZはYにより物理的に負なる影響を受ける (Zが人ならけがをする)。

(iii) Zが人なら, XはZに対して怒る。

(iv) Yが固定した物でなくて, 固体でかつもろいとき, そして PROPEL の道具格のスピードが大きいとき, Yは負の物理的状態になる (こわれる)。

EXPEL であれば, EXPEL される物はもともと INGEST されていたとか, GRASP であれば対象はアクターよりも小さい, といった推論を規則にすることができる。

(2) ATRANS に関する推論



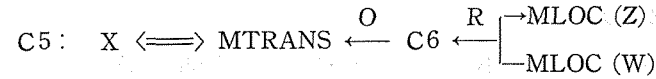
ATRANS は PTRANS と似た推論を行うことができる。主要な違いは物の所有権などの抽象的な関係が ATRANS の対象になっており, 次のような推論が行える。

(i) Zは抽象的関係 Z : Y にある。Wはもはや抽象的関係 F : Y にない。

(ii) Z が C4 を望んでいた場合, Z は Y を使って, 人が普通にやることをやるであろう。

(3) MENTAL-ACT に関する推論

MTRANS と MBUILD は情報の移動と生成に関している。MBUILD の道具格はいつも MTRANS となる。MTRANS の構造 C5 に関し, 次の推論 (i), (ii), (iii) が行える。



(i) C1 の結果として，Zは C6 を知っている。

(ii) X＝Wのとき，X は既に C6 を知っていた。

(iii) MLOC (Z) が X の LTM ならば，Xは C6 を学んだ。

4.2 原因・結果のタイプ

プリミティブ・アクションやプリミティブ・ステートを単位とする Conceptualization は原因・結果の関係で次々につながる。原因・結果の関係は次の四つのタイプが準備されており，それぞれ前方向，および逆方向にたどる推論を行うことができる。

(1) Result Causation

あるイベントがあるステート・チェンジを導く場合で，次のような文の Conceptualization に用いる。

"The lamp broke because John hit it."

(2) Enable Causation

あるステート・チェンジがあるイベントの起ることを可能にする場合で，次のような文の Conceptualization に用いる。

"John ate because there was food on the table."

(3) Initiation Causation

イベントやステート・チェンジがアクションやメンタル・イベントを導く場合で，次のような文の Conceptualization に用いる。

"John realized that Mary was hungry because he saw her crying."

(4) Reason Causation

アクションとその原因となったメンタル・ステートの関係を示すもので，次のような文の Conceptualization に用いる。

"John ate fish because he was hungry."

4.3 スクリプト

日常生活において，我々は多くの場面に関する知識を持っていると考えられる。例えば，レストランに食事に行く場合に，レストランの場面が一般的にどのような大道具，小道具から構成されているかについて知っているし，自分を含み，ウエイトレスやコック，レジの係などのアクターがどのような定形的な行動のシーケンスを取るかについて知っているので，無事に食事をすませて帰ることができる。一方，海外でレストランに入ると，自分の持っているレストランの知識からはずれることが多くなり，「ワインの味見」，「サラダのソース」，「勘定の仕方」，「チップ」などでとまどうことが多くなる。場面に関する知識はレストランの場合と同様に，「店で買物をする」，「切符を買って電車に乗る」，「銀行に行って預金する」，「病院で診察を受ける」など，日常生活のほとんどすべてにおいて重要な役割を演じている。このような知識は，計算機による文章理解のメカニズムを考える上で，前節の推論と同様に文中に陽に述べられない情報を補うものとして重要になる。Schankはこのような知識をスクリプトと呼んで形式化を試みている[34]。レストランのスクリプトの一つ Coffee-Shop は，次のように構成される。

```
Script  : RESTAURANT
Track   : Coffee-Shop
Props   : Tables, Menues, F-Food, Check, Money
Roles   : S-Customer, W-Waiter, C-Cook, M-Cook, M-Casher, O-Owner
Entry-condition : S is hungry, S has money
Result  : S has less money, O has more money,
          S is not ungry, S is pleased (optional)
Scenel  : Entering
          S PTRANS S into restaurant
          S ATTEND eyes to tables
          S MBUILD where to sit
          S PTRANS S to table
          S MOVE S to sitting position
Scene2  : Ordering
```

S PTRANS *menue* to S
S PTRANS food list to CP(S)
S MBUILD choice of F
S MTRANS W to table
S MTRANS "I want F" to W
...

Scene3 : Eating
...

Scene4 : Exiting
...

スクリプトを知識として備えることによって，次のような日常生活を記述した入力文章の近似的な理解状態を作り出すことができる。

「John はバスでニューヨークへ行った。
メトロポリタン・ミュージアムを見た。
列車に乗って帰った。」

入力文中の語をキーワードとして関連するスクリプト ＄BUS，＄MUSEUM-GOING，＄TRAIN が導かれたとする。このような文章を読んだときに，スクリプトの詳細に至るまで思い浮かべているとは思えないが，スクリプトへのエントリーが与えられていることは重要である。というのは「John はミュージアムのドアを通ったか」とか「John はバスの料金を払ったか」と言った質問に対して，それぞれ関連するスクリプトの中へ入ることによって，Yes と答えることができる。逆に，スクリプトに反する定形的でない事象が起るとき，即ち，「出口からミュージアムへ入った」，とか「切符を買わずに列車に乗った」という事象が起っておれば，入力文章の方で陽に述べることが必要になる。

4.4 プランとゴール

日常生活を記述した文章（物語）の理解を考える上で，記述される場面があらかじめスクリプトを準備しておける程には定形化できないような場合が

起ってくる。例えば次の文章を考えよう。

「Willa はひもじかった。

彼女はミシェラン・ガイドを取った。

車の所へ行った。」

この文章は人の状態「空腹」とそれから生じた意図「食事をしたい」を実行するための行動が記述されていると見ることができる。人が「食事をしたい」と思うときに，その場の状況に応じて「冷蔵庫の中をさがす」，「セブン・イレブンへ行く」，「レストランへ行く」など取り得る行動が変わってくる。そこで定形的な場面を記述するスクリプトとは別に，人の意図や目的とそのとき取る行動のパターンに関する知識を蓄えることが必要になる。Schank らはこのような知識をゴールとプランという形で形式化を行っている[36]。

前述の文章を理解するために，次のようなゴールとプランに関する知識が用いられる。

```
(INITIATE
  ((HUNGER ?A ?N) (< ?N 0))
  ((GOAL  (PLANNER ?A)
          (OBJECTIVE (HUNGER ?A 0)))))                    (28)

(PLANFOR
  ((GOAL  (PLANNER ?A)
          (OBJECTIVE (HUNGER ?A 0)))
  ((DO-RESTAURANT-PLAN (PLANNER ?A)
                       (RESTAURANT ?Y))))                 (29)

(SUBGOAL
  ((DO-RESTAURANT-PLAN (PLANNER ?A) (RESTAURANT ?Y)))
  ((GOAL (PLANNER ?A) (OBJECTIVE (PROX ?A ?Y)))
   (IS-A ?Y RESTAURANT)))                                 (30)

(SUBGOAL
  ((GOAL (PLANNER ?A) (OBJECTIVE (PROX ?A ?Y))))
```

```
((GOAL (PLANNER ?A)
       (OBJECTIVE (KNOW ?A (LOCATION ?Y))))))                    (31)

(PLANFOR
 ((GOAL (PLANNER ?A) (OBJECTIVE (PROX ?A ?L))))
 ((USE-VEHICLE-PLAN (PLANNER ?A))))                              (32)

(PLANFOR
 ((GOAL (PLANNER ?A)
        (OBJECTIVE (KNOW ?A (LOCATION ?Y)))))
 ((READ-PLAN (PLANNER ?A) (OBJECTIVE ?W))
  (IS-A ?W RESTAURANT-GUIDE)))                                   (33)

(SUBGOAL
 ((READ-PLAN (PLANNER ?A) (OBJECT ?Y)))
 ((GOAL (PLANNER ?A)
        (OBJECTIVE (POSESS ?A ?Y)))
  (IS-A ?Y BOOK)))                                               (34)

(PLANFOR
 ((GOAL (PLANNER ?A) (OBJECTIVE (POSESS ?A ?Y))))
 ((TAKE-PLAN (PLANNER ?A) (OBJECT ?Y))))                         (35)

(INSTANTIATION
 ((TAKE-PLAN PLANNER ?A)(OBJECT ?Y)))
 ((GRASP ?A ?O)))                                                (36)

(SUBGOAL
 ((USE-VEHICLE-PLAN (PLANNER ?A) (OBJECT ?Y)))
 ((GOAL (PLANNER ?A)
        (OBJECTIVE (PROX ?A ?Y)))
  (IS-A ?Y CAR)))                                                (37)
```

式(28)：「アクターが空腹状態にある」とき，「空腹状態をなくす」ことがアクターのゴールとして生成されることを示す。

式(29)：「空腹状態をなくす」ゴールには，「レストランで食事をする」プランが使えることを示す。

式(30)：「レストランで食事をする」プランを実行するには，「レストランへ行くこと」がサブゴールになることを示す。

式(31)：「ある場所へ行く」ゴールには，「その場所を知る」ことがサブゴールになることを示す。

式(32)：「ある場所へ行く」ゴールには，「車を用いる」プランが使えることを示す。

式(33)：「レストランこの場所を知る」には，レストラン・ガイドを読む」プランがあることを示す。

式(34)：「本を読む」プランは，「本を所有する」ことがサブゴールになることを示す。

式(35)：「物を所有する」ゴールには，「物を取る」ことがプランになることを示す。

式(36)：「物を取る」プランを実現する行動として“GRASP”の Conceptualization があることを示す。

式(37)：「車を用いる」プランは，まず「車の所へ行く」ことがサブゴールになることを示す。

以上のゴールとプランに関する知識から，文中に陽に述べられなかった情報を補うことによって，三つの文「Willa はひもじかった。ミシェラン・ガイドを取った。車の所へ行った。」は内容的にひとつながりの文章として理解できるようになる。

5. 演繹システム上での物語の理解

5.1 物語理解のメカニズム

物語理解のメカニズムを考える上で，入力文章に陽に述べられていない情報（言外の情報）は，4章で述べたプリミティブ・アクションに関する推論，およびスクリプトやプラン・ゴールとして形式化された知識から補うことに

なる。このような推論は，一般的に成り立つ事柄として知っている知識を，入力文章で記述される個別の事実に適用させてゆく過程と見なせるので，演繹的な推論と言うことができる。人工知能の分野において開発された演繹システム[28]を用いることによって，物語理解に必要な推論のメカニズムの大部分を実現することができる。なお，多くの個別の事実から一般的に成り立つ規則を導く帰納的な推論は学習過程の問題と深く関連するので，ここでは扱わないことにする。

演繹システム DUCK[22],[41] はYale 大学の Drew McDermott により開発されたもので，Prolog[27],[29] のような後方向推論のほかに，前方向推論[23]のメカニズム，定理がどの事実やどの規則から導れたかを示すデータ依存関係メカニズム，多重世界メカニズムを備えた大変優れたものである。DUCK は1階述語計算におけるモーダスポーネンス（帰結の法則）を自動的に実行するものと考えることができる。モーダスポーネンスにおける含意記号に対応して，前方向推論記号 “−>”，後方向推論記号の “<−” 二つが用いられる。前方向推論，後方向推論は論理的に見ると共に含意であるが，計算機上の手続として異なった意味を持っている。前方向推論は公理のデータベース中に新しく事実や規則が加えられる時点において適用され，導かれる定理をデータベースに加えてゆく。後方向推論は外部からある式の証明が要求された時点において適用され，モーダスポーネンスを逆方向に，公理のデータベースへと向かってたどってゆく。

入力文章を命題の列であると考えると，各々の命題が演繹システムのデータベースに加えられるときに，あらかじめ推論規則として与えておいた世界に関する知識との間で前方向推論が行われ，物語の理解状態に相当する事実や定理の集合をデータベース上に作ることができる。一方，文章の理解状態を確かめるための外部からの質問は，後方向推論の形をとる。一つの入力文が真なる命題として，データベースに加わるとき，あらゆる可能な推論を前方向に導き出すこともできるが，一つの文の入力時にあらゆる可能な推論を行ってしまうのは我々の文章理解の過程から離れてしまうように思える。一

方，命題で表された入力文がデータベースに加わる際に，何の前方向推論も行わず，すべて質問文からの後方向推論によって見かけ上の文章の理解状態を作り上げることもできるのであるが，文章理解のモデルとしては適当でない。文章理解の過程をモデル化するに当り，必要に応じて前方向推論の深さをコントロールすることも必要になるし，局所的に後方向推論を行うことも必要になると思われる。そこで，文章理解のモデルは演繹システムを単独で用いて実現するのではなくて，演繹システムとそれを外部からコントロールするメタシステム（制御系）を考え，全体の系として文章理解のモデルを考えるのが良いように思える。

5.2 入力文章の命題表現

入力文は第3章で述べた Conceptual Parser により概念依存構造に変換できると考えているので，概念依存構造の命題表現を定めることにより，近似的に入力文の命題表現を得ることができる。概念依存構造はプリミティブ・アクションやプリミティブ・ステートに関してできる概念の関係が基本単位となって，因果関係や導具格の関係により次々につながって構成される。そこで，アクションに対応して "EVENT"，ステートに対応して "STATE" という2変数の述語を導入する。式(38)は EVENT を用いた命題，式(39)は STATE を用いた命題の例を示す。それぞれ，第1項は EVENT および STATE のトークンを，第2項は EVENT および STATE のタイプを示す。トークンは EVENT や STATE の各々を区別して扱うための名前（スコーレム定数）で，重複しないような名前を自動的に割り当てる。

(EVENT %23 (PTRANS John gun1 New-York Boston)) (38)

(STATE %24 (POSESS John gun1)) (39)

アクション "PTRANS" やステート "POSESS" はそのままを述語として用いずに，述語内の関数として用いている。述語計算における関数は評価

を受けて値が求まるようなことはないのであるが，値はアクションやステートそのもの，即ち(38)では「John が Gun1 を New York から Boston へ PTRANS すること」を示し，(39)では「John が Gun1 を所有している状態」を示していると考える。アクションやステートをそのまま述語として用いずに，イベントやステートをトークンとタイプという形で扱うことによって，高階の述語を用いることなしに，イベントやステート間の原因・結果の関係，順序関係などを容易に表すことができる。また，全く同一のタイプのイベントやステートが2度起ったとしても，それぞれのトークンにより区別して扱うことができる。述語 "RESULT"，"INITIATE"，"ENABLE"，"REASON" はイベントやステートのトークンを引数とする2変数の述語である。これらは下記のような形で概念依存構造における原因結果の関係を表すのに用いる。述語 "PRECEDE"，"SAMETIME" は原因・結果の関係よりも弱い単なる時間的な順序関係を表すのに用いる。

(RESULT %31 %32)
(INITIATE %32 %33)
(ENABLE %35 %36)
(REASON %37 %38)

(PRECEDE %41 %42)
(SAMETIME %37 %38)

文章理解のメカニズムを考えるために，次の物語「犬の話」を例にとろう。

「犬の話」

S 1 ：ある日，犬は肉のかたまりを拾いました。
S 2 ：犬は肉をくわえて家へ帰りました。
S 3 ：途中，小川の丸木橋まで，やってきました。
S 4 ：橋から小川を見ると，
S 5 ：川には自分の姿がうつっていました。
S 6 ：犬はそれを肉を加えた別の犬だと思いました。
S 7 ：犬はそれもとってやろうと思いました。

S８：犬は影にほえました。
S９：しかし，口をあけたとたん，肉は川に落ち，
S10：犬は肉をなくしてしまいました。

第１文Ｓ１は「ある日」の部分を除外すると命題Ｐ１に変換される。文中に現れる普通名詞「犬」および「肉」は第１のイベントとして登場した特定の犬であり，特定の肉であると考えて，一般の「犬」や一般の「肉」の概念と区別して扱うために，犬は DOG1，肉は MEAT１という名前（スコーレム定数）を与える。一方，DOG1 や MEAT1 と「犬」,「肉」の概念との関係を与えるために述語 "IS-A" を導入する。これは「こと」の概念に対応して述語 "EVENT" を導入したことに相当し，「もの」の概念におけるトークンとタイプと見ることもできる。もし入力文中に「犬」の代りに「ポチ」のような固有名詞が用いられておれば，スコーレム定数を導入せずに，「ポチ」をそのままの形で用いることになる。

```
P１：(AND (EVENT %1 (PICKUP DOG1 MEAT1))
          (IS-A DOG1 DOG)
          (IS-A MEAT1 MEAT))
```

第２文Ｓ２は命題 P２に変換される。"(HOME-OF DOG1)" は "HOME-OF" という関数で "DOG1" をマッピングすること，即ち犬１の家を示している。関数 "PTRANS" は本来の定義（第３章）を簡略化して用いており，第１項は "PTRANS" の対象，第２項は "PTRANS" の帰着点を示す。関数 "GOAL" は第１項の動作主が第２項を目的として頭の中に抱くことを示している。従って，この命題は，「犬１が肉１を犬１の家へと移動させることを目的とする事象が起ったこと，およびそれをトークン "%２" で示すこと」を主張している。

```
P２：(EVENT %2 (GOAL DOG1
                     (PTRANS MEAT1 (HOME-OF DOG1))))
```

ここで第２文に現れた名詞「犬」や「肉」が第１文で現れた「犬」や「肉」

と同一視するかどうかを定めるメカニズムが必要になる。システムの動作を簡単化して考えるために，Conceptual Parser の機能として，文中に普通名詞「犬」や「肉」を見つけると，あいまいさが生じない限り先行する文に現れた「犬」や「肉」と同一視して，それぞれ "DOG1", "MEAT1" を割り当てることにする。以下同様にして入力文S3～S10を命題に変換すると次のP3～P10が得られる。

P3 : (EVENT %3 (PTRANS DOG1 BRIDGE1))
P4 : (EVENT %4 (ATTEND DOG1 BROOK1))
P5 : (EVENT %5 (REFRECT BROOK1 DOG1))
P6 : (EVENT %6 (MBUILD DOG1 (POSESS DOG2 MEAT2)))
P7 : (EVENT %7 (GOAL DOG1 (POSESS DOG1 MEAT2)))
P8 : (EVENT %8 (BARK DOG1 DOG2))
P9 : (EVENT %9 (PTRANS MEAT1 BROOK1))
P10 : (EVENT %10 (LOSE DOG1 MEAT1))

このような命題のセットは「もの」の概念が DOG1, MEAT 1,……で，「こと」の概念が%1，%2，…などの定数で表される世界（童話の中の世界）において，個体相互を関係づけるものと見ることができる。即ち，次々に入力される命題は童話の中に登場する犬や肉などの個体相互の間に意味的なネットワークを張るものと考えることができる(図8)。ここで，各々の入力文を命題で表した場合に，入力文が次々に述べられたことに含まれていたイベントやステート間の順序の情報が失われている。そこで，物語理解の過程の一部に，前述の述語 "PRECEDE" を用いて，イベントやステート間に順序を与えることが必要になる。

(PRECEDE %1 %2)
(PRECEDE %2 %3)
(PRECEDE %3 %4)
…

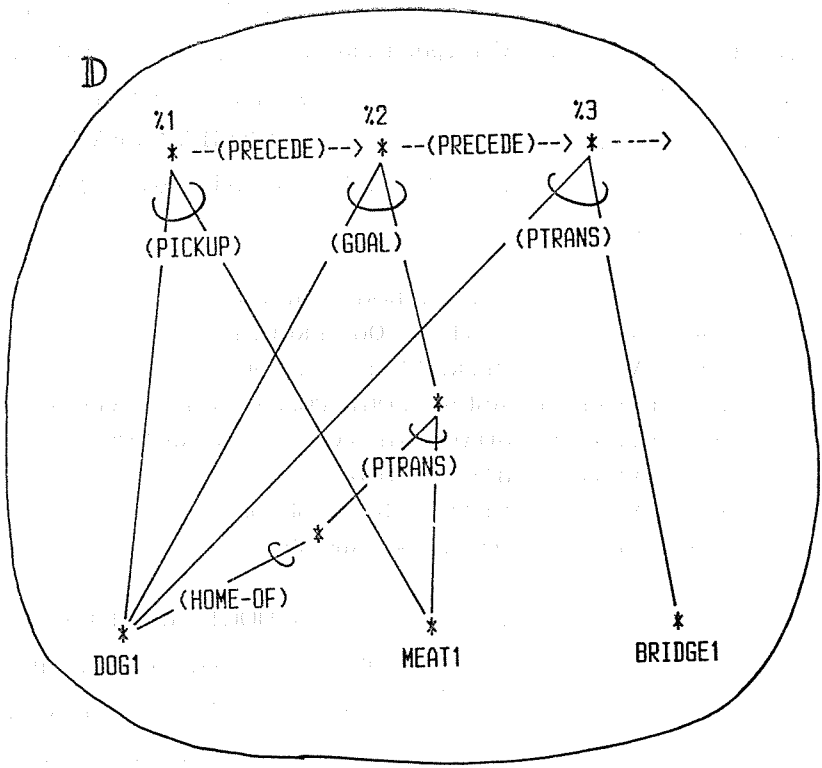


図8　物語「犬の話」の世界

5.3　推論規則による知識の表現

第4章で述べたプリミティブ・アクションに関する推論，および，スクリプトやプラン・ゴールによる推論は演繹システムにおいて，前方向および後方向推論規則の形で表現される。マクロなイベント“PICKUP”はスクリプトに相当する次の推論規則を持っている(40)。記号“?”はそれが付加される変数に全称記号がバインドされていることを示す。第1文S1に対応して命題P1が公理のデータベースに加えられると，前方向推論規則(40)が適用される。演繹システムのメタシステム(制御系)は制御用の述語“S-EXPAND”

を用いて，前方向推論の深さをコントロールする。メタシステムは必要に応じて命題 "(S-EXPAND %1)" を加え，マクロなイベント "PICKUP" をスクリプトへと展開する。

```
(-> (EVENT ?E (PICKUP ?A ?O))
    (-> (S-EXPAND ?E)
        (AND (EVENT (E1-OF ?E) (ATTEND ?A ?O))
             (EVENT (E2-OF ?E) (GRASP ?A ?O))
             (STATE (S-OF ?E) (POSESS ?A ?O))
             (PRECEDE (E1-OF ?E) (E2-OF ?E))
             (PRECEDE (E2-OF ?E) (S-OF ?E)))))          (40)
```

前方向推論規則(41)は物の移動 "(PTRANS ?O ?T)" を動作主 ?A が目的とした場合に中間地点 "(HALFWAY-OF ?E)" まで移動させることがサブゴールになること，および物を移動させる行為そのもの "(PTRANS ?O ?T)" がプランになることを示している。この推論も制御用の命題 "(G-EXPAND ?E)" および "(P-EXPAND ?E)" を加えることによって初めて導かれる。"G1-OF", "P1-OF" はイベントのトークンとしてユニークな名前を与えるために，また "HALFWAY-OF" は物の移動における中間地点の存在を示す中間地点名を与えるためのスコーレム関数である(図9)。

```
(-> (EVENT ?E (GOAL ?A (PTRANS ?O ?T)))
    (AND (-> (G-EXPAND ?E)
             (EVENT (G1-OF ?E)
                    (GOAL ?A (PTRANS ?O (HALFWAY-OF
                     ?E)))))
         (-> (P-EXPAND ?E)
             (EVENT (P1-OF ?E)
                    (PTRANS ?O ?T)))))                  (41)
```

後方向推論規則(42)はプリミティブ・アクション "PTRANS" に関する推論を表すもので，ある物がある場所に移動したことを示すためには，動作

主？Aがその物？Oを所有しており，その後？Aがある場所？Tへと移動したことが証明できれば良いことを示している。この規則により，犬が肉を所有したのち，犬が橋へ移動したのであれば，肉も橋へ移動したことを証明することができる。なお，この規則が正しく働くためには述語“PRECEDE”に関し推移律が成り立つ規則を与えておくことが必要である。“THNOT”は命題が証明されないときに限り真を値として返すオペレータ[18], [19], [40]である(図10)。

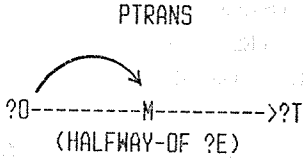


図9　中間地点への移動

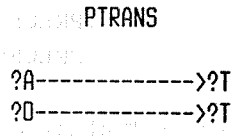


図10　必然的関係

```
(<- (EVENT (E-OF ?S ?E) (PTRANS ?O ?T))
    (AND (STATE ?S (POSESS ?A ?O))
         (EVENT ?E (PTRANS ?A ?T))
         (PRECEDE ?S ?E)
         (THNOT (AND (EVENT ?E2 (UNGRASP ?A ?O))
                     (PRECEDE ?S ?E2)
                     (PRECEDE ?E2 ?E)))))                    (42)
```

同じく“PTRANS”に関する推論規則(43)は物の移動というイベントが起こったときに，その物が帰着点に存在するという状態が生じることを示している。イベントとステートはどちらもトークンとタイプの二つの項を取る述語であるが，命題の列に時間的な解釈を加える場合に取り扱いが異なってくる。イベント命題は時点の記述として取り扱い，データベースに加えられた時点で既に過去を記述したものとして扱う。ステート命題は時間区間を記述したものとして扱い，状態の対立が起こらない限り，現在も成り立つものとして扱う。

(→) (EVENT ?E (PTRANS ?O ?T))
(STATE (S-OF ?E) (LOCATE ?O ?T))) (43)

5.4 物語理解のシナリオ

第1命題P1が加えられると，制御系は“PICKUP”がマクロなイベントであり，ほかにアクティブなスクリプトがないことから制御用の命題“(S-EXPAND %1)”を加えて，入力命題をスクリプトへと展開する。第2の命題P2が入力されると制御系は公理のデータベースに加える前に，その命題で表されるイベントが現在のスクリプトから予測されうるものがどうかテストする。まずイベントのトークンの%記号を？記号に書き換え証明を試みる（？記号は証明の際，存在記号として働く）。この場合，%2に対応するイベントはスクリプト内に存在しないので，証明は失敗する。次に，第2命題 P2が前文 P1と話の内容がつながったものとして扱えるかどうかをテストするために，第2命題に含まれる項が，これまでの命題の中に含まれているかどうかの証明を行う。こちらは DOG1, MEAT1 がすでに存在しているので成功する。そこで，第2命題 P2をデータベースに加え，次に制御系はイベントのタイプが先行する命題とどのような関係にあるかを調べる。第2命題 P2のイベント・タイプが，“GOAL”というメンタル・イベントであること，先行するイベントがアクションに関するマクロ・イベントであることから，先行するイベントが第2イベントを引き起こす原因になったと解釈して，新たな命題“(INITIATE %1 %2)”をデータベースに加える。

第3命題P3が入力されると，この命題が先行する命題から予測されるかどうか調べるためのテストを行う。現在，入力されている命題P3がプリミティブ・アクション“PTRANS”を表すものであり，先行する命題P2がメンタル・イベント“GOAL”であることから，現在のイベント“PTRANS”は先行する命題のプランになっているのではないかと予測し，推論制御用の命題“(P-EXPAND %2)”をデータベースに加え，第2命題をプランへと展開する。しかし，展開された命題は入力命題P3と似てはいるが帰着点が

"BRIDGE1" ではなくて "(HOME-OF DOG1)" となっている。ここで,「犬の家が橋(の下)である」という仮定も成り立つのであるが,制御用の命題"(G-EXPAND %2)" を加え,第2命題P2をサブゴールへと展開し,さらに制御間の命題 "(P-EXPAND (G1-OF %2))" を加えると,再び命題P3と類似の命題が導かれる。この場合 PTRANS の帰着点が BRIDGE1 ではなくて "(HALFWAY-OF %2)" となっている。第3命題P3の "BRIDGE1" と第2命題をサブゴールとプランに展開して得られる命題の中の "(HALFWAY-OF %2)" を等しいものと考えると,話のつじつまが合って文のつながりが良くなることから,これらが同一の個体を指示することを仮定するための命題(44)をデータベースに加える。さらに第3命題P3は第2命題のプランになってたことが導かれたとして命題(45)を加える。第3命題P3が加えられた時点で前方向推論規則(43)により(46)がデータベースに加えられる。

(ALIAS BRIDGE1 (HALFWAY-OF %2)) (44)

(GOAL-PLAN %2 %3) (45)

(STATE (S-OF %3) (LOCATE DOG1 BRIDGE1)) (46)

第4命題も他の命題と同様に,まず先行する命題から予測されうるかどうかのテストを受ける。第3命題に含まれる "BRIDGE1" をキーワードとして「橋と小川」の場面のスクリプトが導かれ,たまたまスクリプト内に第4命題と同じタイプのイベントが含まれておれば,第4命題P4は予測された命題として直接証明されることになる。「橋—小川」のスクリプトを持たない場合においても「橋」と「川」が関連することは最小限の知識として知っているとすれば,命題(47)が知識として存在することになる。

制御系は第4命題が直接証明されなくても,第3命題から DOG1 が BRIDGE1 におり,BRIDGE1 が BROOK1 と ACROSS-OVER の関係

にあることから，先行する命題と話がつながっていると解釈し，第4命題が物理的イベントであることから，第3命題が第4命題のイベントを可能にしたと判断して，次の命題(48)をデータベースに加える。

第5命題に対して「鏡面反射」の場面を記述したスクリプトを準備しておくと第5命題P5は先行する命題から予測できなくても，鏡面反射のスクリプトの方に第4命題の述語 "ATTEND" が含まれており，制御系はスクリプトから与えられる関係を用いて命題(50)をデータベースに加える。

(ACROSS-OVER BRIDGE BROOK) (47)

(ENABLE %3 %4) (48)

(EVENT (E-OF ?E) (REFRECT ?O1? O2)) (49)

(RESULT %4 %5) (50)

同じく「鏡面反射」に関する知識として規則(51)を準備しておく。これは鏡にアクターの状態が映り，イメージが得られることを示している。この規則は命題P4，P5および命題P1のスクリプト内命題 "((S-OF %1))" により命題(52)を導く。これは命題の表すステートのタイプは命題(54)と(55)を仮定すると，第6命題のタイプ "MBUILD" の第2項と一致する。そこで，制御系は第5命題のイベントが原因となり第6命題のメンタル・イベントを引き起こしたと判断して命題(53)をデータベースに加える。

```
(-> (EVENT ?E1 (ATTEND ?A ?O1))
    (-> (STATE ?S (?FUN ?A ?O2))
        (-> (EVENT ?E2 (REFRECT ?O1 ?A))
            (STATE (S-OF ?E1 ?S ?E2)
                   (?FUN (IMAGE-OF ?A) (IMAGE-OF ?O2))))))
```
(51)

```
(STATE (S-OF %4 (S-OF %1) %5)
```

(POSESS (IMAGE-OF DOG1) (IMAGE-OF MEAT1))) (52)

(INITATE %5 %6) (53)

(ALIAS DOG2 (IMAGE-OF DOG1)) (54)

(ALIAS MEAT2 (IMAGE-OF MEAT1)) (55)

第7命題が先行する命題と関連することを証明するために，アクターがオブジェクトを所有したくなるのは，アクターがオブジェクトを好むことを規則にする。これは後方向推論規則(56)となる。一般的な知識として「犬が肉を好む」ことを知識にするため命題(57)を与えておく。命題(56)と(57)より第7命題はもっともなこととして予測される。また先行する命題P6のタイプが“MBUILD”であることと命題P7のタイプが“GOAL”であることから，制御系はP6とP7の関係を命題(58)として与える。

```
(<- (EVENT ?E (GOAL ?A (POSESS ?A ?O))))
    (AND (IS-A ?A ?AA)
         (IS-A ?O ?OO)
         (LIKE ?AA ?OO)))                    (56)

(LIKE DOG MEAT)                              (57)

(INITIATE %6 %7)                             (58)
```

第8命題は第7命題を実現するためのプランであると考えられるので，犬が何かをほしがるときは「吠える」プランを使うという知識を規則にする(59)。

第9命題を説明するために，犬は吠えると口があく状態になるという知識を規則にする(60)。また，犬が物をはなすというイベントは，初め犬が物を所有しており，後に口を開けたことが証明できるときに起こるという知識を

規則にする(61)。

第10命題を説明するために，「物を失う」というイベントは過去に物を所有しており，後にそれをはなすという事象が起こっておればよいという知識を規則にする(62)。

```
(−> (EVENT ?E (GOAL ?A (POSESS ?A ?O)))
    (−> (IS-A ?A DOG)
        (−> (P-EXPAND ?E)
            (EVENT (E-OF ?E) (BARK ?A ?O2)))))                    (59)

(−> (EVENT ?E (BARK ?A ?O))
    (STATE (S-OF ?E) (OPEN (MOUTH-OF ?A))))                       (60)

(<− (EVENT (E-OF ?S1 ?S2) (UNGRASP ?A ?O))
    (AND (STATE ?S1 (POSESS ?A ?O))
         (IS-A ?A DOG)
         (STATE ?S2 (OPEN (MOUTH-OF ?A)))
         (PRECEDE ?S1 ?S2)))                                      (61)

(<− (EVENT (E-OF ?E ?S1) (LOSE ?A ?O))
    (<− (AND (STATE ?S1 (POSESS ?A ?O))
             (EVENT ?E (UNGRASP ?A ?O))
             (PRECEDE ?S1 ?E)
             (THNOT (AND (STATE ?S2 (POSESS ?A ?O))
                         (PRECEDE ?E ?S2))))))                    (62)
```

5.5 システムの動作と問題点

命題の入力を終えた状態で，物語が正しく理解されているかどうかを確かめるには，システムに質問を発すればよい。質問文は入力命題と全く同じ形をしているが，システムのモードによりこの命題を証明せよという形で動作する。命題中に現れる“?”記号は，平叙文の場合と異なり，疑問文では存在記号として働く。システムに発した質問システムからの解答の例をＱ１～Ａ５に示す。

イベントやステート間の順序関係についての質問も行いたいので，順序関係を示す述語“PRECEDE”に関し推移律が成り立つことを示す規則を与える。前方向推論規則(63)で与えるのが一番簡単であるが，すべてのイベントやステートの組合せについて順序関係を示す命題を生成してしまうので，メモリを多く使うことになる。後方推論規則で(63)をそのまま書き換えるとシステムはダイナミック・ストップを起こす。そこで，近接したイベントやステートの順序関係を表す述語“PRECEDE”と区別して，時間的に隔った順序関係も表せる述語“PREC”を導入し，実質的に時間的順序関係の推移律を定義する(64)。

```
(-> (PRECEDE ?A ?B)
    (-> (PRECEDE ?B ?C)
        (PRECEDE ?A ?C)))                    (63)

((- (PREC ?A ?B)
    (OR (PRECEDE ?A ?B)
        (AND (PRECEDE ?A ?X)
             (PREC ?X ?B))))                 (64)
```

命題Ｐ１—Ｐ10をシステムに公理として与えた後，システムに発した質問とシステムからの解答を示す。それぞれ自然語文に翻訳した会話と，実際に行った計算機との会話である。

Ｑ１　：だれが何を拾ったか？
Ａ１　：犬が肉を拾った。
Ｑ２　：肉はどこへ移動したか？
Ａ２　：肉は川へ移動した。
Ｑ３　：犬は何をしたかったか？
Ａ３Ａ：肉を家へ運びたかった。
Ａ３Ｂ：肉を得たかった。
Ｑ４　：犬は吠える前に何を見たか？
Ａ４　：犬は川を見た。
Ｑ５　：犬が川を見たら何が起こったか？

A 5 ：犬が川に映った。

(DUCK)

Q 1 : G?> (EVENT ?E (PICKUP ?A ?O))

A 1 : (EVENT %1 (PICKUP DOG1 MEAT1))

O＝MEAT1

A＝DOG1

E＝%1

?>A

NO MORE

Q 2 : G?> (EVENT ?E (PTRANS MEAT1 ?T))

A 2 : (EVNT %9 (PTRANS MEAT1))

T＝BROOK1

E＝%9

?>A

NO MORE

Q 3 : G?> (EVENT ?E (GOAL DOG1 ?X))

A 3 A : (EVENT %2 (GOAL DOG1 (PTRANS MEAT1 (HOME-OF DOG1))))

X＝(PTRANS MEAT1 (HOME-OF DOG1))

E＝%2

?>A

A 3 B : (EVENT %7 (GOAL DOG1 (POSESS DOG1 MEAT2)))

X＝(POSESS DOG1 MEAT2)

E＝%7

?>A

NO MORE

Q 4 : G?> (AND (EVENT ?E(BARK DOG1 ?O))
(PREC ?X ?E)
(EVENT ?X (ATTEND DOG1 ?A)))

A 4 : (AND (EVENT %8 (BARK DOG1 DOG2))
(PREC %4 %8)
(EVENT %4 (ATTEND DOG1 BROOK1)))

A＝BROOK1

X＝%4

O＝DOG2

```
        E＝%8
        ?⟩A
        NO MORE
Q 5   : G?⟩ (AND (EVENT ?E (ATTEND DOG1 BROOK1))
                 (PRECEDE ?E ?X)
                 (EVENT ?X ?XX))
A 5   :(AND (EVENT %4 (ATTEND DOG1 BROOK1))
            (PRECEDE %4 %5)
            (EVENT %5 (REFRECT BROOK1 DOG1))
        XX＝(REFRECT BROOK1 DOG1)))
        X＝%5
        E＝%4
        ?⟩A
        NO MORE
        G?⟩
```

質問Q１～Q５に対するシステムからの解答Ａ１～Ａ５は，主として演繹システムが持つ関係データベースに対する検索能力と事象や状態間の順序関係についての推論を利用したもので，5.3節および5.4節で述べたスクリプトやプランに基づくものは用いていない。スクリプトやプランから導かれる言外の情報を用いると，事象や状態間の単なる時間的順序関係だけでなく，原因，理由，結果などのより強い意味的関係を導くことができる。また，「吠えた後，川に映っている犬の状態はどのようになっているか」など，入力命題の検索からは導くことのできない命題を推論により導くことができるようになる。

スクリプトやプランを用いる上で，仮定を行うメカニズムが必要になる。例えば第２命題「犬が肉を家へ持って帰ろうと思った」というゴールに対して，サブゴールとプランを用いると，「犬が家までの中間地点へ移動する」という命題が導かれる。ここで「中間地点」と「橋」とを同一視するという仮定を置くと，第３命題はプランの実行を述べたものとして，第２命題との意味的なつながりが良くなる。即ち，第３命題の“BRIDGE1”は“(HAL-

FWAY-OF %2)" で置き換えてやれば第2命題をサブゴールとプランに展開したものから証明することができる。

そこで，どのような仮定をおくと，ある命題が証明できるようになるかという「仮定」を求める操作が必要になる。これは文章理解のメカニズムに「確かな推論」だけでなく「仮定を置くとつじつまが合う」といった「確からしい推論」を行うメカニズムが必要なことを意味している。しかし，確かな推論を行う演繹システムの内部で実現することは難しく，メタシステムを含む全体システムとして実現することになる。

文章理解を演繹システム上に実現する場合に，公理系の本質にかかわる別種の問題がある。「犬の話」では，現時点で犬が肉を所有しているかどうかということが文章の進行に従って変わってくる。即ち，第1命題が入力された後は犬は肉を所有しており，第8命題が入力された後は犬は肉を所有していないことになる。これまで事象および状態を表す命題は入力された時点において過去の記述として扱ったのであるが，現時点の状態を表す述語 "PRESENT-STATE" を導入すると，第1命題のあとで "(PRESENT-STATE (POSESS DOG2 MEAT1))" は真となり，第8命題のあとでは偽となることが必要である。しかし，これまでの論理学では，ある公理系に新たに公理（第8命題）を追加したとき，矛盾が生じたことに相当し，矛盾のある系として，それ以上の議論を行うことができなかった。これは従来の論理学における公理系が単調論理と呼ばれている理由である。文章理解のモデルは公理系に新たな公理が加わることによって，それまで導かれていた定理が取り消されることもありうる非単調論理[8],[18],[19],[20],[24]において考えることが必要である。演繹システム DUCK では式(62)で用いた "THNOT"，および無矛盾がどうか調べることのできる "CONSISTENT"，デフォールト推論（例，鳥ならば飛ばないことが証明されないかぎり飛ぶと推論してよい）"NEGLECTING" などの述語を用いて非単調推論を実現することができる。これらを用いて，事象・状態の対立などの関係を記述することができる。

おわりに

これまで自然言語の計算機処理に関する研究は，文法情報に基づく構文解析[39],[42]の形で進められてきた。構文解析に用いられる規則は，我々が暗黙の内に用いている言語学的な知識を形式化したものである。単一の文の理解を考える上では文法情報に基づき構文解析木を作り出すことも重要なのであるが，3.3節の“took”の例のように構文的に同じでも意味的に異なるものを扱う場合，また多くの文からなる文章の理解を考える場合には，言語の表層を扱った言語学的知識よりもむしろ世界に関する知識（概念体系）の方が重要になってくる。計算機という形で自動機械の存在しなかった時代においては，言語理解のモデルを考えようとしても，どこまでを実現可能なものとして扱って良いかの基準が存在しないため，言語の表層から離れる必要のある言語理解の研究は科学として成り立つことができかなったと思われる。計算機の進歩により，現在，我々は言語理解のモデルを計算機上に実現することが可能な時点にさしかかっている。従来，唯物論において扱うことの難しかった言語の意味・内容や概念の構造を計算機上の情報の構造として具象化して取り扱うことができる。即ち，従来，言語学が避けて通らざるを得なかった意味・内容の問題を科学として正面から取り組むことのできる時代になったのである。Schank の概念依存構造の理論は，このような方法で意味の問題に取り組んだ最初の試みである。近年，計算機による言語理解の研究は計算機科学における人工知能の分野で進められ，この分野の大きなテーマとなっている。国語研究所では早くから言語研究に計算機を取り入れたのであるが，計算機を，言語現象を説明するためのモデルの実験・実証装置として用いるという発想がなかったために，単にデータの並べ換えや集計を行う事務処理タイプの計算機システムが設置されている。計算機上に言語理解のモルデを構成するには，従来の字面レベルの言語処理とは比較にならない程，多くのプログラムを作成して，計算機実験を繰り返すことになる。このように計算機上にモデルを作り実験・実証的な方法で言語研究を進めるには，大

量データ単純処理用に作られたビジネスタイプの計算機は適当でなく，計算機実験に向くすぐれたオペレーティング・システムとエディタ，それにすぐれたLISPを持つリサーチ・タイプの計算機が必要になる。ここで述べたシステムはすべて，東大計算機センターのリサーチ・タイプの計算機VAX11/780 (UNIX) 上において実現している。

謝辞 ここで述べた多くのアイデアは，1982年度に文部省在外研究員として Yale 大学（米国，コネチカット州）計算機科学科，人工知能プロジェクトに滞在中に学んだことに負っている。滞在を許可し，学習と研究の機会を与えていただいた Roger Schank 教授，演繹システム DUCK の使用を許可していただいた Drew McDermott 教授に心から感謝する。

参考文献

[1] Allen, J. "Maintaining Knowledge about Temporal Intervals" Communication of the ACM, Vol. 26, No. 11, 1983

[2] Chang, C., Lee, R. "Symbolic Logic and Mechanical Theorem Proving" Academic Press, 1973

[3] Charniak, E., Wilks, Y. "Computational Semantics" North-Holland, 1976, Amsterdam

[4] Charniak, E., Riesbeck, C., McDermott, D. "Artificial Intelligence Programming" Lawrence Erlbaum Associates, New Jersey, 1980

[5] Chikayama, T. "UTILISP Manual" Techinical Reports, Dept. of Mathematical Engineering and Instrumetation Physics, Univ. of Tokyo, 1981

[6] Clocksin, W., Mellish, C. "Programming in Prolog" Springer-Verlag, Berlin, 1981

[7] Doyle, J. "A Glimpse of Truth Maintenance" Proc. of Sixth International Joint Conference on Artificial Intelligence, Tokyo, 1979

[8] Doyle, J. "A Truth Maintenance System" Artificial Intelligence, Vol. 12, No. 3, 1979

[9] Dyer, M., "In-Depth Understanding" Research Report #219, Dept. of Computer Science, Yale Univ., 1982

[10] Foderaro, J. "The Franz LISP Manual" in UNIX Manual, Univ. of California, Berkeley, 1980

[11] Kanada, Y. "HLISP and Supplementary HLISP-REDUCE Manual" Univ. of Tokyo, 1979

[12] Kornfeld, W. "Equality for Prolog" Proc. of Eights International Joint Cofnerence on Artificial Intelligence, Karlsruhe W. G., 1983

[13] Kowalski, R. "Logic for Problem Solving" Elseview North-Holland, New York, 1979

[14] Lehnert, W., Ringle, M., "Strategies for Natural Language Processing" Lawrence Erlbaum Associates, New Jersey, 1982

[15] Lytinen, S., Schank, R. "Representation and Translation" Research Report #243, Dept. of Computer Science, Yale Univ., 1982

[16] Manna, Z., Pnueli, A. "Verification of Concurrent Programs, part 1 : The Temporal Framework" Dept. of Computer Science, Stanford Univ., 1981

[17] McDermott, D. "Planning and Acting" Cognitive Science, Vol. 2, 1978

[18] McDermott, D. "An Introduction to Non-Monotinic Logic" Proc. of Sixth International Joint Conference on Artificial Intelligence", Tokyo, 1979

[19] McDermott, D., Doyle, J. "Non-Monotonic Logic I" Artificial Intelligence, Vol. 13, 1980

[20] McDermott, D., "Non-Monotonic Logic II" Research Report #174, Dept. of Computer Science, Yale Univ., 1980

[21] McDermott, D. "A Temporal Logic for Reasoning about Process and Plans" Cogniuive Science, Vol. 6, Ablex Publishing Co., 1982

[22] McDermott, D. "Duck : A Lisp-based Deductive System" Dept. of Computer Science, Yale Univ., 1982

[23] Moor, R. "The Role of Logic in Knowledge Representation and Artificial Intelligence" Proc. of AAAI-82, Pittsburg, U. S. A., 1982

[24] Moore, R. "Semantical Considerations on Nonmonotonic Logic" Proc. of Eights International Joint Conference on Artificial Intelligence, Karlsruhe W. G., 1983

[25] Meehan, J. "The New UCI LISP Manual" Lawrence Erlbaum Associates, New Jersey, 1979

[26] Minsky, M. "A Framework for Representing Knowledge" in The Psychology of Computer Vision, McGraw-Hill, New York, 1975

[27] Nakashima, H. "Prolog/KR User's Manual" Dept. of Mathematical Engineering and Instrumentation Physics, Univ. of Tokyo, 1982

[28] Nilsson, N. "Principles of Artificial Intelligence" Tioga Publishing Co., 1980

[29] Pereila, F. "C-Prolog User's Manual" SRI International

[30] Roberts, B., Goldstein, P. "The FRL Manual" M. I. T. AI-Memo 409, 1977

[31] Sacerdoti, E. "A Structure for Plans and Behavior", Elsevier North-Holland, New York, 1977

[32] Sato, M., Sakurai, T. "Qute User's Manual", Dept. of Information Science, Univ. of Tokyo, 1983

[33] Schank, R. "Conceptual Information Processing" North-holland/ American Elsevier, 1975, New York

[34] Schank, R., Abelson, R. "Scripts Plans Goals and Understanding" Lawrence Erlbaum Associates, New Jersey, 1977

[35] Schank, R., Birnbaum, L. "Memory, Meaning, and Syntax" Research Report #189, Dept. of Computer Science, Yale Univ., 1980

[36] Schank, R., Riesbeck, C. "Inside Computer Understanding" Lawrence Erlbaum Associates, New Jersey, 1981

[37] Schank, R. "Reading and Understanding" Lawrence Erlbaum Associates, New Jersey, 1982

[38] Schank, R. "Dynamic Memories" Cambridge Univ. Press, London, 1983

[39] Simmos, R. "Semantic Networks : Their Computation and use for Understanding English Sentences" in Computer Models of Thought and Language, Freeman, San Francisco, Calif., 1973

[40] Sussman, G., Winograd, T., Charniak, E. "Micro-Planner Reference Manual" M. I. T. AI-Memo No. 203A, 1971

[41] Tanaka, T. "Representation and Analysis of Electrical Circuit in a Deductive System" Proc. of Eights International Joint Conference on Artificial Intelligence, Karlsruhe W. G., 1983

[42] Woods, W. "Transition Network Grammars for Natural Language Analysis" Communications of the ACM, Vol. 13, No. 10, 1970