Conceptual Parser and Inferencer
－Yale大学 A.I. projectのノートから－

田中卓史　（国立国語研究所）

はじめに

Yale大学．計算機科学科の人工知能プロジェクトに滞在する機会を得たので，滞在中のノートからConceptual ParserとInferencer，Inferencerに関連してdeductive system DUCKを紹介する。作成したシステムはRoger Schankの自然言語処理の授業（1982年秋学期 CS470/570）のhomeworkで，ほぼMARGIE[4][5]に相当する動作を行い，スクリプト[7]やMOPS[8]の考えは用いていない。Conceptual ParserはChristopher RiesbeckのMcELI[3][11][10]*を参考にし，InferencerはLISPで書いたものとDrew McDermottのdeductive system DUCK（CONNIVERの子孫）を利用したものを作った。

1. Conceptual Parser

従来の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，Conceptual Parserは入力文の意味内容を表す概念表現を作り出す。Syntactic Parserでは構文上のあいまいさを取り除くためにSemanticsを補助的に用いることが多かったが，Conceptual Parserでは言語学的（Syntactic）な知識と非言語学的な知識との差が少なく，概念構造を作り出す上でSyntaxの方がむしろ補助的な役割を演じている。

Conceptual Parserの主要なメカニズムはExpectationである。Parser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ないながら概念構造を作り上げてゆく。過程はdeterministicに行なわれ，Syntactic Parserがよく行うような，文法的にあいまいな文の構造の枚挙は行なわない。

Conceptual Parser McELIはImterpreterの名前が示すように，それ自身には入力文から概念構造を作り上げるための情報をほとんど含んでいない。Parsingに必要な情報はすべて各々の語に付加されたPacketと呼ばれるリストの中に含まれている。このPacketはRequestと呼ばれる小さなプログラムの集合よりなっている。

McELIは入力文を一語づつ読んで，その語のPacketをスタックへ蓄え，スタック中の各々のRequestを実行することによってExpectationを実現し，概念構造を作り上げる。

2. Tiny-ELI

Packetを構成するRequestは次のTEST, ASSIGN, NEXT-PACKETのトリプルよりなっている。

```
((TEST expression)
 (ASSIGN variable expression v.e. ...)
 (NEXT-PACKET request request ...))
```

＊ McELI（イライ）の名前はEnglish Language Interpreterの頭文字を取ったものだが，二人の偉いイライ（人名）に因む。一人はElihu Yaleで Yale大学の名前になった人。他の一人はEli Whitneyで綿と種子を分離する機械を発明した人。大学の近くHamdenに工場の跡があり，大学へ通じる大通りの名前になっている。

Packet 内の一つの Request 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 Request が選択されるかは、各 Request の TEST の項が定める。語を読むことで得られた Packet はスタックへ蓄えられる際に APPEND で Push され、Packet の境界がなくなり Request が単位となる。語が読まれ、Packet がスタックへ加えられるたびに、スタック中の Request はプロダクションの動作を起す。即ち、最も新しく蓄えられた Request を最優先して順に古い方へと各 Request の TEST の項が評価され、T ならば ASSIGN の項が実行されて各変数に値がセットされて、その Request はスタックから取り除かれる。NEXT-PACKET を持つものは、その項に書かれた潜在的 Request が新しく評価を受ける Request へと昇格する。TEST の項を持たない Request は T と解釈されただちにファイヤする。

Homework で作った Tiny-ELI は McELI をさらに短く書き直したもので ほぼ同じ動作を行う。** 異っているのは Record type (後述) の考えを用いて概念構造を表現したことで、解析最終段階の変数代入の操作がほとんど不用になり省略したこと、概念構造の表現が述語の形式に近づいたので、deductive system DUCK との整合が良くなったことである。

3. Record type

Schank の CD-表現 は LISP 上にインプリメントする場合に、幾つか異った S-expression が用いられている。

John ⇔ ATRANS ← book ← Mary / John ---- (1)

(1) の CD-表現は (2) のようなキーパラメータ的な形に表わされること

```
\^ TINY-ELI ***

(DE PARSE (*SENTENCE*)
  (PROG (*STACK* *SYN-SBJ* *SYN-OBJ* *PART-OF-SPEECH*
         *SYN-TO* *SYN-FROM* *SYN-WITH*)
    (MSG "<INPUT> : " *SENTENCE*)
    (MAPC 'PROCESS-WORD (CONS '*START* *SENTENCE*))
    (MSG T T "<FINAL-EXPRESSION> : ")
    (SPRINT *CONCEPT*)))

(DE PROCESS-WORD (*WORD*)
  (APPENDPUSH *STACK* (GET *WORD* 'WD-DEF))
  (SETQ *STACK* (MAPPENDCAR 'EVAL-PACKET *STACK*)))

(DE EVAL-PACKET (*PACKET*)
  (MAPELMOR
    '(LAMBDA (X)
      (SELECTQ (CAR X)
        (TEST (AND (NULL (EVAL (CADR X))) (LIST *PACKET*)))
        (ASSIGN (MAPC 'EVAL (CDR X)) NIL)
        (NEXT-PACKET (CDR X))
        NIL))
    PACKET))
```

[Franz LISP + UCI function 版]

** McELI のプログラム及び動作は (10)(11) に詳しく述べられている。

*** APPENDPUSH : スタックへのプッシュを CONS の代りに APPEND でやる。
MAPPENDCAR : MAPCAR の APPEND 版。
MAPELMOR : リストの要素に函数を APPLY して OR をとる。

```
(ATRANS (ACTOR  John)
        (OBJECT book)
        (TO     Mary)
        (FROM   John)) ... (2)
```

が多いが，ここでは deductive system（後述）との整合を考え，論理述語に近い表現（Record type）を用いている。Schank の primitive ACTion ATRANS, PTRANS などが作る CD 構造を仮りに FRAME と呼ぶことにする。今，ACTion の conceptual roles[6] を ACTOR, OBJECT, TO, FROM, INSTrument, CAUSE として (3) 式を評価すると FRAME という Record type が定義される。(1)

```
(RECORD-TYPE FRAME
  (ACTION ACTOR OBJECT
   TO FROM INST CAUSES))
                        --- (3)
```

Record type FRAME の定義により次のような形で CD 構造を扱うことができるようになる。

```
(SETQ AA
  (FRAME 'ATRANS 'John 'book
   'Mary 'John ()() )) --- (4)
```

Record type の定義を行うことで，同時に，次のような type 付データを操作する函数が定義される。

```
(ACTION:FRAME AA)  → ATRANS
(ACTOR:FRAME AA)   → John
(OBJECT:FRAME AA)  → book
```

Record type 付データを扱う上で便利な他の函数は ":=" である。":=" は使われ方によって次のような幾つかの仕ちきを持つ。

```
(:= X 3)            は (SETQ X 3)
(:= (CAR X) 3)      は (RPLACA X 3)
(:= (CDR X) (FOO 'A))
      は (RPLACD X (FOO 'A))
(:= (GET X 'AGE) 13)
      は (PUTPROP X 13 'AGE)
```

として仕ちき，

```
(:= (OBJECT:FRAME AA) 'ball)
```

を評価すると AA にセットされた FRAME の OBJECT の項が book から ball に変る。

4. Requests

Tiny-ELI の本体は語に付加された Packet をスタックに蓄え，スタック内の Request を評価する動作を定めているだけで，実際の Parsing を行うための情報はすべて Request の中に含まれる。Tiny-ELI は最初に，これから入力される文の性質をおおまかに定める dummy の語 *START* を入力文の先頭に付加して処理を開始する。

START の Packet は (5) に示すようにこれから入力される文が平叙文の文法的性質を持つことを予測させるための情報である。即ち，最初に noun phrase が来て，次いで verb が来て，さらに noun phrase が来ることを示している。

```
(DP *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*)))))))))   --- (5)
```

CS470のhomeworkは次の10語を組み合せ，できるだけ多くの文を作り，正しくparseすることが要求されている。

John, Mary, Boston,
plane, gun, medicine,
to, with,
take, beat

英文としての自然さは別にして，組み合せを考えるとS1～S13のような文が得られる。いずれもtookを動詞にしているが，CD表現はかなり異ったものになる。INSTRUMENTとRESULTを除いたCD構造のtop levelを考えると次のようになる。

S1: John ⇔ PTRANS ←o John -[→ plane (TO Boston)

S2: John ⇔ PTRANS ←o (John / Mary) -[→ plane

S3: John ⇔ ATRANS ←o (OWNERSHIP : Plane) -[→ John

S4: John ⇔ GRASP ←o gun

S5: John ⇔ PTRANS ←o gun -[→ Boston

S6: John ⇔ PTRANS ←o Mary -[→ Boston

S7: John ⇔ DO

S8: John ⇔ INGEST ←o medicine -[→ INSIDE of John

S9: John ⇔ PTRANS ←o medicine -[→ B.

S10: John ⇔ PTRANS ←o medicine -[→ M.

S11: John ⇔ ATRANS ←o Boston ?

S12: plane ⇔ PTRANS ←o John -[→ Boston

S13: plane ⇔ PTRANS ←o gun -[→ B.

```
(:= S1 '(JOHN TOOK A PLANE TO BOSTON *PD*))
(:= S2 '(JOHN TOOK THE PLANE WITH MARY *PD*))
(:= S3 '(JOHN TOOK THE PLANE WITH A GUN *PD*))
(:= S4 '(JOHN TOOK A GUN *PD*))
(:= S5 '(JOHN TOOK THE GUN TO BOSTON *PD*))
(:= S6 '(JOHN TOOK MARY TO BOSTON *PD*))
(:= S7 '(JOHN TOOK MARY WITH THE GUN *PD*))
(:= S8 '(JOHN TOOK MEDICINE *PD*))
(:= S9 '(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*))
```

```
(DP TOOK
    WD-DEF
     (((ASSIGN (:= *PART-OF-SPEECH* 'VERB)))
      ((TEST (AND *SYN-SBJ*(C-ISA *SYN-SBJ* 'PERSON)))
       (NEXT-PACKET
        ((TEST (AND *SYN-OBJ*(C-ISA *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* (C-ISA *SYN-WITH* 'PERSON)))
           (ASSIGN
            (:= (RESULT:FRAME *CONCEPT*)
                (FRAME 'PTRANS *SYN-SBJ* *SYN-WITH*
                       (TO:FRAME *CONCEPT*)()()())))))
          ((TEST (AND *SYN-WITH* (C-ISA *SYN-WITH* 'WEAPON)))
           (ASSIGN
            (:= (RESULT:FRAME *CONCEPT*)
                (FRAME 'ATRANS *SYN-SBJ*
                       (C-PROJECTION 'OWNER-SHIP *SYN-OBJ*) *SYN-SBJ*
                       ()()()))
            (:= (INST:FRAME *CONCEPT*)
                (FRAME 'DO *SYN-SBJ* ()()() *SYN-WITH*
                       (STATE 'MENTAL (C-PROJECTION 'OWNER *SYN-OBJ*)
                              -5 0))))) ))
    . . .
```

--- (6)

Schank の CD 理論では INSTRUMENT は "もの" の概念ではなくて CD-構造になっており，RESULT と共に CD-構造を次々につないで概念のネットワークをなす。(6)

入力文 S1～S13 に対応する CD-構造を作り出すためには "took" の packet の定義が重要になる。(6) に packet の一部を示している。第3行の ASSIGN はただちに実行され，現在の品詞が動詞であることを示す。第4行の TEST は文法的な主語 *SYN-SBJ* の概念が PERSON であるときに条件が満され，6行目以後の NEXT-PACKET が Active な Request となる。6行目の TEST は文法的対象 *SYN-OBJ* が設定され，しかも，それが乗物の概念であるを条件が満される。入力文が S1 の場合この Request がファイアして，主概念構造 *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 Request が Active になる。

5. Inferencer

Inferencer は Conceptual Parser により得られた CD 構造を入力として，推論される別の CD 構造を得る。推論により得られる CD 構造を再び，Inferencer の入力にすると次々に別の CD 構造が得られ，Inference explosion が起る。Inference explosion は人の思考の過程から要求されてくるように思われ，より深く思考の研究を進めるため，自然な文脈の研究が重要になり，MARGIE から SCRIPT の研究へと発展したようである。

home work のテーマは Inferencer を作り，先の 10 words parser からの CD 構造の出力を入力として，inference explosion を起して見ようというものであった。

(Scenario)

S1 の例であれば，John は Boston 行きの飛行機に乗ったのであるから，結果として John は飛行機の中に居ることになり

John ⇔ LOC (plane (TO Boston))

が推論され，飛行機が故障でもしない限りやがて Boston へ到着するので，

plane (TO Boston) ⇔ LOC (Boston)

となり，John はその飛行機に乗っていたので，

John ⇔ LOC (Boston)

となり，また John は Boston で何かやろうと思って，目的をもって Boston へ行ったのであるから，

John ⇔ DO
⇕
LOC (Boston)

も推論される。DO の内容が MIT の AI ラボを訪問したのであれば，その結果として

John ⇔ MENT. ST (+)

となる。

S8 の例であれば John は薬を飲んだので，INSTRUMENT として

John ⇔ PTRANS ← medicine
→ (MOUTH-OF John)
→ (HAND-OF John)

が付加されており，それを可能にした行為は

John ⇔ GRASP ← medicine

である。また，John が薬を飲む原因

となったのは，多分病気をしていたからであろうと推測されるので

John ⇔ PHYS.ST (−)

が得られ，また，飲んだからやがて良くなるであろうということも予想される。

John ⇔ PHYS.ST (+)

このように与えられたCD構造からその原因となるCD構造，あるいは結果，動機や目的，手段，未来の予測，その行為を可能にした条件などのCD構造を推論することができる。このような推論の規則をATRANSやPTRANSのようなPrimitive ACTに関して作ることができれば広範囲に適用できるものになるが，一方，medicineやairplaineなどの特性に深くかかわる推論規則は，際限なく規則を複雑にせねばならない可能性が生じるので，知識の形式化自体を見なおす必要が生じてくる。

Interencerは最初LISPで書いたのであるが，同じ秋学期の人工知能の授業（CS270 Drew McDermott）でTheorem prover ASPTPとそれに関連しDUCKを学んだので，DUCKの紹介をかねて，DUCKルールで推論規則を書くことにする。

6. Deductive System DUCK

DUCK[2]はPLANNERやPrologのようなAI言語が持つRelational Data BaseをインプリメントしたLISPルーチンのセットで，Prologと比べるとより述語計算に近く，ルールの実行順序に意味を持たせていない。従ってカットオペレータはない。Backward chainingによる推論の他に，与えられた事実や規則から前方向に演繹を行ない，定理としてDBに加えて行くForward chainingの機能を持っている。また導かれた定理がどの事実やどの規則によりサポートされているかを示すData dependencyメカニズムを持っている。ある事実やある規則が取り消されると，Data dependencyメカニズムはそれらによりサポートされていた定理をすべて取り除く。またこのメカニズムは"教授ならばPHDを持たないことが証明されないかぎりPHDを持つと推論して良い"というようなDefault reasoningの機能も含んでいる（特定の事実の次如によりサポート）。

また素晴しいことには，DUCKはテンポラルロジックに現れるような少しづつ異った多重の世界（DB）を実現することができる。各々のDBはData poolと呼ばれ（DP-PUSH DP*）と言う函数で簡単に別の世界へ移ることができる。移った世界はもとのDBをコピーして作るのではなく，Data beadsメカニズムと呼ぶ方法で，DBの内の異なりだけを蓄え，インクリメンスを実現している。

Prologと異ってDUCKはLISPルーチンのセットなので，LISPとの相性が非常に良い。LISPからDUCK述語を操作できるだけでなく，DUCK述語からLISPルーチンをForward，Backward chainingで呼ぶことができる。大変面白く感じたのは，Data dependencyの定理取り消しメカニズムと関連して，if-erase-demonを設定できることである。このデモンは特定の定理や特定の述語パタンで監視させることができ，それが取り消されるときどこかを動き出すもので，消去される定理からのForward chainingを行うことができる（応用例(9)）。

DUCKとLISPの相性が良いことはプログラミングにおいて，それぞれ，述語計算と函数計算の仕事の分担を行える。

8. DUCKルールでのCD推論

DUCKにおいてimplicationの記号はforward chainなら"→"となり，backward chainなら"←"となる。

PTRANSが起ればOBJECTの位置は必然的にPTRANSのTOになるということを規則にすると次のようになる。

```
(RULE PTR1-CONSEQ
  (PTRANS ?ACTOR ?OBJ ?TO)
  → (LOCATE ?OBJ ?TO))
```

このルールで推論が起る場合，PTRANSとLOCATEの関係はルール名PTR1-CONSEQで保たれる。

ある目的地行きの乗り物に乗れば，やがてその乗り物も乗ってる人(物)も目的地に着くことを規則にする。

```
(RULE PTR2-NEXT
  (PTRANS ?ACTOR ?OBJ
      (?VEHICLE (TO ?DEST))
  → (AND (LOCATE ?VEHICLE ?DEST)
         (LOCATE ?OBJ ?DEST)))
```

人が薬を飲むのは病気が原因にたっていることを規則にする。

```
(RULE INGESTMED1-CAUSE
  (AND (INGEST ?ACTOR ?OBJ)
       (ISA ?OBJ MEDICINE))
  → (PHYS.ST ?ACTOR -))
```

病人が薬を飲めば良くなることを規則にする。

```
(RULE INGESTMED2-RESULT
  (AND (INGEST ?ACTOR ?OBJ)
       (ISA ?OBJ MEDICINE)
       (PHYS.ST ?ACTOR -))
  → (PHYS.ST ?ACTOR +))
```

病人がJohnならば(PHYS.ST John -)と(PHYS.ST John +)が同一のDBに混在することになる。それぞれサポートを見れば有意味であるが，前述のData poolを用いて別世界へ入れることもできる。

おわりに

言語は本来，頭脳内情報の外部表現であるので，特に言語の意味に関する諸々の問題は，頭脳という情報処理のメカニズムとの関連において解決さるべきである。従来の字面を追ってきた言語研究に比べ，AIの分野の言語研究は大変興味深い。新しい言語についての考えを国語研究所の中へ取り入れたいと考え，Yale大学のAI-projectに滞在した。心良く滞在を許可して戴き，学習と研究の機会を与えて下さった Roger Schank 教授，色々と御指導戴いた Drew McDermott 教授，Christopher Riesbeck 博士，大学院生の David Littleboy 氏に心から感謝する。

参考文献

(1) Charniak, E. Riesbeck, C. McDermutt D. A.I. programming, Lawrence Erlbaum, 1980
(2) McDermott. D. DUCK manual, Yale, 1982
(3) Riesbeck, C. An expectation-driven production system for N.L. understanding, in Pattern directed inference system, Academic Press, 1978
(4) Schank, R. etal MARGIE, IJCAI-3, 1973
(5) Schank, R, Colby, M. Computer model of thought and Language, W.H. Freeman Co, 1973
(6) Schank, R. Conceptual information processing, north-holland, 1975
(7) Schank, R., Abelson, P., Scripts, plans goals and understanding, Lawrence Erlbaum, 1977
(8) Schank, R. Dynamic memory Cambridge, 1983
(9) Tanaka, T. Representation and analysis of electrical circuits in a deductive system IJCAI-8, 1983
(10) 安西，佐伯，難波 LISPで学ぶ認知心理学2. 東京大学出版会，1982
(11) Schank, R. Riesbeck, C. Imside computer understanding Lawrence Erlbaum, 1981

〔付録〕 Parser と Inferencer の動作例
(DEC 2060, TOPS 20, TLISP
Yale 大学 Computer Science)

```
*(INFERENCE S3)

Input : (JOHN TOOK A MEDICINE *PD*)

Final-CD (CD-pool = *CD-0*) :
CD1   :  (INGEST (ACTOR (PERSON (NAME (JOHN))))
                 (OBJECT (EDIBLE-THING (MEDICINE)))
                 (TO (INSIDE (PERSON (NAME (JOHN)))))
                 (FROM (MOUTH (PERSON (NAME (JOHN))))))

Paraphrase mode

Generated Sentence from CD1 :
        (JOHN ATE THE MEDICINE *PD*)
        (THE MEDICINE WAS EATEN BY JOHN *PD*)

Infered-CD from CD1  (CD-pool = *CD-1*) :
CD2   : <INSTR>  (PTRANS (ACTOR (PERSON (NAME (JOHN))))
                         (OBJECT (EDIBLE-THING (MEDICINE)))
                         (TO (MOUTH (PERSON (NAME (JOHN))))))
CD3   : <RESULT> (PTRANS (ACTOR (PERSON (NAME (JOHN))))
                         (OBJECT (EDIBLE-THING (MEDICINE)))
                         (TO (INSIDE (PERSON (NAME (JOHN))))))
CD4   : <RESULT> (LOCATE (OBJECT (EDIBLE-THING (MEDICINE)))
                                                 (NOT T))
CD5   : <RESULT> NIL
CD6   : <RESULT> (HEALTH (OF (PERSON (NAME (JOHN))))
                                          (FROM 'X-) (TO 'X+))
Generated Sentence from CD2 :
        (THE MEDICINE WAS MOVED TO THE MOUTH OF JOHN *PD*)
        (JOHN MOVED THE MEDICINE TO THE MOUTH OF JOHN *PD*)

Infered-CD from CD2  (CD-pool = *CD-2*) :
CD7   : <RESULT> (LOCATE (OBJECT (EDIBLE-THING (MEDICINE)))
                         (AT (MOUTH (PERSON (NAME (JOHN))))))
CD8   : <RESULT> (LOCATE (OBJECT (EDIBLE-THING (MEDICINE)))
                                          (AT NIL) (NOT T))
CD9   : <ENABLE> (DO (ACTOR (MOUTH (PERSON (NAME (JOHN)))))
                     (OBJECT (EDIBLE-THING (MEDICINE))))
CD10  : <RESULT> (MENT-ST (OBJECT
                           (MOUTH (PERSON (NAME (JOHN)))))
                          (TO X+)
                          (FROM X))

Generated Sentence from CD3 :
        (THE MEDICINE WAS MOVED TO THE INSIDE OF JOHN *PD*)
        (JOHN MOVED THE MEDICINE TO THE INSIDE OF JOHN *PD*)

Infered-CD from CD3  (CD-pool = *CD-2*) :
CD11  : <RESULT> (LOCATE (OBJECT (EDIBLE-THING (MEDICINE)))
                         (AT (INSIDE (PERSON (NAME (JOHN))))))
CD12  : <RESULT> (LOCATE (OBJECT (EDIBLE-THING (MEDICINE)))
                                          (AT NIL) (NOT T))
CD13  : <ENABLE> (DO (ACTOR (INSIDE (PERSON (NAME (JOHN)))))
                     (OBJECT (EDIBLE-THING (MEDICINE))))
CD14  : <RESULT> (MENT-ST (OBJECT (INSIDE (PERSON (NAME (JOHN)))))
                          (TO X+)
                          (FROM X))

Generated Sentence from CD4 :
        (THE MEDICINE IS NOT IN SOMEWHERE *PD*)
        (THERE IS NOT THE MEDICINE IN SOMEWHERE *PD*)

Infered-CD from CD4  (CD-pool = *CD-2*) :

Generated Sentence from CD5 :

Infered-CD from CD5  (CD-pool = *CD-2*) :

Generated Sentence from CD6 :
        (JOHN BECOMES HEALTY *PD*)
        (JOHN WAS SICK *PD*)

Infered-CD from CD6  (CD-pool = *CD-2*) :

Generated Sentence from CD7 :
        (THE MOUTH OF JOHN HAS THE MEDICINE *PD*)

Infered-CD from CD7  (CD-pool = *CD-3*) :

Generated Sentence from CD8 :
        (THE MEDICINE IS NOT IN SOMEWHERE *PD*)
        (THERE IS NOT THE MEDICINE IN SOMEWHERE *PD*)

Infered-CD from CD8  (CD-pool = *CD-3*) :

Generated Sentence from CD9 :
        (THE MOUTH OF JOHN WILL DO
                SOMETHING USING THE MEDICINE *PD*)

Infered-CD from CD9  (CD-pool = *CD-3*) :

Generated Sentence from CD10 :
        (THE MOUTH OF JOHN BECOMES HAPPY *PD*)

Infered-CD from CD10  (CD-pool = *CD-3*) :

Generated Sentence from CD11 :
        (THE INSIDE OF JOHN HAS THE MEDICINE *PD*)

Infered-CD from CD11  (CD-pool = *CD-3*) :
CD15  : <ENABLE>
     (INGEST (ACTOR (INSIDE (PERSON (NAME (JOHN)))))
             (OBJECT (EDIBLE-THING (MEDICINE))))

Generated Sentence from CD12 :
        (THE MEDICINE IS NOT IN SOMEWHERE *PD*)
        (THERE IS NOT THE MEDICINE IN SOMEWHERE *PD*)

Infered-CD from CD12  (CD-pool = *CD-3*) :

Generated Sentence from CD13 :
        (THE INSIDE OF JOHN WILL DO
                SOMETHING USING THE MEDICINE *PD*)

Infered-CD from CD13  (CD-pool = *CD-3*) :

Generated Sentence from CD14 :
        (THE INSIDE OF JOHN BECOMES HAPPY *PD*)
```