談話の意味表示

IBM 東京基礎研究所　羽尻 公一郎[1]

1　はじめに

談話の構造に対する計算アプローチは，モンタギュー意味論を出発点として様々な進化を遂げている．そのなかでも，談話表示理論は動的論理と呼ばれる計算を背景として談話記述の強力な方法論として確定しつつある．本稿ではこのような背景を踏まえて，談話表示理論の概説と計算機への実装について述べる．

2　談話表示理論

談話表示理論はもともとロバ文と呼ばれる言語学上の典型的な文脈に関する問題に端を発する．以下に，ロバ文の例を示す．

- I a farmer owns a donkey, he beats it.
- Every farmer who owns a donkey beats it.

これに対してロバ文に相当しない例を以下に示す．

- If a farmer owns a donkey, he is happy.
- Every farmer who owns a donkey is happy.

ロバ文でない例では，a donkey という不定名詞句は，普通の場合の解釈のとおり存在量子化に対応する意味解釈を持っている．それに対してロバ文の例では，同じ不定名詞句が通常の解釈とは異なり，存在量子化ではなく普遍量子化に対応する意味解釈を要求している．では，ロバ文に対する通常の認知活動はどのように記述し説明することが出来るのか，という問題に対する方向性を示唆するのが談話表示理論である．

談話表示理論では，談話の意味解釈を表すために談話表示構造という記述を用いる．談話表示構造はその談話に登場する談話指示子をまとめて列挙する部分と，それらの談話表示子に対する条件を列挙する部分とから成る．そして，談話表示子が談話表示構造に新しく導入される仕方には，以下のような規則がある．

[1] E-mail: khajiri@trl.ibm.co.jp

- 談話表示構造構成規則 1
  固有名詞と不定名詞句は，談話表示構造に新しい談話指示子を導入する．

- 談話表示構造構成規則 2
  代名詞は，談話表示構造に新しい談話表示子を導入するが，その談話表示子はすでに談話表示構造に導入されている談話表示子のうちのどれかと同一視されなければならない．

この二つの規則により，談話表示構造の導入と参照が可能となる．たとえば，

- Hajiri has a model of GUNDAM. He made it.

という2文で表現される談話において，第1文に対しては，
$\exists x \exists y\,[Hajiri(x) \wedge model-of-GUNDAM(y) \wedge have(x,y)]$
という論理式で表現できる談話表示構造を作成することになり，第2文に対しては，
$\exists x \exists y\,[Hajiri(x) \wedge model-of-GUNDAM(y) \wedge have(x,y) \wedge make(x,y)]$
という談話表示構造による参照が可能となる．もとのロバ文に関してこの談話表示構造の導入と参照を計算すれば，
$\forall x\,[x : farmer] \forall y\,[y : donkey]\,[(xownsy) \rightarrow (xbeatsy)]$
という解釈構造を得られる．

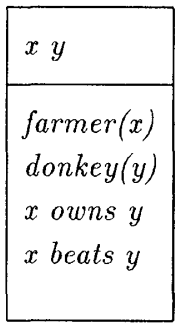
3 計算機への実装

前章で述べた談話表示理論の基本である談話表示構造の導入と参照は，計算機言語における新規のデータ構造の導入とそのデータ構造の参照という操作でほぼ実現できる．実際には，GPSG などの語彙駆動型文法の記述と単一化操作による実装がもっとも対応する．語彙駆動文法は CFG などの書き換え規則を語彙の内部に束性の集合として列挙することにより，あつかう文の長さに対して線形な計算量での生成・受理が可能であり，計算量の観点からも妥当な記述方法である．

実際には，Appendix に示すように個々の品詞体系に対して談話表示構造の雛型を用意し，それを記憶領域に挿入することで談話表示構造の導入とし，存在量子化，普遍量子化のそれぞれの表現に対しては先行する文との参照を記憶領域における単一化操作として実現することが出来る．その様子をつかむために DRT インプリメンテーションによってロバ文がどのような DRS に変換されるかを次に模式的に示しておく．プログラムの挙動の理解の助けになれば幸いである．

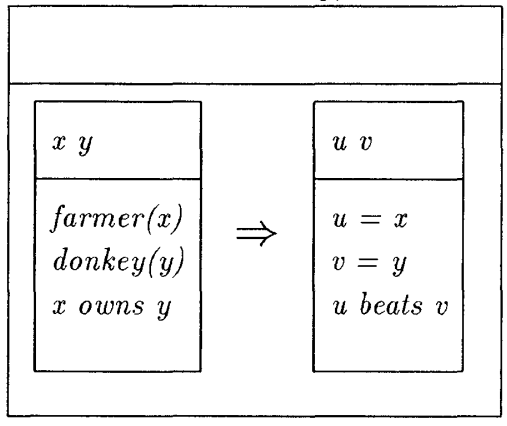
(a) A farmer owns a donkey. He beats it.

(a')



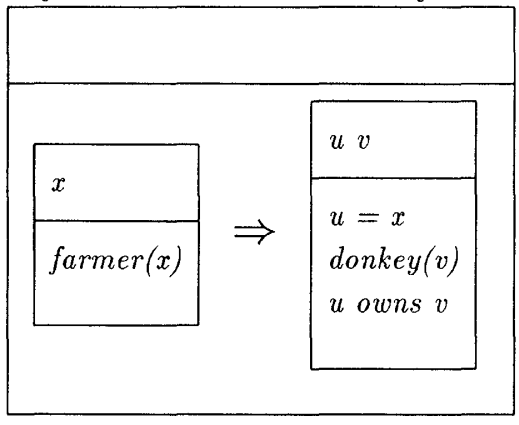
(b) If a farmer owns a donkey, he beats it.

(b')



(c) Every farmer who owns a donkey beats it.

(c')



Appendix に語彙駆動文法の prolog による具体的な記述をつけておく。これは語彙部分のみであり、DRT インプリメンテーション全体ではない.

4 おわりに

談話表示構造という組成の束による記述で談話の展開を把握する談話表示理論の概要と計算機への実装の概要について述べた．談話表示理論は形式意味論のひとつの到達点であり，複文（文脈）の形式的把握のもっとも直接的なものである．しかし，状況意味論など

の他の形式意味論と同じく，会話が暗黙裡に協調的であるという前提を要求する理論である．実際の自然言語による会話は，逸脱やいい間違い，言い直しなど，規則からの逸脱に本質があるとさえ言える．この規則からの逸脱を形式化するというパラドキシカルな問題に大してクリティカルな回答を出さない限り，談話研究は真の意味で理論足りえないだろう．ここに，認知活動の中心的存在たる言語活動の数理的解釈の問題と突破口の共存を認めることができる．

参考文献

1) 三藤 博,「談話と文脈」，岩波書店 (1999)

2) Michael Hess, Recent Developments in Discourse Representation Theory, Communication with Men and Machines, M. King ed. Geneva (1991)

3) William H. Smith, Handling Constrained Clauses in Discourse Representation Theory, Artificial Intelligence Research Report AI-1990-02, The University of Georgia, Athens, Georgia, (1990)

Appendix

prolog に実装された談話表示構造の導入と参照のホーン節による記述例（抜粋)。

```
n(N) --> [Form],
         { proper_noun(Form,lambda(I,Semantics)),
           append(Semantics,Con,NewCon),
           N = syn: (index:I  ::
                     class:proper) ::
               sem: (in:  [drs(U,Con)|Super] ::
                     out: [drs([I|U],NewCon)|Super]) }.

n(N) --> [Form],
         { common_noun(Form,lambda(I,Semantics)),
           append(Semantics,Con,NewCon),
           N = syn: (index:I ::
                     class:common) ::
               sem: (in:  [drs(U,Con)|Super] ::
                     out: [drs([I|U],NewCon)|Super]) }.

v(V) --> [Form],
         { transitive_verb(Form,lambda(A1,A2,Semantics)),
           append(Semantics,Con,NewCon),
           V = syn: (class:transitive ::
                     arg1:A1 ::
                     arg2:A2) ::
               sem: (in:   [drs(U,Con)|Super] ::
                     out:  [drs(U,NewCon)|Super]) }.

v(V) --> [Form],
         { intransitive_verb(Form,lambda(Arg,Semantics)),
           append(Semantics,Con,NewCon),
           V = syn : (class:intransitive ::
                      arg1:Arg) ::
               sem : (in:  [drs(U,Con)|Super] ::
                      out: [drs(U,NewCon)|Super]) }.
```

```
det(Det) --> ([a] ; [an]),
               { Det = sem:in:A,
                 Det = sem:res:in:A,
                 Det = sem:res:out:B,
                 Det = sem:scope:in:B,
                 Det = sem:scope:out:C,
                 Det = sem:out:C }.

det(Det) --> [every],
               { Det = sem:in:A,
                 Det = sem:res:in:[drs([],[])|A],
                 Det = sem:res:out:B,
                 Det = sem:scope:in:[drs([],[])|B],
                 Det = sem:scope:out:[Scope,Res,drs(U,Con)|Super],
                 Det = sem:out:[drs(U,[ifthen(Res,Scope)|Con])|Super] }.

det(Det) --> [no],
               { Det = sem:in:A,
                 Det = sem:res:in:[drs([],[])|A],
                 Det = sem:res:out:B,
                 Det = sem:scope:in:B,
                 Det = sem:scope:out:[DRS,drs(U,Con)|Super],
                 Det = sem:out:[drs(U,[neg(DRS)|Con])|Super] }.

det(Det) --> [not,every],
               { Det = sem:in:A,
                 Det = sem:res:in:[drs([],[])|A],
                 Det = sem:res:out:B,
                 Det = sem:scope:in:[drs([],[])|B],
                 Det = sem:scope:out:[Scope,Res,drs(U,Con)|Super],
                 Det = sem:out:
                     [drs(U,[neg(drs([],[ifthen(Res,Scope)]))|Con])|Super] }.
```