2B3-1

メンタルモデルによる論理推論の形式化と実装

An implementation and formalization of logical reasoning with mental models

杉本 雄太郎 *1　　佐藤 有理 *2
Yutaro Sugimoto　　Yuri Sato

*1慶應義塾大学「思考と行動判断」の研究拠点
The Research Center for Thinking and Behavioral Judgment, Keio University

*2東京大学大学院情報学環
Interfaculty Initiative in Information Studies, The University of Tokyo

計算認知モデリング (computational cognitive modeling) は，計算プログラムによる認知理論の実装であり，認知理論における心的表象やプロセスを具体化することができる．しかし，そのような実装は認知理論における仕様 (specification) を含むものではない．こうしたモデル実装と認知理論のギャップを埋めるために，本研究は形式仕様記述に親和的な強い静的型付き言語を用いた認知理論の実装を提案する．特に，Johnson-Laird らによって提案されたメンタルモデル理論の三段論法推論を事例として分析する．

1. はじめに

認知科学の研究では，心的表象やプロセスを，計算プログラムのデータ構造やアルゴリズムとの類似として捉えるのが主流である．とは言いつつも，認知理論の多くは自然言語で記述されており，幾らかの曖昧さを含むことが多い．曖昧さを取り除く一つの方法として，より直接的に認知理論を (典型的には) 計算プログラムとして実装することがある．こうしたアプローチは，計算認知モデリング (computational cognitive modeling) と呼ばれる [19]．Johnson-Laird ら [8, 10] によるメンタルモデルの研究はこのような認知科学の流れに沿って進められてきた．まず 1980 年代を中心に，文解釈と推論についての認知理論として自然言語の様式でまとめられ，その後，プログラム実装を含むモデル研究へと進展した (幾つかの Lisp プログラムが公開されている: 最近の研究 [9, 11] を参照)．このようなモデリング研究は，それまでの自然言語記述による認知理論よりも心的表象やプロセスを詳細にすることに貢献している．

しかし，認知モデリングにおけるプログラム実装は，心的表象やプロセスのメタ情報についての記載もある認知理論と同等の役割を担うとまでは言えない．McClelland [13] や Cooper and Guest [4] が強調しているように，計算プログラムによる実装には，認知理論における仮定 (assumption) や仕様 (specification) までは通常含まれないためである．実際，上記のメンタルモデル理論の Lisp による実装を見てみても，その仕様は適切な形では与えられていない．さらに重要なことに，この実装には「メンタルモデル」の定義は暗黙的に行なわれており，明示的な仕様になっているわけではない．これは，実装によるモデリング以前から，メンタルモデル理論の曖昧さとして指摘され続けてきた問題であり [6, 17]，モデリング研究においてもなおそれが解消されていないことが伺える．こうした背景を基に，本研究は，メンタルモデル理論を題材として，従来のプログラム実装による認知モデリングを超えて，より充実した仕様記述を持つ認知モデリングの方法について検討し，モデル実装と認知理論のギャップを埋めることを目指す．

連絡先: sugimoto@abelard.flet.keio.ac.jp (杉本雄太郎)
sato@iii.u-tokyo.ac.jp (佐藤有理)

2. メンタルモデル理論の概説

メンタルモデル理論の研究は三段論法推論の定式化から始められており，本研究も分析対象を三段論法推論に限定する．特に，理論の最新版 [1] とそれに対応する計算プログラム [14] に焦点をあてる．本章は，以降の分析の理解が助かる程度に，メンタルモデル理論について簡単に触れておく．

メンタルモデル理論の基本的アイデアは，人々は事態に対応するメンタルモデルを構築することによって文解釈を行い，反例モデルを構築することによって推論を行うというものである．自然言語を用いた場合，通常，以下のような事例をいくつか示すことによって認知理論が漠然と記述される．(1) メンタルモデルは，個体の性質を示す有限個のトークンから成り，それぞれの行が一つの個体を表す．*All A are B* から図 1 の 1 番目のモデルが作られる．a と b の二つのトークンから行が構成される場合，これは *A* であり *B* である個体があることを指している．ただし，トークン a には [a] のように角括弧が付いており，これはその集合がいまのトークンで網羅的に表現されていて，新たなトークンは追加されないことを意味する．一方で，トークン b には角括弧は付いておらず，トークンを新たに加えることによって (*B* であって *A* でない個体を含む) 別のモデルを作ることができる．(2) *Some C are not B* から図 1 の 2 番目のモデルが作られる．一つのトークン b のみから成る行は，*B* であって *C* でない個体があることを示す．二つのトークン c および–b から成る行は，トークン b に否定を表すデバイス “–” を付けることにより，*C* であって *B* でない個体があることを示す．(3) これらの前提モデルが，それぞれのモデルにおけるトークン b を同定して一つのモデルに統合される (図 1 の 3

前提モデル 1	前提モデル 2	統合モデル	代替モデル
[a] b	c –b	[a] b	[a] b c
[a] b	c –b	[a] b	[a] b c
	b	–b c	–b c
	b	–b c	–b c

図 1: メンタルモデル理論における三段論法推論：*All A are B, Some C are not B; therefore, Some C are not A.*

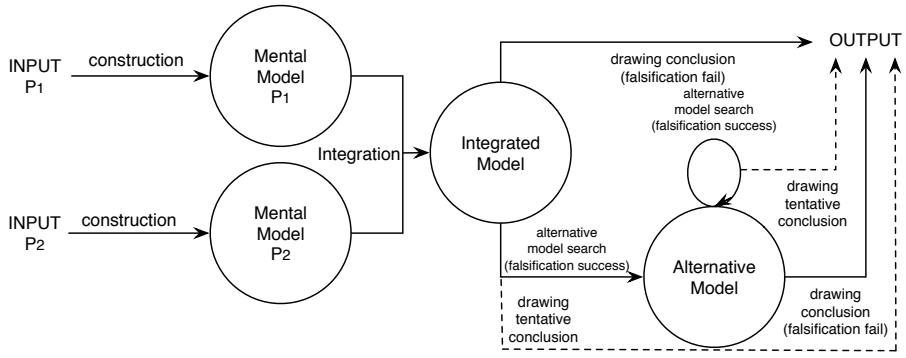

図 2: 三段論法推論システム

番目)．このモデルから，二つの仮結論 *Some A are not C* および *Some C are not A* が抽出される．(4) これらの仮結論に対する反例モデルを見つけようと，トークン c を新たに追加することで代替モデルが構築される (図 1 の 4 番目)．ここで，集合 *A* のすべてのトークンが *C* のトークンに対応しているので，*Some A are not C* は反駁される．一方で，*Some C are not A* は反駁されずに残り，妥当な結論として導出される．

3. モデリングの方針

前章のインフォーマルな認知理論に対して，Johnson-Laird らの Lisp による実装以外に，幾つかのモデリングがこれまで提供されてきた．本章では，それら先行研究を振り返りながら，本研究のモデリング方針を示す．

プログラム実装と認知理論のギャップを埋めるひとつの方法は，自然言語記述の認知理論を形式体系へ変換することである．これはつまり，認知モデルの実装よりも抽象的なレベルで表象やプロセスを再構成することで仕様を与える方法であり，これを形式的認知モデリング (formal cognitive modeling) と呼ぶ．Koralus and Mascarenhas [12] は，Johnson-Laird のメンタルモデル理論の命題推論に概ね基づく形式推論体系を構築した．概略は以下の通りである．(i) メンタルモデルに対応する代数構造，例えば $p \sqcup q$ (論理式 $p \wedge q$ に翻訳可能) が用意される．単位 p および q は命題を，演算子 $\sqcup$ は連言をそれぞれ表す．ここで，文解釈としては非古典的意味論 (モデル論的意味論でない) が与えられる．(ii) 新たな前提情報を既存情報に追加するための更新規則が，質問応答 (QA) システムに基づいて与えられる．(iii) 可能なモデルの探索による反駁推論を実行する操作規則が整備される．(iv) この体系を古典命題論理に変換することにより，体系の健全性および完全性が示される．この仕事は，我々が知る限り，メンタルモデル理論における命題推論の形式化に成功した実質初めての研究である (メンタルモデル推論とオイラー図推論の比較については [15] を参照)．これにより一般的な仕方で，メンタルモデル推論の正しさを保証することができ，計算の複雑さなどについて他の形式推論モデルと比較分析を行うことも可能となる．しかし，こうした形式モデルは，具体的な実装とは独立であり，その点で個別の記述様式には依存しない抽象的な記述である．そのため，抽象的・数学的レベルの仕様は提供しているものの，前段落で指摘したメンタルモデルの定義といった認知理論における特定の表現様式に関わる問題を扱うことが難しいと考えられる．

本研究で我々は，上記の形式的モデリングとの対応を念頭におきながら，仕様記述に親和的なプログラミング言語での実装を提案する．我々は，こうしたアプローチを仕様指向計算認知モデリング (specification-aware computational cognitive modeling) と呼ぶことにする．先駆的な仕事として Cooper らによる COGENT プロジェクト [3] が挙げられる．彼らは宣言型言語である Prolog に制御構造を追加した言語 (“Sceptic” と呼ばれるオリジナル言語) を用いて，認知モデル実装をいくつか提供した．その中には，メンタルモデル理論の三段論法推論の実装も含まれる [2]．確かにこれは，それまでの Lisp や C のような言語による実装と比較すると，仕様記述が充実していると捉えることはできる．しかし，上で問題となっていたメンタルモデルの定義を明示的に含んでいない点などで不満が残る．そこで我々は，強い静的型付き言語による実装を行うことによって [5, 16]，メンタルモデルの定義をシステムの内に埋め込むことを試みる．また，仕様記述を充実させるために，代数的形式モデルへの変換 (表示的意味論) に適した表記のプログラミング言語であることが重要である．これらの理由から，我々は実装に Haskell [7] を用いることを提案する．

4. メンタルモデルによる推論システム

一般にモデルベースの推論のアルゴリズムは，タブローを用いることが多いが，メンタルモデルによる推論アルゴリズムは反例モデルの探索というアイデアが共通していること以外はそれと大きく異なっている．メンタルモデルによる三段論法の推論システム全体は 図 2 で示すように，主に (1)〜(5) のプロセスで記述できる．

1. 前提のメンタルモデルを構築 (constructing mental models of premises)
2. 前提モデルの初期モデルへの統合 (integrating premise models into an initial model)
3. 初期モデルから仮結論を導出 (drawing a tentative conclusion from an initial model)
4. 反駁により代替モデルを構築 (constructing alternative model by falsification)
5. 最終結論を応答 (responding a final conclusion)

4.1 メンタルモデル推論システムの言語

三段論法の言語は，制限された自然言語 (controlled natural lanugage) の一種と考えられ，その文法は文脈自由文法により定義可能である．ここでは Haskell の構文を用いてこれを示す (図 3)．三段論法のためのメンタルモデルは図 4 のように定義

```
type Parser a b = [a] -> [(b,[a])]

symbol :: Eq a => a -> Parser a a
symbol c []                 = []
symbol c (x:xs) | c == x    = [(x,xs)]
                | otherwise = []

pS,pNp,pNegNp,pPred,pNegPred      :: Parser String String
pTerm,pQuant,pNegQuant,pNeg,pCop  :: Parser String String
pS       = pNp <*> pPred <|> pNp <*> pNegPred
                         <|> pNegNp <*> pPred
pNp      = pQuant <*> pTerm
pNegNp   = pNegQuant <*> pTerm
pPred    = pCop <*> pTerm
pNegPred = pCop <*> pNeg <*> pTerm
pTerm    = symbol "A" <|> symbol "B" <|> symbol "C"
pQuant   = symbol "All" <|> symbol "Some"
pNegQuant = symbol "No"
pNeg     = symbol "not"
pCop     = symbol "are"
```

図 3: 三段論法の言語

される *1．メンタルモデルとはモデルのクラスであり，トークンの $m \times n$ $(m \geq 2, 3 \geq n \geq 1)$ 行列である．行 (row) あるいは個体 (individual) とは，トークンの有限列であり，各アトムはたかだか一回現われる．列 (column) あるいは性質 (property) とは，トークンの有限列であり，トークンには異なるアトムは現われない．角括弧 ([]) に囲まれたトークンが列に現われる場合は，否定アトムのみが追加可能である．

三段論法言語からメンタルモデル表現への変換は，最初に三段論法言語パーザにより抽象構文木に変換され，次に合成的意味論 (compositional semantics) によりメンタルモデル表現へと変換されることで行なわれる．

```
type MentalModel  = [Individual]
type Individual   = [Token]

data Token = AToken Atom | FToken Exh Atom
           | NToken Neg Atom | Nil
data Atom  = ASymbol | BSymbol | CSymbol

type Exh = Symbol
type Neg = Symbol
type Nil = []
type ASymbol = Symbol
type BSymbol = Symbol
type CSymbol = Symbol

type Symbol = Char

exh :: Symbol
exh = '*'

asymbol :: Symbol
asymbol = 'a'

bsymbol :: Symbol
bsymbol = 'b'

csymbol :: Symbol
csymbol = 'c'

neg :: Symbol
```

図 4: メンタルモデルの定義

たとえば，今 X，Y が名辞 A,B,C を表わすとすると，三段論法推論に用いられる 4 つの量化文は以下のように変換される：

All X are Y ⇓ $\begin{bmatrix} [x] & y \\ [x] & y \end{bmatrix}$

Some X are Y ⇓ $\begin{matrix} x & y \\ x & \\ & y \end{matrix}$

No X are Y ⇓ $\begin{bmatrix} [x] & -y \\ [x] & -y \\ [y] & \\ [y] & \end{bmatrix}$

Some X are not Y ⇓ $\begin{matrix} x & -y \\ x & -y \\ & y \\ & y \end{matrix}$

*1 網羅的表現 (exhaustive representation) は [1] においては，角括弧 [] で表現されているが，ここでは [14] に従いアスタリスク * で表わす．

4.2 初期モデルへの統合

2 つの前提は，中名辞トークンにより，初期モデルへと統合される．この統合プロセスは $integrate :: P \to P \to M$ という型を与えられる．(ただし，P は P_1, P_2 の前提の型であり，M はモデルの型を表わすものとする).

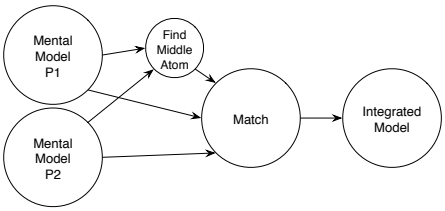

図 5: 統合モデルの形成プロセス

前提と名辞の並び換え：三段論法は前提の順序と名辞 (term) の配置により，いくつかの格 (figure) を持つため，初期モデルへの統合にはこれらの名辞の再配置および前提の入れ換えが前処理として必要である．これは以下の 4 つのパターンに分類される:

(1) P_1 の名辞の並びが AB で，かつ P_2 が BC の時，何もしない.
(2) P_1 の名辞の並びが BA で，かつ P_2 が CB の時，P_2 から開始する.
(3) P_1 の名辞の並びが AB で，かつ P_2 が CB の時，2 番目のモデルを引っくり返して追加する.
(4) P_1 の名辞の並びが BA で，かつ P_2 が BC の時，1 番目のモデルを引っくり返し，2 番目のモデルを追加する.

find-a-middle-atom: 中間アトム a を探す手続きは $find :: P \to P \to a$ という型シグネチャを持つ．これは，否定を含まないトークンについて，共通部分 (intersection) を取り，重複を除く操作であるので，いわば集合の共通部分を取る操作 $\cap$ と同種のものである．例： 二つの前提が図 1 の時, find $[\mathsf{a},\mathsf{a},\mathsf{b},\mathsf{b}]$ $[\mathsf{c},\mathsf{c},\mathsf{b},\mathsf{b}] \rightsquigarrow \mathsf{b}$ である.

match: 前提 P_1, P_2 と中間アトム a とのマッチは $match :: P \to P \to a \to M$ という型シグネチャを持つ．この手続きは前提を初期モデルに統合するために join を呼び出す.

join: この再帰的手続きは中間アトムと二つの個体を取り，二つの個体を一緒にして新たな中間アトムを設定する．これは $join :: a \to Indiv \to Indiv \to Indiv$ という型シグネチャを持つ.

4.3 結論の導出

結論の導出手続き conclude (図 6) は初期モデルを受け取り，否定トークンを持つかどうかで分配する．次に，述語 (all-isa, some-isa, no-isa, some-not-isa) により分配し，対応する回答を返す．conclude の型シグネチャは $conclude :: Subj \to Obj \to Indiv \to Indiv \to Ans$ である.

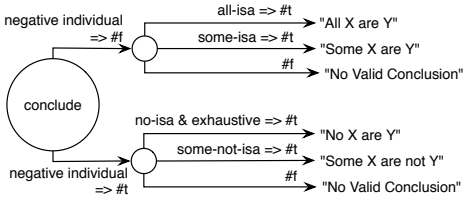

図 6: 結論の導出

all-isa は端名辞 X，Y を持つモデルを受け取り，全ての個体で subject アトム が object アトムであるとき，かつその時のみ，“All X are Y” を返す．例えば，端名辞が A および C であるモデル M : $\begin{bmatrix} [a] & b & c \\ [a] & b & c \end{bmatrix}$ が与えられた時，“All A are C” を返す.

some-isa は端名辞 X，Y を持つモデルを受け取り，少なくとも一つの個体が subject アトムと object アトムの両者において，

否定を含まない出現を含むとき，かつその時のみ，“Some X are Y” を返す．例：モデル M : $\begin{matrix}[a] & [b] & c \\ [a] & [b] & \\ & & c\end{matrix}$ が与えられた時，“Some A are C” を返す．

no-isa は端名辞 X，Y を持つモデルを受け取り，どの個体においても subject アトムが object アトムではないとき，かつその時のみ，“No X are Y” を返す．例：モデル M : $\begin{matrix}[a] & -b \\ [a] & -b \\ [b] & [c] \\ [b] & [c]\end{matrix}$ が与えられた時，“No A are C” を返す．

some-not-isa は端名辞 X，Y を持つモデルを受け取り，少なくとも一つの個体で subject アトムが object アトムなしで出現するとき，かつその時のみ，“Some X are not Y” を返す．例：モデル M : $\begin{matrix}[a] & b & c \\ [a] & b & c \\ & -b & c \\ & -b & c\end{matrix}$ が与えられた時，“Some A are not C” を返す．

4.4 反例モデルの構築

初期モデルが構築され，仮結論が導出されると，この仮結論を反駁するために代替モデルが構築される．反駁手続き falsify (図 7) はモデルを取り，否定トークンを持つかどうかで分配する．その後，述語 (breaks, add-affirmative, moves, add-negative) により，モデルを変更することを試みる．変更が成功した場合，代替モデルを返し，conclude を再帰的に呼び出し，失敗した場合，手続きは終了する．

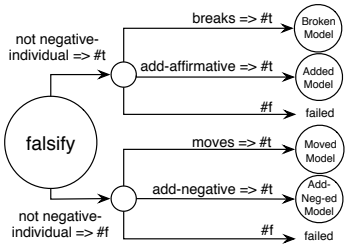

図 7: 反駁手続き

以下は falsify の主な部分手続きである：

breaks は網羅的でない中間名辞を持つ個体を探し，その個体を二つに分割する．さらに新たな (分割された) モデルを返すか，nil を返す．

例: モデル M が $\begin{matrix}a & b & c\end{matrix}$ の時，breaks M $\rightsquigarrow$ $\begin{matrix}a & b \\ & b & c\end{matrix}$.

add-affirmative は add^+ が成功した時，新しいモデルを追加されたモデルと共に返す．さもなくば，もし結論の型が “All X are Y” でない場合，または subject にトークンが追加可能でない場合，nil を返す．

例: M が $\begin{matrix}[a] & [b] & c \\ [a] & [b] & c\end{matrix}$ の時 $\mathsf{add}^+\ M \rightsquigarrow$ $\begin{matrix}[a] & [b] & c \\ [a] & [b] & c \\ & & c\end{matrix}$.

moves は，もし網羅的な端名辞が，一方の端名辞ないしそれらの否定と連結されていない場合 (すなわち結論の型が “No X are Y” の場合), あるいはもう一方の端名辞が網羅的であるか，結論の型が “Some X are not Y” の時，それらを結合する．さもなくば互いのうちの一つを join し，1 番目の端名辞が move できない場合，2 番目が move できたとしても nil を返す．

例: M が $\begin{matrix}[a] & -b \\ [a] & -b \\ [b] & -c \\ [b] & -c \\ [c] & \\ [c] & \end{matrix}$ の時 moves $M \rightsquigarrow$ $\begin{matrix}[a] & -b & [c] \\ [a] & -b & [c] \\ [b] & -c & \\ [b] & -c & \end{matrix}$.

この手続きが falsify により呼び出された場合は，neg-braking (breaks と同種の手続き) も同時に引数として渡される．

add-negative は追加された否定モデルを持つ新しいモデルを返すか，結論の型が “Some X are not Y” でない場合または subject に追加できるトークンが無い場合，nil を返す．

例: M が $\begin{matrix}[a] & b \\ [a] & b \\ & -b & c \\ & -b & c\end{matrix}$ の時 $\mathsf{add}^-\ M \rightsquigarrow$ $\begin{matrix}[a] & b & c \\ [a] & b & c \\ & -b & c \\ & -b & c\end{matrix}$.

謝辞

本研究は MEXT 私立大学戦略的研究基盤形成支援事業 (平成 24 年度〜平成 26 年度) ならびに JSPS 特別研究員奨励費 (25·2291) の助成を受けたものである．

参考文献

[1] Bucciarelli, M. & Johnson-Laird, P.N. (1999). Strategies in syllogistic reasoning. *Cognitive Science*. 23, 247–303.

[2] Cooper, R. (1992) A sceptic specification of Johnson-Laird’s “Mental Models” theory of syllogistic reasoning. *Tech. Rept. UCL-PSY-ADREM-TR4 (2nd ed.)*. Department of Psychology, University College London.

[3] Cooper, R., Fox, J., Farringdon, J., & Shallice, T. (1996). A systematic methodology for cognitive modelling. *Artificial Intelligence*, 85, 3–44.

[4] Cooper, R.P. & Guest, O. (2014). Implementations are not specifications: specification, replication and experimentation in computational cognitive modeling. *Cognitive Systems Research*, 27, 42–49.

[5] van Eijck, J. & Unger. C. (2010). *Computational Semantics with Functional Programming*. Cambridge University Press.

[6] Hodges, W. (1993). Horn clause logic 1992. In *Proceedings of the 4th UK Conference on Logic Programming*, *Workshops in Computing* (pp. 201–217), London: Springer.

[7] Jones, S.L.P. (Ed.). (2003). *Haskell 98 Language and Libraries: The Revised Report*. Cambridge University Press.

[8] Johnson-Laird, P.N. (1983). *Mental Models: Towards Cognitive Science of Language, Inference, and Consciousness*. Cambridge, MA: Harvard University Press.

[9] Johnson-Laird, P.N. (2013). Mental models and cognitive change. *Journal of Cognitive Psychology*, 25, 131–138.

[10] Johnson-Laird, P.N. & Byrne, R. (1991). *Deduction*. Hillsdale, NJ: Erlbaum.

[11] Khemlani, S. & Johnson-Laird, P.N. (2012). Theories of the syllogism: A meta-analysis. *Psychological Bulletin*, 138, 427–457.

[12] Koralus, P. & Mascarenhas, S. (2013). The erotetic theory of reasoning: Bridging between formal semantics and the psychology of deductive inference. *Philosophical Perspectives*, 27, 312–365.

[13] McClelland, J.L. (2009). The place of modeling in cognitive science. *Topics in Cognitive Science*, 1, 11–38.

[14] Mental Models and Reasoning Lab. (n.d.). Syllogistic reasoning code [Computer program]. Last Update: June 15, 2012, http://mentalmodels.princeton.edu/programs/Syllog-Public.lisp

[15] Mineshima, K., Sato, Y., Takemura, R., & Okada, M. (in press). Towards explaining the cognitive efficacy of Euler diagrams in syllogistic reasoning: A relational perspective. *Journal of Visual Languages and Computing*.

[16] Mitchell, J.C. (2003). *Concepts in Programming Languages*. Cambridge University Press.

[17] Stenning, K. & van Lambalgen, M. (2008). *Human Reasoning and Cognitive Science*. Cambridge, MA: MIT Press.

[18] Sugimoto, Y., Sato, Y., & Nakayama, S. (2013). Towards a formalization of mental model reasoning for syllogistic fragments. In *Proceedings of the 1st International Workshop on Artificial Intelligence and Cognition*, *CEUR Workshop Proceedings*, vol. 1100 (pp. 140–145).

[19] Sun, R. (Ed.). (2008). *The Cambridge Handbook of Computational Psychology*. Cambridge University Press.