A specification-aware modeling of mental model theory for syllogistic reasoning

Yutaro Sugimoto[1] **and Yuri Sato**[2]
[1]Department of Philosophy, Keio University [2]Interfaculty Initiative in Information Studies, The University of Tokyo
sugimoto@abelard.flet.keio.ac.jp, sato@iii.u-tokyo.ac.jp,

Abstract

Computational cognitive models can embody the structures and processes proposed in a cognitive theory. However, they do not necessarily reveal the theory's underlying assumptions and specifications. This study aims to bridge the gap between cognitive theory and its computational implementation, focusing on a case of mental model theory on human reasoning. Using a mathematics-based and statically-typed programming language (Haskell), we provide a specification-aware computational implementation of syllogistic reasoning with mental models.

keywords: Mental model theory, Type system, Specification, Cognitive modeling, Logical reasoning.

Introduction

The specification problem in cognitive modeling

In cognitive science research, mental representations and processes underlying human cognition have been treated in analogy with computer programs consisting of data structures and algorithms.However, it is not always easy to understand the relationship between the two in cognitive theories employing natural language for description. This occasionally leads to ambiguous formulations of cognitive theories. To remove these ambiguities, researchers have developed computer implementations of these theories.This type of modeling has revealed more detailed structures and processes than the existing formulations using natural language.

Recently, some researchers have questioned whether or not cognitive models should actually reveal theory *specifications*. McClelland (2009) pointed out that computational cognitive modelings are not full accounts of cognitive theories; implementations are not intended to embody the aspects of a theory's assumptions or specifications that are loosely defined in a cognitive theory. However, Cooper and Guest (2014) argue that *McClelland (2009) downplays the role of modeling in theory specification, and in particular the possibility that a model might embody a set of assumptions (p. 43).* To remedy this, they suggest that it is necessary to improve the current situation whereby modeling approaches are not suitable for theory specification, by bridging the gap between computational implementation and cognitive theory. The present study was conceived in a similar vein. We introduce a modeling method consisting of full-fledged descriptions of theory specifications: "type system based modeling."

A case of mental model reasoning

Cognitive studies involving the mental model theory, which was introduced by Johnson-Laird (1983; 1984), are appropriate examples with which to discuss the specification problem in cognitive modeling. The mental model theory is a cognitive theory of sentence interpretation and inference. The theory was first formulated for categorical syllogisms, but has now been extended to various domains of reasoning (for a review, see Khemlani & Johnson-Laird, 2013). Syllogism with quantificational sentences is one of the most elementary forms of natural language inference, and is important in the analysis of human reasoning (cf. Moss, 2008). The present study considers the domain of syllogisms only, focussing on the recent version (Bucciarelli & Johnson-Laird, 1999) and its corresponding computer implementation (Mental Models & Reasoning Lab, abbreviated as MMRLab.[1]).

[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
1st premise model	*2nd premise model*	*Integrated model*	*Alternative model*
All A are B	*Some C are not B*		

Fig 1: Syllogistic reasoning with mental models

Natural language description Syllogistic reasoning processes in mental model theory as described with natural language are briefly illustrated below. The basic idea underlying the theory is that people interpret sentences by constructing mental models corresponding to possibilities and make inferences by constructing counter-models. (1) Mental models consist of a finite number of tokens, denoting the properties of individuals. *All A are B* has a model illustrated on the leftmost side of Fig.1, where each row represents an individual. A row consisting of two tokens, a and b, refers to an individual that is *A* and *B*. The tokens with square brackets, [a], express that the set containing them is exhaustively represented by these tokens and that no new tokens can be added to it. A sequence of tokens without square brackets can be extended with new tokens so that an alternative model is constructed. (2) *Some C are not B* has a model illustrated on the second from the left of Fig.1. A row with a single token, b, refers to an individual that is *B* but not *C*. A row consisting of two tokens, c and –b, refers to an individual that is *C* but not *B*, by using "–" to denote negation. (3) These two models for premises are integrated into a single model, as shown in the second from the right in Fig.1. Two tentative conclusions, *Some A are not C* and *Some C are not A*, are extracted. (4) To search counterexample for them, an alternative model is constructed by adding new tokens (token c), as shown in the rightmost side of Fig.1. Since each token of *A* is corresponding to tokens of *C*, *Some*

[1]As described on p.14 of Khemlani and Johnson-Laird (2013), the major part of the program code for syllogistic fragments (MMRLab) still lives on in another implementation of the mental model theory, what is called *mReasoner*, which can cope with various domains beyond categorical syllogisms.

A are not C is refuted. Instead, *Some C are not A* survives.

Modeling implementations Following the theory informally described above, Johnson-Laird and his colleagues have developed several computational cognitive models, some of which have been published (p. 443 of Khemlani & Johnson-Laird, 2012). Their implementation of syllogisms is found in MMRLab. It is written in Common Lisp and a mental model is represented as a *list*, which is the most basic data structure in Lisp. For example, the 1st and 2nd premises models in Fig. 1 are represented as lists:

```
(((* A) (B)) ((* A) (B)))

(((C) (- B)) ((C) (- B)) ((B)) ((B)))
```

Their implementation of mental model theory was successful in embodying structures and processes proposed in the theory but it was unable to fully account for its specification. More concretely, it lacked a mathematical formalization of the concept of a "mental model." This ambiguity has resulted in various misunderstandings of the theory, and most of the controversy around the theory in cognitive science and logic has focused on this point (Braine, 1994; Hintikka, 1987; Stenning & Van Lambalgen, 2008). It is therefore important to understand the nature of the theory by addressing the specification problem.

Beyond the specification problem

Formal modeling A possible method to bridge the gap between implementation and theory is to convert natural language description theories to formal systems. This method is called *formal cognitive modeling*: structures and processes described in a theory are decomposed at a more abstract (mathematical) level than existing implementations (e.g., Arkoudas & Bringsjord, 2008; Bosse, Jonker, & Treur, 2006). Koralus and Mascarenhas's (2013) modeling is an example of this approach. They constructed a formal (propositional) inference system, called "the erotetic theory of reasoning", which essentially corresponds to mental model reasoning. (i) They started by converting from mental models to algebraic structures such as $p \sqcup q$, consisting of representational units (p and q standing for propositions) and operations ($\sqcup$ standing for conjunction), which can be translated to logical formulas such as $p \wedge q$. Here non-standard semantics, not model-theoretic semantics, were given for the interpretations of sentences. (ii) Update (erotetic) rules for adding the new premise, which is treated as an answer to the question in discourse, and operation rules for making inference as refutation were given. (iii) In this system, soundness and completeness for classical propositional semantics were shown via translation.

To our knowledge, Koralus & Mascarenhas's work is the first major formalization of mental model reasoning.[2] This approach has at least an advantage in analyzing computational complexities of solving tasks rather than predicting human performance data of them (for more details see e.g. Verbrugge, 2009; Isaac, Szymanik, & Verbrugge, 2014). However, this modeling is abstract in that it can be implementation-independent. Therefore it may not address problems caused by a particular representation in a cognitive theory. Thus, certain specifications, such as the mental models definition may not be provided at an appropriate level.

Specification-aware modeling An alternative way to bridge the gap between implementation and theory is to use a programming language suitable for specification descriptions, which we call *specification-aware cognitive modeling*. This approach was manifested in the seminal study of Cooper et al. (1996). According to them, (i) ordinary (Lisp and C) implementations themselves do not include the specifications of cognitive theories; (ii) formal theories of specifications, including mathematical descriptions, are not appropriate for bridging the gap between implementation and theory since they are implementation-independent. In our view, Johnson-Laird's mental model implementation in Common Lisp corresponds to case (i) and the formal modeling approaches of Koralus and Mascarenhas (2013) correspond to case (ii). Alternatively, Cooper et al. (1996) have provided some cognitive modeling implementations written by an executable specification language *Sceptic*, consisting of the declarative (logic programming) language Prolog, with the additional device of control structures. Their implementations include the mental model theory for syllogistic reasoning (for more details, see Cooper, 1992)[3]. Here, rewrite rules obey certain mathematical manipulations, such as rewriting from one term to another, as is the case in lambda calculus. This rewriting can be regarded as a specification in itself. Furthermore, an advantage of declarative language is that one can check the logical relationships between input and output representations without specifying a strategy of computation. (cf. sec.3 of Cooper & Guest, 2014).

This study put forward the line of specification-aware modeling. We try to give a "computational" (i.e. executable) semantics for the mental model theory. For the purpose of this research, we consider a reconstruction based on type system (cf. Mitchell, 2003). We refer to this approach as *type system based modeling*. Importantly, we chose a purely-functional programming language *Haskell* that has strong static typing and lazy evaluation strategy (Jones, 2003; Marlow, 2010). In contrast to the Common Lisp language used in Johnson-Laird and his colleagues' implementation (MMRLab), Haskell is a purely functional programming language with strong static typing. Explicit type annotations are useful for defining mental models within the system, and Haskell encourages to pro-

[2]Recently, Clark (submitted) straightforwardly provided a specification of mental model theory using total and partial truth functions, which correspond to exhaustive and non-exhaustive models.

[3]Based on Sceptic, the COGENT environment for cognitive modeling has subsequently been developed. An application to mental model theory is found in chap.5 of Cooper (2002). It is intended to provide theory specifications by reconstructing information-processing models such as box-and-arrow diagrams.

gram in a declarative style. This property can contribute to reasoning about the program itself (e.g. to give a denotational semantics for it Haftmann, 2010; Vytiniotis et al., 2013). It is important not only to guarantee the fact that a program is executable (there is no syntactical error), but also to verify whether a program works as intended by theorists (the aspect of semantics.) Note that this brief paper focuses on a part of the long-term study. Just a relatively weak formal verification (static type-checking based on explicit typing) is available here. To provide a formal verification by a theorem prover or other semantic verifications leave for future work.

An Implementation of the Mental Model Reasoning System

An outline of the reasoning system is given in Sugimoto, Sato, and Nakayama (2013). The system is portrayed as a conceptual diagram Fig. 2 that shows translations from one model to another by the following steps: 1. constructing mental models of premises, 2. integrating these premise models into an initial (integrated) model, 3. drawing a tentative conclusion from the initial model, 4. constructing alternative models by falsification, 5. producing a final conclusion.

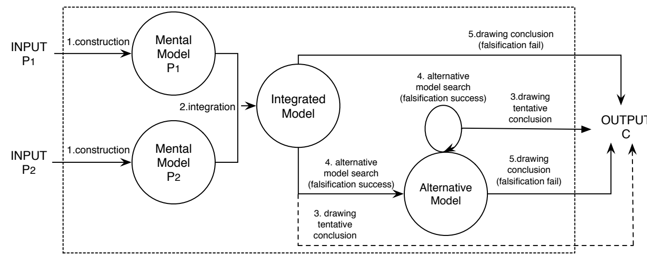
INPUT P1
1.construction
Mental Model P1
INPUT P2
1.construction
Mental Model P2
2.integration
Integrated Model
4. alternative model search (falsification success)
3.drawing tentative conclusion
Alternative Model
4. alternative model search (falsification success)
3.drawing tentative conclusion
5.drawing conclusion (falsification fail)
5.drawing conclusion (falsification fail)
OUTPUT C

Fig 2: Diagram for syllogistic reasoning system

Steps 1 and 5 involve IO actions. Steps 2, 3 and 4 consist of stateless (no side effects) functions only.

Mental model construction

The syllogistic language (input) is considered as a kind of controlled natural language, and its grammar can be defined by BNF^{+}. We define a combinator parser to parse this language instead of using a standard bottom-up parser (MMRLab.) An implementation of the language and the parser is given in Code 1:

```
pS,pNp,pNegNp :: Parser String String
pPred,pNegPred :: Parser String String
pTerm,pQuant,pNegQuant :: Parser String String
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"

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 = []
succeed :: b -> Parser a b
succeed r xs = [(r,xs)]
failp :: Parser a b
failp xs     = []
token :: Eq a => [a] -> Parser a [a]
token cs xs | cs == take n xs = [(cs,drop n xs)]
            | otherwise       = []
         where n = length cs
satisfy :: (a -> Bool) -> Parser a a
satisfy p []               = []
satisfy p (x:xs) | p x       = [(x,xs)]
                 | otherwise = []
```

Code 1: The definition for syllogistic language and its parser

```
data MToken = AToken Atom |
  FToken Exh Atom | NToken Neg Atom | Nil
data Atom  = ASymbol | BSymbol | CSymbol
type Symbol = Char

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

type MModel = [Indiv]
type Indiv  = [Token]
type Token  = [Symbol]

exh :: Symbol
exh = '*'
asymbol :: Symbol
asymbol = 'a'
bsymbol :: Symbol
bsymbol = 'b'
csymbol :: Symbol
csymbol = 'c'
neg :: Symbol
neg = '-'
```

Code 2: The definition for mental model components

The mental models for syllogistic reasoning are defined in Code 2[4]. A *mental model* is a *class of models*[5] s.t. $m \times n$ matrix (multi-list) of *tokens*. A *row* or an *individual* of a mental model is a finite sequence of tokens (*model*) where each atom occurs at most once. A *column* or a (*property*) of a mental model is a finite sequence of tokens where tokens containing different atoms cannot occur. If square bracketed tokens occur in a column, only negative tokens can be added. The translation from the syllogistic language to mental model representations is performed by monadic parsing (a recursive descent parsing technique well known in functional programming community. See, e.g. Van Eijck and Unger 2010). The parser for syllogistic language constructs abstract syntax trees and then converts mental model representations by the following compositional semantics (Fig. 2). As an example, let X,Y denote terms A,B,C, the four types of syllogistic sentences can be translated to mental models as follows:

All X are Y	Some X are Y	No X are Y	Some X are not Y
$\Rightarrow \begin{array}{cc} [x] & y \\ [x] & y \end{array}$	$\Rightarrow \begin{array}{cc} x & y \\ x & \\ & y \end{array}$	$\Rightarrow \begin{array}{cc} [x] & -y \\ [x] & -y \\ & [y] \\ & [y] \end{array}$	$\Rightarrow \begin{array}{cc} x & -y \\ x & -y \\ & y \\ & y \end{array}$

[4]In Bucciarelli and Johnson-Laird (1999) "exhaustive model" is represented in square brackets, here we use the "*" symbol as used elsewhere (MMRLab.)

[5]For a treatment of a mental model as a class of models, see Barwise (1993).

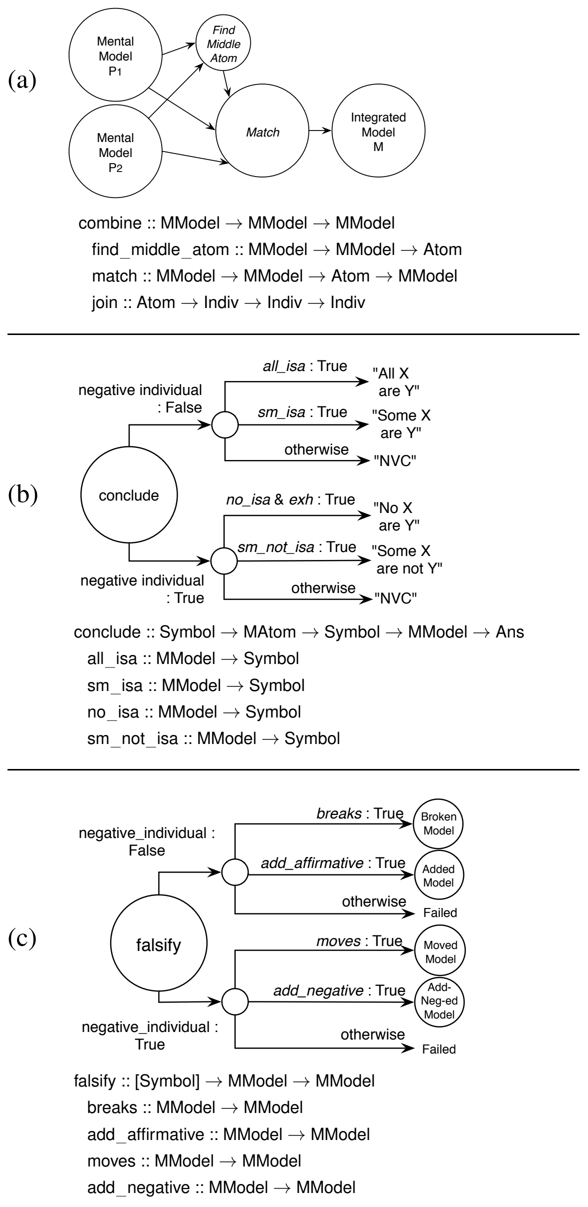
(a)
Mental Model P1
Find Middle Atom
Mental Model P2
Match
Integrated Model M
combine :: MModel → MModel → MModel
find_middle_atom :: MModel → MModel → Atom
match :: MModel → MModel → Atom → MModel
join :: Atom → Indiv → Indiv → Indiv
(b)
negative individual : False
all_isa : True
"All X are Y"
sm_isa : True
"Some X are Y"
otherwise
"NVC"
conclude
no_isa & exh : True
"No X are Y"
sm_not_isa : True
"Some X are not Y"
negative individual : True
otherwise
"NVC"
conclude :: Symbol → MAtom → Symbol → MModel → Ans
all_isa :: MModel → Symbol
sm_isa :: MModel → Symbol
no_isa :: MModel → Symbol
sm_not_isa :: MModel → Symbol
(c)
negative_individual : False
breaks : True
Broken Model
add_affirmative : True
Added Model
otherwise
Failed
falsify
moves : True
Moved Model
add_negative : True
Add-Neg-ed Model
negative_individual : True
otherwise
Failed
falsify :: [Symbol] → MModel → MModel
breaks :: MModel → MModel
add_affirmative :: MModel → MModel
moves :: MModel → MModel
add_negative :: MModel → MModel

Fig 3: Process diagrams and their type information: (a) Integration, (b) Drawing a conclusion, (c) Falsification

Integrating Premises into Initial Model

We give a description of the integration process of premises into an initial model via mid term tokens (Fig. 3-a.) The integration function `combine` takes two models (premises) P_1, P_2 and returns an integrated model M with the help of find_a_middle_atom. The type signature for this function is $integration :: MModel \rightarrow MModel \rightarrow MModel$. This is implemented as follows:

```
combine :: MModel -> MModel -> MModel
combine mod1 mod2 =
  match mid_atom mod1 mod2
    where mid_atom = find_middle_atom mod1 mod2
```

Reordering and Switching Since syllogisms have several "figures" according to the order of their premises and term arrangements, the actual integration should occur after the preprocesses of reordering terms and switching premises. This preprocessing consists of the following four patterns: (1) If the term order of P_1 is AB and P_2 is BC, nothing happens. (2) If the term order of P_1 is BA and P_2 is CB, integration starts with P_2. (3) If the term order of P_1 is AB and P_2 is CB, the second model is swapped round and added. (4) If the term order of P_1 is BA and P_2 is BC, the first model is swapped round and the second model added.

Finding a middle atom The function `find_middle_atom` has a type signature `find_middle_atom :: MModel → MModel → Symbol`. The implementation of this is similar to a set intersection operation for the affirmative tokens (tokens that do not contain negatives.) An example is when two premises are presented as in Fig.1, $\{a,a,b,b\} \cap \{c,c,b,b\} = b$. The following is an implementation:

```
find_middle_atom :: MModel -> MModel -> Symbol
find_middle_atom mod1 mod2
  | null mod1 = []
  | not (null mid) = mid
  | otherwise = find_middle_atom (tail mod1) mod2
  where mid = find_middle_from_ind (head mod1) mod2
```

Match The function for matching premises P_1, P_2, and middle atom a has the type signature `match :: MModel → MModel → Symbol → MModel`. This function recursively calls `join_` to join the premises to an integrated model.

```
match :: Symbol -> MModel -> MModel -> MModel
match mid_atom mod1 mod2
  | null mod1 = mod2
  | not (null (find_poslis_in_indiv mid_atom $ head mod1)) =
    joining mid_atom (head mod1) mod2 :
    (match mid_atom
           (tail mod1)
           (remove_indiv (find_poslis_in_mod mid_atom mod2) mod2))
  | otherwise = head mod1 : (match mid_atom (tail mod1) mod2)
```

Join The recursive function `join_` takes a mid atom and two individuals, and joins two individuals together setting the new mid to exhausted if either the first individual or second individual were exhausted. The type signature of `join_` is: `join_:: Symbol → Indiv → Indiv → Indiv`

```
join_ :: Symbol -> Indiv -> Indiv -> Indiv
join_ mid_atom indiv1 indiv2 =
  if exhausted $ find_poslis_in_indiv mid_atom indiv1
  then indiv1 ++ remove_lis (find_poslis_in_indiv mid_atom indiv2) indiv2
  else remove_lis (find_poslis_in_indiv mid_atom indiv1) indiv2
```

Drawing a Conclusion from a Model

Drawing a conclusion (Fig.3-b) is a function that takes an integrated (initial) model and dispatches whether it contains negative token or not. Based on the predicates (all_isa, some_isa, no_isa, and sm_not_isa) it then dispatches further and returns *corresponding answers* (action). conclude has type signature: `conclude :: Symbol → Symbol → Symbol → MModel → [Symbol]` [6]. If the predicates return `False`, then it returns `"no valid conclusion"`. The below is an implementation:

[6]Note: since possible conclusions have term order: Subj-Obj and Obj-Subj, `conclude` is executed twice. For simplicity, we omit the second execution of `conclude`.

```
conclude :: Symbol -> Symbol -> Symbol -> MModel -> [Symbol]
conclude subj mid_atom obj model =
  if not (negative_individual model)
  then if all_isa subj obj model
       then "all " ++ [subj] ++ " are " ++ [obj]
       else if sm_isa subj obj model
            then "some " ++ [subj] ++ " are " ++ [obj]
            else "no valid conclusion"
  else if no_isa subj obj model &&
          ((exh_ subj model && exh_ obj model) ||
           (exh_ mid_atom model && exh_ subj model) || (exh_ obj model))
       then "no " ++ [subj] ++ " are " ++ [obj]
       else if sm_not_isa subj obj model
            then "some " ++ [subj] ++ " are " ++ "not " ++ [obj]
            else "no valid conclusion"
```

The following are sub functions called by `conclude`:

all_isa takes a model *M* that has the end terms X, Y and returns True iff all subjects are objects in individuals in a model, then *conclude* returns the answer All X are Y. This has a type signature: `all_isa :: Symbol` $\rightarrow$ `Symbol` $\rightarrow$ `MModel` $\rightarrow$ `Bool`. For example, if a model *M*: $\begin{array}{ccc} [a] & b & c \\ [a] & b & c \end{array}$ is given, where the end terms are *A* and *C*, it returns the answer "All A are C." This is implemented as follows:

```
all_isa :: Symbol -> Symbol -> MModel -> Bool
all_isa subj obj model =
  sm_isa subj obj model && not (sm_not_isa subj obj model)
```

sm_isa takes a model *M* that has end terms X, Y and returns True iff at least one individual in the model contains positive occurrences of both subject and object atoms. Then conclude returns the answer "Some X are Y". This has the type signature: `sm_isa :: Symbol` $\rightarrow$ `Symbol` $\rightarrow$ `MModel` $\rightarrow$ `Bool`. For example, if a model: $\begin{array}{ccc} [a] & [b] & c \\ [a] & [b] & \\ & & c \end{array}$ is given where the end terms are *A* and *C*, it returns the answer "Some A are C".

```
sm_isa :: Symbol -> Symbol -> MModel -> Bool
sm_isa subj obj model
  | null model = False
  | not (null $ find_poslis_in_indiv subj $ head model) &&
    not (null $ find_poslis_in_indiv obj $ head model) = True
  | otherwise = sm_isa subj obj $ tail model
```

no_isa takes a model *M* that has the end terms X, Y and returns True iff no subject end term is object end term in any individuals in the model, conclude returns "No X are Y". This has a functional type: *no-isa* : $M \rightarrow A$. For example, if a model *M*: $\begin{array}{ccc} [a] & -b & \\ [a] & -b & \\ & [b] & [c] \\ & [b] & [c] \end{array}$ is given where the end terms are *A* and *C*, it then returns the answer "No A are C".

```
no_isa :: Symbol -> Symbol -> MModel -> Bool
no_isa subj obj model =
  sm_not_isa subj obj model && not (sm_isa subj obj model)
```

sm_not_isa takes a model *M* that has the end terms X, Y and returns True iff at least one subject occurs in individuals without an object, then conclude returns "Some X are not Y". This has a functional type: *sm_not_isa*: $M \rightarrow A$. For example, if a model *M*: $\begin{array}{ccc} [a] & b & c \\ [a] & b & c \\ & -b & c \\ & -b & c \end{array}$ is given where the end terms are *A* and *C*, it returns the answer "Some A are not C".

```
sm_not_isa subj obj model
  | null model = False
  | not (null $ find_poslis_in_indiv subj $ head model) &&
    not (null $ find_poslis_in_indiv obj $ head model) = True
  | otherwise = sm_not_isa subj obj $ tail model
```

Falsification

Once the mental model theory constructs an initial model and draws a tentative conclusion, the theory, according to its rules, tries to construct an alternative model to refute the conclusion (the default assumption). The falsification function (Fig. 3-c) takes a model and dispatches whether or not it contains a negative token. Then, based on the predicates (breaks, add_affirmative, moves, and add_negative), it tries to modify the model. If successful, it returns an alternative model and calls conclude recursively. If it fails, this function terminates and conclude outputs a final conclusion. The below is an implementation of falsify:

```
falsify :: [Symbol] -> MModel -> MModel
falsify concl model =
  if not (negative_individual model)
  then if not (null br_model) then br_model
       else if not (null ad_model) then ad_model
            else []
  else if not (null mv_model) then mv_model
       else if not (null an_model) then an_model
            else []
  where
    br_model = breaks concl model
    ad_model = add_affirmative concl model
    mv_model = moves concl (neg_breaking concl model)
    an_model = add_negative concl model
```

Here are the main constructs of `falsify`:

breaks finds an individual containing two end terms with non-exhaustive mid terms, divides it into two, and then either returns new (broken) model or returns *nil*. Its type signature is: `breaks : MModel` $\rightarrow$ `MModel`. For example: when a model *M* is $\begin{array}{ccc} a & b & c \end{array}$, then breaks M $\rightsquigarrow$ $\begin{array}{ccc} a & b & \\ & b & c \end{array}$. This is implemented as follows:

```
breaks :: [Symbol] -> MModel -> MModel
breaks concl model =
  if falseif model newmod
  then newmod
  else []
    where newmod = breaking concl model
```

add_affirmative has the type signature: add_affirmative :: MModel $\rightarrow$ MModel. If add_affirmative succeeds, then it returns a new model with added item (added model), else it returns nil if the conclusion is not A-type ("All X are Y") or if there is no addable subject item. For example, if a given model *M* is $\begin{array}{ccc} [a] & [b] & c \\ [a] & [b] & c \end{array}$, then add_affirmative *M* $\rightsquigarrow$ $\begin{array}{ccc} [a] & [b] & c \\ [a] & [b] & c \\ & & c \end{array}$.

```
add_affirmative :: [Symbol] -> MModel -> MModel
add_affirmative concl model =
  if not (a_conclusion concl) then []
  else if not (null subj)
       then if subj == first then [[first]] : model
            else if subj == last then model ++ [[[last]]] else []
       else []
    where
      subj      = addable (subject concl) model
      end_terms = find_ends concl model
      first     = head end_terms
      last      = head $ tail end_terms
```

moves has the type signature: moves :: MModel $\rightarrow$ MModel. If there are exhausted end items not connected to other end items or their negatives (i.e E-type ("No X are Y") conclusions), and if the other end items are exhausted, or O-type ("Some X are not Y") conclusions, then it joins them. Otherwise it joins one of each and returns nil if the first end item cannot be moved, regardless of whether or not a second one can be. E.g., if a given model M is

$$\begin{array}{ccc} [a] & -b & \\ [a] & -b & \\ & [b] & -c \\ & [b] & -c \\ & & [c] \\ & & [c] \end{array} \text{, then moves } M \rightsquigarrow \begin{array}{ccc} [a] & -b & [c] \\ [a] & -b & [c] \\ & [b] & -c \\ & [b] & -c \end{array} .$$

When this function is called by falsify, neg_breaking (similar procedure to breaks) is also called as an argument.

```
moves :: [Symbol] -> MModel -> MModel
moves concl model =
  if falseif model newmod
  then newmod
  else []
    where newmod = moving concl model
```

add_negative has the type the signature: add_negative :: MModel $\rightarrow$ MModel. It returns a new model with the added item (add_neged model), or returns nil if the conclusion is not O-type or if there is no addable subject item.
E.g., if a given model M is

$$\begin{array}{ccc} [a] & b & \\ [a] & b & \\ & -b & c \\ & -b & c \end{array} \text{, then add\_negative } M \rightsquigarrow \begin{array}{ccc} [a] & b & c \\ [a] & b & c \\ & -b & c \\ & -b & c \end{array} .$$

```
add_negative :: [Symbol] -> MModel -> MModel
add_negative concl model =
  if falseif model newmod
  then newmod
  else []
    where newmod = adding_neg concl model
```

Concluding Remarks

We have proposed *type system based modeling* as a novel approach in (specification-aware) cognitive modeling. To show the advantages of this approach, we have implemented the syllogistic reasoning system with mental models in Haskell (a popular pure functional statically typed programming language). Compared with other approaches, our implementation includes some aspects of theory specification such as mental model definitions and type information for each process. Our type system based modeling sheds light on the ambiguity problem of mental model theory, which has been discussed at the crossroad between cognitive science and logic. In the modeling work done so far, we have analyzed mental model reasoning for syllogistic fragments. However, our work is not limited to this. There is plenty of scope for further work in several kinds of mental model reasoning (Khemlani & Johnson-Laird, 2013; Johnson-Laird, Byrne, & Tabossi, 1989; Johnson-Laird & Byrne, 1989); more generally, various cognitive activities involving propositions taking some semantic values.

Acknowledgements

This study was supported by MEXT-Supported Program for the Strategic Research Foundation at Private Universities and Grant-in-Aid for JSPS Fellows (25·2291). The authors are grateful to Sangeet Khemlani for helpful comment and Micah Clark for opportunity to read his paper under submission.

References

Arkoudas, K., & Bringsjord, S. (2008). Toward formalizing common-sense psychology: An analysis of the false-belief task. In *PRICAI 2008* (pp. 17–29). Springer.

Barwise, J. (1993). Everyday reasoning and logical inference. *Behavioral and Brain Sciences*, *16*(2), 337–338.

Bosse, T., Jonker, C. M., & Treur, J. (2006). Formalization and analysis of reasoning by assumption. *Cognitive Science*, *30*(1), 147–180.

Braine, M. D. (1994). Mental logic and how to discover it. In *The logical foundation of cognition* (pp. 241–263). Oxford Univ Pr.

Bucciarelli, M., & Johnson-Laird, P. N. (1999). Strategies in syllogistic reasoning. *Cognitive Science*, *23*(3), 247–303.

Clark, M. H. (submitted). Mathematical description of sentential mental models theory: Reconstructing Johnson-Laird's mental models.

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.

Cooper, R. (2002). *Modelling high-level cognitive processes*. Psychology Press.

Cooper, R., Fox, J., Farringdon, J., & Shallice, T. (1996). A systematic methodology for cognitive modelling. *Artificial Intelligence*, *85*(1), 3–44.

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

Haftmann, F. (2010). From higher-order logic to haskell: there and back again. In *Proceedings of the 2010 ACM SIGPLAN workshop on partial evaluation and program manipulation* (pp. 155–158).

Hintikka, J. (1987). Mental models, semantical games and varieties of intelligence. In *Matters of intelligence* (pp. 197–215). Springer.

Isaac, A. M., Szymanik, J., & Verbrugge, R. (2014). Logic and complexity in cognitive science. In *Johan van Benthem on logic and information dynamics* (pp. 787–824). Springer.

Johnson-Laird, P. N. (1983). *Mental models*. Harvard Univ Press.

Johnson-Laird, P. N., & Bara, B. G. (1984). Syllogistic inference. *Cognition*, *16*(1), 1–61.

Johnson-Laird, P. N., & Byrne, R. M. (1989). Only reasoning. *Journal of Memory and Language*, *28*(3), 313–330.

Johnson-Laird, P. N., Byrne, R. M., & Tabossi, P. (1989). Reasoning by model: The case of multiple quantification. *Psychological Review*, *96*(4), 658–673.

Jones, S. L. P. (2003). *Haskell 98 language and libraries: the revised report*. Cambridge University Press.

Khemlani, S., & Johnson-Laird, P. N. (2012). Theories of the syllogism: A meta-analysis. *Psychological Bulletin*, *138*(3), 427-457.

Khemlani, S., & Johnson-Laird, P. N. (2013). The processes of inference. *Argument & Computation*, *4*(1), 4–20.

Koralus, P., & Mascarenhas, S. (2013). The erotetic theory of reasoning: Bridges between formal semantics and the psychology of deductive inference. *Philosophical Perspectives*, *27*(1), 312–365.

Marlow, S. (2010). *Haskell 2010 language report.* Retrieved from `http://www.haskell.org/onlinereport/haskell2010`

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

Mental Models & Reasoning Lab. (n.d.). *Syllogistic reasoning code [computer program].* Retrieved from `http://mentalmodels.princeton.edu/programs/Syllog-Public.lisp`

Mitchell, J. C. (2003). *Concepts in programming languages*. Cambridge University Press.

Moss, L. S. (2008). Completeness theorems for syllogistic fragments. *Logics for linguistic structures*, *29*, 143–173.

Stenning, K., & Van Lambalgen, M. (2008). *Human reasoning and cognitive science*. MIT Press.

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 Vol.1100*, 140–145.

Van Eijck, J., & Unger, C. (2010). *Computational semantics with functional programming*. Cambridge University Press.

Verbrugge, R. (2009). Logic and social cognition. *Journal of Philosophical Logic*, *38*(6), 649–680.

Vytiniotis, D., Peyton Jones, S., Claessen, K., & Rosén, D. (2013). Halo: Haskell to logic through denotational semantics. *ACM SIGPLAN Notices*, *48*(1), 431–442.