4B1-2

題材となるオブジェクトの抽象化による確率文章題の自動解答

Automatically Solving Probability Problems through the Abstraction of Object Types

神谷 翼　　松崎 拓也　　佐藤 理史
Tsubasa Kamiya　Takuya Matsuzaki　Satoshi Sato

名古屋大学大学院 電子情報システム専攻
Department of Electrical Engineering and Computer Science, Graduate School of Engineering, Nagoya University

A computer program that solves probability problems is presented. It solves problems by abstracting the type of the objects appearing in the problems. It converts the problem texts to an intermediate representation written in programming language. By executing the intermediate representation, our program simulates the probabilistic event described in the problem and derives the answer.

1. はじめに

2011 年，国立情報学研究所で,「ロボットは東大に入れるか」というプロジェクトが開始された．このプロジェクトは大学入試問題を計算機で解くことに挑戦するものである．

我々は，このプロジェクトにおいて，数学「場合の数・確率」分野の文章題に対する自動解答システムの開発に取り組んでいる．本システムは問題文をプログラミング言語で書かれた中間表現へと変換し，その中間表現を実行して試行をシミュレートすることで解答を得る．しかし，これまで開発したシステムでは，解くことができる問題タイプが，題材となるオブジェクトが「玉」である問題に限られていた．

これに対し本稿では，題材となるオブジェクトを限定せずに中間表現を生成する手法について検討する．この手法では，題材となるオブジェクトを推定し，それをもとに問題文にタグを付与して問題文を抽象化する．これにより，題材となるオブジェクトや試行を表す表記の違いによらず，中間表現を生成することができる．

以下，本稿は次のように構成されている．まず，2 節でシステムの概要を説明する．3 節で中間表現の構成，4 節で用いるタグについて，5 節で中間表現の生成方法を説明する．6 節では成果と今後の課題について述べる．

2. システム全体の概要

システムのおおまかな流れを図 1 に示す．この図の流れに沿って，以下の問題を例として，処理の流れを説明する．

> 赤玉が 3 個，青玉が 4 個が入った袋がある．袋から 3 個の玉を同時に取り出す．取り出した青玉の個数を X とする．X=2 となる確率はいくつか．

システムではまず問題文 (あるいは節) を，問題で定義された状況を記述する文 (object 文)，試行を記述する文 (event 文)，変数を定義する文 (variable 文)，答えを問う文 (question 文)，の 4 種類に分類する．例題の場合は以下のような分類になる．

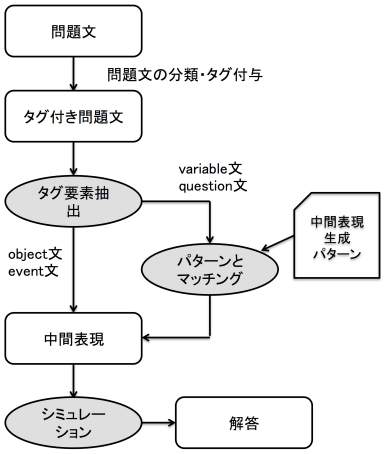

図 1: システムの流れ

> object 文: 赤玉が 3 個，青玉が 4 個が入った袋がある．
> event 文: 袋から 3 個の玉を同時に取り出す．
> variable 文: 取り出した青玉の個数を X とする．
> question 文: X=2 となる確率はいくつか．

さらに分類した問題文にタグを付与すると以下のようになる．個々のタグについては次節で説明する．

```
赤      玉      3 個    青      玉      4 個    が入った袋がある.
attr    object  num     attr    object  num

その袋から  3 個      の  玉      を  同時に      取り出す.
            takenum         object      taketype  verb

取り出した  青      玉      の  個数    を  X     とする.
<―――――――require―――――――>
            attr    object      func        var

X=2     となる  確率            はいくつか.
require
condition       questiontype
```

次にタグ情報を付与した問題文から中間表現へ変換する．object 文では,「赤玉 3 個」は中間表現では「' 玉',{'color'=>' 赤'} , 3」と表される．これはタグ情報を読み取ることで生成でき，オブジェクトを定義する関数の引数に代入する．

連絡先: 神谷翼，名古屋大学大学院 工学研究科 電子情報システム専攻，〒 464-8603 愛知県名古屋市千種区不老町 IB 電子情報館南棟 159 号室，052-789-4435，tubasa_k@nuee.nagoya-u.ac.jp

event 文では，試行タイプを表す情報を付与されたタグから取り出し，試行結果を列挙する中間表現を生成する．

variable 文や question 文では，タグ情報を取り出すとともに，文のパターンを過去問から抽出した中間表現生成パターンと照らしあわせ，確率変数を定義する中間表現，または，場合の数や確率などを求める中間表現を生成する (5 節).「取り出した青玉の個数を X とする」は,「取り出した <attr><object> の <func>」という文パターンに対応する中間表現と変数 X を対応付ける中間表現を生成する．また,「X=2 となる確率はいくつか」は，中間表現生成パターンをもとに，<condition> および <questiontype> のタグが付与された部分に対応する中間表現を生成し，それらを組み合わせることで，確率を求める中間表現を生成する．

これらの手順により，プログラミング言語で書かれた中間表現を生成すると，次のようになる．

```
1.  declareObjectList(
      declareObjects(' 玉',{'color'=>' 赤'} , 3) +
      declareObjects(' 玉',{'color'=>' 青'} , 4) )
2.  declareEvent(:takeSimultaneously , 3)
3.  defineVariable(
      'X', lambda{|elist|elist.attr('color', ' 青').length})
4.  findProbability(lambda{|elist|rev('X') == 2})
```

生成された中間表現を実行し，シミュレートを行うことで解答を得る．

3. 中間表現の構成

この節では中間表現の構成について，前節での中間表現の例 (1〜4) を用いて説明する．本システムは,「袋からオブジェクトを取り出す」方法でモデル化できる問題のみを対象としている．前節の例のように,「袋」に相当するものが明示的に現れる問題のほか，例えば「さいころをふる」では,「1 から 6 の数字が付けられたさいころの目」が袋に入っていると考えることで,「袋から取り出す」方法でモデル化することができる．

1. object 文

個々のオブジェクトは，その種類を表す名前と，属性名と属性値の組を用いて表現される．例えば，数字の 1 が書かれた赤玉は以下のように表される．

```
{name => ' 玉', attr => {'color' => ' 赤', 'number' => '1'}}
```

このように表されるオブジェクトを，仮想世界に定義するため，2 つのメソッドを用いる．declareObjects は，オブジェクトそのものを定義するメソッドである．引数にはそれぞれ「オブジェクトの名前」「属性名と属性値」「個数」を指定する．例題の場合,「赤」の属性を持つ「玉」が「3」個,「青」の属性を持つ「玉」が「4」個定義している．declareObjectList は,「仮想の袋」をつくるメソッドであり，引数にオブジェクトの情報を指定する．

2. event 文

event 文では，object 文で作成したオブジェクト群に関する試行を定義する (declareEvent)．このメソッドの引数には試行タイプと取り出す個数・回数を指定する．現在可能な試行のタイプには「同時に取り出す」「オブジェクトを順に 1 つずつ取り出し，毎回元に戻す」などが存在する．例題の場合，試行を定義する「declareEvent」関数の引数に，取り出し方 (takeSimultaneously) と取り出す個数 (3 個) を指定することで，試行結果を図 2 のようにすべて列挙する．この試行結果は，1 つの試行結果を配列の形で表現する．

```
[ [{id=1, name=' 玉', attr={'color' => ' 赤'}},
        {id=2...}{id=3...}], [...], ...]
```

図 2: 試行結果オブジェクト

3. variable 文

defineVariable は，第 1 引数に変数名，第 2 引数にプログラム (lambda...) を指定し，試行結果によって値が定まる変数 (確率変数) と，それに対応するプログラムを内部的に保存するメソッドである．第 2 引数は，event 文で作成した個々の試行結果 (elist) に対する関数を指定する．例の場合，elist に含まれるオブジェクトから，属性 'color' の値が' 青' であるものを全て取り出し，その個数 (length) を計算する関数を指定している．

4. question 文

引数に条件を指定し，その条件に合う試行結果を数えることで確率や場合の数などを求めるいくつかの関数が定義してある．findProbability は確率を求める関数である．引数の lambda 式の中で用いられている rev('X') は，variable 文で定義した変数に対応するプログラムを呼び出すメソッドである．例題では，X=2 となるもの，つまり 2 で列挙した試行結果のうち，青球の個数が 2 個となるものの数から解答を導く．

4. 問題の抽象化のためのタグセット

この節では問題文を抽象化する際に用いるタグセットについて説明する．本稿では，予備校各社が出版しているセンター試験形式の問題集をタグセットを設計する際の開発データとして用いた．具体的には，河合 (32 問)，駿台 (16 問)，代ゼミ (14 問)，Z 会 (21 問) の合計 74 問のうち,「袋から取り出す」モデルでシミュレートすることが可能な問題 46 問を開発データとして用いた．

表 1 は問題文に付与するタグの一覧である．以下，表のタグの詳細を説明する．

attrname1, attr1, num1「赤色の玉が 2 個ある」であれば，attrname1=' 色'，attr1=' 赤'，num1='2'，となる．

attrname2, attr2, num2「赤色の玉が 2 個ある．その玉には数字の 1，2 が書かれている」のように，オブジェクトに 2 つの属性がある場合，これらのタグが適用され，attrname2=' 数字'，attr2='1，2'，num2='1'，となる．

termination　「赤球を 2 回取り出した時点で試行を終了する」のように，ある条件を満たした時に終了するタイプの試行の場合，その終了条件を表す表現に付与される．

questiontype「確率」「場合の数」「期待値」などの，求める値を表す表現に付与される．

require「〜となる <questiontype> を求めよ」の，〜の部分を require とする．

when「X が 2 以下のとき，Y=0 とする」，のように問題文に場合分けがある場合の,「のとき」「ならば」などの表現に when タグを付与する．

表 1: タグセット一覧

object 文	
object	オブジェクトの名前
attrname1	属性 1 の名前
attr1	属性 1
num1	属性 1 をもつオブジェクトの個数
attrname2	属性 2 の名前
attr2(default=nil)	属性 2
num2(default=nil)	属性 2 をもつオブジェクトの個数
event 文	
verb	試行を表す動詞
takenum(default=1)	一回の試行で取り出す個数
taketimes(default=1)	試行回数
taketype(default=同時)	試行方法 (同時，1 個ずつ，など)
termination(default=nil)	終了条件
variable 文および question 文	
var	変数の名前 (アルファベット)
questiontype	求めるもの (場合の数，確率など)
require	条件を定義する文
when	条件文 (〜のとき，〜ならば)
require の中身	
formula	変数を用いた数式
condition	変数を用いた条件
define	変数を定義する数式
func, pred	知識表現 (4. 節にて説明)

formula「a+b が 3 となる <questiontype> はいくつか」など，演算を表す表現に付与する．
condition「a=b となる <questiontype> はいくつか」など，条件を表す表現に付与する．
define「a=10 とする」など，変数の値を定義する表現に付与する．
func 数学的演算など，オブジェクトが持つ属性についての操作に関する表現に付与するタグである．(和・積・差・絶対値・割った余り・最大値・種類の数など)「取り出したカードに書かれている数字の和を X とする．」は,「数字」という属性に対して,「足し合わせる」操作を行うものである．
pred オブジェクトが持つ属性についての条件に関する表現に付与するタグである (奇数・偶数・〜の倍数・同じ・異なるなど).「カードに書かれている数字が奇数となる確率はいくつか.」では,「数字」という属性が条件を満たす確率が問われている．

func や pred タグを付与する表現には，それぞれ対応する中間表現パターンを用意しておく．上記の例のように，「<attrname> の和」という形の表現で，<attrname> の部分が「数字」であった場合，これに対応する中間表現は「elist.attr('number').inject(:+)」である．シミュレーションによって生成した試行結果オブジェクト (elist) から,「数字」の属性を取り出し (attr('number'))，すべてを足し合わせる (inject(:+)) ことを意味している．

「<attrname> の種類の数」に対応する中間表現は，「elist.attr([attrname]).uniq.length」である．[attrname] のように [ ] でタグ名が囲ってあるものは，そのタグに対応する中間表現を意味する．この中間表現は，elist からある属性 (attrname) の要素を取り出し，同じ要素は 1 まとめにして (uniq)，個数を数える (length)．

5. 中間表現の生成方法

この節では，4 節で説明したタグを付与し，中間表現を生成する方法について文タイプごとに述べる．

5.1 object 文

この節ではオブジェクトの特定方法を説明する．初めに問題文を構文解析し，形態素情報と係り受け関係を用いてオブジェクトを特定する．その方法は次のとおりである．(優先順位は 1>2>3)

1. (数字)+(助数詞) にかかる一般名詞のうち，最も出現回数の多いもの
2. 「○○ている」「○○がある」の形の動詞にかかる一般名詞のうち，最も出現回数の多いもの
3. 品詞が一般名詞であるもののうち，最も出現回数の多いもの

1 は,「2 個の赤玉」や「3 枚のカード」のように，オブジェクトは数量を表す語にかかることが多いことを利用している．2 は,「袋の中に 2 個の玉がある」「カードには 1 から 3 の数字が書かれている」のように，状況を表す「○○ている」「○○がある」の形の動詞にかかることが多いことを利用している．3 は，問題文中にオブジェクトそのものの登場回数が多いことを利用している．

object を定義する上で必要なものは,「オブジェクトの名前 (object)」「属性名 (attrname) と属性値 (attr)」「個数 (num)」である．これ以外のタグ情報は必須ではない．これらの情報を取り出すために,「<attr> の <object> が <num> 個」，などのパターンとのマッチングにより，どのオブジェクトがどの属性を持ち，いくつ存在するのか特定する．

5.2 event 文

event 文に用いるタグは表 1 の 5 つである．5 つのタグそれぞれに関しての抽出方法を述べる．
verb 本稿では，event は袋からオブジェクトを「取り出す」または「並べる」試行のみを考えている．このため，開発データでは，verb タグが付けられる動詞は「取り出す」「並べる」「(サイコロを) 振る」の 3 種類しか見られなかった．そのため自動でタグを付与することは比較的容易である．現時点では,「袋から取り出す」モデルの問題のみを扱うが，今後シミュレートできる試行タイプが増加することを考え，verb タグを設計している．
takenum 取り出す個数は，(数詞)+(助数詞) または「すべての」などの量を表すフレーズから特定できる．
taketimes 試行の回数の定義は，(数詞)+「回」というフレーズから特定できる．
taketype 試行方法は,「同時に」「一度に」「無作為に」「一列に」「(数字)+(助数詞) ずつ」「に戻す」「に戻さない」のうちどれにマッチするかによって決定する．
termination たとえば,「赤玉を 2 回取り出した時点で試行を終了する」という場合は，この文全体に termination タグを付与する．しかし，現時点で終了条件を中間表現に変換するプログラムはできていない．

event 文においては，付与されたタグ情報を取り出すだけでシミュレートに必要な情報がそろう．例えば,「3 個同時に取り出す」ことを意味する中間表現「declareEvent(:takeSimultaneously, 3)」では，下線部の 2 つの情報を taketype，takenum タグから取り出せばよい．

5.3 variable 文・question 文

variable 文は「require の主部 (reqS)」「require の述部 (reqV)」の 2 つに，question 文は reqS，reqV,「どのような値を求めるかの指示 (value)」の 3 つに分解できると仮定する．すなわち，question 文および variable 文は以下のパターンに従って分解される：

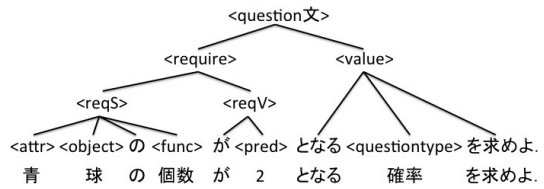

図 3: question 文の解析例

```
<variable> → <reqS><reqV>
<question> → <reqS><reqV><value>
```

reqS をさらに分解するパターンとしては，例えば以下のようなものがある：

```
<reqS> → <attr><object>の<func>
（例：青/attr　玉/object　個数/func）
```

このパターンが適用された場合，<reqS> に対応する中間表現 [reqS] は以下のように合成される：

```
[reqS] = lambda{
   |elist| elist.attr([attrname]).[func]}
```

ここで [attrname]，[func] は <reqS> の分解パターンの右辺の <attrname>, <func> から生成された中間表現を表す．同様に，<reqV> に対する規則は以下のようなものがある：

```
<reqV> → が <number> となる
[reqV] = lambda {|x| x == [number] }
<reqV> → が <pred> となる
[reqV] = [pred]
```

<value> に関する規則は

```
<value> → <questiontype> を求めよ
[value] = [questiontype]
```

であり，<questiontype> は以下のように関数名に対応する：

```
<questiontype> → 確率 | 期待値 | 場合の数 | ...
[questiontype] = findProbability
  | findExpectedValue | findCombination | ...
```

最後に <question> に対応する中間表現を以下の合成規則によって組み立てる：

```
[question] = [questiontype](lambda{
       |elist| [reqV].call([reqS])})
```

図 3 は，question 文「青球の個数が 2 となる確率を求めよ」に対する解析例を表している．図に示した分解の結果，<question> に対応する中間表現として以下のものが得られる：

```
findProbability(lambda{
  |elist| elist.attr('number').inject(:+) == 5}
```

現在用意されている中間表現生成パターンは，<reqS> は 35 パターン，<reqV> は 22 パターン，<value> は 12 パターンである．

表 2: オブジェクト推定結果

	○	△	×
1. 数詞	36	3	7
2. ている	26	3	5
3. 名詞の数	30	1	15
1〜3 併用	40	0	6

(△:○と×が同数)

表 3: 中間表現への変換の評価

	○	×
variable 文	18 文	17 文
question 文	76 文	65 文

6. 評価・検討

本システムは，題材となるオブジェクトの推定と，ラベルが付与された問題文から中間表現を生成することができる．しかし，タグの付与に関しては完全に自動化できていないため，タグ付与の評価はできない．よって 6.1 節では，オブジェクト推定の評価，6.2 節では，タグが付けられた問題文から中間表現を生成する方法の評価を行った．

6.1 オブジェクト推定

5.1 節で述べたオブジェクト推定方法の評価を行った．異なる手がかりを用いた 3 つの方法により問題文を解析し，オブジェクトの推定を行った結果が表 2 である．ただし，方法 2 では，「ている」の形の動詞がない問題が 12 問存在したた．3 つの方法を併用することで，46 問中 40 問に対してオブジェクトを正しく特定することができた．また，それぞれの方法を単体で用いた場合より高い精度で特定ができていることがわかる．

6.2 中間表現への変換

人でタグ付けを行った問題文を入力とし，正しい中間表現が出力されるものの割合を調べた．object 文と event 文は，タグが正しく付与されていれば多くの問題文において正しい中間表現が出力できるため，この評価には，variable 文と question 文のみを用いた．評価用データとして駿台模試 (10 回分) と代ゼミ模試 (10 回分) を用い，その結果を表 4 に示す．

6.2.1 出力できたものの検討

5.3 節で述べたように，中間表現生成パターンを reqS，reqV，value に分割する方法により中間表現を出力することができた問題文は，出力できた 94 文中 13 文であった．残りの 81 文は，中間表現生成パターンとタグ置き換えされた入力文が全く同じであり，分解する必要はなかった．

6.2.2 中間表現を出力できなかったもの

中間表現を出力できなかった原因として最も大きなものは，パターンの不足である．その他の原因によるものとしては，試行に名前がつけられているものが変換できなかった例が 7 文あった．例えば「操作 B を 1 回行った後の左端のカードの数字を X とする」という variable 文では試行自体を変数で表しており，現在のシステムではこれに対応していない．

知識 (func，pred タグに相当) として持っていなかったものも変換できない原因の一つである．例えば，「X が 5 で割ると 1 余る数である確率は」では，余りに関する表現を知識として持っていなかったため，変換することができなかった．

7. 結論

本稿では，題材となるオブジェクトを限定せずに中間表現を生成する手法について検討した．問題文にタグがつけられている状態では，半数以上の文を中間表現に変換することができるが，パターン数や知識量が少なく，マッチしないものが多く存在するという問題が残っている．また，「袋から取り出す」モデルでは対応できない問題など，シミュレータがもともと対応していない問題に解答するため，シミュレータの拡張も視野に入れなければならない．