数学確率文章題の自動解答システムの開発

神谷 翼 † 松崎 拓也 ‡ 佐藤 理史 ‡
† 名古屋大学 工学部電気電子情報工学科
‡ 名古屋大学大学院 工学研究科 電子情報システム専攻

1 はじめに

自然言語で書かれた数学の問題をコンピュータに解かせる研究は、これまで多くの試みがある。しかしながら「場合の数・確率」の問題に対する研究は 70 年代の Gelb による研究 [2] 以降大きくは進んでいない。

我々が現在開発しているシステムは、まず問題文をプログラム言語で書かれた中間表現へと変換し、次に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。この方式は、静的な関係の表現を得意とする論理式ではなく、状態の変化を指定する道具であるプログラミング言語を中間表現として用いることで世界の変化を自然に表現するアプローチであると言える。本稿ではシステムを、シミュレータ部と言語処理部に大別し、それぞれの成果と課題について述べる。

2 システムの全体像

以下の問題例を用い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したとき、3 個とも青玉である確率はいくつか。

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)、答えを問う文 (answer 文)、の 3 種類に分類する。例題の場合は以下のような分類になる。

> object: 赤玉 3 個、青玉 4 個が入った袋がある。
> event: 袋から 3 個の玉を同時に取り出したとき、
> answer: 3 個とも青玉である確率はいくつか。

次にそれぞれの文を中間表現に変換する：

```
declareObjectList(
    declareObjects(' 玉',{'color'=>' 赤'} , 3) +
    declareObjects(' 玉',{'color'=>' 青'} , 4) )
declareEvent( :takeSimultaneously , 3)
findProbability(lambda{|x|{x.attr('color',' 青',3)})
```

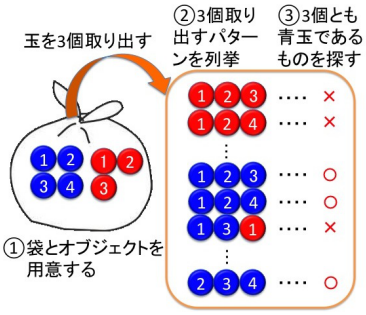

図 1: シミュレーションの流れ

ここで、declareObjectList は、問題で操作対象とするオブジェクトの集合を定義する関数、declareObjects は玉やカード、さいころといった操作対象を表すプログラム言語上のオブジェクトを生成する関数、declareEvent は、試行に関する情報を定義する関数、findProbability は、引数として与えた条件を満たす試行結果を得る確率を出力する関数である。上記の中間表現はそれぞれの関数の定義と併せることで実行可能な ruby プログラムとなり、以下のような流れでシミュレーションを行う (図 1)。

1. declareOcjectList 関数で「仮想の袋」を用意し、declareObjects 関数で属性 'color' が '赤' である '玉' を '3' 個、属性 'color' が '青' である '玉' を '4' 個入れる。
2. 合計 7 個の玉が入った仮想の袋から '3' 個の玉を '同時に'(takeSimultaneously) 取るパターンを全て列挙する。
3. findProbability 関数の引数に、条件として「取り出したオブジェクトの 'color' 属性は 3 つとも '青'」を指定し、2 で列挙したパターン数のうち、この条件に当てはまるものがいくつあるのかを数え、その比によって解答を得る。

以下 3 節ではシミュレータ部、4 節では言語処理部について詳細を述べ、5 節ではシステムの評価について述べる。

3　シミュレータ部

上述のように、シミュレータ部では仮想世界に置かれるオブジェクト群の定義、シミュレートされる試行タイプの定義、ある条件下での解答の出力の 3 ステップにより解答を得る。以下では各ステップについて詳細を述べる。

3.1　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'}}

また、コイン (投げ) は、表と裏をあらわすオブジェクトを 1 つずつ含む下記のようなオブジェクト群から 1 つを選び出す操作として表現される。

[{name => ' コインの面', attr => {'side' => ' 表'}},
{name => ' コインの面', attr => {'side' => ' 裏'}}]

3.2　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。現在可能な試行のタイプは以下の 4 種類である：

- 同時に取り出す (takeSimultaneously)：
  例題のように、オブジェクト群から一度に複数のオブジェクトを抽出する。
- オブジェクトを順に 1 つずつ取り出し、毎回元に戻す (takeSequentiallyWithBack)：
  コインやさいころが対象の場合は基本的にこのタイプの試行になる。
- オブジェクトを順に 1 つずつ取り出し、元に戻さない (takeSequentiallyWithoutBack)
- オブジェクトを順に 1 つずつ取り出し、戻し方をパラメータで制御 (takeSequentially)：
  例えば、「袋の中から玉を 1 個ずつ 3 回取り出す。赤玉を取り出したときは袋の中に戻し、それ以外のときには戻さない」場合、取り出した後のオブジェクト群を直接計算する関数 (lambda{…}) をパラメータとして以下のように与える：

```
declareEvent(:takeSequentially, 3,
    lambda{|v, set|
            v.attr['color'] == ' 赤' ?
                set : set.select{|x| x != v} } )
```

3.3　answer 文

オブジェクト及び試行の定義から定まる、場合の数，確率，期待値，最大値，最小値などの数量を求める。いずれの場合も、試行結果を列挙したパターンの中から条件に当てはまるものを数えることで計算を行う。求めたいものが場合の数ならば、当てはまったパターン数そのものが解答に、確率を求めるならば、場合の数を列挙したパターンの総数で割ったものが解答になる。期待値の計算の場合は、試行の結果によって値の定まる変量を別に記述しておく必要がある。以下に問題と中間表現の例を示す。

```
さいころを 1 回投げる。出た目の 2 倍を得点とする。このときの得点の期待値はいくらか。
score = lambda{|e| e.attr('number')*2}
findExpectedValue(score)
```

中間表現の 1 行目で変数 score は、玉の'number' 属性の値を 2 倍にして返す関数であり、findExpectValue は、score を用いて列挙されたパターンから期待値を計算する関数である。

4　言語処理部

現在のところ、言語処理部の開発は「玉」を扱うものに限定して進めている。これは「玉」を扱う問題が「場合の数・確率」の分野で多く存在しているためである。「玉」の問題に対する言語処理部の実現を通して得られた知見は、「玉」以外の問題にも適用できるものと考えている。開発・評価データとしてチャート式参考書 [1] の「場合の数・確率」に関する問題のうち、「玉」が対象の問題 70 問を抽出し、そのうち 46 問を開発用、24 問を評価用に用いた。

4.1　問題文の分類と書き換え

問題文はあらかじめ人手で object 文、event 文、answer 文のいずれに属するかを付与している。正規表現を用いたパターンマッチングを多く用いているため、あらかじめ問題文を書き換えておくことで、以降の処理が簡単になる。以下は書き換えルールの一部である。

- 漢数字 => 算用数字
- すべて or 全部 => 全て
- 1 つ => 1 個、2 つ => 2 個、…
- {n} 個の {color} 玉 or {color} 玉が {n} 個 => {color 玉 }{n} 個
  例: 1 個の赤玉 => 赤玉 1 個

- {color} 玉に {from} から {to} までの番号 => {from} から {to} まで数字が書かれた {color} 玉
  例: 赤玉には 1 から 4 までの番号がつけられている => 1 から 4 までの数字が書かれた赤玉

4.2 中間表現への変換

object 文 ここで必要な情報は、オブジェクトの「名前」「属性名と属性値の組」およびオブジェクトの「個数」である。

> 1 から 6 までの数字をつけた赤玉 6 個を入れた袋がある。

この例で必要な情報は次の通り。名前: '玉', 属性 (色): '赤', 属性 (数字): 1,…,6, 個数: '6'。これら情報を正規表現で取り出し、中間表現に変換する。

event 文 基本的にはオブジェクトに対して何らかの操作を行う表現なので、動詞に注目する。さらに、例えば「取り出す」ときは、「同時に」取るのかそうでないのかをキーワード「同時に」「いちどに」の存在によって識別する。また、「同時」なら 1 度に取る個数を、そうでないなら何回取り出すのかの情報を数量に対するパターンマッチで取り出す。

answer 文について 求める値が、場合の数, 確率, 期待値 , 最大値, 最小値のいずれに対応するかを識別する。問題文に「場合の数」または「通り」とあれば、場合の数、それ以外は、「確率「」, 「期待値」 などの語そのものが出現しているかどうかで分類する。次の例は最も簡単な例である。

> object: 赤玉 3 個、青玉 4 個が入った袋がある。
> event: 袋から 3 個の玉を同時に取り出したとき
> answer: 球の取り出し方は何通りあるか。

この場合、answer 文はキーワード「通り」によって場合の数を求める関数へ変換される。次に取り出し方の条件が存在する場合について考える。object 文と event 文は先ほどと同じとする。

> answer: 全て青玉であるような球の取り出し方はいくつか。

このとき、「全てが青玉」という取り出し方の条件を場合の数を求める関数に引数として渡す必要がある。「全てが {color} 玉」というフレーズが answer 文に存在するとき、試行で取り出す玉の個数を {n} とすれば、条件を満たす場合の数を求めるための中間表現は以下の様になる。

表 1: シミュレータ部の評価 (開発用データ)

文の種類	文問題総数	変換可能	率 [%]
object	53	37	69.8
event	52	35	67.3
answer	75	59	78.7
all	75	48	64.0

表 2: 言語処理部の評価 (開発用データ)

文の種類	文問題総数	変換可能	率 [%]
object	53	31	58.4
event	52	30	57.7
answer	75	30	40.0
all	75	18	24.0

```
findCombination(
    lambda{|x|x.attr('color','{color}', {n} ) } )
```

ここで、lambda 式の引数 x.attr は、オブジェクトのもつ属性のリストを返す。

5 システムの評価

5.1 シミュレータ部

まず、現在の中間表現およびシミュレーションの枠組みで表現可能な問題の割合を調べた。自動生成ではなく人手で書いた場合、問題文の意味することを現在の中間表現で表すことができた割合を表 1 に示す。object, event, answer はそれぞれ文単位で中間表現として書き表せたもの、all は、必要な object 文、event 文、answer 文が全て中間表現で表せ、問題全体を通して正しい解答が得られたものの割合である。シミュレータはセンター試験過去問を開発データとして作成されたので、このチャート式の問題に対する評価はオープンテストである。object 文や event 文よりも all の方が変換可能数が多いのは、object 文 1 つに対し、answer 文を複数含む問題が存在するためである。同じ理由で、all の変換可能率も object, event 文の変換可能率に比べ極端には下がっていない。

次に言語処理部が正しく中間表現に変換できた文および問題の割合を表 2 と表 3 に示す。言語処理部はシミュレータ部とは異なり、全て変換できたものの割合は低い。現在のプログラムでは、answer 文の変換可能率が特に低いことが主な原因となり、必要な文を全て正しく中間表現に変換できる割合が低下していると考えられる。

表 3: 言語処理部の評価 (評価用データ)

文の種類	文問題総数	変換可能	率 [%]
object	27	15	55.6
event	30	15	50.0
answer	44	16	36.4
all	44	7	15.9

6 課題

現状では解けない問題の原因を中間表現の枠組みによる限界と、言語処理部の問題に分けて整理する。

6.1 シミュレータ部の問題

現在のシミュレータでは、仮想的な袋が 1 つだけ存在し、そこからいくつかのオブジェクトを取り出す、というモデルによって表現できるような試行のみが想定されている。そのため、以下のような問題は現在の枠組みでは表現できない：

1. 「袋」に相当するものが 2 つ以上存在する問題
2. 「玉を n 個取り出す」のように変数が用いられた問題

1 は、仮想的な袋を 2 つ (またはそれ以上) 保持することで比較的容易に解決可能であるが、取り出す操作がどの袋を対象とするかを認識することが新たな課題となる。2 は、実際に試行をシミュレートする現在の枠組みでは本来扱えない問題だが、変数の値を変えて何度もシミュレーションを行い、異なる変数値に対する結果から一般解を推測する方法が考えられる。

6.2 言語処理部

5 節で述べたように、文単位での変換成功率はおよそ 60%である。本システムでは問題を 3 つの部分に分けるため、最低 3 文以上を変換できなければ正解が得られない。そのため解答が得られる割合は 15.9%と低いのが現状である。現在、中間表現での表現は可能だが、言語処理部が変換できない問題文の例を挙げる。

・得点や期待値などを答えさせる問題

> 袋の中に赤球 3 個，白球 2 個が入っている。この袋から 2 個の球を同時に取り出す。赤球 1 個につき 1 点，白球 1 個につき 2 点が与えられる。このとき，もらえる合計点の期待値を求めよ。

試行結果によって定まる得点や、その期待値に関する問題では、各試行結果と得点の対応関係を問題から読み取る必要がある。現在の言語処理部では、これが実現できていない。

・試行結果ががある条件を満たすとき、試行全体を終了する問題 (試行回数があらかじめ数値で与えられていない問題)

> 赤玉 10 個と白玉 5 個の入っている袋から 1 個の玉を取り出し，色を見て袋に戻す。この試行をどちらかの色の玉が 3 回取り出されるまで繰り返す。

中間表現では終了条件を関数 (lambda 式) で表し、シミュレータに引数として渡すことが出来るが、この条件は複雑であることが多いため、自動変換は実現できていない。

・answer 文の条件が頻出パターンでない

> ・袋の中に赤球と白球が 4 個ずつ入っている確率を求めよ。
> ・2 で割り切れるが、3 では割り切れない確率を求めよ。
> ・3 回以下の試行で袋の中が白の玉が 4 個となる確率を求めよ。

多くの answer 文は確率を求めるべき様々な事象の記述を含むため、object 文や event 文と比較すると、定型的な文であることが少ない。そのため、パターンを網羅することが容易ではない。

7 おわりに

言語処理部とシミュレータ部を組み合わせることで、「場合の数・確率」の文章題に自動で解答するシステムを開発中であるが、現状では正答率は低い。シミュレータ部では、多くの問題タイプに対応できるようにして、より汎用的なプログラムにする必要がある。言語処理部では、問題文を変換するパターンを増やしていき、変換可能率をあげるとともに、オブジェクトが玉でないものにも対応する必要がある。

謝辞

本研究で使用したチャート式問題集データを提供いただいた数研出版に感謝いたします。

参考文献

[1] Studyaid D.B. チャート式データベース数学 I+A 統合版 (DVD-ROM). 数研出版, 2014.

[2] Gelb J.P. Experiments with a natural language problem-solving system. In IJCAI-1971, 1971.