数学・確率文章題の条件記述の自動解釈

Interpretation of Conditional Expressions in Probability Problems

名古屋大学　大学院工学研究科　教授
佐藤　理史

京都大学大学院工学研究科電気工学第二専攻博士課程研究指導認定退学。博士（工学）。
北陸先端科学技術大学院大学、京都大学を経て、2005 年より現職。

1 はじめに

大規模言語モデル（Large Language Model, LLM）の登場により、言語処理が新たな展開を見せている。これに伴い、これまでプログラムで記述していたコンピュータやロボットに対する命令や指示を、自然言語で伝えることが増えると予想される。

大規模言語モデルを中核とする言語処理系の問題点のひとつは、系がブラックボックスであるという点である。そのため、何が正しく解釈できて、何が正しく解釈できないか、「動かしてみないとわからない」。そして、その再現性が保証されない。チャットボットのような応用では問題とはならないが、命令や指示を与える対象が物理的実体をもち、それが実世界を操作できる場合には、大きなリスクを伴う可能性がある。そのような考えに立つと、アルゴリズムに基づく伝統的な言語処理技術の研究を放棄するのは得策ではない。

本稿では、数学の確率・期待値の文章題を解釈するアルゴリズムの研究[1]を紹介する。確率・期待値の問題文は、演算記述、条件記述、条件分岐、名付けなどを含んだ文章であり、意味解釈の例題としての十分な複雑度を有している。意味解釈では、正しく解釈できたかどうかをどのように判断するかが難しいが、確率・期待値の文章題の意味解釈では、その解釈結果によって正解を導けたかどうかで客観的に評価できるという利点がある。本研究の中心は、確率・期待値の文章題に含まれる条件記述を正しく解釈し、それを実行可能なプログラムコードに変換する方法にある。

2 確率文章題と条件記述

確率や期待値を問う文章題は、次のような特徴を持つ。

①サイコロ・玉・カードなどの題材を用いて仮想世界が記述され、その世界の中で問題が記述される。

②確率を計算すべき事象を規定するための条件の記述が、かなり複雑になりうる。

③答を導く際に必要となる値（典型的には、期待値問題における「得点」）の計算法が、問題文中で定義される。

これらの特徴のうち、本研究では、特に 2 番目の特徴に焦点を当て、事象を規定する条件の記述（以下、条件記述と呼ぶ）を正しく解釈する問題を扱う。以下の問題文では、下線部が条件記述である。

（例 1）2 つのサイコロを同時に振る。出た目の和が偶数である確率を求めよ。

我々が作成した解答器（ソルバー）は、この問題文を解析・解釈し、次のような Ruby プログラムコードに変換する。

(例 2) SampleSpace.dice(2).確率{|r| r.目.和.偶数?}

このコードの前半「SampleSpace.dice(2)」は、サイコロ 2 個を振ることによって構成される、36 個の要素からなる集合（標本空間 , SampleSpace）を作成する。残りの「.確率{|r| r.目.和.偶数?}」は、集合のそれぞれの要素（r）に対して、その目の和が偶数であるかどうかを調べ、偶数であった要素の数を集合の要素数で割って、確率を返す。すなわち、このコードの実行により、問題に対する答が得られる。プログラミング言語 Ruby はオブジェクト指向言語であり、そのコードでは

メソッドチェインが多用される。メソッドチェインとは、あるメソッドの適用結果（出力）に、次のメソッドを適用することを繰り返す形式である。条件記述「出た目の和が偶数である」の解釈結果である「r. 目 . 和 . 偶数 ?」も、メソッドチェインの形式で、r に「. 目」を適用し、その結果に「. 和」を適用し、その結果に「. 偶数 ?」を適用することを意味する。

3 CCG-Ruby による条件記述の解釈

条件記述の自動解釈には、Combinatory Categorial Grammar（CCG）[2][3] を流用した枠組み CCG-Ruby を用いる。標準的な CCG との違いは、以下の通りである。

①基底カテゴリとして、S や NP などの構文カテゴリではなく、Ruby コードのデータ型を用いる

②意味記述として、論理式ではなく、Ruby コードを採用する（解析が成功した場合、その解釈結果として Ruby コードを出力する）

③解析には、語彙項目と 4 タイプ 8 種類のコード合成規則を用いる。

（例 1）に含まれる条件記述「出た目の和が偶数である」を解釈するために必要な語彙項目は、図 1 に示す 5 つである。語彙項目は、「語彙 ⊢ 型 : コード」の形式で記述される。

出た ⊢ *result* : r
目の ⊢ Int\Dice : .目
和が ⊢ Int\LofInt : .和
偶数で ⊢ Bool\Int : .偶数?
ある ⊢ (Bool\X)\(Bool\X) : ϵ

図 1　語彙項目の具体例

サイコロ問題の場合、基底カテゴリ（基本となる型）は、Dice（サイコロ）, Int（整数）、Bool（真偽値）の 3 種類である。この他に、型 T の要素を複数持つ型として、LofT（リスト）、SofT（列）、AofT（and リスト）、OofT（or リスト）がある。

「Y \ X」や「Y ／ X」はメソッドの型で、そのメソッドがデータ型 X に対して適用でき、データ型 Y の出力を返すこと（つまり、入出力仕様）を意味する。たとえば、メソッド「. 偶数 ?」の型「Bool\Int」は、このメソッドが整数（Int）に対して適用できるメソッドで、真偽値（Bool）を返すことを意味する。スラッシュやバックスラッシュは、言語表現の結合の方向を示す。たとえば、バックスラッシュは X が左から結合する（Y より前に現れる）ことを示す。このため、「. 偶数 ?」の型「Bool\Int」は、「偶数かどうかを調べる対象（整数）」が、「偶数で」という表現よりも前に現れることを意味する。

ほとんどの語彙に対してはあらかじめ型を定義するが、標本空間の要素の型を意味する「result」だけは例外で、これは動的に決定する。（例 1）の試行は「2 つのサイコロを同時に振る」なので、result は、LofDice（2 つのサイコロを要素とするリスト）となる。

コード合成規則は、図 2 に示す 4 タイプ 8 種類がある。これらは、文脈自由文法の規則に対応し、水平線の上が規則の右辺、下が規則の左辺に対応する。たとえば、規則（1b）は、型「X」を持つ要素と「Y \ X」を持つ要素の並びが存在したら、それらを「Y」にまとめること、そして、「X」のコード v に「Y \ X」のコード .m を結合して、v.m というコードを合成することを意味している。

$$(1b)\quad \frac{X\colon v \quad Y\backslash X\colon .m}{Y\colon v.m}\,{<}$$

$$(1f)\quad \frac{Y/X\colon .m \quad X\colon v}{Y\colon v.m}\,{>}$$

$$(2b)\quad \frac{X|Z\colon .m \quad Y\backslash X\colon .n}{Y|Z\colon .m.n}\,{<_c}$$

$$(2f)\quad \frac{Y/X\colon .n \quad X|Z\colon .m}{Y|Z\colon .m.n}\,{>_c}$$

$$(3b)\quad \frac{X\colon x \quad Y\backslash_a X\colon .m(\cdots)}{Y\colon .m(\cdots, x)}\,{<_a}$$

$$(3f)\quad \frac{Y/_a X\colon .m(\cdots) \quad X\colon x}{Y\colon .m(\cdots, x)}\,{>_a}$$

$$(4b)\quad \frac{X\colon v \quad \mathrm{Qof}(Y\backslash X)\colon f(.m_1, .m_2, \ldots)}{\mathrm{Qof}\,Y\colon f(v.m_1, v.m_2, \ldots)}\,{<_q}$$

$$(4f)\quad \frac{\mathrm{Qof}(Y/X)\colon f(.m_1, .m_2, \ldots) \quad X\colon v}{\mathrm{Qof}\,Y\colon f(v.m_1, v.m_2, \ldots)}\,{>_q}$$

‘|’は、‘/’か‘\’のいずれかを表す。

図 2　合成規則一覧

図 3 に、（例 1）に含まれる条件記述「出た目の和が偶数である」の解釈過程を示す。CCG では解析木（導出過程）を通常の木構造と逆方向に、葉を上、根を下に記述する。ここでの注目点は、「出た（LofDice:r）」と「目の（Int\Dice: . 目）」の結合である。前者は標本空間の

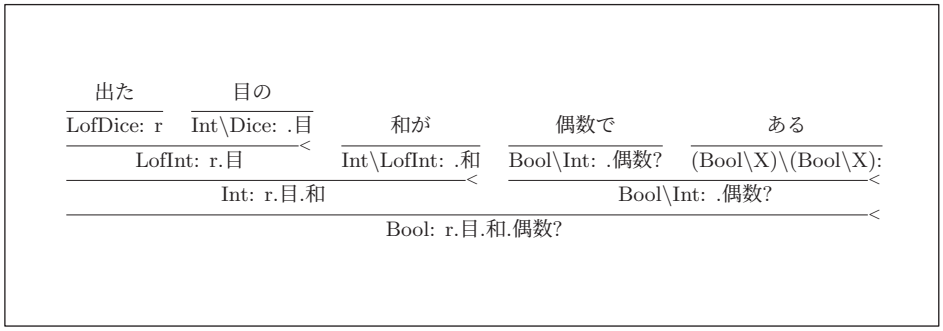

図３　(例 1) の条件記述の解釈結果

要素（LofDice）、後者はサイコロ（Dice）からその目の値（Int）を取り出すメソッドである。このメソッドはサイコロのリスト（LofDice）にはそのままでは適用できないが、サイコロのリストの各要素に「. 目」を適用し、その結果をリストとして返すように拡張解釈すれば、適用できる。システムには、このような拡張機能が組み込まれており、サイコロのリスト（LofDice）から整数のリスト（LofInt）が得られる。これは、「サイコロを振る。出た目」はサイコロ１個の目（単数）を表すのに対し、「２個のサイコロを振る。出た目」はサイコロ２個の目（複数）を表すことに対処するためである。日本語の名詞の多くは単複同形であり、「目」だけでは単数か複数かを判断することはできず、文脈（この場合は、何個のサイコロを振ったか）を考慮する必要がある。

このように、CCG-Ruby では、解析時に合成するコードの入出力の型チェックを行う。そのため、型が整合しない解釈は出力しない。実際、次のような問題文は、解釈不能となる。

（例 3）サイコロを振る。出た目の和が偶数である確率を求めよ。（「出た目」は単数であるため、和をとることはできない）

（例 4）サイコロを２個振る。出た目が偶数である確率を求めよ。（「出た目」は複数であるため、偶数であるかどうかを判定できない）

このように、解釈できない記述を「解釈不能」と判定できることは、アルゴリズムに基づくアプローチの利点の一つである。

4　複雑な条件記述とその解釈

条件記述は、等位接続表現（「かつ」、「または」）、否定表現、限量子に相当する表現（「すべて」、「いずれか」）などで複雑化する。具体的には、次のような問題である。

（例 5）サイコロを振る。出た目が２か３で割り切れる確率を求めよ。（「か」が「または」を意味する）

（例 6）サイコロを５回振る。１が２回、２が２回出る確率を求めよ。（読点が「かつ」を意味する）

（例 7）サイコロを２個振る。偶数の目がひとつも出ない確率を求めよ。（「ひとつも出ない」）

（例 8）サイコロを２個振る。ひとつしか偶数の目が出ない確率を求めよ。（「ひとつしか出ない」＝１つだけ出る）

（例 9）サイコロを３個振る。すべて偶数の目が出る確率を求めよ。（「すべて」）

（例 10）サイコロを３個振る。出た目がすべて異なる確率を求めよ。（「すべて異なる」）

（例 11）３個のサイコロを同時に振る。３個のうち、いずれか２個のサイコロの目の和が５になる確率を求めよ。（「いずれか２個」）

これらの問題文の条件記述は、適切な語彙項目を定義すること、および、コード縮退と呼ぶコードを簡単化する規則を導入することにより、CCG-Ruby で正しく解釈することができる。これらの問題文のうち、（例 6,8,11）の解釈結果を図４に示す。

作成したソルバーは、次のような名付けを含む期待値文章題も解くことができる。

（例 12）サイコロを２個振る。出た目の和をＳとするとき、Ｓの期待値を求めよ。

この問題文は、以下のような Ruby コードに変換される。

（例 13）SampleSpace.dice(2)

```
. 名付け('S', 'Int/LofDice'){|r| r. 目 . 和 }
. 期待値 {|r| r.ref('S')}
```

このコードでは、「Ｓ」の計算法をコード化して定義し、期待値の計算でそれを利用する。

５冊の高校生用の問題集から収集したサイコロを題材

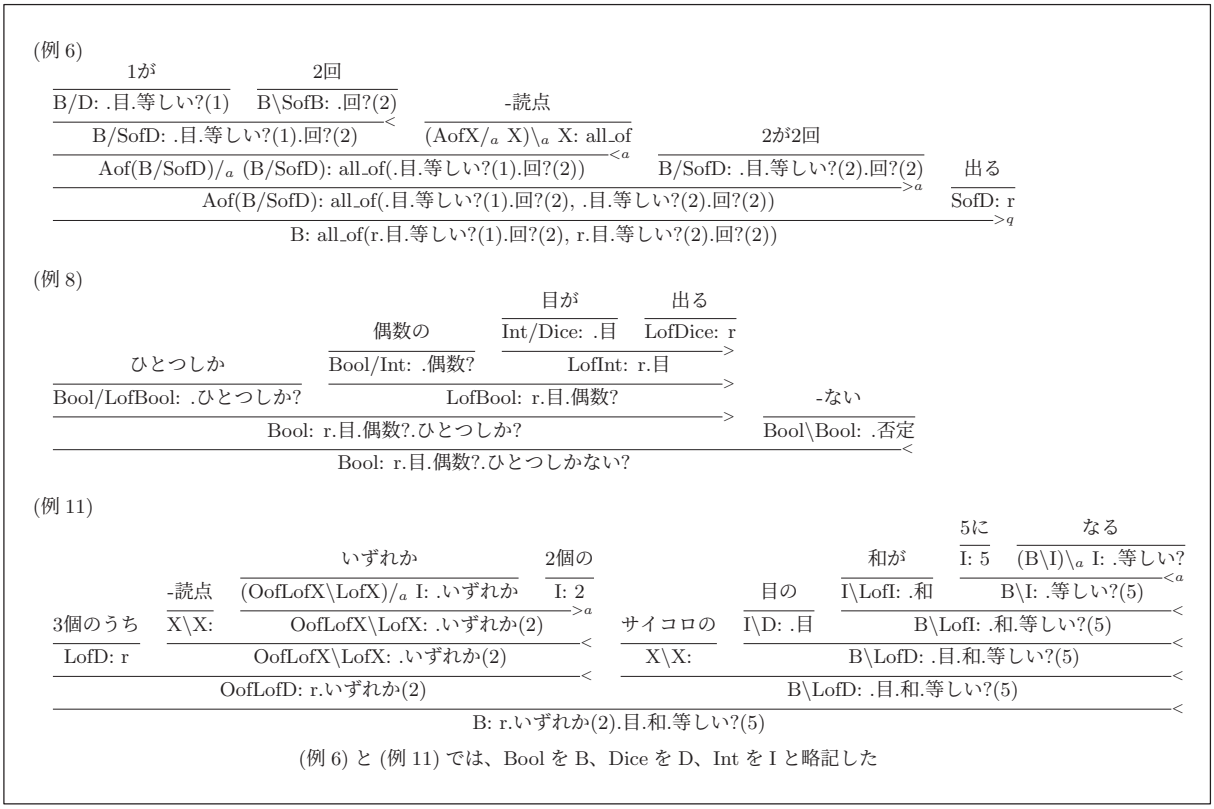

図４　(例 6,8,11) の解釈結果

とした問題計 41 問のうち、現在のシステムで解けない問題は、以下の２問である。

（例 14）２個のさいころを同時になげる。１つのさいころの目だけが３の倍数である確率を求めよ。

（例 15）さいころを４回投げる。１つの目が他のどの目よりも多く出る確率を求めよ。

前者では「１つの～だけ」をどのように解釈するかが問題となる。この問題は、「３の倍数の目がひとつしか出ない確率を求めよ」と同じ意味で、こちらの記述であれば現在のシステムで解けるのであるが、「ひとつ」と「だけ」が離れて出現することが自動解釈を難しくしている。

後者の問題では、「１つの目が他のどの目よりも多く出る」という表現全体が、ある特定の状況を簡潔な形で言い表している点に難しさがある。我々人間は、そのような状況を頭に思い浮かべ、さいころ４個の出た目がその状況に当てはまるかどうかを判断できるが、それをアルゴリズムとして実現する方法は自明ではない。

5 おわりに

本稿では、数学の確率・期待値の文章題に含まれる条件記述を解釈し、プログラムコードに変換する方法を紹介した。我々の興味は、日本語の条件記述を構成する要素にどのようなものがあり、それがどのように解釈されうるか、そして、どこに解釈の難しさがあるのかを知りたいという点にある。プログラム言語で条件を記述することを思い浮かべれば想像できるように、条件記述を構成する中核的要素は、論理演算（AND, OR, NOT）と、真偽値を返す述語である。これに、定数、演算、付加条件などが組み合わされる。CCG-Ruby の語彙項目は、日本語の表現とこれらとの対応関係を示し、コード合成規則は、それらの並びが実行可能な条件として組み立てられるための条件を規定している。

参考文献

[1] 佐藤理史 . CCG-Ruby による確率文章題の条件記述の解釈 . 情報処理学会研究報告 , Vol.2024-NL-259 No.6, 2024.

[2] Steedman, M. The Syntactic Process, MIT Press, 2001.

[3] 戸次大介 . 日本語文法の形式理論 . くろしお出版 , 2010.