卒業論文

Prolog における否定の述語「not」

氏名　徳山駿平

名列番号　226

学籍番号　1151020087

指導教員名　足立英彦

提出年月日　2015 年 1 月 13 日

論文要旨

論理学に基づくプログラミング言語である Prolog は、人間が物事を判断する際に行う推論の過程を明らかにする可能性を秘めている。しかし、その表現能力は十分ではなく、「否定的な情報を明示できない」という問題を抱えている。この欠点を克服するための代表的な手段の一つとして挙げられるのが、証明の失敗を否定とみなす「失敗としての否定」の形式によって否定を表す述語「not」を導入することである。

本論文では、この否定の述語 not の仕組み及び欠点を明らかにすることにより、not が否定を有効に表現しうる一方でその効果は限定されたものになるということを示す。

第 1 章においては、本論文で扱う Prolog とはそもそも一体何なのかということを、プログラミング言語における位置づけという観点から紹介する。まず、プログラミング言語と密接な関わりを持つ計算システムについてふれ、プログラミング言語の役割について説明した上で水準及び計算モデルの観点からプログラミング言語の分類を行い、Prolog が属する論理プログラミング言語の特徴を述べる。

続く第 2 章では、否定の述語 not を理解するために最低限必要な Prolog の基礎知識について説明する。具体的には、Prolog における情報の形式である項とホーン節、三段論法を基礎に置く Prolog の計算の中核となる機能ユニフィケーション、ある計算が失敗したときにその計算を後戻りさせる機能バックトラックを紹介する。また、それらを基にした計算が実際にはどのように行われるのかということを確認するために、家系図を表すプログラムを用いた推論を実例として行う。

そして、これら第 2 章の内容を踏まえて、第 3 章で否定の述語 not を扱う。最初に、not が基づく失敗としての否定について説明した後、not の導入方法（定義）及びそれを用いた計算の仕組みと、not が抱えている欠点を、家系図プログラムの例を交えながら示す。

目次

はじめに

人間はどのように物事を判断しているのか。すなわち、どういう基準で情報を取捨選択し、どのような法則に従って推論を行い、その結果どういった結論が導き出されるのか。これらを形式的に表現するための学問が論理学であり、その論理学に基づいて人工的に生み出された言語が Prolog[1]である。

この Prolog においては情報の形式のみが問題となり、その内容は、たとえ常識とかけ離れていても推論に影響を与えることはない。したがって、Prolog は人間の、価値観に囚われない純粋な推論としての思考過程の可視化に資するものであるといえる。そしてその客観的な推論は、しばしば先入観に判断を左右されがちな人間が見習うべき姿勢でもある。

一方で、Prolog の表現能力は決して十分なものではない。物事を判断するための材料としての情報の中には、「○○ではない」という、あるものを否定する情報が含まれることがあるが、Prolog はそういった否定的な情報を明示できない。情報の幅を広げ、様々な推論を表現するためにも、Prolog において否定を表す何らかの手段が必要になる。

本論文は、論理プログラミング言語 Prolog において否定を表現するための述語「not」の仕組み及び欠点を明らかにすることを目的とする。そのためにまず、プログラミング言語について簡潔に説明した上でその分類を行い、Prolog が属する論理プログラミング言語の位置づけを示す（第 1 章）。次に、not を導入するために最低限必要な Prolog の基礎知識を説明し、家系図プログラムを実例として推論を行う（第 2 章）。最後に not の導入方法（定義）とそれを用いた計算の仕組みを紹介した後、not が抱えている欠点を、家系図プログラムを用いて示す（第 3 章）。

第 1 章　プログラミング言語

programming logic に由来する名前から分かるように、Prolog は、論理プログラミング言語と呼ばれるプログラミング言語の一つである。言い換えると、プログラミング言語の集合に属する論理プログラミング言語の一種が、Prolog である。

したがって、Prolog を扱う前提として、プログラミング言語及び論理プログラミングとはどのようなものかを把握しておくことが必要ということになる。

第 1 節　計算システムとプログラミング言語[2]

計算システムとは、定義された計算を行うシステムであり、計算を定義するもの・定義に従って作業を行うもの（処理装置）・計算に必要な材料とその置き場所・計算に伴うメッセージの授受を行うための出入口から構成されている。なお、ここでの計算とは、計算

---

[1] Robert Kowalski、Alain Colmerauer らによって 1970 年代初期に開発されたプログラミング言語である。参照,［Sterling & Shapiro］p. 131。

[2] 所真理雄（1988） pp. 1-15 を参考にした。

システムが行う作業全般を指し、例えば算術演算や表・グラフの作成などさまざまなものがある。

この計算システムに対して「計算せよ」というメッセージ（計算の種類、計算の依頼者の名前、その他補助的な情報[3]）が送られると、置き場所にあるデータ（一定の決まりに基づいて表された事実・考え）を材料として処理装置が計算を実行し、その結果がメッセージ（返答者の名前、結果の内容）として返送される。

このようにして進行する計算の内容を定義するものはプログラムと呼ばれ、その記述に用いられる言語がプログラミング言語である。（[所] pp. 1-15）

第 2 節　プログラミング言語の分類[4]

このように、プログラミング言語には計算システムに対して指示を行うという役割がある。しかし一方で、プログラミング言語は問題を解くための方法を明確に表現している、ひいては人間の思考過程やものの表現法に密接に関連しているともいえる。

したがって、プログラミング言語が問題解決の際の思考媒体と考えられることから、問題の種類に応じて多種多様なプログラミング言語が存在することになる。

(1)　水準による分類

プログラミング言語を、機械寄りの低水準なものと、人間寄りの高水準なものに分けることができる。

低水準言語は計算機[5]（計算システムの基本的機能を備えた機械）の理解能力に応じた命令を用いる言語であり、計算機にとっての言語である機械語や、機械語を人間にとって理解しやすくした言語であるアセンブリ語がその例である。これに対し、計算機の特性を考えずに、処理を行おうとする問題に合わせて設計されているプログラミング言語を高水準言語という。高水準言語は（人間には理解が困難な）機械語と直接には関係していないため、人間にとって簡潔・明解な形式による表現が可能である[6]。

(2)　計算モデルによる分類

プログラムやデータの処理過程を抽象化したものを一般に計算モデルという。計算モデルの考え方を用いることにより、処理装置がプログラムやデータをどのように扱うかということを知ることができる。

---

[3] 情報とは、何らかの定義によるデータの解釈である。参照，[所] p. 9。
[4] 武市正人（1994） pp. 1-9 を参考にした。
[5] 参照，[所] p. 23。
[6] なお、高水準言語は、コンパイラと呼ばれるプログラムによって機械語に翻訳される。参照，[武市] p. 3。

例えば、計算機中のデータが記憶されている場所を参照する命令によって記憶内容を操作して計算過程を進行させる、すなわちそのような場所の状態変化が計算の効果であるとする計算モデルに基づく言語は、命令型言語あるいは手続き型言語と呼ばれる。一方で、数学における関数を用いることによって問題解決の方法を表現しようとするものを、関数型言語あるいは作用型言語という。数ある表現のうち人間にとって最も簡潔なものを見つけ出す処理が、この言語にとっての計算である。状態をどのように変化させるのかを指令するための命令を主体とする命令型言語と異なり、関数型言語には、記憶する場所の状態という概念はない。

これらに対し、ある論理体系の中で命題を証明する際の推論規則の適用を計算過程とみなすような言語は、論理型言語と呼ばれる。すなわち、この言語においては「論理式の証明を、計算であると考える[7]」。([武市] pp. 1-9)

以上の分類に従うと、論理プログラミング言語は高水準言語かつ論理型言語である。

そして、「論理プログラミング言語の理想を部分的に実現したもの[8]」が Prolog である。

第 2 章　Prolog

プログラミング言語である Prolog には、プログラムの記述や計算の手順において様々な決まりごとが存在する。そこで本章では、Prolog における計算を理解するための基本知識を説明する。なお、Prolog にはさまざまな処理系[9]が存在するが、ここでは特に DEC-10 Prolog[10]に基づいて説明を行う。

第 1 節　Prolog の文法

Prolog におけるプログラムは、項と呼ばれる要素から構成されるホーン節の集合である。

(1)　項[11]

項には、アトム・変数・数字・複合項の 4 種類がある。

まず、アトムは小文字から始まる文字列である。引用符で囲まれた文字列もアトムとみなされ、例えば book や aB、'3.14'、'Tom'のようなものがアトムである。これに対し、大文字から始まる文字列を変数という。例えば X や Tom などである。また、数字は 23 や-3.14 のように表す。アトムと数字が定まった値を表しているのに対し、変数はその値が確定し

---

[7] 参照，[井田] p. 7。
[8] 参照，[中村] p. 2。
[9] プログラムの解釈や計算を行うシステムの総称。参照，[川合] pp. 30-32。
[10] 1977 年にエジンバラ大学で開発された Prolog 処理系。参照，[新田，佐藤] p. 1。
[11] 新田克己・佐藤泰介（1986） pp. 4-9 を参考にした。

ていないものを表している。

そして、アトムの後にいくつかの項をカッコで囲ったものを複合項という。このときのアトムをファンクター、カッコに囲まれた項を引数と呼び、f(X)や foo(g(10),20)といった形をしている。

ここで重要なことは、複合項が、対象としているものの間に成り立つ関係を表す述語を用いて物事を表現したもの、すなわち述語表現であるということである。例えば、

Tom loves Susie.

という文を、Tom と Susie のあいだに love という関係が成り立っているとみなして、

love (‘Tom’ , ’Susie’).

と表現するのが述語表現である。

なお、こういった述語表現で物事を表すときに注意しなければならないことがある。それは、引数の順序・述語名・変数の使用・真偽値の 4 つである。

まず、引数の順序は、一度決めると勝手に変えることができない。言い換えると、引数の順序を変えたものは、全く違うホーン節として扱われることになる。例えば、love(A,B)が「A は B を愛する」を表現しているとみなす場合、love(‘Tom’ , ’Susie’)と、その引数を入れ替えた love(‘Susie’ , ’Tom’)は逆の意味になる。

また、述語名はアトム、すなわち単なる識別記号であることから、述語表現では特別な注意が必要となる。例えば次の 2 つの文

parent (‘Tom’ , ‘Bill’). 「Tom は Bill の親である」

child (‘Bill’ , ‘Tom’). 「Bill は Tom の子である」

は常識的には同じ内容であるが、述語名が異なるため、述語表現においては別のものとして扱われる。このように、ホーン節を意味づけているのはその形式であり、その内容が常識と一致しているかどうかは問題にならない。

次に、変数は個々のデータの一般形を表している。したがって、father(john ,X)は「john は（誰かは分からない）X の父である」を意味している、すなわち X は、john を父に持つすべての者を表しているということになる[12]。また、同じ変数が引数として 2 つ以上現れる場合、それらは同じものを表しているため、love(X,X)は「愛する者と愛される者が同一人物である」ことを意味しているとみなすことができる。

最後に、述語表現は、述語の意味を考慮したときにその引数間の関係が正しい場合には真、正しくない場合には偽という真理値を持つ。例えば f(X,Y,Z)が「X と Y を足したものは Z である」という意味であるとすると、f(2,3,5)は真であるが、f(2,3,6)は偽である。ところが「X と Y を掛けたものは Z である」という意味であるとすると、f(2,3,5)は偽であ

[12] ただし、john を父に持つという情報がプログラム上で明記されている、あるいは暗示されている（推論により導くことができる）者に限る。

るが、f(2,3,6)は真である。（[新田，佐藤] pp. 4-9）

(2)　ホーン節[13]

Prolog プログラムの構成単位は、節と呼ばれるものである。

A：－B1 , B2 , … , Bn.

節は頭部（head）と本体（body）から成り、節を構成している項をゴールという。この節では、A が頭部、B1 , B2 , … , Bn が本体である。この節が意味しているものは「B1、B2、…、Bn がすべて真であるならば、A は真である」、あるいは「A が真であることを示すには、B1、B2、…、Bn がすべて真であることを示せ」ということになる。なお、前者を宣言的な読み方、後者を手続的な読み方という。すなわち、この節は宣言的には、

B1∧B2∧…∧Bn　→　A

という論理式を表していることになる。そして、節の中でも特に Prolog において記述することが可能なものは規則・事実・質問と呼ばれ、これら 3 種類の節を総称してホーン節という。

以下では、これら 3 つの節について説明する。

まず、節の一般的な形式は規則と呼ばれる。

mother (X,Y) ：－ parent (X,Y) , woman (X).

mother(X,Y).が「X は Y の母である」、parent(X,Y).が「X は Y の親である」、woman(X).が「X は女性である」を意味しているものとすると、この規則は宣言的に「X が Y の親で、かつ、X が女性ならば、X は Y の母親である」と読むことができる。なお、規則の中に同じ変数が複数現れる場合、それらは同じものを表しているということに注意しなければならない。

また、規則の本体の部分が、頭部が真であるための条件を表していると考えると、本体のない節は無条件に真になるものとみなすことができる。このような節を事実という。

mother (‘Jane’ , ‘Tom’).

この事実は「Jane は Tom の母親である」を意味している。

こういった規則・事実から Prolog プログラムが構成されるが、Prolog が計算（証明）を開始するのは、質問と呼ばれる節が与えられたときである。質問は頭部のない節であり、「本体の各ゴールが成立するか否かを調べよ」という意味を表すものして扱う。

?－ mother (‘Susie’ , ‘Tom’).

この質問は「Susie は Tom の母親であるか否かを調べよ」を意味している。

ところで、規則が意味しているような「～が成り立つならば、～が成り立つ」という表現には、以下の 4 つの基本形がある。

---

[13] 新田克己・佐藤泰介（1986） pp. 9-12 を参考にした。

(i)　B かつ C が成立するならば、A が成立する（B∧C→A）

(ii)　B または C が成立するならば、A が成立する（B∨C→A）

(iii)　C が成立するならば、A かつ B 成立する（C→A∧B）

(iv)　C が成立するならば、A または B が成立する（C→A∨B）

(iv)をホーン節で表すことはできないが、(i)～(iii)を表現することは可能である。

(i)　A：－B , C.

(ii)　A：－B.　　A：－C.

(iii)　A：－C.　　B：－C.

規則は、単独では(i)の意味でしか用いることができないということに注意が必要である。（[新田，佐藤] pp. 9-12）

第 2 節　Prolog の計算

Prolog は、論理学を基礎に置くプログラミング言語である。ゆえにその計算は、論理学の推論規則である三段論法に則ったものとなっている。

(1)　三段論法[14]

三段論法は、大前提と小前提から結論を導くものである。例えば、「人間は必ず死ぬ（大前提）」と「ソクラテスは人間である（小前提）」から、「ソクラテスは必ず死ぬ（結論）」を導くことができる。

この推論を節で表現する。初めに、大前提と小前提はそれぞれ次のように表される。なお、理解しやすくするために、右側にはそれぞれの節に対応する意味を示してある。

（大前提）　mortal (X)：－human (X).　　「X が人間ならば、X は必ず死ぬ」

（小前提）　human ('Socrates').　　「ソクラテスは人間である」

ただし、human(X)は「X は人間である」を、mortal(X)は「X は必ず死ぬ」を意味しているものとする。まず大前提の変数 X は、human であるすべてのものを表していることから、そこに human である'Socrates'を代わりに当てはめても構わないということになる（このときの当てはめを代入という）。また、規則の中に現れる変数は同じものを表しているため、mortal(X)の X にも'Socrates'が代入される。ゆえに上記の大前提から、

mortal ('Socrates')：－human ('Socrates').

という新たな規則が導かれる。これは「ソクラテスが人間ならば、ソクラテスは必ず死ぬ」を表しており、推論に用いることができる。ここで、新たな規則の条件部分「ソクラテスは人間である」はすでに小前提において事実（常に真である）として示されている。したがって、新たな規則と前提として与えられていた小前提から、結論である「ソクラテスは

[14] 新田克己・佐藤泰介（1986）　pp. 19-23 を参考にした。

必ず死ぬ」が導かれることになる。

以上のような証明方法は前向き推論と呼ばれ、節を組み合わせていくことにより、新たな情報が次々に追加されていくという特徴がある（上のソクラテスの例では、新たな規則 mortal(‘Socrates’) : －human(‘Socrates’)が得られた）。

しかしこの推論には、効率的な節の組み合わせについての手がかりがない場合には余分な推論をしてしまうという欠点がある。そこで Prolog の計算は、前向き推論とは逆の推論方法である後向き推論に対応している。前向き推論が、導き出された結論と証明したいものが一致したときに証明成功となるのに対し、後向き推論は証明したいものを結論として遡り、その結論が導き出されるための前提条件が満たされている場合に証明成功となる。

例えばソクラテスの例で後向き推論をする場合、「ソクラテスは必ず死ぬ」という結論を証明するためにはどのような条件が必要かをまず考える。

mortal (X) : －human (X).　「X が人間ならば、X は必ず死ぬ」

すると、この大前提の X に’Socrates’を代入して得られる次の規則は、手続的に「ソクラテスが必ず死ぬことを示すためには、ソクラテスが人間であることを示せばよい」と読むことができる。ただし、この規則はプログラムに追加されたわけではなく、この推論においてのみ有効であるに過ぎないということに注意が必要である。

mortal (‘Socrates’) : －human (‘Socrates’).

新たに得られた規則から、次に証明すべきは「ソクラテスは人間である」ということになる。しかしこれは先程と同様に、すでに小前提

human (‘Socrates’).　「ソクラテスは人間である」

において事実として示されていることから、真であることは明白である。また、次に証明すべきものは何もない。こうして証明すべきものがなくなったとき、証明は成功となる。

この後向き推論において重要なことは、推論の過程において問題の書き換えが行われているということである。すなわち、予め前提として与えられている規則を用いて証明したい節を変換していき、証明すべきものがなくなったとき、言い換えれば、それ以上変換できなくなった節と同じ形をしている事実が存在するとき、証明が成功したということになる。（[新田，佐藤] pp. 19-23）

(2)　ユニフィケーション[15]

節を用いた三段論法では、ゴール（証明したいもの）と同じ形の頭部を持つ規則あるいは同じ形の事実を選び、その中に現れている変数に項を代入することで、ゴールとその規則の頭部あるいはその事実を等しい形にする操作が行われる[16]。この操作はユニフィケー

[15] 新田克己・佐藤泰介（1986） pp. 23-25 を参考にした。

[16] ここでは、同じファンクターを持ち、かつ引数の数が同じであるものを「同じ形」、な

ションと呼ばれ、Prolog において最も重要な機能の一つとなっている。

ユニフィケーションでは 2 つの節の引数を同じ形にするため、変数に対して項の代入が行われる。変数は定まっていないものを表しているため、アトム・数字・変数・複合項のいずれを代入してもよい。例えば

student (tom)　と　student (X)　　…X に tom を代入
family (john ,X)　と　family (Y,4)　　…X に 4 を、Y に john を代入
f (g (X))　と　f (Y)　　…Y に g (X)を代入

などは、ユニフィケーションによって同じ形にすることが可能である。逆に言うと、アトム・数字・複合項間の代入は不可能である。この場合、定まっているものとしてそれらを比較し、同じ形をしていればユニフィケーションは成功となる。しかし次のような場合、例えば、異なるアトム・数字を持つ 2 つの節は同じ形にならないため、ユニフィケーションができない。

student (tom)　と　student (susie)
family (john ,3)　と　family (bill ,6)

また、ユニフィケーションにおける代入には、他にもいくつかのルールが存在する。

まず変数について、1 つの変数に対する代入は一回しかできず、同じ節の中に複数現れる変数にはそれぞれ同じ項が代入される。例えば f(X,X)と f(tom ,susie)でユニフィケーションを行う場合、前者の第一引数である X に tom が代入されると、第二引数の X にも tom が代入されることになる。すると、すでに tom が代入されている X には後者の第二引数である susie（アトム）を代入することができず、両者を比較した結果、同じ形ではないためユニフィケーションは失敗となる。また、これとは別の例として、f(X,Y,X)と f(A,A,B)はファンクターも引数の数も同じであるが、X に A（第一引数）・Y に A（第二引数）・X に B（第三引数）が代入されると考えると、計算の過程でこれらの変数すべてに同じ項が代入されなければ両者は同じ形にはならず、ユニフィケーションは成功しないことになる（X=A・X=B より A=B が、X=A・Y=A より X=Y が導かれることから、X=Y=A=B となる）。

もう 1 つ注意しなければならないことがある。それは、ユニフィケーションにおいて比較できるのは形式のみであるということである。例えば、f(2+3)と f(5)はユニフィケーションすることができない。この場合、あくまで 2+3 という形式のみを問題としているにすぎず、足し算を行い、その結果の 5 として解釈することはできないからである。同様の理由から、f(2+3)と f(X)をユニフィケーションすると、5 ではなく 2+3 が X に代入される。（[新田，佐藤] pp. 23-25）

おかつ引数の形もすべて一致しているものを「等しい形」と呼んでいる。

第 3 節　実例

ここからは Prolog のプログラミングの実例として、文字通り家系図を表したプログラムである家系図プログラムを用いた推論を行い、Prolog の計算がどのように進められるのかを明らかにする。

(1)　家系図プログラム[17]

最初に、hanako・shinji・taro を構成員とする 3 人家族を表す家系図プログラムの記述を行う。parent(X,Y)が「X は Y の親である」を表しているとすると、この家族の親子関係を次のように記述することができる。なお、ホーン節を左に、それが表している意味を右に、説明を簡潔にするための番号を中心に示してある。

parent (hanako , taro).　　…①　「hanako は taro の親である」

parent (shinji , taro).　　…②　「shinji は taro の親である」

また、man(X)と woman(X)がそれぞれ「X は男性である」、「X は女性である」を意味しているとすると、これらを用いて各構成員の性別を次のように記述することができる。

man (shinji).　　…③　「shinji は男性である」

woman (hanako).　　…④　「hanako は女性である」

man (taro).　　…⑤　「taro は男性である」

parent と man・woman を組み合わせることで、父母を表す規則を導入することができる。父親は「親でありかつ男性である」、母親は「親でありかつ女性である」ことから、それぞれ次の規則で表現される。

father (X,Y)：－parent (X,Y) , man (X).　　…⑥

mother (X,Y)：－parent (X,Y) , woman (X).　　…⑦

以上の事実・規則から成る家系図プログラムを、プログラム 1 とする。

ここで、プログラム 1 に対して次の質問を与える。

?－mother (hanako , taro).　　…⑧　「hanako は taro の母親か」

まず、この質問⑧のゴール mother(hanako,taro)と同じ形の節、すなわちファンクターが mother であり、引数を 2 つ持つ規則の頭部あるいは事実の探索が行われる。すると、先程導入した母親についての規則⑦がこれに該当し、ユニフィケーションが行われる。規則に現れる同じ変数はすべて同じものを表していることから、規則⑦中のすべての X には hanako が、すべての Y には taro が代入される。その結果、次の規則が得られる。

mother (hanako , taro)：－parent (hanako , taro) , woman (hanako).　　…⑨

この規則は「hanako が taro の母親であることを示すためには、hanako が taro の親であり、かつ hanako が女性であることを示せばよい」ということを表している。すなわち、

[17] 新田克己・佐藤泰介（1986）　pp. 12-16 を参考にした。

証明すべき問題が「hanako が taro の親であること」と「hanako が女性であること」の 2 つに分解されたことになる。これらはそれぞれ次のように表される。

?－parent (hanako , taro).　　…⑩　「hanako は taro の親か」

?－woman (hanako).　　…⑪　「hanako は女性か」

Prolog では、左側にあるゴールが先に推論される。そのため規則⑨のゴールとして、質問⑩・質問⑪の順番で推論が行われるが、質問⑩は事実①と、質問⑪は事実④と一致するため、それぞれ真であることは明らかである。したがって、証明すべきゴールがなくなり、質問⑧の内容が真であることが示されたということになる。こうして証明が成功するとき、Prolog は「yes」と表示する。（[新田，佐藤] pp. 12-16）

(2)　バックトラック[18]

ファンクターの形と引数の数を手掛かりに節の探索を行うユニフィケーションにおいて、該当する節が複数ある場合、それらの中からどれを選ぶべきかが問題となるが、Prolog ではより上にある（先に記述されている）節を優先的に選ぶ決まりになっている。

一方で、プログラムを構成する節の中には、証明にとって有用なものもあれば、全く役に立たないものもあり、前者が常に上にあるとも限らない。したがって、この決まりに従った結果、証明を成功に導く節が揃っているにも関わらず、それより上にある間違った節を選択し、証明が失敗するといったことが起こりうる。

そこで Prolog には、計算過程を途中まで後戻りして、その時点で別の節を選択する機能が備わっている。この機能をバックトラックという。

ここでは、バックトラックが現れる例として、(3)のプログラム 1 に対する次の質問について検討する。

?－father (X, taro).　　…⑫　「X は taro の父親か」

変数 X は定まっていないものを表していることから、質問⑫は「taro の父親である X は存在するか」あるいは「taro の父親は誰か」を尋ねるものであるともいえる。

この場合も(3)での mother に関する質問⑧と同様に、まず質問⑫のゴール father(X,taro) と同じ形の節を探索し、その結果見つかった規則⑥とのユニフィケーションが行われる。その結果、規則⑥中のすべての Y に taro が代入され、次の規則が得られる。

father (X, taro) : －parent (X, taro) , man (X).　　…⑬

この規則は「X が taro の父親であることを示すためには、X が taro の親であり、かつ X が男性であることを示せばよい」ということを表している。

初めに、新たに得られた規則⑬のゴールから次の質問が与えられ、ユニフィケーションが行われる。

---

[18] 新田克己・佐藤泰介（1986）　pp. 31-36 を参考にした。

?－parent (X, taro).　　…⑭　「X は taro の親か」

ファンクターが parent でかつ引数を 2 つ持つ節のうち、一番上にあるのは事実①であることから、ユニフィケーションの結果、質問⑭の変数 X に hanako が代入される。

parent (hanako , taro).　　…①　「hanako は taro の親である」

ここで、質問⑭は規則⑬のゴールであり、また、規則の中の同じ変数はすべて同じものを表しているため、規則⑬の X すべてに hanako が代入されることになる。すると、次の規則が得られる。

father (hanako , taro)：－parent (hanako , taro) , man (hanako).　　…⑮

この規則の 1 つ目のゴールはまさしく事実①であり、すでに真であることが示されている。ゆえに、次に真であることを証明すべきものは 2 つ目のゴールであるから、新たな質問が与えられる。

?－man (hanako).　　…⑯　「hanako は男性か」

しかし、この質問のゴールとユニフィケーションできる節は、プログラム 1 には存在しない。man をファンクターに持つ事実③・⑤のいずれとも、引数が一致しないからである。したがって、X を hanako とする、この証明は失敗となる。

このようなとき、証明が成功するかもしれない別の選択肢について調べるために、バックトラックが発生する。バックトラックでは、直前の節の選択が間違いだったとみなして、その時点に遡り、別の節を選択した上で再度証明を試みることになる。

father に関する質問⑫における、証明が失敗する直前の節の選択とは、質問⑭のユニフィケーションの対象として、事実①を選んだことである。

?－parent (X, taro).　　…⑭　「X は taro の親か」

parent (hanako , taro).　　…①　「hanako は taro の親である」

この選択が間違いだったとみなされ、新たな節の探索が行われる。すると、質問⑭とユニフィケーションが可能なもう一つの節が見つかる。

parent (shinji , taro).　　…②　「shinji は taro の親である」

ユニフィケーションの結果、質問⑭の X、すなわち規則⑬のすべての X に shinji が代入され、次の規則が得られる。

father (shinji , taro)：－parent (shinji , taro) , man (shinji).　　…⑰

規則⑮と同様に、この規則の 1 つ目のゴールは真であることが示されているから、次に与えられる質問は次のようになる。

?－man (shinji).　　…⑱　「shinji は男性か」

なお、この質問は事実③と一致している。

man (shinji).　　…③　「shinji は男性である」

以上から、X が shinji であること、すなわち shinji が taro の父親であることが証明され、Prolog は「X=shinji」と表示する。これは、X が shinji であるとき質問のゴールが真

になることを示すものである。（[新田，佐藤] pp. 31-36）

第 3 章　否定の述語 not

論理プログラミング言語 Prolog は論理式の証明を計算とみなす一方で、否定を明示することができないという欠点を抱えている。このことは、証明のための推論規則を最大限に活用することができないということを意味する。

そこで、Prolog において否定を実現する手段の 1 つとして挙げられるのが、証明の失敗を否定とみなす「失敗としての否定」である。

第 1 節　失敗としての否定[19]

否定的な情報の解釈に関して、閉世界仮説という仮説がある。この仮説においては、ある肯定文に対する証明が存在しない場合、その文の否定が真であると仮定される。例えば、hanako・shinji・taro から構成される家族に jiro はいないことから「jiro はこの家族の一員である」という肯定文の証明は存在せず、その否定である「jiro はこの家族の一員ではない」が仮定されることになる。なお、このことは暗黙的に立証されているため、予め明示される必要はない。すなわち、家族の一員であることが明言されている者以外は言うまでもなく家族に含まれないということを意味しており、極めて自然な考え方であるといえる。

このような仮説に基づく否定の形式が、失敗としての否定である。例えば、P という論理式が偽であることを示したい場合、まず P を証明しうる選択肢をすべて探索する。そして、もしそれらすべてが失敗に終わるならば P でない、すなわち P が偽であることが推論されることになる。

閉世界仮説を Prolog に導入する利点は、真である情報よりもはるかに多いであろう偽である情報を逐一明示する必要がないため、簡潔なプログラムの記述が可能であること、そして何より、その表現に際し特別な機能の追加を必要としないということである。（[Li] pp. 122-124）

第 2 節　not の定義

失敗としての否定は、not という述語で表現される。

not の定義は、次の 2 つの節から構成される[20]。

```
not (P) : －call (P) , ! , fail.
not (P).
```

---

[19] Li, D. (1985) pp. 122-124 を参考にした。

[20] 参照，［新田，佐藤］p. 46。

なお、1 つ目の節（規則）に現れている「call(P)」「fail」はいずれも組込み述語[21]であり、それぞれ「変数 P に代入された項をゴールとみなして証明する」「必ず失敗する」を表している。

この定義において特に重要な働きをしているのは、記号「!」である。この記号はカットと呼ばれ、計算の順序を制御する唯一の機能として Prolog に組み込まれているものである。

(1) カット[22]

カットは節の本体中に使われ、次のような効果をもたらす。

(i) カットを含む節のゴールが通常の順番（左から右）で計算されるときは、カットの存在は無視される（必ず真であるともいえる）。

(ii) カットの直後にあるゴールの証明が失敗してバックトラックが起こったとき、カットの前のゴールへのバックトラックはできず、このカットを含む節を選択したゴールの証明が失敗となる。

ここで、カットの有無が計算にどのような影響を与えるかを検討する。

まず、カットを含まない例として、次の 4 つの節から成るプログラムを想定し、これをプログラム 2 とする。

a (X) : －b (X) , c (X).　　…①

「a(X)が真であるためには､b(X)と c(X)がともに真であればよい」

a (X) : －d (X).　　…②

「a(X)が真であるためには、d(X)が真であればよい」

b (X).　　…③　「b(X)は真である」

d (X).　　…④　「d(X)は真である」

このプログラム 2 に対して次の質問を与える。

?－a (X).　…⑤　「a(X)は真か」

この場合､a(X)を頭部に持つ 2 つの規則のうち､上にあるものが先に選ばれることから、規則①がユニフィケーションされる。そして、その本体のゴールのうち左にあるものから順に計算されるため、続いて次の質問が与えられる。

?－b (X).　…⑥　「b(X)は真か」

質問⑥のゴールが事実③と一致することから、b(X)の証明は成功する。次に証明すべきゴールは c(X)であるが、このゴールとのユニフィケーションが可能な節はプログラム 2 に存在しない。したがって c(X)の証明は失敗し、バックトラックが発生する。

---

21 予め処理系に組み込まれている述語。参照，［新田，佐藤］pp. 68-79。

22 新田克己・佐藤泰介（1986） pp. 40-44 を参考にした。

この直前の選択、すなわち質問⑥において再び節の探索が行われるが、事実③の他にユニフィケーション可能な節がないため、b(X)の証明も失敗する。するとまたバックトラックが起こり、さらに前の選択が行われた時点である最初の質問⑤に戻ることになる。

このとき、規則①以外に a(X)を頭部に持つ節である規則②が選ばれ、そのゴール d(X)が事実④と一致し、a(X)の証明が成功となる。

以上の計算が行われるプログラム 2 に対し、例えば規則①とそれを修正した次の規則

a (X) : －b (X) , ! , c (X).　　…①'

を入れ替えたものをプログラム 3 とし、同様の質問を行うとどうなるか。

```
a (X) : －b (X) , ! , c (X).　　…①'
a (X) : －d (X).　　…②
b (X).　　…③
d (X).　　…④
?－a (X).　…⑤
```

まず、規則①'が選択・ユニフィケーションされ、そのゴールである b(X)の証明は事実③が存在することにより成功する。カットは無視されるため、続いて c(X)の証明が試みられるが、ユニフィケーション可能な節が存在しないため失敗する。

ここまではプログラム 2 における計算と同じである。しかし c(X)の証明に失敗し、バックトラックが起こるときに違いが現れる。

このとき、c(X)がカットの直後のゴールであるために、バックトラックが発生しない。そればかりか、カットを含む節（規則①）を選択したゴール、すなわち a(X)の証明が失敗したとみなされる。そのため、証明を成功に導くはずの規則②を選び直すことなく、Prolog は証明失敗を表す「no」を表示する。

このように、バックトラックを禁止することによって、調べることができる節を省略する機能がカットである。この働きによって、証明が成功するかもしれない選択肢を消してしまう恐れはあるが、逆に、証明が成功する見込みのない選択肢を省くことができる。

not は、この特性を利用してわざと証明の失敗を引き起こすことにより、失敗としての否定を表現するものである。（[新田，佐藤] pp. 40-44）

(2)　not を使う計算[23]

not がどのような働きをするのかを検討するため、第 2 章で導入したプログラム 1 に not の定義を加えた、次のプログラムを用いる。

```
parent (hanako , taro).　　…①
parent (shinji , taro).　　…②
```

[23] 新田克己・佐藤泰介（1986）　pp. 46-48 を参考にした。

man (shinji). …③

woman (hanako). …④

man (taro). …⑤

father (X,Y) : －parent (X,Y) , man (X). …⑥

mother (X,Y) : －parent (X,Y) , woman (X). …⑦

not (P) : －call (P) , ! , fail. …⑧

not (P). …⑨

これをプログラム 4 とする。

まず、プログラム 4 に対して次の質問を与える。

?－not (mother (hanako , taro)). …⑩

プログラム 4 の中から、質問⑩のゴールと同じファンクター及び同じ数の引数を持つ節（⑧と⑨）のうち、より上にある規則⑧が選ばれ、ユニフィケーションが行われる。その結果、規則⑧の変数 P に mother(hanako,taro)が代入される。

not (mother (hanako , taro)) : －call (mother (hanako , taro)) , ! , fail. …⑪

さらに、より左にあるゴールの証明が先に試みられることから、次の質問が与えられる。

?－call (mother (hanako , taro)). …⑫

call により、P に代入された項である mother(hanako,taro)がゴールとみなされ、その証明が行われる。なお、これは第 2 章第 3 節(1)においてすでに示されたように真であるから、質問⑫の証明は成功する。すなわち、規則⑪の 1 つ目のゴールが真であることが示されることになる。

しかし、規則⑪の 2 つ目のゴールであるカットは無視されるものの、さらにその次のゴールである fail の証明が失敗する。そして、このとき起こるはずのバックトラックはカットの効果によって禁止され、このカットを含む規則⑧を選択した質問⑩の証明が（事実⑨を選び直すことなく）失敗したとみなされる。

したがって、mother(hanako,taro)が真であるのに対し、not(mother(hanako,taro))は偽となる。

では、このように計算される質問⑩とは逆に、not の引数が失敗する次の質問をプログラム 4 に与えるとどうなるか。

?－not (father (hanako , taro)). …⑬

質問⑩の場合と同様に、最初に規則⑧が選択され、ユニフィケーションが行われる。その結果、規則⑧の変数 P に father(hanako,taro)が代入され、質問が次々に与えられる。

not (father (hanako , taro)) : －call (father (hanako , taro)) , ! , fail. …⑭

?－call (father (hanako , taro)). …⑮

?－father (hanako , taro). …⑯

ここで、第 2 章第 3 節(2)において示されたように質問⑯の証明は失敗するため、質問

⑮、すなわち規則⑭の 1 つ目のゴールが失敗することになり、バックトラックが起こる。すると、計算がこの直前の選択に後戻りし、質問⑬のユニフィケーションの新たな対象として、事実⑨が選ばれる。そして、father(hanako,taro)を事実⑨の変数 P に代入して得られる次の事実は、最初の質問⑬のゴールと一致しているため、質問⑬の証明は成功となる。

not (father (hanako , taro)).　　…⑰

したがって、father(hanako,taro)が偽であるのに対し、not(father(hanako,taro))は真となる。

以上のように、P の証明が成功するとき not(P)の証明は失敗し、P の証明が失敗するとき not(P)の証明が成功することから、not(P)はまさしく P の否定を表現するものであるといえる。（[新田，佐藤] pp. 46-48）

第 3 節　not の欠点[24]

以上のような形式で否定を表現することができるとはいえ、not は決して完璧な手段というわけではない。Prolog の特性に起因する欠点を抱えており、場合によっては深刻な矛盾を引き起こす可能性がある。

ここでは、もう一度プログラム 4 を用いてその例を示す。

parent (hanako , taro).　　…①
parent (shinji , taro).　　…②
man (shinji).　　…③
woman (hanako).　　…④
man (taro).　　…⑤
father (X,Y) : －parent (X,Y) , man (X).　　…⑥
mother (X,Y) : －parent (X,Y) , woman (X).　　…⑦
not (P) : －call (P) , ! , fail.　　…⑧
not (P).　　…⑨

このプログラム 4 に対して、not を含む次の質問を与える。

?－parent (X, taro) , not (man (X)).　…⑩「X は taro の親であり、かつ男性でないか」

この質問を言い換えると「taro の親であり、かつ男性でない X は存在するか」となる。

Prolog のルールにより、左にあるゴール parent(X,taro)の証明が最初に行われる。ユニフィケーション可能な節を探索した結果、より上にある節を優先的に選ぶという Prolog のもう 1 つのルールによって事実①が選択される。こうして質問⑩の 1 つ目のゴールが証明されたことになり、その X すべてに hanako が代入され、次のゴールを証明すべく、新たな質問が与えられる。

---

[24] Sterling, L. & Shapiro, E. (1988) pp. 213-217 を参考にした。

parent (hanako, taro) , not (man (hanako)).　　…⑪

?－not (man (hanako)).　　…⑫　　「hanako は男性でないか」

ユニフィケーション可能な節のうち、より上にある規則⑧が選択・ユニフィケーションされ、変数 P に man(hanako)が代入される。

not (man (hanako)) : －call (man (hanako)) , ! , fail.　　…⑬

さらに組込み述語 call により、その引数である man(hanako)が次に証明すべきゴールとなるが、それとユニフィケーション可能な規則・事実はプログラム 1 に存在しない。すると証明失敗によってバックトラックが起こり、直前の選択の時点に計算が戻される。そこで、規則⑧でなく事実⑨が選び直され、その変数 P に man(hanako)が代入される。

not (man (hanako)).　　…⑭

この事実⑭は質問⑫のゴールと一致し、これで質問⑩のゴールがすべて証明されたことになる。したがって、質問⑩は真となる。

これとは別に、プログラム 1 に対してもう 1 つ質問を与えてみる。

?－not (man (X)) , parent (X, taro).　…⑮　「X は男性でなく、かつ taro の親か」

この質問の意味は明らかに質問⑩と同じであるが、ただ、ゴールの順序が逆になっている。

左にあるゴール not(man(X))の証明が先に行われ、規則⑧が選択・ユニフィケーションされることにより、その変数 P に man(X)が代入される。

?－not (man (X)).　　…⑯

not (man (X)) : －call (man (X)) , ! , fail.　　…⑰

そして、call により証明すべきゴールとなった man(X)は、変数である X には何を代入してもよいことから、プログラム 1 中の man をファンクターに持つ節のうち最も上にある事実③とユニフィケーション可能である。

?－man (X).　　…⑰

man (shinji).　　…③

これにより、規則⑬の 1 つ目のゴールの証明が成功したことになる。

しかし、本章第 2 節で示したように、その 3 つ目のゴールである fail が失敗してバックトラックが発生することにより、カットの効果が発揮される。これによって、このカットを含む節である規則⑧を選択した節、すなわち質問⑯の証明が失敗したものとみなされる。さらに、それ以前に節の選択はなされていないため、さらなるバックトラックは不可能である。したがって、1 つ目のゴールの証明に失敗したことにより、質問⑮は偽となる。

しかし、プログラム 1 には「男性でなく、かつ taro の親である」hanako が存在しており、質問⑮もまた真として評価されるべきはずである。そうすると、質問⑮の計算は誤った推論を行ったということになる。

以上の例から明らかなように、より上の節を、より左のゴールを選ぶという Prolog の特性により、変数を含む質問のゴールの順序が入れ替わっただけで、異なる結論が導き出

されることがある。not はこのように、失敗としての否定を表現するものとしては不完全であり、それを用いてプログラムを記述する際には節及びゴールの順序にも気を配らなければならない。

しかし、Prolog では元々、そのような順序が推論過程を変化させることはあっても、結論に影響を与えることはない。したがって、手に入れた情報の順番次第でプログラムの意味が変わり、推論が誤った方向に進みうるというのは、Prolog においては望ましいことではない。（[Sterling & Shapiro] pp. 213-217）

おわりに

本論文では、論理プログラミング言語 Prolog の計算の手順について説明し、失敗としての否定を表現するための述語 not を紹介した。そして not が、否定としての役割を部分的に担いうること、同時に、not の引数に代入された項が変数を含む場合には、節やゴールの順序が結論に影響を与えうるという欠点を持っていることを示した。

not がもたらしたような事例は、人間が物事を判断する上でも起こりうることである。すなわち、カットによって他の選択肢が省略されたことは、人間が思い込みによって結論を急ぐ場合と類似している。推論の効率性という意味では有利に働く場合もあるかもしれないが、Prolog の推論において示されたのと同様に、このような判断が誤った結論を導くこともある。この点では、not の欠点を招いた Prolog はいわば人間にとっての反面教師として振る舞っているともいえる。ただし、人間における事例は必ずしも情報を得た順番に依存するものではない。また、人間が勘違いや物忘れといった偶発的な原因によって思い込みをしてしまうのに対し、not の欠点は、条件さえ整えば Prolog の正常な推論手続きの結果として必然的に引き起こされるものである。したがって、結果的に共通する部分があるとはいえ、人間の思考過程を表現するという観点からも not の欠点は望ましくない。

ちなみに、この欠点を克服するための解決法はすでにいくつか提案されている。例えば、not の引数に代入された項が変数を含まなくなるまで、not の計算を先送りするという方法がある[25]。上記の質問⑮の場合、最も左のゴールである not(man(X))の証明は延期され、その変数 X に定数を代入すべく、右のゴール parent(X,taro)の証明が先に行われる。この証明の順番は、質問⑮とはゴールが逆順である質問⑩の場合と同じであり、当然、結論も同じく真になる。しかし、プログラムの内容次第では、not がいつまで経っても計算されないという事態が起こりうるため、効率性が損なわれる可能性は否定できない。

この方法のように、Prolog の処理系そのものに対する変更を必要とするものは高度な情報科学の知識に基づいているため、本論文で詳しく扱うことはできなかった。推論の正しさと効率性を兼ね備えた否定の形式は実現可能か。この点を、今後の課題としたい。

---

[25] 参照，［Naish］pp. 4-5。

参考文献

・井田哲雄（1991）『計算モデルの基礎理論』(岩波講座ソフトウェア科学 12)，岩波書店
・川合慧（1988）『プログラミングの方法』(岩波講座ソフトウェア科学 2)，岩波書店
・Sterling, L. & Shapiro, E.（1988）『Prolog の技芸』(松田利夫訳)，構造計画研究所
・武市正人（1994）『プログラミング言語』(岩波講座ソフトウェア科学 4)，岩波書店
・所真理雄（1988）『計算システム入門』(岩波講座ソフトウェア科学 1)，岩波書店
・Naish, L.（1986）. *Negation and control in Prolog.* Lecture notes in computer science ; 238. Berlin, Heidelberg, New York, London, Paris, Tokyo. Springer-Verlag.
・中村克彦（1985）『Prolog と論理プログラミング』，オーム社
・新田克己・佐藤泰介（1986）『Prolog』(人工知能言語シリーズ/1)，昭晃堂
・Li, D.（1985）『PROLOG データベース・システム』(安部憲広訳)，近代科学社