数学入門公開講座

平成５年８月３日（火）から８月６日（金）まで

京都大学数理解析研究所

講師及び内容

1．論理とコンピュータ（6時間）

京都大学数理解析研究所・助手　服　部　隆　志

コンピュータのプログラムは多くの場合、コンピュータの動作を順序正しく指定するような形になっています。それに対して、論理プログラミングと呼ばれる方法は、数学的な論理式をそのままプログラムとして使います。ここでは、論理プログラミングがどのように働くのか、その数学的基礎はどうなっているのかを考えていきます。また、論理プログラミングに関係する話題もいくつか紹介します。

2．組紐群について（6時間）

京都大学数理解析研究所・助教授　織　田　孝　幸

組紐は太古からある日本の伝統工芸であるが、数学として組紐群（braid group）の研究は1930年ごろから、トポロジーの問題である結び目（knot）や絡み輪（link）の研究の助けとするため、E.Artin によって系統的に調べ始められた。

ここ10数年これが数学の他のいろいろな分野（物理数学、共形場の理論、C*環、整数論 etc.）とも関連することが分かってきて、多くの研究がなされている。

組紐群について基本的な事実について簡単な入門的話しをする。

3．渦運動と乱流（6時間）

京都大学数理解析研究所・助手　大木谷　耕　司

流体力学に現われる基礎方程式の解の振舞いについては、数学的には未知な点が多い。ここでは、渦運動を中心に基礎方程式の解説からはじめ、現在までに知られている厳密解を紹介する。さらに、数値解析的に得られた解を通して、乱流について話しをする。乱流の‘大きな’カオスとしての側面にも触れる予定である。

時　間　割

時間＼日	8月3日（火）	4日（水）	5日（木）	6日（金）
11：00～12：30	服部	服部	服部	服部
12：30～13：45	休憩			
13：45～15：15	織田	織田	織田	織田
15：15～15：30	休憩			
15：30～17：00	大木谷	大木谷	大木谷	大木谷

平成5年度(第15回)数学入門公開講座テキスト(京都大学数理解析研究所, 平成5年8月3日～8月6日開催)

論 理 と コ ン ピュ ー タ

京都大学数理解析研究所・助手　服 部 隆 志

1993, AUGUST 3,4,5,6,　　11:00 ~ 12:30

論理とコンピュータ

服部隆志

コンピュータの能力というと、何万人もの銀行口座の残高を計算したり、複雑な流体力学の問題を解くところを想像します。しかし、コンピュータはそのような数値の計算だけでなく、論理的な命題を取り扱い、推論を行なう能力も備えているのです。ただし、推論を行なうといっても、SF に出てくるコンピュータのように自分で勝手に考えて答を出してくれるわけではありません。どのような推論を行なうかということは、人間がプログラムしてやらなければならないのです。さて、そのようなプログラムを作るためには、まず数理論理学を学ばなければなりません。これは、論理的な思考を、少しの曖昧さもないように厳密に表現するための道具です。そして、数理論理学に基づいたプログラミング言語として、PROLOG があります。また、さらに高度な推論や学習をするためのシステムも存在します。

1 命題論理

1.1 論理式と解釈

「私は男です」、「これは犬です」のように、真偽がはっきりと定まる文を命題と呼びます。命題に「ではない」を付けたもの、2つの命題を「または」、「かつ」、「ならば」などで組み合わせたものも命題になります。このようにして出来た命題の真偽を考える時、元の命題の具体的な内容に関わらず、その真偽だけが問題になります。たとえば、「私は男であるか、または男でない」という命題は、「私は男である」という命題の真偽に関わらず、真です。これを、「私は男である」というのを「これは犬です」に置き換えて、「これは犬であるか、または犬でない」としても同じことです。ですから、論理学では具体的な命題の代わりに、$A, B, C, \cdots$ といった記号で命題を表し、$\vee, \wedge, \rightarrow$ などの記号で命題を組み合わせて論理式を作ります。こうすれば、論理式 $A \vee \neg A$ は A の真偽に関わらず、常に真であると書くことが出来ます。

論理式を扱う時は、記号と、それが持つ意味を混同しないように気をつけなければいけません。上の論理式 $A \vee \neg A$ は、$A, \vee, \neg, A$ という4つの記号を並べて書いたもので、それ以上でもそれ以下でもありません。この論理式がどのような意味を持つかということを定めるのは、解釈です。解釈とは、命題の集合から真理値の集合 $\{\mathrm{t}, \mathrm{f}\}$ への写像です。今、解釈 I があって $I(A) = \mathrm{t}$ だとすると、A は真の命題であると考えることに相当します。

解釈 I は、論理式の集合から真理値の集合への写像に拡張することが出来ます。たとえば、$I(\neg A), I(A \vee B)$ の値は、$I(A), I(B)$ の値によって表1のように決まります。

論理式は解釈によって真になったり偽になったりしますが、$A \vee B$ のように、真になるような解釈が存在する論理式を充足可能であるといいます。その中でも特に $A \vee \neg A$ の

$I(A)$	$I(B)$	$I(\neg A)$	$I(A \vee B)$
t	t	f	t
t	f	f	t
f	t	t	t
f	f	t	f

表 1: 解釈の拡張

ように、どんな解釈を考えても常に真になるような論理式を、恒真であるといいます。また、ある論理式αを真にするような解釈Iを、αのモデルと呼び、$I \models \alpha$と書きます。もし論理式βが、αの任意のモデルで真になれば、βはαの論理的帰結であるといいます。

1.2 証明

ある論理式を証明する時には、すでに証明された論理式を使います。論理式の集合Lを使えば、論理式αが証明できる時、$L \vdash \alpha$と書きます。しかし、そのLに含まれる論理式を証明するためには…と考えていくと、出発点になる論理式として公理が必要になることが分かります[1]。また、証明の道筋を誰が見ても納得できるようにするために、すでに証明された論理式から新しく論理式を証明する方法として、推論規則を明確に与えます。公理から推論規則を用いて証明できる論理式を定理といいます。公理と推論規則を決めれば、それによって証明できる範囲も決まります。ここで、命題の集合、命題から論理式を構成する方法、公理の集合、推論規則の集合を組み合わせたものを理論と呼ぶことにします。

公理と推論規則は、解釈とは無関係に、勝手に決めることができます。ですから、公理や、公理から証明できる論理式が任意の解釈で真であるとは限りません。ただし、公理は証明の出発点ですから、ある理論の中では、その公理を真にするような解釈だけに限定して考えることにします。つまり、論理式αが理論Tで真であるとは、αがTの公理の論理的帰結である時とします。そして、理論Tの任意の論理式αに対して、αがTで証明できるならばαがTで真である時、Tは健全であるといいます。逆に、αがTで真ならば必ず証明できる時、Tは完全であるといいます。

2 述語論理

2.1 述語と項

命題論理では、「宮沢首相は男である」という命題と「宮沢りえは男である」という命題は全く無関係な命題であり、それ以上分解できないものと考えます。しかし、この場合、「…は男である」という部分は共通であり、「…」のところに他の人を入れても真偽を決めることができますから、述語論理では「は男である」の部分を述語と呼び、「…」

[1]自然推論という体系では公理がなく、推論規則だけで証明を行ないます。

の部分をその述語の引数と呼びます。また、「…」に入れることができるものの集合をユニバースといいます。

ユニバースの要素のどれかを表す記号を変数といいます。「…は男である」という述語が記号 p で表され、x が変数であるとすると、論理式 $p(x)$ は「x は男である」ということを表しています。しかし、これでは x に何が入っているのか分からないので、真偽を決めることができません。変数の集合からユニバースへの写像を変数割当といいます。変数割当 V が、$V(x)=$宮沢りえだとすると、$p(x)$ は、「宮沢りえは男である」ということを表します。

命題の中には、ユニバースの要素を間接的に使うものもあります。例えば、「長島一茂の父親は男である」という命題を考えると、男であるかどうかが問題になるのは長島茂雄ですが、ここでは「長島一茂の父親」という形で間接的に示されています。「…の父親」という部分は「…」に他の人を入れても同じように誰か一人の人を指し示しますから、これを関数と呼ぶことにします。「…の父親」という関数が記号 f で表されるとすると、$f(x)$ は x の父親を表します。引数が 0 個の関数は、ユニバースの中の特定の要素を指すので、定数と呼ばれます。例えば、人の名前は (同姓同名はいないとすると) その人を指す定数です。

変数と関数記号から、次のようにして項を作ることができます。

1. x を変数とすると、x は項である。
2. f を定数 (0 個の引数を持つ関数記号) とすると、f は項である。
3. f を n 個の引数を持つ関数記号 $(n>0)$、$t_1,\cdots,t_n$ を項とすると、$f(t_1,\cdots,t_n)$ は項である。

述語の引数には、任意の項を入れることができます。述語と項を組み合わせたものを原子式といいます。原子式は命題論理の時と同じように、$\neg,\vee,\wedge$ などで組み合わせることができます。原子式は正リテラルということもあり、これに対して原子式に否定 $\neg$ を付けたものを負リテラルといい、正リテラルと負リテラルをまとめてリテラルといいます。

述語論理においても、論理式の意味を定めるのは解釈です。まず、関数記号の解釈というのは、n 個の引数を持つ関数の場合、ユニバース U の要素 n 個から 1 個の要素を割り当てるような写像 $U^n \longrightarrow U$ です。また、述語記号の解釈というのは、n 個の引数を持つ述語の場合、ユニバース U の要素 n 個に対して真理値を定めるような写像 $U^n \longrightarrow \{\mathrm{t},\mathrm{f}\}$ です。変数割当と解釈を合成することによって、論理式の真偽を定めることができます。

2.2 「すべて」と「ある」

命題の中には、同時にユニバースの複数の要素を対象にするものもあります。たとえば、「すべての人の父親は男である」というのは、$\forall x p(f(x))$ と書くことができます。また、「父親が男であるような人が存在する」というのは、$\exists x p(f(x))$ と書くことができます。

$\forall x\alpha$ と $\exists x\beta$ という論理式に対して、α,β の中に現れる x は $\forall$ で束縛されているといい、逆に、束縛されていない変数は自由であるといいます。$\forall x p(f(x))$ は、今考えてい

る変数割当の x の値を、どのユニバースの要素と取り替えても $p(f(x))$ が真になる時、真になります。自由な変数を含まない論理式は閉論理式といい、変数割当とは関係なく、解釈だけによって真偽が定まります。論理式 α に出てくる自由な変数をすべて $\forall$ で束縛して閉論理式にしたものを全称閉包といい $\forall(\alpha)$ と書きます。$\forall$ を $\exists$ に代えたものは特称閉包といい、$\exists(\alpha)$ と書きます。また、全く変数を含まない項や論理式を基礎項、基礎論理式といいます。

3 論理プログラミング

論理式をコンピュータで扱おうという最初の試みは人工知能の研究においてなされ、そこでの中心的な目標は、定理を自動的に証明することでした。一般的な数学の定理を証明することに関しては、現在でも満足すべき成果は得られていません (おかげで数学者はまだ失業していません) が、1965 年に Robinson が導出原理を発表し、1972 年に Kowalski と Colmerauer はこれを利用すれば「論理をプログラミング言語として使うことができる」という考えに到達しました。そして PROLOG (PROgramming in LOGic の略) という言語が設計され、マルセイユ大学で最初の処理系が作られました。

3.1 PROLOG プログラム

PROLOG のプログラムは次のような形をしています。

```
append([], Y, Y).
append([V|X], Y, [V|Z]) :- append(X, Y, Z).
```

これは二つのリストを結合するプログラムです。このプログラムを走らせるには、質問を与える必要があります。質問の仕方によって、いろいろな結果が得られます。以下の例で、'| ?-' の後にあるのがユーザが入力した質問で、その下が出力された結果です。

```
| ?- append([a,b,c], [d,e,f], Z).
Z = [a,b,c,d,e,f] ?
yes
```

これは、[a,b,c] と [d,e,f] というリストを結合すると、[a,b,c,d,e,f] というリストになることを示しています。PROLOG の面白い点は、入力と出力が決まっていないということです。次に示すように、質問を変えると、同じプログラムで逆向きの計算ができます。

```
| ?- append([a,b,c], Y, [a,b,c,d,e,f]).
Y = [d,e,f] ?
yes
```

これは、[a,b,c] というリストに何を結合すると、[a,b,c,d,e,f] というリストになるかを求めると、[d,e,f] というリストであることを示しています。また、次のように解答が複数ある時に、しらみ潰しに探すこともできます。

```
| ?- append(X, Y, [a,b,c]).

X = [],
Y = [a,b,c] ? ;

X = [a],
Y = [b,c] ? ;

X = [a,b],
Y = [c] ? ;

X = [a,b,c],
Y = [] ? ;

no
```

これは、何と何を結合すれば [a,b,c] というリストになるかを求めると、4種類の答が見つかったことを示します。

3.2 宣言的意味

このプログラムは、論理的にはどういう意味を持つか、調べてみます。まず、プログラムを論理式の形に書き直してみます。

$$\forall y \ \ \mathrm{append}(\mathrm{nil}, y, y)$$

$$\forall x \forall y \forall z \forall v \ (\mathrm{append}(\mathrm{cons}(v, x), y, \mathrm{cons}(v, z)) \leftarrow \mathrm{append}(x, y, z))$$

1行目は、空リストと何かを結合しても変化しないということを表しています。2行目は、x と y を結合すると z になるとすれば、x の前に v を付け足したものと y を結合すると、z の前に v を付け足したものになるということを表しています。これに対して、上の例の3番目の質問を論理式の形に書き直すと、次のようになります。

$$\exists x \exists y \ \ \mathrm{append}(x, y, \mathrm{cons}(a, \mathrm{cons}(b, \mathrm{cons}(c, \mathrm{nil}))))$$

PROLOG の動作は、与えられた質問がプログラムの論理的帰結になっているかどうかを判定することです。そのためにはすべての解釈を考える必要があるように思われますが、幸いなことに、節という形の論理式だけを使うことにすれば、ある特殊な解釈を使って論理的帰結かどうかを判定できます。

節とは、リテラルを $\vee$ で結んだ閉論理式です。リテラルが変数を含んでいる時は、節の先頭に $\forall$ があってその変数を束縛しているものとします。$A_1, \cdots, A_m, B_1, \cdots, B_n$ が原子式ならば、

$$\forall (A_1 \vee \cdots \vee A_m \leftarrow B_1 \wedge \cdots \wedge B_n)$$

は

$$\forall (A_1 \vee \cdots \vee A_m \vee \neg B_1 \vee \cdots \vee \neg B_n)$$

と同値なので、節です。

エルブラン・ユニバースとは、基礎項の集合です。エルブラン解釈とは、関数記号の解釈がエルブラン・ユニバース上のそれ自身になっているような解釈です。エルブラン解釈では関数記号の解釈が定められていますから、述語記号の解釈、つまり基礎原子式の真偽を定めることによって、エルブラン解釈が一つ定まります。エルブラン解釈で、モデルになっているものをエルブラン・モデルといいます。節の集合 S のエルブラン・モデルの中で真になる基礎原子式がもっとも少ないものを最小エルブラン・モデルといいます。すると、次の定理が成り立ちます。

[定理] 節の集合 S の最小エルブラン・モデルと、S の論理的帰結となる基礎原子式の集合は等しい。

したがって、質問がプログラムの論理的帰結になっているかどうかは、質問がプログラムの最小エルブラン・モデルで真であるかどうかと言い替えることができます。プログラムの最小エルブラン・モデルは、質問とは関係なく一定ですから、これをそのプログラムの意味と呼びます。さらに、この意味は計算手順による定義ではなくて、論理学的に定義されることから、宣言的意味と呼ぶこともあります。

3.3 導出原理

PROLOG の実際の動作は、最小エルブラン・モデルを計算しているわけではなく、導出原理という一種の背理法による定理証明システムに基づいています。

まず、命題論理の場合を考えてみます。節の集合 S から論理式 $A_1 \wedge \cdots \wedge A_n$ を証明するためには、その否定を考えて $S \cup \{\neg A_1 \vee \cdots \vee \neg A_n\}$ が矛盾することを示せばよいのですが、ここで $\neg A_1 \vee \cdots \vee \neg A_n$ が節の形をしているところと、矛盾がリテラル 0 個の節として表せるところがミソです。一般に、導出という推論規則は次のように書けます。

$$\frac{A \vee \alpha \quad \neg A \vee \beta}{\alpha \vee \beta}$$

ここで重要なのは、正リテラル A と、その否定である負リテラル $\neg A$ を消去していることと、元の論理式が節なら結果も節になることです。$S \cup \{\neg A_1 \vee \cdots \vee \neg A_n\}$ の中から、このような正リテラルと負リテラルの組を探すことによって、次々と導出を繰り返していき、最終的にリテラルをすべて消去することができれば、つまりリテラルが 0 個の節が導ければ、元の節の集合は矛盾していることが分かり、$S \vdash A_1 \wedge \cdots \wedge A_n$ がいえます。

次に述語論理の場合を考えてみます。質問は $\exists(A_1 \wedge \cdots \wedge A_n)$ という形なので、その否定は $\forall(\neg A_1 \vee \cdots \vee \neg A_n)$ となり、ちゃんと節の形になります。導出は、基本的には命題論理の場合と同じですが、今度はリテラルの中に項が現れる分、複雑になります。項を構成している関数記号や変数がぴったり一致していれば導出が適用できるのは明らかですが、そうでなくてもよい場合があります。導出の目的を思い出してみると、証明したい命題の否定が成り立たないことを示すことでした。節の中の変数は $\forall$ で束縛されていますから、変数へある代入をした結果が矛盾すれば、その節に対する反例を示したことになって、目的は達成されます。そこで、項の中の変数に適当な代入をして、二つの項がぴったり一致するようにしてから、導出を適用するようにします。この代入のことを単一化子と

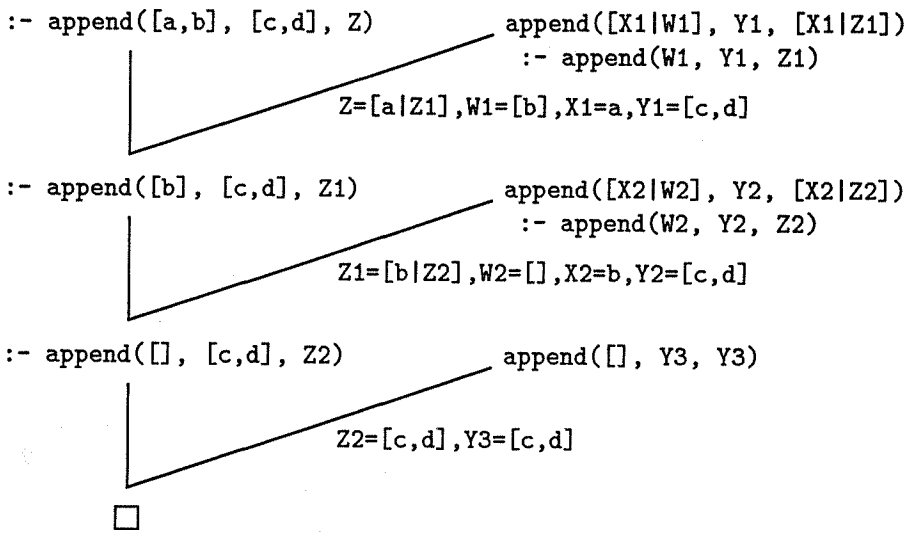


図 1: 導出の例

呼びます。単一化子は複数個あることもありますが、PROLOG では余分な代入を行なわない最汎単一化子を使います。

導出原理の利点は、導出という推論規則を繰り返し適用することによって機械的に証明ができる点です。これは一般の論理式ではうまくいかないので、節だけに限定しているわけですが、もっと効率を良くするため、プログラムは正リテラルが 1 個、ゴール (質問を否定したもの) は正リテラルが 0 個の節に制限します。正リテラルが 0 個または 1 個の節をまとめてホーン節と呼びます。ゴールとプログラム節から導出を行なうと、結果は正リテラル 0 個の節になるので、これもゴールといいます。そして、導出はゴールとプログラム節からゴールを導くものだけに制限します。例えば、図 1 は導出を 3 回繰り返して矛盾が導けた例です。このように最後が矛盾になるような導出列を反駁といいます。あるゴールに対して反駁が存在すれば、その各最汎単一化子をすべて合成したものが、元のゴールに対する反例を表しています。

あるゴールに対して、導出列は一通りではありません。ゴールからどのリテラルを選ぶか、どのプログラム節と組み合わせるかという選択の余地があります。PROLOG では必ずゴールの一番左のリテラルを選び、それと単一化できるようなプログラム節を選びます。単一化できるプログラム節が複数あれば、まず最初の節 α を選んで導出を進め、うまくリテラルをすべて消去できればそれが反駁になりますし、途中でそれ以上導出ができないゴールができてしまうと、α を選んだのが間違いだったのだと考え、α を選ぶ前の状態に戻し、単一化できる 2 番目の節 β を選んで導出を進めます。このように前の状態に戻してやり直すことをバックトラックといいます。

このようにして、反駁が見つかればその時の変数への代入と 'yes' を答として出します。バックトラックを繰り返して、あらゆる可能性を試しても反駁が見つからなかった時は、'no' という答を出します。また、反駁が見つかった時に強制的にバックトラックさ

せることによって、すべての反駁を探すこともできます。

3.4 PROLOG の完全性

プログラム P の成功集合とは、基礎原子式 A で $P \cup \{\neg A\}$ が反駁を持つようなもの全体の集合です。すると、次の定理が成り立ちます。

[定理] プログラム P の成功集合は、その最小エルブラン・モデルに等しい。

これは、A に対する反駁が存在すれば A は最小エルブランモデルで真であり、逆に A が最小エルブランモデルで真なら反駁を見つけることができるということですから、導出原理の健全性と完全性を示しています。しかし、実際の PROLOG の動作は完全性があるとは限りません。その原因は、反駁の探し方にあります。ゴールの中のリテラルと単一化できるプログラム節が複数ある時は、その一つを選んで導出を進め、うまく行かなければ別の選択肢を試してみます。この時、バックトラックはそれ以上導出ができない状態になって初めて行なわれるので、もし無限に導出が可能な選択肢を最初に選んでしまうと、他に反駁 (有限回で矛盾に達する選択肢) があっても、バックトラックが行なわれず、無限に実行を続けてしまいます。このような探索の方法を深さ優先の探索といいます。これに対して、複数の選択肢を並行に調べていく方法を幅優先の探索といいます。幅優先の探索を使えば、完全性を保証することができますが、非常に効率が悪いので、実際にはほとんど使われません。

4 非単調論理

今まで見てきた命題論理、述語論理では、A と $\neg A$ の両方が証明できる時、その理論は矛盾しているといいます。矛盾した理論では任意の論理式が証明できてしまうので、役に立たない理論であるといえます。しかし、現実の世界では一見矛盾する命題も多く見られます。例えば、「鳥は飛べる」、「ペンギンは鳥である」、「ペンギンは飛べない」という 3 つの命題を考えてみますと、「鳥は飛べる」と「ペンギンは鳥である」から「ペンギンは飛べる」という命題が証明できるので、「ペンギンは飛べない」という命題と矛盾します。ところが、この事実から「ペンギンは言葉を話す」という命題が証明できるとは誰も思いません。これは矛盾ではなくて、単にペンギンが例外であるとみなされるわけです。このような命題を扱うためには、非単調論理を使う必要があります。普通の論理では公理を増やせば、そこから証明できる論理式の範囲は広がるのですが、例外があると、「鳥は飛べる」、「ペンギンは鳥である」だけが公理なら証明できていた「ペンギンは飛べる」が、「ペンギンは飛べない」を付け加えることによって証明できなくなります。このように、公理集合を増やしても、証明できる論理式の集合が単調に増加するとは限らない論理体系を非単調論理と呼びます。

4.1 デフォールト推論規則

デフォールト推論規則とは、「α が証明可能で、β が他の証明可能な論理式と矛盾しないならば、γ が証明可能だと仮定する」という形の推論規則です。「x が鳥である」と

いうのを $p(x)$、「x が飛べる」というのを $q(x)$ で表すとすると、

$$\frac{p(x) : q(x)}{q(x)}$$

はデフォールト推論規則の例になります。これは、x が鳥であって、x が飛べるということが他の知識と矛盾しなければ、x は飛べると仮定することを表しています。デフォールト推論規則を使うことによって証明できる定理の集合を拡張世界といいます。拡張世界は一つとは限らず、証明の順序によって異なる拡張世界が得られることがあります。

4.2 ATMS

非単調論理に基づいて、仮説間の整合性を維持するためのシステムとして TMS(Truth Maintenance System の略) と、それを発展させた ATMS(Assumption-based Truth Maintenance System の略) があります。ATMS はそれ自身で推論を行なうわけではなく、他のシステムが推論した結果を受けとり、その時点でどのような拡張世界が可能であるかということを常に計算しなおすようなものです。

ATMS が扱うデータはノードと呼ばれます。ノードはデータ名、ラベル、正当化の 3 つから構成されます。データ名とは、ATMS と通信する推論システムが扱うデータの名前です。ラベルとは、そのノードが表すデータを矛盾なく成り立たせるような環境 (ノードの集合) の集合です。正当化とは、推論システムから送られてくるデータで、

$$a_1, \cdots, a_n \Rightarrow b$$

という形をしています。これは、$a_1, \cdots, a_n$ から b が推論できたということを表していて、b というノードに $(a_1, \cdots, a_n)$ という情報を付け加えます。また、$a_1, \cdots, a_n$ が矛盾することが推論できた場合は、b として矛盾ノード $\perp$ という特別なノードを指定します。

例えば、次のような 4 つのノードがあるとします。

$$\begin{aligned}
N_1 &: \langle \text{ペンギンは飛べる}, \{\{N_1\}\}, \{(N_1)\}\rangle \\
N_2 &: \langle \text{この鳥はペンギンである}, \{\{N_2\}\}, \{(N_2)\}\rangle \\
N_3 &: \langle \text{この鳥は飛べる}, \{\}, \{\}\rangle \\
N_4 &: \langle \text{この鳥は飛べない}, \{\{\}\}, \{()\}\rangle
\end{aligned}$$

N_1, N_2 は正当化として自分自身を持ちます。このようなノードを仮説と呼びます。N_4 は正当化として空リストを持ちます。これは無条件に正しいことを確信できるということなので、このようなノードを前提と呼びます。一般に n 個の仮説があると、その中でどの仮説を仮定するかという選び方は 2^n 個あります。ATMS の動作は、それぞれの仮説の組合せごとに、他のノードがそこから導かれるかどうかを判定することです。そして、その情報を記憶するのがラベルの役割です。ラベルの要素となる環境は、その環境に属する仮説を仮定すれば、正当化が表している推論を行なった時に、そのノードが導けるということを意味しています[2]。

[2]正確には、そのような環境の中で極小のものがラベルの要素になります。

推論システムが「この鳥はペンギンである」と「ペンギンは飛べる」から「この鳥は飛べる」を推論したとすると、正当化として $N_1, N_2 \Rightarrow N_3$ が送られてきます。すると、ATMS はノード N_3 を次のように変更します。

$$N_3 : \langle \text{この鳥は飛べる}, \{\{N_1, N_2\}\}, \{(N_1, N_2)\}\rangle$$

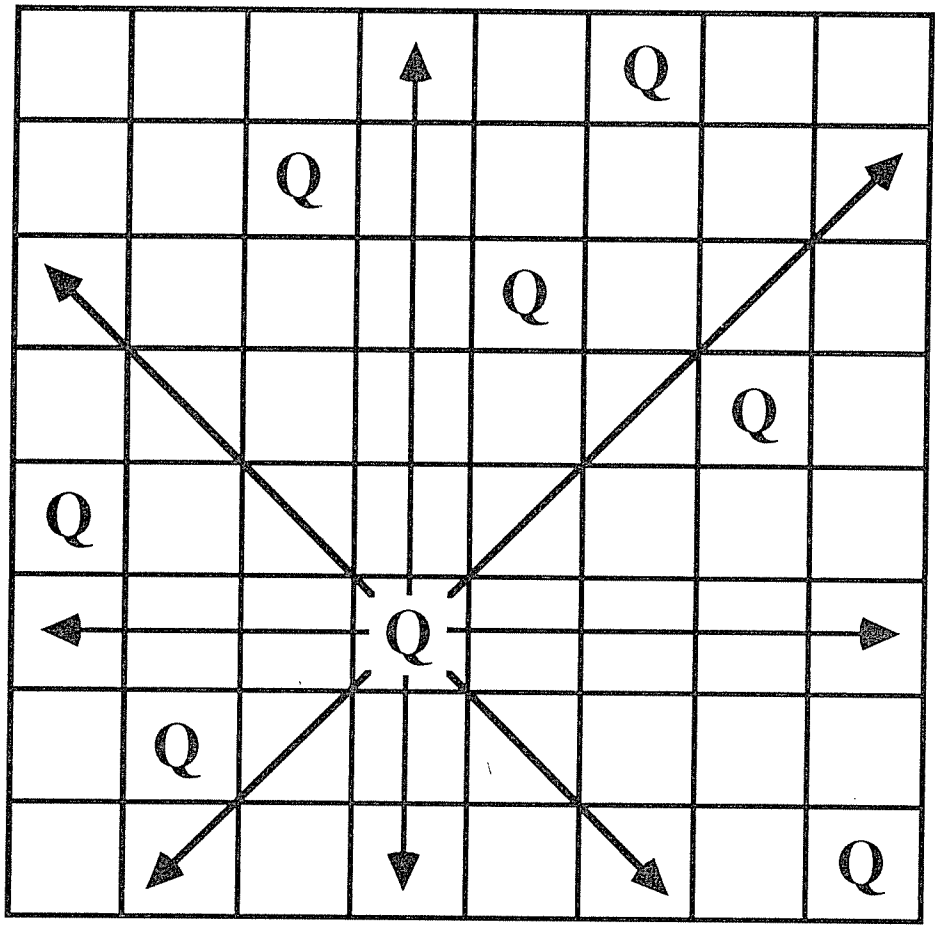
これは、N_1 と N_2 を仮定すれば N_3 が導けることを表しています。

次に推論システムが「この鳥は飛べる」と「この鳥は飛べない」が矛盾していることを発見したとすると、正当化として $N_3, N_4 \Rightarrow \bot$ が送られてきます。今 N_3 のラベルには $\{N_1, N_2\}$ が、N_4 のラベルには $\{\}$ が入っていますから、ATMS は矛盾ノードのラベルとして $\{N_1, N_2\}$ を登録し、すべてのノードのラベルからこれを含むような環境を消去します。これは、N_1 と N_2 を同時に仮定することは許されないということです。そして N_3 は再び変更されて次のようになります。

$$N_3 : \langle \text{この鳥は飛べる}, \{\}, \{(N_1, N_2)\}\rangle$$

ラベルの要素が一つもないので、このノードを導くことができないということが分かります。

このようにして、推論の各段階でどのような仮説の組合せが可能であるかという情報が得られるので、推論システムはこの情報を利用して、推論の戦略を立てることができます。



8 queen 問題の解

8 queen 問題を解くプログラム

```
queen(Answer) :-
    queen-step([1,2,3,4,5,6,7,8],
                              [], Answer).

queen-step([], Done, Done).
queen-step(Yet, Done, Answer) :-
    select(Yet, S, R),
    check(S, Done),
    queen-step(R, [S|Done], Answer).

select([Head|Tail], Head, Tail).
select([Head|Tail], S, [Head|R]) :-
    select(Tail, S, R).

check(New, Olds) :-
    Up is New+1, Down is New-1,
    check-step(Olds, Up, Down).

check-step([], _, _).
check-step([Head|Tail], Up, Down) :-
    Head =\= Up, Head =\= Down,
    Up1 is Up+1, Down1 is Down+1,
    check-step(Tail, Up1, Down1).
```

実行例

```
| ?- queen(X).

X = [4,2,7,3,6,8,5,1] ? ;

X = [5,2,4,7,3,8,6,1] ? ;

X = [3,5,2,8,6,4,7,1] ? ;

                (中略)

X = [5,7,2,6,3,1,4,8] ? ;

no
| ?-
```

解釈の論理式への拡張

$I_A(p(a))$	$I_A(p(b))$	$I_A(p(c))$	$I_A(\neg p(a)\vee p(b)\wedge p(c))$
真	真	真	真
真	真	偽	偽
真	偽	真	偽
真	偽	偽	偽
偽	真	真	真
偽	真	偽	偽
偽	偽	真	真
偽	偽	偽	偽

$\forall x(p(x)\to p(x))$の証明

(1) $p(x)\to((p(x)\to p(x))\to p(x))$ A1

(2) $(p(x)\to((p(x)\to p(x))\to p(x)))\to((p(x)\to(p(x)\to p(x)))$
$\to(p(x)\to p(x)))$ A2

(3) $(p(x)\to(p(x)\to p(x)))\to(p(x)\to p(x))$ MP(1),(2)

(4) $p(x)\to(p(x)\to p(x))$ A1

(5) $p(x)\to p(x)$ MP(3),(4)

(6) $\forall x(p(x)\to p(x))$ GEN(5)