データベース・システム　80－1
人　工　知　能　73－1
（１９９０．１１．８）

逐次単節導出による演繹推論アルゴリズム

小幡　卓
防衛庁

導出操作は本来、特定のリテラルを節の対から消去し、節構造を簡略化するために行われるが、古い節がそのまま残ったのでは節全体から変数を消去したことにならない。

逆に、導出の都度必ず古い節が消去されるならば演算時間は大幅に短縮される可能性がある。この論文で、それが満足されるための条件は、ルールの中に単節が存在する場合であり、それによって NP-Complete でないアルゴリズムが構築可能であることを示す。

INFERENCE ALGORITHM BASED ON SEQUENTIAL UNIT RESOLUTIONS

Takashi Obata
Defense Agency

The purpose of deduction is to simplify given set of clauses by eliminating specific pair of literals. However, it may not reduce the complexity as a whole unless old clauses are absorved by newly created one. This in turn implies that significant computation time reduction may be accomplished if every deduction process guarantees the elimination of old literals.

In this paper, it will be shown that such condition might be satisfied if at least one unit clause exists in a given set of rules. An efficient non NP-Complete deduction algorithm can be introduced using sequential unit deductions.

1　まえがき

命題論理を対象とした演繹推理問題は、一般に演算時間がデーターベースのサイズの指数乗に比例して増加する NP-Complete 問題として知られている。

Robinson の導出原理 [24]は、この問題に対する有力なアプローチであるが、この手法もまた最悪ケースでは NP-Complete であることが証明されている。[10]

その主な理由は、導出操作が必ずしも節構造を簡略化することにつながらず、多くの場合は、むしろ全体の節の数を増大させる結果となっているからである。

本来、導出は特定のリテラルを節の対から消去し、順次リテラルの数を減少させるために行われる。従って、次の段階で古い対がそのまま残ったのでは節全体からは変数を消去したことにならない。

だが、古い節が削除されるためには、それが単節であるか、あるいは導出によって得られた新しい節に吸収されなければならない。そうでない限り、導出操作は単にルール全体に新しい節を付加しただけの結果に終わり、これが演算時間増加の主因となっている。

従って、節数を増加させず、しかもその都度、必ずリテラルの数を減少させるような導出条件が見つかれば、演算時間が大幅に短縮される可能性がある。この論文で、その可能性があるのは、ルール中に単節が存在する場合であることを証明し、それによって NP-Complete でない推論アルゴリズムが構築可能であることを示す。

2　論理プログラムに使用される言語の定義

2・1　ルール形式

許容されるルール形式は if.. then 形式の命題論理表現のみとする。ただし、ホーン節には限定せず、英語として理解できる文章であれば、ルール中に then, or, and, not をどのような順序で使用してもよい。例えば次のルールは許容される表現形式である。なお if はオプションであり、省略しても意味は変わらない。

```
if a and not b then c or not d
wine or wiskey then contain alcohol
lion or tiger then wild animal and carnivorous
not rain then I will go
man and woman
```

2・2　リテラル

a, not b など、概念や事象を表す記号をリテラルと呼ぶ。リテラルは単語や文章であってもかまわない。not b においては not を符号、b を論理値という。アルゴリズムでは、if、then, or, and, not 以外の文字はすべてリテラルとして取り扱われる。例えば

if red lamp flickers then engine fire or computer malfunction

においては red lamp flickers, engine fire および computer malfunction はリテラルである。

2・3　節

リテラルが or だけで接続される形式のルールを節とよぶ。例えば not a or b, p or not q などは節形式である。リテラル１個からなる節を単節、２個以上のリテラルを含む節を複節、リテラルを含んでいない節を空節という。

2・4　ルールの節形式への変換

ルール形式の方が知識表現に適しているため、プログラムは通常ルール形式で表現されるが、演繹演算を行うためには節形式に変換する必要がある。

しかし、次の変換公式に示すように、任意のルールは、適切な変換操作により節形式に変換可能である。中には自明のものもあるが、アルゴリズム説明の便宜上記載した。

{原式}		{節形式}
a and b	⇔	a, b （２つの独立式）
a then b	⇔	not a or b
not (a and b)	⇔	not a or not b
not (a or b)	⇔	not a and not b
		not a; not b
not (not a)	⇔	a
a and (b or c)	⇔	a, b or c
	⇔	(a and b) or (a and c)
a or (b and c)	⇔	(a or b) and (a or c)
		a or b, a or c
a or (b or c)	⇔	a or b or c

a and (b and c) ⇔ a and b and c
a, b, c

次に、複節 A, B, C, D は次式で表されるものとする。

A ⇔ a1 or a2 or a3 ... or an
B ⇔ b1 or b2 or b3 ... or bm
C ⇔ c1 and c2 and c3 ... and ci
D ⇔ d1 and d2 and d3 ... and dj

これから次式が導かれる。

A then s
⇔ a1 or a2 or a3 ... or an then s
⇔ not (a1 or a2 or a3 ... or an) or s
⇔ not a1 or s,
not a2 or s,
.....
not an or s

t then B
⇔ t then (b1 or b2 or b3 ... or bm)
⇔ not t or b1 or b2 or b3 ... or bm

C then u
⇔ not(c1 and c2 and c3 ... and ci) or u
⇔ not c1 or not c2 or ...not ci or u

v then D
⇔ not v or (d1 and d2 and d3 .....and dj)
⇔ not v or d1,
not v or d2,
.....
not v or dj

さらに複雑な式は、これらの式から誘導することができる。例えば

(a or b) and c then d
⇔ not(a or b) or not c or d
⇔ (not a and not b) or not c or d
⇔ not a or not c or d,
not b or not c or d

となる。このように、式の変形によって任意のルールは節形式に変換できるので、以降は節形式だけを対象として議論を進める。なお、これらの変換式は、リテラルの符号の正負に関係なく、例えば ai の代わりに not ai, bj の代わりに not bj 等と置き換えても成立する。

3 逐次単節導出法

3・1 単節導出の考え方

複数個の節からなるルール集合を論理方程式とみなし、それを解く操作が演繹推論である。論理方程式から解を得るためには、まず、与えられた複数個のルールを節形式に変換し、双形リテラル中に含まれる正、補リテラルを導出によって順次消去する必要がある。リテラルを導出操作で逐次消去する試みは最初に J.A Robinson によってなされた。[24]

異なる二つの節に、正・補のリテラルがただ一組含まれているとき、この二つの節は導出可能であるという。例えば

not a or b
not b or c

という二つの節において、b と not b は正と補のリテラルであるから導出による消去が可能であって

not a or c

という新しい節が得られる。親が子を生むことに例え、最初の二つの節を親節、新しく得られた節を子節という。導出は、本来ルール数や変数の数を減少するために行われるが、これによって必ずしも節の数やリテラルの数が減少するとは限らない。つまり、新しい子節が得られても、それによって無条件に親節が削除できるわけではない。何故ならば、親節が削除できるのは、それが単節か、あるいは子節が親節の部分集合（親節が子節に吸収される）の場合に限られるからである。

上の例ではこの条件は満足されず、次の導出のステップではルールは３つに増える。導出のペアは、常に残っている全節を対象にして探索されるから、このような場合には、導出によってかえって「探索空間」が増加する結果となる。親節が削除されなければリテラルの数も減少しない。上の場合、子節ではリテラル b

は消去されているが、親節の中に依然として残っているからである。

従って、このような導出方法では探索空間が増大し、リテラルの数も減少せず、簡単化のための導出操作が逆に問題を複雑化させる結果となっている。この困難を避ける導出方法を探すために、一般化した導出を考える。

節 C1、C2 は同じ論理値で異なる符号のリテラルを含まない節であるとするとき、C1 + p, C2 + not p から節

$$C = C1 \cup C2$$

を導く操作が導出原理である。ここで、C は C1 と C2 の導出節を表わし、p が導出変数である。[24]

この目的は、リテラル p, not p を導出によって消去し、節構造を簡単化させることにある。従って、次の導出のステップに親節が残らないためには、親節が単節であるか、あるいは子節が親節を吸収しなければならない。このためには次式が成立する必要がある。

$$C1 \cup C2 \in C1 + p$$

および

$$C1 \cup C2 \in C2 + \text{not } p$$

この条件は次の場合に満足される。

（1） C1, C2 が等しい場合

C1、C2 がともに C に等しければ、子節 C1 ∪ C2 は単に C となり、C1 + p、C2 + not p の両方を吸収するので

p or C
not p or C

から導出により C が得られる。従って以降の計算で残るのは C のみとなる。もっとも、このような条件は、任意の節の組み合わせに対して成立するわけではない。

（2） p, not p が単節の場合

例えば p が単節（C1 が空節）の場合

p
not p or C2

から C2 が得られる。これは親節 C2 + not p を吸収する。一方、単節はそれ以上簡略化できないので、以降の親節の演算対象からは削除できる。

これ以外の場合には親節は消去されない。従って、双形リテラルにおいて正、補の対を探索する場合には多くの組み合わせが考えられるが、導出によって親節が削除され、ルール構造が簡略化される可能性のあるのは以上の２ケースだけである。

導出に単節の存在を前提とするため、単節導出はあまり一般性がないように見えるが、そうとは限らない。

単節がないと演繹が行われないアルゴリズムはむしろ大きな利点になり得る。何故ならば、単節と複節は意味が異なるからである。複節は「・・であれば・・である」という知識形式を表す。一方、単節は「・・である」という情報（物理的事実、あるいは論理的結論）を表す。

演繹推理は「ルールに情報がインプットされ、解が得られる」過程であると解釈すれば、この形式がかなり一般性を持つものであることがわかる。

従って、対話形式で単節がインプットされたときに、それを初期値として推論を行なうアルゴリズムが構築可能となれば大きな意味がある。これは、アウトプットが単節の場合にはインプットにフィードバックされ、次々と連続的に単節導出を行う機能を持つ。

単節導出はルール数が増加せず、また、導出操作の都度必ずリテラルの数が減少することが保証されるため、演算時間は大幅に縮小されることが予想される。次に、この考えに基づく逐次単節導出アルゴリズムを示す。

３・２ 単節導出による逐次消去アルゴリズム

STEP 1 与えられた節の、リテラルの符号を横の係数を要素とするマトリックスを構成する。（not の符号に対しては -1, そうでなければ 1 なる係数を対応させる。）マトリックスの上部にはリテラルの論理値を書き入れる。

STEP 2　マトリックスの項の横行のうち、1 または -1 を持つ項がただ一つあり、それ以外の項がすべて 0 であるとき、1 または -1 を持つ項の変数は単節である。マトリックスにこのような単節がある場合、その項の縦列で逆の符号を持つ数字を消去する。これは単節導出に対応する。

また、その項の縦列で、同じ符号を持つ数字の行は以後の演算対象から外す。これは節の吸収に対応する。この操作を、各々の単節について縦の列に 0 以外の数字がなくなるまで続行する。操作によって新たに単節が得られた場合にはそれに順位をつけ、STEP 2 を繰り返す。複数個の単節の順位の付け方は任意である。

STEP 3　すべての単節について STEP 2 を繰り返し、新たな単節がなくなったら操作は終了する。この段階でマトリックス上で単節として残っている項が方程式の「解」を表す。

例として次のような演繹問題を考える。

not p or q
p or r or q
s or t
r or x
not r or not s
not q

上記のルールの制約下で t の真偽を決定せよ。

STEP 1,2　次のマトリックスには not q なる単節があるので、まず not q を導出変数とし、q の項の縦列で 1 の数字があればこれを消去する。

p	q	r	s	t	x	
-1	1	0	0	0	0	
1	1	1	0	0	0	
0	0	0	1	1	0	
0	0	1	0	0	1	
0	0	-1	-1	0	0	
0	-1	0	0	0	0	← 単節

これにより、マトリックスは次のようになる。

p	q	r	s	t	x	
-1	0	0	0	0	0	← 単節
1	0	1	0	0	0	
0	0	0	1	1	0	
0	0	1	0	0	1	
0	0	-1	-1	0	0	
0	-1	0	0	0	0	

新しく得られた単節は not p であるから、これが次の導出変数となり、下のマトリックスが得られる。

p	q	r	s	t	x	
-1	0	0	0	0	0	
0	0	1	0	0	0	← 単節
0	0	0	1	1	0	
0	0	1	0	0	1	吸収により削除
0	0	-1	-1	0	0	
0	-1	0	0	0	0	

新たに得られた単節は r である。

p	q	r	s	t	x	
-1	0	0	0	0	0	
0	0	1	0	0	0	
0	0	0	1	1	0	
0	0	0	-1	0	0	← 単節
0	-1	0	0	0	0	

上のマトリックスで、新たに得られた単節は not s である。これから

p	q	r	s	t	x
-1	0	0	0	0	0
0	0	1	0	0	0
0	0	0	0	1	0
0	0	0	-1	0	0
0	-1	0	0	0	0

STEP 3　この段階で新たな単節がなくなったので、手順は終了する。最後に t の項が残るが、これは求める

解と一致する。この問題は not p, not q, r, not s, t なる解を持っている。x については不定である。

リテラルの総数を n とするとき、最初の導出に該当する変数を探すに要する時間は、高々 n-1 のオーダーである。２つ目の導出変数を探すに要する時間は n-2 のオーダーである。従って、全ての導出を行うのに必要な探索時間は (n-1)+(n-2)+....+1 = n(n-1)/2 、つまり n の２乗のオーダーを超えない。従って、このようなアルゴリズムは NP-Complete ではない。

３・３　単節導出の Consistency

アルゴリズムが consistent であるということは、ルールに矛盾が含まれていないとき、常に論理的に正しい解が得られるという事である。導出操作が consistent な解を導く事は既に証明されている。[24]

単節導出は、導出の特殊な場合であるから、解の consistency は保証されている。

３・４　単節導出の Completeness

アルゴリズムが complete であるということは、解を有しているとき、必ずその解が得られるという事である。この意味では、単節導出は、単節が存在しない場合には演繹が行われないから complete ではない。既に見てきたように単節が存在しなくとも解が得られる導出は有り得る。例えば、次のような問題を考える。

```
    a or c
not a or c
not c or p
```

マトリックス形式では

a	c	p
1	1	0
-1	1	0
0	-1	1

この問題は c, p なる解を持っているが、単節導出ではこのような解を求めることはできない。しかし一方、このような導出の全てを求めようとすれば、アルゴリズムは NP-Complete となることが既に証明されている。[10]

従って、この意味では単節導出は complete でないという代償のもとに高速性を得ているアルゴリズムである。ただ、既に述べたように、単節導出以外にリテラルの数が減少し、単節解が得られる可能性があるのは

```
    a or c
not a or c
```

のような形に限定されている。これは、「a であるなしにかかわらず c である」という意味である。このような節形式が導出の過程であらわれることを避けるようにプログラムを組めば、残りは全て単節導出で処理出来る事になり、さほど実用上の不便はない。

４　逐次単節導出アルゴリズムの例題と応用

演繹推論は、ルールに隠されているが、あからさまには記述されていない結論を「陽」に導き出すことができる。次の例題によりこれを見てみよう。

（１）　一般に鳥は飛ぶが、例外がある。それはペンギン、駄鳥、ロードランナーである。カナリヤは鳥であるが、これら３種の鳥とは異なっている。xは鳥であり、カナリアである。xは飛ぶか？
という論理学上たびたび引き合いに出される非単調推論問題を解いてみよう。ルールを次に示す。

- bird and not penguin and not ostrich and not roadrunner then flyable
- canary then bird and not penguin and not ostrich and not roadrunner
- penguin or ostrich or roadrunner then not flyable and bird
- X then tweety
- tweety then canary

ルールには単節がないために、アルゴリズムは解をアウトプットしない。しかし、ここで 「X」 をインプットすれば次のような演繹解が得られる。

```
X                ← input
tweety
canary
```

```
bird
not penguin
not ostrich
not roadrunner
flyable
```

これは、ルールに含まれている情報をもとに、論理的に X が満足すべき条件を全てアウトプットしている。次に「鳥でない」ものは何かを知るために、質問をクリアして、not birdとインプットすると

```
not bird            ← input
not canary
not penguin
not ostrich
not roadrunner
not tweety
not X
```

これもまた、論理的に鳥でない条件を満足するアウトプットとなっている。最後に、これらのルールが意味するペンギンとは何かを知るために、再度質問をクリアして、「ペンギン」というインプットを行うと

```
penguin             ← input
not canary
not flyable
bird
not tweety
not X
```

なる解が得られる。これらも論理的に妥当な解である。ペンギンは駄鳥やロードランナーとは異なるが、ここではそのようなアウトプットは得られていない。何故ならば、与えられたルールにはペンギンが駄鳥やロードランナーではないという条件は含まれていないからである。演繹アルゴリズムでは、ルールに示されている内容だけが知識の全てであって、それ以外の内容は推論の過程に入り込まない。

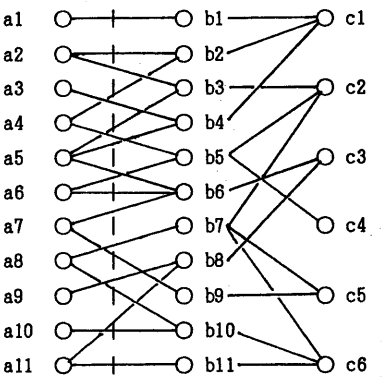
このような推論アルゴリズムの特性は、データ検索に応用できる。人間がデーターベースを利用するのは、そこから、意図する結論が引き出せるかどうかを知りたいという場合が多い。

これに演繹推論を適用できれば、上の例のように not bird, penguin のようなキーワードのインプットによって膨大なデーターベースにアクセスし、機械的に読みこんで推論を行い、短時間に結論を導き出すことが出来るだろう。即ち、ユーザーがデーターベースを読まなくとも、単にキーワードのインプットだけで、求める結論が得られることになり、大幅な省力化が実現できる可能性がある。

（２）演繹推論アルゴリズムの有力な応用分野に、故障原因探求や病気診断システムがある。一般に、複雑なメカニカル・システムや人体に於ては、ある部品が故障すればどのような現象が起こるか、あるいはある病気の場合にどのような症状が発生するかが分かっていても、逆にある現象が発生した場合に何が故障の原因か、あるいはある症状が現われた場合にどのような病気かを知ることは簡単ではない。これは、同一の現象や症状を起こす原因が複数個あり、その原因が常に一つとは限らないからである。

例として、次のような故障原因探求問題を考える。次の図のようなネットワークにおいて、部品レベル２の部品が部品レベル１の部品に連結され、さらに部品レベル１は警報灯につながっている。

ここで、例えば c1 が故障すればそれに接続されている b1, b2, b4 が不作動状態になる。これにより a1, a2, a3, a4, a5 の警報灯が点灯する。では、a2, a4, a5, a6, a8 の警報灯が点灯している場合には、どの部品が故障、あるいは不作動状態にあるか？



警報灯　　部品レベル１　　部品レベル２

警報灯 ai が点灯している際には単に ai，点灯していない場合には not ai と表わす。また、bj，ck が不作動状態にある場合には単に bj，ck と表わし、作動

状態にある場合には not bj, not ck と表わすこととすると次の if ...then rule が成立する。

```
not a1 and a2 and not a3 and a4 and a5 and a6
not a7 and a8 and not a9 and not a10 and not a11
c1 or c2 or c3 or c4 or c5 or c6
a1 then b1
a2 then b2 or b3
a3 then b4
a4 then b2 or b5
a5 then b3 or b4 or b6
a6 then b5 or b6
a7 then b6 or b9
a8 then b7 or b10
a9 then b8
a10 then b10
a11 then b8 or b11
b1 then a1
b2 then a2 and a4
b3 then a2 and a5
b4 then a3 and a5
b5 then a4 and a6
b6 then a5 and a6 and a7
b7 then a8
b8 then a9 and a11
b9 then a7
b10 then a8 and a10
b11 then a11
b1 then c1
b2 then c1
b3 then c2
b4 then c1
b5 then c2 or c4
b6 then c3
b7 then c2 or c5 or c6
b8 then c3
b9 then c5
b10 then c6
b11 then c6
c1 then b1 and b2 and b4
c2 then b3 and b7
c3 then b6 and b8
c4 then b5
c5 then b7 and b9
c6 then b7 and b10 and b11
```

上記ルールで

a1 then b1 と
b1 then a1

は異なる条件である。前者は、a1 が不作動なのは b1 が不作動だからだ、という意味であり、後者は b1 が不作動であれば a1 も不作動となる、ということを意味している。最初の二つのルールに単節が含まれているので、このプログラムを走らせると次のような解が得られる。

```
not a1 /a2 /not a3 /a4 /a5 /a6 /not a7 /a8
/not a9 /not a10 /not a11 /not b1 /not b4
/not b6 /not b9 /not b8 /not b10 /not b8
/not b11 /not c1 /not c1 /b3 /b5 /not c3
/not c5 /b7 /not c6 /not c6 /not b2 /c2 /c2
```

上記で not の符号のないのが不作動状態にある部品である。多少の重複は見られるが、極めて効率的に解が得られている。なお、部品レベル２の部品で故障しているのは c2 である。

この例題のルール数は４２個あり、80286 CPU 使用の J－3100 GX で quick basic 作成のプログラムを走らせた際、演算所要時間は 1.25 秒であった。

（３）単節がインプットされれば、それに対応した演繹解がアウトプットされるという特性を生かすことにより、単節導出アルゴリズムを高性能のエキスパートシステムとして使用できる。then の両側に not, and or が無制約で使用できるため、知識表現能力はプロダクション・ルールに比較して大幅に向上する。

一例として、医者が患者に投与する内服薬の量を決定するプログラムを構築する。原則として一日に投与する錠剤の数は１５才以上の男子は５錠、女性は４錠である。８才以上１５才未満では男女ともに３錠、８才未満は２錠である。しかし、妊娠している女性、心臓の悪い患者には投与してはいけない。他の薬を服用している場合、それが風邪薬、化膿止めであればこの薬を投与してはならない。それ以外であれば問題はない。ルールは次のようになる。

```
Do you have heart problem? [y0/n0]
```

n0 then Are you taking another pills? [y1/n1]
y1 then Is it pills for cold? [y2/n2]
n2 then Is it pills for infection? [y3/n3]
y0 or y2 or y3 or y6 then I will prepare another pills.
n1 or n3 then Are you a male? [y4/n4]
y4 then Are you older than 15? [y5/n5]
y5 then Take 5 pills a day.
n4 then Are you pregnant? [y6/n6]
n6 then Are you older than 15? [y7/n7]
y7 then Take 4 pills a day.
n5 or n7 then Are you older than 8? [y8/n8]
y8 then Take 3 pills a day.
n8 then Take 2 pills a day.

如何なるコンピューターアルゴリズムも、中に含まれる情報量より短いプログラムを組むことは出来ない。ここでは、コンピューターだけに必要な人工的な記号が一切使用されていないので極限まで短いプログラムとなっている。run させると、最初の一行はそのままアウトプットされるが、それ以外はすべて条件文であるので、何らかのインプットが行われるまではアウトプットされない。まず

Do you have heart problem? [y0/n0]

というメッセージのみが現れる。

n0

とインプットすれば

Are you taking another pills? [y1/n1]

という表示が現れる。以下順次質問に応えて行くと

n1 /Are you a male? [y4/n4]
n4 /Are you pregnant? [y6/n6]
n6 /Are you older than 15? [y7/n7]
n7 /Are you older than 8? [y8/n8]
y8 /Take 3 pills a day.

このように、対話形式で質問に応えてゆけば求める答が得られる。

参考文献:

(1) Allen, J. F., "Toward a General Theory of Action and Time," Artificial Intelligence, 23(2) : 123-154, 1984.

(2) Barr, A. and Feigenbaum, E.(eds.), The Handbook of Artificial Intelligence, Vol. I and II. Reading, MA: Addison-Wesley, 1981 and 1982.

(3) Boyer, R.,S., Locking: A Restriction of Resolution, Austin, TX: University of Texas at Austin, 1971 (Ph.D. dissertation).

(4) Buchanan, B.G. and Shortliffe, E. H., Rule based Expert Sustems: The MYCIN Experiments of the Stanford Heuristic Programming Project.

(5) Davis, M., and H. Putman, A Computing Procedure for Quantification Theory, Journal of the ACM 7 (1960) 201-215

(6) Gallaire, H., and J. Minker, Logic and Data Bases, Plenum Press, New York (1978)

(7) Gallaire, H., J. Minker, and J.-M. Nicolas, Logic and Databases: A Deductive Approach, Computing Surveys 16 (1984) 153-185

(8) Garey, M.R., and D.S. Johnson, Computers and Intracterbility: A guide to the Theory of NP-Completeness, W.H. Freeman, San Francisco (1979)

(9) Genesereth, Michael R. & Nilsson Nils J., "Logical Foundations ofArtificial Intelligence", Morgan Kaufmann Publishers, Inc., 1987

(10) Haken, A., The Intracterbility of Resolution, Theoretical Computer Science 39 (1985) 297-308

(11) Hooker, J. N. "A Quantative Approach to Logical Inference", Decision support system 4 (1988), 45-69

(12) Israel, D., "The Role of Logic in knowledge Presentation," IEEE Computation, 16(10): 37-42, 1983.

(13) 伊藤史朗、石塚満、数理計画法の適用による仮説推論システムの高速化 情報処理学会 第３８回（昭和６４年前期）全国大会論文集2F-4（1989）422-423

(14) 伊藤史朗、石塚満、シンプレックス法に基づく仮説推論システム 情報処理学会、第３９回（平成元年後期） 全国大会論文集5C-8,（1989）333-334

(15) Levesque, H.," Knowledge Representation and Reasoning," Annual Review of Computer Science,1: 255-288, 1986.

(16) Loveland, D.W., Automated Theorem Proving: A Logical Basis, North-Holland (1978)

(17) Moore,R.C., "Reasoning About Knowledge and Action," Technical Note 191. Menlo Park, CA:SRI International, Artificial Intelligence Center, 1979

(18) Moore,R.C., "The Role of Logic in Artificial Intelligence", in Benson,I.,(ed.), Intelligent Machinery: Theory and Practice. Cambridge, UK: Cambridge University Press, 1986.

(19) McMullen, C., and J. Shearer, Prime Implicants, Minimum Covers and the Complexity of Logic Simplification, IEEE Transaction of Computers C-35(1986) 761-762

(20) Pospesel, H., Introduction to Logic: Predicate Logic. Englewood Cliffs, NJ: Prentice-Hall, 1976

(21) Quine, W.V. The problem of Simplifying Truth Functions, Americans Mathematical Monthly 59, (1952) 521-523

(22) Quine, W.V. A Way to Simplyfy Truth Functions Americans Mathematical Monthly 62 (1955) 627-631

(23) Quine, V.W. Two Dogmas of Empiricism, in: From a Logical Point of View, Harvard University Press (1961)

(24) Robinson, J.A.,"A Machine-Oriented Logic Based on the Resolution Principle,", Journal of the Association for Computing Machinery, 12(1): 23-41, 1965

(25) 庄司功、伊藤秀則 ロジック・データベースの Consistency チェックに関する一方法 情報処理学会 第３６回（昭和６３年前期）全国大会論文集7E-7 (1988)481-482

(26) Van Benthem, J., The logic of Time. Hingham, MA: Kluwer Academic Publishers, 1983