知識工学と人工知能 46－9
（1986. 5. 9）

プログラム変換による定性的推論の効率化

川本 啓之　中川裕志
横浜国立大学 工学部 電子情報工学科

常識的知識の表現法として定性的推論というものがある。これは人間が物理の問題を考えるときに行う定量的な正確さに重きをおかず、状態の変化に注目するといった方法を定式化したものである。ところがこの推論法は極めて小さい系でしか実現されていない。これは対象となる系が単純なものであってもそれのとりうる状態はたくさんあり、全体としての推論ができないからである。また定性的推論は変化率の増加、減少といったことを中心とした計算が多くそうした計算技術があまり確立されていないことが原因である。

そこで、本稿ではPROLOG言語上で定性的方程式をプログラムの形で記述し、プログラム変換の技法により部分計算することで、効率化を計ることを考えた。また、その際の変換アルゴリズムを見いだし変換の自動化を達成した。

WG AI 46-9
Efficient qualitative reasoning by program transformation (in Japanese)

Hiroyuki KAWAMOTO and Hiroshi NAKAGAWA
Department of Electrical and Computer Engineering, Yokohama National University,
Yokohama, 240, Japan

Qualitative reasoning is a hopeful common sence method. But it wastes too much space and time in execution.
In this paper, we describe a qualitative reasoning in Prolog. We apply a program transformation especially a partial evaluation for a qualitative reasoning system in Prolog in order for efficiency. By this transformation, small system like a LC oscilater can be qualitatively reasoned about 10 to 50 times faster than the original qualitative reasoning programs.

1. はじめに

人工知能において知識の表現法を確立することは中心問題であり、現在さまざまな研究が行われている。ひとつはエキスパートシステムに代表される様に、知識を表現するのに適したプログラミング言語を作り知識表現をはかるといった方向である。

しかし、こうした形式で表現出来ない部分もある。その一つが定性的推論のもとになる人間のメンタルモデルの概念を利用した知識表現技術である。定性的推論は人間が行っているような問題解決の方法を抽出しモデル化することによりエキスパートシステムに常識を持たせたり、物理現象を定性的に説明するのに役立つ。定性的推論はこのような問題に対する新しいアプローチである。人間が物理の問題を考えるときに行う推論は定量的な正確さに重きをおかず状態の変化に注目するという性質を定式化したものである。今までの定性的推論の研究として De Kleer等の constraint oriented な方法[1]、Ken Forbus等の process oriented な方法[2]、Kuipersの質的シュミレーション[3]などが知られている。ところがこうした方法は極めて小さな系でしか実現されていない。これは対象となる系が単純なものであってもそれのとりうる状態はたくさんあり、全体としての推論が出来ないからである。また定性的推論は変化率の増加、減少といったことを中心とした計算が多くそうした計算技術があまり確立されていないことが原因である。現在、この問題に対して De Kleer の “Choice without Backtracking”[4]などが定性的シュミレーションシステムを稼働するための基礎技術としてあるが、構造が複雑で多大なメモリを消費してしまうという欠点がある。

そこで、本稿では、PROLOG言語上で定性的方程式をプログラムの形で記述し、プログラム変換の技法により部分計算することで、効率化を計ることを考える。

2. PROLOG での定性的推論

この章では定性的推論を PROLOG 言語上で行う方法を説明してゆく。De Kleerはpropagation という方法を用いて定性的な平衡方程式を解析している。これは決定した変数の値を次々と伝ぱんさせることで全ての解を求めようという方法である。これは１種の前向き推論であると言えるであろう。PROLOG 言語は本来後向き推論システムであるが、バックトラック機構を用いれば propagetion と同等な事ができる。

また、定性的推論で時間推移による状態変化のシュミレーションをおこなう際系の次の状態を予測し、その挙動を “視る”ことができるという意味で、エンビジョニング(envisioning)という概念を導入している。これは変数の微分値により次の状態を予測する述語を導入することで PROLOG 上に実現している。それらを具体的な例を用いて説明してゆく

EXAMPLE 1

PROLOG言語で物理モデルを表現するにはまず系の拘束条件を表している定性的方程式をプログラムの形にしなければいけない。例として次の様な系を考える。

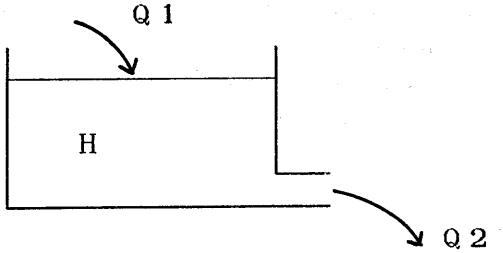

図１　穴の開いたタンク

上の図の示す系は底面積がＦであるタンクに単位時間にＱ1の水が流入し、下の穴からはＱ2の水が流失している。タンクの水位をＨとし、Ｑ2は水位Ｈに比例しその比例定数を

k（＞０）とする。この系を通常の方程式で表すと

$$Q2 = k * H \qquad (2.1)$$

$$dH/dt = (Q1-Q2)/F \qquad (2.2)$$

となる。ここで、(2.1)式の定数kは正であるから、Q2の定性値は H のみによって決まる。よってこれに対応する定性的方程式は(2.3)式のようになる。同様に(2.2)において定数Fは正であるので、dH/dtの定性値とQ1-Q2の定性値は一致する。dH/dtに対応する定性的変数を　Hで表すと、(2.2)式は(2.4)式の様になる。

$$Q2 = H \qquad (2.3)$$

$$H = Q1-Q2 \qquad (2.4)$$

これをＰＲＯＬＯＧ上で表現するには定性的方程式中の＝に対応する＝,加算減算に対応する QADD,時間推移の推論をする INT という三つの述語を用いればよい。まず、＝ の定義は ((= * *)) のようになっている。 定性的加算は QADD という述語を用いる。QADD は３引数の述語で３つの引数の関係は後述する表１のようになっている。減算については普通の代数系と同じに扱うことができる。たとえば　Ａ－Ｂ＝Ｃ　は加算を用いて　Ｃ＋Ｂ＝Ａ　の様に表されるので (QADD *C *B *A) とすればよい。次に時間による状態変化も含めて推論するには現在の変数の定性値とその変数の微分の定性値より次の変数の定性値を求める操作が必要である。そのため INT という述語を用いる。INT は表２の関係を表現したものである。第１引数は現在の変数、第２引数はその微分値、第３引数はそれらによってもとまった次の時刻の変数である。これらの他に、＋ → － や － → ＋ などの不連続な変化を排除する述語 CONT を用い (CONT *LDH *DH) というゴールを入れて連続性をチェックしてやると、(2.3),(2.4)式は

```
((TANK *LDH *LH *Q1 *DH *H *Q2)(INT *LH *LDH *H)
                               (= *Q2 *H)
                               (QADD *DH *Q2 *Q1)
                               (CONT *LDH *DH))
```

の様に表される。

3. 各述語の定義

さて、今まではっきり示していなかった ＝,QADD,INT,CONT の定義を与える。
＝ は PROLOG 言語で定義されているとおり ((= * *)) を用いている。QADD,INT は表１、表２のとおりである。

y＼x	－	0	＋
－	－	－	?
0	－	0	＋
＋	?	＋	＋

x＼dx	－	0	＋
－	－	－	-,0
0	－	0	＋
＋	+,0	＋	＋

？は +,0,- のいずれも取りうることを示す

表１ QADD の定義　　表２ INT の定義

これらの定義をする際 QADD であれば ((QADD + + +))等の様にするのが普通であろう。しかし、ここで SIGN,SIGN1,EX という述語を導入することでこれらの述語の定義をコンパクトにできる。まず QADD の定義を (3.1) に示す。

```
((QADD *A *A *A)(SIGN *A))
((QADD *A *B *C)(EX *A *B)(SIGN *C))
((QADD  0 *A *A)(SIGN1 *A))
((QADD *A  0 *A)(SIGN1 *A))
```

式 (3.1)　コンパクトにした QADD の定義

SIGN の定義	SIGN1 の定義	EX の定義
((SIGN +))	((SIGN1 +))	((EX + -))
((SIGN 0))	((SIGN1 -))	((EX - +))
((SIGN -))		

同様に INT , CONT も定義すると式 (3.2)(3.3)の様になる。

```
((INT *A *A *A)(SIGN *A))
((INT *A  0 *A)(SIGN1 *A))
((INT  0 *A *A)(SIGN1 *A))
((INT *A *B *A)(EX *A *B))
((INT *A *B  0)(EX *A *B))
```

式 (3.2) INT の定義

```
((CONT *A *A)(SIGN *A))
((CONT *A  0)(SIGN1 *A))
((CONT  0 *A)(SIGN1 *A))
```

式 (3.3) CONT の定義

ここで注目されたいのは、 HEAD の部分に＋，－といった実際の値が現れていないことである。これは SIGN,SIGN1,EX の述語を導入することで定義の抽象化が行われたと解釈できる。もし、EXAMPLE 1 にでてきたようなプログラムを評価するだけであったなら PROLOG 言語の性質上コンパクトにしないもとの形のほうが効率が良い。このような形で QADD や INT を定義したのはこれらの述語をできるだけ抽象化することでプログラム上で字面として現れる性質をプログラム変換の力をかりて抽出し、goal どうしのインターラクションで goal 探索中発生する不要な枝を刈り取り述語の効率化を図るためである。次章ではこの考えを実現する方法について述べる。

4. プログラム変換による定性的方程式の効率化

定性的な平衡方程式はその変数間の関係により拘束条件をかたちづくっているのである。PROLOG 上での実行を考えると、その条件に束縛されながらバックトラックを繰り返して系の満たす解を求めている。ところが、拘束条件が厳しく、方程式があまり多くの解を持たない場合はほとんどの選択枝は失敗するわけである。これが系の推論に時間がかかる原因である。そこで、実行させたとき探索中失敗する枝を全て刈り取ってしまうことを考える。その際、定性的方程式が表す実際の物理構造には一切言及しないでこれを変数間に働く単なる束縛条件としてみる立場をとる。

簡単な具体例でその様子をみてみよう。仮に下に示す様な定性的方程式が成り立っていたとする。

$$A + B = C$$
$$A = C$$

これを PROLOG で表現すると

```
((SAMPLE *A *B *C)(QADD *A *B *C)(= *A *C))
```

のようになる。これを変換してゆく。

この変換で用いる最も一般的な技法は unfolding である。まず第２ゴールをunfoldすると

```
((SAMPLE *A *B *A)(QADD *A *B *A))
```

(4.1)

の様になり＝の関係が変数名の付けかえにより吸収されているのがわかる。これは HEAD の形が変化し、特定化されてバインド可能な CLAUSE を制限することにより余分な探索枝を取り去ったと解釈することができる。

つぎに、式(4.1) の第２GOAL の QADD を unfoldすると

```
((SAMPLE *A *A *A)(SIGN *A))
((SAMPLE *A *B *C)(EX *A *B)(SIGN *A))
((SAMPLE  0  0  0)(SIGN1  0))
((SAMPLE *A  0 *A)(SIGN1 *A))
```

のように展開される。ただし QADD の定義は式（3.1）のものであり、SIGN,SIGN1,EX という述語は前節に用いた物と同じである。ここでSIGN1の定義を思い出して頂きたい。第３CLAUSE をみると（SIGN1 0）の様な定義は存在しないことに気がつくだろう。このようなゴールは実行時には失敗するわけであるからこれを含む CLAUSE を取り除くべきである。これを行うには失敗するゴールを unfold してやればよい。その結果を下に示す。

```
((SAMPLE *A *A *A)(SIGN *A))
((SAMPLE *A *B *A)(EX *A *B)(SIGN *A))
((SAMPLE *A  0 *A)(SIGN1 *A)
```

次は第２CLAUSE に注目する。すると SIGN が冗長になっている。

```
((SIGN +))
((SIGN 0))
((SIGN -))
```

なぜなら（EX *A *B）の *A に返ってくる結果は全て（SIGN *A）の test において成功するからである。
そこで、第２ゴールを削除すると

```
((SAMPLE *A  0 *A)(SIGN1 *A))
((SAMPLE *A *A *A)(SIGN *A))   (4.2)
((SAMPLE *A *B *A)(EX *A *B))
```

```
((EX + -))
((EX - +))
```

の様になる。
これを行うには第２ CLAUSE に等価則を適用すればよい。今の場合は

```
((EX *A *B)) <--- ((EX *A *B)(SIGN *A))
```

という等価則を用いた。他にも等価則としてつぎのようなものがある。

```
((SIGN1 *A)) <--- ((SIGN1 *A)(SIGN *A))
((SIGN1 *A)) <--- ((SIGN *A)(SIGN1 *A))
((= *B *C)(EX *A *B)) <--- ((EX *A *B)(EX *A *C))
((= *B *C)(EX *A *B)) <--- ((EX *B *A)(EX *C *A))
((= *B *C)(EX *A *C)) <--- ((EX *B *A)(EX *C *A))
((= *B *C)(EX *A *B)) <--- ((EX *A *B)(EX *A *C))
((EX *A *B)) <--- ((EX *A *B)(SIGN *A))
((EX *A *B)) <--- ((EX *B *A)(SIGN *A))
((EX *A *B)) <--- ((EX *A *B)(SIGN *A))
((EX *A *B)) <--- ((EX *B *A)(SIGN *A))
```

そうして生成したCLAUSE 群の中にはまだCLAUSE どうし同値な物が存在することがある。そのようなCLAUSE は一つにまとめてしまう必要がある。それはメモリ効率の点でみてバックトラックにより冗長な解を発生する CLAUSE を保持していることは無駄だからである。また、この操作により変換で求められた CLAUSE は互いに独立であることが保証される。この場合 CLAUSE の独立性はモデルの中の冗長性を取り去った事を意味している。次は実際の物理系に即したモデルで変換による効果をみてみる。

EXAMPLE 2

EXAMPLE 1 でタンクに流れ込む液体の量と液面の高さ、流出量についてモデルを作りそれをPROLOGのプログラムの形で表した。さて、この述語を用いて最初タンクの中が空であるとして推論を進める述語を作る。次ぎに示す N2-TANK という述語は３状態後の状態を推論する様に作ったものである。

```
((N2-TANK *Q1 *NDH *NH *NQ2)(TANK 0 0 *Q1 *DH *H *Q2)
                            (TANK *DH *H *Q1 *NDH *NH *NQ2))
                            (TANK *NDH *NH *Q1 *N2DH *N2H *N2Q2))
```

最初のタンクの状態は液面の高さ、液面の高さの変化、共に０である。そして常に正の

量の液体が流入することが表現できるようになっている。そこで効率化をするために、この述語を変換の対象としてみよう。まず、述語 TANK を unfold してさらに INT,QADD,= も展開する、そしてゴールのマージ、等価則の適用をして等価な CLAUSE を一つにまとめると下の式の様になる。

```
((N2-TANK 0 0 0 0))
((N2-TANK *N2DH *N2DH *N2DH *N2DH)(SIGN1 *N2DH))
((N2-TANK *DH *N2DH *DH *DH)(EX *DH *N2DH))
((N2-TANK *NDH 0 *NDH *NDH)(SIGN1 *NDH))
```

さてこれがどのくらい効率化されているかみてみよう。
これには Q1 が＋のときの全ての解を生成するのに要する時間を計ることでおこなう。計測は富士通 FM-11 上で行った。これより示すデータは全て FM-11 の上で行ったものである。すると変換前では約１１秒かかったものが変換後では１．６秒程になり、約７倍ほど速くなるという結果が得られた。

EXAMPLE 3

つぎはタンクを２つつなげた系の３状態後の推論をする述語を対象に考えてみる。初期条件は全てのタンクの中身は空で、そこへ液体を上のタンクから注ぎ入れるものとする。まず、２つのタンクの状態をシュミレートする DTANK という述語を示す。

```
((DTANK *LDH1 *LH1 *LDH2 *LH2 *Q1 *DH1 *H1 *Q2 *DH2 *H2 *Q3)
 (TANK *LDH1 *LH1 *Q1 *DH1 *H1 *Q2)
 (TANK *LDH2 *LH2 *Q2 *DH2 *H2 *Q3))
```

この述語を用いて作った述語 N2-DTANK を次に示す。

```
((N2-DTANK *Q1 *N2DH1 *N2H1 *N2DH2 *N2H2)
 (DTANK 0 0 0 0 *Q1 *DH1 *H1 *Q2 *DH2 *H2 *Q3)
 (DTANK *DH1 *H1 *DH2 *H2 *Q1 *NDH1 *NH1 *NQ2 *NDH2 *NH2 *NQ3)
 (DTANK *NDH1 *NH1 *NDH2 *NH2 *Q1 *N2DH1 *N2H1 *N2DH2 *N2H2 *N2Q3))
```

これを変換の対象とすると下の様な結果が得られる。

```
((N2-DTANK 0 0 0 0 0))
((N2-DTANK *Q1 *Q1 *Q1 *Q1 *Q1)(SIGN1 *Q1))
((N2-DTANK *Q1 *Q1 *Q1 0 *Q1)(SIGN1 *Q1))
((N2-DTANK *Q1 0 *Q1 *Q1 *Q1)(SIGN1 *Q1))
((N2-DTANK *Q1 0 *Q1 0 *Q1)(SIGN1 *Q1))
((N2-DTANK *Q1 *N2DH1 *Q1 *Q1 *Q1)(EX *Q1 *N2DH1))
((N2-DTANK *Q1 *N2DH1 *Q1  0  *Q1)(EX *Q1 *N2DH1))
```

これにより変換前と変換後で実行時間を計って比べてみる。すると変換前約８０秒かかっていたものが変換後には２．６秒程に短くなった。これは３１倍ほど効率化されたことになる。このように実行にかなり時間のかかる系でもそれの費やすほとんどの時間は探索中失敗する枝を系が保持しているために起こっている。特に PROLOG 言語は depth first な探索であるので生成する解が少ないときは失敗する選択枝の方が圧倒的に多くこれが実行時間の増加を招いている。次はその典型的な例をおみせする。

EXAMPLE 4

図２ は De Kleer の論文で引用されている pressure regurater の例で流入する液体の圧力に関わらず流出量を一定に保つように動作する機械系である。

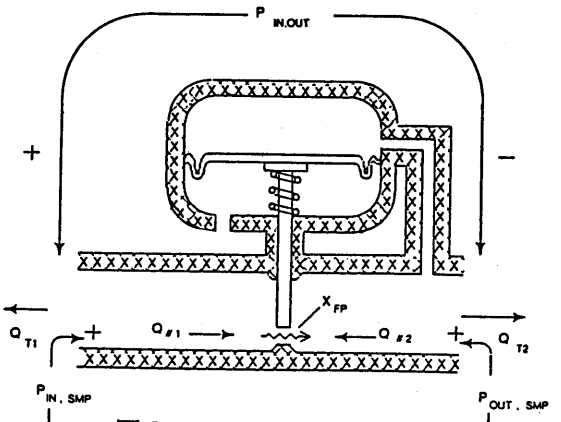

図2 pressure regurater

図を見てもらえばわかるようにこのモデルに対応する定性的方程式は次のようになる。

D•PIO + D•XF = D•Q1V　　D•XF + D•POS = 0
D•PIO + D•POS = D•PIS　　D•QT2 + D•Q2V= 0
D•QT1 +D•Q1V = 0　　D•Q1V + D•Q2V = 0
D•QT2 = D•POS

これを PROLOG 上で表現すると次の様になる。

```
((PRESS *DPIO *DXF *DPOS *DPIS *DQ1V *DQ2V *DQT1 *DQT2)
 (QADD *DPIO *DXF *DQ1V)
 (QADD *DXF *DPOS 0)
 (QADD *DPIO *DPOS *DPIS)
 (QADD *DQT2 *DQ2V 0)
 (QADD *DQT1 *DQ1V 0)
 (QADD *DQ1V *DQ2V 0)
 (= *DQT2 *DPOS))
```

式 (4.3) pressure regurater の変換前

これを今までの手法を用いて変換すると次に示すようになる。

```
((PRESS 0 0 0 0 0 0 0 0))
((PRESS *DPIO *DXF *DPIO *DPIO *DPIO *DXF *DXF *DPIO)
 (EX *DXF *DPIO))
```

式 (4.4) pressure regurater の変換後

式 (4.3) をみてみるとこの CLAUSE はたくさんの GOAL より成っているが、実行させると極、少数の解しか持たないことがわかる。これはまえにも述べたように選択枝の数が多く探索空間が広いにも関わらず制約条件が厳しく解を少ししか持たない場合である。全ての解をもとめる時間を比べてみると変換前には９０秒程かかっていたものが変換後は約２秒に短縮されている。これはほぼ４５倍効率化されたことになる。

5. 自動変換システムによる効率化戦略

今までの変換手順はコマンド列の並びと捉えることが出来る。そこであらかじめプログラムされたコマンド列を順に実行するインタープリタ“ＰＡＲＯＴＳ”上で今までの変換プロセスを記述することで自動変換を実現する。そのアルゴリズムは次の２段階に分けると考え易い。

まず第一段階は全ての GOAL を展開する事である。これを行うには unfold (U) で展開をしたあと auto-unfold (AE) というコマンドを施せばよい。AE というコマンドは一意展開を行う他、実行時失敗する GOAL を含む CLAUSE に対しては閉世界仮説によるdeletion

を行うので失敗する枝の枝刈りができる。これを繰り返し施し全ての GOAL が SIGN,SIGN1,EX という PRIMITIVE なものになるまで続ける。

第２段階はで冗長性を取り去る事を行う。ここでは等価則の適用(AE コマンド)を行い冗長な解を生成する GOAL 列を取り去り、CLAUSE どうしが等価なものについてもその様な CLAUSE をひとつにまとめることで(FC コマンド)冗長性を排除している。

以上のアルゴリズムにのっとって変換手順を記述したものを次ぎに示す。

```
((TRY (U)
      AU
      (TRY (U)
           AU
           (TRY (U) AU END)
             END)
       END)
 AM
(TRY (UG) (IF AE)(THEN AM AU END)
                  (ELSE CONTINUE))
MC
END))
```

効率化変換戦略ファイル

上のプログラム中 TRY という文は WHILE 文形式で探索をおこなう文である。例えば ((TRY (U) END)) というプログラムは unfold の可能な GOAL を探索し実行する。UG は GOAL の順序を入れ換えるコマンドで、これと TRY 文の探索により AE つまり等価則の適用できる GOAL 列をみつけているわけである。MC は等価な CLAUSE をひとつにするコマンドである。この変換ファイルを効果が無くなるまで適用することで自動変換を達成している。

むすび

変換によって効率化を計るこの方法は“SIGN” “SIGN1” “EX” によって表された述語をもちいたモデルならばどのようなものでも変換の対象とすることができる。この方法はモデルの構造に依存しないため変換システムの上で自動的に変換を進め効率化できるようになった。

次に今までのモデルを変換するのに要した時間を示す。 EXAMPLE 1 の N2-TANK という述語では約５分、EXAMPLE 2 の N2-DTANK という述語では約２５分、EXAMPLE 3 の PRESS という述語では約１０分かかっている。この変換にかかる時間はもとのプログラムを直接 PROLOG 上で実行させる時間より長いが実時間応答を得ようとしたり、同じモデルを系の中で何度も用いる場合には効率向上の為の有効な方法であると思われる。

【参考文献】

（１）J.De Kleer , J.S.Brown ：”A Qualitative Phisics Based on Conflence”,Artficial Intelligence Vol.24 pp.7-83, 1984

（２）Ken Forbus：”Qualitative Process theory”,Artficial Intelligence Vol.24 pp85-168, 1984

（３）B. Kuipers：”Commonsence Reasoning about Causality:Deriving Behavior from Structure”, Artficial Intelligence Vol.24 pp169-203, 1984

（４）J.De Kleer ：”Choice without Backtracking” Prof. of AAAI,pp 79-85,1984

（５）西田豊明、川村正、堂下修司：“定性的推論におけるあいまい性と不連続の取扱について”、知識情報処理シンポジウム論文集、1985

（６）堂下修司：“定性的推論のソフトウェア化に関する研究”

（７）中村直人：Prolog プログラムの等価変換と変換戦略に関する研究、横浜国立大学修士論文