記号処理 12－8
（1980. 6. 20）

DIALOG-I：図式プログラミング言語

田村浩一郎　梅山伸二
（電子技術総合研究所）

1．はじめに

2次元図形を利用した図式（iconic）プログラミング言語は人間と機械のインタフェースとして優れている。特に分散（並列）プログラムの場合にそうである。システムDIALOG（Distributed computing, Iconic programmable, Asynchronous, Logic based, Optical hosing, general purpose）の一機能として，われわれは図式プログラミング言語DIALOG-Iを設計中であり，ここにそのスケッチを与える。この言語の基礎言語は一階述語論理であり，その表式にただちに変換できるため，変換後の表式の「証明」というかたちでプログラムが解釈される。システムDIALOGの概要については文献[1]を参照していただきたい。

2．プログラミング言語への要請

良いプログラミング言語とは，

（0）あらゆる問題領域に対して，正しく効率の良いプログラムが容易に作成でき，かつ作成されたものが理解しやすいこと

というように要約できる。LiskovとZillesは仕様言語への要請として，(1)形式的である，(2)書きやすい，(3)わかりやすい，(4)簡潔である，(5)一般的である，(6)修正が容易，という条件を挙げている[2]が，これらはプログラミング言語への要請として読みかえても全く妥当なものである。さらに，欲張った要請を追加するならば

（7）思考（の明晰化）を助ける。――いまや良いプログラミング言語は正しい（つまり論理的な）思考を助ける補助手段になりつつある。

（8）仕様から実現までの表現が統一していることにより，その流れに一貫性を保たせる。――仕様言語とプログラミング言語が分離していなければならない必然性はない。

（9）宣言的表現と手続的表現の融合。――KRLではこの表現が分離されているが，それはトラブルメーカになっている[3]。

（10）学習が容易。――人工言語なのであるから，学ぶ項目が少なく，明確な概念構成で組立てられているもの程良い。これは実上がりの言語処理系につながる。

この他，分散処理システムのためのプログラミング言語を目指すならば，

（11）できるだけ分散（並列）処理を生かしたプログラムを作りあげるように仕向けるような表現形式。

（12）分散（並列）処理のもたらす誤りを防ぐような表現形式。

が要請される。

これらの要請に対する満足度の評価に客観的な尺度を与えることはできない。しかし，DIALOG-Iの設計に際して絶えず心掛けるべきチェックリストとして利用して行くことにしたい。

3. 基礎言語 DIALOG-L

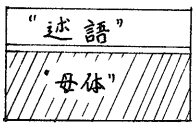
図式表現をとるDIALOG-Iの意味はDIALOG-Lで与えられる。つまり、DIALOG-IからDIALOG-Lへの変換規則が設定されている。これは、DIALOG-Lが中間言語として設定されていることとは必らずしも同一ではない。直接データ構造や機械語に変換されても何ら差支えないばかりでなく、むしろ効率を向上させる。しかし、DIALOG-Iの意味を明確に示すためには、意味の明確な何らかの言語表式が必要になる。この意味でDIALOG-LはDIALOG-Iの基礎言語と呼ぶ。

DIALOG-Lは一階述語論理のHorn集合である。この点に関してはPROLOGと同じであるが、PROLOGでの証明手順は逐次処理を前提にしているのに対して、DIALOG-Lの証明手順は非決定的である。但し、Horn集合の特徴を活かし、input resolutionを行なう。表現形式はスコーレム標準形を用い、正リテラルには＋、負リテラルには－を述語の前につける。定数は、引数なしのスコーレム関数としてあらわす。実用的にあれば便利な組込み句をいくつか持つ。たとえば算術演算や不等号など。この他、分散処理に固有な、「あれば便利な」組込み句もある。これについては後述。

4. DIALOG-Iの文法

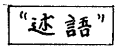
統語変数は〈 〉で示す。DIALOG-Lでの表示を右側に置く。

4.1 グラス箱（glass box） ⇒ 句（clause）



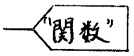
＋"述語" "母体"

4.2 暗箱（black box） ⇒ 負リテラルの述語



－"述語"

4.3 三角箱（triangle box） ⇒ スコーレム関数



"関数"

4.4 結線（connection line）

結線は母体内で、ガラス箱の内壁，暗箱および三角箱の外壁を結ぶ線分である。

4.5 結合点（connection point）

結合点で結ばれた結線は同一変数を持つことを示す。

4.6 接点（touch point） ⇒ 項（term）

箱の内外壁と結線との接続点を言う。

4.7 矢印（arrow）

結線の向きを与える。 —▷—

4.8 接点から項への変換

接点のあらわす項はすべて P（向き，項）の形になる。ここで向きは，in（ ）ないし，out（ ）ないしは変数に限られる。Pは予約スコーレム関数。

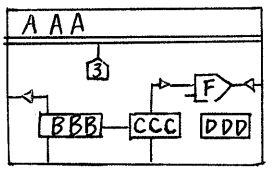
接点から項への変換を考えるために、便宜上、箱の壁にチャージ(charge)を考える。チャージは、ガラス箱の内壁を＋、暗箱の外壁を－とする。また、三角箱の外壁には、その三角箱が接続されている壁と等しいチャージを与える。こうすると、各接点の表わす項の向きは次のように決定される。

(1) 接点のチャージが＋(－)であり、接点に接続された結線の向きが接点に向かっていれば out( ) (in( )), その逆であれば in( ) (out( )).

(2) 接点のチャージが何であれ、結線の向きが規定されていなければ変数.

4.9 引数の場所. それぞれの箱の出発点を ○— で示す。この出発点から反時計方向に辺を辿り、その順により引数の場所(place)が与えられる。

4.10 例題



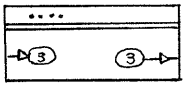
＋AAA (p(out( ), x1), p(d2, x2), p(d3, x3), p(in( ), F(p(out( ), y1), P(ddd2, y2))), P(d4, 3( )))
－BBB (p(out( ), x1), P(ddd1, x2), p(dddd2, y3))
－CCC (p(ddd3, y3), p(ddd4, x3), p(out( ), y1))
－DDD.

4.11 変数名

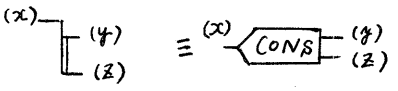
変数の識別は結線で示せるから、変数名を陽に出す必要はない。しかし、コメントとして、あるいは、結線がより組んで図が見にくくなるのを防ぐために、陽に与える手段がある。これは.

—("変数名")— あるいは, —"変数名" のように書く。

たとえば.

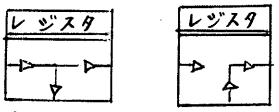


4.12 特殊スコーレム関数 CONS



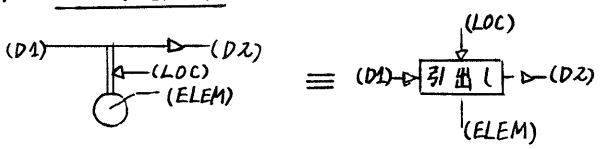
これは頻繁に使用するので特別な図形で示すことによりわかりやすくする。LISPのようにyをCAR, zをCDRと見立てるのは自由だが、これ自体そういう意味は全くなく、単なるスコーレム関数である。

4.13 例(レジスタ)

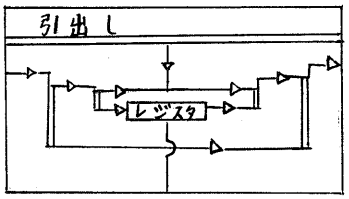


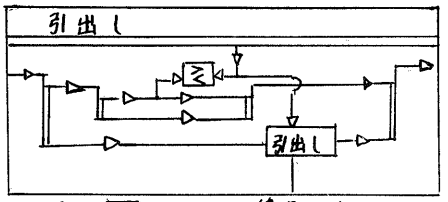
左側が読み出し、右側が書き込みになる。書き込みは、いわゆる single assignment になっている。

4.14 例(引出し)

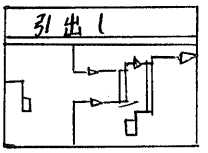


LOC(場所)を与えることによってELEM(要素)の値の読み出し、書込みを行なう。組込み句とする。定義は次頁。





(→≷← は不等号の組込み句)



(┐は NIL( ).)

これを組み込み句とするのは，LISPの属性リスト（Property list）に対応する単純なデータベースに使えること，およびハッシュコードなどにより効率の良い実現が可能なことによる。ただしこのままでは，アクセスの順序付けを規定しない共有メモリにはなっていないことに注意。

4．15 制御句

DIALOG-I(L)がプログラム言語となるためには，証明装置（prover）が不可欠である。この証明装置はHorn集合の上で完全でなければならないが，それだけに終らせれば，たとえ並列処理しようとも無駄なプロセスを多数発生し，実用的な意義がなくなる。そこでユーザのプログラム作成意図にできるだけ自然に沿った形で，しかも意味が明確な制御句が必要となる。前述した矢印（⊃）も実はそのひとつで，矢印の根本から先端へデータが送られる，と解釈される。これにより計算（＝証明）の方向性についての（部分的な）ユーザの意図を知ることができ，証明装置が利用できる。また前述のCONSも同様で，CAR部が時間的に最初のデータCDR部が残りのデータと解釈し，これによりデータストリームを表現したことになる。この他いくつかの制御句を以下に述べる。

4．16 ブレーキ（brake）

証明装置は，だまっていれば勝手に次々とプロセスを作りデータの送受を行ない，失敗（偽）とわかった時点で，前に送ったデータが誤まっていたことを知らせることになる。これにブレーキをかけるのがこの制御句「ブレーキ」である。あるがラス箱（句―プロセスに対応する）からの出力に際して，このブレーキ句のついたものは，その句が成功（真）になったことを確かめて初めて出力する。図では ——||—→ のように書く。これはPROLOGの"/"に相当し，意味合いは全く異なるが役割が似ている。無駄を省こうとしてやたらとブレーキをかけると餓死状態（starvation）になる。後述のソート例参照。

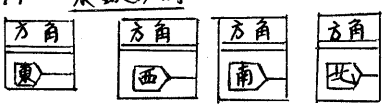
4．17 集録（collection）

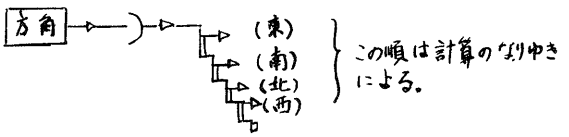
操作意味論的に見るならば，実行時には結線上を値（データ）が一般に複数個走る。これらを一度集めて，それ以降の処理に備える仕組みが応用上重要である。制御句「集録」は，結線上の確定値（発生したプロセスの真偽および値が確定したもの）を集合としてまとめあげるものである。たとえば発生プロセスの述語をPとすれば，集録の結果は $\{x \mid P(x)\}$ となる。図であらわせば，

(x)→)→

結果の集合はリスト構造化する（CONSを用いる）。この制御句の機能は，Friedmanらの FRONS関数[4]と全く同一ではないが良く似ていることに注意。

4．17′ 集録の例

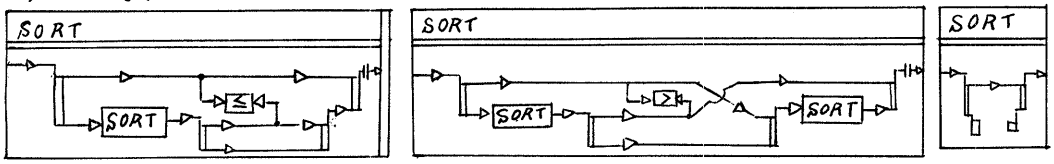




5. 例題　ソートプログラム

5.1 単純なソートプログラム

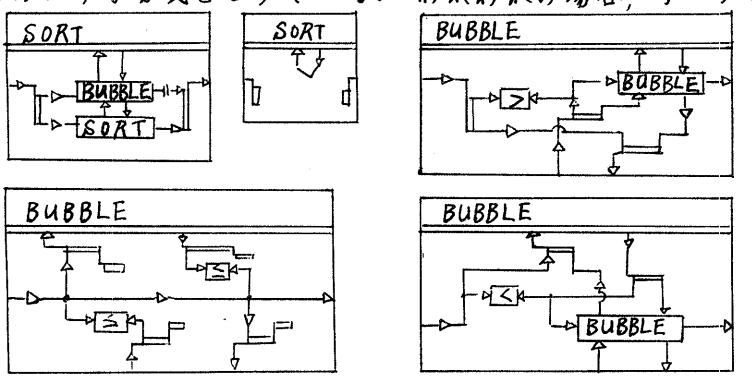
まずあまり凝ってない，従って効率は良くない，しかし安全確実なプログラムを作ってみる。



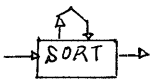
これだとCDR側のソートが終らなければ先きに行かないし，それに中の図の場合，もう一度ソートしなければならず，分散並列処理のうまみが活かされていない。もう少しましな方法はないだろうか。

5.2 泡型（bubble）ソート

小さいものは上に浮きあがり，大きいものは下に沈むことを繰返すように分散並列処理させる方式を考える。ここで難かしいのは全体がいつ終るのかを決める方法である。Francez [5] は，こういった，分割並列タイプのプログラムの制御について考察し，Sorting partition の例題を与えているが，制御信号をデータ信号に付加する方式をとっている。われわれの場合，データ信号だけを使う。



（使用時）

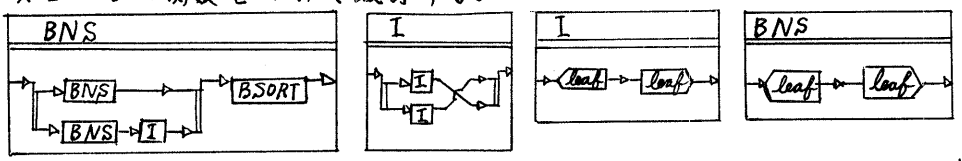


上行左端のガラス箱にあるブレーキに注意。

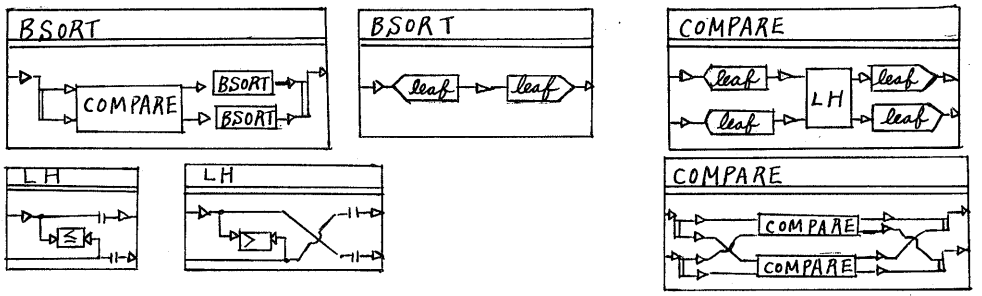
下行左端の箱にある NIL（F）が制御信号の役割を果たしている。Francez [5] の例題プログラムと対比させると興味深い。

5.3 双調（bitonic）ソート

Batcher [6] はハードウェア化したソーティング網をふたつ提案している。そのひとつが双調（bitonic）ソートである。これを Batcher Network Sort，略して BNS としてプログラムしてみる。入力も出力も2進木構造を仮定する。木の葉は，leaf というスコーレム関数をつけて識別する。



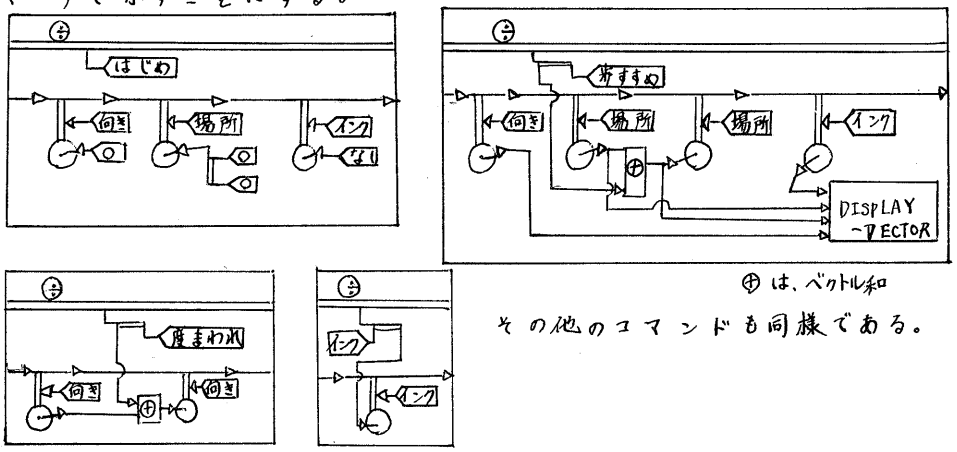
（次頁へ）



これが正しく働くことを簡単に見てとる訳には行かない。それを示すためには特にCOMPAREについての入出力規定の証明がいる（Batcher〔6〕がほぼ与えている）。また，COMPAREの再帰プログラムが動く部分で時間がかかるため，ハードウェア化した場合のように $O((\log n)^2)$ では動かない。

6、例題——タートル（turtle）

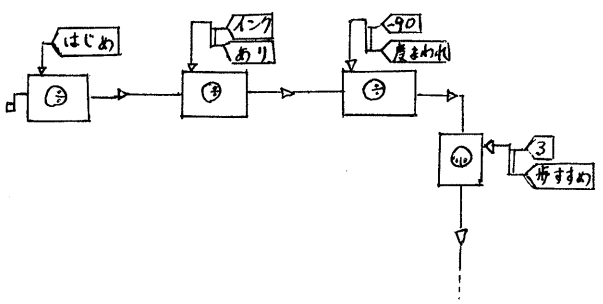
オブジェクト志向型言語の1例としてSMALLTALKをとりあげ，その簡単なプログラムをDIALOG-Iで記述してみる。SMALLTALKは必らずしも分散処理に適しているとは考えられないが，ユーザには極めてわかりやすい。SMALLTALKでのturtleを㊥のマークで示すことにする。



㊥は、ベクトル和

その他のコマンドも同様である。

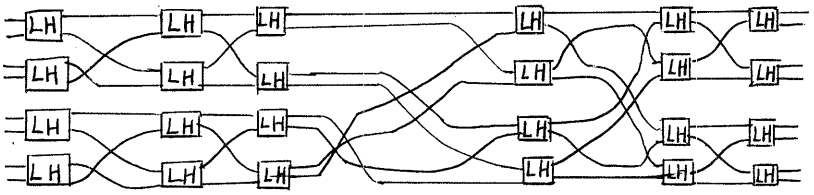
ここでturtleに図形を描かせるプログラムは，たとえば，



turtleの「状態」を知りたい時には，この図で軌跡のように使っている「引出し」から読み出せば良い。それによってturtle集団のプログラムが書ける。

7. 部分評価とハードウェア化

ここでいう部分評価（partial evaluation）とは，入力値として未確定なものを含んだまま計算を行なうことである。計算は確定値についてのみ進み，その他の部分は未評価のまま残される。これは一種のコンパイリングと考えることもできるが，DIALOG-Iでは，アルゴリズムのハードウェア化に役立つことを示す。よい3のBNSに8個のデータ（値は未定）を入れたとしよう。スコーレム関数は統合したあと消す。こうすると以下のようにBatcher網が自動的に出来あがる。



（矢印は略した）

今後LSIとCAD技術の進歩によりアルゴリズムの（ファームウェア化でなく）ハードウェア化が進むと思われる。その場合DIALOG-Iを使えば結線図の段階まで素直に自動化できる。これはDIALOG-Iの大きな副産物である。

8. おわりに

今後に残された課題は多い。効果的な制御句，効率の良い証明装置，表現力はどうか，ユーザにわかりやすいだろうか，といった問題がある。要請(0)は極めて厳しいものである。さらに実験と改良を進めたいと思う。なおこの研究を刺激していただいた当所佐藤孝平制御部長，並びに論理システム研究室のみなさんに感謝する。

文献

[1] 田村他：述語論理型プログラムの分散処理型アーキテクチャによる実現。情報処理学会記号処理研究会 11-1，(1980)

[2] Liskov & Zilles : Specification techniques for data abstractions, IEEE trans. Softw. Eng. 1, 1, pp. 9-19 (1975)

[3] Bobrow & Winograd : Experience with KRL-0, One cycle of a knowledge representation language. IJCAI V. pp. 213-222 (1977)

[4] Friedman & Wise : An approach to fair application multiprogramming. Semantics of Concurrent Computation, PP. 203-225 (1979)

[5] Francez : On achieving distributed termination, pp. 300~315, ibid.

[6] Batcher : Sorting networks and their applications, SJCC pp. 307-314 (1968)