システムソフトウェアと
オペレーティング・システム 67－17
(1994. 12. 9)

図によるベクトルデータフロー記述言語を自動的にプログラムに変換するシステム

平原貴行 *, 山之上卓 *, 安在弘幸 **

* 九州工業大学, ** 九州共立大学

804 北九州市戸畑区仙水町 1-1

あらまし　図によるベクトルデータフロー記述言語を自動的にプログラムに変換するシステムを現在開発中である．このシステムは，作図ツール Tgif+ により入力図を編集し，obj 形式でセーブしたものを入力してこれと等価なプログラムを得るものである．図にはデータフローを矢印で記述し，ボタンをつけることでデータの流れを制御する．このボタンを操作するための制御ルールは，スイッチの形で入力図内に記述する．本システムは，入力図をＣ＋＋プログラムに変換することを目指している．

和文キーワード　ベクトルデータフロー, 図, プログラム, 変換

A system which transforms a vector data flow description in a figure to a program

Takayuki Hirahara*, Takashi Yamanoue*, Hiroyuki Anzai*

*Kyushu Institute of Technology, **Kyushu Kyoritu University

1-1 Sensui, Tobata ward, Kitakyushu city 804

Abstract　We are constructing a system that transforms a vector data flow description in a figure to a program. This description is written by Tgif+, a drawing tool. In the figure, data flows are described by arrows, and they are controlled by buttons. The rule of controlling button is described in the figure by switchs. This system will transform the figure to a C++ program.

英文 key words　vector data flow, figure, program, transform

図によるベクトルデータフロー記述言語をプログラムに変換するシステム

平原貴行　　山之上卓　　安在弘幸

1994年11月10日

1　はじめに

計算機の処理内容を記述する方法の一つとして，図を用いてこれを記述する方法がある．図で記述することにより，システムの内容を直観的に把握できる．

我々は以前に，図で記述されたMachine Descriptionを自動的にプログラムに変換するシステム[4]を開発した．このシステムは，データ路図と制御線図の2つの図を入力することにより，これを等価なプログラムへ変換するものである．このシステムはスカラデータを扱うことを前提として開発したものであるが，これをベクトルデータに拡張して並列処理プログラムを生成するシステムを開発中である．本稿では，ベクトルデータフローを図で記述する手法とこれをプログラムに変換するシステムについて論じる．

本システムに入力される図は，Abelson & Sussman[1]が示したレジスタ計算機の概念に基づくものであり，これをベクトルデータフローへと拡張した．また，入力図の編集における負担を軽減するため，これまでデータ路図と制御線図に分けていた入力図を一本化するとともに，図を編集するツールとして，UNIX上で普及しているTgif+を使用することにした．

2　図によるMachine Descriptionの記述

図によるMachine Description記述の概念について説明する．

図はデータ路図と制御線図の2つからなる．データ路図は，レジスタ，演算器，条件判断器などの要素と，これらをつないでデータの流れの方向を示す矢印からなる．すなわち，計算機のハードウェア部分を記述するものである．設計者の意図通りに処理が行なわれるためにはデータの流れを制御することが必要である．そこで，“ボタン”という概念が用いられている．ボタンのついている矢印は通常はデータは流れず，ボタンが押されている間に限りデータが流れる．ボタンのない矢印には常にデータが流れていることになる．

このボタンが正しい順序で押されなければ意図した結果は得られない．そこで，その順序および条件を制御線図に記述する．すなわち，制御線図は計算機のソフトウェア部分を記述したものであるといえる．データ路図に記述したボタンには名前（ラベル）が与えられ識別される．制御線図では，ボタンを押す順序をこのラベルにより指定する．

3 入力図の記述

次に, 本システムへの入力図の記述方法について説明する.

3.1 ベクトルデータフローの記述

本システムでは従来のスカラデータに加え, ベクトルデータも扱う. そこで, 次のようにスカラーとベクトルを区別する. ベクトルデータが流れる矢印には／印をつけ, ベクトルの要素数, すなわち, そこに本来記述されるべき矢印の本数をそばに記述する.

また, すべての矢印（スカラー, ベクトルとも）にデータ型名を記述する. これはCプログラムへの変換に際して必要となる. ベクトルデータの矢印については,

{データ型名} * {要素数}

のように記述する（例： double*(N+1)）.

3.2 スイッチの導入

従来の手法ではデータ路図, 制御線図の２つの図を編集する必要があったが, これを１つにしぼって編集上の負担を軽減するため, 制御線図として独立していた制御ルールをデータ路図内に記述する方法をとる. そのために, 新たにスイッチという概念を導入する.

スイッチとは, データ路図中のレジスタ, 演算器などの要素の枠中に小さい枠を書き, その中に制御ルールを記述するものである. 通常, そのレジスタ等の内容および動作に関係する制御ルールを記述する. また, １つの要素中に複数のスイッチがあってもよい.

通常, スイッチには次のようなフォーマットの条件文を記述する.

{条件} ? : {処理}

条件には, 式や条件文（done,pass,start など）のほかに, ボタン名を書き, そのボタンが押されることを条件とすることができる. 処理には通常ボタン名を書くが, レジスタの場合, 値を書いて条件成立時にその値を代入させることもできる. また, end とした場合, 条件が成立すると処理を終了する.

3.3 Tgif+ による図の編集

今回, 入力図の編集に Tgif+ を用いることとした. Tgif+ は次のような利点を持つ作図ツールであり, これにより入力図の作成が簡単に, またあまり時間を書けずに行なえる.

- ほとんどの操作をマウスで行なえる.
- 矩形, 多角形, 直線, 円などの基本図形を簡単な操作で描画できる.
- 矢印を簡単に描画できる（頭部を自動的に描画する）.
- 図中にテキスト文字を記述できる.
- セーブファイルはテキストで書かれ, フォーマットも理解しやすく, プログラムなどで容易に解析できる.

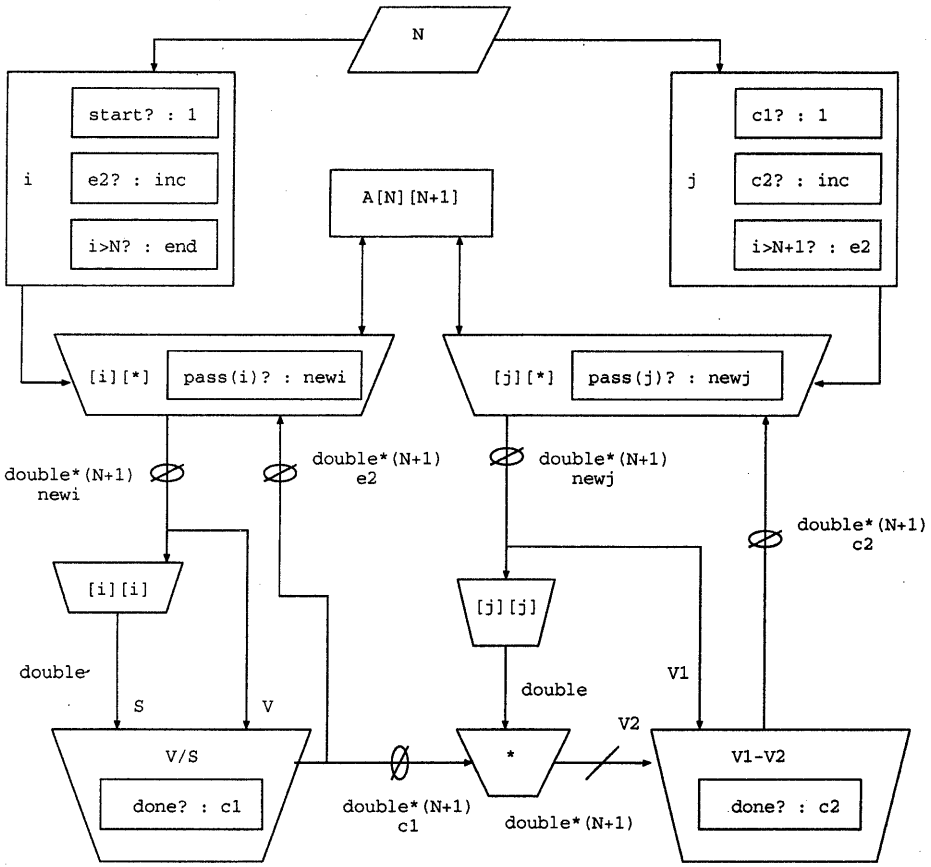

図 1: 入力図の例（ガウス掃き出し法）

4 図の認識, 解析処理

本システムの処理は (1) 図の入力および認識, (2) リンク処理, (3) 内容解析, (4) 変換処理の 4 段階からなる. システムは現在開発中で, リンク処理後に処理手順を解析する段階までができている. ここまでの処理について以下に説明する.

4.1 図の入力および認識

Tgif+ で編集した図の obj 形式のセーブファイルをシステムに入力する. obj 形式ファイルには図 に示す形で各図形の位置, 大きさ, 形状などを示すデータが記述されている. これより, 認識に必要なパラメータを読み込む. 多角形についてはその形状認識も行なわれる. 直線は矢印, ベクトル記号, 多角形に分類される.

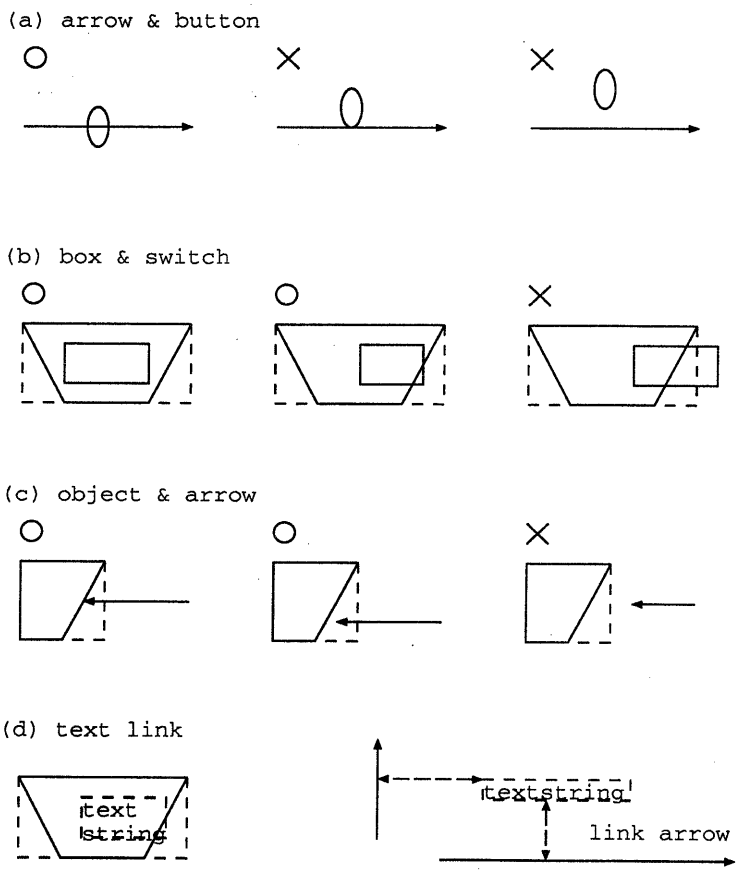

図 2: リンク判定の例

4.2 リンク処理

次に, 各図形, テキストなどを関係づけるリンク処理が行なわれる. リンクしているか否かの判定は図 で示した各図形のパラメータを比較することにより行なわれる.

まず矢印の枝分かれを調べ, 始点が他の矢印に接している矢印があれば, その接点から接している矢印の始点までの点を加え, 別々の矢印として認識させる.

次に矢印とボタン, ベクトル記号のリンクが行なわれる. 矢印がボタンを貫く形になっていればその矢印はボタンありと判定する (図 2(a), ベクトル記号についても同様) .

次に矩形, 多角形とスイッチ枠とのリンクが行なわれる. ここでは, 中に含まれているかの判定のために多角形を覆う矩形枠 (矩形の場合はそのもの) を用いる (図 2(b)) . ここでリンクされた矩形, 多角形とそのスイッチの集合をオブジェクトとして扱う.

次にオブジェクトと矢印をリンクして入出力関係を認識する. オブジェクトと矢印のリンクは前述の矩形枠を用い, その中に矢印の端点が入っているかで判定する (図 2(c)) . 続いて矢印により結ばれた 2 つのオブジェクトの組について, その入出力の関係が記述される.

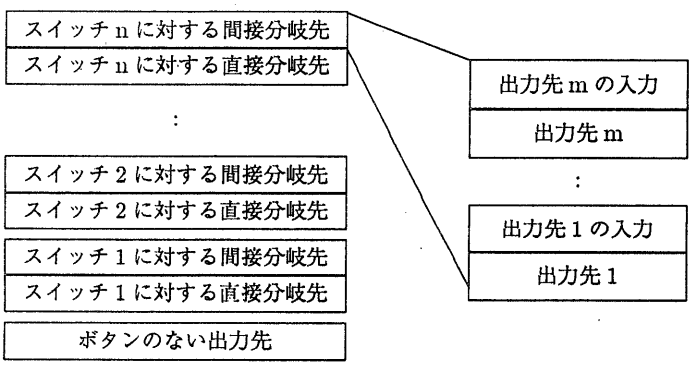

図 3: 分岐スタック

最後にテキストのリンクが行なわれる. 各テキストについて, まずオブジェクトとのリンクを調べ, どのオブジェクトともリンクできない時は矢印とのリンクを調べる. テキストのリンクはテキストを囲む枠を用いて調べ, オブジェクトとのリンクはスイッチの場合と同様にして行なう. スイッチがあればそのスイッチとのリンクも調べる. 矢印とのリンクはテキスト枠と矢印との距離で判定し, 最短距離の矢印（ただし, ある一定以下の距離であること）とのリンクを行なう（図 2(d)）.

4.3 処理手順解析

リンク処理が終了すると内容解析に移る. これは, 各オブジェクトの内容を解析して中間コードを生成しながら処理の手順を調べていくものであるが, 現在, 処理手順の解析のみができている.

処理手順の解析は次のように行なう. start が記述されたオブジェクトからスタートして, 各オブジェクトの内容を解析した後, 出力オブジェクト番号を分岐スタックにプッシュし, その中から最後に収めた 1 つをポップして処理を移していく.

分岐スタックには次の優先順位に従い分岐先がプッシュされる. まず, 条件文の後件部がボタン名となっているスイッチがある場合, そのボタンに対応する分岐先をプッシュする. この分岐先のリストは内容解析前にあらかじめ作成されたもので, そのボタンを持つ矢印からの入力先（直接分岐と呼ぶ）と, そのボタンが条件文の前件部に書かれているスイッチを持つオブジェクト（間接分岐と呼ぶ）に分けて作成されている. スタックには先に間接分岐先を, 続いて直接分岐先を収める. これに続きボタンのない矢印による出力先をプッシュする. オブジェクト内のスイッチがボタンを押す処理を指示しないとそのボタンつきの矢印にデータを流せないので, この手法によりデータの流れの制御が実現できる（分岐スタックは図 3参照）.

スイッチによる分岐が生じた場合, その分岐先のオブジェクトにどこからデータが入力されるのかが問題となる. そこで, 分岐スタックには分岐先とともに, その入力オブジェクト（そのボタンを持つ矢印につながるもの）もプッシュすることとする（図 3）. 従って読み出す際には 2 回ポップする必要がある.

演算器のような複数の入力を必要とするオブジェクトの場合には, 1 つの入力だけ処理して次のオブジェクトへ移ろうとした場合, 他の入力のコードを生成せずに終ったり, 違う場所にコードが書き込まれる可能性

```
(現在オブジェクト)   (入力オブジェクト)   (分岐スタック)
  0:i                                    stack(0 2 )
  2:[i][*]          {<-0:i}              stack(2 3 2 4 )
  4:V/S             {<-2:[i][*]}         stack(2 3 )
  3:[i][i]          {<-2:[i][*]}         stack(3 4 )
  4:V/S             {<-3:[i][i]}         stack(10 1 4 5 )
  5:*               {<-4:V/S}            stack(10 1 )
  1:j               {<-10:N}             stack(10 0 4 2 1 8 )
  8:[j][*]          {<-1:j}              stack(10 0 4 2 8 7 8 6 )
  6:V1-V2           {<-8:[j][*]}         stack(10 0 4 2 8 7 )
  7:[j][j]          {<-8:[j][*]}         stack(10 0 4 2 7 5 )
  5:*               {<-7:[j][j]}         stack(10 0 4 2 5 6 )
  6:V1-V2           {<-5:*}              stack(10 0 4 2 10 1 6 8 )
  8:[j][*]          {<-6:V1-V2}          stack(10 0 4 2 10 1 )
  1:j               {<-10:N}             stack(10 0 4 2 )
  2:[i][*]          {<-4:V/S}            stack(10 0 )
  0:i               {<-10:N}             stack()
```

図 4: 処理手順解析結果

がある．そこで，これらのオブジェクトに入力フラグを設け，入力を受け取ったらフラグを立てる．立ってないフラグがある場合，つまり未入力データがある場合はオブジェクト解析の前に分岐スタックからポップされた分岐先へ移って処理を行なう．フラグがすべて立ったらオブジェクト解析に移る．

すでに解析済みのオブジェクトに達した場合，解析を行なわずに分岐スタックからポップした分岐先へ移る．この時分岐スタックが空であれば解析処理を終了する．

処理手順解析の結果を図 4に示す．現在オブジェクトの列は現在処理中のオブジェクトの番号および名前を示し，上から下の順に解析が行なわれている．内は現在のオブジェクトにデータを渡すオブジェクトの番号および名前を示す．() 内は分岐スタックの状況を示す．右側より，分岐先のオブジェクト番号とその入力オブジェクトが分岐の優先順に並び，次のオブジェクトに移る際，右側からポップされる．

例えば，3 行目の 4:V/S では，オブジェクト 4(V/S) オブジェクト 2([i][*]) からデータを受けとって V に代入（図 1）する．しかし，S（ここではオブジェクト 3([i][i])）のデータがまだ入力されていないので，入力スタックから 2 つデータをポップする．最初にポップされたオブジェクト 3([i][i]) が次の処理対象オブジェクトとなる（4 行目）．次にポップされたオブジェクト 2([i][*]) はこのオブジェクト 3 にデータを送るオブジェクトとして読み込まれる．オブジェクト 3 は 1 入力であるからこのまま処理され，現在のオブジェクト 3 とその出力オブジェクト 4(V/S) をスタックにプッシュした後これをポップする．よって次に処理されるオブジェクトは 4 となり，オブジェクト 3 からデータが渡されることになる．このデータは S に代入（図 1）され，これにより V/S の入力データが出そろう（実際はここでオブジェクト 4 の解析が行なわれる）．次にオブジェクト 4 の出力が分岐スタックにプッシュされる．オブジェクト 4 にはスイッチが 1 つあり，その条

件成立時の処理がc1となっている（図1）．ボタンc1をもつ矢印はオブジェクト5(*)に向かっている（直接分岐）．またオブジェクト1(j)にはc1が条件であるスイッチ（図1）が含まれる（間接分岐）．間接分岐を優先してプッシュするので，まずオブジェクト1に入力するオブジェクト10(N)とオブジェクト1自身がプッシュされ，続いて直接分岐であるオブジェクト5が矢印c1による入力オブジェクト4とともにプッシュされる．オブジェクト4にはボタンのない出力矢印はないため，この後スタックからのポップによりオブジェクト5へ移る．

最終行の0:iでは，すでにオブジェクト0(i)が最初の行で解析済みのため，スタックの検査のみが行なわれる．この時スタックは空であるのでここで処理は終了する．

5 おわりに

今回，図によるベクトルデータフロー記述言語を自動的にプログラムに変換するシステムを提案した．このシステムは入力図の編集にTgif+を用い，制御ルールをデータ路図中に記述することにより図の編集作業を簡単なものとしている．今後，オブジェクト解析，中間コード生成，プログラムへの変換部分を作成していく．最終的には，C＋＋プログラムに変換することを目指している．

参考文献

[1] H.Abelson, G.J.Sussman, J.Sussman, Structure and Interpretation of Computer Programs, MIT Press, 1984

[2] T.Yamanoue, H.Hayata, H.Anzai, A Figure language for distributed system descriptions and its pre-compiler which can parse figures, Proceedings ICSC'92: Second International Computer Science Conference, pp.425-431, 1992

[3] T.Yamanoue, H.Hayata, H.Anzai, A figure programming language for paralel supercomputers Transputes and Parallel Aprications, IOS Press, pp.209-214, 1993

[4] 山之上卓，平原貴行，早田弘一，安在弘幸，図で記述されたMachine Descriptionを自動的にプログラムに変換するシステム，平成5年後期情報処理学会全国大会,5,121-122,1993