ビジュアルプログラムが持つ図形情報を利用した動的プロセスネットワークの可視化手法

~KLIEG VPE におけるプログラム実行の可視化

A Technique for Visualizing Dynamic Process Network Using the Diagram Information of Visual Program:

Visualizing the Dynamic Behavior of KLIEG Program Execution

志築 文太郎[†] 豊田 正史[†] 高橋 伸[†] 柴山 悦哉[†]
Buntarou SHIZUKI Masashi TOYODA Shin TAKAHASHI Etsuya SHIBAYAMA

[†]東京工業大学 情報理工学研究科 数理・計算科学専攻
Department of Mathematical and Computing Sciences Tokyo Institute of Technology

概 要

データフロー図に基づくビジュアルプログラミング言語を対象とし，実行過程をアニメーション表示する手法を示す．この手法では，実行時に生成されるプロセスネットワークの形状を歪みなく表示するため，Bartram らが提案した Continuous Zoom アルゴリズムを修正し用いる．しかしながらプログラムは実行過程において同種のプロセスを繰り返し生成する場合が多く，同アルゴリズムを単純に適応しただけでは表示がうまくゆかないことが多い．我々の手法ではプロセスネットワーク中の繰り返し構造を抽出し，省略表示することによって対処する．この手法を利用したトレーサを我々が開発中の KLIEG プログラミングシステムの一部として実現した．

1 はじめに

プログラムの実行過程を視覚化することによって実行時の挙動を分かりやすく提示するための方式として，現在までに様々なものが提案されている．ビジュアルプログラミング言語の場合，プログラムとして入力された図形の形状をヒントとした実行の可視化が可能であり，テキストベースのプログラミング言語に比較して，分かりやすい実行過程の表示を(半) 自動的に行える可能性が高い．しかしながら，面積の限られた画面にプログラム実行の過程を表示するためには以下の 2 点を可能にする必要がある．

1. 同一画面に概観表示と詳細表示を行う
2. 詳細に表示すべき場所をユーザが容易に指定できる

入力されたビジュアルプログラムの形状に類似した形態でプログラムの実行過程を表示するシステムとしては Pictorial Janus [1], VIPR [2] などが挙げられる．しかし，これらのシステムにおいては 1 は実現されているが 2 についてはあまり考慮されていない．小さなプログラムでも表示が複雑になり得る，参照することができない部分を生じるないしは参照したい部分を詳細表示させるのに多くの操作を必要とする，という問題がある．

本稿では，データフロー図に基づくビジュアルプログラミング言語を対象とし，言語の特性を生かした省略表示を行うことによって上記の 1, 2 を満たすようプログラム実行過程を視覚化する手法を提案する．

以下，2節においては，本論文が視覚化の対象とするビジュアルプログラミング言語について説明し，その言語で記述されたプログラムの実行過程を Continuous Zoom アルゴリズム [3] を利用して表示する方法を示す．3節では単純に Continuous Zoom を適用する際に生じる問題点を指摘し，今回我々がその問題点に対してとった解決法を述べる．さらに，その手法を利用したビジュアルプログラミング環境 KLIEG [4] のトレーサについて 4節で述べる．

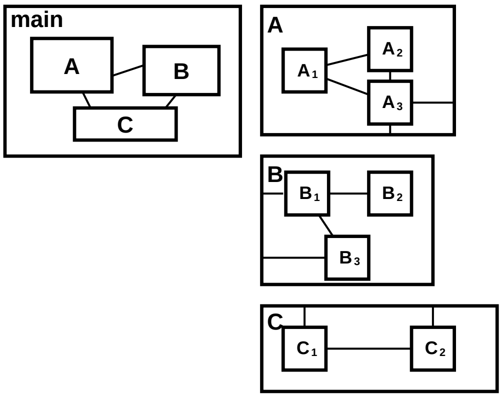

図 1　データフロー図に基づく **VPL** でのプロセス定義: ネットワーク図により定義する.

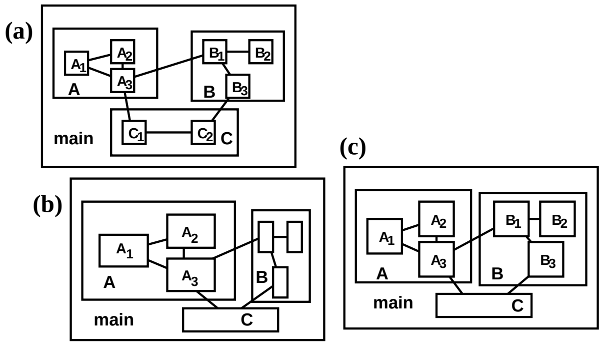

図 2　階層的に構成されるプロセスネットワークと, そのネットワークの **Continuous Zoom** による表示変更

2 Continuous Zoom アルゴリズムを利用したデータフロー型 VPL の動的プロセスネットワーク表示

本論文の手法ではデータフロー図に基づくビジュアルプログラミング言語を扱う. プロセスは階層的に配置され, 通信路によって結ばれる. 実行の可視化は, 実行時に生成されるプロセスネットワークを表示することによって行う. このためには Continuous Zoom アルゴリズム [3] を利用する.

データフロー型 VPL

図 1はデータフロー図に基づくビジュアルプログラムの定義を模式的に表現したものである. ここではプロセス main, A, B, および C が定義されているとする. この図において, 長方形はプロセスを表し, 長方形を結ぶ線分はプロセス間通信路を表現する. プロセス main は子プロセス A, B, C から構成されており, A は A_1, A_2, A_3という 3 つの子プロセスから構成されている. プロセス main を実行すると図 2(a) に示されるような階層的なプロセスネットワークが生成される.

図 2(a) に示したプロセスネットワークは比較的単純である. しかし一般には, プロセスは再帰的に定義される可能性があるため, 実行時に生成されるプロセスネットワークは複雑になり得る.

Continuous Zoom アルゴリズム

Continuous Zoom アルゴリズム [3] は fisheye view [5] と呼ばれる表示手法で, 図 2(a) のように階層的な構造を持つネットワークを表示対象とし, 概観表示と任意の部分の詳細表示を同一画面内に表示するアルゴリズムである. 画面表示の調整法としては以下の 2 つをユーザに提供する.

- あるノードを拡大 ·縮小する
- あるノードの内部を参照する必要がない場合に内部を見せないよう指定し他のノードに画面空間を割り当てる

例えば図 2(a) のネットワークに対して Continuous Zoom アルゴリズムを用いて変更を加えた後の表示を図 2(b), (c) に示す. 図 2(b) は (a) 中のノード A を拡大表示したものである. A が拡大した結果 B, C が対応して縮小されている. 図 2(c) は 2(a) 中, ノード C の子ノード C_1, C_2を表示しないよう指定したものである. 結果としてノード A, B に空間が配分され拡大されている.

さらに Continuous Zoom アルゴリズムの特徴としては

- ノードの相対位置の歪みがよく抑えられる
- 複数点の詳細表示ができる
- 各ノードに定義された Degree-Of-Interest(以下 DOI と略す) による重み付けが可能である
- 拡大, 縮小の際になめらかなアニメーション表示が可能である

などが挙げられる.

このようにして Continous Zoom アルゴリズムは概観表示, および複数個所の詳細表示を両立させ, しかも歪みの少ない表示を可能としている. さらに, 表示されていないノードに対しても, その親ノードに視点を当て拡大することによって表示することができる. したがってネットワーク中の任意のノードが間接的に参照可能である.

データフロー型 VP への適用

プロセスネットワーク図を Continuous Zoom アルゴリズムを用いて表示するためには, 各プロセスの座標 (歪み表示を行う前の座標), および DOI をアルゴリズムへの入力として与える必要がある. 本論文の方式ではこれらを以下のように定める.

座標　各プロセスの座標はビジュアルプログラムとして入力された図形の形状に基づいて定め

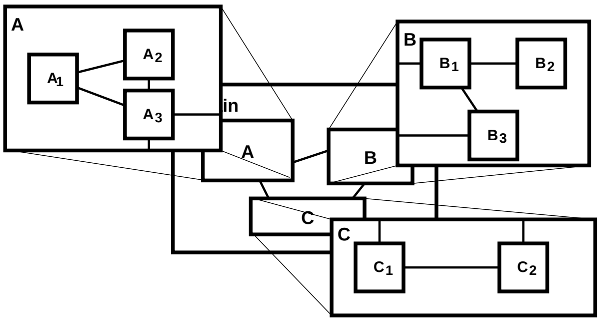

図 3　**Continuous Zoom** アルゴリズムに与えるネットワーク配置の決定法

る. 例えば図 1のようにプロセスが定義されている場合, プロセス main において A, B, C が占めている領域に図 3のようにそれぞれのプロセスの定義をそのまま当てはめる. これによって図 2(a) のような配置を得る.

DOI　各プロセスに DOI を定義する. 新しく生成されるプロセスの DOI は親プロセスの DOI と同じ値を継承する. 動作中のプロセスの DOI は正の値とし, 計算が終了したプロセスには 0 を割り当てる. これにより動作中のプロセスに対して画面空間を多く配分し, 表示になるべくプログラムの動作状態を反映させる.

なお, プロセス同士が近接しすぎないよう, 表示の際には空白部分にも一定以上のスペースが確保されるようにする*.

3　Continuous Zoom の問題点と繰り返し構造の抽出・省略による解決

Continuous Zoom では, あるノードの子ノードが複数存在する場合, すべての子ノードをすべて見せるかすべて見せないか, のどちらかしか許されない. このため [3] に指摘されているように, 一つのノード内に非常に多数の子ノードが存在するようなネットワークを画面内に収めて表示できないという問題がある. 例えばノードが一列に整列した場合, 表示可能なノード数は高々数十である. このためプロセスネットワークを表示する際には以下のような問題点が生じる.

例えば map というプロセスはリストを受け取り, それぞれの要素に対して proc プロセスを生成し計算させるものとする. この場合, 入力されたリストの要素の数だけ proc プロセスが生成される. 結果として図 4(a), あるいは (b) のようなネットワークが生成される. ここで Continuous Zoom は, 子ノードをすべて見せるかすべて見せないかのどちらかで表示を行うため, 図 4(b) のように proc の数が増加するにつれて map プロセスの内部を表示することができなくなる.

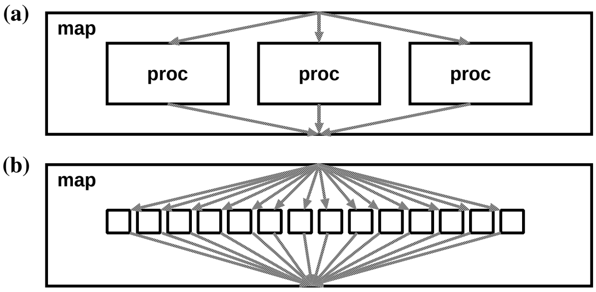

図 4　多くのノードが繰り返し現れるネットワーク

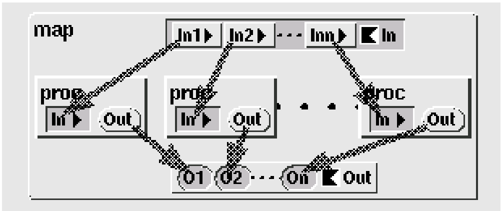

図 6　複数の proc プロセスを生成するシーケンスパターン **map** の定義

この問題に対して本論文で提案する方式では以下のように対処する. 一般に, プログラムを実行すると同種のプロセスが繰り返し生成されることがよくある. プロセスの大部分がこのような形で生成されるプログラムも珍しくない. この繰り返し構造を抽出し, その部分を省略表示することによって効果的に表示量を抑制し, 上記の問題を回避できる可能性がある. 上の例では, 生成されたすべての proc を表示するのではなく, そのうちの幾つかのみを表示することによって情報の損失を少なく表示量を抑制することができる. また, 省略されていることを「···」のように示すことによってその部分に同種のプロセスが存在することを示すことができる. さらにこの「···」部分にアクセスし省略されているプロセスを参照できるようにしておくことで, すべてのプロセスを参照する手段を確保できる. 結果的に, 情報の損失を抑さえつつ効果的に表示量を抑制することが可能になる.

4　KLIEG VPE におけるトレーサの実現

我々は KLIEG ビジュアルプログラミング環境 [4] の開発の一環として, トレーサを作成した. トレーサは KLIEG プログラムが実行する過程で生成・変

---

* これは Continuous Zoom アルゴリズムでの blank section に対しても minimum requirement を設定することによって行う.

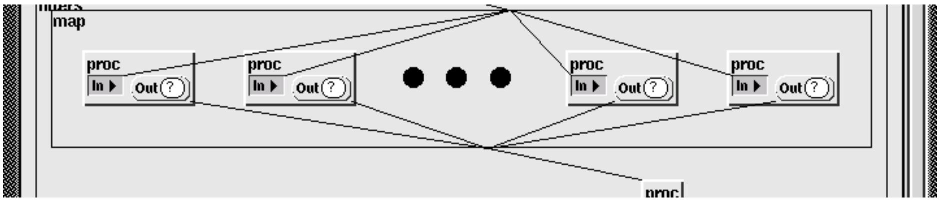

図 5　トレーサにおける同種のプロセス列の表示: 本論文で提案した省略表示を行った場合

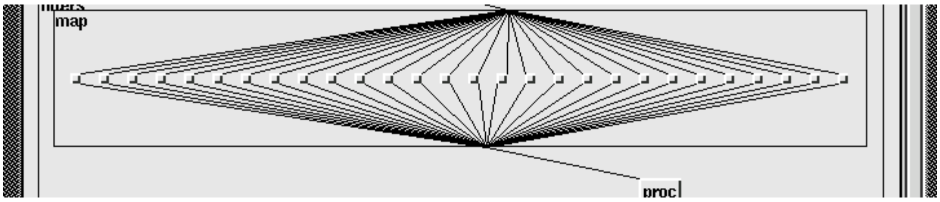

図 7　トレーサにおける同種のプロセス列の表示:省略表示を行わない場合

化するプロセスネットワークをアニメーション表示する.

これまでに述べた手法を利用したトレーサの表示例を図 5に示す. この図では多数生成されている proc プロセスの列が省略表示されている. 同種プロセスの繰り返し構造は,KLIEG 言語にプロセス列を生成するシーケンスパターンと呼ばれる機構を組み込むことによって実現した. すなわち, シーケンスパターンを用いて生成されたプロセス列を省略表示する. 例えば, 図 5で省略表示されているプロセス列は図 6に示されるシーケンスパターン map から生成されたものである. map は In から入力されたリストの要素 (図 6では In$_1$,In$_2$, ..., In$_n$) それぞれに対して proc プロセスを生成する. また, それぞれの proc プロセスから得られた計算結果はリスト (図では O$_1$, O$_2$, ..., O$_n$) として出力する.

逆に, 同じプロセスネットワークを省略表示を用いない場合のトレーサ表示を図 7に示す. 図 7 では生成された proc プロセスが増えすぎたため一つ一つに配分される空間が少なくなっている. これ以上 proc プロセスの数が増加した場合, map プロセスの内部は表示されなくなる. それに対して図 5のように省略表示を行う場合, proc プロセスの数に多少にかかわらず map プロセスの内部を参照することができる.

5　まとめと今後の課題

データフロー図に基づくビジュアルプログラミング言語の実行過程を表示するために Continuous Zoom アルゴリズムを用いた表示方法を提案し, これを KLIEG プログラミング環境のトレーサに適用した. これによって一画面にプロセスネットワークの概観を表示するとともに, その任意の部分を詳細に表示することができる. さらに Continuous Zoom アルゴリズムの問題点であるノード数の多いネットワーク表示についてはプロセスネットワーク内の繰り返し構造を抽出・省略表示することによって対処した.

今後の課題としては, 木構造, メッシュ構造など様々な形態のネットワークを省略する方法を考えること, DOI にユーザの意図を反映させることによってユーザに要求される操作を減少させること, などが挙げられる.

参考文献

[1] Kenneth M. Kahn, Vijay A. Saraswat: Complete Visualizations of Concurrent Programs and their Executions, Proc. 1990 IEEE Workshop on Visual Languages, October 1990.

[2] Wayne Citrin, Michael Doherty, Benjamin Zorn: The Design of a Completely Visual Object-Oriented Programming Language in "Visual Object-Oriented Programming: Concepts and Environments", Margaret Burnett, Adele Goldberg, Ted Lewis, Editors, New York, Prentice-Hall, 1995.

[3] Lyn Bartram, Albert Ho, John Dill and Frank Henigman: The Continuous Zoom: A Constrained Fisheye Technique for Viewing and Navigating Large Information Space, Proceedings of UIST '95, November 1995.

[4] 豊田正史, 志築文太郎, 高橋伸, 柴山悦哉:並列ビジュアルプログラミング環境 KLIEG:プロセスネットワークパターンによる柔軟な再利用機構の導入:情報処理学会研究報告 96-PRO-6(1996 年 3 月 26 日・27 日), 電子情報通信学会技術研究報告 SS95-46(1996-03).

[5] George W. Furnas: Generalized Fisheye Views, Proceedings of ACM SIGCHI '86, pp. 16–12, April 1986.