代数的仕様の解析・検証・変換のための視覚的支援環境

Visual Support Environment for analysis, verification, and transformation of Algebraic Specifications

河口 信夫† 坂部 俊樹† 稲垣 康善†
Nobuo Kawaguchi Toshiki Sakabe Yasuyoshi Inagaki

†名古屋大学工学部
Faculty of Engineering, Nagoya University

abstract

In this paper, we propose an environment for algebraic specifications with Graphical User Interfaces. The aim of the environment is to support analysis, verification and transformation of algebraic specifications by virtue of visual representations. Four kinds of visualization technics are described in this paper. *Term Visualization* helps us to analyze the structure of a term by providing us with graphical tree representation. *Computation Visualization* reveals the dynamic behavior of the computation by the Sequence Viewer. The Tower of Hanoi example shows the usefulness of the visualization.

1 はじめに

ソフトウェアの信頼性を向上させるために，様々な形式的手法が提案されている．その一つである代数的仕様記述は，プログラムの意味を代数として，すなわち抽象データ型として捉える手法であり，等式論理を用いて厳密に意味を記述することができる．

代数的仕様の検証法には，Knuth-Bendix アルゴリズム [1]や被覆集合帰納法 [6]など，すでに多くの成果がある．しかし，これらの手法で検証が行なえるのはプログラムの型エラー，もしくは論理的誤りであり，仕様作成者の勘違いによる誤り，すなわち型エラーや論理的誤りではないような，本質的な誤りを発見することはできない．このような誤りを見つけることは非常に困難であり，実際にプログラムを動作させてみて初めて見つけられる場合が多い．また，正しく動作しているプログラムの効率を改善する場合，プログラムの解析を行ない，効率のネックとなっている部分を調査する必要があるが，これも多くの場合実際にプログラムを動作させることを必要とし，非常に困難な問題である．このような実際の実行時のプログラムの動的な振舞いの解析を支援するツールについての研究が最近注目を浴びている [9].

一方，等式の集合からなる代数的仕様を左辺から右辺への書換え規則と考えると，これは項書換え系であり，一種の実行可能仕様ととらえることができる．項書換え系は近年注目を集めている関数型の単純な計算モデルであり，

- 記号の書換えのみで計算が進む．
- 参照透過性を持つため，部分的な検証が可能．

などの特徴を持つ．そのためプログラムの検証や変換などの操作が比較的容易に行なうことが可能で，すでに様々な成果が得られている [14].

項書換え系にはすでに様々な処理系が存在し，多くの事例研究が行なわれている [5]．しかし，ほとんどのシステムがテキスト処理を基本としており，直観的な洞察を必要とするような「本質的な誤りの発見」，「効率の向上」を行なうには機能が不十分である．そこで我々は「視覚化 (Visualization)」の概

念を用いて直観的理解・洞察の支援を行なうことを提案する．まず，項書換え系の計算における視覚化を次のように分類する．

- **項の視覚化:** 項を様々な情報により拡張した木で表現する．
- **計算の視覚化:** 項書換えの計算の進行を表す．
- **操作の視覚化:** 項や規則に対する操作を視覚的に行なう．
- **情報の視覚化:** 計算から得られる様々な数値情報をグラフ化する．

本稿ではこれらの視覚化について議論するとともに視覚的手法による項書換え系の検証・変換について述べる．また，我々はこれらの視覚化を実際に実現した「視覚的項書換え支援システム:TERSE」[15] [16] [17]を作成している．

以下，2節で項書換え系の基本概念について，3節では各々の視覚化についての詳細を述べる．4節では，視覚化を用いた実際の解析，変換例を示す．5節では我々が実現したシステムについての解説を行なう．最後に，6節で他の支援システムとの比較を行ない，本稿で提案された視覚化についての評価を行なう．

2 項書換え系

本稿では主に項書換え系を扱うので，代数的仕様記述については[4]を参照されたい．項書換え系について以下に本稿での議論に必要な定義を示す．詳しくは[3]等を参照されたい．

可算無限個の変数記号の集合 V，アリティをもつ関数記号の集合 F から帰納的に定義される項集合を $T(F,V)$ と書く．変数を含まない項は基底項と呼ぶ．関数記号は構成子関数記号の集合 C と被定義関数記号の集合 D に区別されるとする．

項 t の出現の集合 $O(t)$ は次の条件を満たす最小の自然数列の集合である．

1. $\varepsilon \in O(t)$
2. $iu \in O(f(t_1,\ldots,t_n))$ **if** $u \in O(t_i),\ 1 \leq i \leq n$

出現 u,v に対して w が存在して $v = uw$ となるとき $u \preceq v$ とする．もし $w \neq \varepsilon$ ならば $u \prec v$ と書く．また $v = uw$ のとき v/u は w を表す．$u \not\preceq v, v \not\preceq u$ の時は $u \mid v$ と書く．項 α に対し出現 u が指す部分項を α/u と書く．

項書換え系は2項の対 $\alpha_i \rightarrow \beta_i$ の集合からなり，この対を書換え規則と呼ぶ．規則の左辺 α_k の変数に適当な代入を施した項が項 t に一致するとき，t をリデックスと呼ぶ．一般に項は複数のリデックスを部分項として持つ．また，リデックスを部分項に持たない項を正規形と呼ぶ．項の書換えとは，その複数のリデックスから戦略と呼ばれる方針に従って書換える部分項を選び，右辺にその代入を施した項に書換えることをいう．項書換え系に無限の書換え系列が存在しないとき，その項書換え系は停止性を持つという．

【定義 2.1 】出現の残余

項 t から t' への規則 $\alpha \rightarrow \beta \in R$，出現 u における書換えについて，出現 v の残余 V は以下のように定義される．

$$V = \begin{cases} \{v\} & \textbf{if} \quad v \prec u \quad \textbf{or} \quad v \mid u \\ \{uwv' \mid \beta/w = x\} & \textbf{if} \quad v = uw'v' \\ & \textbf{and} \quad \alpha/w' = x \in V \\ \{u\} & \textbf{otherwise} \end{cases}$$

□

出現の残余は，書換えによる出現 v の移動先の出現の集合を表す．出現 v がリデックスの部分項を指している場合 $(u \prec v)$，残余は書換えた部分項の出現になる．規則の左辺の変数 x にマッチングしている部分項を出現 v が指しており，規則の右辺に x の出現がない場合，残余は空集合になる．逆に右辺に複数の x の出現があれば，残余はコピーされた複数の出現の集合となる．

また，以下では無曖昧，左線形の項書換え系を扱う．この2つの条件を満たす項書換え系は合流性を持つことが示されている[3]．

3 視覚化

視覚化 (Visualization) には様々な定義があり，視覚的プログラミング言語[7]のように言語そのものが視覚的要素からなるものや，視覚的環境[8]のように本来テキストによるものに視覚的な操作環境を与えたものなどがある．本稿で述べる視覚化は，後者の立場に立ち，計算機から得られる，もしくは計算機に与える情報を人の直観的理解が行ないやすい形に変換することを指す．一般のプログラミング言語に対し，代数的仕様は単純であるため，その視覚化についての議論は形式的に行なうことができる．本節では，代数的仕様の直接実行における視覚化を項の視覚化，計算の視覚化，操作の視覚化，情報の視覚化に分類し

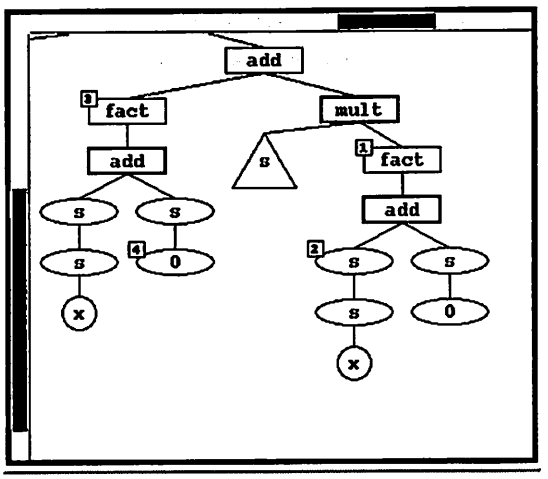

図1　視覚化された項

て説明する．

3.1　項の視覚化

書換え対象の項は計算の静的な状態を表す．また，項書換え系は書換え規則の集合であり，規則は項の対であるので，項の視覚化は同時にプログラムの視覚化でもある．項の視覚化は以下のことの実現を目的としている．

- 巨大な項の任意の部分の解析．
- 項の構造の視覚的理解．

項の構造解析を視覚的に行なう場合，項からできるだけ多くの情報を得られることが望ましい．また視覚化することで初めて可能になることもある．これらをまとめると以下の要求になる．

1. 項のソートの種類を判別したい．
2. リデックスの出現を調べたい．
3. 構成子記号，被定義関数記号，変数記号の判別がしたい．
4. 特定の出現を判別したい．
5. 画面上に入らないような巨大な項を解析したい．
6. 部分木を一つのノードとして見たい．

これらの要求に対し単なる木表現は十分とはいい難く，より高度な表現が必要となる．我々はこれを項の視覚化と呼ぶ．

これらの要求に対応した項の視覚化の例を図1に示す．この図では各ノードの図形は構成子記号は楕円，被定義関数記号は長方形，変数記号は円で表され，リデックスは太い線で描かれている．ソートは色により表すことができる．また，部分木は三角形で表されている．数字が書かれている小さな正方形は出現のマークを意味している．任意の出現に対し番号によるマークづけを行なうことで，巨大な項であっても注目している部分項を即座に発見することが可能になる．

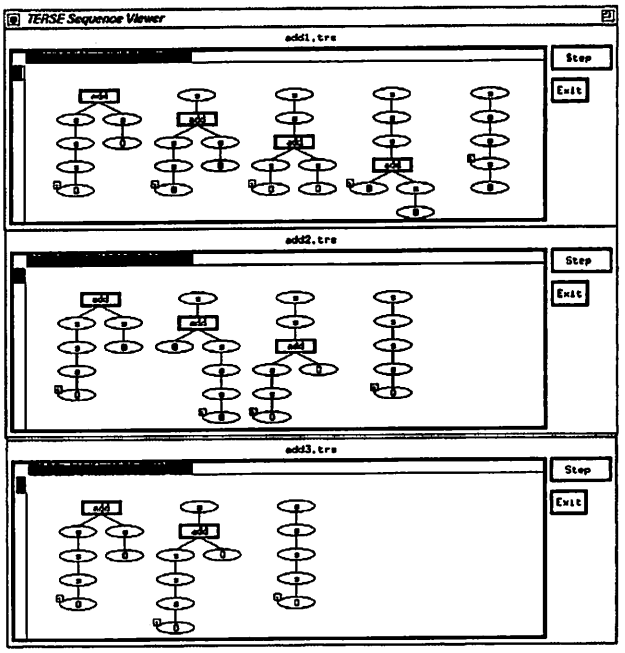

図2　計算の視覚化

様々な視覚的効果を目的とした項の視覚化を行なうことで，項から得られる情報が大きく増加し，従来のテキストベースの処理系では行なえないような直観的な解析が可能になる．

3.2　計算の視覚化

計算の視覚化とはすなわち，プログラム実行の視覚化である．プログラムの実行は次の2つからなると考えて良い．

- 各々の時点での実行状態．
- 状態間の遷移（制御）．

つまりプログラムの実行は，状態から状態への遷移の繰り返しで表すことができる．項書換え系における実行状態とは，書換え対象の項に他ならない．つまり，前節で述べた項の視覚化を実行状態の視覚化とみなすことができる．状態間の遷移は，項書換え系では書換え戦略を意味する．つまり書換え対象の部分項のうちどれを選ぶか，ということである．戦略には最内戦略，最外戦略など様々なものがあるので，

同じ初期状態からでも異なる複数の計算を行なうことができる.

計算の視覚化の一つの手法としては, 単一の戦略について各々の状態を順に並べて表示することである. 図2は同じ項に対して異なる規則を用いて計算を行なった場合の計算の視覚化を示している. これを**書換え系列の視覚化**と呼ぶ. この図を用いることで各々の規則の特徴をつかむことができる. 例えば, 上段の書換え系列は左の引数を順に減らしていくのに対し, 最下段の系列は右の引数を減らしている. 中央の系列では, 左右の引数が書換えごとに入れ換えられていることが, 出現のマークにより確認することができる. 出現のマークは出現の残余の概念により書換えの前後の項を伝搬する. このように, 書換え系列の視覚化により計算の実行状態がどのように変化するかを直観的に把握することができる.

一方, 同じ規則に基づき異なる書換え戦略を用いた複数の計算について, それらがどのような関係にあるかを表現する**書換え関係の視覚化**も考えられる. 各々の実行状態である項に対し, その状態間がどのような戦略で遷移するかを有向グラフで表すことで, 代数的仕様によって表された計算の意味を把握するとともに, どのような戦略を用いることで効率的な計算を行なうことができるかを直観的に確認することができる.

3.3 操作の視覚化

視覚化された項や計算に対し解析を行なう場合, それらに対する様々な操作もグラフィカルユーザインタフェースを用いて視覚的に行なえることが望ましい. 本節では, 各々の操作に対してどのような視覚的実現を行なうべきかについて考察を行なう.

まず, 項に対しては次のような操作が考えられる.

1. 書換え規則の選択.
2. 書換え戦略の選択.
3. 任意の書換え対象の項の選択.
4. 任意の出現のマーキング.
5. 部分木の表示に対する指示.
6. 項, 規則の編集.

1.,2. は項の計算に対しての属性を選択する操作であり, すでに与えられてる規則や戦略を選ぶ操作である. これらは, 選択枝のリストの中からポインタによって選択する操作で実現する. また, 項書換え系を

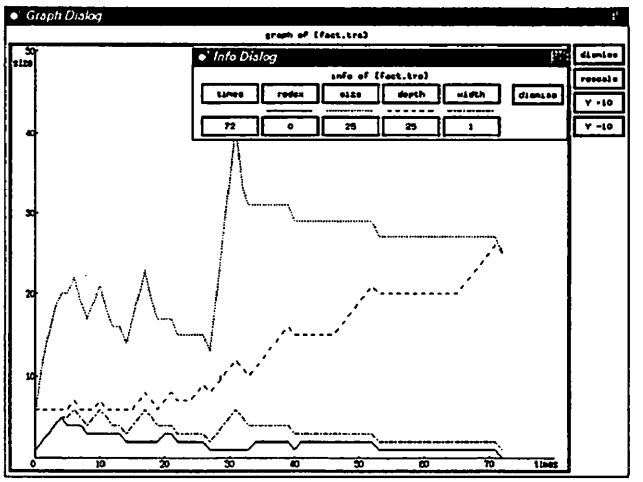

図3 情報の視覚化

ファイルから読み込むこと可能にするためにファイルの選択ダイアログが必要である. 3. は戦略で指定できないような書換え対象の項を指示する操作である. 4.,5. は項の視覚化に対しての指示であり, 4. は項の任意の出現について覚えのためのマークをつける操作, 5. は任意の部分項を部分木として表示することを指示する操作である. 3.,4.,5. の操作は視覚化された項の図形に対しポインタを用いて直接指示することで実現する. 6. は項や規則の入力や変更に対する支援を行なう操作であり, 視覚的構造エディタを用いて実現する. 関数記号の入力では, その引数に応じて自動的に引数のノードを出現させることや, 部分項のコピーなどを視覚的に行なう.

次に項書換え系やその計算に対する操作を挙げる.

1. 書換え規則の編集.
2. 書換えの実行.
3. 書換え系列の選択.
4. ブレークポイントの設定.

1. の操作は例えばある項書換え系を左辺の先頭に現れる関数記号ごとに分割するような操作や, 逆に分割された項書換え系を組み合わせる操作などを意味する. これらは規則のリストをポインタで選択することで行なう. また, 編集結果の項書換え系のファイルへの保存や読み込みが行なえることが必要であり, ファイルダイアログが必要である. 2. は1ステップもしくは数ステップの書換えの指示を行なう操作で, ボタンをポインタで押すような操作で実現する. 書換えの前後では出現のマークや部分木を出現の残余の概念を用いて移動させる. 3. は書換え系列上の任

意の状態へ現在の状態を遷移させる操作である．これにより，任意の時点での計算の再実行が可能になる．視覚化された書換え系列上でのポインタ指示で実現する．4. は項書換え系のデバッグを行なうための操作であり，任意の規則にブレークポイントを設定し，その規則が用いられるまで書換えを行なう．これは規則のリスト選択によって実現する．

3.4 情報の視覚化

項書換え系の計算の解析において，項の状態や計算の状況を数値で表すことはしばしば行なわれる．例えば項のサイズ，項の深さ，項の幅，リデックスの数やそれらの書換え全体に対する平均値，最大値，最小値などがあり，正規形を得るための書換え回数などは最も重要な値の一つである．これらの数値のグラフ化を行ない，直観的理解を助けることを情報の視覚化と呼ぶ．計算を実行した後にデータを収集するだけでなく，計算途中に動的にグラフ化することで，現在の状態の情報のみでなく，過去の状態との比較を直観的に行なうことができる．また，すでに定められている情報のみではなく，解析を行なうユーザが必要な値を随時計測できることが，より柔軟な解析を行なうためにも必要である．これを実現するために，計測値を得るためのプログラムを記述でき，随時支援システムに組み込めるような仕組みが必要となる．我々の支援システム TERSE はシステムの実行中にユーザが任意の計測関数を新しく追加する機能を持つ．

図3はある階乗の計算における情報の視覚化である．この図では計算が 72 回の書換えで終了し，この時点での項のサイズ及び深さが 25 であり，幅が 1 であることを示している．この視覚化された情報により，1 ステップづつの書換えにともない動的に項が変化していく様子を確認することができる．

4 視覚化による解析・検証・変換の支援

本節では視覚化を用いることでどのように代数的仕様の解析・検証・変換が行なえるかを具体的な例によって示す．

代数的仕様の例としてハノイの塔の解法を用いる．　この例では塔　A,B,C　に対し小さい順に $0, s(0), s(s(0))$ の 3 つの輪があるとして，これを A から C に動かす場合の解を求める．以下に仕様を示す．

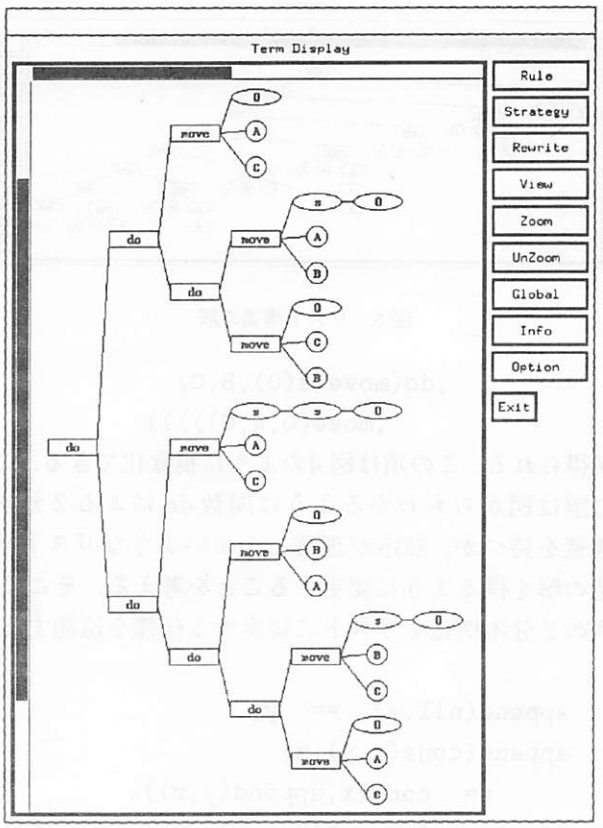

図4　構造を持つハノイの塔の解

```
var x,y,z,from,other,to;
1: hanoi(s(x),from,to,other)
      == do(hanoi(x,from,other,to)
           ,do(move(s(x),from,to)
                ,hanoi(x,other,to,from)));
2: hanoi(0,from,to,other)
                      == move(0,from,to);
end;
```

ここで，$hanoi(x, from, to, other)$ は「x 以下の輪を塔 $from$ から塔 to へ $other$ を使って移動させる」という意味の関数である．また $move(x, from, to)$ は「輪 x を塔 $from$ から塔 to へ移動させる」，$do(x, y)$ は「x を行なってから y を行なう」という意味である．

仕様を項書換え系とみなし $hanoi(s(s(0)), A, C, B)$ を入力として書換えると 7 回の書換えの後，正規形

```
do(do(move(0,A,C)
     ,do(move(s(0),A,B),move(0,C,B)))
  ,do(move(s(s(0)),A,C)
     ,do(move(0,B,A)
```

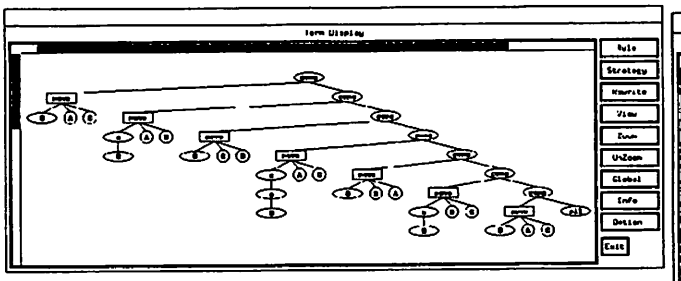

図5　リスト構造の解

```
,do(move(s(0),B,C)
    ,move(0,A,C)))))
```

が得られる．この項は図4のように視覚化できる．この解は図からもわかるように関数 do による2分木構造を持つが，順序が理解しやすいようなリスト構造の解を得るように変更することを考える．そこで，次の2分木構造をリストに変換する仕様を追加する．

```
3: append(nil,y)  ==  y;
4: append(cons(x,y),z)
       ==  cons(x,append(y,z));
5: list(move(x,y,z))
       ==  cons(move(x,y,z),nil);
6: list(do(x,y))
       ==  append(list(x),list(y));
```

この仕様を実行させると，確かに図5のようなリスト構造の解を得られることが確かめられる．このように視覚化された項により，リスト構造を得るための仕様に間違いがないことが直観的に検証できる．ところがこの計算は非常に効率が悪く，リスト化するための入力 $list(hanoi(s(s(0)),A,C,B))$ に対し34回もの書換えを必要とする．そこで，この効率を改善するために計算の解析を行なう．最初に書換え系列の一部を視覚化(図6)する．図を見ると，$list(hanoi(x\ ,f,t,o))$ という形，もしくは $list(do(hanoi(p),do(q)))$ の形の繰り返しがあることが直観的に発見できる．視覚化された書換え系列に対し解析を行なう場合，このような繰り返し構造を持つパターンに着目することが重要である．このように多く出現する関数の組合せは，組合せられた状態で何らかの意味を持つものと考えられる．そこで $list(hanoi(x,f,t,o))$ と等価な関数として新しい関数 $lhanoi(x,f,t,o)$ を導入する．

```
7: lhanoi(x,from,to,other)
     ==  list(hanoi(x,from,to,other))
```

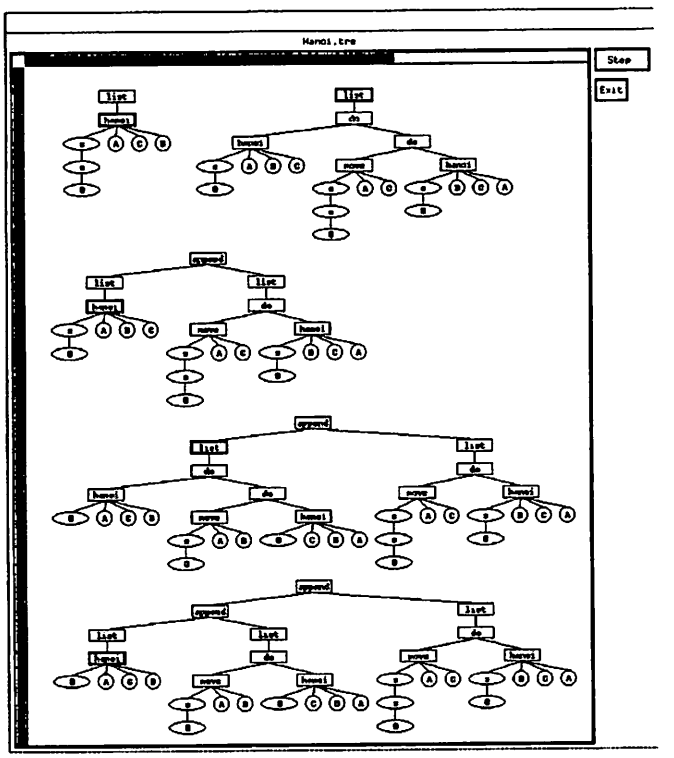

図6　ハノイの塔の計算の視覚化

7の等式と1,2の等式より次の等式を導くことができる．

```
8: lhanoi(s(x),from,to,other)
    == list(hanoi(s(x),from,to,other))
    == list(do(hanoi(x,from,other,to)
            ,do(move(s(x),from,to)
               ,hanoi(x,other,to,from))))
    == append(list(hanoi(x,from,other,to))
            ,list(do(move(s(x),from,to)
               ,hanoi(x,other,to,from))))
    == append(lhanoi(x,from,other,to)
        ,append(list(move(s(x),from,to))
            ,list(hanoi(x,other,to,from))))
    == append(lhanoi(x,from,other,to)
       ,append(cons(move(s(x),from,to),nil)
            ,lhanoi(x,other,to,from)))
    == append(lhanoi(x,from,other,to)
         ,cons(move(s(x),from,to)
    ,append(nil
             ,lhanoi(x,other,to,from)))
    == append(lhanoi(x,from,other,to)
         ,cons(move(s(x),from,to)
              ,lhanoi(x,other,to,from)));
```

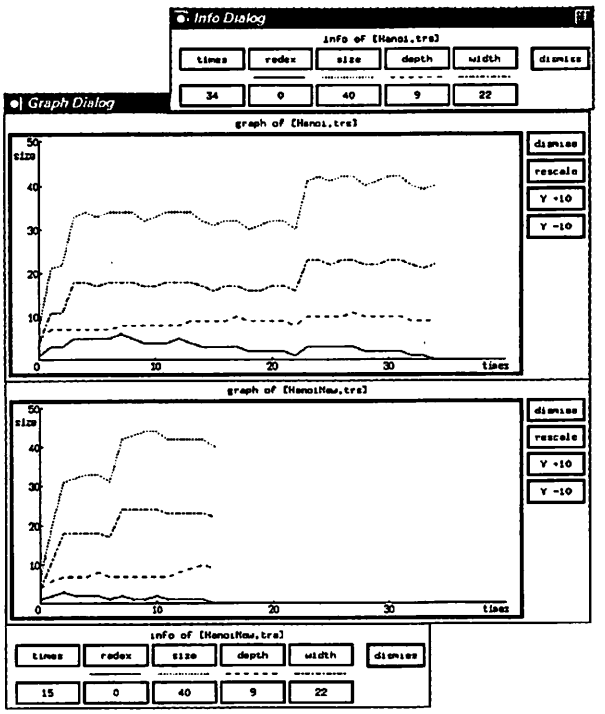

図 7　変換の前後の視覚化情報

```
9: lhanoi(0,from,to,other)
    ==  list(hanoi(0,from,to,other))
    ==  list(move(0,from,to))
    ==  cons(move(0,from,to),nil);
```

8,9 の等式を用いて書換えを行なうと，劇的に効率が向上する．入力 $lhanoi(s(s(0)), A, C, B)$ に対し，正規形を得るまでの書換え回数は当初が 34 回であるのに対し 15 回で書換えが終了し，図 5と同じ解を得ることができる．

この等式の変換の作業そのものは [2]で用いられている fold/unfold 変換と同一であるが，どの関数に着目して変換を行なうかを見つけることが従来の手法では困難であった．我々が提案する視覚化を用いた解析・検証手法を用いることで，直観的にどの関数に変換を適用すれば良いのかを見つけることができ，実際の例で確かめることができた．図 7に変換前と変換後での視覚化情報を示す．上のウィンドウが変換前を示し，下のウィンドウが変換後を示している．変換後の仕様の実行は変換前と比べ書換え回数，サイズ共に小さくなっていることを直観的に確かめることができる．また，これらの情報は書換えの途中で任意の時点で参照することができるため，解析の手助けとなる．

5　視覚的支援環境の実現

前節に挙げた例はすべて我々が実現した視覚的項書換え支援環境 TERSE によるものである．TERSE [16]は SML/NJ の上に実装された CML (Concurrent ML) [13]で書かれており，X ウィンドウシステムを使うために eXene ライブラリ [12]を利用している．ML は強力な型推論の機構とモジュール化の仕組を持つ関数型言語であり，優秀なコンパイラにより非常に効率良く動作する高級言語である．CML は並列化された ML であり，マルチスレッドで動作し，スレッド間ではチャネルを使った同期通信を行なうことが可能である．eXene は X ウィンドウシステムに対するインタフェースを CML 上に記述したライブラリであり，図形描画関数のみでなく，ボタンやリスト，メニューなどの基本的なウィンドウオブジェクトもサポートしている．

TERSE の実現においてはデータ構造の決定やモジュール分割の設計を慎重に行なったため，柔軟に拡張を行なうことができる．TERSE は大きく 3 つのパート（ユーザインタフェース部，項書換え系インタプリタ部，木表現部）から構成されている．それぞれはモジュールとして独立しており，独立に変更することが可能である．また，環境自身がコンパイラを含むため，動作中に動的にシステムを拡張することができる．この仕組みを用いてユーザが情報の視覚化の際に新しい計測関数を導入することが可能になっている．また，視覚化された項もウィンドウ上のみでなく，同じ描画関数を用いてポストスクリプトプリンタに出力することが可能である．TERSE では関数型言語による実現の利点を生かし，多くの部分で高度に多重化された実現を行なっている．

しかし高級言語であるためのオーバーヘッドは存在し，巨大な項に対する多数の書換えの表示はある程度の時間を必要とする．これは項の視覚的表示を書換えにより変更された最小限の部分にとどめるなどの努力により高速化をはかることなどで改善できるであろう．なお，実現については [17]に詳しく述べてあるのでそちらを参考にされたい．

6　他のシステムとの比較

代数的仕様や項書換え系に対する支援システムはすでにいくつか存在する．ReDuX [11]はテキストインタフェースによって実現された項書換え系のワー

クベンチである．ReDuX は様々なサブコンポーネントを持ち，Knuth-Bendix 完備化アルゴリズムや AC-matching, AC-unification, AC-completion などを実現している．MERILL [10] は等式を扱うシステムであり，TERSE と同様に SML により実現されている．順序づけされた項上での等式推論や，AC 演算子の利用，完備化アルゴリズムなど様々な解析をテキストレベルのコマンドを入力して行なうことができる．MERILL は ReDuX より若干洗練されたインタフェースを持つが，やはりテキストベースのシステムであり，視覚的な表示を行なう機能は持たない．これらのシステムでは代数的な解析を行なうことは可能であるが，本システムが可能としたような直観的発見による解析，効率の向上などを行なうことは不可能である．形式的な操作が困難な場合における本システムの優位性は高く，将来的にいっそう複雑化する仕様記述において，必須なシステムであると考えられる．

7 まとめ

代数的仕様の解析・検証・変換のための視覚的支援の手法を項，計算，操作，情報のそれぞれの視覚化を用いて提案した．これらは関数型言語の特徴を生かしたものであり，プログラムに対する直観的な理解を深めることができる．この手法に基づき，我々は視覚的項書換え支援環境 TERSE を並列関数型言語 CML を用いて実現した．また，実際に解析・検証・変換の例を示し視覚的支援環境の有効性を確かめた．項や計算の視覚化により，従来では直観的に発見することが困難な変換の着眼点を見つけたり，効率の問題点を指摘することができる．また，情報の視覚化により計算状態（項の構造）とその具体的な数値や数値変化の情報を同時にまた動的に得ることができる．操作の視覚化により，思考を妨げない直観的な操作が可能となる．また CML を用いた実現が十分実用的な速度で動作することは，関数型言語がユーザインタフェースの実現に応用可能であることを示している．

今後の課題としては，現在木表現で行なっている視覚化をグラフ表現に拡張することや，複数の書換え規則の選択を可能にすること，現在まだ実現できていない項の構造エディタの作成，書換え関係の視覚化の実現，デバッガの作成などが挙げられる．

謝辞

日頃御指導いただく平田富夫教授，御討論下さった山本晋一郎助手，直井徹岐阜大学助教授，外山勝彦中京大学助教授，酒井正彦北陸先端大助教授，並びに研究室の皆様に感謝します．

参考文献

[1] Knuth,D.E.,Bendix,P.B.: "Simple Word Plobrems in Universal Algebras" , Computational Probrems In Abstract Algebra , J.Leech, ed., Pergamon Press, NewYork, pp.263-297(1970).

[2] Burstall, R.M., Darlington, J. :"A Transformation System for Developing Recursive Programs", J.ACM, Vol.24, No.1, pp44-67(1977).

[3] Huet, G.: "Confluent Reductions:Abstract Properties and Applications to Term Rewriting Systems", J.ACM, Vol.27, No.4, pp797-821 (1980).

[4] 稲垣, 坂部: 抽象データタイプの代数的仕様記述法の基礎 (1)-(4) , 情報処理, Vol.25, No.1, No.5, No.7, No.9,(1982).

[5] 山本, 直井, 坂部, 稲垣: "項書換えシステムにおける必須な書換えと戦略の効率", 信学技法, COMP 87-37 (1987).

[6] 酒井, 坂部, 稲垣: "代数的仕様の検証のための被覆集合帰納法" , 信学技法, COMP90-5(1990).

[7] 布川, 富樫, 野口: "図式をシンタックスに持つ関数型言語", コンピュータソフトウェア, Vol.6, No.2, pp11-23 (1989).

[8] 田中, 太田: "GHC プログラムの視覚的入力システム：FE'92" , 信学技法, COMP90-67 (1990).

[9] 岩永, 大森: "動的振舞いを用いたソフトウェア性能分析支援システムの構築" , 情報処理学会研究会資料, SE-93-6(1993).

[10] Matthews,B.:"MERILL: An Equational Reasoning System in Standard ML" , Rewriting Techniques and Applications, LNCS 690,pp441-445(1993).

[11] Bundgen,R.:"Reduce the Redex → ReDuX" , Rewriting Techniques and Applications, LNCS 690,pp446-450(1993).

[12] Reppy,J.H.,Gansner,E.R.: "The eXene Library Manual(Version 0.4)" ,AT&T Bell Laboratories (1993).

[13] Reppy,J.H.: "CML: A higer-order concurrent language" ,Prceedings of SIGPLAN'91, pp293-305 (1991).

[14] 河口, 坂部, 稲垣: "可換則に基づく項書換え系の変換と必須呼びによる効率の評価" , 信学技法, COMP93-64(1993).

[15] 河口, 坂部, 稲垣: "グラフィカルユーザーインタフェースを持つ項書換え系の解析・変換支援環境" , 信学技法, SS93-44(1994).

[16] N. Kawaguchi, T. Sakabe, Y. Inagaki :"TERSE: TErm Rewriting Support Environment" ,Proceedings of the 1994 ACM SIGPLAN Workshop on Standard ML and its Applications , pp91-100 (1994).

[17] 河口, 坂部, 稲垣:" 視覚的項書換え環境の SML による実現" , ソフトウェア科学会第 1 1 回大会論文集, pp285-288 (1994).