Drag and Drop 手法を用いた 代数的仕様記述言語における視覚的プログラミング環境

Drag-and-Drop based

Visual Programming Environment for Algebraic Specification Language

小川 徹[†] 田中 二郎[††]
Tohru OGAWA Jiro TANAKA

†筑波大学 博士課程 工学研究科
Doctoral Program in Engineering, University of Tsukuba
††筑波大学 電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

概 要

代数的仕様記述言語をビジュアルプログラミング言語の対象とした場合、項書換えによる実行の視覚化ばかりではなくプログラム編集系の視覚化も考えるのが自然である。我々は、より直感的な編集操作を行うためには、視覚化された言語の操作においても言語の特徴を考慮すること、また、単純な操作のみに統一することが重要であると考えた。これに基づき、 Drag and Drop 手法を用いた視覚的プログラミング環境を試作した。

1 はじめに

代数的仕様記述言語 (ASL) はソート、演算、等式のような基本要素を使って現実世界を表現する仕様記述言語である。 ASL が項書換え系とすることで実行可能言語と言えることから、 C や Pascal 等と同じプログラム言語と見なすことができる。

視覚的プログラミングシステム (VPS) の分野において、様々な研究がなされている [1]。 VPS ではプログラムを図形などの視覚的表現によって表現することで、ユーザが実際のプログラムを直感的に理解しやすいように工夫している。プログラムのデバッグ等の過程において、ユーザはプログラムの一部に変更を加え実行するといった作業を繰り返すことがよくある。しかし、既存の VPS にはプログラム編集と実行部分を分けて考えているものが多く、変更個所を実行して確認するのに手間がかかる。従って、プログラムの実行を行いながらプログラムを編集する等が容易な VPS が望まれている。

我々は代数的仕様記述言語 CafeOBJ [2]のための視覚的プログラミング環境 **CafePie** を開発した。CafeOBJ の仕様記述はモジュール構造の集まりである。 CafePie ではこのモジュールを視覚化し、編集し、実行できる環境を提供する。プログラムの編集にはマウスを使った直接操作を用い、プログラムの実行における表現はプログラム構造と同じものを使用する。プログラムの編集と実行を１つのウィンドウに表示することで、プログラムの変更を実行に直接的に反映させることが容易になる。

2 プログラム構造の視覚化

プログラムの視覚化とは、プログラム構造を図形などの視覚的表現を使用して表現することであり、PP [3]等のプログラムの視覚化システムがある。

我々は CafeOBJ のプログラム構造を視覚化するために、プログラムの基本要素を図形で表すことを考えた。この図形をアイコンと呼ぶ。 CafeOBJ の

基本要素には、ソート、演算、変数、等式がある。これらの図形は以下のように視覚化される。

CafeOBJ におけるソートは順序ソートであり、各ソート間には下位 - 上位という関係がある。これをソートを頂点、関係を有効線とした有効グラフで表す。ソートはラベル付きの緑の矩形とし、下位ソートから上位ソートへ有効線を引く。

演算や変数を考える前に、項の視覚化を考える。我々は項を表現するために一般に良く用いられている木構造を用いる。即ち、演算や変数は頂点、それら要素間の下位 - 上位関係は下位から上位への有効線によって表現される。演算や変数はソートと区別するために楕円で表す。項の視覚化規則に合わせるため、演算の引数は楕円下方に返却値は上方に配置される。

等式は項書換えの規則を表す。CafeOBJ において、等式は左辺と右辺の項、等式ラベルから成る。等式ラベルを中央に置き、その左下には左辺の項を、その右辺には右辺の項を配置する。左辺から右辺に書換えるということを明確にするため左辺から (ラベルを通して) 右辺へと有効線を引く。

CafeOBJ において、プログラムはモジュールの集まりである。モジュールはモジュール名と基本要素の組合わせからなる。モジュールは基本要素であるソート、演算、変数、等式を含む。モジュールは灰色の矩形で表し、その中にラベルとこれらの要素を表現する。ユーザはこのモジュールアイコン上でプログラムの定義を行うことができる。

3 プログラム編集操作

我々はプログラムの編集を行う際、直接的な操作、数少ない単純な操作のみで可能にするような方法を考えた。直接操作は見えてる操作が具体的で見ているものが実際の操作となるから、現在自分が行っている操作がすぐに認識できる。また操作を単純にすることで、習得を容易にし、操作におけるユーザの負担を軽減することができる。

編集の操作にはマウスによってアイコンを動かすことが基本になる。アイコンを移動する際、移動先に他のアイコンが無い場合はアイコンの移動と定義する。もし、移動先に他のアイコンがある場合には、２つのアイコンは重合わせられることになる。

２つのアイコンを重合わせる場合の操作手法として Drag-and-Drop 手法がある。Drag-and-Drop 手法は、アイコンを選択しながら移動して (Drag)、目的の地点までマウスボタンを離す (Drop)、という操作である。この操作は Drag 動作によってマウスカーソルに追従するようにマウスの移動を行うため、対照アイコンが何であるかが一目で認識できる。また、Drag と Drap とは合わせて 1 つの操作であるため２つの動作間に時間の空きは生じない。

我々は、この良く知られた Drag-and-Drop 手法を再検討し、プログラムの編集作業をこの手法を用いることにした。

あるアイコン (A) を他のアイコン (B) に重合わせるとき、アイコン (A) を Source、アイコン (B) を Destination と呼ぶことにする。例えば、演算に引数を１つ加える編集作業は、引数がソート名から与えられることから、Source をソート、Destination を演算のように割当てることで定義できる。プログラムの編集は、このような作業の繰返しによって行うことが可能である。

4 CafePie システム

プログラム構造は２章で示したアイコンの組合わせによって記述される。それゆえ、プログラムはアイコン操作で編集される。

CafePie の実行画面を図 1に示す。画面中央がモジュールを作成する部分である。ここには例としてモジュール SIMPLE-NAT [2]が定義されている。ソート (Nat,Zero,NzNat) とその下側に演算 (0,s,_+_) があり、中央に変数 (N,M)、その右側に書換え規則を示す２つの等式 (0 + N:Nat = N:Nat, N:Nat + s(M:Nat) = s(N:Nat + M:Nat)) がある。

ユーザは既に作成されているモジュールの中身を３章で用いた操作を用いて編集していく。モジュール構造を記述する場合には、まず、ソートを記述し演算と変数とをソートを元に作成し、最後に等式を作成する流れになる。編集作業に用いる操作には、ソート間の関係作成と削除、演算への引数の追加、項の変数部分への演算の追加、項の変数部分への項の置換等がある (表 1)。

項を編集する場合には、項の作成、項の置換という操作を用いて行う (表 1)。変数部分に演算を追加した場合には、項の変数部分は演算に置換わり、さらに、演算の引数にあたる部分に新たな変数部分が

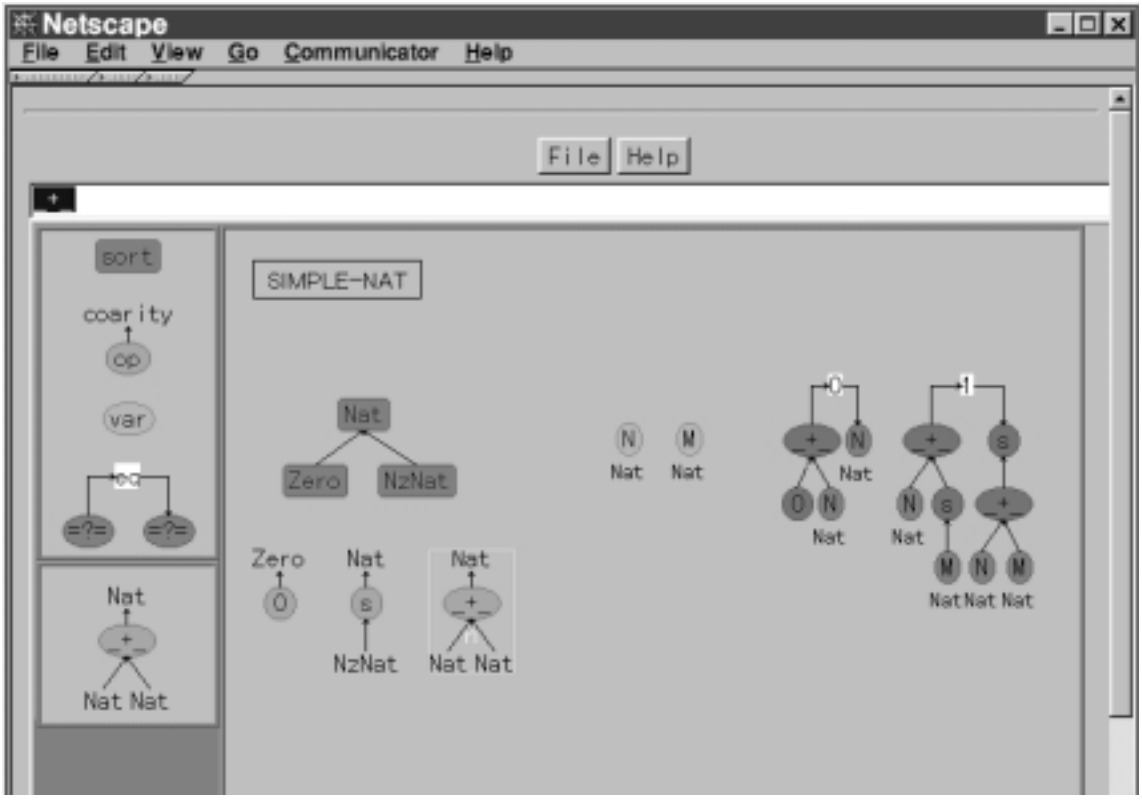

図 1　CafePie システムの画面

Event	Source	Destination	Action
ソート関係	ソート	ソート	関係作成、削除
引数追加	ソート	演算	演算に引数追加
引数交換	引数	引数	同演算上の引数位置交換
演算型決定	ソート	返却値	演算の返却値を決定
項の作成	演算	変数	演算に置換、部分項の作成
項の置換	項	変数	項に置換
変数型決定	ソート	変数	変数のソートを決定
新規作成	新規 Icon	Module 内	新規 Icon の作成
削除	Icon	Multi 内	Icon の削除

表 1　Drag-and-Drop に対応付けた編集

自動的に作成される。この変数部分に同じような作業を繰返すことにより再帰的に項を作成することができる。変数部分に項を追加した場合は、変数部分がそのままその項に置換わるだけである。例えば、等式 1 の左辺 (N:Nat + s(M:Nat)) は図 2に示すように作成されていく。

プログラム実行の視覚化: CafeOBJ におけるプログラムの実行は、項書換えによって行う。プログラム実行の視覚化とは項書換えの過程を視覚化することに他ならない。CafePie において、項書換えの処理

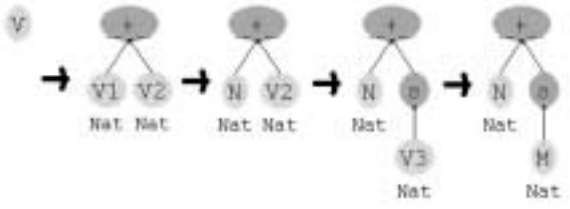

図 2　項の作成手順

は他のシステム (CafeOBJ インタプリタ) が行う。CafePie システムは、編集したモジュールと書換えするための (入力) 項をネットワークを利用してこのインタプリタに渡し、その結果を受取り次第、項書換えの過程を表示する処理を行う。従って、ユーザが実行を行うために行う作業は、モジュールを用意し、入力項を作成し、インタプリタにそれらを受渡すことになる。

入力項は Module フィールド内にソートや演算などの要素と同じように記述することができる。インタプリタへの受渡しは、入力項をモジュールのラベル上に Drag-and-Drop することで行う。入力項が正しく解釈されれば、インタプリタは項書換えの結果を返す。この結果は、書換え前と書換え後などの

情報からなる簡約の組から構成される。

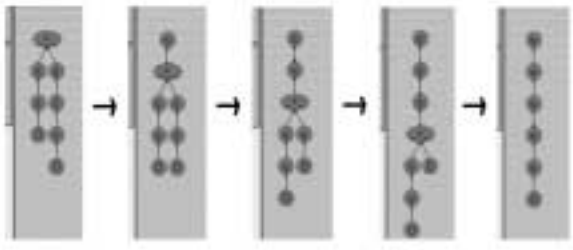

図 3　動的な実行の視覚化

CafePie はインタプリタから実行結果を受取ると、その簡約手順に従って入力項を動的に変化させていく (図 3)。

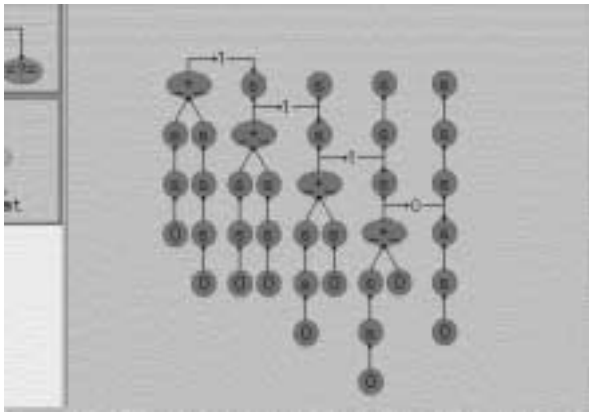

図 4　静的な実行の視覚化

簡約が終了すると入力項があった場所には簡約結果の項が表示される。その表示が終わると結果の項は左に移動し今までの書換え過程が静的に表示される (図 4)。マウスで静的表示をクリックすることで動的に切り替えることも可能である。書換え過程の動的表示は大まかな書換えの流れの調査するのに適しており、逆に静的表示は１つ１つの簡約を正確に判断できる。例えば途中に予期せぬ書換えが行われていた場合、プログラムのどこを書き直せば良いか判断する必要があったとする。マウスで等式を選択する度にその書換えで用いられた等式が何であるかという情報を与えることで、ユーザは直ちにその等式の修正に取り掛かることができる。しかし、書換えの回数増加やより複雑な場合には、修正個所を静的な表示から見つけ出すのは困難なことが予想される。こういう場合には書換え過程を順次に動的に表示することで、大まかな流れを見て、特定を早めることが可能である。

5　他システムとの比較

代数的仕様や項書換え系に対する支援するシステムは既に幾つか存在する。ReDuX [4]はテキストインター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 は様々なインターフェースを持ち Knuth-Bendix 完備化アルゴリズム等を実現している。インターフェースに工夫が見られるが直感的解析は不可能である。TERSE [5]は視覚的に実現された項書換えのための支援環境である。ユーザインターフェースや項書換え系の表示などを用いて直感的発見による解析や効率の向上など工夫し項書換えの検証システムを実現している。しかし、実際の検証においてはもとのプログラムとの比較が必要な場合が良くある。操作を簡略化し、プログラム編集と実行表示とを結び付けた本システムは有用性は高い。

6　おわりに

我々は CafeOBJ のための視覚的プログラミング環境 CafePie を作成した。プログラム構造はアイコンによって視覚的に表現され、Drag-and-Drop 手法を用いて直感的に操作可能である。項書換えによるプログラムの実行においても同じアイコンを用いて視覚化される。

代数仕様記述言語において、プログラムの検証は重要なテーマである。このためにプログラムの実行過程の視覚化のために、省略化等のより高度な視覚化方法についての研究が必要である。

参考文献

[1] B. A. Myers. Taxonomies of Visual Programming and Programming Visualization. *Journal of Visual Languages and Computing*, 1(1):97–123, 1990.

[2] Ataru T. Nakagawa, Toshimi Sawada, and Kokichi Futatsugi. *CafeOBJ User's Manual*. IPA, 1997.

[3] Jiro Tanaka. PP : Visual Programming System for Parallel Logic Programming Language GHC. *Parallel and Distributied Computing and Networks '97*, pages 188–193, August 11-13 1997. Singapore.

[4] R. Bundgen. Reduce the Redex → ReDuX. In *Rewriting Techniques and Application*, LNCS 690, pages 446–450. 1993.

[5] Nobuo Kawaguchi, Toshiki Sakabe, and Yasuyoshi Inagaki. TERSE: Term Rewriting Support Environment. In *Workshop on ML and its Application*, pages 91–100, florida, june 1994. ACM SIGPLAN.