解　説

Gandalf システムにおける構文向きエディタ ALOE†

斎　藤　信　男††

1. はじめに

総合的なソフトウェア開発環境として開発されているGandalfシステムは，カーネギメロン大学計算機科学科の一つのプロジェクトとして推進されている[1]．これは，システムバージョン制御，個々のプログラムの構築，およびプロジェクト管理の三つの要素から成り，多数のプログラマ／システムエンジニアが参加する大規模ソフトウェアの開発を支援する環境を提供することを目指している．Gandalfシステムが，ユーザのサイトに応じたソフトウェア開発環境（Gandalf環境）を生成できれば，理想的な体系となる．現在，Gandalf C（C言語を少し仕様変更したもの）に対する環境が開発されている．これは，Vax Unix (4.1 BSD) 上で稼動する．さらに，カーネギメロン大学では計算機の入門教育用に Pascal を使用しているので，特に構造エディタの評価などを行うために Pascal 用の環境の構築を目指している．最終的には，米国防省の推進する Ada に対する開発環境の実現もねらっている．

Gandalfシステムの一つの要素である個々のプログラムの構築においては，構造エディタを利用することを初めから考えていた．これは，エディタに対する新しい考え方[2]を実験し，それを評価したいためである．エディタは，従来文字列の入力編集を目的としていたが，その文字列が文章であろうとプログラムのソースコードであろうと同一のエディタを使ってきた．これに対し，入力編集する対象がある特定の言語であるという情報を使えば，能率の良いエディタが実現できる可能性がある．コーネル大学の Cornell Program Synthesizer[3]，INRIA の Mentor[4] などは，その種の構造エディタの例であり，Gandalfシステムでもその種のものを採用している．一般に，構造エディタの中でも，構文向きエディタ (syntax directed editor) と呼ぶ型のものは，正しい構文の文字列だけを入力編集できるエディタであり，Gandalfシステムにおけるものもこの型に属する．構文向きエディタは，また，構文木を編集するシステムと考えることができる．言語の仕様に合致する構文木だけを許すことにすれば，正しい構文を持つプログラムだけが入力編集される．

Gandalfシステムのバージョン制御においては，ソフトウェアの部品としてモジュールという単位を設定する．各モジュールは，論理仕様を持つ．これに対し，それを実現する方式を一つ決めるとバージョンが設定される．実現方式が異なれば，異なったバージョンができるが，これを並列バージョン (parallel version) ともいう．一方，一つのバージョンに対し具体的にプログラムを作成してゆくとき，誤りの検出により修正を加えてゆくので，時間的に並べた逐次バージョン (serial version) ができる．これを，リビジョン (revision) という．一つのモジュールに対し一つのリビジョンが選択され，ソフトウェアの部品として使われる．バージョンおよびリビジョンを識別子をつけてすべて保存しておけば，任意の時点でのソフトウェアシステムが再構築できる．

このバージョン制御においては，バージョンやリビジョンの相互関係はすべて木で表わすことができる．したがって，バージョンやリビジョンの構築は，結局木構造を作成することになる．そこで，Gandalfでは，システムバージョン制御のユーザとのインタフェースとして構文向きエディタを使っている．ユーザから見ると，ソースプログラムの構築も同種の構文向きエディタを使うので，統一的なインタフェースを持つことができ，システム全体が使い易くなる．

以下には，構文向きエディタ ALOE について，その仕様，使い方，実現法などを述べ，また，実際の使用上の問題点などについて検討する．

† Syntax-Oriented Editor ALOE in Gandalf System by Nobuo SAITO (Department of Mathematics, Faculty of Science and Technology, Keio University).

†† 慶応義塾大学理工学部数理科学科

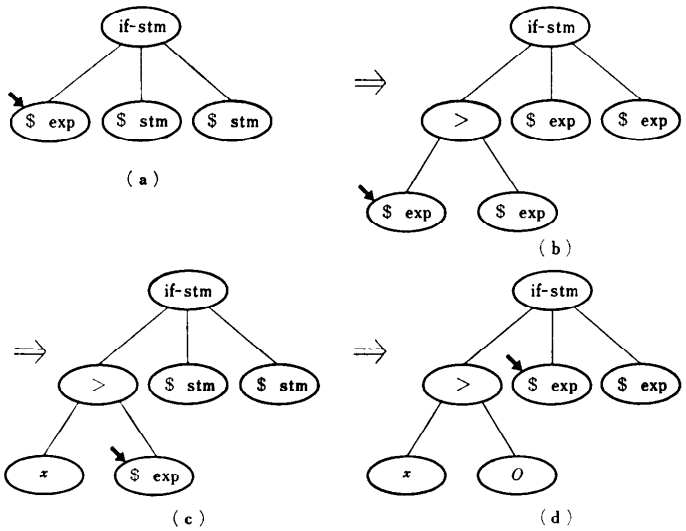

図-1 構文木の構築

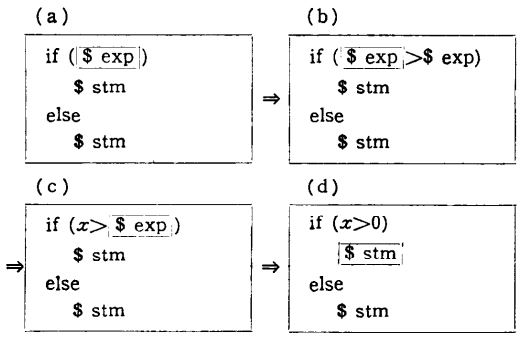

図-2 画面上の表示形式

2. ALOE コマンドとその機能

2.1 ALOE によるプログラムの編集手順

ALOE は，編集の対象となるプログラムの言語に依存し，ある特定の言語に対してその正しい構文の入力編集が可能である．ユーザは，たとえば手続きのプログラムを作成するとき，手続きの構文の最上位の要素から始める．各構文要素に対しては，その定義を示すテンプレートが与えてある．いま，if 文についての例を考えてみよう．図-1 に示すように，画面に表示されたテンプレートの一つの構文要素（非終端要素であり記号$で始まる名前をもったもの）にカーソルが位置している．カーソルの存在は，その要素がハイライト化していることからわかる．ユーザは，カーソルの指している要素を，一つ下の要素に展開することを指示するが，展開できる要素は選択子として画面の右端にある支援窓に表示することができる．図に示すように，式 ($ exp) として比較演算子“>”を使った2項式を先ず選ぶと，$ exp が部分木と置き換わり “>” の左側の $ exp にカーソルが移る．この部分に対して変数名“x”を選択すると，これは終端要素なので以後の展開はせず，カーソルは演算子“>” の右側の $ exp に移る．この部分に対して整数型定数“0”を選択すると，これはやはり終端要素でありこれ以上の展開はしない．ここで，最初の $ exp の展開がすべて終了し，カーソルは $ stat に移る．以上の構文木の変化に対応する画面表示を図-2 に示す．

テンプレートは，対象の言語の文法にそって作られているから，作成する構文木は文法的に正しく，得られる文字列も文法的に正しいものである．エディタとしては，文字列を作るというよりも，構文木を組み立てるシステムと考えた方がよい．

2.2 ALOE のコマンド

ALOE は，構文木を作成するエディタと考えると，そのコマンドも理解し易くなる．コマンドの入力は，コマンド窓にコマンド名の文字列を打ち込むことによって行うが，ユーザの打鍵の回数を減らすために emacs エディタで採用しているように特殊なキーから入力できるようにしている．たとえば，コントロールキーと英数字キーの同時打鍵を使うが，これは，英数字キーに↑をつけて表わす（↑a など）．

以下に，ALOE コマンドの概要を示す．

（1） カーソルの操作コマンド

カーソルは，構文木のどれかの節点に位置しており，これを移動させるコマンドがある．現在のカーソルの位置している節点に対して，移動は相対的に行う．

① cursor-in（下向き矢印キー）
現在の節点の子供の節点の最初にカーソルを移す．

② cursor-out（上向き矢印キー）
現在の節点の親の節点にカーソルを移す．

③ cursor-next（右向き矢印キー）
現在の節点の兄弟の節点の中で次のものにカーソルを移す．

④ cursor-previous（左向き矢印キー）

現在の節点の兄弟の節点の中で前のものにカーソルを移す.

⑤ cursor-home (ホームキー)

現在の画面に表示されている木の根の節点にカーソルを移す.

⑥ find (↑f)

現在画面に表示されている木の中で,変数名,演算子名などを文字列として前方向に捜し,カーソルを移す.

⑦ reverse-find (↑xb)

後方向に変数名などを文字列として捜す.

カーソルの移動した節点に対しては,それを根とする部分木全体がハイライト化し,どこにカーソルが位置しているかユーザが理解できるようにする.たとえば,cursor-home コマンドを実行すると,画面に表示してある文字列がすべてハイライト化する.

(2) 木操作のコマンド

構文木の作成編集に対して,部分木の移動,木の構造の変形などのコマンドがある.

① clip-subtree (↑k)

現在の部分木に名前をつけて,クリップ領域に保存する.

② insert-subtree (↑x↑i)

指定した部分木をクリップ領域から見つけ,現在の節点の位置に挿入する.

③ movefrom (↑xf)

現在の部分木を画面の木から取り除き,移動領域に保存する.

④ moveto (↑xt)

移動領域にある部分木を,現在の節点の位置に挿入する.

⑤ extend-list (↑e)

木の節点には,子供の節点が固定数のものと,可変数のもの(リスト型)とがある.現在の節点がリスト型であれば,子供の先頭に一つ節点をつけ加え,リストのメンバであれば自分の直後に一つ節点をつけ加える.

⑥ prepend-to-list (↑x↑a)

現在の節点がリストのメンバであれば,リストの先頭に新しい節点を一つつけ加える.

⑦ append-to-list (↑x↑e)

現在の節点がリストのメンバであれば,リストの末尾に新しい節点を一つつけ加える.

⑧ delete (↑d)

現在の部分木を消去する.

⑨ replace (↑r)

現在の部分木を消去し,メタ節点($で始まる要素に対応する節点)を挿入する.

(3) 入出力操作などのコマンド

上記のほかに,構文木のファイルへの読み書き操作などのコマンドが用意してある.

① read-program (↑x↑r)

ファイルから構文木形式のプログラムを読み込む.

② write-tree (↑x↑w)

構文木形式のプログラムを,ファイルへ書き出す.

③ unparse-into-file (↑wu)

構文木形式のプログラムを,文字列形式に変換してファイルへ書き出す.

④ set-mode (↑x↑m)

支援窓の情報を表示するかどうかを決める novice モードなどの値を設定する.

2.3 構文木の内部表現

ALOE で用いる木構造には,次の二種類の節点がある(図-3).

(1) 一定の数の子供をもつ節点

一定の数の要素に分解される非終端要素を表わす節点であり,その子供の節点の相対位置によりどんな要素であるかが決まる.

(2) 任意の数の子供をもつ節点

文の列やパラメータの列などのように,任意個の要素から構成される非終端要素を表わす節点である.

上記の二つの型を,言語の文法に応じて使い分ける.これについては,ALOE 生成系で詳述する.

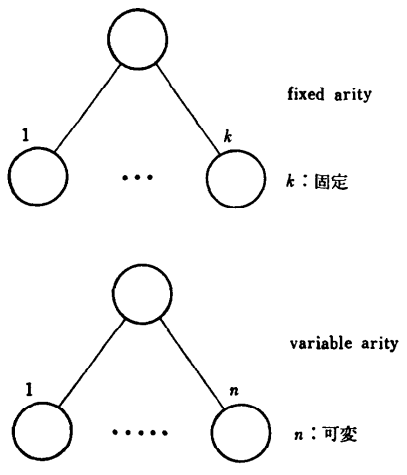

図-3 木の種類

2.4 字句要素の入力

終端要素は，何らかの字句要素になるので，それに対する入力が容易にできなければならない．標準字句要素の入力ルーチンとして，整数，実数，文字，文字列，論理値，変数名などがある．字句要素の入力に際しては，その型を入力してから値を入力しなければならないが，ユーザの打鍵の回数がふえてしまう．変数名の入力が多くなるので，ある構文要素に対しては，入力した文字列を変数名として受け取ることを標準にできるような工夫もしている．

標準の字句要素入力ルーチンがユーザの要求に合わない場合，たとえば特殊記号の扱いが異なる場合など，ユーザの作成した字句要素入力ルーチンを組み込むことも可能である．その際のシステムとのインタフェースは，公開してある．

3. 画面の制御

ALOE は，標準の CRT 端末を入出力装置として使う．構文木を扱っていること，ユーザにいくつかの情報を示しながら会話をスムーズに行うことなどの要求を満たすために，端末の画面の表示の仕方を工夫している．

構文木を CRT 端末を使って直接見せることは非能率的である．また，システムで作成する構文木は，いわゆる抽象構文木でよく，予約語などは除いてあっても構わない．そこで，抽象構文木に予約語を追加したり刻みをつけ加えたりして画面上で見易くする．この操作を，アンパース (unparse) という．木構造を画面に表示する形式は，後述する ALOE 生成系において指示することができる．

ALOE が画面に表示すべき情報には，構文木をアンパースした文字列，入力すべきコマンドやそのパラメータ，クリップしてきた木構造，状態，ユーザに示す選択子などがある．これらの情報のいくつかは，同時に画面上で見れると都合が良い．そこで，CRT 端末の画面を複数個の窓に分割し，それぞれに特定の情報を表示する (図-4)．窓として，構文木窓，状態窓，コマンド窓，支援窓，クリップ窓がある．また，これらの窓を操作するために，窓の上下のスクロール，左右のスクロール，窓の選択，などのコマンドが用意してある．

4. ALOE の生成

ALOE は，特定の言語ごとにその文法に従って機能しなければならない．これを多様な言語の種類に対して能率良く作成するために，ALOE を生成する系を実現している．このエディタ生成系を，ALOEGEN と呼ぶ．ALOEGEN に対して，一つの言語Lの文法，画面の表示の仕方，意味処理などを記述し入力すると，図-5 に示すように言語Lの ALOE エディタが実現する．

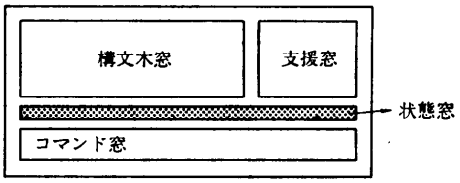

図-4 画面のいろいろな窓

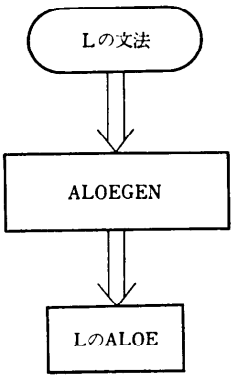

図-5 ALOEGEN の役割

ALOEGEN への入力情報は，次の三つの部分から成る．

(1) 終端要素に対する記述

終端要素がどんな字句要素として入力されるべきか，どのように画面に表示されるか，同義語は何かなどを与える．

(2) 非終端要素に対する記述

非終端要素がどんな要素（終端および非終端）に分解されるか，それをどのように画面に表示するか，同義語は何か，意味処理はどれを呼び出すかなどを与える．

(3) クラスに対する記述

一つの非終端要素に対して，分解される方式が複数個あるとき，これを同一のクラスとして扱う．ここでは，一つのクラスがどのような非終端または終端要素から構成されるのかを与える．

終端要素および非終端要素の記述は，次のようなも

のから成る.

（i） アンパースの方式

画面に木構造を表示すること，すなわち，アンパースの仕方を指定する．木構造らしく見せるための段づけの仕方，構文木では省略してある予約語の挿入などが，その指定項目になる.

（ii） 同義語の指定

終端および非終端要素の名前は，通常，文字列で与えられる．ユーザがそれらを指定する際に，完全な文字列の名前を打鍵するのは，少々面倒である．そこで，一文字の特殊記号などを使って，打鍵の回数が減少するような名前を同義語として定義することができる.

（iii） 優先順位

非終端要素に対して，それをアンパースするときに自動的にかっこをつけるために，優先順位を指定することができる.

（iv） アクションルーチン

各要素を作成したり消滅させたりする時に，意味の処理をするアクションルーチンを呼び出すことを指定できる．アクションルーチンは，ユーザがあらかじめ用意しなければならないが，これにより，宣言との一貫性のチェックなどが行える.

（v） 字句読み込みルーチン

終端要素に対しては，字句読み込みのルーチンを指定する．標準的に用意されている字句読み込みルーチンは，変数名，整数，実数，文字，文字列などが扱える．ユーザが用意したルーチンを指定することもできる.

ある言語における if 文に関連する定義の例を，以下に示す.

```
/* terminal */
{INT={i}|action: a INT|synonym: "i"|
      "@C";
VAR={v}|action: <none>|synonym: "v"|
      "@S";
⋮
}
/* non-terminal */
{PLUS=exp exp|action: <none>|
       synonym: "+"|precedence: 1|
       "@1+@2";
GRT=exp exp|action: <none>|
      synonym: ">"|"@1>@2";
IFSTM=exp stm stm|action: <none>|
      synonym: "if"|
      "if (@1) @+@n then @+@n @2
      @-@n else @+@n @3 @-@-";
⋮
}
/* class */
{exp=INT VAR PLUS TIMES GRT LSS;
stm=ASNSTM IFSTM WHILESTM
      FORSTM;
⋮
}
```

```
if ( $ exp )
  then
    $ stm
  else
    $ stm
```

図-6 IFSTM の表示形式

上記の terminal の集合において，{i},{v} はそれぞれ整数，変数のクラスを表わす．また，@C, @S は，それぞれ定数，記号表の中の記号を画面上に表示するアンパース方式を示す.

上記の non-terminal の集合に属する IFSTM は，exp, stm, stm の三つの要素から成る部分木に展開される．このとき，端末画面上の表示は，図-6 のようになる．ここで，@1, @2, @3 は，それぞれ exp, stm, stm を表わす．また，@+ は刻みのレベルを一つ上げること，@− は一つ下げること，@n は改行することを示す．if, ( ), then, else の予約語は，表示のときだけつけ加えることがわかる.

5. ALOE の使用とその評価

ALOE を具体的に使用をし，そこから得られた評価は，次の通りである.

（1） Gandalf システムのユーザインタフェースとして効率よく働く.

Gandalf システムにおけるシステムバージョン制御の機能におけるユーザインタフェース（コマンドなど）として ALOE のコマンドや考え方を利用している．ソフトウェアを構築するための部品が木構造として組み立てられるので，ALOE の考え方が適用できる.

（2） ユーザとのインタラクションがあまり能率的でない.

ALOE では，ユーザの入力はすべて構文木のテン

プレートに従って行う．したがって，論理式“$x>0$”を入力するときには，“>”，“x”，“0”の順に入力していかなくてはならない．これについては，Waters の指摘[5]と，それに対する Habermann ら[6]の反論がある．文字列を入力して，それが構文的に正しいかどうかを確かめるためには，文字列の構文解析が必要になる．構造エディタでは，テンプレートによる入力と，文字列の入力とその構文解析の両者を能率よく融合することがユーザにとって使い易いものとなるであろう．このため，構文解析は，たとえば文や式の一部分を解析するものでなければならない[7]．

（3） エディタの生成系は，便利に使える．

ALOEGEN 生成系は，文脈自由型の言語に対する構文向きエディタを生成することができる．言語の定義は，C や Pascal に対して 200～250 行位であり，比較的容易にエディタが実現できる．前述したテンプレート方式だけを使っているので，生成系の実現が比較的容易になる．

（4） 意味処理の強化が必要である．

ALOE では，意味処理をアクションルーチンで行うことにしている．それ以上の意味処理の内容は，ユーザに任せられている．システム側が，標準的なアクションルーチンを用意しておくと便利である．

6. おわりに

Gandalf システムにおける構文向きエディタ ALOE の概要を示した．この型のエディタの一つの原型として良くできている．

今後，この型のエディタの発展方向はいろいろになるであろうが，構文木が生成されること，ユーザとのインタフェースが柔軟に選択できることなどを考え合わせると，一つの知的言語処理系として集大成化してゆくのが面白いと思われる．

参考文献

1) Habermann, A. Nico et al.: The Second Compendium of Gandalf Documentation, Dept. of Computer Science, Carnegie-Mellon University (May 1982).
2) Denning, P.: ACM President's Letter: Smart Editors, CACM, Vol. 24, No. 8, pp. 491-493 (Aug. 1981).
3) Titlebaum, T. and Reps, T.: The Program Synthesizer: A Syntax-Directed Programming Environment, CACM, Vol. 24, No. 9 (Sep. 1981).
4) Donzean-Gouge, V. et al.: Programming Environments Based on Structured Editors: The Mentor Experience, Technical Report No. 26 (July 1980).
5) Waters, R.: Program Editors Should Not Abandon Tex Oriented Commands, SIGPLAN Notices, Vol. 17, No. 7 (July 1982).
6) Notkin, D. and Habermann, A. Nico et al.: The Letter to the Editor, SINGPLAN Notices, Vol. 18, No. 4 (Apr. 1983).
7) Kaiser, G. and Kant, E.: Incremental Expression Parsing for Syntax-Directed Editors, CMU-CS-82-141, Dept. of Computer Science, Carnegie-Mellon University (Oct. 1982).

（昭和 59 年 6 月 4 日受付）