オペレーティング・システム 27－5
（１９８５．５．１０）

論理型言語Uranusのプログラミング環境

中島秀之
（電子技術総合研究所）

１．はじめに

本論文ではUranus［Nakashima 1985, 中島1985］のプログラミング環境について、エディタ、ステッパ、エラー処理を中心に述べる。

Uranus(Universal Representation-Aimed Novel Uranus System)は、述語論理に基づく知識表現システムである。Uranusは、Prolog/KR［中島1984］の発展形であり、現在Symbolics Lisp Machine上で動いている。Prolog/KR とともにPrologの拡張版になっており、Prologのプログラムはシンタックスと多少の述語名の変更だけでUranus上で実行可能である。そのためのライブラリも用意してある。

Uranusでは手続き的知識と宣言的知識が同様に容易に表現できる。自動バックトラックを行なう非決定的なものと、バックトラックの範囲を限定する決定的なものができる限りペアで用意してあるので、ユーザは任意のスタイルを選択することが可能である。このため、プログラム開発の初期の段階でアルゴリズムがはっきりしない場合には宣言的にプログラムを記述して、効率が悪くともバックトラックしながら走らせ、後に制御構造を追加して、効率のよい手続き的なものへと変えていくことが可能である。

Uranusの最大の特徴は、多重世界機構とその制御のメカニズムにある。これにより知識の階層構造や時間の表現が可能になる。ちなみにUranusというのはギリシャ神話に現われる天空の支配者で、多重世界機構にちなんで本システムもUranusと命名された。

Uranusのプログラミング環境に関しては、柔軟性を第一に考慮してある。エディタやエラーの場合の動作はユーザによって任意に変更可能である。また、冗長性を各所に持たせてユーザの負担を少なくしている。例えば、制御用述語も互いにオーバーラップする部分が多いことや、エディタやステッパの中からプログラムの実行ができるので、ユーザはトップレベルと往復する必要がないことが挙げられる。

UranusはLisp Machine上にインプリメントされているため、独自のウィンドウを持つなど、Lisp Machineの機能をそのまま使っている部分も多い。その部分については、本論文では触れない。また、もともとはProlog/KR としてTTY ベースの設計になっているため、完全にはマルチ・ウィンドゥの機能を使いこなしていない。これらの理由により、本論文で述べるのは、基本的にはTTY ベース（画面制御をしないものの意味）のプログラミング環境だと思っていただきたい。

2．エディタ

Uranusは最初Prolog/KR として東大大型計算機センターの、半二重回線の環境で作成された。このため、画面エディタの使用は非常に困難な状況であった。そこで Prolog/KR ではTTY ベースの構造エディタ(AMUSE) ［Nakashima, Tomura 1983］を採用することにした。現在Uranusでは、この構造エディタの画面版と、Lisp Machine組込みの画面エディタZMACS (EMACSの発展版) ［Symbolics 1984］が使用できる。筆者自身は、ファイルを基本とするプログラム開発にはZMACS を、実行時のちょっとした変更にはAMUSE を用いている。

構造エディタではコメントの取り扱いが困難で、その点、大きなプログラムに関しては、文字ベースのエディタに分がある。しかし、構造の操作は、構造エディタの方が楽なようである。

AMUSE はLispの構造エディタとしてはInterlisp のものと基本的には同じ構成をしている。ただし、Prolog的パターン・マッチングの機能を取り入れることによりエディット能力の向上がみられる。例えば

```
RA (FOO &1 &2) (FOO &2 &1)
```

によりすべてのFOO の第１、第２引数の順序を入れ替えることができる。 RA は Replace All の意味である。

AMUSE の最大の特徴はエディット能力以外の部分にある。一つは、Ｓ式を基本とする言語なら何でも扱えるという点である。これはシンタックスに限らずセマンティクスを含んでいる。例えばAMUSE にはX コマンド(eXecuteの意) があるが、これはLispエディット時にはLispとして、Uranusエディット時にはUranusとしてその時点で対象となっているＳ式を実行するものである。この、言語ごとに異なった機能をサポートするため、AMUSE は呼出し時に５つの引数をとる：

定義の読み方
定義の格納の仕方
ファイルの読み方
ファイルへの格納の仕方
式の実行の手段

である。これらはすべてLisp関数として記述される。例えば、定義の読み方はLispの場合なら

```
(lambda (x) (get x 'expr))
```

のようになろうし、Uranusの場合は

```
(lambda (x) (get x (car :current-world)))
```

となる。ただし、これらはシステム作成者が作ればよく、ユーザは知る必要がない。

AMUSE のもう一つの特徴はマクロが対象言語で記述できるという点である。Lispのエディット中にはLispで、Uranusのエディット中にはUranusで、各々エディット・マクロ

が記述できる。これまでのエディタのマクロ記述は、TECOのようにそれ専用の言語で書くか、EMACS のように条件判定のない単純なコマンド列だけがマクロとして定義できるかのどちらかであったが、AMUSE ではプログラム・エディタという特色をフルに生かして、一般言語によるマクロが書ける。このために、すべてのプリミティブはコマンドとしても関数／述語としても呼べるようになっている（コマンド名の頭にec: を付けたものが関数／述語名である）。例えばすべてのFOO の呼出しに、各々異なる第１引数を追加したい場合には、

```
(and (ec:f foo)                    ; foo を探す (Find)
     (loop (ec:v)                  ; foo を含むリストを印刷する (View)
           (ec:read *x)            ; 端末から第１引数を読む
           (ec:in *x)              ; foo の次に挿入する (Insert Next)
           (or (ec:fn) (exit))))   ; 次のfoo を探す (Find Next)
```

のようなマクロを使えばよい。これは、述語として定義しておいて、エディト中に呼出すことが可能である。

ZMACS のほうはLisp Machineの標準エディタで、文字を基本単位とする画面エディタである。ただし、様々な言語モードが用意してあり、例えばLispモードではＳ式単位の操作ができる。Uranusとのインターフェースを作るにあたってもLispモードに若干の修正を加えることで達成できた。ZMACS はFlavor［Weinreb, Moon 1980］で書かれているため、ほとんどの処理はLispモードから継承し、少数のUranus特有のメソッドを追加したにとどまる。オブジェクト主導の考え方のソフトウェア工学的有用性の一端である。

ZMACS のUranusモードでは、基本機能の他に、Uranusプログラムの実行と、バッファーの中身を実行してUranusに戻るコマンドがサポートされている。後者に関しては、前の定義に追加するもの（ほとんど使わない）と、前の定義を置換えてしまうものがある。assertを使った定義は、それまでの定義を消さないが、ファイルをエディットしてロードし直す場合には、前の定義が残っていては困るからである。この機能はUranusではload・reloadの区別、DEC-10 Prolog では consult・reconsult の区別に対応する。

３．ステッパ

ステッパはUranusプログラムを１ステップづつユーザと会話しながら実行するものである。新たな述語が呼ばれると、それを表示し、その後の動作の指示を受ける。単に実行過程を表示するだけのトレースの機能もあるが、ステッパのほうが柔軟性に富んでいる。トレースにも、引数の状態によって動作を指定するなど、機能の拡張が望まれる。

現在ステッパには以下のコマンドが用意されている：

[ ]	次の呼出しへ進む（c と同じ）
[a]ll	すべての呼出しで止まる
[b]acktrace	呼出しの履歴の表示

[c]ontinue	次の呼出しへ進む（空白と同じ）
[f]inish	止まらずに実行を最後まで続ける
[g]o	現在の呼出しのみを止まらずに実行する
[l]evel <number>	印刷の深さを指定する
[n]ext alternative	現在の呼出しを失敗させ、次の選択枝に移る
[p]rint	現在の呼出しを印刷する
[q]uit	実行を中止し、トップレベルに戻る
[s]elect <predicate>	選択的に止まる述語名を指定する
[u]p	一つ上の呼出しまで止まらずに実行する
e[x]ecute <predicate>	述語を実行する
<predicate>	述語を実行する（x と同じ）
[HELP]	コマンド一覧と意味の印刷
[その他]	コマンド一覧の印刷

各々[]内１文字で起動され、[]以外の部分は自動的に補われる。例えば、c と打てば、表示ではContinueとなり（これはLisp Machineの組込み関数でサポートされている）、そのコマンドが実行される。

continue, finish, go, next, quit, up以外のコマンドでは実行は先には進まず、次のコマンド待ちとなる。

なお、ステッパに入るには、トップレベルで (step <predicate>) を実行するか、実行中に ^G を押せばよい。後者の場合は最初にall あるいはselectコマンドを指定しないとステップ・モードにはならない。^G はgoやfinishで通常モードの実行をしている最中にステップ・モードに戻るのにも使える。

インプリメンテーションの都合でユーザの実行のイメージどおりの表示のできない部分がある。例えば、ある述語を呼んだ時点での表示は可能であるが、それが成功した時点ではスタックに情報が残っていないので表示できない。より正確に言うと、テール・リカージョンのオプティマイズなどのため、スタックは既に呼ばれた述語に関する情報は（バックトラック用のものを除いて）一切保持していない。呼出しが成功した場合には、次の呼出しへと制御が移るので、一応は成功が確認できるが、失敗した場合にはそれが表示されないと何が起こっているのか追跡が困難になる。そこでステッパ使用時には失敗の情報をわざわざスタックに残すようにしている。

４．エラー・割込み処理

４．１．標準動作

エラーの際にはERROR 、割込み（ ^G による）の際にはBREAK という述語が各々呼ばれる。システムの標準動作は両者ともステッパ（３節参照）に入るというものである。ステッパでall コマンドを発行すると、ステップ実行モードに入る。これは特に、実行途中か

らステッパに入るのに便利である。また、continueコマンドを発行すれば実行はそのまま継続する。即ち、割込みの場合はその直後から実行が継続するし、エラーの場合には、その原因となる述語の実行をスキップしてその次の述語の実行に移る。

4．2．動作の変更

Uranusは柔軟なエラー・割込み処理の機能を持っている。エラーや割込みの際の処理を、ユーザが一般のプログラムと全く同等に指示できる。動作の変更にはerrorsetのような特別な設定は不要で、単にERROR あるいはBREAK という述語を再定義すればよい。

ERROR は２引数で、第１引数がメッセージ、第２引数がエラーを起こした呼出しになっている。メッセージごと、あるいは第２引数を見てより細かい対応を指示することが可能である。

例えば、Uranusでは未定義述語が呼ばれたときにはエラーになるが、これを Dec-10 Prologのように単なる呼出し失敗に変更するには

```
(assert (error "UNDEFINED PREDICATE" *) (fail))
```

とすればよい。他のエラーに対してはシステムの標準ERROR が呼ばれる。更に細かく述語ごとに指示をして、

```
(assert (error "UNDEFINED PREDICATE" (foo . *arg)) (bar . *arg))
```

とすることも可能である。これは例えば(foo a b) が呼ばれたが、foo が未定義の場合には、代りに(bar a b) を呼ぶことを指示している。未定義の述語が呼ばれたときに、それを同名のファイルからロードするいわゆるオート・ロードの機能は

```
(assert (error "UNDEFINED PREDICATE" (*p . *arg))
        (load *p)                    ;ファイル*pの中身をロードする。
        (*p . *arg))                 ;*pを呼ぶ。
```

のように実現できる。同様にして、エラーの中身を調べた後再びエラーにするということも可能である。エラーを起こすには単にERROR を呼べばよい。

5．今後の課題

ステッパの中からのスタック・フレームの操作は、今のところ、スタック内の情報があまりにもインプリメンテーションに依存する形なのでユーザには解放していない。将来より自然な形での解放を考えたい。

また、ステッパの中から変数の値を見る機能などの追加も望まれる。これには、Prolog系の言語の変数の、入出力に関するセマンティクスを見直す必要がある [Nakashima, Tomura, Ueda 1984]。即ち、一度出力した変数をそのまま読込んだ場合に元と同じ変数にならないという問題がある。Lispなどのパラメータ引渡しを基本とする言語では、現在の環境のX などという考え方でよいが、UranusやPrologではユニフィケーションの問題があり、一般には任意の環境の変数が指示できる必要がある。Uranusでは、同一名の変

数の異なる出現を区別するために*X_012 のような記法を用いているが、これを読み込んだ場合には、そういう名前の別の変数だと思われてしまう。

6．まとめ

Uranusのプログラミング環境について述べた。特色としては、(1) 柔軟性を第一に考慮してある、(2) 冗長性を各所に持たせてユーザの負担を少なくしている、という２点が挙げられる。

謝辞

本論文にコメントをいただいた、電総研siganyのメンバー（梅山伸二、戸村哲、新田克巳、樋口哲野、二木厚吉各氏）に感謝する。

参考文献

Nakashima, H. and Tomura, S.: Introduction to AMUSE, A Multi-Use Structure Editor, METR 83-2, Univ. of Tokyo (1983)

Nakashima, H., Tomura, S. and Ueda, K.: What is a Variable in Prolog? Proc. of the International Conference on FGCS 1984, pp. 327-332 (1984)

Nakashima, H.: Uranus Reference Manual, Research Memo. ETL-RM-85-1, Electrotechnical Lab. (1985)

Symbolics, Inc.: ZMACS Manual (1984)

Weinreb, D. and Moon, D. : Flavors: Message Passing in the Lisp Machine, MIT AI Memo 602 (1980)

中島秀之：知識表現用言語としてのProlog/KR, 情報処理学会論文誌25巻 2号(1984)

中島秀之：超時空プログラミングシステムUranus, 第26回プログラミング・シンポジウム予稿集, pp.13-23 (1985)