都市・地域計画のための言語処理とプログラム連係ロジック

㈱東洋情報システム ○山崎明，村上明，岡修，国領茂

1．緒言

都市・地域計画のシステム的アプローチのためのコンピュータアプリケーションを考えた場合，統計解析，数理計画，システムダイナミックス，計量経済モデル分析，グラフ・マトリックス手法，図形処理，地域情報データベースなど，多くの種類のプログラムを準備し，利用しなければならない。

都市もしくは地域計画を担当する者は，各種解析に関してはスペシャリストではあるが，コンピュータに関してはそれほど詳しい知識は持ち合わせてはいない。この様な人々の作業を支援するコンピュータシステムとしては，都市・地域計画に必要なプログラムを総合的に統括するものが必要となる。このため今回開発したシステムは，多くのプログラム群を統一した問題向言語を持たせて利用し易くし，さらには各プログラムを問題に応じて有機的に結合できるようにしたものである。また，昨今のコンピュータの利用形態を考慮すると，これらのプログラムは対話形式のＴＳＳ処理が必須となってきている。ただ，多量の出力をともなう場合や計算時間のかかる場合にはバッチ処理による利用もなされるので，この言語処理プログラムは，バッチ・ＴＳＳ処理併用可能な汎用プログラムでなければならない。

2．汎用言語処理システム

2．1　汎用言語処理の概念

問題向言語は場合によっては1つのステートメントでかなり大きな仕事をする。言いかえると非常にマクロ的である。また，ステートメントは一定の仕事をするサブルーチンへのアーギュメントである。そこで，ステートメントに対応する実行ルーチンを別に作り，システムライブラリーに登録し，言語処理プログラムにはそのステートメントを自由に設定できるようにする。そうするとこの言語処理は用途に限定されない汎用言語処理システムとなる。汎用言語処理システムの基本構造は，ステートメントを翻訳するための共通手段はすべて持っているが，個別のステートメントを翻訳するための情報は別に登録するというシステムになっている。汎用言語処理システムによって翻訳される言語は基本的な制御命令等のステートメントだけを持っていて，ある特定の問題に対して機能するステートメントは登録後使用する。また同時にこのステートメントに対応する実行ルーチンを登録しなければならないことは当然である。もし，ステートメント翻訳情報や実行ルーチンを登録制にしなければ，すなわち，問題向言語をクローズド・システムとして機能を固定してしまうならば非常に不便なものとなる。というのは実際のジョブはたいへん流動的で，問題解決の方法もたえず変っていくのが常であるからである。

この汎用言語処理システムはどのような分野にも適用できるので用途に制限はない。またどのような分野への適用にも耐えるものでなくてはならないから，電子計算機の専門家でない一般使用者向に作られる。

2．2　命令文の翻訳

（1）ステートメントの翻訳

翻訳時に特殊処理を必要とする一部の命令文は，それぞれの命令文に応じた命令処理ルーチンによって翻訳される。算術命令文は，逆ポーランド記法を利用して，数式翻訳ルーチンによって翻訳される。

手法命令文，入出力命令文，制御命令文のうち特殊処理を必要としないものは，すべて構文解析ルーチンと呼ばれる1つの汎用言語解析ルーチンによって翻訳される。構文解析については後で説明する。

ほとんどのステートメントは1ステートメント毎に翻訳されていくが，ある種の命令文は1つのステートメン

トだけでは意味を持たない。

例えば DO 文がそうであり，これには特殊処理が必要である。

（２）ＴＳＳ時の対話処理

本システムはＴＳＳ処理においても，今までのＴＳＳ処理専用のプログラムより使い易い工夫を行った。すべて誘導方式による入力形式ではなく，利用者は自分の理解している所まで入力すれば，常にそれ以降について対話的な誘導が得られ，慣れれば一括入力に移って対話モードのわずらわしさを避けられるようにしたので，最も効率的に命令文入力を行える。

いま利用者が

ＷＡＴＥＲ＝（ＨＯＵＳＥＨＬＤ，ＦＡＣＴＯＲＹ，ＯＦＦＩＣＥ，ＴＩＭＥ）

Ｃ１，ＰＬＯＴ＝（ＶＡＲＩＡＢＬＥ）

という形の処理を希望し，これはある地域の水の需要量をその地区の世帯数，工場数，事務所，および時間的趨勢によって説明するモデルをつくるためのものと想定する。

ＴＳＳ処理の場合は，前命令文の処理が終了すると“ＥＮＴＥＲ　ＣＯＭＭＡＮＤ”というメッセージが出力される。ここで，命令文の全文をタイプしたとすれば，一時的にバッチモードとなり，バッチ処理の場合と同じ過程の翻訳が実行される。

命令文の書き方がわからないときには命令部の直後に“？”を入力すると，記述部の書き方について，リファレンスカードに印刷されている程度のメッセージが出力されるので，利用者はそれを見ながら入力できる。

また，命令部のみを入力すると，この場合には最初の項目の“従属変数名”の入力を求め，応答がなされると，つづいて“独立変数名”の入力が求められる。ここで記述部の書き方が理解できたので残りの部分を一括して

ＦＡＣＴＯＲＹ，ＯＦＦＩＣＥ，ＴＩＭＥ

と入力したとすれば，直ちに仮バッチ・モードに移り，入力された要素をすべて翻訳し終るまでメッセージは出さない。この状態では，まだ命令文は終了していないのであるから，自動的に，ふたたび対話モードにもどり，次の“係数を格納すべきベクトル名”の入力が要求される。

以下同様にして対話は命令文が終るまで進行し，必要な要素がすべて入力されるとその命令の実行に移る。

（３）構文解析の概念

一部の特殊な命令文と数式を除けば，ステートメントの記述部は構文解析ルーチンという汎用ルーチンによって翻訳される。

まず予めステートメントの構成及び要素に関する情報を登録しておく。ステートメントに応じて，これらの情報を書き表わしたものが構文であり，要素の種類を表わす記号が重要な役割を果たしている。これは例えば次のようなものである。Ｉ（整数），Ｅ（実数），Ｖ（ベクトル），Ｌ（ラベル），＆（キーワード），・・・

これらを命令文に応じて並べたものによってステートメントを解釈するのである。例えば“ＧＯ　ＴＯ　ラベル”の構文は“Ｌ”である。また“ＧＯ　ＴＯ　（ラベル１，ラベル２，・・・），整数”の構文は“’（’ ｎＬ ’）’ Ｉ”である。ここでｎは整定数で，くり返しを表わし，カッコの中に並べることのできる最大のラベル数である。’　’で囲まれた内容は，ステートメントの要素と直接比較される。記述された構文はユーティリティによって登録され，構文解析ルーチンに都合のよい形式に修正されて表形式で保存される。これが構文表である。ところで先のＧＯ　ＴＯ　の例では同じ命令に対して２通りの構文があり，さらに後者においてｎは不定である。このように１つの命令に対して記述部は固定されているわけではなく，いろいろなバリエーションが存在する。そこで構文解析によって，ステートメントがどのような変数のストリングによって構成されているか決定してやらなければならない。

2.3 構文解析

（1）構文の記述

構文を表形式で貯える際にインプットとなるものが，構文ステートメントである。構文ステートメントは次の要素から成り立っている。 a)変数の種類を表わす記号 b)要素又はその集合を表わす記号 c)構文の特殊処理を表わす記号 d)変数の定義を示す記号 e)ＴＳＳ処理時の対話メッセージ ここで変数とは通常の変数の他にキーワードを含んでいる。要素とは構文の特殊処理を表わす記号である。本来の構文ステートメントは要素と対話メッセージが混在しているため，ここで説明している形式のものではないが，構文ステートメントを変換するプログラムを通すことにより容易に要素と対話メッセージアドレスの組の並びと変換できる。また要素にはすべて対話メッセージのアドレスがつくが，ここでは，これも省略した形で説明を行う。

まず変数の種類を表わすには，整数ならＩ，実数ならＥ，といった記号を使う。キーワードは' 'で囲んで'ｋｅｙ'の様に書く。

（例１） ステートメント "v_1 key v_2" に対応する構文ステートメントは "V 'key' V" となる。

変数の種類を表わす記号（変数記号）のくり返しを表わすには nV と書く。また変数記号の集合についても n(……) という形でくり返しを表わすことができる。

（例２） ステートメント："v_1 ＩＮ v_2，v_3 ＩＮ v_4" 構文：２(Ｖ 'ＩＮ' Ｖ)

くり返し数が不定のときは特定の語を¥記号の後に置くことによってくり返しの打切りを指定できる。

（例３） ステートメント："(v_1，v_2，…v_n)" 構文：'('６(Ｖ)¥')'

特殊記号としては上に述べた（ ）のほか［ ］，{ | | }，/* free free */ がある。省略可能な要素又はその集合は［ ］で囲む。いくつかのキーワードのうち，どれか１つを選択するときには{ }で囲み，|で区切る。

（例４） {'ＩＮ'|'ＡＴ'|'ＦＯＲ'}

ここで，各選択肢の内容は，１要素のみでなく，要素の集合等でも可能である。

いくつかの要素もしくはその集合の入力順が自由な場合がある。この時には /* */ で囲み，freeで区切る。

（例５） ステートメント： ＣＯＥＦ ＩＮ Ｖ Ｒ ＩＮ ｅ

又は Ｒ ＩＮ ｅ ＣＯＥＦ ＩＮ Ｖ

構 文 ： /*'ＣＯＥＦ''ＩＮ'Ｖ free 'Ｒ''ＩＮ'Ｅ*/

変数がその時点で定義されるものには変数記号の前に＊をつける。（例 ＊Ｖ） 既定義でなければならない変数には何もつけない。

（例６） ステートメント： ＲＥＧＲＥＳＳ $v_1 = v_2$(v_3，v_4，…v_{11}) ＣＯＥＦ ＩＮ Ｍ Ｒ ＩＮ ｅ （ここでＣＯＥＦ…，Ｒ…は順序任意で省略可能）

構 文 ： ＊Ｖ '=' Ｖ '(' ９Ｖ¥')' /* ['ＣＯＥＦ' 'ＩＮ' Ｍ] free ['Ｒ' 'ＩＮ' ＊Ｅ] */

（２）構文表

構文ステートメントに書かれた構文解析に必要な情報は，構文解析ルーチンに便利な形に変換されて構文表に納められる。ここではその過程と，結果としての構文表を説明する。

構文表は第１行目にはくり返しの個数が入り，第２行目には変数記号，及び（ ） ［ ］ 等の特殊処理記号が入る。第３行目には，変数記号に対しては定義するかどうかの記号＊が入り，キーワードに対してはそれぞれのテーブルのアドレスが入る。また，特殊記号の場合には，処理の流れを示す飛び先が入っている。第４行目には，その変数が何番目のアーギュメントであるかを示す Argument Number が入る。第５行目には，ＴＳＳ処

理で，その要素が入力されなかった場合，又は誤入力の場合の入力要求メッセージのアドレスが入っている。

構文ステートメントで記述された構文は上に述べた様な形式で，おおむね元の形のまま表に入れられるが，キーワードはテーブルのアドレスに変換されている。またキーワードの記号は&になっている。また特殊記号などは，前にJUMPコードが入って複数記号に展開されるものもある。

くり返し個数	
要素記号	ＶＶ＆Ｖ＊＆：
定義又はアドレス	＊　3　5
アーギュメントNo.	1 2 3 4
メッセージアドレス	1 2　3

構　文　表

3．ダイナミックなプログラム連係

今まで述べてきた方式によれば，手法の追加と，それにともなう命令文の翻訳機能の追加は事実上無制限に行いうる。ただ現在のオペレーティングシステムのもとで，標準的なオーバーレイ構造により，手法群を連係した場合には，手法の個数に制限ができる．また，都市及び地域計画の基礎は生きた学問であり，成長期にある学問である。新しい手法は日々に出現し，新しい分析計画は年々に新分野を拓いて行く。本システムが本来的な自律成長を前提とした場合，各手法プログラムはそのプログラム単位で追加・修正のできるものでなければならない。

本システムは，これを実現するために，コントロールプログラム以外は手法プログラム単位でプログラムライブラリーに保存するようにし，コントロールプログラムは必要な手法プログラムを必要時点にダイナミックにリンクする構造にした。

言語処理プログラムは，命令文を解読して，それに対応する手法名（これはプログラムライブラリーに登録するときのメンバー名）と引数表とをつくり出す。ここで，この手法名をもとに，アセンブラ・ルーチンにより，プログラムライブラリーから手法プログラムを読み込み，コントロールをこれに渡す。

この時，別プログラムから別プログラムへとダイナミックにコントロールが渡されるため，原則的には共通ルーチンの使用はできない。すなわちI/Oモジュールであろうと別になるので，実際の実行時にはI/Oバッファが累積されるという事が起った。これを回避するため，コントロールプログラムをリンクする場合，まず共通ルーチン（特にI/Oモジュール）のアドレスを渡し，手法モジュールから共通ルーチンをCALLした場合，そのアドレスへブランチする構造にした。

4．結言

本システムの特徴をもう一度，整理・集約してみると，次の諸点にまとめることができる。

- 応用プログラムに問題向言語を附加する形式で，本システムは利用される。
- 本システムを利用すると，応用プログラムは入力部と処理部とが独立のモジュールになる。したがって，応用プログラムとして，入力仕様を変更したい時も，その処理部には影響がない。
- 入力解釈は構文表を基本にしているので
  1）対話型で入力中でも自動的にバッチ型に移行して，入力手順を能率化できる。
  2）入力型式の設計を，対話型とバッチ型の二者択一にしなくてもよい融通性がある。
  3）入力仕様の変更が容易である。
- プログラムの追加修正が容易であり，プログラム数の制限も無くなる。