ソフトウェア工学　44－4
（1985. 10. 2）

計算機誘導型構造エディタPARSE*の自動生成システム

田中　厚　　岸本　芳典　　中所　武司
（日立製作所システム開発研究所）

1　はじめに

マイクロコンピュータをはじめとする計算機の応用分野の拡大に伴い、ソフトウェアの生産性と信頼性の向上は、ますます重要な課題となってきている。1970年代には、ソフトウェア工学という新しい研究分野が確立され、ソフトウェア開発工程全般にわたる方法論、技法、ツールの開発が行われてきた。中でも、構造化プログラミングに代表されるモジュール設計やプログラミング技法の進歩は著しく、その支援言語[1]も数多く開発され実用に付されてきた。

しかしながら、ソフトウェア開発費用全体に比べてこれらの言語の寄与部分には限度があることから、最近では、言語周辺のツールを含めた統合プログラミング環境[2),3)]が重要視されている。

われわれは、統合プログラミング環境開発の基本方針として、各ツールの有機的結合による統合化と対象言語への適応による高機能化を掲げ、これを「言語適応型プログラミング環境」[4),5)]と名付けた。そして、まず、そのユーザインタフェースとなる構造エディタPARSE[6),7)]を開発することにした。従来のテキストエディタでは、入力したプログラムの文法誤りがなくなるまで修正を繰り返さねばならなかったが、PARSEでは、対象とするプログラミング言語の文法を内蔵し、これに基づいて計算機が文法誤りのないプログラムを誘導するため、信頼性の高いプログラムを効率良く作成できる。

一方、種々のプログラミング言語に対して類似した機能を持つ構造エディタを作成する場合には、構造エディタを自動生成するツールを用意しておくと効率的であることから、PARSEの自動生成システムPARSE－G**[7),10),11)]を開発した。

本文では、構造エディタPARSEとその自動生成システムPARSE－Gの開発思想および機能について述べる。

2　構造エディタPARSEの設計思想

2．1　基本方針

プログラミング環境にはエディタが不可欠であるが、従来のテキストエディタには、ユーザからみて、次のような問題点があった。

(問題1)　プログラミング言語に依存しないようにして汎用性を持たせているため、機能的に低水準である。

(問題2)　開発者の経験に合わせて作成されているため、プログラミングの初心者には操作しずらい。

そこでこの問題に対処するため、次の基本方針が有効であると考えた。

(方針1)　言語適応化による高機能性の実現

(方針2)　ユーザ適応化による高操作性の実現

方針1は、エディタの機能をそれが対象とするプログラムを記述した言語に適したものにすることを意味する。

方針2は、エディタのユーザインタフェースを、それが対象とするユーザのレベルに適合させることを意味する。すなわち、プログラミング経験の浅い初心者からベテランプログラマである熟練者まで、すべてのユーザの習熟度に応じた柔軟なユーザインタフェースを与える。

2．2　言語適応化の条件

方針1の言語適応化には、次の条件が必要である。

(1) プログラムへの視点の高度化

従来のテキストエディタは、プログラムを単なる文字列とみなし、文字単位あるいはまとまった文字列からなる行単位のテキスト編集を行うものであった。このように、プログラマはプログラムを操作対象とするときはそれを文字とみる一方、思考対象とするときは、それを意味ある文章とみる。即ち、書くレベルと読むレベルの間に大きなギャップがあり、これが、単にプログラミング効率の低下ばかりでなく、誤り発生の原因になっている。そこで、PARSEでは、言語の構文単位の視点を提供することにより、従来のエディタに関する意味的ギャップを除き、操作対象のレベルを思考対象のレベルにまで引き上げる。

(2) 対象言語へのプログラミング技法の適用化

構造化プログラミングなどのプログラミング技法は、計算機言語に組み込む形で支援されているものもあるが、一般には言語に依存せず汎用的である。そのため、プログラムの記述に当ってこれらの技法を直接適用することは困難である。そこで、プログラミング技法を対象言語の機能に適応させ、容易に適用可能とする。

2．3　システム構成

---

*) PARSE　: Production And Reduction Structure Editor

**) PARSE-G : PARSE Generator

図１に、構造エディタＰＡＲＳＥのシステム構成を示す。図の(1)はソフトウェア構成で、言語適応化により必要となるルーチン群をプログラム解析系としてまとめるとともに、各種コマンド処理ルーチンで共用可能とするため、バックエンドプロセッサとする。一方、ユーザ適応化により必要となるコマンド解析や画面出力ルーチン群をユーザインタフェース処理系としてまとめ、フロントエンドプロセッサとする。図の(2)は、ソフトウェア構造を階層的データ抽象構造で表現したものである。

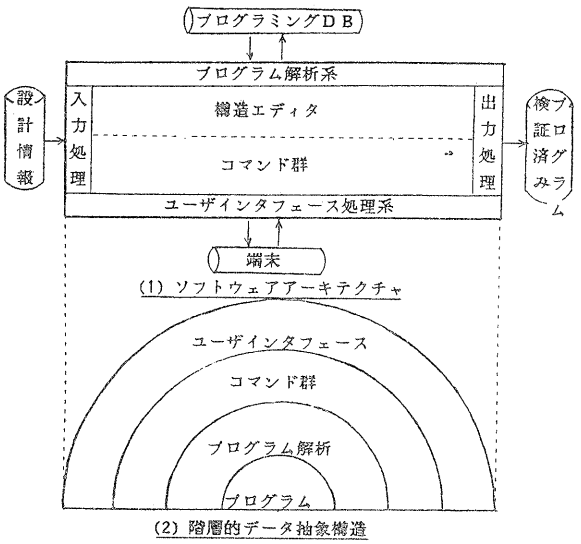


図１　構造エディタＰＡＲＳＥのシステム構成

３．構造エディタＰＡＲＳＥの基本機能

前章で述べた設計思想から、構造エディタＰＡＲＳＥには次のような機能を持たせた。

(1) 段階的詳細化機能
(2) 構造化コーディング機能
(3) 構文要素単位の編集機能
(4) 構文要素単位の画面編集機能
(5) コマンド入力の誘導機能
(6) システム状態表示機能

以下、各機能ごとに説明する。

３．１　段階的詳細化機能

E.J.Dijkstra教授の提唱する[7]「一度に一つの決定」の原則に沿った段階的詳細化法によるプログラミングを支援する。そのため、ＰＡＲＳＥが扱うプログラムに次のような詳細化未定義部分(Refinable)を導入する。

・言語の文法を表すＢＮＦ規則に用いる非終端記号
（例：<STATEMENT>）
・プログラムの文の代わりになる擬似文
（例：”データ　ヲ　スタック　ニ　ツム”）
・プログラムの手続き参照
（例：ＰｕｓｈＤａｔａ（Ｓ，Ａ））
・プログラムの新データ型参照
（例：ＶＡＲ　Ｓ：Ｓｔａｃｋ）

これら４項目のうち、後の２項目はプログラミング言語自身が有する段階的詳細化支援機能である。第２項は第３項の手続き参照に相当するが、構文規則に制約されないでコメント風に記述できるようにするため導入した。また、第１項は、文法規則に沿った詳細化を促すために導入した。

(1) 処理概要の段階的詳細化機能

まず、処理概要を擬似文で記述した場合、この擬似文の詳細化を要求すると、それがコメントに変更され、<STATEMENT>の入力要求状態になる。一方、処理概要を手続き参照形式で記述した場合、詳細化要求すれば擬似文と同様にインライン展開形式になり、定義要求すればその手続き本体の枠組み（テンプレート）が表示され、手続き定義の要求状態になる。

次に、データ構造の詳細化機能として、ユーザ定義データ型参照部分に対して詳細化要求すれば、型名がコメントに変更され、データ型の入力要求状態になる。

(2) 構文要素の段階的詳細化機能

構文要素の段階的詳細化では、Refinable(詳細化未定義部分)を逐次表示しその詳細化を促す。現在詳細化要求中のRefinableを、Current Refinable(以下CRと略す)と呼ぶ。このとき、可能な詳細化の形式の一覧を表示するが、その中から一つを選択する方式と、対応するテキストを直接入力する方式を設けた。

尚、作成済みのソースプログラムをＰＡＲＳＥで編集できるようにするため、上述のような端末からの入力のかわりにファイルから入力する一括入力機能を導入した。

３．２　構造化コーディング機能

構造化プログラミングでは、基本制御構造を選択、反復、複合（これらを基本制御文と呼ぶ）としている。ＰＡＲＳＥでも基本制御文を用いた構造化コーディングを支援するため、入力したい基本制御文のメニュー選択あるいは専用コマンド（テンプレートコマンド）の入力により、そのテンプレートを自動生成して表示するようにした。尚、操作に慣れたユーザ用に、直接テキストで基本制御文を入力する機能

を設けた。

3．3　構文要素単位の編集機能

プログラムの修正は、プログラム入力の場合と同じく、構文要素単位に行えるようにして、修正時のエラー誘発を防止する。修正としては、挿入、削除、置換、移動、探索が基本であり、次のような機能を設けた。

(1) 挿入は、まず、該当部分にカーソルを移動後、挿入コマンドによって、カーソル位置に挿入可能な非終端記号を挿入する。その後、プログラム入力時と同じ方法で挿入したいテキストに置換する。

(2) 削除は、該当部分にカーソル移動後、削除コマンドによって行う。

(3) 置換は、還元コマンドでカーソル位置のテキストを非終端記号に置換した後、プログラム入力時の方法で新しいテキストに置換する。

(4) 移動は次のように行う。ＰＡＲＳＥは、プログラムを格納する領域を二つ持つ。一つは、言語仕様の定義の開始記号<PROGRAM> から始まり、文法的に常に正しいプログラムを格納する領域（プログラム領域）で、一つは、プログラムの一部分を格納する領域（ワーク領域）である。そして、まず、移動すべき部分をワーク領域へ退避し、つぎに、それをプログラム領域の挿入すべき部分に挿入コマンドで移す。

(5) 探索は、探索コマンドを用いて構文要素単位に行う。探索方向としては前後両方向が可能である。

このように、プログラムの修正は、すべて非終端記号を経由して行うようにし、一般のテキストエディタのような、文字列操作を禁止した。

ＰＡＲＳＥを用いたプログラム編集の例を図２に示す。

3．4　構文要素単位の画面編集機能

プログラムの言語構造に適した画面表示／編集機能として、次の機能を設けた。

(1) 自動インデンテーションによるソースプログラム表示機能（プリティプリンティング）

(2) モジュールおよび手続き単位の画面スクロール機能

(3) 構文要素単位のCR移動機能

3．5　コマンド入力の誘導機能

初心者でも簡単に操作可能とするため、次のようなコマンド構文誘導方式を導入した。

(a) 入力可能なコマンド名あるいはオペランドの一覧をメニューとして表示する。

(b) メニューの選択は、メニューと１：１に対応したキーボード上のファンクションキーの押下によるワンタッチ入力で行う。

(c) 選択されたコマンド名あるいはオペランドは、逐次画面上に表示する。

(d) 一コマンドの誘導が完了すると、自動的にコマンド実行処理に移る。

一般的にメニュー選択方式では、選択項目が多い場合にそれらを一度に表示しきれないという問題がある。そこで、選択項目はグループ分けし、そのグループ単位に表示するようにした。そしてメニューをスクロールするコマンドを導入することにより、この問題点を解決した。また、コマンド構文の誘導途中でその入力を中止するコマンドを導入した。さらに、テキストで直接入力しなければならないオペランドについてはその旨を示す表示を行う。

尚、メニュー選択方式は慣れるに従って煩わしくなる傾向があるため、コマンドをテキストで直接入力する従来方式も併用する。

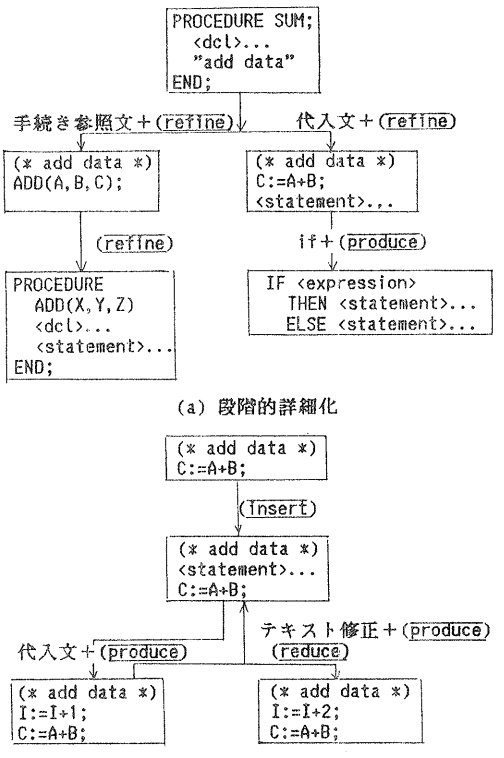


図２　ＰＡＲＳＥを用いたプログラム編集例

3．6　システム状態表示機能

ユーザがシステムの状態を見失わないようにするため、コマンド実行のたびに、画面のスクロール単位、プログラム入力モード、現在実行中のコマンドなどのシステム状態を表示する。

3．7　その他

(1) 直前のプログラム操作の取り消し
(2) プログラムのファイルへの退避
(3) システムの終了
尚、表1にＰＡＲＳＥのコマンド一覧を示す。

表1　ＰＡＲＳＥのコマンド一覧

分類	コマンド	機能
プログラム生成	produce	構文要素の詳細化
	refine	関数・新データ型の詳細化
	omit	省略可能構文要素の省略
プログラム修正	insert	構文要素単位の挿入
	reduce	構文要素単位の還元
	delete	構文要素単位の削除
	move	構文要素単位の移動
	copy	構文要素単位の複写
	find	構文要素単位の探索
コマンド入力誘導	help	ヘルプメッセージの出力
	cancel	コマンド入力の途中取り消し
	scroll menu	コマンドメニューのスクロール
画面制御	scroll screen	画面のスクロール
	move cr	Current Refinable の移動
システム制御	save	プログラムのファイルへの退避
	undo	直前操作の取り消し
	end	システムの終了

4．構造エディタ自動生成システムの開発目的

今まで述べた構造エディタを、他言語／機種用に作り変えたり、マンマシン性を改善しようとすると、次の事項が問題となる。
(問題1) 対象とするプログラミング言語への依存性が強いため、他の言語への移行性が低い。
(問題2) ユーザインタフェースは試行錯誤的に開発されるため、処理プログラムの保守性が低い。
(問題3) 稼動するＯＳや端末機種への依存性が強いため、他の機種への移植性が低い。

そこで、上記問題点を解決するため、次に示す方式による構造エディタ自動生成システムＰＡＲＳＥ－Ｇを開発することにした。
(1) 構造エディタの仕様記述言語の導入
プログラミング言語の仕様およびユーザインタフェースの仕様を記述するためにＢＮＦ(Backus Naur Form)を拡張した定義言語（仕様記述言語）を導入する。ＰＡＲＳＥ－Ｇは、仕様記述言語で書かれた仕様を入力とし、プログラミング言語とユーザインタフェースの文法的正しさ、構文解析可能性を自動検証する。
(2) 高級言語で記述したプログラムの生成
プログラミング言語の仕様からプログラム解析系を、ユーザインタフェースの仕様からインタフェース処理系を、それぞれ高級言語で記述したソースプログラム形式で出力する。さらに、高級言語としては、Ｃ、Pascalなど、複数のものを可能とする。尚、プログラム解析系には、ソースプログラムを構文解析し構文木を作成するパーサ（スキャナを含む）と、構文木からソーステキストを再生するアンパーサなどがあり、ユーザインタフェース処理系には、コマンド解析部とメッセージ出力部がある。
(3) 個別端末機種対応の入出力パッケージの導入
画面入出力など機種に依存する部分はモジュール化しておき、対象とする端末機種ごとに用意する。他機種への移植はこのパッケージの交換で行う。

(1),(2) により、言語仕様の書き換えのみで他の言語へ移行でき、ユーザインタフェース仕様の書き換えのみでマンマシン性が改良できる。また、生成するソースプログラムはＯＳからの独立性が高いので、他ＯＳへの移植性も向上する。さらに、(3) により他端末機種への移植も容易になる。

図3に、ＰＡＲＳＥ－Ｇのシステム構成を示す。

5．構造エディタに適した言語仕様の記述形式

従来のコンパイラ生成系とは異なり構造エディタ自動生成システムＰＡＲＳＥ－Ｇには、構造エディタがその機能を効率良く実行するための特別な記述形式が必要である。本章では、構造エディタに適した言語仕様の記述形式について述べる。

ＰＡＲＳＥの諸機能のうち、対象言語に依存した機能を列挙する。
(1) 文法チェック機能：ユーザの入力するソーステキストを即時に構文解析し文法誤りの検出を行う。

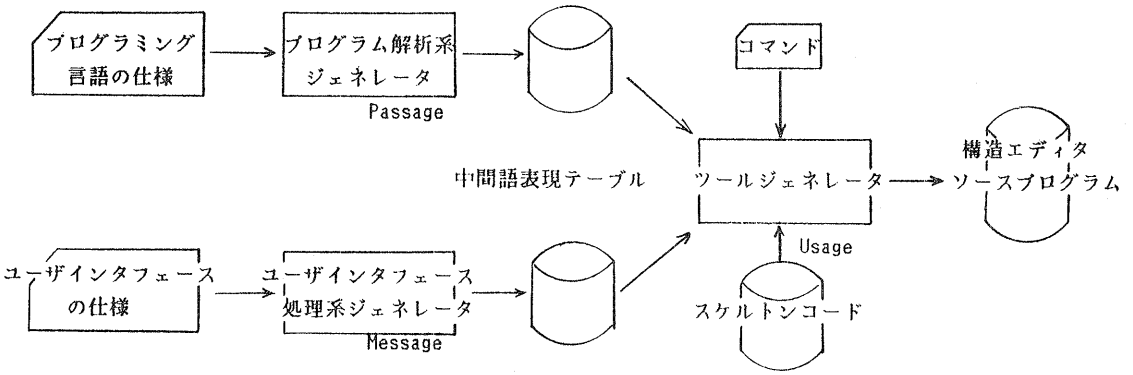


図3　構造エディタ自動生成システムＰＡＲＳＥ－Ｇのシステム構成

解析対象はプログラム全体ではなく、毎回入力されるプログラムの断片である。

(2) 構文誘導機能：プログラミング言語の構文構造を自動表示し、穴埋め方式により入力を誘導する。また入力可能な構文をメニューとして表示し、メニュー選択による入力を可能としている。

(3) 構文木生成機能：入力されたプログラムを構文解析し構文木を作成する。ＰＡＲＳＥはこの構文木に対して編集作業を行う。

(4) プリティプリント機能：構文木を画面に表示する際、インデンテーションや欄揃え等を行う。

以下の節では、これらの機能を効率良く実現するためＰＡＲＳＥ－Ｇに必要な言語仕様の記述形式を説明する。

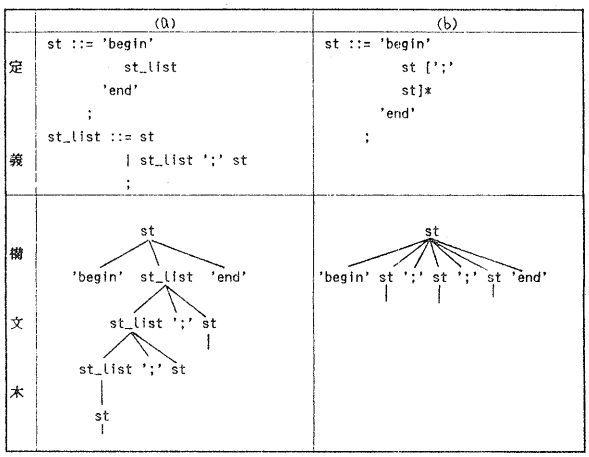


図4　構文規則とその構文木

5．1　構文規則の定義形式

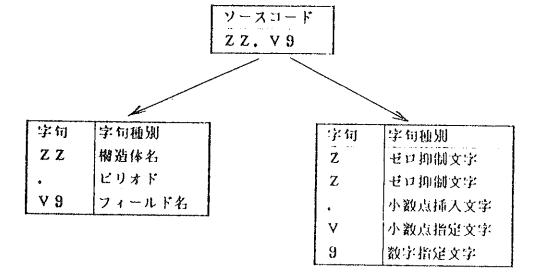
ＰＡＲＳＥはソースプログラムを構文解析して構文木を生成し、この構文木に対して各種編集操作を行う。この構文木形式は、ＰＡＲＳＥでの編集操作の効率を考慮して構文規則に準拠したものとしている。ところでＢＮＦにより構文要素の列(繰返し)を表現する場合には、通常再帰的なＢＮＦを用いる。このＢＮＦに基づいた構文木は図4(a)に示す形となる。構造エディタでは、繰返し可能な構文要素の挿入・削除処理が頻繁に行われるが、図4(a)の様な木では、構文要素(例えばst)の挿入／削除に際して部分木全体(st-list)の作り直しが必要となり、極めて効率が悪い。

この問題は、図4(b)に示す様に、繰返しとなる構文要素を木の同レベルに並べる形式の構文木を用いる事により解決できる。ＰＡＲＳＥ－Ｇではこの繰返し型の構文規則を記述する方法として、0回以上及び1回以上の繰返しを表す超記号を導入し、効率的な編集操作ができる構文木の生成を可能にした。

5．2　字句規則の定義形式

字句解析では、入力文字列を字句規則に従ってトークンに切分けるが、この切分け方は構文解析過程に動的に変わる事がある。例えばＰＬ／Ｉでは、図5に示す様に同一の文字列を式とみるかピクチャとみるかにより切分け方が変わる。これに対処する方法として、複数のスキャナを用意しておき、構文解析実行時に適切なスキャナへの動的切換えを行う方法が考えられる。Ｌｅｘ／Ｙａｃｃ等のコンパイラジェネレータでは、構文解析時のアクションとしてスキャナ切換えを実現できる。しかしこの方式には次に示す問題がある。

(1) 切換えのタイミングは、仕様定義者自身が指定



(a)「式」として解析する場合　　(b)「ピクチャ」として解析する場合

図５　字句の切り分け方の異なる例

する必要がある

(2) 各トークン毎にどのスキャナに属するかを定義しなければならない

そこで本システムでは、切分け方が同種の字句集合を１つの字句クラスと考え、この集合に対してクラス定義を行う。そして各クラス毎に個別のスキャナを生成し、構文解析実行中異なるクラスの解析を開始する時点に自動的にスキャナを切換えられるようにした。図５の例では、ピクチャという構文要素に対してクラス定義をするだけでよい。

５．３　誘導仕様の定義形式

ＰＡＲＳＥは、構文規則上入力可能な構文をメニューとして表示し、その中の１項目を選択させる事により入力誘導を行う。しかしながら構文規則そのものを用いて誘導を行った場合、選択ステップ数が多くなる可能性がある。例えば図６(a) に示す構文規則では、実行文が入力できる状態でif-then-else文を入力したい場合には、先ずif文を選択した後if-then-else文を選択しなければならない。

このような煩しさは、if文に対し構文規則を適用した展開形（図６(b)）を用いる事により解決する。そこで本システムでは、誘導ステップに用いる構文記号を別途定義できるようにし、誘導時にはこれらの記号のみを用いた展開形の構文規則を使用することとした。

６．ユーザインタフェースのモデル

本章では、構造エディタＰＡＲＳＥのユーザインタフェースを形式的に記述する方式について述べる。

ＰＡＲＳＥの諸機能のうち、ユーザインタフェースに関係する機能は次のものである。

(1) コマンド解析機能：ユーザが入力するコマンドを構文解析し、文法誤りの検出を行う。

(2) コマンド入力誘導機能：次に入力可能なコマンドあるいはオペランドを誘導し、入力を促す。

(3) コマンド処理機能：(1) あるいは(2) によりコマンドが正常に入力された場合に、そのコマンドを実行する。

(4) 画面制御機能：コマンド実行後に出力されるメッセージを、画面上のあらかじめ定められた位置に出力する。

ユーザインタフェースを定義することは、これらの機能を形式的に記述することである。われわれは、これらの機能を記述するために必要な基本要素を抽出するとともに、ユーザインタフェースも言語仕様の記述に用いたＢＮＦ形式と類似の形式で記述可能とした。

```
実行文 ::= 代入文
         | if文
         | for文
if文   ::= if-then文
         | if-then-else文
```

(a) 構文規則

```
実行文 ::= 代入文
         | if-then文
         | if-then-else文
         | for文
```

(b) if文の展開を施した構文規則

図６　構文規則と構文誘導

６．１　対話の基本要素

ユーザとシステムの対話を構成する基本要素を次のように分類する（図７）。

(a) 指令:ユーザがシステムに指示すること

(b) 応答:システムがユーザに表示すること

(c) 実行:システムがコマンドを処理すること

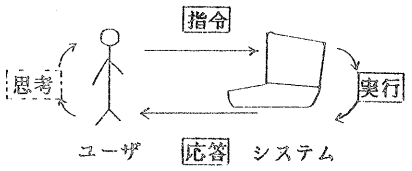


図７　対話の構成要素

図8に、基本要素の組み合わせによる対話の実現例を示す。

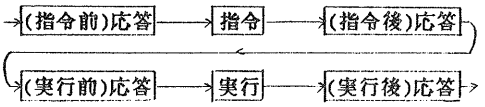


図8　ユーザとシステムの対話の順序

これらの基本要素を用いれば、先に示したユーザインタフェースに関する機能は次のように解釈することができる。

(1) コマンド解析機能：複数の指令が連続して入力されるとき、それぞれをプログラミング言語の字句とみなし構文解析を行う。

(2) コマンド入力誘導機能：構文上次に入力可能な指令の直前に存在する応答の集合を、ユーザに提示すること。

(3) コマンド処理機能：実行すべきアクションルーチン（実行）を起動すること。

(4) 画面制御機能：応答を発行すること。

6．2　対話仕様の記述形式

上記のモデルに対応する対話仕様Gは、次の4項組で表現できる。

$G = \{ P, B, V_N, V_T \}$

ただし、Pは生成規則の集合、Bは開始記号、V_Nは非終端記号の集合、V_Tは終端記号の集合とする。すなわち、V_Tは指令、応答、実行を表現する記号で、V_NとBとは、対話の順序すなわちPを段階的に記述するための記号である。このとき、指令の入力方式と応答の出力方式、および実行の方式は、V_Tがもつ"値"と考えることができる。以下その詳細を述べる。

(1) 指令の仕様

指令を表す記号$t \in V_T$に対して、次の入力方式を指定する。

1)テキスト：文字キーで入力する方法
2)ボタン　：特殊キーで入力する方法
3)ポイント：ポイント装置で入力する方法

(2) 応答の仕様

応答を表す記号$t \in V_T$に対して、次の操作（出力方式）を指定する。

1)ロード　　：画面の上に、画面を載せる
2)アンロード：画面の上の画面を取り去る
3)代入　　：画面に書き込む

ここで、画面形式の定義方法について述べる。

画面仕様Sは、次の4項組とする。

$S = \{ R, G_S, G_W, G_f \}$

G_Sは、画面にロード／アンロードされる単位の集合、G_fは$s \in G_S$上の論理的最小区画の集合で、$f \in G_f$には値（文字列）を代入可能である。G_WはG_fをグループ化するための記号であり、fの列からなる。画面構成RはG_S、G_W、G_fを用いて、段階的に記述する。図9に例を示す。

さきに述べた出力方式のうち、1)と2)はG_S、3)はG_fを用いて記述する。尚、G_fはそれが持つ"値"として、画面の大きさ、表示文字、表示強度、アクセス方法などが指定できる。

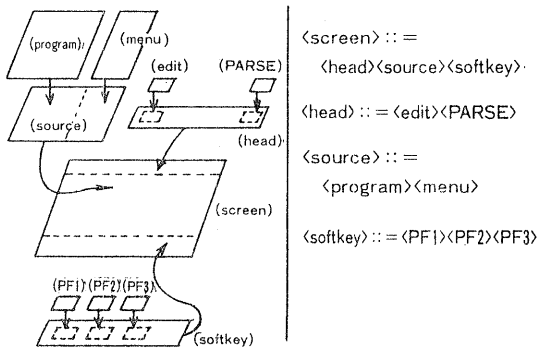


図9　画面形式の定義例

(3) 実行の仕様

実行を表す記号$t \in V_T$に対して、実行の方式すなわちコールすべきモジュール名とインタフェース（引数）を指定する。

(4) 対話の仕様

指令、応答、実行を表す記号$t \in V_T$と、tとnをまとめた記号$n \in V_N$、および開始記号Sを用い、言語の構文規則同様にＢＮＦで記述する。

7．構造エディタ自動生成システムの機能

7．1　仕様記述と仕様解析機能

(1) プログラミング言語の仕様

a) 字句の仕様：字句の構成を正規表現で記述する。字句が予約語か否か、コメントの開始・終了記号、字句構成の動的変更なども指定する。

b) 構文の仕様：構文をＢＮＦを拡張した形式で記述する。優先度・結合性、構文上コメントの記述できる位置などが指定可能である。

c) 意味の仕様：字句と構文では記述できない意味仕様は、アクションルーチンを用いて手続きで記

述する。

d) 解析の仕様：字句・構文・意味解析中に発生するエラーを処理するため、誤り検出時の修復処理方法、およびエラーメッセージを指定する。

e) 誘導の仕様：言語の構文誘導時に、構文要素が、誘導対象となるか否かを指定する。また、言語の意味上、定義と参照の関係にある構文要素で、参照側から定義側の実体を入力誘導する場合に、それらが定義参照関係にあることを指定する。

f) 表示の仕様：プログラムの画面への表示形式を、インデンテーションコマンドを用いて指定する。コマンドには、改行指定、字下げ指定、絶対欄位置指定などがある。

この仕様記述を入力として、Passage （図３）は次の解析を行い、中間表現テーブルを出力する。

a) プログラミング言語の文法の無矛盾性と、LALR(1) パーサによる構文解析の可能性

b) 表示仕様の無矛盾性と、アンバースの可能性

c) 誘導手順の無矛盾性

(2) ユーザインタフェースの仕様

コマンドは１個以上の指令、画面は１個以上の応答から構成する。

(2-1) コマンドの仕様

a) 字句仕様：指令の入力形式を指定する。

b) 構文仕様：構文をＢＮＦ拡張形式で記述する。

c) 意味仕様：アクションルーチンで記述する。

d) 解析仕様：誤り検出時の処理を指定する。

e) 誘導仕様：指令前に表示する応答を指定する。

f) 実行仕様：実行前後に表示する応答を指定する。

(2-2) 画面の仕様

a) field の仕様：$f \in G_f$ の応答形式として大きさ・表示文字・表示強度・アクセス方式を指定する。

b) screenの仕様：$s \in G_S$ を構成するｆを指定する。他にｓが透明か否かを指定する。

この仕様記述を入力として、Message （図３）は次の解析を行い、中間表現テーブルを出力する。

a) コマンド言語の文法の無矛盾性と、LL(1) パーサによる構文解析可能性

b) 誘導手順および画面構成の無矛盾性

７．２　生成機能

Usage （図３）は前述のテーブルを入力とし、次の方式で構造エディタＰＡＲＳＥを出力する。

(1) ソースプログラムの自動生成

指定された機種用のＩ／Ｏパッケージを取り込み、構造エディタのソースプログラムを出力する。

(2) 分割生成

(1) で、ユーザインタフェース処理系とプログラム解析系とを別々に生成できる。

８．おわりに

以上、言語適応型プログラミング環境の構造エディタＰＡＲＳＥと、それを自動生成するシステムＰＡＲＳＥ－Ｇの設計思想と機能について述べた。

既に、Pascal用、Ｃ用、マイコン用高級言語Super-PL/H用、同PL/M-86用などの構造エディタを生成し、自動生成システムＰＡＲＳＥ－Ｇの有効性を確認した。

参考文献

1) Proceedings of ACM-SIGPLAN Symposium on the Ada Programming Language, SIGPLAN Notices, Vol. 15, No. 11(1980)

2) T. Teitelbaum et al.: The Cornell Program Synthesizer: a Syntax-directed Programming Environment, CACM, Vol. 24, No. 9, pp. 563-573(1980)

3) P. Medina-Mora et al.: An Incremental Programming Environment, IEEE Trans. Software Eng., Vol. SE-7, No. 5, pp. 472-482(Sep. 1981)

4) T. Chusho et al.: A Language-adaptive Programming Environment Based on a Program Analyzer and a Structure Editor, 9th World Computer Congress, IFIP'83

5) 中所　他：言語適応型プログラミング環境の設計思想、情報処理学会第27回全国大会, PP. 525-526 (昭58-10)

6) 田中　他：言語適応型プログラミング環境における構造エディタPARSE の機能と実現方式, 同上, PP. 527-528

7) E. W. Dijkstra: Notes on Structured Programming: Structured Programming, Academic Press, London and New York, PP. 1-82(1972)

8) 田中　他：言語適応型プログラミング環境用マンマシンインタフェースとしての構造エディタPARSE, 情報処理学会ソフトウェア工学研究会34-8 (1984)

9) 田中　他：ICASにおける構造エディタ自動生成システムの概要、情報処理学会第30回全国大会, 3S-5(昭60-3)

10) 田中　他：構造エディタジェネレータPARSE-Gにおけるマンマシンインタフェース仕様の定義機能、情報処理学会第31回全国大会、1F-8(昭60-9)

11) 岸本　他：構造エディタジェネレータPARSE-Gにおけるプログラミング言語仕様の定義機能、情報処理学会第31回全国大会、1F-9(昭60-9)