コンパイラを支援する構造エディタについて

5G-3

有田 隆也　永松 礼夫　森下 巖
(東京大学 工学部)

1．はじめに

プログラムの作成と実行形式への変換を統一的に考えた場合、エディタがコンパイル処理の一部を実行することはプログラムの開発効率の向上につながることが期待できる。

コンパイラを支援する構造エディタに望まれる機能およびその実現方式について、試作した生成系に基づいて述べる。

2．概要

ここで提案する構造エディタは通常のエディタのように文字列であるテキストを出力するのではなく、構文規則に直接対応する解析木から冗長な部分を取り除いた抽象構文木と名前やデータに関する情報を出力するものである。

コンパイルの前半の処理である字句および構文解析は、従来の構造エディタによくみられるテンプレート式の入力時に、上から木を作っていくことにより、比較的容易に置換することが可能である。ここでは、テンプレートを選択した時点で直接抽象構文木を作成する処理をしている。

しかし、そのような入力方法だけですぐれたテキストエディタの高い操作性や編集性を実現するのは困難であると思われる。そこでテンプレート選択式の入力とスクリーンエディタ式の入力を、生成する内部木構造表現の統一性も含めて、うまく融合することが必要となる。そこで、トップダウンな木の作成とボトムアップな木の作成を、プログラムの入力の際の任意の段階で選択することのできる構造エディタを考えた。

構造エディタの対象とする言語に関するデータは、字句規則、構文規則、および付随情報を、ジェネレータに入力することにより得られる。扱う文法はＬＡＬＲ(1)であり、演算子の優先順位と結合性の記述を付加することができる。

エディタで入力および編集中は常に、その時点で副木作成の目標とする構文構成要素(ノード)名が画面最下部に表示されており、また同時にそのときに選択可能なテンプレート候補が画面下部に表示される。その中から１つを機能キーによって選ぶとそれがプログラム編集領域に開かれる(図１)。

テンプレートを選ばずに直接キーボードからテキスト入力を開始することもできる。その際、特に後で具体的に入力するために、構文要素を表す特別の終端記号を埋めこんでおいてもよい。テキスト入力が終わると即座に解析が行われる。

木構造を直接意識して編集するようなコマンドは，なるべく取り除いている。カーソルは２文字以上の幅をもつことがあり，それにより副木のノードを指し示す。その範囲内で，従来のスクリーンエディタ的な編集をおこない、編集終了時にパースしなおす。

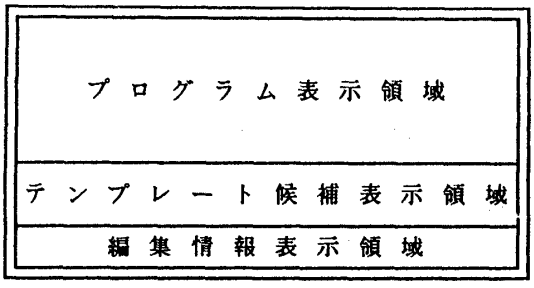

図１．編集時の画面構成

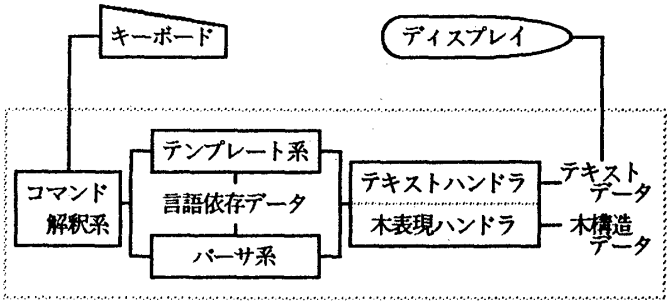

図２．内部構成

A Syntax Directed Editor Assisting Compilers
Takaya ARITA, Leo NAGAMATSU and Iwao MORISHITA
University of Tokyo

3．実現方式

エディタの内部構造を図２に示す。自由なプログラム表記を認めることおよび逆パース処理を軽減することを考え、プログラムはテキストと抽象構文木の２つの形式で保持するようにした。抽象構文木の各ノードは図３に示すようにノードの種類、テキストへのポインタ、テキスト上の長さ、右あるいは親のノード番号、そして一番左の子のノード番号あるいはデータディスクリプタから構成される。ここでの抽象構文木は構文規則に対応する解析木から、

ａ）子が１つの非終端記号のノード

ｂ）冗長なリスト表現

ｃ）予約語の終端記号のノード

を除去したものである。ｂ）は繰り返しによる冗長な縦方向の木の伸びを避けるためのものである。

テンプレートを選択した場合や、テキスト入力を終了する場合に、抽象構文木表現のノードを段階的に作成していく。

パーサ系に関しては、各非終端記号に対応した特別の終端記号を付加する手法[1]をとり、ＬＲパーサ自体の変更を避けた。例えば、

```
START -> #statement STATEMENT
EXPRESSION -> #expression
```

などのように、特別の終端記号を含む生成規則を付加することにより、前者では文１つだけの部分構文解析が可能となり、後者では式の内容を決定していない段階でのパーシング、つまり段階的な詳細化を支援する構文解析を可能にした。なおここで、"START"は開始記号、"STATEMENT"は文、"EXPRESSION"は式を表す非終端記号、#で始まる記号はそれぞれの非終端記号に対応する特別の終端記号を意味している。

4．評価

ＵＮＩＸマシン上で言語に依存するデータの生成系を、パーソナルコンピュータ上で構造エディタ本体を、Ｃを用いて開発し，データを生成してダウンロードした後に実行した(図４)。

ＰＬ／０を対象とする構造エディタを試作し、動作の確認と評価をおこなった。部分構文解析のために付加した規則のために33％、段階的詳細化支援のために付加した規則のために14％、両者の場合約50％のパーサの状態数が増加した。支援のための付加ハードウェア[2]の使用を想定した場合は、使用する連想メモリのワード数は約65％増加する。特にデータ量(メモリ量)が制約される環境のもとでは、適用が可能な非終端記号を限定することが考えられる。

5．まとめ

対象とする言語としてＰＬ／０を選び、所期の動作の確認を行ったが、より実用的な言語での実現を進めている。コンパイルの意味的な処理への対応、さらに実行系をも含んだ検討が今後の課題である．

参考文献

[１]　J.R.Horgan, D.J.Moore, " Techniques for Improving Language-Based Editors", ACM SIGSOFT/SIGPLAN Softw. Eng. Symp. on Practical Software Development Environments,pp.7-14,1984.

[２]　有田、永松、森下：字句および構文解析用コプロセッサについて、情報処理学会第３２回全国大会、pp.567-568、1986.

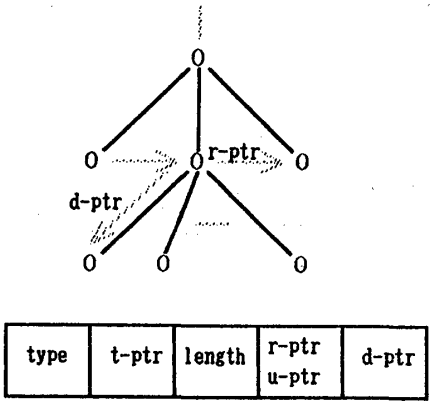

図3．ノードの内部表現と概念

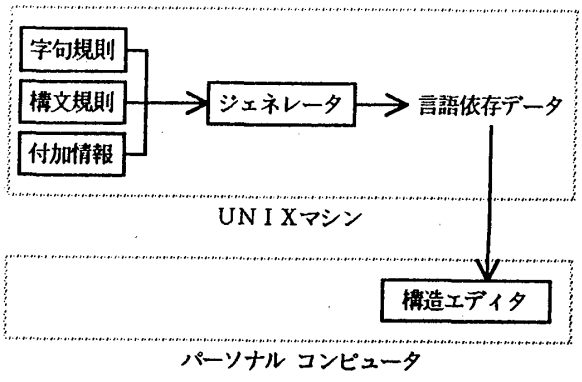

図4．実行環境