3E-1

C-Prolog コンパイラの開発（１）

――― 設計思想について ―――

松本 憲幸，小林 茂，落合 正雄，本位田 真一
㈱東芝

1．はじめに

エジンバラ大学で開発されたC-Prologについて、スーパーミニコンをターゲットとしたコンパイラを開発したので報告する。本コンパイラの設計に当たっては、処理の高速化とともに言語の一貫性を重視し、コンパイルの前後でプログラムの動作が可能な限りインタプリタと同一となり、かつ処理の効率を極力低下させないような方式を採用した。ここではC-Prologに対するコンパイラとしての設計思想について述べる。

2．C-Prologコンパイラの特徴

今回開発したコンパイラの特徴を以下に挙げる。

① C-Prologインタプリタ環境下で組込み述語 "compile" により起動される。

② コンパイルされた述語（オブジェクト述語と呼ぶ）は機械語に落とされるが、コンパイルされない述語（ソース述語と呼ぶ）と同様に、インタプリタの下で管理される。

③ 述語の動的な書換えの手段を与える等、インタプリタと高い親和性を持つ。(dynamic宣言)

④ インタプリタ機能の明示的な制限の宣言による高速化オプションを持つ。(local宣言等)

3．コンパイラの位置づけ

本コンパイラは、C-Prologインタプリタ環境下でプログラムを高速に実行するための副機能として位置づけられる。物理的にもコンパイラ機能の一部とオブジェクト述語管理機能をインタプリタに組み込んだものとなっており、一体化されたC-Prolog処理系という形態をとっている。コンパイル機能をインタプリタの内部処理と別タスクとしての外部処理に分離した理由は５章で述べる。

4．コンパイラの処理方式

コンパイラの操作性、およびインタプリタによる実行との動作の同一性という観点から、以下のような問題点の抽出およびその処理方式の決定を行なった。

4．1 コンパイル単位

コンパイラに対して、コンパイル対象をどのように指定するかということは、コンパイラの操作性を決める重要な要素である。指定の与え方としては、

① その時点でインタプリタ内に定義されている全述語をコンパイルする。

② 述語単位で指定する。

③ ソースファイル単位で指定する。

等が考えられる。ここでは③を標準とした。ソースファイル単位での指定が可能であるということは、①に比べ、デバックの終了した部分からコンパイルしていくことにより、効率よくプログラム開発が行なえるという利点がある。また、②に比べ、大規模なプログラムの開発時に実際的である。

```
compile （ファイル名orファイル名リスト）
```

図１コンパイル指定形式

4．2 ソースファイルの解釈

ファイルをコンパイル単位にするということは、指定された個々のファイルを各々静的に解釈するという意味ではない。一般にソースファイルにはその構文解析に必要な全ての情報を含んでいるとは限らない。インタプリタはソースファイル中に含まれる演算子定義や他のファイルの読込み指示等を実行しながら、動的にファイルの内容を解釈する。このようなソースファイル内容の解釈問題に対する完全な解決策として、本コンパイラはその第１フェーズ（ソースファイル解析フェーズ）において、インタプリタのファイル読込み機能を利用して一旦prologデータベース上に、インタプリタの解釈したソースファイルの内容を展開し、その内部表現を第２フェーズ以降のオブジェクト生成タスクに渡す方式とした。

4．3 述語のスコープ

あるファイルで定義された述語がどのようなスコープを持つかということは言語仕様に係る問題である。C-Prologのインタプリタが唯一のグローバルな述語参

An implementation of C-Prolog Compiler (1)
N.Matsumoto, S.Kobayashi, M.Ochiai, S.Honiden
TOSHIBA CORPARATION

照のスコープを持つのであるから、コンパイラも同様なスコープを持たなければならない。（４．２）で採用したファイル解釈方式によってこのスコープは保証される。

但し、ファイルを一つのモジュールと考え、ある述語の定義がそのファイル内でのみ有効であるとする（ローカルスコープ）解釈はインタプリタとの整合性を崩す一方で、大規模システムの構築を容易にする。コンパイルすることにより階層化されたスコープを持つことができるという発想は言語仕様の上では適当でない。しかし、我々が導入した高速化オプションのlocal 宣言はローカルスコープという副作用を生じる。

４．４　述語定義の修正

ソース述語は、組込み述語assert/retractにより節単位で述語定義を修正することができる。しかし、オブジェクト述語はその定義内容全体が一連の機械語となる為、一般には節単位での修正を禁止する。インタプリタとの整合性の為に節単位での修正を必要とする述語は、コンパイラに対する宣言によって例外的に認められる。

```
dynamic （述語名or述語名リスト）
```

図２．dynamic 宣言の形式

この述語は、指定された述語を実際にはソース述語として取込み、コンパイルしないことをコンパイラに通知する。

４．５　述語の再定義（追加・更新）

C-Prologにはソースファイルの内容により、述語に節の追加を行なうconsult と、述語の再定義を行なうreconsult がある。オブジェクト述語(dynamicでないもの）では前述のように定義の部分的な修正が困難な為、追加型の変更は許さず警告を伴って再定義及いとする。

既定義　新定義処理	consult	reconsult	compile
ソース述語	追加	再定義	再定義
オブジェクト述語	再定義（警告）	再定義	再定義

図３．述語の再定義

ある述語が再定義された場合、これを参照していた述語（ソース述語、オブジェクト述語共）は新定義を参照するようになる。

４．６　述語の呼出し

dynamic 宣言の導入により、あるソースファイル中の内容がソース述語とオブジェクト述語の混在した形となる為、相互間での呼出しを可能としなければならない。また、これによってプログラムを部分的にコンパイルしてもインタプリタと同程度に対話的な実行が保証される。

５．コンパイラの構成

コンパイラはＣ言語で記述し、インタプリタにその機能の一部（第１フェーズ）を含み、第２フェーズ以降を別タスクとした。これはインタプリタの実行時サイズを小さくし、かつ（４．２）の実行時環境の解析を容易に行なうことができるという利点を持つ。コンパイル処理の流れを図４に示す。

６．おわりに

ここで述べた設計思想に基づいたコンパイラはインタプリタと高い親和性を持ち、その処理がインタプリタ実行と異なる可能性が生じた場合はエラーまたは警告を生じる。また、エラー・警告を受けた述語のdynamic 宣言によりその処理を完全にインタプリタと整合させることが可能となる。

参考文献

[１] 小林　他：C-Prologコンパイラの開発（２），第３３会大会予稿集，３Ｅ－２，（１９８６）

[２] Quintus Computer Systems, Inc:Quintus Prolog User's Guide （１９８５）

図４．コンパイル処理の流れ