3Y-1 ポータブルなＰｒｏｌｏｇコンパイラーの実現

丸山圭一　濱利行　江藤博明　小松秀昭　浅川康夫　大場充
日本アイ・ビー・エム株式会社　東京基礎研究所

はじめに

我々は、実用的なPrologを研究している。研究は、言語の整理・拡張、抽象化マシンの改良及びそれらに基づいたコンパイラ・システムの構築などからなる。

今回、コンパイラ・システムを複数の機械、ＯＳ（IBM3081KのCMS,IBM RT/PCのAIX並びに4.3BSD)に移植したので、それについて報告する。

ポータブルなシステムを構築する際の問題点

一般にシステムの移植性を高めるには、できるだけ機械に依存した部分を排除し、機械から独立にシステムを構築すればよい。このことは、移植性の高い高級言語を用いれば、ほとんど実現できる。多くの場合、問題になるのは、Ｉ／Ｏと文字コードである。我々のアプローチもこれに沿ったもので、システム全体を我々のＰｒｏｌｏｇとＣ言語で記述し、Ｉ／Ｏと文字コードについては、特別に配慮した。

Ｐｒｏｌｏｇシステム

今回構築したＰｒｏｌｏｇシステムは、WAM (Warren's Abstract Machine)を基に修正・拡張を行った抽象化マシンＡＰＭ (Abstract Prolog Machine)のエミュレータ上で動作するものである。

システムの構成要素は、以下のものである。

コンパイラ
ＡＰＭ　アセンブラ
リンカ
ＡＰＭ　エミュレータ

コンパイラは、Ｐｒｏｌｏｇのソースを ＡＰＭ命令にコンパイルするものである。ＡＰＭ　アセンブラは、ＡＰＭ命令を対応するバイト列に変換するものである。リンカは、こうしてできたＡＰＭオブジェクト・ファイルをリンクするものである。ＡＰＭエミュレータは、リンカの出力したロード・モジュールを実行するものである。

コンパイラ、アセンブラは、我々の作ったＰｒｏｌｏｇ自身で記述されている。リンカ、エミュレータは、Ｃ言語で記述されている。

Ｐｒｏｌｏｇ自身で記述されたコンパイラ・システム

我々のＰｒｏｌｏｇは、ソース・レベル、ＡＰＭ命令・レベル、ＡＰＭバイトコード・レベルでポータブルである。そして、そのＰｒｏｌｏｇ自身でコンパイラ・システムが記述されているので、リンカ、エミュレータさえ移植できれば、バイトコードまで変換されたコンパイラ・システムを持ってくることによって、アプリケーション・プログラムのコンパイル、アセンブル、実行を一貫して行えるＰｒｏｌｏｇシステムの移植ができたことになる。このような特長を持ったコンパイラ・システムを構築することによって、我々が提案する実用的なＰｒｏｌｏｇの言語仕様の整理・拡張が妥当なものであることを間接的に実証しようとしている。

システムの構築で鍵となったＰｒｏｌｏｇの特長

我々のＰｒｏｌｏｇの言語仕様の詳細については、他の文献([1],[2],[3],[4],[5])にゆずるとして、ここでは、コンパイラ・システムの構築において有効であった特長について簡単に述べる。

１．パッケージ・システム　大規模なプログラム開発においては、全体を複数の部分に分割して開発する方法が非常に有効である。例えば、部分間のインタフェースだけを定めて何人かのプログラマで分担して開発したり、部分ごとに独立にデバッグやコンパイルすることが可能になる。特に、独立コンパイルが可能な場合、よく使われる述語をあらかじめコンパイルしておいてライブラリとして利用することもできる。

２．テーブル　実用的なプログラムを書こうとすると、大域的なデータを扱えることが望ましい。このような場合、従来はassertを用いていた。しかし、本来、プログラムを変更するために用意された述語であり、好ましくない。そこで、大域データ処理機能を副作用を伴う配列として導入している。

システムの構築で鍵となったＡＰＭ命令の特長

我々の構築したＡＰＭ命令セットは、Ｐｒｏｌｏｇから ＡＰＭオブジェクトへコンパイル、アセンブルする際の中

Portable Prolog Compiler
Keiichi Maruyama, Toshiyuki Hama, Hiroaki Etoh, Hideaki Komatsu, Yasuo Asakawa, Mitsuru Ohba
IBM Research, Tokyo Research Laboratory

間表現という消極的な意味ばかりでなく、それ自体が独立した言語として完結性を有している。つまり、Ｐｒｏｌｏｇをコンパイルしても出力されないような低レベルの命令も多数、用意されている。Ｐｒｏｌｏｇで用いられる組込み述語は、できるだけ拡張Ｐｒｏｌｏｇで記述し、パッケージとして提供した。Ｐｒｏｌｏｇで記述できない部分は、このＡＰＭアセンブリ言語で記述した（これもパッケージとして扱われる）。

移植の作業

移植の作業は、大きく分けて２つになる。我々のＰｒｏｌｏｇ自身で記述されたコンパイラとＡＰＭアセンブラをブートストラップすることと、Ｃで記述されたリンカとＡＰＭエミュレータを目的の機械に移植することである。

ａ．ブートストラップ　以下の方法でブートストラップを進めた。

１．比較的簡単な変換でＶＭ／Ｐｒｏｌｏｇ(IBM3081K上で動作するインタプリタ型のProlog)で実行可能になる拡張Ｐｒｏｌｏｇのサブセットを決めた。

２．そのサブセットをＶＭ／Ｐｒｏｌｏｇに変換・コンサルトするツールをＶＭ／Ｐｒｏｌｏｇで記述した。コンパイラ内の各フェーズ間のデータの受け渡しは大域データ処理機能([5])を使うが、この機能は変換・コンサルト・ツールによってＶＭ／Ｐｒｏｌｏｇのaddax/delax/ax(DEC-10 Prologにおけるassert/retract/clause)に置き換えられる。コンサルト・ツールではパッケージ内およびパッケージ間の述語参照などのチェックも行っている。

３．コンパイラを３２のパッケージ、約５０００行で、ＡＰＭアセンブラを１１のパッケージ、約３０００行で記述した。

４．生成されたバイトコードをリンクし、エミュレータ上で動作させてテストした。最後にコンパイラ・システム自身をコンパイル、アセンブルできるかどうか確かめた。

ｂ．リンカとＡＰＭエミュレータの移植　Ｃ言語で記述したため、移植性が高いのは言うまでもない。具体的には、IBM3081KのCMSとIBM RT/PCのAIX並びに4.3BSD上に移植した。

Ｉ／Ｏについて

システムの移植性にＩ／Ｏが大きく影響する。我々は、今回移植した機械やＯＳ及び今後移植する予定のものを検討し、ポータブルなＩ／Ｏ組込み述語を定義した。具体的には、次のものである。

```
open(FileSpec, Option, FileDesc)
close(FileDesc)
read(Term, FileDesc)
write(Term, FileDesc)
```

openは、ファイルのオープンである。FileSpecでファイルを指定し、ファイル・ディスクリプターを受け取る。Optionでは、様々なオプションを文字列の形式で指定できるが、コンパイラ・システムの記述には、機械やＯＳに依存しないものしか使っていない。closeは、ファイルのクローズである。read,writeは、それぞれ項の読みこみと書きこみである。

文字コードについて

IBM3081KはEBCDICコードを用い、IBM RT/PCはASCIIコードを用いている。従って、ＡＰＭオブジェクトは文字を扱う部分については、２つのコード体系で異なる。我々のコンパイラ・システムでは、自身のコード体系と異なるＡＰＭオブジェクトをオプション指定することによって出力できるようにした。こうすることによって、コード体系の違う機械で生成されたオブジェクトと混在させてリンクすることができ、プログラミング開発並びに実行環境の自由度を大幅に向上させることができた。

おわりに

Ｐｒｏｌｏｇコンパイラ・システムを我々のＰｒｏｌｏｇ自身で記述することによって、ポータブルなシステムを構築し、実際に複数の機械やＯＳに移植した。また、その際に問題となるＩ／Ｏと文字コードについて、その解決法を示した。今回の作業を通じて、我々の提案するＰｒｏｌｏｇの言語仕様の整理・拡張が妥当であったことを確認することができた。

参考文献

［１］浅川ほか、"実用的な使用を目指したＰｒｏｌｏｇコンパイラ・システム"，情報処理学会第３５回全国大会、１９８７

［２］江藤ほか、"Ｐｒｏｌｏｇへのパッケージ・システムの導入"，The Logic Programming Conference '87

［３］江藤ほか、　"プロローグとパッケージ・システム　－　ネームスペースの取扱いについて"，情報処理学会第３５回全国大会、１９８７

［４］濱ほか、"Ｗａｒｒｅｎ'ｓ　Ａｂｓｔｒａｃｔ　Ｍａｃｈｉｎｅの拡張　－　新しい制御機構の実現"，情報処理学会第３５回全国大会、１９８７

［５］小松ほか、"プロローグにおける副作用を伴う配列の導入とその効率的実現について"，情報処理学会第３５回全国大会，１９８７