3Y-9

モジュール構造を持つプロローグのデータベースインターフェース

江藤博明　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

我々は現在、実用的なプロローグの処理系を目指して研究、開発を行なっている［1］。この処理系はプログラムを複数の部分に分割して開発するためのモジュール化機能を備えている。またモジュール管理用の様々な機構が導入されている。今回はこの処理系において大規模なデータをも扱えるように言語拡張したことについて報告する。

DEC-10 Prolog では、プロローグ内部データベースを扱うために assert/retract 述語を提供している。しかし、このデーターベースはプロローグのワークスペースを消費するため大規模なものを扱うことはできない。さらに読出し専用のデータベースの利用時にもワークスペースを消費するためメモリの利用量を低下させることになる。

一般に大規模なデータを扱うときにはファイル等の二次記憶を利用するのが好ましい。プロローグのデータ読み出し方式はパターンマッチングによる項の取り出しである。したがって二次記憶としてはインデックスサーチ等の可能な外部データベースが良い。

ここではＳＱＬ等の外部データベースとのインターフェースを取る際のプロローグデータベースとの整合性について論じる。

データベースの導入

プロローグ処理系にデータベースインターフェースを導入するにあたって考慮すべき点について説明する。

（1）プロローグユーザーが容易にデータベースを使用できること。

関係データベースＳＱＬの他言語インターフェースは、対象言語に拘束されず、同一の言語インターフェースを与えている。たしかにＳＱＬ構文を知っているものにとっては対象言語を意識せずにデータベースを利用することができる。しかしながらＳＱＬを知らないものにとっては複雑なＳＱＬ構文を覚えなければならない。

また既存のＳＱＬ構文は手続き型言語向きに設計されたものであり宣言型言語であるプロローグには適応しがたい。

（2）プロローグの内部データベースから外部データベースへの移行が容易であること。

大規模なデータベースを扱うプログラムの開発ではモジュール単位のデバッグからシステム全体へのデバッグへと移るものである。前半のデバッグフェーズでは数十から数百のサンプルでテストするため、各サンプル間のデバッグ間隔はできるだけ短かくなければならない。そこで前半のデバッグは変更の容易なプロローグのデータベースを使用し、後半で外部データベースに切り換えられると都合がいい。この切り換えのさいのバグの混入を防ぐために、プログラムの変更はできるだけ小さくなければならない。できればデータベースを記述した部分は他のプログラム部分と明確に分離できた方がいい。

（3）モジュール化した言語仕様に適合すること。

我々が研究開発中のプロローグはモジュール化機構［1］とデータ格納機構［2］をベースに言語拡張をしたものである。この仕様によりプログラムは、

コード部
データ部
メタ記述部

の３つの部分に区別されかつ明確に記述することができるようになった。

この処理系に外部データベース用のデータ部処理機構を入れることは、データ部のアクセス方法を複雑にし繁雑な処理系になりかねない。したがってデータ部のアクセス方法は既存のものと同一視できるものでなければならない。

インターフェース仕様

論理型言語であるプロローグのデータベースインターフェースの対象には関係論理から容易に変換することができる関係型データーベース（ＳＱＬ，ＤＢ２等）が具合がいい。また関係論理によるデータベースアクセス方式を提供することはＳＱＬ等の複雑な言語仕様をプロローグ使用者は知る必要がなくなり、より良いインターフェースを与えられる。さらにプロローグ使用者にとってデータベースアクセスをプロローグのものと同等に扱うことができる。

データベースの問合せはプロローグの事実の問合せと同様に行なう。データベースのフィールド名と問合せのアーギュメントの対応は後に述べるデータベース宣言の順番で決定される。

Database interface for the Prolog with module structure.
Hiroaki Etoh, Mitsuru Ohba.
IBM Research, Tokyo Research Laboratory.

次の例は社員番号が５０００以上の社員の名前を問合せるものである。

```
employee(Name, No) &
ge(No, 5000)
```

この問合せにより指定されたデータベースから最初に適合したレコードが取り出される。このときバックトラックポイントは保持され、ｆａｉｌにより制御が戻ってきたとき次のレコードを取り出す。空のレコードが取り出されたときこの問合せはｆａｉｌする。
このようにデータベースの問合せはプロローグ使用者にとって自然に記述することができる。

本処理系ではネームスペースの分離のためにパッケージを導入した。これを利用することでファクト・データベースをプログラムから分離し、一つのパッケージに封じ込めることができる。

次の例は社員の名前と社員番号からなるｅｍｐｌｏｙｅｅというデータベースをパッケージの中に定義したものである。またパッケージｆｏｏはデータベースの使用例である。パッケージ間のデータの使用権の受け渡しはｉｍｐｏｒｔ／ｅｘｐｏｒｔ宣言により行なわれる。

```
package(bar).
    export(employee(*,*)).

    employee('etoh', 5000).
    employee('ohba', 3000).
        .
        .
endpackage(bar).

package(foo).
    import(bar:employee(*,*)).

    <- employee(Name, No) &
       ge(No, 5000) &
       write(Name).

endpackage(foo).
```

外部データベースの扱いはパッケージｂａｒから類推されるように、一つのパッケージに定義されたファクトの集まりであるとモデル化した。

外部データベースに定義されたテーブルを利用するためには、次の宣言が必要である。

```
import (database,
        データベース名（フィールド名，・・））
```

データベース名、フィールド名はそれぞれ外部データベースのテーブル名および選択すべきフィールド名を書く。

したがって、上記の例で波線の宣言を以下のように変更するだけで外部データベース中のテーブルｅｍｐｌｏｙｅｅを参照することができる。

```
import(database, employee('NAME', 'NO')).
```

以上のような仕様により「できるだけ小さなプログラム変更でプロローグ・データベースから外部データベースへと切り換えられること。」を満たすことができた。

データベースの変更

データベースの検索だけでなくデータベースの作成、データベース内のレコードの追加・削除もＳＱＬ構文を知らずに使用することができる。

```
レコードの追加・削除用ｂｕｉｌｔ－ｉｎ
  ｉｎｓｅｒｔ（データベース用の項）
  ｄｅｌｅｔｅ（データベース用の項）
```

データベースの作成は外部データベースの使用宣言の第３アーギュメントにＣＲＥＡＴＥと書くことにより、ｉｎｓｅｔ　ｂｕｉｌｔ－ｉｎ述語が使用された時、テーブルは作成される。次の例は、ｅｍｐｌｏｙｅｅデータベースを新たに作成するものである。

```
import(database,employee('NAME','NO'),CREATE).
<- insert(employee('etoh', 5000).
<- insert(employee('ohba', 3000).
```

おわりに

プロローグとデータベースとのインターフェースは、いわゆるＳＱＬ演算より理解し易い関係論理をもとに導入した。またプロローグの持つ内部データベースと外部データベースとの統一的なアクセス方法を提供した。さらにデータベース間の切り換えを容易にすることで大規模データベースを構築するためのデバッグ支援になるであろう。

現在このデータベース仕様を受け取って、プロローグと外部データベース呼び出し用のｂｕｉｌｔ－ｉｎに変換するプリプロセッサを作成中である。

参考文献

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