2Y-4

ESP上のタイプシステムの実装と並列言語への展開

小中裕喜、田中英彦
(東京大学工学部)

1. はじめに

我々は、逐次型推論マシンPSI 上の論理型オブジェクト指向言語ESP を対象として、タイプ及びモードに基づく新しいプログラミング環境TypeMasterを提案している。

論理型言語の変数や述語のタイプに関する情報をプログラムの編集やデバッグに応用することにより、プログラミング効率を向上させるのが本システムの狙いである。

今回はタイプシステムの実装上の問題点となったタイプライブラリについて述べる。ESP 上のタイプシステムの概要については [1]を参照されたい。また並列言語を対象としたシステムの開発にも言及する。

2. タイプライブラリ

2.1 タイプシステムの構成

タイプシステムの構成を図1に示す。

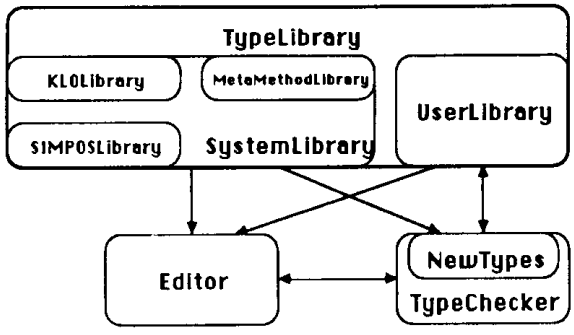

図1 TypeMasterの構成

タイプライブラリは既存の述語のタイプ及びモードやスロットのタイプの仮定やインヘリタンスの関係などの情報を含むデータベースを持つ。そしてタイプチェッカやエディタが情報を要求すると、データベースから必要な情報を検索、加工してそれらに渡す。タイプライブラリは4つの構成要素からなる。

2.2 タイプライブラリの保持する内容

userLibrary はユーザが登録したクラスに関して、インヘリタンス関係と、スロットのタイプ、述語の情報を保持している。

ESP では多重継承機能が実現されていて、メソッドの結合の優先順位の決定には depth-first-left-to-right の方式がとられている。インヘリタンスについてはそのような順序関係を保持している。

スロットに関してはタイプを保持しているが、要素スロットと属性スロットではインヘリタンスの関係で格納形式が異なる。

述語の情報はクラスメソッド、インスタンスメソッド、ローカル述語に分類される。そして各述語に対しクラス名とArity 、その述語のタイプ及びモード、その述語を定義しているクローズごとのタイプ及びモードと、クローズそのものを持っている。

なお2.3.2と関連するが、クローズのタイプ・モードは、ボディのゴールだけを手がかりに推論されている。そして述語のタイプ・モードは、それを定義するクローズのタイプ・モードからインヘリタンス・デーモンを考慮して合成されている。従ってメソッドの第1引数のタイプに、そのクラスが明示的に現われない場合もある。典型的な例を以下に示す。

```
class a has                    - Type -
 :nothing(Class);        =>    nothing(not_referred)
end.
```

klOLibraryは、KLO 組込み述語のArity とタイプ及びモードをファクトの形で保持している。

metaMethodLibrary は、new などのメタなメソッドのArity を保持している。またタイプ及びモードを導出するための特別なルールを保持しているが、それについては後述する。

simposLibrary はSIMPOSの提供するクラスに関して、メソッドのArity やタイプとモード、スロットのタイプ、インヘリタンス関係を保持している。

2.3 タイプライブラリへの情報の要求

2.3.1 エディタからの要求

エディタでは既存の述語のタイプやモードを見せるほかに、ユーザが定義した述語に関する情報をクローズ単位で見せたい。このとき例えばクローズ内の各変数やボディの述語のタイプ及びモードも見せたいが、これらは直接保持していない。従ってクローズそのものからタイプチェックアルゴリズムを用いてそれらを導出しなければならない。

2.3.2 タイプチェッカからの要求

タイプチェッカはクローズ単位でタイプ・モードを推論・チェックしていくが、その際ボディの各ゴールのタイプ及びモードを決定する必要がある。そのゴールが新たに定義されたクラスに属する述語・スロットの場合、まだそのタイプ・モードが決定されていなければ、それを先に推論するので、いずれにせよ必要な情報はタイプチェッカのワークスペースから得られる。しかしそれが既存の述語か、あるいは既存かどうかもわからないような場合は、タイプライブラリに必要な情報を要求する。以下に、述語のタイプ・モードを要求された場合に対する処理を示す。

スロットについては省略する。

まずpred(...) という形のゴールの場合、これがKLO

Implementation of Type System on ESP and Concurrent Languages
Hiroki KONAKA, Hidehiko TANAKA
Univ. of Tokyo

組込み述語であれば、klOLibraryから対応するタイプ・モードを検索してそのまま返す。ローカル述語であればタイプチェック中のクラスに属しているはずなので、必要な情報はタイプチェッカのワークスペースに存在するはずである。

次にメソッドの場合だが、これはメタなものとそれ以外のSIMPOSやユーザによるメソッドでは処理が異なる。

先に通常のメソッドについて説明する。

:pred(#classA,...)という形のゴールの場合、そのクラスメソッドのタイプ・モードを取り出す。第1引数のタイプにそのクラスオブジェクトが含まれることを確認の上、そのクラスオブジェクトに置換してから返す。

:pred(Arg1,...) というゴールの場合、タイプ推論の過程でArg1がどのオブジェクトを指しているかわかっている場合は上と同様であるが、そうでない場合は述語名とArity からsimposLibrary 、userLibrary 及びタイプチェッカのワークスペースの全領域にわたって、クラス／インスタンスメソッドに関わらず全てのタイプ・モードを収集しなければならない (*)。またそれら全てについて第1引数を、対応するクラスのクラス／インスタンスオブジェクトに（確認の上）置換しておかねばならない。

クラス名指定がある場合 (a:pred(...),:b:pred(...)) はそれに対応するタイプ・モードをそのまま返せばよい。

最後にメタなメソッドの場合だが、ESP ではメタなメソッドは特別な機構で実現されているので、metaMethod Library にはタイプ・モードの取得のための特別なルールがメソッドごとに用意されている。ここではnew について具体的なタイプ・モードの例をあげるにとどめる。

ゴールの様式	返されるタイプ
:new(#a,A)	new(#a,^($(#a)))
:new(Class,Obj)	new(α,^($(α)))
a:new(Class,Obj)	new(not_referred,^($(#a)))

2．4　userLibraryの更新

新しいクラスに対してタイプチェッカが行った処理の結果はuserLibrary に登録される。このとき場合によっては既存の情報も更新する必要がある。すなわち、新しく登録されたメソッドと同じものをボディに持っていて、そのゴールが2．3．2の(*) のケースに相当していたクローズと、その述語を直接／間接に用いている全ての述語に関して再計算を行わなければならない。

2．5　タイプ情報のその他の応用

タイプ・モードの情報はプログラム実行の最適化に大きな役割を果たす。例えば無用なタイプチェックを省略したり、通常のメソッド呼び出しをクラス名指定呼び出しに変換するなどの効率化が図れるであろう。そういった応用も検討すべき点である。

また当研究室で開発されたESP のブラウザにタイプライブラリの情報を活用することも検討している。

3．　並列言語を対象としたタイプシステム

3．1　並列論理型言語のタイプ推論

GHC や当研究室で開発されたFLENG といったような並列論理型言語を対象とする場合、モードに関してはガードなどのチェックを付加した推論アルゴリズムが必要である。

タイプに関しては、基本的にはPrologのタイプ・モード推論アルゴリズム（ESP に対するアルゴリズムはこれを包含している）を応用できる。しかしオブジェクト指向プログラミングに対応させる場合、mutable なタイプとimmutable なタイプとを区別する必要がある。また、mutable なタイプを持つ引数に注目して、メッセージ伝達の誤りがないかどうかを確認することが考えられる。

3．2　並列オブジェクト指向言語に対するシステム

並列論理型言語の上位言語として、並列オブジェクト指向言語がいくつか開発されている。A'UM、VULCANなどがその例だが、当研究室でもFLENG の上にFLENG++ を開発中である。

FLENG++ におけるクラスの管理にはライブラリが必要となり、そこにはタイプ・モードの情報も格納されることになる。プログラミング支援環境としてはESP におけるTypeMasterと同様の構成のものを検討している。

FLENG++ はFLENG にコンパイルされてから実行されるわけだが、タイプ推論を行う場合もFLENG++ の段階で行う方法と、コンパイルされた結果に対してFLENG のタイプ・モード推論アルゴリズムを適用する方法と2通りが考えられる。

前者はオブジェクト指向という面でESP のタイプ推論アルゴリズムが参考となる。タイプ推論の結果は、プログラムのデバッグを行う上で有力な情報となるであろう。またクラスの再利用性を高めるための情報としての側面も持つ。

後者の結果はより低レベルな情報であり、プログラム実行の最適化に応用するのに向いている。またタイプ・モード推論においても、コンパイル時の情報を利用できれば処理の負担は軽くなる。

4．　今後の課題

ESP を対象としたタイプシステムについては、その評価を行うとともに不備な点の改善を図っていく。

並列言語については、まずFLENG に対して、オブジェクト指向プログラミングに対応したタイプ・モード推論アルゴリズムを実装する。

次にESP 上のシステムの評価をもとにしてFLENG++ に対するタイプシステムの構築を行っていく。

その後、そのような並列言語のプログラム実行の最適化への応用についても実現を図りたい。

参考文献

[1] 小中他,"オブジェクト指向言語ESP におけるタイプチェックシステムの試作",情報処理学会第36回全国大会,2H-5,1988.

[2] "ESP 説明書（第2版）",ICOT,MP1005-2,1987.

[3] 溝他,"並列論理型言語GHCとその応用",共立出版,1987.

[4] 吉田他,"A'UM －KL1上の並列オブジェクト指向言語－",プログラミング言語,14-4,1987.

[5] K.Kahn,"Vulcan : Logical Concurrent Objects",Technical Report,Xerox Palo Alto Research Center,1986.