ISLISP処理系TISLのためのパッケージシステム

泉 信人† 伊藤貴康†

Lisp言語のISO標準であるISLISPの処理系TISLが筆者等により作成されている．ISLISPはCommon Lisp系の言語でオブジェクト指向機能を備えている．しかし，パッケージ，モジュール，グラフィックス，他言語とのインタフェース機能が現在のISLISPには含まれていない．大規模なアプリケーションプログラムの開発を容易にするために，Common Lispを参考に新しいパッケージ機能を設計し，TISLシステムに実装した．本稿ではTISLのパッケージシステムの機能と実装について報告する．TISLのパッケージシステムは，パッケージの定義にはdefpackageとin-packageの2つの構文のみを用い，定義時に作成される名前解決優先度リストを用いて名前の衝突問題を解決する．また，全ての定義形式にアクセス修飾子を追加することにより，名前の隠蔽を行うことが可能になっている．したがって，TISLのパッケージシステムはCommon Lispのパッケージに比べ機能としても，実装上からも簡明なものになっている．

A Package System for an ISLISP Processor TISL

NOBUTO IZUMI† and TAKAYASU ITO†

ISLISP is the ISO standard Lisp language. We implemented its processor, called the TISL system. ISLISP is designed as a compact Lisp language with compact object-oriented facility. However, the current ISLISP does not support module/package, graphics, interfaces to other languages, etc. In paticular, the package system is important in developing large Lisp applications. We designed and implemented a package system for the TISL system. The TISL package system is simpler than the package system of Common Lisp. In the TISL package system a package is defined using two constructs "defpackage" and "in-package". Name conflicts are resolved by name precedence list to be created at defining packages, and name hidings from other packages are realized by access qualifiers added into all defining forms.

1. はじめに

プログラミング言語Lispは，人工知能や記号処理の研究のために1958年に誕生した言語で，FORTRANに劣らない長い歴史を持っている．最初の実用的な言語であるLISP1.5以来，諸種の言語の拡張が行われ，様々なLISP方言が生まれた．1980年代になって，産業用言語としてのLisp国際標準を制定する気運が高まった．国際標準化の活動[3]が1987年以来行われ，1997年5月にLisp言語のISO標準としてISLISP[1]が制定され，1998年7月にJIS標準[2]が制定された．また，いくつかのISLISP処理系が作成されている[4),7),8)]．

ISLISPは，処理系が開発され応用が進めば，改良や機能の追加が必要とされることが予想されている．モジュール／パッケージ，グラフィックス，他言語とのインタフェース機能が機能拡張の候補として標準化の段階で取り上げられた．特に，産業応用向き言語としてはパッケージやモジュールの機能および他言語インタフェース機能☆が望まれている．

筆者等はISLISPの処理系としてTISL（Tohoku university ISLisp）を設計，試作している[4)]．そこで，TISLによる大規模アプリケーションを可能とするためパッケージ機能を導入することを考えた．Common Lispのパッケージ[5)]を参考としながらも，パッケージの定義時に作成される名前解決優先度リストを用いて名前衝突を解決する新しいパッケージを設計し，TISLに実装した．このパッケージ機能を利用可能にすることにより，TISLを用いて大規模なアプリケーションの開発が行いやすくなる．

本論文の構成は次のとおりである．2章でISLISPの概要を述べる．3章ではTISL処理系に導入するパッ

† 東北大学大学院情報科学研究科
Department of Computer and Mathematical Sciences, Graduate School of Information Sciences, Tohoku University

☆ 他言語インタフェース機能については文献9）の研究がある．

ケージ機能について説明した後，Common Lisp パッケージとの比較について述べる．4 章では TISL 処理系について概説した後で，TISL でのパッケージ機能の実現について説明する．5 章では Gabriel Benchmark を用いてパッケージ機能の有無による効率の違いなどを実験結果を用いて説明する．6 章は結論である．

2. ISLISP の概要

Lisp 言語の ISO 標準として制定された ISLISP は日本が提案した核言語を基に設計されたものである[3]．ISLISP は Common Lisp 系の Lisp 言語でありながら，Scheme 並みにコンパクトにまとめられ，また，オブジェクト指向プログラミング機能を備えている．ISLISP は 6 つの名前空間 (変数，動的変数，関数，クラス，ブロック，タグ) を持つ多重名前空間方式を採用している．変数束縛は基本的に静的束縛であるが，(dynamic *var*) と明記することにより変数 *var* を動的変数として扱うことができる．

ISLISP の詳細については，その言語標準ドキュメント[1),2)] に譲り，ここでは，評価とオブジェクト指向について簡単に言及しておくに留める．

評　　価

ISLISP テキストは評価形式の列によって構成される．評価形式は，リテラル，識別子，複合形式からなり，複合形式は特殊形式，定義形式，関数適用形式，マクロ形式からなる．評価形式は評価されると，非局所脱出などの特別な場合を除いて，1 つのオブジェクトをその値として返す．ISLISP テキストの評価には，2 つの段階があり，ISLISP テキストは，まず実行準備され，次にその準備されたテキストが実行される．

オブジェクト指向とクラス

ISLISP のオブジェクト指向機能は CLOS を参考にしながらもコンパクトに設計されており，CLOS と同様にクラスと包括関数 (generic function) に基づくオブジェクト指向機能を備えている．

ISLISP のクラスシステムにおいては，全ての ISLISP オブジェクトはあるクラスのインスタンスである．クラスは <object> を根とする継承グラフを構成し，多重継承も可能となっている．クラスのクラスであるメタクラスとして <built-in-class> と <standard-class> が用意されている．

ユーザがクラスを定義するときには defclass 定義形式を次のように使用する．

(defclass <user-class> (<standard-class>)
((slot :reader get-slot)) (:abstractp nil))

クラスは，クラス名 (<user-class>)，上位クラスのリスト ((<standard-class>))，スロット指定 (((slot :reader get-slot)))，クラス任意機能 ((:abstractp nil)) からなる．クラスには複数の名前付きのスロットを定義することが可能である．スロットへのアクセス関数はクラスの定義時に指定する．また，クラスの定義時に上位クラスを指定しスロットの構造を継承することができる．この上位クラスからクラス優先度リストを作成する．

クラス優先度リストは自分自身を含めた上位クラスに関する全順序であり，上位クラスの局所優先順序と矛盾しないものとなっている．すなわち，あるクラスのクラス優先度リストは自分自身を先頭とし，上位クラスのクラス優先度リストを順に連結したものとなる．このとき，重複するものは左側のものを削除する．クラス優先度リストは包括関数の呼び出しの際に使用される．

3. ISLISP へのパッケージの導入

大規模なアプリケーションプログラムの作成においては，異なるプログラマによって作成されたプログラム間での名前の衝突が重大な問題となる．Common Lisp や他の言語ではこの問題に対してパッケージ機能を導入するすることによって対応している．

TISL へのパッケージの導入に当たっては，Common Lisp のパッケージを参考にしつつ，ISLISP のコンパクトな言語仕様を考慮し，コンパクトなパッケージシステムを設計する方針をとった．また，ISLISP ではクラス優先度リストがオブジェクト指向機能とその実現に利用されるが，TISL パッケージの設計と実現に名前解決優先度リストを導入し，これを用いて名前の衝突の解決を行う方法を取ることにした．

名前解決優先度リストとはパッケージの定義時に与えられる名前解決に使用するパッケージ名のリストである．複数のパッケージで同じ名前が公開されている場合，名前解決優先度リストにおいて優先度の高い方のパッケージで設定されている束縛が参照されることが保証される．ここで，束縛とは，1 つのパッケージに属しており，1 つの記号と 1 つの ISLISP オブジェクトの組であるとする．Common Lisp パッケージでは記号の印字名から記号への対応を行っているが，TISL パッケージでは，記号の印字名はコロンによって区切られた記号のリストである識別子に変換され，識別子は名前優先度解決リストを用いた名前衝突の解決法によって対応づけられる束縛の値を返す．

パッケージの名前が他の変数やクラスなどの名前と衝突することを避けるために，ISLISP にパッケージ

名前空間を追加し，パッケージ名はこの名前空間に定義される．パッケージの定義時には新しいパッケージの名前，アクセス修飾子，名前衝突の解決に使用するパッケージのリストを使用する．アクセス修飾子は新しいパッケージが他のパッケージから参照可能か否かを指定する．使用するパッケージのリストは他のパッケージにある束縛をパッケージ修飾子なしの識別子で参照するために使用される．

TISL パッケージでは Common Lisp パッケージと同様にコロンによって記号を区切り名前にパッケージ修飾子を付けている．また，Common Lisp では名前の衝突が発生した場合例外が発生するが，TISL パッケージシステムの場合には名前解決優先度リストによって優先度の高い方の名前が優先される．

3.1 定義済みパッケージ

定義済みのパッケージの階層構造を図 1 に示す．

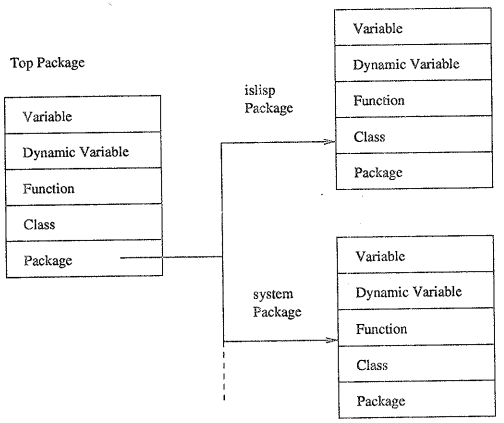

図 1 定義済みパッケージ
Fig. 1 Predefined Packages.

最上位にあるパッケージは名前のないパッケージであり，islisp パッケージ，system パッケージはこの最上位パッケージのパッケージ名前空間で定義されている．system パッケージは処理系定義または処理系依存の名前を定義する場所として使用され，islisp パッケージは islisp で定義されている名前を定義する場所として用いられる．これ以外に，ユーザは任意の名前を付けてパッケージを定義することが可能である．

3.2 識別子の拡張

3.2.1 パッケージ修飾子

他のパッケージにおいて定義されている値を参照するために識別子を拡張する必要がある．そこで Common Lisp を参考にして，コロンによるパッケージ修飾子付きの識別子を扱えるようにする．ISLISP では記号の名前がコロンを含むかどうかは処理系定義となっているので，TISL ではコロンはパッケージ修飾子を区切るための特別な文字として扱うことにする．

ISLISP では識別子が評価されると，静的環境での変数名前空間でその識別子が指定するオブジェクトを結果として返さなければならない．また，function, dynamic, class などの関数，動的変数，クラス名前空間で指定されるオブジェクトを返す特殊演算子は評価されない引数として識別子を取る．この様に識別子の意味はその識別子の使用される文脈によって変化する．

識別子は文脈によって指定される名前空間により，*VariableName*, *DynamicVariableName*, *FunctionName*, *ClassName*, *PackageName*, *BlockName*, *TagBodyTagName* に分類され，それらは次のように定義される．

VariableName ::= *symbol*
| *PackageName* : *symbol*
DynamicVariableName ::= *symbol*
| *PackageName* : *symbol*
FunctionName ::= *symbol*
| *PackageName* : *symbol*
ClassName ::= *symbol*
| *PackageName* : *symbol*
PackageName ::= *symbol*
| *PackageName* : *symbol*
BlockName ::= *symbol*
TagBodyTagName ::= *symbol*

各識別子は:を使用して記号を区切り，どのパッケージで定義されたか限定することが可能である．*BlockName* と *TagBodyTagName* には局所的な名前しか存在しないのでパッケージを指定することはできない．

以上により，識別子は 1 つの記号のみではなく，コロンで区切られた複数の記号からなるものとなる．

なお，ISLISP の各名前空間は独立しており，異なる名前空間に属する同名の束縛が，それぞれ別々のパッケージに存在したとしても，名前空間が異なる場合には名前の衝突は発生しない．名前の衝突は同じ名前空間の束縛に対してのみ発生する．

3.2.2 オブジェクトとしての記号

TISL パッケージは記号から束縛への対応を行うものとなっている．そのため，カレントパッケージの情報に依存せずに記号は管理される．次の例を考える．

```
(eq ':islisp:car 'car)
```

TISL パッケージではリーダによる記号の印字名から記号への対応を行っていないので，カレントパッケージの状態にかかわらず，2 つの記号，:islisp:car と car は別の記号と扱われ結果は nil を返す．

3.3 パッケージの定義

defpackage 定義形式はパッケージ名前空間に新し

い名前付きのパッケージを定義する．パッケージの定義は次のものを含む．

- 新しいパッケージの名前
- アクセス修飾子
- 名前解決に使用するパッケージのリスト

(**defpackage** *package-name* [*access-qualifier*] (*use-package**)) ⟶ <symbol> 定義演算子

ここで，

package-name ::= *symbol*
access-qualifier ::= :public | :private
use-package ::= *identifier*

defpackage 定義形式は，実行結果として *package-name* という名前の記号を返す．

引数 *package-name* は名前であり，新しいパッケージの名前となる．*package-name* の定義位置は **defpackage** の直後とする．

次のアクセス修飾子が使用できる．

- :public アクセス修飾子は新しい名前 *package-name* が他のパッケージに公開されていることを示す．
- :private アクセス修飾子は新しい名前 *package-name* が他のパッケージに非公開であることを示す．非公開とされた名前は他のパッケージからアクセス不能となる．

アクセス修飾子が省略された場合は非公開 (:private) であるとする．

引数 *use-package* のそれぞれは，識別子であり，新しいパッケージの使用するパッケージを指定する．名前を決定するときにこれらのパッケージが使用される．

名前解決優先度リストは使用するパッケージのリストから作成される．パッケージ $P_1, \cdots, P_n$ が，この順序でパッケージ P の defpackage 定義形式に指定されているとすると，名前解決優先度リストは $P, P_1, \cdots, P_n$ とする．$P_1, \cdots, P_n$ の中に同じパッケージが存在した場合には違反とする．

名前解決優先度リストはパッケージ中の記号に対応する束縛を決定するためにのみ使用されるものであり，このリストにより，パッケージの管理する大域束縛に束縛が追加されることはない．

3.4 カレントパッケージの変更

(**in-package** *package-name*) ⟶ <symbol> 特殊演算子

この特殊形式は，実行結果として *package-name* という記号を返し，カレントパッケージを *package-name* と名付けられたパッケージに設定する最上位形式であるとする．現在の静的環境のパッケージ名前空間において，識別子 *package-name* に対する束縛がない場合には，エラーが発生する．

3.5 他の定義演算子の拡張

他の定義演算子にもアクセス修飾子を付けることによって情報隠蔽を導入することが可能になる．

定義演算子は新しい名前と値を作成するための引数を取る．そこで名前の後にアクセス修飾子を記述できるようにする．アクセス修飾子が省略されている場合には非公開 (:private) であるとする．ただし，ISLISP の定義済みの名前は全て公開 (:public) として定義されているとする．また，ここで指定される新しい名前は一つの記号からなる名前でなければならない．定義形式を評価することにより，カレントパッケージに新しい大域束縛を追加することが出来る．各定義形式の構文は以下のように拡張される．

(**defconstant** *name* [*access-qualifier*] *form*) ⟶ <symbol> 定義演算子
(**defglobal** *name* [*access-qualifier*] *form*) ⟶ <symbol> 定義演算子
(**defdynamic** *name* [*access-qualifier*] *form*) ⟶ <symbol> 定義演算子
(**defun** *name* [*access-qualifier*] *lambda-list* *form**) ⟶ <symbol> 定義演算子
(**defclass** *name* [*access-qualifier*] (*sc-name**) (*slot-spec**) *class-opt**) ⟶ <symbol> 定義演算子
(**defgeneric** *name* [*access-qualifier*] *lambda-list* { *option* | *method-desc* }*) ⟶ <symbol> 定義演算子

但し，包括関数に対するメソッド定義を行う **defmethod** 定義形式は，同じパッケージ内の包括関数に対してのみメソッド定義を行えるものとする．

定義形式に対してこれらの拡張を行うことによって，自分以外のパッケージに対して非公開または公開となる変数を作成することが可能になる．

Common Lisp ではある記号を外部記号にするためには，export 関数を使用して別途明示する必要があるが，TISL パッケージでは定義時にその名前を公開するか否かを明示する．

3.6 名前優先度リストを用いた名前衝突の解決法

ISLISPでは識別子が評価されると，静的環境での変数名前空間でその識別子が指定するオブジェクトを結果として返す．また，function，classなどの特殊演算子は引数に識別子をとり，各名前空間でその識別子が指定するオブジェクトを結果として返す．また関数適用形式の演算子が識別子の場合，その識別子は静的環境の関数名前で評価され，呼び出すべき関数が作成される．このように識別子が評価に使用されるとき，静的環境での名前空間で対応するオブジェクトを取り出さなければならない．

名前の衝突は1つの名前に対して複数の候補となるオブジェクトが存在し，どれを選択するかが明らかでなくなるときに発生する．Common Lispではパッケージの構造の変更時に名前の衝突の検査が行われるので参照時に名前の衝突の検査をする必要はなく，パッケージ自身の外部記号，内部記号と使用しているパッケージの外部記号を検索する．TISLパッケージでは名前解決優先度リストを用いることにより名前の衝突を検査する必要はないが，参照時に次のようにして対応するオブジェクトを決定する．

最初に識別子が1つの記号からなるか否かで場合分けを行う．

(1) 1つの記号からなる場合
最初に対応する名前空間の局所的な名前を検索する．対応する束縛が存在しない場合には，カレントパッケージの名前解決リストの順番に従って，各パッケージが保持している大域束縛が検索される．

(2) コロンで区切られた複数の記号からなる場合
この場合は局所的な名前は検索しないことにする．ISLISPの仕様では，局所的な名前によって隠蔽された大域の名前を直接参照することはできなかったが，パッケージ修飾子を使用することにより，局所的な名前と大域の名前を使い分けることができるようにする．
複数の記号の中で最後の記号だけが構文的に指定された名前空間で解決され，その他の記号はパッケージ名前空間で決定される．さらに，最初の文字によって次のように場合分けされる．

- コロンで始まる場合
左側の記号から順に，最上位パッケージからパッケージ名前空間のパッケージを検索して行き，最後の記号を指定されている名前空間で検索し束縛を決定する．この場合の識別子はパッケージを最上位パッケージから完全に限定していることになる．
- コロン以外で始まる場合
左側の記号から順に，パッケージの名前解決優先度リストに従い，パッケージ名前空間の値が検索されていく．パッケージが存在しなければ次の優先度のパッケージで検索する．パッケージが存在した場合，そのパッケージに移動し，識別子の次の記号を移動したパッケージの名前解決優先度リストに従い，パッケージ名前空間の中から検索する．最後の記号を指定されている名前空間で検索し束縛を決定する．

3.7 パッケージ間における記号の扱い

大規模なプログラムをパッケージに分割して記述する場合，パッケージ間における記号の扱いが重要になる．この節では複数のパッケージにおいて使用される記号について説明を行う．

3.7.1 静的束縛

他のパッケージで定義した関数を呼び出す場合，呼び出す側のカレントパッケージに依存して名前の解決が行われ，参照される静的束縛が変化することは許されない．そのため，関数を呼び出し，その関数本体を実行している場合には，関数を定義したときのカレントパッケージの名前解決優先度リストを用いた名前解決が行えるようにしておく必要がある．次の例を考える．

```
(in-package :islisp)
(defglobal x 'islisp)
(defun foo () x)

(in-package user)
(defglobal x 'user)
(:islisp:foo)
```

この例では，islispパッケージにおいて大域変数xを定義しておき，同時に大域変数xの値を返す関数fooを定義する．ユーザ定義のパッケージuserに移動し，同じ名前の大域変数xを定義し，islispパッケージの関数fooを呼び出している．

islispパッケージの関数fooを呼び出した場合，その本体を実行する間の名前の解決は関数fooを定義したパッケージの名前解決優先度リストを用いて行われる．そのため，この評価形式の評価の結果は記号islispとなる．次の評価形式を考える．

```
(eq 'islisp (:islisp:foo))
```

この評価形式を評価した場合，呼び出された関数fooの結果の値はカレントパッケージに依存しない記号islispであり，結果はtとなる．

3.7.2 マクロ展開

次のマクロ展開について考える.

```
(in-package :islisp)
(defmacro caar (x)
  (list 'car (list 'car x)))

(in-package user)
(defglobal list '((foo)))
(:islisp:caar list)
```

islisp パッケージで定義されたマクロ caar をユーザ定義のパッケージ user で使用することを考える. マクロ形式はまず他の評価形式に展開される. この例において, (:islisp:caar list) が展開されるとき, 記号 list はマクロ caar が定義されたパッケージ islisp の名前解決優先度リストを使用して名前が解決され, 対応する関数 list が呼び出され展開が行われる. 展開された評価形式 (car (car list)) の記号 car, list は, カレントパッケージ user の名前解決優先度リストに従って名前の解決が行われる.

マクロ展開のために呼び出される関数 list はカレントパッケージ user に依存しないが, 展開後の評価形式中の記号 car, list に対応する束縛はカレントパッケージ user の名前解決優先度リストに従って決定されるために, その評価結果はカレントパッケージ user に依存したものとなり, 他のパッケージ islisp で定義されたマクロを使用する場合は注意が必要となる. カレントパッケージ user 中に islisp パッケージの関数 car を隠蔽する束縛が存在する場合には, カレントパッケージ user で定義されている関数 car が呼び出され, マクロ caar を定義したユーザの意図しない結果を引き起こす可能性がある.

確実に特定の束縛を参照したいときには, 上例のマクロ caar の場合, 次のようにパッケージを完全に限定した識別子:islisp:car を使用する必要がある.

```
(defmacro caar (x)
 (list ':islisp:car (list ':islisp:car x)))
```

3.8 ISLISP の仕様との関係

ISLISP の記号の名前が, コロン (:) やアンパサンド (&) を含むかどうかは処理系定義となっている. コロンで始まるキーワードが記号として表現されるのか別のものとして表現されるかも処理系定義となっている. よって TISL においてコロンは特別な区切り文字であるとし, コロンを含む識別子はパッケージを指定した特別な識別子であるとして評価される. また, コロンから始まるキーワードは最上位パッケージの変数名前空間において自分自身を返す定数と定義する.

TISL の処理系は, カレントパッケージを:islisp パッケージとし, カレントパッケージの変更を行わず, コロンを含む識別子を使用しなければ, ISLISP の仕様に準拠した処理系となる.

3.9 Common Lisp パッケージとの比較

TISL のパッケージシステムは Common Lisp のパッケージシステム [5] と ISLISP のクラス優先度リストによるオブジェクト指向機能の実現法を参考に設計されている. コロンを用いて記号を区切りパッケージ修飾子付きの名前を使用している.

Common Lisp では公開する名前を外部記号にするために export 関数などを用いる必要があるが, TISL パッケージシステムにおいては, 名前の定義時にその名前を公開するかどうかをアクセス修飾子を用いて明示している.

また, Common Lisp では名前の衝突が発生した場合, 例外が発生する. TISL パッケージの場合には名前解決優先度リストによって名前の衝突が解決されるために, 名前の衝突による例外は発生しない.

Common Lisp では defpackage や in-package 以外にもパッケージを操作する関数が多数定義されているが, TISL パッケージでは仕様を簡潔にするために, 2 つの構文 (defpackage, in-package) 以外は用いていない. そのため, TISL のパッケージシステムは Common Lisp のパッケージに比べ機能上から簡明なものになっている.

[例] TISL パッケージにおいて次の場合の名前の解決法について考える.

```
(in-package :islisp)
(defpackage test1 :public
            (:islisp :system))
(defpackage test2 :public
            (:islisp :system))
(defpackage test3 :public
            (test1 test2 :islisp :system))

(in-package :islisp:test1)
(defglobal test :public 'test1)
(in-package :islisp:test2)
(defglobal test :public 'test2)

(in-package :islisp:test3)
test
-> test1
```

上の例では, islisp パッケージにおいて test1, test2, test3 パッケージを公開定義し, test3 パッケージの名前

解決優先度リストを test3, test1, test2, :islisp, :system となるように定義している．そして，test1, test2 パッケージにおいて変数名前空間に test を公開定義し，test3 パッケージにおいて，test を参照している．test3 パッケージにおいて識別子 test は test3 パッケージの名前解決優先度リストにより，test1 パッケージで公開されている束縛 test を参照することになり，test1 を値として返す．

Common Lisp で同様のことを考えると，次のようになる．

```
(defpackage test1 (:use common-lisp)
                  (:export test))
(defpackage test2 (:use common-lisp)
                  (:export test))
(defpackage test3 (:use common-lisp
                        test1 test2))
```

上の例では，test が外部記号として宣言されているパッケージ test1, test2 を定義し，test1 と test2 を使用するパッケージ test3 を定義している．しかし，Common Lisp では test3 を定義した時点で，test1:test と test2:test の名前の衝突が起き例外が発生する．もし，例外を発生させずに，test1 パッケージの test を参照するようにパッケージを定義するためには次のようにする必要がある．

```
(defpackage test4 (:use common-lisp
                        test1 test2)
  (:shadowing-import-from test1 test))
```

shadowing-import-from を使用し，test1 パッケージの test を import し，他のパッケージの test を隠蔽している．このことを行うためには，使用するパッケージにおいて外部記号となっている記号がお互いに衝突するかどうかを熟知している必要がある．

4. TISL 処理系とパッケージの実現

ISLISP の処理系を試作し，TISL(Tohoku university ISLisp) と名付けた[4]．この章では，まず TISL の特徴について述べた後，パッケージ機能を備えた TISL の実現について説明する．

4.1 TISL 処理系の概要

TISL インタプリタは入力されたソースプログラムを直接解釈実行するのではなく，評価形式ごとに，一度中間コードに変換してから解釈実行する形式をとっている．図 2 に TISL 処理系の構成図を示す．

TISL インタプリタではユーザから入力されたテキストはリーダによって評価形式毎に内部表現に変換される．その内部表現は変換部によって中間コード形式

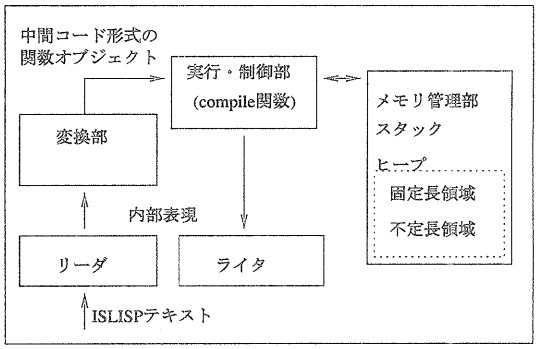

図 2 TISL 処理系の概要
Fig. 2 Outline of TISL System.

の関数オブジェクトに変換される．関数オブジェクトは実行・制御部によって実行され，実行結果がライタを通してユーザに表示される．この間に作成されるオブジェクトはメモリ管理部によって管理されるヒープ領域に作成される．また，パッケージ機能を追加するために，カレントパッケージを示す状態変数が追加されている．この変数の初期値は:islisp パッケージを示している．

TISL コンパイラは，TISL インタプリタの実行・制御部に用意されているコンパイル関数 compile によって実現されており，それを呼び出して中間コード形式の関数オブジェクトを C 言語プログラムにコンパイルする．得られた C 言語プログラムと既に作成されている共通プログラムを一緒に C コンパイラによってコンパイルし，アプリケーションとして利用可能な独立した実行ファイルを作成する．

4.2 パッケージ構造

パッケージはパッケージごとに束縛を管理し，それ自身階層構造をなす．また，TISL ではパッケージの再定義を許していないので名前解決優先度リストなどの構造が変更されることはない．よって，パッケージは専用の構造体を用意して表現している．

パッケージ構造体は次のものによって構成されている．

- 名前解決に使用するパッケージの数
- 名前解決に使用するパッケージ名配列へのポインタ
- 変数名前空間用束縛テーブル
- 関数名前空間用束縛テーブル
- 動的変数名前空間用束縛テーブル
- クラス名前空間用束縛テーブル
- パッケージ名前空間用束縛テーブル
- 属性リスト用のテーブル

各名前空間用の束縛テーブルで使用する束縛を表現するオブジェクトも専用の構造体を使用している．束縛構造体は次のものからなる．

- 名前 (記号へのポインタ)
- 値を保持する OBJECT 構造体
- この束縛が公開されているか否かなどを示すフラグ

定義形式が実行されると，カレントパッケージの定義形式の対応している名前空間に新しい束縛を追加する．定義演算子の第1引数である識別子から束縛の名前が決定され，アクセス修飾子によってこの束縛が公開されるかどうかが決定される．残りの引数によって束縛の値が作成され，束縛が設定される．

記号は，同じ印字用文字列を持つ複数の記号が作られないよう処理系で管理されている．そして，識別子は記号のリストとして作成され，名前の解決などで使用される．

4.3 中間コードにおけるパッケージの扱い

TISL 処理系で用いている中間コードは基本的に後置コードであり，スタックを用いて演算を行う．命令は，リテラルオブジェクトをスタックに積む命令，ISLISP の定義済みの関数に対応しスタック上の値を用いて演算を行う命令，ユーザ定義の関数を呼び出す命令，条件分岐などを行う制御用の命令に分類できる．

パッケージ機能を追加したことにより defpackage 定義演算子に対応する DEFPACKAGE 命令と in-package 特殊演算子に対応する IN-PACKAGE 命令が追加される．

DEFPACKAGE 命令は新しいパッケージの名前，名前解決優先度リスト，名前が公開されるか否かのフラグを中間コード中の引数にとり，これらの引数から新しいパッケージを作成し，束縛を設定する．

IN-PACKAGE 命令は識別子を1つ引数にとり，この識別子で指示されるパッケージにカレントパッケージを変更する．

その他の定義演算子に対応する各命令に対しても，中間コード中にとる引数として新しく設定される束縛の名前が公開されるか否かを表わすフラグが追加されている．

また，大域変数を参照する命令 PUSH-GLOBAL-VARIABLE は引数に1つの識別子をとり，実行時にカレントパッケージの名前解決優先度リストに従って，各パッケージの保持している束縛の中から対応する束縛を検索し，値をスタックにつむ．この命令は実行時に検索を行うため，オーバーヘッドが比較的大きなものとなってしまうことがある．

4.4 関数オブジェクトにおけるパッケージの扱い

関数オブジェクトは TISL 処理系において実行の単位となるオブジェクトである．この節では関数オブジェクトにおけるパッケージの扱いについて述べる．

4.4.1 関数オブジェクトの作成

TISL インタプリタは最上位の評価形式ごとに中間コードに変換し，解釈実行を行う．関数オブジェクトは引数の数や中間コード配列などからなる．最上位の評価形式は引数無しの関数オブジェクトとして作成される．

識別子が評価されると，静的環境での名前空間でその識別子が指定するオブジェクトを返す必要がある．関数オブジェクト中で PUSH-GLOBAL-VARIABLE などの静的環境での大域変数を参照するような命令の場合，静的なカレントパッケージの情報が必要になる．そのため，関数オブジェクトの作成時のカレントパッケージの情報を関数オブジェクトに保持させている．

中間コードへの変換の段階で演算子の識別子によって，一度対応する束縛が検索される．対応する束縛がカレントパッケージに存在する束縛である場合や，識別子がコロンから始まる完全に限定された識別子である場合には，束縛の値を調べ，組み込み演算子の場合には対応する専用の命令に変換し，効率の良い実行を行っている．処理系の初期状態では ISLISP の仕様に準拠させるために，カレントパッケージは組み込み演算子の定義されている islisp パッケージとなっている．このため，パッケージ関連の命令を使用しない限りこの専用の命令への変換は必ず行われる．

4.4.2 関数オブジェクトの実行

関数オブジェクトは，最上位の評価形式を変換して作成した場合を除いて，CALL などの命令から呼び出される．関数オブジェクトが実行される間，カレントパッケージを関数オブジェクトが作成されるときに指定されたパッケージに変更し，中間コードの中でカレントパッケージが必要な命令が正しくカレントパッケージを参照できるようにしておく．

関数オブジェクトの中間コード配列には，各命令に対応する C 言語の関数へのポインタが保持されており，この関数を順に呼び出すことにより，関数オブジェクトの実行を行う．

関数オブジェクトの実行が終了した後で，カレントパッケージを関数オブジェクトの実行前の状態に戻し，次の実行に移る．

4.5 TISL コンパイラへのパッケージの実装について

TISL パッケージについてのこれまでの説明は TISL

インタプリタを念頭に置いたものである．TISLコンパイラにおいても中間コードや関数オブジェクトでのパッケージの扱いは同じであるが，インタプリタ実行とコンパイラ実行での違いによる相違がある．よって，ここではその点について述べる．

TISLコンパイラのコンパイル単位は1つの評価形式を対象としている．コンパイラはその評価形式が使用する全てのオブジェクトを抜き出し，評価形式の実行時に使用できるよう環境を初期化するための関数を作成する．また，評価形式を変換して作成した関数オブジェクトの中間コードに対応して，C言語による関数を作成し，関数オブジェクトの実行をこの関数の呼び出しに対応させている．

コンパイル単位が1つの評価形式であることより，評価形式の実行中に大域の束縛が新たに追加されることはない．インタプリタの場合は，最上位形式の実行にともなって，新しい束縛が設定される可能性があるので，識別子の評価ごとに名前の衝突の解決を行う必要がある．一方，コンパイラの場合は，コンパイル時にパッケージに基づく名前の検索を行い，実行時に値を直接参照する命令に変換することによって，名前の解決にかかるコストを削除できる．識別子の表示などで必要になるのでパッケージによる束縛の管理は行っているが，実行時にはパッケージによる名前の解決は行っていない．

現在のTISLコンパイラのコンパイル単位は1つの評価形式となっているが，大規模なアプリケーションに対応するためにも，パッケージ単位でのコンパイル機能を持たせる必要がある．

ISLISPの標準化の段階で日本から提案されたモジュール機能[10]では，モジュールはインタフェース部と本体部に分かれ，インタフェース部によってimportするモジュールや，公開する各名前空間の名前などを宣言する必要があった．TISLパッケージでは定義時にこれらの情報を与えるので，インタフェースと本体部を分ける必要がない．

5. 評　価

TISLはWindowsおよびUNIX上で動作している．表1にGabriel BenchmarkプログラムをPC(CPU Intel-Celeron300A 300MHz，Memory 128MB，OS WindowsNT4.0)上のTISLインタプリタで実行したときの実行時間(単位は秒)を示す．

(1) 「パッケージなし」
パッケージ機能を導入する以前のTISL処理系[4]を用いた場合．

(2) 「islispパッケージ」
TISLにパッケージ機能を導入し，islispパッケージの中でGabriel Benchmarkで使用する関数を定義し実行した場合．

(3) 「userパッケージ」
Gabriel Benchmarkのプログラムをユーザ定義のパッケージ中で定義し，実行した場合．例えば，takは次のようなパッケージ中で定義され，実行される．

```
(in-package gabriel-benchmark)
(defpackage tak :public (:islisp))
(in-package tak)
(defun tak (x y z)
  (if (>= y x)
      z
      (tak (tak (- x 1) y z)
           (tak (- y 1) z x)
           (tak (- z 1) x y))))
```

「パッケージなし」と「islispパッケージ」とを比較することによって，TISLインタプリタにパッケージ機能を導入したことによるオーバヘッド，すなわち，名前解決優先度リストを用いた名前衝突の解決のためのオーバヘッドを見ることができる．

表1 Gabriel Benchmarkプログラムによる結果(TISLインタプリタ)
Table 1 Results of Gabriel Benchmarks (TISL interpreter).

(sec)

	パッケージなし	islispパッケージ	userパッケージ
tak	0.10	0.08	0.28
stak	0.15	0.31	0.50
ctak	0.13	0.15	0.34
takl	0.81	0.89	2.24
takr	0.10	0.14	0.38
boyer	1.49	2.78	6.60
browse	2.02	3.52	7.34
destr	0.25	0.28	0.92
trav-i	1.59	1.72	6.81
trav-r	10.69	13.17	35.31
deriv	0.30	0.35	0.77
dderiv	0.31	0.37	0.78
div2-i	0.22	0.26	0.75
div2-r	0.21	0.23	0.72
FFT	0.54	0.56	2.04
puzzle	1.71	2.07	6.55
triang	27.66	34.27	84.61

なお，表1において，「パッケージなし」の処理系での結果とパッケージ機能を導入した処理系における「islispパッケージ」での結果を比較すると，処理系が改善されておりtakでは若干実行時間が短いものと

なっている．

tak や div2 などのように，大域束縛を参照しないプログラムに関しては，実行時に名前の衝突の解決を行わないため「パッケージなし」の場合と同程度の実行時間となる．頻繁に名前衝突の解決を行うベンチマーク stak や takr などでは 1.5 倍 ~2.0 倍の速度低下が見られる．

カレントパッケージを islisp 以外にした「user パッケージ」の場合は，組み込みの関数呼び出しに対する最適化がまったく行われなくなる．このため，「islisp パッケージ」と「user パッケージ」との比較から知られるように，2~4 倍の速度低下が見られる．

TISL インタプリタにおいては，パッケージを使用したときのオーバヘッドが大きなものになってしまっている．インタプリタでは実行時に参照のため名前の検索を行う必要があるからであると考える．

TISL コンパイラの場合について，いくつかの Gabriel Benchmark による実行時間 (単位は秒) を表 2 に示す．

表 2 Gabriel Benchmark プログラムによる結果 (TISL コンパイラ)
Table 2 Results of Gabriel Benchmarks (TISL compiler).

(sec)

	パッケージなし	islisp パッケージ	user パッケージ
tak	0.06	0.06	0.06
ctak	0.08	0.10	0.11
takl	0.19	0.18	0.18
takr	0.08	0.08	0.11
FFT	0.20	0.20	0.21

コンパイラの場合には，4.5 に述べたようにコンパイル時にパッケージに基づく名前の検索を行い，実行時に値を直接参照する命令に変換できるので名前衝突の解決に要するコストを削除することができる．その結果，「user パッケージ」の場合でも「パッケージなし」の場合とほぼ同程度の効率が得られている．

6. まとめ

TISL のパッケージシステムの設計とその実装について報告した．TISL パッケージは Common Lisp のパッケージを参考にしつつ，ISLISP のオブジェクト指向機能の実現で利用されるクラス優先度リストにヒントを得て導入された名前解決優先度リストを用いて設計されている．パッケージのための構文としては defpackage と in-package のみであり，実装も Common Lisp パッケージに比べ簡単な方式となっている．

プログラマは，パッケージを用いることにより，大域への名前の集中を避けることが可能になる．また，各定義演算子を拡張することによって，定義時に名前を公開するか否かを宣言することが可能になる．また，defpackage 構文と defclass 構文の類似性から，<package> メタクラスなどを用意することによって，defpackage 構文の代わりに，defclass 構文を用い，変更をより少なくして ISLISP の言語仕様にパッケージ機能を追加することも可能である．

なお，パッケージ機能を備えた TISL コンパイラの実用化は今後の課題である．また，本格的な大規模アプリケーションプログラムの開発に当たっては，パッケージ単位でのコンパイルを行い，中間的なオブジェクトファイルを作成し，必要に応じてロードして使用できるようにすることが望まれるが，これは今後の課題である．

謝辞 本論文は 1999 年 6 月 17 日に開催された情報処理学会プログラミング研究会において配布の資料に，発表時における討論と研究会論文誌の査読報告に基づく改訂を加えたものである．研究会における討論に参加頂き有益な意見やコメントを頂いた方々に謝意を表します．また，研究会論文誌の査読者からは，パッケージにおける記号の扱いなど，懇切な御指摘を頂きました．査読者の方々に感謝の意を表します．

参考文献

1) ISO/IEC 13816 : 1997, Information technology – Programming languages, their environments and system software interfaces – Programming language ISLISP, p. 126, ISO/IEC (1997).
2) JIS X 3012 : 1998, プログラミング言語 ISLISP, p. 121, 日本規格協会 (1998).
3) 伊藤 貴康 : LISP 言語国際標準化と日本の貢献, 情報処理, Vol. 38, No. 10, pp. 932–937 (1997).
4) 泉 信人, 伊藤 貴康 : ISO 標準 Lisp 言語 ISLISP のインタプリタおよびコンパイラ, 情報処理学会論文誌, Vol. 40, No. 9, pp. 3510–3523 (1999).
5) Steele Jr., G. L.: *Common Lisp : The Language*, 2nd Edition, Digital Press (1990).
6) Gabriel, R.P.: *Performance and Evaluation of Lisp Systems*, MIT Press (1985).
7) Jullien, C.: OpenLisp, "http://www.ilog.fr:8001/Eligis/" (1998).
8) Lisp の ISO 標準規格 ISLISP の処理系の研究開発, "http://www.okilab.com/islisp/" (1999).
9) 高橋 順一他 : ISLisp 処理系とその複合システムインターフェイスの可搬性について, 情報処理学会プログラミング研究会資料 (1999 年 3 月).

10) Japanese SC22 LISP WG : Contribution on MODULES, ISO/IEC JTC 1/SC 22/WG 16 LISP N159 (1995).

(平成 11 年 5 月 28 日受付)
(平成 11 年 10 月 13 日採録)

泉 信人 (学生会員)

1974 年生. 1998 年東北大学工学部情報工学科卒業. 同年東北大学大学院情報科学研究科情報基礎科学専攻博士前期課程進学.

伊藤 貴康 (正会員)

1940 年生. 1962 年京都大学工学部電気工学科卒業. スタンフォード大学コンピュータサイエンス学科および人工知能プロジェクト研究助手, 三菱電機中研を経て 1978 年から東北大学. 現職, 東北大学情報科学研究科教授. 工学博士. 本会理事, 東北支部長等を歴任. 現在, Information and Computaion の Editor, Higher-Order and Symbolic Computation の Associate Editor, IFIP TC1 member 等. 専攻分野, ソフトウェア基礎科学. 日本ソフトウェア科学会, 電子情報通信学会, 人工知能学会, ACM 各会員.