ISLISP の処理系 TISL とそのパッケージシステム

泉　信人[†]　伊藤　貴康[†]

TISL は Lisp 言語の ISO 標準 ISLISP のインタプリタおよびコンパイラからなる処理系である．また，TISL には大規模アプリケーションの開発を容易にするため，コンパクトに設計されたパッケージ機能が備えられている．TISL パッケージの名前衝突の解決は名前優先度リストを用いることによって実現されている．本稿では，例を示しながらアプリケーション開発時の TISL パッケージ機能と使用法の概要を述べる．また，パッケージ機能を使用したプログラムの実行結果についても紹介する．

An ISLISP Processor TISL and Its Package System

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 particular, the package system is important in developing large Lisp applications. We designed and implemented a package system for the TISL system. 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 hiding from other packages are realized by access qualifiers added into all defining forms. In this paper we explain how to use the TISL package system, and also we report some experimental results on the TISL package, using the Gabriel benchmarks.

1. はじめに

ISLISP[1,2]は 1997 年に制定された ISO 標準 Lisp であり，多重名前空間をサポートし，オブジェクト指向機能を有するコンパクトな Lisp 言語である．

TISL(Tohoku univ. ISLisp)[3]はインタプリタとコンパイラを備えた ISLISP の処理系である．また，大規模なアプリケーションの開発を容易にするためのパッケージ機能が設計され，TISL に実装されている [4]．

TISL のパッケージ機能は ISLISP のコンパクトな言語仕様を考慮し，コンパクトな設計となっている．パッケージ機能を支援する新規の関数として defpackage と in-package の二つの関数が導入され，名前の衝突はパッケージの定義時に作成される名前優先度リストによって解決する．

本稿では TISL の概要について説明したのち，TISL のパッケージ機能の概要を説明し，例を示しながら機能の説明をする．最後に Gabriel benchmark[5]を用いた実験評価結果について報告する．

2. TISL システムの概要

インタプリタとコンパイラを備えた処理系である TISL は，C 言語によって記述されており，Windows や UNIX で動作する移植性の高い処

† 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, Sendai, Japan

理系となっている．TISL インタプリタは ISLISP の最初のインタプリタである OpenLisp[6]並みの実行効率をもち，TISL コンパイラは TISL インタプリタよりも数倍高速に動作する．

TISLインタプリタは入力された評価形式を直接解釈実行するのではなく，仮想マシンの命令に変換し，効率的に実行している．また，TISLコンパイラはISLISP評価形式を仮想マシン命令に対応するC言語プログラムにコンパイルし，C言語コンパイラによって実行ファイルが作成される．

TISL には処理系依存の機能としてパッケージ機能が実装され,処理系については 3),4)において発表したが，本稿の TISL ではパッケージ機能下でのインタプリタとコンパイラの実装改善が行われている．

3. TISL のパッケージ機能

3.1 パッケージ機能の概要

大規模なアプリケーションの作成では，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大きな問題となる．Common Lisp[7]や他の言語ではこの問題に対してパッケージを導入することによって対応している．

TISL のパッケージ機能は，識別子と束縛する ISLISP 評価形式との対応関係を扱うシステムとなっており，ISLISP のコンパクトな言語仕様を考慮し，コンパクトな設計となっている．

定義形式で定義される最上位の有効範囲を持つ識別子は，定義時に有効になっているカレントパッケージの最上位の有効範囲を持つ．他のパッケージから識別子を参照するためにはパッケージ修飾子などの特別な方法をとるか，パッケージの定義時に，他のパッケージの有効範囲を参照することを明示する．

ISLISP ではクラス優先度リストにより，継承するクラスの特徴や包括関数の呼び出し時のメソッドの構成を決定しているが，TISL パッケージ機能では名前の衝突を解決する方法として，名前優先度リストを導入している．名前優先度リストとは，有効範囲を使用するために，パッケージの定義時に与えられるパッケージ名の優先度付きのリストである．複数のパッケージの有効範囲において同じ名前で定義されている変数が存在した場合，名前優先度リストにおいて優先度の高いほうのパッケージで定義されている変数が参照されることを保証する方式を用いる．

定義形式の引数にアクセス修飾子を追加し，名前を他のパッケージに対して公開するかどうかを定義する．

また，他のパッケージで定義されている名前を参照するためにコロンによるパッケージ修飾子付きの識別子を扱い，パッケージ修飾子付きの識別子は各パッケージで最上位の有効範囲で定義された識別子を参照することにする．

以下の節では，クラス優先度リストに基づき設計された TISL パッケージシステムについて TISL での動作を示しながら説明する．

3.2 パッケージの構成

ISLISP は多重名前空間を採用している．そこで，パッケージ名を変数や関数などの名前と衝突することを避けるために，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．定義形式で設定される束縛は，カレントパッケージの対応する名前空間に保持される．**図 1** に定義済みのパッケージの階層構造を示す．

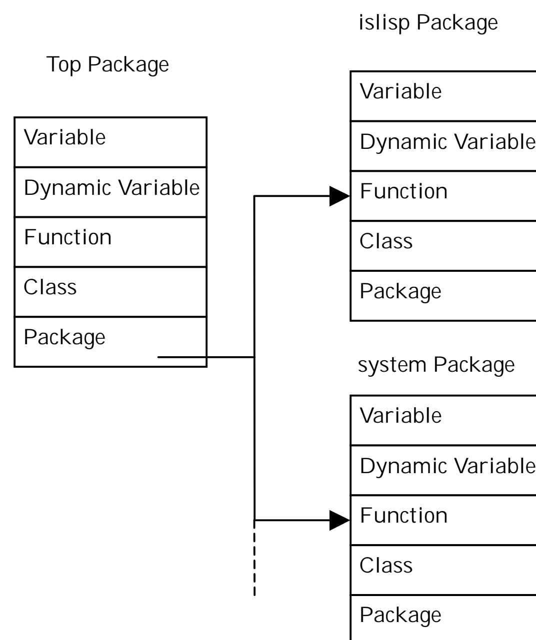

図 1　定義済みパッケージ
Fig.1 Predefined Packages

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依存の名前が定義され，islisp パッケージは ISLISP で定義済みのオブジェクトが定義されている．ユーザは任意の名前をつけてパッケージを定義することができる．

他のパッケージの最上位の有効範囲で意味が定義されている識別子を参照するために，パッケージ修飾子を使用する．このときコロン(:)から始まる識別子は最上位パッケージで定義されている識別子から決定していく．このコロンから始まる識別子を完全限定名と呼ぶ．他の場合には，カレントパッケージの名前優先度リストに従いパッケージ修飾子から検索していく．

3.3 パッケージの定義

ユーザがパッケージを定義するときには **defpackage** 定義形式を使用する.定義時には新しいパッケージの名前，アクセス修飾子，名前解決に使用するパッケージのリストを与える．次に例を示す.

```
:islisp > (defpackage user :public   (:islisp))
```

:islisp >はシステムで有効になっているカレントパッケージが islisp パッケージであることを示している．上の例では，islisp パッケージのパッケージ名前空間に他のパッケージに公開される名前 user でパッケージを定義している．このパッケージは islisp パッケージの有効範囲を使用し，user パッケージがカレントパッケージの時には，カレントパッケージで定義された名前だけでなく islisp パッケージで公開定義された名前もパッケージ修飾子のない名前によって参照することができる．このときの user パッケージの名前優先度リストは(user :islisp)となる．

in-package 特殊形式はカレントパッケージを変更する他の評価形式の中に入れ子となっていない最上位形式であり，次のように使用する.

```
:islisp > (in-package user)
user
:islisp:user >
```

この特殊形式を評価すると，islisp パッケージであったカレントパッケージが識別子:islisp:user に対応するパッケージに変更される.

3.4 パッケージ間の記号の扱い

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，パッケージ間の記号の扱いが重要になる.

3.4.1 静的原理による変数参照

ISLISP は静的可視性の原理に従うように設計されている．TISL パッケージシステムにおいてもこの原理に従って，変数の意味を決定している．次の例を考える.

```
:islisp > (defglobal x 'islisp)
:islsip > (defun x :public () x)
…
:islisp:user > (defglobal x 'user)
:islisp:user > x
user
:islisp:user > (x)
islisp
:islisp:user > (eq 'islisp (x))
t
```

islisp パッケージの最上位の有効範囲の変数名前空間と関数名前空間に x との束縛を設定しておく．また，カレントパッケージを 3.3 節の user パッケージに移動し，変数名前空間の最上位の有効範囲に x との束縛を設定しておく.

user パッケージで x を評価すると，変数名前空間で識別子 x の意味が決定され，最も優先度の高い user パッケージで定義された変数が参照され，記号 user を結果として返す.

user パッケージで(x)を評価すると，関数名前空間で識別子 x の意味が決定され，islisp パッケージで定義された関数が呼び出される．この関数の中で，変数 x が参照されるが，静的可視性の原理に従い，islisp パッケージの変数名前空間で定義された変数 x が参照される．結果として，記号 islisp を返す.

また，印字名と記号との対応を管理する Common Lisp のパッケージシステムと異なり，TISL パッケージシステムは識別子と，束縛する ISLISP 評価形式との対応関係を管理するシステムとなっているため，(x)の評価値，記号 islisp と user パッケージでの'islisp の評価値，記号 islisp は同じものであり，eq において t を返す.

3.4.2 マクロ使用時の注意

TISL パッケージシステムでは, Common Lisp パッケージのような印字名から記号への対応関係を扱っておらず，また，記号はパッケージと

独立して存在しているために, 次の例のように, 局所的に定義される変数によってプログラマが意図したものと異なる動作をする可能性がある.

```
:islisp > (defmacro caar :public (x)
            `(car (car ,x)))
:islisp:user > (defun car (x) (cdr x))
:islisp:user > (caar '((a) b . c))
c
:islisp > (flet ((car (x) (cdr x)))
            (caar '((a) b . c)))
c
```

islisp パッケージにおいてマクロ caar を上のように定義しておく．マクロは展開され，展開された評価形式が評価される．このとき，TISL パッケージシステムでは記号にパッケージ情報を持たせておらず，展開された先のカレントパッケージによって，識別子に対応する値が決定される．そのため，解決された名前がユーザの意図したものと異なる可能性がある. 同様に, flet などによって局所的に定義される識別子によって，ユーザの意図した識別子が隠蔽され，動作が変更される.

　このような問題の場合，マクロで使用する識別子には次のように完全限定名を使用することによって解決できる.

```
:islisp > (defmacro caar :public (x)
                 `(:islisp:car (:isllisp:car ,x)))
:islisp:user > (caar '((a) b . c)
a
:islisp > (flet ((car (x) (cdr x))) (carr '((a) b . c)))
a
```

このマクロ定義では，関数 car を参照するために:islisp:car と最上位パッケージからのパッケージ修飾子を付け，他のパッケージで設定されている束縛を参照しないようにすることができる．また，パッケージ修飾子を使用することにより，局所的な有効範囲を持つ変数を参照することも避けられる.

3.5 パッケージの使用例

　この節では TISL のパッケージシステムの使用例として算術パッケージを挙げ，その機能の説明を行う.

```
:islisp >
  (defpackage arithmetic :public (:islisp))
  (in-package arithmetic)
:islisp:arithmetic >
  (defun + :public (:rest arglist)
    (cond   ((null arglist) 0)
            ((consp arglist)
             (plus (car arglist)
                   (cdr arglist)))))
(defun plus (x rest)
    (cond   ((null rest) x)
            ((consp rest)
             (plus (plus2 x (car rest))
                   (cdr rest)))))
(defgeneric plus2 (x1 x2)
  (:method ((x1 <number>)
            (x2 <number>))
     (:islisp:+ x1 x2))
  (:method ((x1 <general-vector>)
            (x2 <general-vector>)
        ....))

:islisp:arithmetic >
    (defclass <complex> :public ()
      ((real ...) (imag ..)))
    (defmethod plus2
      ((x1 <complex>) (x2 <complex>)) ...)
```

上の例では，arithmetic パッケージを定義し，加算を行う公開関数として＋を定義し，非公開の関数として plus，plus2 を定義している．arithmetic パッケージは:islisp パッケージを使用することが明示されており，名前優先度リストは(arithmetic :islisp)となる．ここで 2 項加算関数 plus2 は包括関数として定義されており，整数のみでなくベクトルや複素数などさまざまなオブジェクト同士の加算を定義することが可能となっている．パッケージを使用することにより，+， plus， plus2 と名付けられた関数が他のプログラマによって既に定義されているかどうかを気にする必要がなくなる. また，識別子 + のみを公開関数とすることによって，arithmetic パッケージを使用する他のプログラマに対して必要のない関数 plus，plus2 を隠蔽することができる.

```
:islisp > (defpackage user2 (arithmetic :islisp))
:islisp:user2 > (+ #(1 2 3) #(4 5 6))
#(5 7 9)
```

user2 パッケージを定義し，そこで arithmetic パッケージを使用する場合，arithmetic パッケージを islisp パッケージよりも優先度を高く設定することにより，このパッケージの名前優先度リストは(user arithmetic :islisp)となる． よって関数＋は islisp パッケージで定義されている組込み関数＋ではなく，arithmetic パッケージで定義された拡張加算関数+を参照することになる．また，user2 パッケージからは直接 plus, plus2 といった公開されていない関数を呼び出すことはできない．

4. 実験評価

表 1 に Gabriel Benchmark プログラムと包括関数を利用して作成したプログラム(gtak, gderiv, gqsort)を PC (CPU Intel Celeron-300A 300MHz, Memory 128MB, OS Windows2000)上で実行したときの実行時間(単位は秒)を示す．TISL インタプリタ，コンパイラのそれぞれについて次の条件で実行を行っている．

- **no package** パッケージ機能の実装前の TISL 処理系において実行．
- **islisp** ISLISP の構文のみによるベンチーマークプログラムをパッケージ機能を実装した TISL 処理系において実行．
- **use package** パッケージ機能を利用したベンチマークプログラムをパッケージ機能付きの TISL 処理系において実行．

TISL コンパイラの islisp の場合については use package の場合とほぼ同程度の実行時間となるので省略する．

use package の場合には，次のようにパッケージ機能を利用して Gabriel Benchmark プログラムを実行している．

```
:islisp:gabriel >
 (defpackage tak
            (:islisp:gabriel :islisp :system))
 (in-package tak)

:islisp:gabriel:tak >
 (defmacro 1- (x) `(- ,x 1))
 (defun tak :public (x y z)
  (if (not (< y x))
      z
      (tak (tak (1- x) y z)
           (tak (1- y) z x)
           (tak (1- z) x y))))
```

```
:islisp:gabriel > (tak:tak 18 12 6)
7
```

gabriel パッケージを用意し，各プログラム用のパッケージ，tak の場合は:islisp:gabriel:tak パッケージを定義し，その中で tak 関数を公開定義している．また，マクロ 1-が定義されているが，1-識別子は公開されておらず，他のパッケージから直接参照されることはない．各パッケージにベンチマークプログラムを定義することによって，各ベンチマークプログラムで定義されている名前の衝突を気にする必要もなく多くのベンチマークプログラムを実行することができる．

表１ Gabriel Benchmark プログラムによる結果
Table 1 Results of Gabriel Benchmarks.

(sec)

	TISL interpreter			TISL compiler	
	no package	islisp	use package	no package	use package
tak	0.08	0.14	0.14	0.03	0.04
stak	0.16	0.62	0.73	0.06	0.23
ctak	0.13	0.23	0.26	0.06	0.11
takl	0.81	1.37	1.36	0.16	0.33
takr	0.10	0.17	0.16	0.06	0.06
boyer	1.49	4.14	4.27	0.43	2.46
browse	2.41	5.80	7.04	0.92	3.06
destruct	0.26	0.43	0.43	0.10	0.16
tra-init	1.65	2.69	2.65	0.43	0.93
tra-run	10.75	18.12	18.07	3.56	6.21
deriv	0.30	0.50	0.69	0.16	0.31
dderiv	0.31	0.54	0.74	0.17	0.37
div2-iter	0.23	0.38	0.44	0.09	0.18
dvi2-rec	0.22	0.36	0.45	0.09	0.18
FFT	0.54	0.88	0.88	0.17	0.32
puzzle	1.71	2.96	2.96	0.39	0.89
triangle	27.72	46.11	46.05	7.15	13.17
gtak	0.14	0.25	0.25	0.04	0.14
gderiv	0.70	1.36	1.84	0.52	1.28
gqsort	0.22	0.60	0.84	0.14	0.36

表 1 より，パッケージ機能の実装前と実装後のインタプリタの実行結果を比較すると，動的変数を扱う stak で 3.9 倍,属性リストを扱う boyer，browse で 2.4~2.8 倍,多重継承したオブジェクトの比較を行う gqsort で 2.7 倍,その他のプログラムで 2 倍弱の実行速度の低下が見られた. 動的変数や属性リストは実行時に識別子の意味を決定する必要があるため，有効範囲の複雑なパッケージ機能下において，大きなオーバヘッドとなっている．今後，これらの処理を中心に処理系の改善を行っていく必要がある．

パッケージ機能実装後の処理系において，ISLISP の機能のみを使用した場合と，パッケージ機能を使用した場合を比較すると動的変数や属性リストを使用した stak, boyer, browse で 1.2 倍程度,包括関数を使用する gderiv, gqsort で 1.4 倍程度の実行時間となっているが，それ以外のものはほぼ同程度の実行時間となっており,パッケージ機能の使用によるオーバヘッドはほとんど見られない.

インタプリタとコンパイラを同条件のもので比較すると，パッケージ機能実装前の処理系において 1.3〜5.1 倍，パッケージ機能実装後の処理系において 1.5 倍〜4.1 倍の高速化が得られている．TISL コンパイラは包括関数の呼び出しなどで TISL インタプリタと共通のものを使用したインタプリタベースの処理系となっているため,これらの処理が実行時間の多くを占めるものについてはあまり効率化が得られていない．また，パッケージ機能実装後のコンパイラがパッケージ機能実装前のインタプリタと比較して遅くなっているプログラムもあるので，今後，実行効率の改善を行っていく必要がある.

5. まとめ

TISL のパッケージ機能について報告した．TISL のパッケージ機能は 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をもとに導入された名前優先度リストによる識別子の参照と defpackage と in-package の 2 つの構文の導入によって実現されておりコンパクトな仕様となっている．パッケージ機能を用いることにより，識別子の有効範囲をパッケージごとに区切り，名前の衝突を避けたり，名前の公開非公開による情報隠蔽を行えるなど，大規模なアプリケーションを複数のプログラマによって作成するために便利な機能が提供されている．

[今後の課題]

アプリケーションの開発を行っていく上で，オーバヘッドの少ない他言語インタフェース，特に C 言語とのインタフェースを用意しておくことは，アプリケーションの相互利用ができ有用である．現在，C 言語および Java と ISLISP とのインタフェースを TISL において実装する試みを行いつつある．

大規模なアプリケーションの開発を行うための分割コンパイル機能の実現，また，型推論機能の充実などによる処理系の実行効率の改善，TISL に基づくエキスパートシステムや記号処理システムの実現なども今後の課題である．

[TISL システムの利用可能性]

TISL システムのうち，インタプリタについては，研究室での経験から実用的なレベルにあると考えている．コンパイラについては，他言語インタフェースとの関連も含め整備中である[†].

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) 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. 10， pp.932-937 (1999).
4) 泉 信人，伊藤 貴康 : ISLISP のためのパッケージシステム，情報処理学会論文誌 : プログラミング, Vol.40, No. SIG 10 (PRO 5), pp17-27 (1999).
5) Gabriel, R. P. : *Performance and Evaluation of Lisp Systems*, MIT Press (1985).
6) Jullien, C. : OpenLisp, http://www.ilog.fr:8001/Eligis/
7) Steele Jr., G. L. : *Common Lisp : The Language*, 2nd Edition, Digital Press (1990).

---

† コンパイラや他言語(特に，C 言語)インタフェースを完成の後，公開予定であるが，TISL システムの試用に関心のある方は izumi@ito.ecei.tohoku.ac.jp に，連絡されたい.