A RAND NOTE

THE PORTARE LISP TRANSLATION TOOL

Jed Marti

September 1984

N-2152-ARPA

Prepared for

The Defense Advanced Research Projects Agency

Rand
1700 MAIN STREET
P.O. BOX 2138
SANTA MONICA, CA 90406-2138

The research described in this report was sponsored by the Defense Advanced Research Projects Agency under ARPA Order No. 3460, Contract No. MDA903-82-C-0061, Information Processing Techniques Office.

Published by The Rand Corporation

A RAND NOTE

THE PORTARE LISP TRANSLATION TOOL

Jed Marti

September 1984

N-2152-ARPA

Prepared for

The Defense Advanced Research Projects Agency

Rand
1700 MAIN STREET
P.O. BOX 2138
SANTA MONICA, CA 90406-2138

PREFACE

Compared with contemporary translation tools, the PORTARE interdialect LISP translation program developed at The Rand Corporation offers enhanced functionality and improved interface. This manual describes the construction, syntax, operation, and deficiencies of the system in the Portable Standard LISP (PSL) environment.

PORTARE was developed to satisfy a need to translate the ROSIE™[1] computer programming environment from Interlisp into PSL to increase both its execution speed and its availability. This initial goal has colored much of the construction of PORTARE, and many of its most significant features were designed with this translation in mind. Although the goal of complete mechanical translation has not been met in the case of ROSIE, such a significant subset of Interlisp can be translated as to make the system useful in a much wider context. Programs can now be translated from Interlisp to PSL and back and from Franz Lisp into PSL. Other translations are currently being implemented for Zeta Lisp.

This work was supported by the Information Processing Techniques Office of the Defense Advanced Research Projects Agency under Contract MDA903-82-C-0061.

---

[1]ROSIE is a trademark and service mark of The Rand Corporation.

SUMMARY

There are several important dialects of the LISP programming language, namely, Interlisp, Franz Lisp, Zeta Lisp, Common Lisp, Portable Standard LISP (PSL), Cambridge Lisp, and Lisp 370. Programs implemented in one dialect are usually not directly transportable to another. The PORTARE program was developed to automatically translate programs from one LISP dialect to another and to support their execution in the target system. It features a pattern-directed translator coupled with a scheme for incremental translation of the individual modules of large programs. The run time environment supports deficiencies of the target LISP system. PORTARE is currently capable of translating between Interlisp and PSL (in both directions), and from Franz Lisp into PSL.

CONTENTS

PREFACE iii

SUMMARY v

Section
1. INTRODUCTION 1

2. ".PORTO" TRANSLATION FILE SYNTAX 4
   2.1. Annotation 5
   2.2. Number of Arguments 5
   2.3. Functions That Are the Same: ASIS and WARNING .. 6
   2.4. Functions for Which No Equivalent Exists: NOEQUIV 8
   2.5. Link to the System Editor: EDIT 9
   2.6. Stopping the Scan: STOPSCAN 9
   2.7. Ignoring Functions: IGNORE 10
   2.8. Renaming Functions: => 10
   2.9. Pattern-Directed Translation 11
   2.10. Special Foms: SPECIAL 14
   2.11. Conditional Translation: IF 15
   2.12. Translation Time Expressions 16

3. ".PREPAS" SYNTAX CONVERSION AND SPECIAL TRANSLATIONS ... 17
   3.1. Prepass Functions 17
   3.2. Extended Translation Functions 18
   3.3. Target LISP Interface Functions 18

4. ".EXTEND" EXTENSIONS TO THE TARGET LISP 19

5. ".TRANS" FOR SEGMENTED PACKAGE TRANSLATION 20

6. SETTING UP THE PORTARE ENVIRONMENT 22

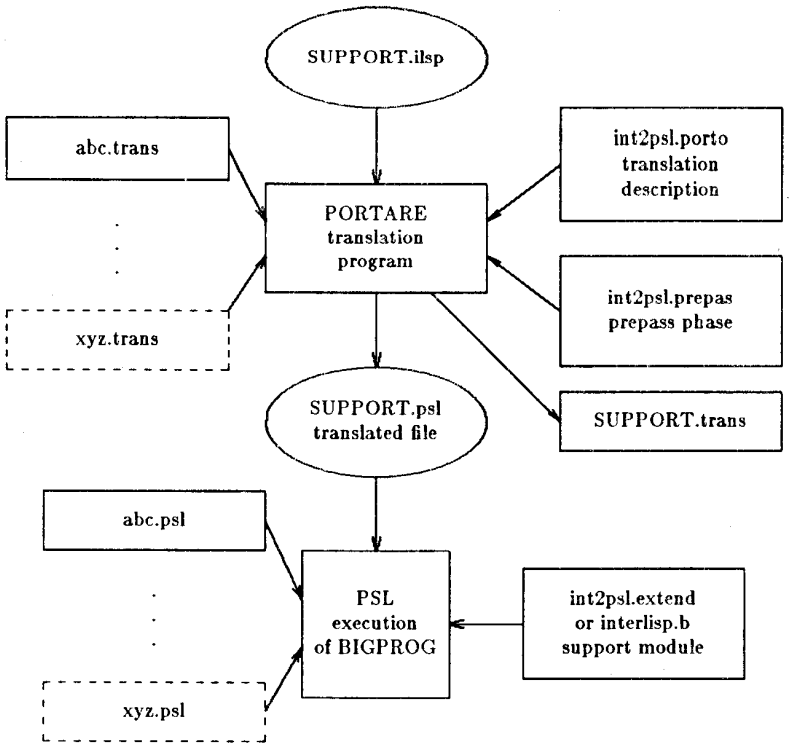
7. SPECIFIC TRANSLATION NOTES 24
   7.1. Interlisp -> PSL 24
   7.2. PSL -> Interlisp 29
   7.3. Franz Lisp -> PSL 30

1. INTRODUCTION

The PORTARE translation tool provides an environment for translation of programs between LISP dialects, with an emphasis on translation rather than emulation. This manual describes its use and the implementation and maintenance of PORTARE translations, which consist of collections of artifacts for translating from a "source" to a "target" dialect and a run time environment supporting target dialect deficiencies.

The translation name is formed from the first three letters of the names of the source and target LISP (all lower case), separated by 2. For example, the Interlisp-to-PSL translation named *int2psl* consists of a number of files named *int2psl.porto*, *int2psl.prepas*, etc.

The following diagram briefly outlines the translation of the Interlisp module *SUPPORT.ilsp* into the PSL file *SUPPORT.psl* and then its execution. *SUPPORT.ilsp* is part of a larger Interlisp program, BIGPROG,



Overall Flow and Organization of PORTARE

which also contains *abc.ilsp* to *xyz.ilsp*. These files have been previously translated into *abc.psl* to *xyz.psl*. Since *SUPPORT.ilsp* may reference functions or data structures in these modules, special files *abc.trans* to *xyz.trans* were created which contain interface information to aid the translation.

The file *int2psl.prepas* contains special lexical translations from Interlisp into PSL. PORTARE generates two files: *SUPPORT.psl* is the PSL version of *SUPPORT.ilsp* and *SUPPORT.trans* is translation information which may be of use in translating other modules. To execute the program, all translated files are loaded together with *int2psl.extend*, a collection of functions to support Interlisp functions not present in PSL.

As an example of the translation process, consider the following Interlisp, *FACT* file for the factorial function:

```
(FILECREATED "24-MAY-84 15:39:54" {DSK}FACT.;1 387
      changes to:  (VARS FACTCOMS))

(PRETTYCOMPRINT FACTCOMS)

(RPAQQ FACTCOMS ((fns fact)))
(DEFINEQ

(fact
  [LAMBDA (n)          (* edited: "24-MAY-84 15:30")
    (COND
      ((LESSP n 2)
        1)
      (T (TIMES n (fact (SUB1 n])
)
(PRETTYCOMPRINT FACTCOMS)

(RPAQQ FACTCOMS ((FNS fact)))
(DECLARE: DONTCOPY
  (FILEMAP (NIL (151 308 (fact 161 . 306)))))
STOP
```

To convert this file into PSL, the following sequence of commands is entered:

```
(OFF RAISE)
(LOAD lmeta portare)
(PORTARE "fact" 'interlisp 'psl NIL NIL)
```

PORTARE converts this file into the following PSL code (some cosmetic editor changes have been made):

```
(DE FACT (N)
  (!* edited !: "24-MAY-84 15:30")
  (COND ((LESSP N 2) 1)
        (T (TIMES N (FACT (SUB1 N))))))

STOP
```

Sections 2 and 3 of this manual present the conventions and syntax for the files describing a translation. Section 4 describes the implementation of the support environment for the target LISP. Section 5 details the operating procedures, and Section 6 covers the features and deficiencies of individual translations. Persons wishing to use the translator may ignore Sections 2 through 4 and skip directly to Section 5.

2. ".PORTO" TRANSLATION FILE SYNTAX

The *.porto* file specifies the translation between source and target LISP. The file contains the name of every function defined in the source LISP, together with actions to translate a function call to the target LISP. The functions are usually presented in the order in which they occur in the source LISP's User's Manual. The names of the manuals used in constructing a translation appear in the annotation in the prologue of the *.porto* file.

Every function in the source dialect has a translation process and annotation connected with it. These translations (which we call "statements" for convenience) are terminated with a semicolon. The following segment implements three such translations:

```
ASIS "Chapter 5" CAR(1), CDR(1);
DSUBST => SUBSTIP(3);
```

The first line declares that CAR and CDR remain "as is"; the target dialect CAR and CDR closely resemble those of the source dialect, only their arguments being translated. The digits following the function names indicate the number of arguments the target function requires, a useful piece of information for translating from LISPs that do not require all arguments to ones that do (Interlisp to PSL, for example). The second line dictates that the DSUBST function name is to be replaced in the target LISP by the SUBSTIP function name, and its arguments are to be translated. The target SUBSTIP must have 3 arguments; if it does not, PORTARE generates sufficient NILs to fill out the actual parameter slots. The annotation connected with CAR and CDR is the string "Chapter 5"; the annotation connected with DSUBST is a default annotation set at some other time.

2.1 ANNOTATION

A percent sign (%) prefixes comments, terminated by the end-of-line, within the *.porto* file. This annotation is discarded during parsing. Each source LISP function has an associated PSL string: a set of characters enclosed in double quotation marks ("). Two double quotes next to each other are used if the quotation marks are to appear in the output. The translator displays this "remark" as a target LISP comment when it has trouble translating the associated form, both before the function in which it appears and in a summary at the end of the translated file.

The annotation is connected to the function in either of two ways. The ANNOTATE statement provides a default remark for PORTARE statements:

ANNOTATE *string*;

For example, the following sequence defines some annotation for Chapter 5 of the source LISP manual and then provides three translations, the first two of which use the default annotation and the last of which has a specific remark:

```
ANNOTATE "Chapter 5";
NOEQUIV BOOGIE, BANGER;
ASIS CRANK, GRIND, FLAIL;
EDIT "Very hard form, page 43, chapter 5" BROBDINGNAGIAN;
```

2.2 NUMBER OF ARGUMENTS

Some LISPs (Interlisp in particular) permit calling functions with more or fewer actual parameters than the number of formal parameters. To translate one of these LISPs into a LISP that requires the number of arguments to match, the translator must provide a mechanism for

remembering the number of formal parameters for each function and for supplying NIL when the number is deficient. In the descriptions that follow, the form *fn* corresponds to the following syntactic construct:

id[(*number*)]

Here *id* names some function being translated (in the source LISP), and *number* enumerates actual parameters the function should have. If a number is not supplied, it is assumed that the function can have any number of arguments.

The NIL FILL command determines whether or not functions with missing arguments should have them filled in and whether or not a message should be printed saying that the action has occurred:

NIL FILL (ON | OFF)[,] [SHORT (ON | OFF)];

If NIL FILL is ON, function calls that do not have enough arguments will be filled; if it is OFF, nothing will be done about them. If the SHORT option is ON, a message will be displayed each time a function does not have enough arguments; if it is OFF, no message will be displayed.

2.3 FUNCTIONS THAT ARE THE SAME: ASIS AND WARNING

The ASIS command enumerates a set of functions that are left "as is" by the translation process. These functions may have zero or more arguments, but the arguments must be in the usual form: (f a[0] a[1] ... a[n]). COND is an example of a function that cannot be ASIS because its arguments are lists of lists:

ASIS [*string*] [[*fn*],* *fn*];

The ASIS command accepts an optional string argument (the annotation connected with the function) and a set of identifiers separated by commas. The statement is terminated by a semicolon.

If the source LISP is PSL and the target LISP is Interlisp, the following translations are necessary:

```
ASIS "Page 7.2" CAR, CDR,
    CAAR, CADR, CDAR, CDDR,
    CAAAR, CAADR, CADAR, CADDR,
    CDAAR, CDADR, CDDAR, CDDDR,
    CAAAAR, CAAADR, CAADAR, CAADDR,
    CADAAR, CADADR, CADDAR, CADDDR,
    CDAAAR, CDAADR, CDADAR, CDADDR,
    CDDAAR, CDDADR, CDDDAR, CDDDDR;
```

A form very similar to ASIS performs an equivalent translation and also displays a warning about possible violation of the semantics:

WARNING [*string*] [[*id*],* *id*];

The warning message is displayed prior to the form in which the translation occurs. It is in the form of a comment in the target LISP, the first characters of which are three asterisks (more severe translation problems are signaled by five or seven asterisks).

This translation most often occurs when a target LISP support function does not have the complete semantics of the source LISP function. The PSL support routine for the Interlisp FILEPOS function does not support the complete Interlisp semantics. The following rule occurs in the Interlisp-to-PSL translation file:

```
WARNING "FILEPOS: Can't read/write at the same time."
                FILEPOS(7);
```

An occurrence of FILEPOS in an Interlisp file would cause the following comment to appear in the translated program:

```
(* *** FILEPOS: Can't read/write at the same time. )
```

2.4 FUNCTIONS FOR WHICH NO EQUIVALENT EXISTS: NOEQUIV

The NOEQUIV statement is used to enumerate functions of the source LISP for which no reasonable form exists in the target LISP. This does not include functions that will be part of the run time package. The NOEQUIV designation is used for functions that will not be used in programs intended to be portable. Examples of such functions are not difficult to find; they are usually device-specific I/O routines, functions for manipulating internal structure of LISP data items, functions in packages for which the sources are not available, etc. Annotation is particularly important on NOEQUIV statements, as a comment is printed in the translated LISP program in which the function occurs. The comment lets the implementer know exactly where to look for a definition of the function in the source LISP manual:

NOEQUIV [*string*] [[*id*],* *id*];

As an example, consider the Interlisp RESETVARS function, which has no reasonable equivalent in PSL. The *.porto* statement to signify this is:

```
NOEQUIV "Chapter 3" RESETVARS;
```

The following Interlisp function would be translated into a PSL function with annotation marking the lack of a change:

```
(DEFINEQ (BANG (LAMBDA (X) (RESETVARS X]

          to

(* RESETVARS no form, see 'Chapter 3')
(DE BANG (X) (RESETVARS X))
```

2.5 LINK TO THE SYSTEM EDITOR: EDIT

Functions enumerated by this form will cause the system structure editor (whatever it might be) to be invoked, with the form in which the named function occurs as an argument. The arguments of the function are not processed before the editor is called and must be mechanically processed during editing by calling the PORTARE function *doform* on them:

EDIT [*string*] [[*id*],* id] ;

2.6 STOPPING THE SCAN: STOPSCAN

Normally the translation process walks the program tree to the atomic level where nonfunction names are left as they are. In some cases, it is necessary to stop this process at a higher level. The STOPSCAN statement provides a means of stopping the translation at the function level:

STOPSCAN [*string*] [[*id*],* *id*];

A good example of a function that should stop the scanning process is the QUOTE function. The FUNCTION function is not a candidate for STOPSCAN because the its argument is normally an evaluable LISP expression. Systems that use QUOTE in place of FUNCTION will need to use a pattern (see later) for QUOTE that causes translation of quoted LAMBDA expressions.

2.7 IGNORING FUNCTIONS: IGNORE

Occasionally a LISP system will have declarations to internal functions which have no analog in the target system and are not useful in the LISP environment of the target LISP. These forms can be excluded from the translation scheme by the IGNORE feature.

The difference between functions that are ignored and those for which no equivalent exists (see the NOEQUIV statement) is that the ignored functions do not appear in the translated file; they are not even replaced with NIL (which is why they are treated only at the top level). On the other hand, functions for which no equivalent exists are copied directly to the translated file, and only their arguments are translated:

IGNORE [*string*] [[*id*],* *id*];

2.8 RENAMING FUNCTIONS: =>

This form specifies a source-to-target name translation. Every occurrence of the source name is replaced by the target name in the translated file.

id_{source} => fn_{target};

As an example, consider the Interlisp DSUBST function, an equivalent to the PSL function SUBSTIP. Its *.porto* command looks like:

```
DSUBST => SUBSTIP(3);
```

Since PORTARE is case-sensitive, MacLisp derivatives will need name changes on many functions but will not usually need the number of arguments:

```
CAR => car;
CDR => cdr;
```

2.9 PATTERN-DIRECTED TRANSLATION

In many cases, it is necessary to do more than just translate from source to target function names. Some form of argument translation other than the default is also required. PORTARE includes a syntax for accessing the Little Meta pattern matcher:

id: [*rule* -> *action*],* *rule* -> *action*;

A pattern-directed translation is named after the function being translated. It consists of a number of rule-action pairs separated by commas, with the whole statement terminated by a semicolon. The rule describes when the action side should be the translation. A rule is any of the following.

- *identifier*: If an identifier appears in the rule, it must be matched exactly with an identifier in the form being matched, for the rule to succeed (PORTARE is case-sensitive).
- *number*: If a number appears in the rule, it must be matched exactly with a number in the form being matched, for the rule to succeed.
- *string*: Same as numbers and identifiers.
- *&n*: This form will match any LISP expression, atomic or otherwise. Furthermore, the &n item will be bound to the matched expression and can be used in the action side of the rule. Here, n is any positive integer greater than 0. Note that &0 will be bound to the entire expression if a match occurs.
- **n*: This form functions like &n, except that it matches any number of expressions. If present, *n must be the last pattern primitive in any list as in (LIST &1 *2), though it can be used more than once in an expression as in (LIST (BANG &1 *2) &3 *4).

- *&<.../.../...>n*: This form will match anything determined by the evaluation of the expressions inside the brackets. Thus, &n is bound to the expression to be matched, and each of the forms inside the brackets is evaluated until one is true or all are false. If all are false, the match fails; if at least one is true, the match succeeds and &n is bound to the expression matched.
- *(...)*: Any combination of parentheses and tests can be enclosed in parentheses. The structure of the expression being tested must be the same as the structure of the rule, for the match to succeed.

During a pattern match, the patterns are tried one at a time in the order of their appearance until one succeeds, at which time the action side is executed. An action is a set of expressions for output of text or generation of translated expressions. The following forms are implemented.

An output clause is a series of expressions that are printed when the clause is encountered in an action side:

=>(: [*expression*],* *expression*)

An *expression* is any of the following:

- *id*: The value of the identifier is displayed in READ-readable format.
- *number*: The value of the number is displayed.
- *string*: The string is displayed without the enclosing double quotation marks.
- *&n*: The value of the pattern variable &n is displayed in READ-readable format.
- *COL:n*: Enough blanks are displayed to move to column number n.
- *SPACE:n*: n spaces are displayed.
- /: A TERPRI is executed; TERPRI is not automatic at the end of the output clause.

- *S-expression*: The LISP expression is evaluated, and its result is displayed in READ-readable format.

The other form of expression allowed at the top level of an action is a LISP S-expression template for the translated expression. The elements of this expression can be any of the following:

- *identifier*: The identifier will be copied into the resulting structure.
- *number*: The number will be copied into the resulting structure.
- *string*: The string will be copied into the resulting structure.
- *&n*: The value bound to &n during the pattern match will be placed into the resulting structure.
- *^n*: The value bound to &n will be translated and then placed into the resulting structure.
- *(...)*: Denotes a substructure.
- *-expression*: The expression (any of the above) is spliced into the resulting expression rather than just replaced.
- *=expression*: The expression is evaluated and its value is placed in the template.

The translation of the Interlisp PRINT into PSL PRINT is demonstrated in the following example:

```
PRINT:
  (PRINT) -> (PRINT NIL),
  (PRINT &1) -> (PRINT ^1),
  (PRINT &1 &2) -> (CHANNELPRINT ^2 ^1),
  (PRINT &1 &2 &3) -> =>((pCOMSTART),
                         "***** no read tables for PRINT",
                         (pCOMEND))
                       (CHANNELPRINT ^2 ^1);
```

Since Interlisp substitutes NIL for missing arguments, and PSL does not permit this practice, all four possibilities must be taken care of. The fourth case illustrates how illegal forms can be dealt with. The two functions pCOMSTART and pCOMEND will be explained later.

If no rule in a pattern succeeds, PORTARE will generate its own error message in the form of a comment in the target LISP dialect. The arguments of the unmatched form are translated in the usual manner (like the ASIS type), and the translation proceeds.

2.10 SPECIAL FORMS: SPECIAL

There are some forms that are not amenable to translation with the use of the pattern matcher. These functions are generally FEXPRs or MACROs which have an arbitrary number of arguments or other functions in which the current context is important. The best examples of this type are PROG, LAMBDA, and COND. The connection between the translation phase and the source dialect name is done by the SPECIAL statement:

SPECIAL FOR id_{source} DO id_{eval};

Here, id_{source} is the name of the function to be translated, and id_{eval} is the function to evaluate to do the translation. These functions should be defined as part of the *.prepas* file which is loaded automatically during translation.

As an example of this type of translation, consider the COND FEXPR. In the *.porto* file, the form is declared as requiring special translation. By convention, these translation functions have the same special prefix used in naming the translation files:

```
SPECIAL FOR COND DO INT2PSL!-EXCOND;
```

The translation function is named INT2PSL!-EXCOND and appears in the *.prepas* file as follows (in PSL form):

```
(DE INT2PSL!-EXCOND (X)
  (CONS 'COND (INT2PSL!-SCCOND (CDR X))))
(DE INT2PSL!-SCCOND (X)
  (COND ((NULL X) NIL
        (T (CONS (MAPCAR (CAR X) 'doform)
                 (INT2PSL!-SCCOND (CDR X))))))
```

The INT2PSL!-SCCOND function scans each of the COND arguments and handles implied PROGNs correctly.

2.11 CONDITIONAL TRANSLATION: IF

Differing translation requirements and possibilities are met by the setting of flags prior to translation. These flags are translation-specific, with the default setting corresponding to the most general translation offered by the system. The IF statement and translation blocks control which of any number of translation options is selected. The IF statement is conventional in format:

IF *expression* **THEN** { *statements* }

or

IF *expression* **THEN** { *statements* }
ELSE { *statements* }

The *expression* is a simple infix expression with AND, OR, NOT, and function calls. The BNF description for *expression* follows (note that the ε in the description is the empty production, which always is successful):

```
<expression> ::= <id> := <expression> | <brel>
<brel> ::=
     <bexp> = <bexp> |
     <bexp> EQ <bexp> |
     <bexp> IN <bexp> |
     <bexp> < <bexp> |
     <bexp> > <bexp> |
     <bexp>
<bexp> ::= <bterm> <bexp'>
<bexp'> ::= OR <bterm> <bexp'> | ε
<bterm> ::= <bsec> <bterm'>
<bterm'> ::= AND <bsec> <bterm'> | ε
<bsec> ::= NOT <bquote> | <bquote>
<bquote> ::= '<blist> | <bprimary>
<blist> ::= <id> | <unsigned-integer> | ( <blist-elts> )
<blist-elts> ::= <blist-elts> <blist> | <blist>
<bprimary> ::= <id> | <id> ( <expression-list> ) |
     <unsigned-integer> | ( <expression> )
<expression-list> ::= <expression-list>, <expression> |
     <expression>
```

The = relation is translated into the LISP EQUAL function, and IN is translated into a MEMQ test. Function calls consist of a name followed by arguments enclosed in parentheses and separated by commas.

The translation from Franz Lisp to PSL relies on characteristics of the target system hardware. In Franz Lisp, car and cdr of nil return nil, and this is also true for VAX PSL. However, CAR and CDR of NIL in Motorola 68000 and DEC 20 implementations cause an error, and hence special support routines are required. To exploit the efficiency of CAR and CDR on the VAX and yet maintain Franz Lisp semantics on the 68000, the following (partial) translation is used. In the following translation, CAR!*, CDR!*, and the composite forms are special functions which check for atomic arguments and return the appropriate value:

```
IF CxRNIL OR NOT VAXPSL THEN
   {car => CAR;  cdr => CDR;
    caar => CAAR;  cadr => CADR;  cdar => CDAR;  cddr => CDDR;
    caaar => CAAAR;  caadr => CAADR;  cadar => CADAR;
    caddr => CADDR;  cdaar => CDAAR;  cdadr => CDADR;
    cddar => CDDAR;  cdddr => CDDDR }
ELSE
   {car => CAR!*;  cdr => CDR!*; caar => CAAR!*;
    cadr => CADR!*;  cdar => CDAR!*;  cddr => CDDR!*;
    caaar => CAAAR!*;  caadr => CAADR!*;  cadar => CADAR!*;
    caddr => CADDR!*;  cdaar => CDAAR!*;  cdadr => CDADR!*;
    cddar => CDDAR!*;  cdddr => CDDDR!* }  ;
```

2.12 TRANSLATION TIME EXPRESSIONS

The <*expression*> syntax above can be used to define and set variables during the translation loading process. The syntax is:

```
. expression ;
```

3. ".PREPAS" SYNTAX CONVERSION AND SPECIAL TRANSLATIONS

Many LISPs are not syntax-compatible with each other. For example, Interlisp uses] and [as special parentheses, whereas PSL uses them to denote vectors. Likewise, annotation conventions vary considerably across dialects. To counter this problem, a scan of the source LISP file is made converting the source dialect into something that can be read by the PSL READ function. Extensions to the translation scheme (the functions marked by the SPECIAL command) are also included in the *.prepas* file. This section describes the functions that must be implemented as part of this file.

3.1 PREPASS FUNCTIONS

The prepass section should contain a definition of a function called PASS1PROC with two arguments. The first is the file handle of the input file to be translated, and the second is the handle of a file in which the results of the prepass are placed. These files will already have been opened when control is transferred to PASS1PROC. They should not be closed by PASS1PROC. Unless the source and target LISP are sufficiently similar, PASS1PROC will probably be a character at a time translation.

As an example, the Interlisp-to-PSL translation routine performs the following three actions on the source LISP program:

- Counts parentheses and expands [...] super parentheses into the correct number of standard (...) parentheses.
- Converts the Interlisp % escape character into the PSL ! escape character.
- Puts escape characters in Interlisp identifiers to make them compatible with PSL identifiers.

3.2 EXTENDED TRANSLATION FUNCTIONS

The second part of the *.prepas* file is a collection of functions for translating special forms as declared by the SPECIAL statement in the *.porto* file. As described previously, each of these functions is passed a single value: the form to be translated, including the function name. It should return the translated form.

3.3 TARGET LISP INTERFACE FUNCTIONS

Three functions should be defined as part of the *.prepas* file to interface the translation process and the host LISP system to the translator:

- *pCOMSTART*: This function has no arguments. It should display a comment-starting sequence for the target LISP on the currently selected output file.
- *pCOMEND*: This function should display a comment-ending string on the currently selected output file.
- *pEDIT*: This function has one argument. It should call the system structure editor on the value of this argument and should return the edited value.

As an example, consider the PSL target support of these functions. PSL is modified to support the Interlisp (*...) style of comment. The PSL structure editor is called EDIT:

```
(DE pCOMSTART () (PRIN2 "(* "))
(DE pCOMEND () (PRIN2T " )")
(DE pEDIT (x) (EDIT x))
```

4. ".EXTEND" EXTENSIONS TO THE TARGET LISP

Normally there are many functions that have no reasonable equivalence in the target LISP. To solve this problem, a collection of functions is built to provide a source LISP environment in the target LISP. Two conventions should be adhered to:

- Functions in the source LISP that have the same name as but do not translate into a function name in the target LISP have defined for them a special target LISP function whose name is prefixed with 'p-'.
- Functions for which no name clash exists are defined in the *.extend* file with the source LISP name.

The *.extend* file also should contain any changes to the target LISP environment to make it more compatible with the source LISP. An example of this type of action is an extension to PSL to accept Interlisp-style comments.

5. ".TRANS" FOR SEGMENTED PACKAGE TRANSLATION

The subdivision of large LISP programs into interdependent source modules poses a problem for the translation implementer. PORTARE transfers translation information between modules through *.trans* files which contain LISP expressions to evaluate during translation initiation. The contents initialize the translation environment with function names and required number of arguments, as well as data structure details for complex translation.

The contents of the *.trans* file are generated by functions in the extensions to the translation included in the *.prepas* file. The file is opened prior to the start of translation, and a header message is printed. The control routine saves the file handle in the global variable TRANSFILE. To write something to the file, the user first selects it, writes an S-expression, and then write-selects the previous output file. The *.trans* file is automatically closed when the translation is completed.

PORTARE keeps translation information about functions and their arguments on the property list of the function name under a number of different indicators. The following are important:

- **PORTARE**: This indicator has as its value the type of translation to perform for the function, either ASIS, WARNING, NOEQUIV, EDIT, STOPSCAN, or IGNORE. If the value is a list and the first element is SPECIAL, the second element will be the name of a special translation function. If the first element is NEWNAME, the second will be a translated name for the function. A first element of PTTRN indicates that a pattern-directed translation follows.
- **ANNOTATION**: This indicator contains any annotation associated with the function, default or otherwise.
- **NumberOfParameters**: A numeric value indicates the number of parameters a function must have. The value AnyNumber signifies that any number of parameters are accepted.

Other indicators are possible. The Interlisp-to-PSL translation stores much data-structure information under record names.

6. SETTING UP THE PORTARE ENVIRONMENT

For each possible translation, there should be four files with names set up as defined previously and with the following extensions:

- *.porto*: contains details of the translation.
- *.prepas*: contains the prepass and extensions to the translation.
- *.extend*: contains extensions to the target LISP to support a source LISP environment.
- *.b*: the compiled (UNIX PSL) version of the *.prepas* file.

The *abc2def.prepas* file should be compiled into a file called *abc2def.b* which will then be loaded during the translation setup. The *abc2def.extend* file should be compiled for loading with the translated source file, but its name is not critical.

Two other files are required for operation of the translator (for UNIX PSL):

- *portare.b*: The fast load (compiled/object) code for the translation control program.
- *lmeta.b*: The Little Meta translator writing system.

The main translation function, named PORTARE, has five parameters:

- *file*: the name of the file (a string) to be translated.
- *source-dialect*: the source dialect name (all lower case) as an identifier. The following names are used: interlisp, psl, franz, zeta.
- *target-dialect*: the target dialect (all lower case), using the same convention as *source-dialect*.
- *support-list*: a list of file names (strings) of *.trans* files generated during the translation of modules. These contain intermodule information for the translator.

- *startup*: a list of expressions to evaluate before the translation files are read. For example, global variables might be declared and assigned values for a conditional translation.

The following is an example run for translating the Interlisp program *ELLIE* into the PSL program *ellie.psl*:

```
psl
(OFF RAISE)
(LOAD lmeta portare)
(PORTARE "ELLIE" 'interlisp 'psl NIL NIL)
```

In this case, the Interlisp file *ELLIE* is translated into the file *ELLIE.psl*. Please note that (OFF RAISE) makes PSL case-sensitive.

It is possible to translate more than one file during a run. If the PORTARE function detects that the appropriate translation files have been loaded, the setup phase will be bypassed.

In the following example, the first translation creates a file of information which is used in the translation of the second:

```
psl
(OFF RAISE)
(LOAD lmeta portare)
(PORTARE "records" 'interlisp 'psl NIL NIL)
 ...
psl
(OFF RAISE)
(LOAD lmeta portare)
(PORTARE "support" 'interlisp 'psl
         '("records.trans")
         '((GLOBAL '(CxRNIL)) (SETQ CxRNIL T)))
```

The second translation (which can be repeated without retranslating "records") loads in the information in *records.trans*, declares the variable CxRNIL to be GLOBAL, and sets it to T so that (CAR NIL) is permitted in the translated program.

7. SPECIFIC TRANSLATION NOTES

Some of the translations have been worked on more than others and hence are more complete. This section describes features and deficiences of the translations that have been worked on to date.

7.1 INTERLISP -> PSL

This translation has received the most attention and is the most complete. It has translated the ROSIE programming environment into PSL from the original VAX Interlisp. The manuals used in the construction of the translation are *Interlisp Reference Manual*, Warren Teitelman, Xerox Palo Alto Research Center, October 1978, and *The Portable Standard LISP Users Manual*, Utah Symbolic Computation Group, January 1983.

The translation consists of the files:

- *int2psl.prepas*: the prepass and translation extensions.
- *int2psl.b*: the compiled version of *int2psl.prepas.*
- *interlisp.b*: the compiled version of *int2psl.extend* for run time support of Interlisp.
- *int2psl.porto*: translation specification.
- *basic.psl*: basic support routines for Interlisp run time support.
- *carcdr.psl*: CAR/CDR and composites PSL run time support for target systems that do not support (C...R NIL).
- *comment.psl*: PSL run time support for Interlisp (*...) comments.
- *for.psl*: PSL run time support for CLISP FOR macro and associated forms.
- *fpckg.psl*: PSL run time support for translated programs that require random access files.
- *hash.psl*: PSL run time support for translated programs that require hash tables.

- *int2psl.bld*: a PSL file that builds *interlisp.b*.
- *io.psl*: PSL run time support for Interlisp I/O (not including random access I/O).
- *resets.psl*: PSL run time support for Interlisp RESETxxx forms.
- *strings.psl*: PSL run time support for Interlisp string manipulation functions.

To run a translated program, the run time package *interlisp.b* must be loaded before the program is either compiled or executed. To compile a translated file into a PSL FASL file, the following should be done:

```
pslcomp
(LOAD interlisp)
(FASLOUT "fasl file name")
(DSKIN "translated file name")
(FASLEND)
```

Interlisp files should be remade with MAKEFILE, with the NOCLISP option set before translation is attempted. There are a number of deficiencies and translations that deserve attention:

- CAR, CDR, etc., are translated into CAR!* and CDR!* (part of the *interlisp.b* run time support package) because psl does not permit CAR or CDR of anything which is not a dotted-pair. This translation can be turned off if the target machine is a DEC VAX running UNIX, as this version of PSL does support this feature. To translate to CAR!* and its compatriots, enter the following:

```
(PORTARE file 'interlisp 'psl ...
    '((GLOBAL '(CxRNIL)) (SETQ CxRNIL T)))
```

- The prepass phase has the following restrictions: Atoms may not begin with a period (.) unless prefixed with a % escape character, and an asterisk (*) should not be the first element of a list that is terminated by a right bracket.
- Functions that have PSL names but for which PSL has different semantics are provided with an Interlisp interpretation in *interlisp.b* run time support. These functions have the Interlisp name prefixed with p- to indicate they are part of the portability package (with the exception of the composites of CAR and CDR). Functions without PSL counterparts maintain their Interlisp name. Thus Interlisp FOR becomes p!-FOR in PSL, but Interlisp L-CASE remains L-CASE.
- Most of the I/O functions in the translated code do not support the use of read tables.
- Not all of the iterative forms of FOR are supported, particularly REPEATWHILE, REPEATUNTIL, IN OLD ..., ON OLD ..., DECLARE, and ORIGINAL. Likewise, the internal structure is somewhat different, and GOs to the Interlisp hidden labels are not supported.
- Files cannot be opened in the UPDATE mode.
- RPAQ and RPAQQ are translated into SETQ and do not check the value of dfnflg.
- RESETSAVE must be in the scope of a RESETLST. There is no implied RESETLST at the top level. RESETLST does not compile open.
- Odd numbers of elements and lists not ending in NIL are not allowed in LISTPUT and related functions.
- The following Interlisp functions are not supported or translated: *DECLAREDATATYPE, FETCHFIELD, REPLACEFIELD, NCREATEM, GETFIELDSPEC, GETDESCRIPTORS, USERDATATYPES, MAKESYS, HERALD, SWPARRAYP, SCODEP, MKSWAP, MKUNSWAP, MKSWAPP, SETBSIZE, CONSTANT, GETTOPVAL, SETTOPVAL, ADDTOVAR, EQUALN, HCOPYALL, CSUBST, MERGE, ALPHORDER, MERGEINSERT, COMPARELIST, SAVEPUT, ADDPROP, REMPROPLIST, CHANGEPROP, PROPNAMES, GETLIS, MOVD, MOVD!?, CCODEP, SCODEP, ARGTYPE, ARGLIST, SMARTARGLIST,*

DEFINE, SAVEDEF, UNSAVEDEF, DEFEVAL, EVALA, RPT, RPTQ, FRPTQ, ARG, SETARG, DCHCON, U-CASEP, RSTRING, GNC, GLC, RPLSTRING, STRPOS, ARRAYTYP, ARRAYBEG, HARRY, HARRYSIZE, HARRAYP, SWPARRAYP, ELTD, SETD, TYPENAME, TYPENAMEP, TYPENAMEFROMNUMBER, TYPENUMBERFROMNAME, NTYP, TYPEP, GCMESS, GETTYPEDESCRIPTION, SETTYPEDESCRIPTION, STORAGE, GCTRP, CONSCOUNT, CLOSER, OPENR, MAPCONC, FUNARG, BLIPVAL, SETBLIPVAL, BLIPSCAN, STKPOS, STKNTH, STKNAME, SETSTKNAME, STKNTHNAME, DUMMYFRAME, REALFRAME, REALSTKNTH, STKSCAN, STKARG, STKARGNAME, SETSTKARG, SETSTKARGNAME, STKNARGS, VARIABLES, STKARGS, ENVEVAL, ENVAPPLY, STKEVAL, STKAPPLY, RETEVAL, RETAPPLY, RETFROM, RETTO, EVALV, STACKP, RELSTK, RELSTKP, CLEARSTK, CLEARSTKLST, NOCLEARSTKLST, COPYSTK, BACKTRACE, MAPDL, SEARCHPDL, SMALLP, GCD, ANTILOG, SIN, COS, TAN, ARCSIN, ARCCOS, ARCTAN, ARCTAN2, LOC, VAG, FULLNAME, OPENFILE, IOFILE, OPENP, GETFILEINFO, SETFILEINFO, DELFILE, RENAMEFILE, UNPACKFILENAME, FILENAMEFIELD, SEPRCASE, WHENCLOSE, RATOM, RATOMS, SETSEPR, SETBRK, GETSEPR, GETBRK, ESCAPE, RATEST, PEEKC, LASTC, READP, WAITFORINPUT, READLINE, SKREAD, PRINTBELLS, DOBE, PRINTLEVEL, PRINTNUM, NFORMATCODE, DEFPRINT, HPRINT, HREAD, HCOPYALL, READTABLEP, GETREADTABLE, SETREADTABLE, COPYREADTABLE, RESETREADTABLE, GETSYNTAX, SETSYNTAX, SYNTAXP, READMACROS, INREADMACROP, SETREADMACROFLG, TERMTABLEP, GETTERMTABLE, SETTERMTABLE, COPYTERMTABLE, RESETTERMTABLE, ECHOCONTROL, ECHOMODE, GETECHOMODE, DELETECONTROL, GETDELETECONTROL, RAISE, GETRAISE, CONTROL, GETCONTROL, CLEARBUFF, LINBUF, SYSBUF, BKLINBUF, BKSYSBUF, RESETBUFS, RADIX, FLTFMT, SETLINELENGTH, SETTERMCHARS, POSITION, SYSOUT, SYSIN, SYSOUTP, LOAD, LOAD!?, LOADFNS, LOADVARS, LOADFROM, LOADBLOCK, LOADEFS, LOADCOMP, LOADCOMP!?, PRETTYPRINT, PP, PF, GETCOMMENT, MAKEFILES, LISTFILES, COMPILEFILES, CLEANUP, FILES!?, WHEREIS, MARKASCHANGED, UNMARKASCHANGED, FILEPKGCHANGES, ADDTOFILES!?, GETDEF, PUTDEF, COPYDEF, DELDEF, SHOWDEF, EDITDEF, SAVEDEF, UNSAVEDEF, LOADDEF, CHANGECALLERS, RENAME, COMPARE, COMPAREDEFS, HASDEF, TYPESOF, FILEPKGTYPE, FILEPKGCOM, INFILECOMS!?, ADDTOFILE, DELFROMFILES, ADDTOCOMS, DELFROMCOMS, MAKENEWCOM, MOVEITEM, FILECOMSLST,

FILEFNSLST, FILECOMS, SMASHFILECOMS, PRETTYDEF, PRINTFNS, PRINTDATE, PRETTYHEADER, FILEDATE, PRETTYCOMPRINT, PRINTDEF, COMMENTL, BREAK1, BREAKO, BREAK, TRACE, BREAKIN, UNBREAK, UNBREAKO, UNBREAKIN, REBREAK, BREAKREAD, CHANGENAME, VIRGINFN, BAKTRACE, BAKTRACELST, ERRORX, HELP, RESET, ERRORN, SETERRORN, ERRORMESS, ERRORSTRING, ERSETQ, NLSETQ, INTERRUPTCHAR, INTERRUPTABLE, INTERRUPTABLEP, DWIM, DWIMIFY, ADDSPELL, MISSPELLED!?, FIXSPELL, FNCHECK, SPELLFILE, FINDFILE, ASKUSER, MAKEKEYLST, COMPILE, COMPILEL, TCOMPL, RECOMPILE, EXPANDMACRO, RELINK, BLOCKCOMPILE, BLKNAME, BCOMPL, BRECOMPILE, ADVISE, UNADVISE, READVISE, ADVISEDUMP, GETTEMPLATE, SETTEMPLATE, CALLS, CALLSCCODE, FREEVARS, MASTERSCOPE, SETSYNONYM, PARSERELATION, GETRELATION, TESTRELATION, MAPRELATION, UPDATEFN, UPDATECHANGED, MSMARKCHANGED, DUMPDATABASE, UNSAVEFNS, TIME, DATE, IDATE, GDATE, CLOCK, DISMISS, CONSCOUNT, BOXCOUNT, GETRP, PAGEFAULTS, GETBLK, RELBLK, SUBSYS, KFORK, FILDIR, LOADAV, ERSTR, JSYS, USERNAME, USERNUMBER, HOSTNAME, HOSTNUMBER, SYSTEMTYPE, TENEX, DIRECTORY, OPNJFN, GTJFN, RLJFN, JFNS, OPENF, ADDBUFFER, MAPBUFFERCOUNT, MAPPAGE, MAPWORD, WORDOFFSET, WORDCONTENTS, SETWORDCONTENTS, CLEARMAP, LOCKMAP, UNLOCKMAP, DRIBBLE, DRIBBLEFILE, DISPLAYTERMP, GAINSPACE, VALUEOF, TESTMODE, LISPX, USEREXEC, LISPXREAD, LISPXREADP, LISPXUNREAD, PROMPTCHAR, LISPXEVAL, HISTORYSAVE, LISPXSTOREVALUE, LISPXFIND, HISTORYFIND, HISTORYMATCH, ENTRY!#, CHANGESLICE, SAVESET, UNSET, UNDOSAVE, !/RPLNODE, !/RPLNODE2, NEW!/FN, LISPX!/, UNDOLISPX, UNDOLISPXL, UNDONLSETQ, RESETUNDO, PRINTHISTORY, LISPXSTATS, ADDSTATS, LISPXWATCH, GREET, SWAP, CHANGE, MULTIFILEINDEX, SINGLEFILEINDEX, DUMPDB, LOADDB, WHEREIS, WHEREISNOTICE, JS, XWD, BIT, JSYSERROR, SCRATCHLIST, ADDTOSCRATCHLIST, JOB!#, TTY!#, DETACH, DETACHEDP, LINKTOTTY, LINKTOUSER, BREAKLINKS, CNDIR, !/DELFILE, !/UNDELFILE, EXPUNGE, COPYALLBYTES, DSKSTAT, MEMSTAT, GETPASSWORD, TELNET, FTP, MAKENEWCONNECTION, CLOSECONNECTION, CHECKCONNECTION, NETSERVER, NETUSER, FORCEOUT, CREATEHASHFILE, OPENHASHFILE, HASHFILEP, PUTHASHFILE, GETHASHFILE, HASHFILEPROP, HASHFILENAME, CLOSEHASHFILE, MAPHASHFILE, REHASHFILE, COPYHASHFILE,

HASHFILEPLST, LOOKUPHASHFILE, GETPAGE, DELPAGE, GETPNAME, DECLTYPE, COVERS, SUBTYPES, SUPERTYPES, DECLOF.

7.2 PSL -> INTERLISP

This translation has been partially tested but no large programs have been translated. The manuals used in the construction of the translation are the same as those used in the inverse translation. The following deficiencies are noted:

- PSL comments started by % are removed from the source.
- The first pass creates a FILECOMS variable which may be incorrect where FLUIDs and some top-level forms are concerned. This part needs to be restructured to maintain consistency with PSL.
- The set of translated functions is not complete. The following PSL functions are not correctly translated: *NEWID, INT2ID, ID2INT, ID2STRING, STRING2LIST, LIST2STRING, STRING, VECTOR, VECTOR2STRING, STRING2VECTOR, VECTOR2LIST, LIST2VECTOR, DECR, CEILING, FLOOR, ROUND, MOD, DEGREESTORADIANS, RADIANSTODEGREES, RADIANSTODMS, DMSTORADIANS, DEGREESTODMS, DMSTODEGREES, COT, COTD, SEC, SECD, CSC, CSCD, ACOT, ACOTD, ASEC, ASECD, ACSC, ACSCD, EXP, LOG2, LOG10, FACTORIAL, STRINGGENSYM, REMOB, INTERNP, FINDPREFIX, FINDSUFFIX, REMPROPL, DESETQ, PSETQ, SETF, PSETF, MAKEUNBOUND, !\CREATEPACKAGE, !\SETPACKAGE, !\PATHINTERNP, !\PATHINTERN, !\PATHREMOB, !\PATHMAPOBL, !\LOCALINTERNP, !\LOCALINTERN, !\LOCALREMOB, !\LOCALMAPOBL, ON, OFF, ADJOIN, ADJOINQ, UNIONQ, INTERSETCTIONQ, LIST2SET, LIST2SETQ, DELETE, DEL, DELETEIP, DELQ, DELQIP, DELASC, DELASCIP, DELATQ, DELATQIP, ASS, SASSOC, PAIR, SUBLA, MAKE!-STRING, MKSTRING, STRING, COPYSTRINGTOFROM, COPYSTRING, VECTOR, COPYVECTORTOFROM, ISIZES, IGETS, IPUTS, MAKE!-WORDS, MAKE!-HALFWORDS, MAKE!-BYTES, SIZE, INDX, SETINDX, SUB, SETSUB, SUBSEQ, SETSUBSEQ, CONCAT, TOTALCOPY, STANDARD!-CHARP, GRAPHICSP, STRING!-CHARP, ALPHAP, UPPERCASEP, LOWERCASEP, BOTHCASEP, DIGITP, ALPHANUMERICP, CHAR!=, CHAR!-EQUAL, CHAR!<, CHAR!>, CHAR!-LESSP, CHAR!-GREATERP, CHAR!-CODE, CHAR!-BITS,*

CHAR!-FONT, CODE!-CHAR, CHARACGTER, CHAR!-UPCASE, CHAR!-DOWNCASE, CHAR!-INT, INT!-CHAR, RPLACHAR, STRING!=, STRING!-EQUAL, STRING!<, STRING!>, STRING!<!=, STRING!>!=, STRING!<!>, STRING!-LESSP, STRING!-GREATERP, STRING!-NOT!-GREATERP, STRING!-NOT!-LESSP, STRING!-NOT!-EQUAL, STRING!-REPEAT, STRING!-TRIM, STRING!-LEFT!-TRIM, STRING!-RIGHT!-TRIM, STRING!-UPCASE, NSTRING!-UPCASE, STRING!-DOWNCASE, NSTRING!-DOWNCASE, STRING!-CAPITALIZE, NSTRING!-CAPITALIZE, STRING!-TO!-LIST, STRING!-TO!-VECTOR, SUBSTRING, STRING!-LENGTH, IF, WHEN, UNLESS, CASE, WHILE, PREPEAT, NEXT, EXIT, FOR, FOR!, FOREACH, DO, DO!*, DO!-LOOP, DO!-LOOP!*, LET, LET!*, CATCH, THROW, CATCH!-ALL, UNWIND!-ALL, UNWIND!-PROTECT, COPYD, REMD, DE, DF, DN, DM, DS FUNBOUNDP, FLAMBDALINKP, FCODEP, MAKEFUNBOUNDP, MAKEFLAMBDALINK, MAKEFCODE, GETFCODEPOINTER, CODE!-NUMBER!-OF!-ARGUMENTS, EXPRP, FEXPRP, NEXPRP, MACROP, FLUID, GLOBAL, UNFLUID, FLUIDP, GLOBALP, UNBOUNDP, UNBINDN, LBIND1, PBIND1, CLOSURE, EVALINENVIRONMENT, APPLYINENVIRONMENT, CAPTUREENVIRONMENT, RESTOREENVIRONMENT, CLEARBINDINGS, EVLIS, LAMBDAAPPLY, LAMBDAEVALAPPLY, CODEAPPLY, CODEEVALAPPLY, IDAPPLYO, IDAPPLY1, IDAPPLY2, IDAPPLY3, IDAPPLY4, EVPROGN, EXPAND, FILEP, SHUT, EVSHUT, OUT, EVOUT, CHANNELPRINTF, PRINTF, ERRORPRINTF, CHANNELPOSN, CHANNELLPOSN, LPOSN, CHANNELLINELENGTH, RPRINT, PRETTYPRINT, PRIN2L.*

- The translator should include patterns for some of the p!-functions created by the Interlisp -> PSL translator so that reverse translation is possible.

7.3 FRANZ LISP -> PSL

The manuals used in construction of the translation are *The Portable Standard LISP Users Manual*, Utah Symbolic Computation Group, January 1983, and *The FRANZ LISP Manual*, J. K. Foderaro and K. L. Sklower, April 1982. The translation consists of the following files:

- *fra2psl.porto*: the translation specification; does not include pass-through code for nested translations for any other LISPs.
- *fra2psl.prepas*: the prepass section and a redefinition of the PASS2PROC and doatom functions of PORTARE to keep PSL style comments and to translate T and NIL into upper case.
- *fra2psl.b*: the compiled version of *fra2psl.prepas*
- *fra2psl.extend*: a set of more than 70 functions that serve as extensions to PSL to support Franz Lisp functions that have no reasonable counterparts in PSL.
- *franz.b*: compiled version of *fra2psl.extend.*

The following are known deficiencies of the translation:

- Only a very small subset of the different array types are supported. More of the forms can be implemented with PSL word-arrays, but this could be very tricky.
- Big numbers are not supported as a separate data type. Rather, the integer data type includes both small fixed integers and big numbers without distinguishing between them.
- The sorting functions *insert* and *merge* are not implemented.
- The following functions (in order of occurrence in the manual) are not implemented: *quote!, bignum-to-list, list-to-bignum, insert, merge, lsubst, hunkp, signp, copysymbol, rematom, get-pname, getchar, nthchar, getcharn, substring, substringn, makunbound, marray, *array, array, getaccess, getaux, getdelta, getdata, getlength, arraycall, listarray, putaccess, putaux, putdata, putdelta, putlength, fillarray, hunk, makhunk, *makhunk, hunkp, hunksize, cxr, rplacx, *rplacx, putdisc, getl, bcdad, copyint*, cpyl, getaddress, ptr, replace, scons, sort, sortcar, Emuldiv, haipart, haulong, bignum-leftshift, sticky-bignum-leftshift, rot, *mod, arg, break, *break, *catch, cvttointlisp, cvttomaclisp, cvttoucilisp, debug, debugging, evalframe, evalhook, exec, exece, freturn, frexp, funcallhook, getdisc, I-throw-error, listify, mfunction, progv, purcopy, purep, setarg, throw, *throw, unwind-protect, cfasl, drain,*

*ffasl, filepos, fseek, nwritn, portp, probef, removeaddress, resetio, setsyntax, allocate, argv, chdir, fake, fork, gcafter, getenv, hashtabstat, maknum, monitor, opval, process, restorelisp, retbrk, *rset, segment, shell, showstack, signal, sizeof, small-segment, sstatus, status, syscall, wait.*

- Semicolon comments inside functions will be lost, and top-level comments will be translated into the PSL % comment form.
- The eval-when construct requires its arguments to always be in the order compile, eval, load.