The research reported in this paper was sponsored by the Advanced Research Projects Agency Information Processing Techniques Office and was monitored by the Electronic Systems Division, Air Force Systems Command under contract F1962867C0004, Information Processing Techniques, with the System Development Corporation.

TECH MEMO *a working paper*

System Development Corporation / 2500 Colorado Avenue / Santa Monica, California 90406
Information International Inc. / 11161 Pico Boulevard / Los Angeles, California 90064

TM- 3417/360/00

AUTHOR Jeff Barnett
J. Barnett

TECHNICAL Jeff Barnett
J. Barnett

RELEASE [illegible]
C. Weissman, S.D.C.
[illegible]
D. Anschultz, I.I.I.
for J. I. Schwartz

DATE 4/26/67 PAGE 1 OF 16 PAGES

(Page 2 is blank)

LISP 2 Compiler Type Resolver Language and Processor Specifications

ABSTRACT

This document describes the language and processor required for the Type Resolver pass of the LISP 2 compiler proposed for the IBM S/360 computer. The Type Resolver (pass III of the LISP 2 compiler) is used to transform CRIL input into TRIL by making type conversions and confluence-point branches explicit.

TABLE OF CONTENTS

1. INTRODUCTION

Type-Resolved Interlude Language (TRIL), the output language of the Type Resolver pass of the LISP 2 compiler, bears a strong resemblance to CRIL. In TRIL, all confluence point branches and type conversions are explicit. Also, arithmetic forms such as addition and multiplication have been parsed into floating, fixed, and general groupings.

A type consists of both a format (integer, general, real, etc.), and a precision (the number of bits comprising the datum). Type equality is not a useful consideration in the description of TRIL; rather, an inclusive relation is necessary. We say $type_1 \varepsilon\ type_2$, if and only if one of the following conditions is met:

let p_1 = the precision of $type_1$

p_2 = the precision of $type_2$

f_1 = the format of $type_1$

f_2 = the format of $type_2$

1. $f_1 = f_2$ and $p_1 \geq p_2$
2. f_1 = either signed integer or unsigned integer, and

   f_2 = either signed integer or unsigned integer and $p_1 \geq p_2$
3. f_1 = array subtype and

   f_2 = array and $p_1 \geq p_2$ (similarly for records and tables)
4. f_1 = array or record or table, and

   f_2 = general and $p_1 \geq p_2$

If $type_1 \varepsilon\ type_2$, then a need for a datum of $type_2$ may be satisfied by a datum of $type_1$. This relationship is transitive, i.e., if $t_1 \varepsilon\ t_2$ and $t_2 \varepsilon\ t_3$, then $t_1 \varepsilon\ t_3$.

2. TYPE-RESOLVED INTERLUDE LANGUAGE

The TRIL language may best be described as a series of additions to and deletions from the CRIL language. The deletions from CRIL are as follows:

plus

times

minus

terminal

pterm

return

preturn

cond

drive

locof (reduces to ℓfield)

The additions to CRIL (and their syntax and semantics) are described below.

convert = (CONVERT type type expression)

Convert calls for conversion of the datum produced by evaluating the embedded expression from the first type specified to the second type specified. For example,

(CONVERT REAL INTEGER (DATUM REAL (NUMBER 3.0))) = 3

The following three forms handle fixed-point arithmetic in the obvious way:

iplus = (IPLUS integer-expression$_{*+1}$)

itimes = (ITIMES integer-expression$_{*+1}$)

iminus = (IMINUS integer-expression)

The following three forms handle floating-point arithmetic in the obvious way:

rplus = (RPLUS real-expression$_{*+1}$)

rtimes = (RTIMES real-expression$_{*+1}$)

rminus = (RMINUS real-expression)

The following forms handle general arithmetic in the obvious way:

```
gminus = (GMINUS general-expression)
gplus = (GPLUS atype type general-expression expression)
gtimes = (GTIMES atype type general-expression expression)
gdiff = (GDIFF atype type general-expression expression)
diffg = (DIFFG atype type general-expression expression)
atype = type
```

Atype is the value type of a form for doing general arithmetic. The other type is the type of the expression to be arithmetically combined with the general-expression. Gminus is for unary minus, and in fact will be the output of the Type Resolver, not gdiff and diffg, which will be introduced by the machine link.

For example, to specify an integer value for a general minus real expression requires the following form:

```
(GPLUS INTEGER REAL general (RMINUS real))
```

In the TRIL form,

```
fork = (FORK boolean-expression {label|NIL} {label|NIL})
```

the embedded boolean-expression is evaluated. If the value is true, control is transferred to the first label. If NIL is used instead of a label, the fork statement falls through. A similar action is taken with the second label if the result of the evaluation is false.

The syntax of the TRIL net expression is as follows:

net = (NET type statement confluence-plex)

confluence-plex = (confluence-point$_{*+1}$)

confluence-point = (label ctype type {label|NIL})

ctype = type|datum

evalgo = (EVALGO label expression)

A net is an expression with a very complex rule of evaluation. The type of datum produced as a value is explicitly given by the outer type (the one between the word "NET" and the statement). The embedded statement is executed. Evalgo's, go's, or computed go's executed within the statement may reference labels within the confluence-plex.

Let (L_1 T_1 T_2 L_2) be a member of the confluence-plex. When label L_1 is referenced by an evalgo, etc., control is transferred to a place unique to this label (determined by subsequent compiler passes) with a datum of the type specified by T_1 in the heaven-box. If T_1 is an explicit datum, then such a datum is to be moved to the heaven-box at this time. A conversion of the datum (in the heaven-box) of type T_1 to type T_2 is now performed. Program control now passes to label L_2. L_2 may designate another label in the confluence-plex or may be NIL. If it is NIL, this implies a fall-through of the entire net. In the fall-through case, the net-type $T_N \ni T_2$. Also, if the expression body of an evalgo has type T_E, then $T_E \epsilon T_1$.

The syntax of the TRIL pnet expression is as follows:

```
          pnet = (PNET dlabel dlabel elab elab elab elab statement)
          elab = label|NIL
          dlabel = label|NIL
```

A pnet may be viewed as a traffic director for predicates. The two dlabels specify places to transfer program control on true and false evaluations. Either may be NIL, specifying a fall-through of the pterm on the appropriate condition. The four elabels are to be used for GGO, NGGO, BGO and NBGO.*
Further compiler passes will place these labels at appropriate spots and make the branch test on the contents of the heaven-box (which is loaded with a datum by an evalgo). Labels outside a pnet are visible to statements within the pnet.

*Reference to the BGO label puts a boolean datum in the heaven-box; reference to the NBGO label puts an inverted boolean datum in the heaven-box; reference to the GGO label puts a general datum in the heaven-box; reference to the NGGO label puts an inverted general datum in the heaven-box.

3. TYPE RESOLVER

The Type Resolver is a one-pass transducer that changes CRIL input to TRIL output. A major amount of branch optimization is done, and the number of type conversions produced is minimal. As type resolution proceeds, tailing of form operators is changed from CRIL. to TRIL.. Arbitrary nesting of CRIL and TRIL may exist through this pass. However, at completion of this pass, only TRIL will be left. The input CRIL listing is assumed to be made of totally unique list structures, and as such may be RPLACA'ed and RPLACD'ed. The output of the Type Resolver is also unique list structure, so that subsequent compiler passes may change the contents of nodes. Appropriate error checking is done and diagnostics are issued.

3.1 TYPE RESOLVER UTILITY FUNCTIONS

3.1.1 CRTYPE

The CRTYPE function is provided with the following declaration:

```
(FUNCTION (CRTYPE GENERAL)((CR GENERAL)))
```

CRTYPE takes as an argument a CRIL expression, CR, and deduces the natural type of the value of the expression. This function is used by many parts of the Type Resolver to make initial guesses. For example, the natural type of

```
(+ (DATUM REAL (NUMBER 5.3)) (FIELD INTEGER ...))
```

is real.

3.1.2 PRECISION

The PRECISION function is provided with the following declaration:

```
(FUNCTION (PRECISION INTEGER) ((T GENERAL)))
```

PRECISION takes as an argument any legal type, T, and deduces the precision (number of bits) of T, whether it is stated implicitly or explicitly.

3.1.3 FORMAT

The FORMAT function is provided with the following declaration:

```
(FUNCTION (FORMAT GENERAL) ((T GENERAL)))
```

FORMAT takes as an argument any legal type, T, and deduces the format of T, whether it is stated implicitly or explicitly.

3.1.4 TYPE

The function TYPE with an argument T is provided:

```
(FUNCTION (TYPE GENERAL) ((T GENERAL)) (CONS (FORMAT T)
   (PRECISION T)))
```

3.1.5 INTYPE

The INTYPE function is provided with the following declaration:

```
(FUNCTION (INTYPE BOOLEAN) ((T1 GENERAL)) (T2 GENERAL))
```

The arguments T1 and T2 are any legitimate types. INTYPE has value true if T1 ϵ T2, and false otherwise.

3.1.6 TRMST

The TRMST function is provided with the following declaration:

```
(FUNCTION (TRMST GENERAL) ((EX GENERAL)))
```

TRMST has as an argument a mixed CRIL-TRIL statement. If the statement is a COND, the TRMST of the embedded statement is returned. If the statement is a COMPOUND, etc., the TRMST of the last statement embedded in the COMPOUND, etc., is returned.

3.1.7 GNFUNC

The function GNFUNC is provided with the following declaration:

```
(FUNCTION (GNFUNC GENERAL) ((NAM GENERAL) (DEC GENERAL)
   (ARG GENERAL)))
```

GNFUNC generates an appropriate TRIL function call to the function NAM with the declaration DEC and list of TRIL arguments ARG.

3.2 TYPE RESOLUTION OF ARITHMETIC

3.2.1 Minus

The resolution of minus consists of the resolution of the argument with DTYPE* and PDTYPE, the same as for the minus itself. The form name is then changed to iminus if the embedded expression is fixed point; to rminus if the embedded expression is floating point; and to gminus if the expression is general format.

*Refer to Table 3 (page 16 of this document) for the use of public variables.

3.2.2 Plus and Times

(In the following discussion about addition, all remarks and algorithms apply equally to multiplication.) If either an argument of plus has natural type real or DTYPE is real, the arguments are resolved with DTYPE real and no PDTYPE. If there is a PDTYPE, it is used, otherwise PDTYPE is made the old DTYPE and DTYPE is not used for argument resolution. Arguments of general type should be resolved first. If any original general type argument resolves as real, resolution should continue for the rest of the arguments with DTYPE real and no PDTYPE. This introduces an order-dependency on the quality of produced code, but to do a better job could cause resolution time to rise exponentially with the depth of nesting of forms. All data should be collected and summed before the resolution commences. The resolved arguments are then parsed into fixed-point, floating-point, and general forms. These forms and necessary conversions are merged to produce the final result for the resolution. The values of DTYPE and PDTYPE are investigated for information concerning type optimization. The algorithms for this are not complicated but are extremely long, so will not be included here.

3.3 TYPE RESOLUTION OF DRIVE

If the natural type of the embedded expression is of the type specified by the embedded type, the drive is discarded and resolution continues for the argument as if the drive had never been in the CRIL. Otherwise, the argument is resolved with the embedded type as DTYPE and no PDTYPE. If the type of the expression after resolution $T_R \varepsilon T_D$, the drive type, then the resolution is finished; otherwise an appropriate conversion replaces the drive. Care must be exercised so that drives that are desired allow confluence point branches to be optimized appropriately.

3.4 CALLIT RESOLUTION

Callit does no top-driving while resolving its argument. However, in almost all cases, it will be a self-typing expression even in CRIL form.

3.5 PREDICATE RESOLUTION

3.5.1 Preturn Resolution

If the expression body of a preturn is a pterm, the preturn and pterm are thrown away and the statement body of the pterm shares the confluence points of an outer pterm. Otherwise, the argument is resolved. A pointer to the evalgo (which replaces the preturn) is then added to either TGO, FGO, BGO, or GGO.

3.5.2 Pterm Resolution

The statement body is resolved, with the variables TGO, FGO, BGO, NBGO, GGO, and NGGO rebound to NIL. Things referencing TGO and FGO are then patched to reference the appropriate labels. According to the number of references from each of BGO, NBGO, GGO and NGGO, the following action is taken: If the number of references from, say, BGO is greater than one, the references are changed to a genid. If the number of references is one, the test branching is patched into the referenced evalgo. If any of BGO, NBGO, GGO, or NGGO are referenced more than once, a pnet is created. It is assumed that whenever a pterm is encountered, a TGO and FGO are in existence.

TGO collects evaluations known to be true

FGO collects evaluations known to be false

GGO collects general values to be tested as predicates

NGGO collects general values to be tested as inverted predicates

BGO collects boolean values to be tested as predicates

NBGO collects boolean values to be tested as inverted predicates

3.6 TERMINAL RESOLUTION

3.6.1 Return Resolution

If the argument of a return is a terminal, the return and the terminal are discarded so that the embedded statement may share outer confluence points. Otherwise, the argument of return is resolved with the same DTYPE and PDTYPE as the embedding terminal. The type of the argument is now examined and the return is added to a confluence point in TRMLST.

3.6.2 Terminal Resolution

The embedded statement is resolved, thus accumulating various entities on TRMLST. TRMLST is then sorted by format and precision. PTYPE and PDTYPE are examined; based on their values and the types in TRMLST, a decision is made as to what type the terminal will produce. The net is then made. If there is only one element of TRMLST of a particular type, and that is not the type produced by the terminal, the conversion is introduced at the fork or return referenced. The other forks are then updated to specify the label in the net to use, and returns are changed to evalgos with the appropriate label.

PDTYPE is important in resolution of terminals. If it has a value, the terminal is known to be in arithmetic, and therefore may produce a numeric rather than general value in many instances.

3.7 RESOLUTION OF CONDITIONAL STATEMENTS

The consequences of a conditional statement and the next statement (if in a compound) are checked. If either is a go, preturn, or a return of a datum, special action is taken in the resolution of the predicate. A TGO and FGO, etc. may be picked up from an embedding pterm or a datum return label used from a terminal (in TRMLST). If the consequence is a return false, TGO is rebound to FGO, NBGO to BGO, etc. If the predicate is to fall through on transfer to TGO or FGO, then the variable is bound to NIL to indicate this. If the predicate behaves as a pterm, and a preturn or go is the consequence, no fork need be generated. If no special conditions exist, a fork is generated and the consequence is moved to the "fall-through" position. If a go statement is found, the label becomes the appropriate label of the fork.

References to TGO, BGO, etc. are shown by using the labels TGO., BGO., etc. If the generated fork is to be referenced by a net or pnet, it is added to the appropriate plex. In all cases of resolution of the predicate of the conditional, a TGO and FGO must be established.

3.8 RESOLUTION OF ASSIGNMENTS

The left member of an assignment is resolved. Its type and the values of DTYPE and PDTYPE are then examined to determine further action.

The following two tables list actions taken for various formats within assignment expressions. If, after resolution, the format of the right and left member differs, a terminal holding a binder is created. In the binder, a genid variable is introduced and preset to the unconverted value of the right member. The statement body of the binder is an assignment of the left member to an explicit conversion of the genid. All terminals created in the process must be resolved into nets.

Assignment <u>statements</u> always wrap drives around their right members before resolution.

Table 1. Assignment Expression DTYPE Format Actions

Format of Left Member	Format of DTYPE				
	INTEGER	REAL	GENERAL	FUNCTIONAL	BOOLEAN
INTEGER	DTint	Preal	( )	err	DTint RT
REAL	Dint	DTreal	( )	err	DTreal RT
GENERAL	Dgen	Dgen	DTgeneral	Dgeneral	Dgeneral
FUNCTIONAL	err	err	( )	DTfunctional	DTfunctional RT
BOOLEAN	DTint ST	DTreal ST	( )	DTfunctional ST	DTboolean

Table 2. Assignment Expression PDTYPE Format Actions

Format of Left Member	Format of PDTYPE				
	INTEGER	REAL	GENERAL	FUNCTIONAL	BOOLEAN
INTEGER	( )	Preal	( )	err	DTint RT
REAL	Preal	DTreal	( )	err	DTreal RT
GENERAL	( )	( )	DTgeneral	( )	( )
FUNCTIONAL	err	err	( )	DTfunctional	DTfunctional RT
BOOLEAN	Preal ST	Preal ST	( )	DTfunctional ST	DTboolean

DT_+	= resolve right member wrapped with a drive to type +
P_+	= resolve right member with PDTYPE = +
D_+	= resolve right member with DTYPE = +
()	= resolve right member with no top driving
RT	= make right member into assignment statement in terminal and return value true
ST	= make into terminal, setting left member true and returning the driven left member as value
err	= an impossible condition met

Table 3. Type Resolver Public General Variables

Name	Transmission	Usage
DTYPE	VALUE	Contains the top-driving type; may be NIL, meaning not known, or NOVALUE, meaning that no conversions are to be done.
PDTYPE	VALUE	Contains a passive top-driving type; the drive induced by this variable is merely a suggestion to help certain resolutions make choices
TRMLST	VALUE	TRMLST has the following format: ({type|datum} {return|fork}$_{*+1}$); the elements of TRMLST are pointers into the CRIL-TRIL forms that use a particular terminal
TGO	LOC	TGO has format ({preturn|fork|go}$_*$); the elements of TGO are pointers into the CRIL-TRIL forms that use a particular pterm
FGO	LOC	FGO has format ({preturn|fork|go}$_*$); the elements of FGO are pointers into the CRIL-TRIL forms that use a particular pterm
GGO	LOC	GGO has format ({preturn|fork|go}$_*$); the elements of GGO are pointers into the CRIL-TRIL forms that use a particular pterm
NGGO	LOC	NGGO has format ({preturn|fork|go}$_*$); the elements of NGGO are pointers into the CRIL-TRIL forms that use a particular pterm
BGO	LOC	BGO has format ({preturn|fork|go}$_*$); the elements of BGO are pointers into the CRIL-TRIL forms that use a particular pterm
NBGO	LOC	NBGO has format ({preturn|fork|go}$_*$); the elements of NBGO are pointers into the CRIL-TRIL forms that use a particular pterm