M. Richards December 29, 1967

This is part of an experimental PAL manual which was written to demonstrate the usefulness of L and Rvalues when describing PAL.

1.0 PAL Syntax

The syntactic notation used in this manual is basically BNF with the following extensions:

(1) The symbols E and D are used as shorthand for <expression> and <definition> .

(a) The metalinquistic brackets '<' and '>' may be nested and thus used to group together more than one constituant sequence (which may contain alternatives). An integer subscript may be attached to the metalinquistic bracket '>' and used to specify repetition; if it is an integer n, then the sequence within the brackets must be repeated at least n times; if the integer is followed by a minus sign, the sequence may be repeated at most n times or it may be absent.

1.1 Hardware Syntax

The hardware syntax is the syntax of an actual implementation of the language and is therefore necessarily dependent on the character set that is available. To simplify the description of any implementation of PAL a canonical syntax has been defined and this is given in section 1.2. The canonical representation of a PAL program consists of a sequence of the symbols from the following set.

1.1.1 PAL Canonical Symbols

The 41 list canonical symbols are given in the following list:

<name> <number> <string const> true false nil

J dummy ( ) λ . % $ ↑ x / + -

∼ = < > ∧ ∨ → aug , := goto val

resultis : ; where let def and within rec in

The symbols <name> <number> and <stringconst> are composite items which have associated sequences of characters.

1.1.2 The 1050/938 Hardware Representation

This hardware representation was designed for an IBM 1050 electric typewriter with a 938 golf ball. Although it is sometimes possible to juxtapose system words, numbers, and identifiers, it is wiser to separate such symbols with spaces.

(a) A system word is any sequence of two or more small letters not followed or preceded by a letter or a digit. These words are reserved by the system and cannot be used as names. The following table gives the correspondence between system words and canonical symbols.

System Word	Canonical Symbol	System Word	Canonical Symbol
true	<u>true</u>	or	$\vee$
false	<u>false</u>	aug	<u>aug</u>
nil	<u>nil</u>	goto	<u>goto</u>
jj	<u>J</u>	where	<u>where</u>
dummy	<u>dummy</u>	let	<u>let</u>
ll	λ	def	<u>def</u>
ls	$<$	and	<u>and</u>
gr	$>$	within	<u>within</u>
not	$\sim$	rec	<u>rec</u>
		in	<u>in</u>
		val	<u>val</u>
		res	<u>resultis</u>

No other system words may appear in a PAL program.

(b) A number is either a single digit or a sequence consisting of digits and at most one dot. A dot which is not adjacent to a digit stands for itself; it is used in the syntax of lambda expressions.

(c) A name is any sequence of letters and digits which is not a system word or a number; thus the following are names:

x   A3   2nd   Hd   consL   3hl

(d) User's comments may appear in a program between a double slash '//' and the end of the line. Example:

```
let  Y  f  =  let  x =  0  in  //  This defines a function  Y
                 x  := f x;      //  such that
                 x               //  f(Yf) = Y f
                                //  for some function f
```

(e) The hardware representations of all the other cononical symbols are given by the following table:

Hardware Symbol	Cononical Symbol	Hardware Symbol	Cononical Symbol
(	(	-	-
)	)	-*	→
**	↑	,	,
*	x	:=	:=
/	/	:	:
+	+	;	;
&	∧		

1.2 The Cononical Syntax of PAL

The syntax given in this section specifies all the legal constructions in BCPL without defining either the relative binding powers or the association rules of the expression or definition operators. These are given later in the manual in the descriptions of each construction. The terminal symbols of this syntax are the cononical symbols of PAL which are listed in section 1.1.1.

E ::= <name> | <stringconst> | <number> | true | false | dummy |
nil | J | (E) | E E | E() | E <diadic op> E |
~E | + E | -E | E → E, E | E $\langle$, E$\rangle_1$ | E %<name> E |
E := E | <name> : E | goto E | E ; E |
let D in E | E where D |
λ $\langle$bv$\rangle_1$ .. E | val E | resultis E

<diadic op> ::= ↑ | x | / | + | - | < | = | > | ∧ | v | aug

$$\begin{array}{lll}
D & ::= & \langle name\rangle \; < \; , \; \langle name\rangle \; >_0 \; = E \; \mid \\
 & & \langle name\rangle \langle bv\rangle_1 \; = E \mid \underline{rec} \; D \mid \\
 & & D \; \underline{and} \; D \mid D \; \underline{within} \; D \mid (D) \\
\langle bv\rangle & ::= & \langle name\rangle \mid () \mid (\; \langle name\rangle \; < \; , \; \langle name\rangle \; >_0 \;) \\
\langle program\rangle & ::= & < \underline{def} \; D >_0 < E >_{1-}
\end{array}$$

2.0 Data Items

2.1 Rvalues and Types

An RVALUE is a machine representation of an object in the domain of discourse of PAL. The actual kind of object represented is called the type of the Rvalue.

The complete set of PAL types is given below:

(a)	Integer	- This is a set of positive and negative integers of limited magnitude.
(b)	Real	- This is a set of real numbers of limited precision and magnitude.
(c)	Booâean	- This has only two values: truth and untruth.
(d)	String	- This is the set of all strings of zero or more characters.
(e)	Function	- This is the set of all functions which can be defined by a PAL function definition.
(f)	N-tuple	- An N-tuple is a set of Lvalues (see 2.2) each of which can be obtained by applying the N-tuple to the integer 1,2 --- N.
(g)	Label	- An Rvalue of type label represents a position in the program (see).
(h)	Program Closure	- An Rvalue of type program closure is obtained by applying the special function J to an N-tuple, a function or a program closure; for more details see section 3.6.

2.2 Lvalues and Extent

An LVALUE is a storage cell which contains an Rvalue. In PAL an Lvalue has the following properties:

(a) It contains an Rvalue of any type.

(b) The Rvalue contained can only be replaced by executing an assignment statement that updates the cell. The type of the new Rvalue of the cell need not be the same as the type of the old Rvalue.

(c) An Lvalue remains in existance as long as there is a reference to it.

The concept of Lvalues is useful when describing any programming language in which one can update variables by assignment. It is particularly necessary for PAL since most of the language features depend directly or indirectly on the sharing properties of the variables and these are best described in terms of Lvalues.

2.3 Mode of Evaluation

Any legal expression can be evaluated in one of two modes: Lmode or Rmode, to yield an Lvalue or an Rvalue respectively. The actual mode of evaluation of an expression occurring in a PAL program is determined by its syntactic context. The following table defines the mode of evaluation for every context in PAL. The symbols E, $\bar{E}$ and $\tilde{E}$ are used to define the mode as follows:

E denotes a right-hand context,

$\bar{E}$ denotes a left-hand context, and

$\tilde{E}$ denotes a context for which the mode of evaluation is the same as that of the expression of which the context is a part.

Mode of Evaluation Table

$E\ \bar{E}$	$E\ ()$	$\$\ E$
$E \uparrow E$	$E * E$	E / E
$E + E$	$E - E$	$+ E$
$- E$	$E < E$	$E = E$
$E > E$	$\sim E$	$E \wedge E$
$E \vee E$	$E \rightarrow \tilde{E}, \tilde{E}$	E <u>aug</u> $\bar{E}$
$\bar{E} <, \bar{E} >_1$	<u>goto</u> E	$E\ ;\ \tilde{E}$
$\langle name \rangle : \tilde{E}$	$\tilde{E}$ <u>where</u> D	<u>let</u> D <u>in</u> $\tilde{E}$
$\lambda \langle bv \rangle_1 . \bar{E}$	<u>val</u> $\bar{E}$	<u>resultis</u> $\bar{E}$
$\bar{E}\ \%\langle name \rangle\ \bar{E}$	$(\tilde{E})$	
$\langle name \rangle <, \langle name \rangle >_0 = \bar{E}$		
$\langle name \rangle \langle bv \rangle_1 = \bar{E}$		

2.3.1 The Functions δ and $\mathscr{C}$

These two functions are of supreme importance in PAL but are not directly accessible to the programmer.

δ takes an Rvalue as argument and yields an Lvalue containing the Rvalue as result, the Lvalue created during the application of δ is disjointed from all other Lvalues previously in existence, and it is an important feature of PAL that Lvalues can only be created by applying this function.

$\mathscr{C}$ is the inverse function; it takes an Lvalue as argument and yields the corresponding Rvalue as result.

The following set of expressions are called Rtype expressions and always yield Rvalues when evaluated.

$\langle$string const$\rangle$ $\langle$number$\rangle$ <u>true</u> <u>false</u> <u>nil</u>

<u>dummy</u> <u>J</u> $\$E$ $\sim E$ $+E$ $-E$

$E <, E >_1$ $E := E$ $\lambda < bv >_1 .E$ $E \times E$ E/E $E \uparrow E$

$E + E$ $E - E$ $E = E$ $E < E$ $E > E$ $E \wedge E$ $E \vee E$ E <u>aug</u> E

If such an expression occurs in a left-hand context, then the PAL system automatically invokes the transfer function δ.

The function $\mathcal{C}$ is automatically invoked if one of the following expressions occur in a right-hand context:

E E	E()	name
E % <name>	E	val E

It should be noted that the programmer loses no power nor convenience by not having direct access to $\mathcal{S}$ or $\mathcal{C}$ since he can always produce the right context for their automatic insertion by using the no-share operator $ (see section 3.8) and indeed he could define functions which are equivalent to $\mathcal{S}$ and $\mathcal{C}$ as follows:

```
let S   x  =  $ x
and C   y  =  $ y
```

3.0 Primary Expressions

The primary expressions are those whose syntactic form is:

E ::= <name> | <string const> | <number> | true | false | nil | dummy | J | (E) | $ E | E E | E ()

3.1 Names

Syntax:	The syntax of names is given in 1.1.2(c).
Semantics:	A name can always be evaluated to yield the Lvalue which was associated with the name at its declaration (see section). If the name occurs in a right-hand context, then C is applied automatically to yield the corresponding Rvalue.

3.2 String Constants

Syntax: $' \langle \text{string alphabet character} \rangle_0 '$

A string alphabet character is any single hardware character except ' and *, or it can be any of the following character pairs which are used to represent single characters as follows:

**	represents	*
*'	"	'
*n	"	newline
*s	"	space
*t	"	tab
*b	"	back space

Semantics: A string constant is an explicit representation of an Rvalue of type string.

3.3 Numbers

Syntax:	The syntax of numbers is given in 1.1.2(b).
Semantics:	If the number does not contain a decimal point in its written form, then it is an explicit representation of an Rvalue of type integer; otherwise, it represents a real Rvalue.

3.4 True and False

Syntax: <u>true</u> or <u>false</u>

Semantics: They are explicit representations of the two boolean Rvalues.

3.5 Nil and Dummy

Syntax: <u>nil</u> or <u>dummy</u>

Semantics: <u>nil</u> is an explicit representation of the Rvalue of the 0-tuple.
<u>dummy</u> is an explicit representation of the Rvalue which is the result of an assignment; it is useful in the few situations where it is syntactically necessary to have an expression but where no effect is required.

3.6 The Special Variable <u>J</u>

Syntax: <u>J</u>

Semantics: The meaning of <u>J</u> is difficult to define exactly without a detailed study of an evaluating mechanism for PAL; in this manual the description of <u>J</u> will be slightly incomplete.

The result of evaluating <u>J</u> is a special function which may be applied to certain Rvalues to yield program closures. It is only meaningful if the argument of <u>J</u> is a function, a program closure or an n-tuple; however, an erroneous application of <u>J</u> will only be detected when the resulting program closure is applied.

Suppose <u>J</u> was evaluated and applied to an ordinary function f, and let us call the program closure obtained g. Then g is the function f with an abnormal return. The difference between f and g is best described by an example:

$3 + f\ 2 - y / x$ (i)

At some point during the evaluation of this expression f will be applied to 2 to yield a result, R say; evaluation will then continue as if the sub-expression f 2 had been replaced by R:

```
          3   +   R   -   y / x
```

However, if the expression (i) had been

```
          3   +   g 2 -  y / x
```

then the effect would have been different. As g is the function f with an abnormal return, the evaluation proceeds as before until the moment when the result R is obtained; execution then suddenly jumps to a totally different place, just as if some other expression, E say, had just yielded the result R. The expression E is called a J-context and depends on the dynamic context of J's evaluation. The J-context associated with an occurrence of J is the smallest textually enclosing expression which is of one of the following syntactic forms:

a λ-body,
a function body.
a val expression,
a let expression body,
a where expression body,
or the defining expression in a within definition.

We will complete this discussion of J by analyzing the following example:

```
Print (  let f x = 3 + x
         and j,x,g = 0, 0, 0
         in
         let F t = (( j := J;
                      t ))
         in
         x  :=  F 5;
         g  :=  j f;
         x = 5 → g 6 ,  x + 10 )
```

In this example f is a simple function and j, x and g are local variables. The function F has the important side effect of assigning to j the result of evaluating J. The assignment

```
          x := F 5
```

has two effects: J is evaluated and assigned to j, and x is set to 5. The next command then applies j to f to yield a program closure of f which is then assigned to g.

The next statement compares x with 5 and, since they are equal, applies g to 6. As g is a program closure of f, evaluation first proceeds by applying f to 6 yielding the result 9; however, at this point the abnormal return takes place and execution resumes as if the body of F (which is the J-context of J), namely,

```
( j  :=  J;
  t )
```

had just yielded the result 9; the particular activation of this body is the one that was invoked when F was originally applied to 5 and so evaluation now continues as if F 5 had produced 9 in the assignment:

```
x  :=  F 5
```

and therefore x is set to 9. The next statement

```
g  :=  j f
```

has exactly the same effect as before, but the result of the final expression in the sequence is different. Since x now has value 9, the value of this conditional expression is the second alternative, namely, x + 10, which is 19. Thus the effect of the whole program is to print the number 19.

3.7 Bracketted Expressions

Syntax: (E)

Semantics: Parentheses may enclose any expression without changing its value or effect; their sole purpose is to specify grouping.

3.8 Noshare Expressions

Syntax: $ E

Where E may be any primary expression.

Semantics: The operator $ can be thought of as a transfer function from any expression to an Rtype expression. Its main use is to produce the right context for the automatic insertion of δ

The following examples illustrate the two most common uses of $.

The declaration

<u>let</u> t = $x

declares t to have the same Rvalue as x but different Lvalues and so the assignment to t does not change x.

The use of $ in

x := y, $ t

causes δ to be automatically invoked when the second number of the 2-tuple is evaluated.

After the assignment x is a 2-tuple whose first element shares with y and whose second element has an Lvalue which is disjoint from all other Lvalues previously existing.

3.9 <u>Combinations</u>

Syntax: E1 E2 or E1()

Where E1 is a primary expression and E2 is one of the following forms:

(E) <name> <number> <stringconst>

<u>true</u> <u>false</u> <u>nil</u> or <u>J</u>

note that the combinations associate to the left. The form E1 () is syntactic sugar for E1 <u>nil</u>.

Semantics: The expression E1 is evaluated in Rmode to yield the Rvalue of the operator of the combination and E2 is evaluated in Lmode to yield the Lvalue of operand. The order of evaluation is not defined. The operator is then applied to the operand and the effect depends on the type of the operator as follows:

(a) If the operator is an n-tuple, then the Rvalue of the operand, n say, is obtained; a check is then made to insure that n is an integer greater than zero and less than or equal to the length of the n-tuple. If the check is satisfied, then the Lvalue of the k element of the