Losp for Predicate Calculus

William Bricken

June 1985

Contents

1 Extension to Predicate Calculus

To extend the representational utility of Losp to predicate calculus, mechanisms for expressing quantification and for handling functional and relational terms must be introduced.

1.1 Representation of Terms

The representation of both functions and relations is PROLOG-like:

function-label [argument1 ... argumentn]
relation-label [argument1 ... argumentn]

The *square bracket* is not a parens, it is a new token that contains a representational space with attributes such as arity, type and order of the contents. In Losp, attributes normally associated with tokens are associated instead with the space within which the tokens are embedded. These attributes are determined by the dominant function or relation for the space.

The innovations of variarity and argument sets apply to functions and predicates as well as to boundary logic operators. I will illustrate the Losp innovations in representation and computation with the theory of equations.

1.2 The Theory of Equality

The axioms that describe how we should interpret the equality sign are:

- **Reflexivity**, $x = x$:
  An thing is equal to itself.
- **Symmetry**, $(x = y) \rightarrow (y = x)$: If a thing is equal to something else, then that something else is equal to the thing.
- **Transitivity**, $((x = y) \wedge (y = z)) \rightarrow (x = z)$: Things that equal the same thing are equal.
- **Extensionality**, $(x = y) \rightarrow (F(x) = F(y))$: The same operation can be performed on both sides of an equality without effecting the equality.

These axioms are quite simple, and are familiar from grade school arithmetic and from high school algebra. As stated, however, they are computationally misleading and inefficient. What is missing is the purpose, the computational utility, of each axiom.

Reflexivity, for example, is more general than merely bringing to our attention the redundancy that an x is an x. Rather the reflexivity axiom establishes the connection of equality to the computation of a logical truth. It is the definition as logically true that two recognizably identical things are equal. Usually, the two things on either side of an equal sign are not represented by identical tokens, as in the example $2 + 3 = 5$, and are thus not symbolically identical. We regard the statement $2 + 3 = 5$ as TRUE because we know an operation that converts the left-hand-side to the right-hand-side. Reflexivity extends further than things that are printed the same, it includes things that can be converted to one another by following rules. The rules are usually furnished by a formalization of the domain theory that defines the "things". In the example, the addition rule comes as part of the package of rules for "number things". A traditional mathematical notation looses this process information, that things might not already be in their simplest form when compared for equality. A definition of the reflexivity of equality that is extended to include computational transformations might be:

$$simplified(X) = simplified(X) \text{ is TRUE}$$

where the variable X now refers to a class of things that may not look the same, but can be made to look the same by permitted transformation rules. I will use a capital letter variable to represent this notion of a reducable set of expressions that form an equivalence class.[1]

The Losp rules for the theory of equality are identical in function to the traditional rules, but they are in a notation that is more general. The generality of the notation removes the computational complexity implied in traditional sequences of deduction.

[1] Traditional logical connectives are not converted to boundary logic in this example in order to avoid confounding innovations in logic with the relevant innovations in equality.

An initial notational change that is common to all the axioms is the use of a boundary to contain the arguments of the equality token. $X = Y$ is written as `=[X Y]`. The labelled boundary `=[...]` has properties of reflexivity, symmetry, transitivity, and extensionality. Symmetry is subsumed under argument sets and transitivity is subsumed by variarity.

- Reflexivity, `=[ =[X X] TRUE]`: When two expressions reduce to the same expression, their equality is true. Here the equal sign is used twice, once to relate the two expressions that reduce to one, and once to declare the fact that if two expressions reduce to the same expression, then their equalness is logically true. This axiom now serves as an evaluation condition for simplification. When the two expressions are simplified, if they look the same, we can replace their equality with truth.

- Symmetry, `=[X Y]`: The arguments to the generalized equality sign are unordered and ungrouped (set arguments), therefore which is written first is irrelevant.

- Transitivity, `=[ (=[X Y]` $\wedge$ `=[Y Z]) =[X Y Z]]`: Arguments can form a set of any number. If two things are both equal to a third, the three are equal. This generalization permits the specification of a collection of equal things.[2] Note the permissability of logical structures in the arguments to the top level equality.

- Extensionality, `=[ =[X Y] =[F[X] F[Y]]]`: This rule remains unchanged except the implication is replaced by a bi-directional equality.

The motivation for generalizing the concept of equality is to make computation efficient. The Losp theory of equality does this by two important techniques: substitution of simpler expressions and using equality as a semantics for logical expressions.

1.2.1 Simplification by Substitution

Things that are equal can replace one another. Always choose to replace a more complex expression by a simpler expression. Simplicity is determined by the size of the expression, by the inability to apply simplification rules and definition expansion rules. Thus, the axioms might be considered to be ways to replace a complex expression of a problem by a simpler expression. The arrow indicates an appropriate replacement:

[2] Traditional transitivity is simplified by the notion of *subsumption*. In the definition, the collection of three equal things subsumes, or is more general than, the collections of two equal things, and thus the collections of two can be deleted during simplification of the top level equal.

- Simplification: `X` $\Longrightarrow$ `simplify(X)`
- Reflexivity: `=[X X]` $\Longrightarrow$ TRUE
- Symmetry: `=[X Y]`
- Transitivity: `=[X Y]` $\wedge$ `=[Y Z]` $\Longrightarrow$ `=[X Y Z]`
- Extensionality: `=[F[X] F[Y]]` $\Longrightarrow$ `=[X Y]`

1.2.2 Equality of logical expressions

The arguments of an equality can be complex logical expressions. This permits an easy integration of logical and equational approaches. At the top level, all expressions are equated either to a truth value or to another expression. That is, all expressions are expressions of equality. The variable `X` is actually the equality `=[X` TRUE`]`. Since `X` is variable, we do not know it is true, but we can adopt the convention that representing `X` at least permits the possibility that it is true. Mixing the theory of equality with the theory of logic permits powerful simplification theorems.

1.3 Clausal Form Algorithm

The algorithm for conversion of logical expressions to *clausal form* serves as an example of the utility of boundary notation. This conversion is required by resolution theorem proving systems, since the resolution rule requires a standardized form when deducing over arbitrary logical expressions. The sequence of transformations is from logical sentence to prenex matrix to Skolem normal form to conjunctive normal form to clausal form. When applying the Losp proof method, conversion to conjunctive normal form (and therefore to clausal form) is unnecessary since EXTRACT penetrates any depth of nested boundaries.

Clausal Form Conversion Algorithm

1. Transcribe the logical sentence into boundary notation and CLARIFY. Rename all quantification variables using unique labels.
2. To create the prenex matrix from the expression, erase all quantifiers from the expression, recording their depth of nesting. CLARIFY. Quantifiers with even recorded depths are universal; quantifiers with odd depths are existential. The depth of the outermost context is 0.
3. To create the Skolem normal form of the prenex matrix, replace existentially quantified variables with a unique function whose arguments are those quantifiers with even depths less than the depth of the Skolemized variable. Discard all quantifiers.

4. To convert the Skolemized prenex matrix to conjunctive normal form, If any term in the matrix is at a depth n, $n > 3$, distribute the contents at depth $n - 2$ into the context of that term.

5. Finally, each set of expressions at depth 2 is a *clause*.

To illustrate the process of conversion to clausal form, consider this example from Robinson [2]:

$$\forall x(Px \equiv \exists y(Rxy \wedge \forall z Rzy))$$

Transcribe. Numerals refer to variables standardized apart:

```
[1](((P1) ([2](R12) ([3]R32))) ([4](R14) ([5]R54) P1))
```

Erase the quantifiers to get the prenex matrix. Record their depths for the Skolemization step.

```
(((P1) ((R12)(R32))) ((R14)(R54) P1))
```

The table of substitutions for this problem follows:

Quantifier	*Depth*	*Prenex Type*
`[1]`	0	universal
`[2]`	3	existential
`[3]`	4	universal
`[4]`	2	universal
`[5]`	3	existential

Skolemization requires these substitutions:

Variable 2 becomes `g[1 4]`
Variable 5 becomes `h[1 4]`

The resulting expression (which does not simplify) completes the normal form conversion:

```
( ((P1) ((R1g[14]) (R3g[14]))) ((R14) (Rh[14]4) P1) )
```

Conjunctive normal form requires `(P1)` to be distributed (Skolem substitutions have been omitted for readability):

```
( ((P1) R12) ((P1) R32) ((R14)(R54) P1) )
```

The clauses and their equivalent traditional forms are nested within the AND template:

```
( (clause1) (clause2) (clause3) )
```

Universal	$\forall x.E[x] \Longrightarrow$ `[x] E[x]`
Existential	$\exists x.E[x] \Longrightarrow$ `([x] (E[x]))`
Irrelevant Quantification	$[x]a \Longrightarrow$ `a`

Table 1: The Map from Quantifiers to Losp

The Losp and conventional notations for each of the three clauses is:

Losp Form	*Resolution Form*
`(P1) R12`	$(\neg P1 \vee R12)$
`(P1) R32`	$(\neg P1 \vee R32)$
`(R14)(R54) P1`	$(\neg R14 \vee \neg R54 \vee P1)$

1.4 Quantification

Table 1 contains the map from quantifiers to Losp. The unlabelled square bracket identifies a quantification variable. The token $E[x]$ is shorthand for any expression containing x.

Quantification indicators, `[x]`, at even depths are universal; quantification indicators at odd depths are existential.

This representation obsoletes quantifier transformation rules such as the quantifier negation equivalencies. The square bracket containing quantification variables is treated as any other token except when it is irrelevant: a quantification environment may be erased from a context which does not refer to the contents of that environment. For example, consider the transformation rule:

$$\exists x(p \rightarrow Ax) \equiv (p \rightarrow \exists x Ax)$$

The Losp transcription is:

`([x] ((p) Ax))` $\Longleftrightarrow$ `(p) ([x] (Ax))`

This equivalence is a case of Transposition. These steps convert the left-hand-side to the right-hand-side:

	`([x] ((p) Ax) )`	
$\Longrightarrow$	`(( ([x] p) ([x] (Ax)) ))`	distribute `[x]`
$\Longrightarrow$	`([x] p) ([x] (Ax))`	clarify
$\Longrightarrow$	`( p) ([x] (Ax))`	irrelevant `[x]`

When a quantification environment is distributed across terms that do use its bound variables, variable renaming is necessary:

`[x] ((Ax)(Bx))` $\Longleftrightarrow$ `(([x] Ax) ([y] By))`

Extract	`A (A B)` $\Longrightarrow$ `A (B)`
Extract-and-Unify	`A1 (A2 B)` $\Longrightarrow$ `g[A] (g[B])`

Table 2: Combined Pervasion and Unification

Transitivity	`(((A) B)((B) C))` $\Longrightarrow$ `(A) C`
Unifying Transitivity	`(((A) B1)((B2) C))` $\Longrightarrow$ `(g[A]) g[C]`

Table 3: Combined Transitivity and Unification

1.5 Unification Proofs

To generalize the Losp proof method to predicate calculus, the only modification necessary is to combine Pervasion with *unification*. In Table 2, `g[s]` represents the most general unifier of `A1` and `A2`. In the case of EXTRACT-AND-UNIFY, the extraction of a term must be accompanied by the substitutions resulting from the unification of the extracting and the extracted term.

The selection of which terms to extract in complex expressions with multiple extraction possibilities is the source of the computational complexity associated with searching proof spaces for solutions. The Losp extraction process is *opportunistic* since the parens operator partitions the problem into dominant (shallower) and extractable (deeper) spaces.

EXTRACT-AND-UNIFY requires a generalization to be fully equivalent to resolution. In Table 3, `g[s]` represents the most general unifier of the terms `B1` and `B2`. Transitivity is very similar to resolution. To generate the right-hand-side, the clauses `(A)B` and `(B)C` in the CNF on the left-hand-side are combined in the same context, while the literal `B` and its negation `(B)` are erased. An alternative way to read this rule is Transitivity of Implication. From $a \rightarrow b$ and $b \rightarrow c$, we can conclude that $a \rightarrow c$. A generalization of Transitivity of Implication occurs when the `B` terms are unified.

References

[1] Spencer-Brown, G. *Laws of Form* Bantam New York 1969

[2] Robinson, J. A. *Logic: Form and Function* North-Holland New York 1979

[3] McCarthy, J. and Talcott, C. *LISP Programming and Proving* Stanford University Stanford, Calif. 1980

[4] Varela, Fransico J. *Principles of Biological Autonomy* North Holland New York 1979

[5] Kauffman, Louis H. and Varela, F. J. Form Dynamics *Journal of Social and Biological Sciences* 3 1980 171-206

[6] Kauffman, Louis H. Network Synthesis and Varela's Calculus *International Journal of General Systems* 4 1978 179-187

[7] Banaschewski, B. On G. Spencer Brown's Laws of Form *Notre Dame Journal of Formal Logic* 18 1977 507-509

[8] Klenk, Virginia *Undertstanding Symbolic Logic* Prentice-Hall New Jersey 1983

[9] Manna, Z. and Waldinger, R. *The Logical Basis for Computer Programming, Volume 1* Addison-Wesley Massachusetts 1985

[10] Lemmon, E. J. *Beginning Logic* Hackett Indianapolis, Indiana 1978

[11] Mellish, C. and S. Hardy *Integrating PROLOG in the POPLOG Environment* Campbell, J. A. Implementations of Prolog Ellis Horwood New York 1984

[12] Robinson, J. A. and E. E. Sibert *Logic Programming in LISP* Rome Air Development Center 1981 RADC-TR-80-379, Vol. I

[13] Genesereth, M. R. et al *MRS Manual* U.S. Department of Commerce 1981 AD-A123 256

[14] Schwartz, D. G. Isomorphisms of Spencer-Brown's Laws of Form and Varela's Calculus for Self-Reference *International Journal of General Systems* 6 1981 239-255

[15] Kohout, L. J. and V. Pinkava The Algebraic Structure of the Spencer-Brown and Varela Calculi *International Journal of General Systems* 6 1980 155-171

[16] Bundy, Alan *The Computer Modelling of Mathematical Reasoning* Academic Press London 1983

[17] Clocksin, W.F. and C. S. Mellish *Programming in Prolog* Springer-Verlag New York 1981

[18] Norris, F. R. *Discrete Structures* Prentice-Hall New Jersey 1985