AN OVERVIEW OF THE LOSP DEDUCTIVE ENGINE

William Bricken
February 1987

CONTENTS

OVERVIEW

The goal of logic programming is to develop both a logical specification language and a deductive engine that computes theorems from axioms written in the specification language. Ideally, the Predicate Calculus would form the specification language, and known inference techniques would form the deductive engine.

Losp is an implementation of a relatively unknown deductive technique, Spencer-Brown's LAWS OF FORM. The Losp Deductive Engine accepts any set of well-formed sentences in predicate logic. If one sentence is specified as a goal, Losp deduces whether or not that goal is consistent with the other sentences.

Traditionally, the validity of an expression has been determined either by an exhaustive analysis (the truth-table method) or by the successive application of inferential rules that transform one set of valid sentences into another (the proof method). Recently, the resolution proof method has dominated automated theorem proving, presumably because it is computationally simple and logically complete.

The theoretical basis of Losp has been shown to morphic with Boolean algebra, and thus with Propositional Logic. In a computational environment where quantification is constrained to finite sets, the Losp proof procedure subsumes other existing procedures. In addition, by a change in representation, Losp achieves an elegant simplification both of the set of inference rules and of the form of the logical input sentences. Losp effectively removes all redundancy from propositional sentences. Losp rewrite and inference rules erase irrelevant structures from the representation, resulting in rapid convergence on a solution without the

overhead of memory. Perhaps most importantly, Losp provides deep insight into the structure of logical deduction since it achieves full generality within a representation that is unencumbered by redundancy, exceptions and special representational forms.

The Losp Deductive Engine is currently simulated in Pure LISP. The rest of the paper describes this engine, and within the context of the implementation, describes the theory of Losp. Since all Pure LISP function definitions can be transcribed into Losp, our future work will be to build a deductive engine expressed entirely in Losp.

REPRESENTATION

The Redefinition of Nothing

The core of Losp's representational power resides in the redefinition of the concept of nothing, as expressed in LISP by the token NIL. LISP overlays another meaning upon the concept of nothing: that of the EMPTY LIST.

Losp-DEFINITION: The empty list is not NIL.

The intention is to define NOTHING literally as non-representation. The empty list, being something (a set of parentheses containing nothing), is not nothing. This distinction allows two kinds of failure. We can still interpret NIL as falsity. In addition, we can interpret the empty list as not false, that is, as true. The usual list processing end-test, (NULL list), is expressed directly in Losp as (list). If the variable named list has no members, then list = NIL, and its emptiness is expressed by the token (NIL) which has the truth-value of true.

Non-representation is carried a step further in Losp: the token NIL, which until now we have used to label the concept of non-representable nothingness, is formally not represented. Thus:

NIL ==> which is interpreted as the truth-value FALSE

( ) ==> emptiness, which is interpreted as TRUE

Here the blank space is used to convey the non-representational intent of NIL. In other words, the token NIL never occurs in Losp.

The definition of NEGATION arises naturally out of the fundamental distinction between nothing and nothingness.

Losp-DEFINITION: The representation of the act of listing, called PARENS, is the representation of NEGATION

Making a list of nothing negates the nothing. [See the FOOTNOTE on PHI.]

The point is that NIL has two representations: one when standing alone and one when indicating positive nonexistence of other elements. Ambiguity is avoided, since the standing alone representation is in fact a non-representation.

We retain the representation of emptiness, ( ), as the positive indication of the empty list. This observation of state is meta-theoretic since it comments upon our ability to observe the unobservable. The meta-level is directly constructive, since we must create the PARENS tokens in order to represent to ourselves that there is nothing inside. The more fundamental meaning of NIL is .

Put succinctly: there are TWO VOIDS, the absolute, which cannot be mentioned directly, and the relative, which can.

In general, (a) represents a list with existent contents. There is one case of special interest, a = ( ):

(( )) = NIL =

Using the definition of negation for the outer parens, the negation of the positive representation of the empty list is a loss of representation. The list of the empty list conveys nothing.

We can now represent both nothing, (( )), and nothingness, ( ).

Collections

By means of the previous equation, we can approach the unmentionable, representing it indirectly. Further, we can use this property to disambiguate listing as negation from listing as grouping.

Losp-DEFINITION: A collection, or set, is represented by enclosing the members in a double parens.

For the case of existence: ((a)) is the collection of a, (a) is not-a. For sets with many members, we write: ((a b c d)).

For the case of NIL:

((NIL))	=	(( ))	interpreted as FALSE
(NIL)	=	( )	interpreted as TRUE

Intolerance of Redundancy

The second fundamental law in Losp is the non-representation of duplicate objects:

( ) ( ) = ( )

We intuitively recognize that non-representational forms do not possess the concept of duplication or of number. This unwritten rule determines, for example, our willingness to treat all empty pages as indistinguishable. In the case of

(( )) =

it does not make sense to inquire "How many blanknesses are on the right hand side of the equation?" Similarly, it does not matter how many positive indications of NIL we observe, all are indistinguishable.

The concept of REPRESENTATIONAL SPACE helps to constrain the locale of non-redundancy:

Losp-DEFINITION: PARENS delineates a REPRESENTATIONAL SPACE.

In common writing, we have many implicit (and obvious) rules about the delineation of common representational spaces. The words that constitute a sentence are delineated between punctuation marks, and not by the physical placement of lines on the sheet of paper. Set braces, { }, and list parentheses, ( ), delineate representational contexts; they issue a meta-instruction for the reader to form partitions between objects within and objects without. In Losp, as in set theory, duplicates are not permitted to share the same representational space.

Variables

I have identified two states, one represented and one non-represented. By introducing variable tokens to stand in place of either state, we can generalize the two fundamental axioms of Losp:

((a)) = a COLLECT

a a = a REPEAT

where the token a may indicate either the state of existence or that of non-existence. One other axiom specifies the relation between duplicate variables in different representational spaces:

(a) a = ( ) a EXTRACT

This can be easily demonstrated by case analysis: the equation holds in the case of erasing all the variable tokens, and it holds in the case of each token being a positive indication.

Case I, *a* is the absolute void:

( ) = ( )

Case II, *a* is the relative void:

(( )) ( ) = ( )

Functions

In LISP, functions are represented as lists. The first token of the list must be either a recognizable (defined) function name or the token QUOTE, which means that the following list is a data structure.

Losp uses the same representation for function calls as does LISP: if the first token in a list is a recognized function name, that function is invoked. However, in Losp, the QUOTE is assumed by default:

> Losp-DEFINITION: If the first token in a list is not a recognized function name, functional invocation is not indicated.

Function invocation is defined identically as in LISP, by the binding of the instance arguments following a function name to dummy arguments in a lambda structure that is the abstract definition of the function. Function invocation places arguments into the context of the function definition. In contrast to LISP, the arguments do not need to be literal values, they can be expressions containing variables.

Relations

Losp supports relations with the same representation as functions: if the first token in a list is a recognized relational name, that relation is invoked. The invocation of a relation is different to that of a function. A relational token asks that the following arguments be bound to values; the values are found by pattern matching the structure of the relational invocation against a database.

Quantifiers are defined in the context of a relational invocation. The token ALL instructs the interpreter to consider all matches of the invocation pattern that can be found in the database. The token EXIST instructs the interpreter to verify that one pattern match can be found in the database.

Theorems

Since Losp does not require variables to be bound to values, it can be used for abstract theorem proving without reference to a database. In this case, relations are not invoked; rather they are passed as literals. Quantification is also carried structurally rather than being instantiated literally.

POWER

By extending the concept of a variable to represent any arbitrary Losp expression, that is, any well-formed configuration of parens, we gain the full representational power of Propositional Logic. By permitting both functional and relational invocation, we form a language that has the expressive power of First Order Logic.

Losp is an equational logic, not an implicational logic. The equational proof technique of substitution of equals is implicit to this logic.

In order to bridge the gap between the theory of the Void, and more conventional logical knowledge, I will next discuss the Losp parser which coverts both logical notation and LISP control structure into Losp notation. The three axioms mentioned above are then sufficient both to derive logical theorems and to execute program code.

FOOTNOTE

The set theoretic parallel to the empty list is the EMPTY SET, PHI, which is the negation of the concept of no members.

Losp-DEFINITION: The only set that contains PHI is PHI.

That is, PHI is not a member of any set that has existent members. This turn of definition allows us to use PHI as a positive test for set emptiness, since it comes into existence representationally only when there are no other members remaining in a set.

By the self-referential quality embodied in the definition of PHI, we know that PHI contains a representation of itself. Since the only existent representation is ( ), this definition is expressed as

PHI = (( ))

But PHI is also non-represented nothing; therefore,

(( )) =