A Hybrid Predicate Calculus*

William W. Wadge

Abstract

We present the Hybrid Predicate Calculus (HPC), a hybrid logical system which incorporates a fairly conventional first-order predicate calculus, but which also include elements of modal logic and relational algebra. A special effort has been made to produce a (syntactically and semantically) well-integrated whole, rather than just a disjoint union.

Our calculus, from a formal point of view, is equivalent (in terms of expressiveness) to conventional first-order systems. It exhibits, however, some promising pragmatic advantages. In particular, many statements of so-called “everyday” language can be formulated more directly and concisely, often without resort to any bound variables.

1 Introduction

This system presented here is the result of years of gradual refinement.

The main motivation originally was to simplify the teaching of introductory formal logical systems. Traditionally, logic courses present two distinct but related systems, namely propositional and predicate logic. However, these are not enough for computing-related students. They also need to learn $\lambda$-calculus, Horn logic (Prolog) and relational algebra (for relational databases).

Furthermore, modal logic is useful for AI applications.

My interest in hybrid systems began with a simple hybrid logic I developed in 1975 to simplify proofs of relational algebra equations [2]. This system extended an equational/algebraic system for binary relations by allowing formulas in which individual variables sandwich relation expressions (e.g., $aR; Sc$). This allowed natural deduction inference rules such as $aRc, cSb \vdash aR; Sb$. Of course, the system presented here is far more ambitious.

There are two key ideas that permit a harmonious—in fact synergistic—union of the separate systems. One is to use prefix notation throughout (except for Boolean operations). The other is to conflate analogous features. We identify the Boolean operations on propositions and the corresponding set operations on relations. We also collapse existential quantification, relational projection, and modal possibility.

One unexpected advantage arises from the almost unavoidable fact that hybrid formulas—formulas using features from different systems—are allowed. This enables people to express themselves in a way they cannot in any one of the original systems. In particular, the logic allows expressions corresponding closely to the quantifier phrases of natural languages.

2 Propositional Logic

Propositional logic is the basis for our whole system. Mine is thoroughly conventional. Propositional variables are upper-case letters (we will not worry here about what to do if we need more than 26). I use the usual symbols for the Boolean operations, using fully parenthesized infix notation.

*Presented to the Hybrid Logics Workshop, Birmingham, August 2000.
sequences of formulas. This allows, in particular, a sequent style implication, which makes up for all those parentheses; for example,
\[(A (A \to B) \neg B \to \neg A)\].

3 Modal Logic

The modal logic is obtained by adding ♦ and □ used as prefix operators, in the usual way.

The informal semantics is as follows: there is a domain of discourse (people, days, places), the variables A, B, C, etc. are properties, and ♦ and □ are interpreted as unrestricted generalization—always, everywhere, everything.

Even in this simple system, students meet what for many is the biggest obstacle in logic—relative generalization. For example, they have no trouble expressing “someone is happy” as ♦H or “it rains every day” as □R. Problems arise with “some student is happy” or “it rains every winter day”. They almost inevitably confuse the two and write ♦(S → H) and □(W ∧ R). I experimented with a number of generalizations. Eventually I settled on the idea of splitting ♦ and □ into square and angle parentheses which in the case of relativized quantification enclose the relativizing formulas. Thus “some student is happy” can be expressed as ⟨S⟩H and “it rains every winter day” as [W]R.

More generally, the modal parentheses can enclose an arbitrary sequence of relativizing formulas, treated as if they were conjoined. Thus “every Biology or Chemistry student who is not lazy is happy” becomes [(B ∨ C)¬L]H. This kind of ‘syntactic sugar’ has no foundational significance but dramatically improves the pragmatic usability of the notation.

4 Hybrid Modal Logic

Extending to a hybrid modal logic is simple. We add a set of individual/world variables—lowercase letters—and allow an unmodalized formula to be preceded by an individual variable. For example, if G is the property of being Greek and M that of being mortal, and s is Socrates and a is Apollo, then sG asserts that Socrates is Greek, and a(G ∧ ¬M) asserts that Apollo is Greek but not mortal. The syllogism “Socrates is Greek; all Greeks are Mortal; therefore Socrates is mortal” is captured by the implication (sG [G]M → sM).

One intriguing point is that individual variables—at least syntactically—are treated very much like ♦ and □. It’s possible (following Montague) to treat them the same syntactically, as both being instances of the notion of quantifier phrase. Thus s is “Socrates”, ♦ is “someone”, and [G] is “every Greek”.

Relativized modalization apart, this logic is essentially the system LPC2 described in [1] (this work was the main inspiration for the development of HPC).

5 Relational Algebra

The next step is to generalize from a logic of properties to one of relations. This is the hardest, both syntactically and semantically. If we restrict ourselves to binary relations, we can use the scheme in Reference [2]. But from a pragmatic point of view this would be far too restrictive. For example, any practical treatment of relational databases must allow relations with higher arities (described as tables with many columns).

The obvious approach is to introduce some syntactic/semantic classification of relations according to their arities. In practice, however, this proves to be very complicated. For example, can we form Boolean combinations of relations of different arities? Is there only one empty relation, or one for each arity?

The solution I chose was to make no a priori restriction on the ways in which relations can be combined. We allow any Boolean combination; and any expression can have an individual variable or a modality prepended.
The key semantic idea is to treat all relations as being (formally) of infinite arity. In other words, every relation is a set of \( \omega \)-sequences of individuals. Then a traditional unary relation (a set) is embodied as the collection of all sequences whose first element is in the set. A traditional binary relation is embodied as the set of all sequences whose first two elements are in the given traditional relation. There is thus only one empty relation, and a simple set is embodied by exactly the same object that embodies the binary relation consisting of all pairs whose first element is in the given set.

The Boolean operators are interpreted as the corresponding set operators. Thus, for example, \( (A \lor B) \) denotes the union of the relations (sets of sequences) denoted by \( A \) and \( B \).

In this semantics, \( \Diamond \) becomes the projection operator on the first component: a sequence is in the relation denoted by \( \Diamond G \) iff the result of removing its first element is in the relation denoted by \( G \). The operator \( \Box \) is the dual, i.e., a sequence is in the relation denoted by \( \Box G \) iff every result of prepending any individual to that sequence is in the relation denoted by \( G \).

Individual variables correspond to a kind of selection operator. A sequence is in the relation denoted by \( sG \) iff the result of prepending to that sequence the individual denoted by \( s \) yields a sequence in the relation denoted by \( G \).

6 Relations as Properties

The pragmatic consequences of the definitions given above are not immediately obvious. We will give formal definitions in a later section. However, it is more enlightening to describe the operations using informal natural language. We extend ordinary English with special symbols \( \odot, \odot_1, \odot_2, \) etc., which stand for the components of the infinite sequences; in other words, the arguments of the relations. So instead of saying that \( G \) denotes the property of being Greek, we say that \( G \) means "\( \odot_0 \) is Greek". And instead of saying that \( L \) denotes the property of liking, we say that \( L \) means "\( \odot_1 \) likes \( \odot_0 \)" (note the descending order). It is not hard to compute the corresponding informal meanings of the following expressions:

\[
\begin{align*}
\odot_0 G & \quad \text{Socrates is Greek} \\
\odot_1 L & \quad \text{Socrates likes Apollo} \\
\odot_0 L & \quad \text{\( \odot_0 \) likes Apollo} \\
(L \land G) & \quad \text{\( \odot_0 \) likes \( \odot_0 \), who is Greek} \\
\Diamond L & \quad \text{\( \odot_0 \) likes someone} \\
\Diamond \Diamond L & \quad \text{someone likes someone} \\
s\Diamond L & \quad \text{Socrates likes someone} \\
\odot_0 L & \quad \text{someone likes Apollo}
\end{align*}
\]

Notice that individual variables and modal operators are still treated analogously, as quantifier phrases, much as in English itself (in the sense that "John", "some student", and "everyone who knows Jill" are treated, syntactically, as belonging to the same category).

There are simple incremental rules for computing the informal characterizations. These rules assume that the informal expression is in a standard form, in which the individual symbols appear at most once each, in decreasing numerical order (e.g., \( \odot_2 \) appears before \( \odot_1 \), which appears before \( \odot_0 \)).

The expression for (say) \( (A \land B) \) is the result of and-ing together the expressions for \( A \) and \( B \), then rewording to put it in normal form.

The expression for \( \Diamond A \) is the result of taking the expression for \( A \), replacing \( \odot_0 \) by "someone", and decreasing by one each of any remaining circled numbers. Similarly for \( \Box A \), with "everyone" instead of "someone". (We are assuming the universe of discourse is a set of people).

The expression for (say) \( sA \) is the result of replacing \( \odot_0 \) by "Socrates" and decrementing any remaining circled numbers.

The rules can be extended to relativized modalities: for example, the expression for \( \langle G \rangle A \) is the result of replacing \( \odot_0 \) by "some Greek" and decrementing.
7 Relational Operations

It should be apparent that the system described so far falls far short of predicate logic in expressiveness. For example, we cannot even formulate the assertion that “Apollo likes all those who like themselves”. We need to add combinatorial operators to make up for the lack of bound variables. The following three are (almost) enough: the converse operator ⨿, the shift operator ∗, and the diagonal operator /.

Informally, the relation denoted by ⨿G consists of all sequences formed by taking a sequence in the relation denoted by G and swapping the first two components. That denoted by ∗G consists of all sequences formed by prepending any individual to an element of the relation denoted by G. Finally, that denoted by /G consists of all sequences formed by removing the first element from a sequence in the relation denoted by G in which the first two components are equal.

The rules for transforming natural language characterizations are simple. To get the description of ⨿A from A, we swap ⨿1 and ⨿0 (and reword, since ⨿1 and ⨿0 are out of order). To get that of ∗A, we increment by one every circled numeral. And to get that of /A, we replace ⨿1 by ⨿0, decrement the other circled numbers, and reword (since ⨿0 may now appear twice).

It should be easy enough to see that /L means “◯ likes him/herself” and therefore that a[//L]L means “Apollo likes all those who like themselves”. Unfortunately, the three operators are not enough, basically because they work only on the first one or two arguments. We also need, for each i > 0, three analogous operators ⨿i, ∗i, and /i that operate (roughly speaking) starting with argument i. For example, /3 swaps ⨿3 and ⨿4.

8 Predicate Logic

The final step, to predicate logic, is relatively simple. Almost everything is already here: relations, individual variables, Boolean combinations, even universal and existential generalization (of a kind). The only difference between □ and ∀ is that the latter binds a variable.

We add abstraction (binding) as a separate feature. The new formation rule is: we can prepend any formula with an individual variable followed by the symbol “:”. We can think of x:A as denoting the ‘set’ of all x with the property A. For example, p : (pG ∧ apL) is the set of all Greeks that Apollo likes. Nested abstraction gives us relations of higher arity. Thus q : p : (qpL ∧ pqL) is the relation of liking each other. The informal rendering of x : A is obtained from that of A by incrementing the circled numerals and replacing all occurrences of x by ⨿0 (and rewording).

The traditional quantifiers correspond exactly to the result of combining the ‘modal’ operators and abstractions. Thus the conventional formula

\[ \forall x\left(G(x) \rightarrow \exists y\left(L(y, a) \land \neg L(x, y)\right)\right)\]

can be transcribed literally as

\[ \square x : (xG \rightarrow \Diamond y : (yaL \land \neg xyL))\]

Of course, the idiomatic way to express in HPC the idea that “every Greek likes someone who likes Apollo” is [G](aL)L.

The claim that the relational operators are combinatorially complete is easily formulated.

**Theorem 1.** Every HPC formula is logically equivalent to one in which “:” does not appear.

The proof uses the following lemma, which states that any variable can be extracted from the body of a formula and brought to its head.

**Lemma 1.** Given any “:”-free HPC formula A and any individual variable x, there is a “:”-free HPC formula A’ in which x does not occur such that A is logically equivalent to xA'.
The proof proceeds by induction on the structure of $A$; essentially, we show that the collection of formulas of the form $xA'$ is closed under the formation rules.

Given the lemma, we take a formula of the form $x : A$, rewrite it as $x : xA'$, and the latter is equivalent to $A'$. If a formula contains buried and/or nested “$:$”s, we apply the lemma repeatedly from the inside out.

9 Formal Semantics

We now summarize the formal semantics. An HPC interpretation $\mathcal{I}$ specifies a nonempty domain of discourse (set of individuals). To each individual variable $x$, $\mathcal{I}$ assigns an element $\mathcal{I}(x)$ of the domain of discourse. And to each relation variable $V$, $\mathcal{I}$ assigns an $\omega$-ary relation $\mathcal{I}(V)$ over the domain of discourse; in other words, $\mathcal{I}(V)$ is a set of $\omega$-sequences of elements of the domain of discourse.

An interpretation by itself is not, in general, enough to determine the truth or falsity of an HPC formula; we need to specify as well the values of $\emptyset$, $\{\}$, $\{2\}$, $\ldots$. Our satisfaction relation $\models$ therefore brings together an interpretation $\mathcal{I}$, a formula $A$, and an $\omega$-sequence $d_0, d_1, d_2, \ldots$ of elements of the domain of discourse. In symbols,

$$\mathcal{I}, d_0, d_1, d_2, \ldots \models A$$

means that $A$ is satisfied by $\mathcal{I}$ and $d_0, d_1, d_2, \ldots$.

The Boolean operations are interpreted in the standard way, e.g.,

$$\mathcal{I}, d_0, d_1, d_2, \ldots \models (A \land B)$$

iff

$$\mathcal{I}, d_0, d_1, d_2, \ldots \models A \quad \text{and} \quad \mathcal{I}, d_0, d_1, d_2, \ldots \models B$$

The semantics of the remaining constructs is presented, as usual, as a collection of equivalences, one for each construct. In each case the satisfaction relation on the left holds iff that on the right holds (we have omitted the definitions of $\emptyset$, $\{\}$, $\{2\}$, $\ldots$, $/2$, $/3$, \ldots).

10 Acknowledgement

This was made possible by generous support from the Canadian Natural Sciences and Engineering Research Council (NSERC).

References