A Design Proposal For An Event Query Language

Nicholas T. Karonis

July 19, 1995

1 Introduction

The current Event Query Language (EQL) was designed to allow experimenters to formulate queries in order to extract smaller and smaller sets of events. Its syntax is based upon another query language, PAW. PAW, and hence EQL, are both seemingly based upon the first-order logic. Unfortunately, because EQL embodies only a part of logic, it is currently possible to validly interpret a single EQL query in semantically different ways. In other words, the language is currently ambiguous.

Our mission was to re-evaluate the syntax and, more importantly, the semantics of EQL. In doing so, we believe that we have a new design for the language that minimizes changes to its syntax while disambiguating it and extending its expressive power to that of the first-order logic.

Although this document pertains to EQL, an existing language, we purposely did not address issues associated with implementation. We do make requirements of an implementation of this proposal, we simply do not address how those requirements should be met. Our intent here is to lay a sound theoretical foundation upon which an implementation can be built.

We start with a brief overview of the first-order logic and introduce one of its forms, the prenex normal form. We then present our proposal for the language in the context of prenex normal form formulas in the first-order logic. That is, we illustrate the one-to-one correspondence between queries in our proposed language and prenex normal form formulas in the first-order logic. Finally, we address the hidden semantics of an operator that already exists in EQL, the infix binary operator ".", by interpreting it as a membership operation and exploring some of the ramifications of explicitly interpreting the operator in this way.

2 The First-Order Logic

In this section we give a *brief* overview of the first-order logic (FOL), not a comprehensive treatment of the subject. We motivate it by comparing it to the less powerful propositional logic, and then go on to present and describe the symbols used to construct statements (formulas) in the logic. Finally, we describe one of the forms formulas in the FOL can take, prenex normal form, and the necessary assumptions to convert any FOL formula into its prenex normal form equivalent.

2.1 The Propositional Logic

In the propositional logic there are simple statements that are either true or false, e.g., "Today is Tuesday.," "Socrates is a man.," etc. Although it is a powerful logic, it is not powerful enough to express simple arguments like "Every man is mortal. Socrates is a man. Therefore, Socrates is mortal." Part of the problem lies in the fact that the propositional logic has no mechanism to connect Socrates in the second sentence to one of the men referred to in the first sentence. To do that, we need to introduce variables and quantifiers over those variables. In doing so, we have described a new logic, the FOL.

2.2 Formulas and Their Symbols in the FOL

The Boolean symbols in the FOL are the same as in the propositional logic and are probably familiar to most. There is the unary operator negation $\neg$ and the binary operators conjunction $\wedge$ (logical AND), disjunction $\vee$ (logical OR), implication $\rightarrow$, and equivalence $\leftrightarrow$ (if and only if).

In addition, the FOL adds variables two new symbols in the form of quantifiers over those variables. They are the universal quantifier $(\forall x)$ (read "for all x") and the existential quantifier $(\exists x)$ (read "there exists an x"). In order to evaluate the truth value of a formula in the FOL, all of the variables must be quantified, either with $\forall$ or $\exists$, exactly once. So, for example, the truth value of the FOL formula $(\forall x)(x > y)$ cannot be determined while the truth value of $(\forall x)(\exists y)(x > y)$ can. The FOL formula $(\forall x)(\exists y)(\exists x)(x > y)$ is not valid because the variable x is quantified twice.

Boolean symbols, variables, and their quantifiers may appear anywhere in a FOL formula. By convention, the scope of a quantifier is delimited by parenthesis, or in the absence of parentheses, applies only to the nearest variable. This provides for an expressively rich logic.

For example, consider the following three statements about the natural numbers.

- For every number, there is one and only one immediate successor.
- There is no number for which 0 is the immediate successor.
- For every number other than 0, there is one and only one immediate predecessor.

Let $p(x)$ and $s(x)$ represent the predecessor and successor functions over natural numbers, respectively. The above three statements can be expressed in the FOL as follows.

- $(\forall x)(\exists y)((y = s(x)) \wedge (\forall z)((z = s(x)) \rightarrow (y = z)))$
- $\neg((\exists x)(0 = s(x)))$
- $(\forall x)(\neg(x = 0) \rightarrow ((\exists y)((y = p(x)) \wedge (\forall z)((z = p(x)) \rightarrow (y = z)))))$

2.3 Prenex Normal Form

There are rules in the FOL that allow us to manipulate formulas so that, although they look different and in some cases radically different, they are still logically equivalent. Two formulas are said to be logically equivalent if one is true when the other is true and one is false when the other is false, i.e., the two formulas always have the same truth value.

Two examples of such rules are the ones that allow us to eliminate all equivalence $(\leftrightarrow)$ and implication $(\rightarrow)$ symbols from our formula. Any FOL formula of the form $p \leftrightarrow q$ can be converted to the logically equivalent formula $(p \rightarrow q) \wedge (q \rightarrow p)$ and any FOL formula of the form $p \rightarrow q$ can be converted to the logically equivalent formula $\neg p \vee q$. With these two rules any formula in the FOL can be converted into a logically equivalent one that does not contain any implication $(\rightarrow)$ or equivalence $(\leftrightarrow)$ symbols.

There are two more rules that are used to move negation symbols from the left to the right side of a quantifier. Any formula in the FOL of the form $\neg(\exists x)p$ can be converted to the logically equivalent formula $(\forall x)\neg p$ and any formula of the form $\neg(\forall x)p$ can be converted to the logically equivalent formula $(\exists x)\neg p$. These two rules can be used to propagate negation signs through a sequence of quantifiers. For example, the formula $\neg(\exists x)(\forall y)(\forall z)(x+y+z = 0)$ can be converted to the logically equivalent formula $(\forall x)(\exists y)(\exists z)\neg(x + y + z = 0)$.

There is a subtle assumption that must be made when applying the second rule. In order for the two formula to be logically equivalent, we must be assured that the sets from which our variables draw their values are non-empty. If we allow empty sets then $\neg(\forall x)p$ is not logically equivalent to $(\exists x)\neg p$.

There are more rules that allow us to move all the quantifiers to the left of the formula. Their presentation and explanation are beyond the scope of this brief overview. It is sufficient to say that, under the assumption of non-empty sets, we can always convert statements in the FOL into logically equivalent statements having the form $(Q_1x_1)\ldots(Q_nx_n)(M)$ where every $(Q_ix_i), i = 1,\ldots,n$, is either $(\forall x_i)$ or $(\exists x_i)$ and M is a formula containing no quantifiers, implication symbols ($\rightarrow$), or equivalence symbols ($\leftrightarrow$). Formulas in such a form are said to be in *prenex normal form (PNF).*

2.4 Interpretation of Formulas in PNF

Quantifiers appearing in formulas that are in PNF are interpreted from left to right. In many cases, the order of interpretation does not matter. Unfortunately, there are times when the order does matter.

Formulas with only universal or only existential quantifiers may interpret their quantifiers in any order. That is, $(\forall x)(\forall y)(x + y = 0)$ is logically equivalent to $(\forall y)(\forall x)(x + y = 0)$. Likewise, for existential quantifiers the formula $(\exists x)(\exists y)(x + y = 0)$ is logically equivalent to $(\exists y)(\exists x)(x + y = 0)$.

Even *some* formulas having both types of quantifiers may change the order in which they are interpreted. For example, $(\forall x)(\exists y)(x = 0 \wedge y = 5)$ is logically equivalent to $(\exists y)(\forall x)(x = 0 \wedge y = 5)$.

However, the formula $(\forall x)(\exists y)(x + y = 0)$ has a very different meaning than $(\exists y)(\forall x)(x + y = 0)$. The first formula means given any value of x you can always find a value y having with the same magnitude but opposite sign. The second formula means that there is a value y that has the same magnitude but opposite sign of all values of x.

Loosely speaking, the ability to interpret a string of quantifiers of different types in an arbitrary order is dictated by the interaction between the variables they are quantifying. When the quantified variables have no interaction, i.e., they are independent of each other, as in the formula $(\exists y)(\forall x)(x = 0 \wedge y = 5)$, the the oder of the quantifiers, and hence their interpretation may be arbitrarily changed while preserving logical equivalence. On the other hand, when there is an interaction between the quantified variables, as in the formula $(\exists y)(\forall x)(x + y = 0)$, then the order of the quantifiers does matter. In other words, in that case the order of the quantifiers has a material effect on the semantics of the formula.

In short, the order of adjacent quantifiers of the same kind may be swapped freely and the semantics of the formula remains unchanged. However, adjacent quantifiers that are not the same kind may not always be swapped while preserving logical equivalence.

3 Queries And Their FOL PNF Interpretations

Our primary goal in this proposal is to design a query language that has the full expressive power of the FOL and is unambiguous. In this section we present our proposed language by illustrating that there is a one-to-one correspondence between statements in our proposed language and sets of logically equivalent formulas in the FOL.

3.1 Existential Quantification

Consider the following query.

```
electron.px > 5
```

By default the reference to `electron` is an *existentially quantified reference*, or more specifically an *unextended existentially quantified reference.* The distinction between unextended and extended references will be explained shortly. The above query will select all the events that satisfy the FOL formula

$$(\exists electron)(electron.px > 5)$$

which means there is an electron that satisfies the inequality. Now consider the following query involving two unextended existential references.

```
electron.px > 5 && electron.py > 10
```

Because the order in which we interpret adjacent quantifiers of the same type does not matter, this query represents either of the logically equivalent FOL formulas

$$(\exists electron_a)(\exists electron_b)(electron_a.px > 5 \land electron_b.py > 10)$$

or

$$(\exists electron_b)(\exists electron_a)(electron_a.px > 5 \land electron_b.py > 10)$$

both of which mean there is an electron that satisfies the first inequality and an electron to satisfy the second inequality. For brevity, we will no longer list all the FOL formulas resulting from changing the order of adjacent quantifiers of the same type. We simply point out that a single query often represents an entire set of logically equivalent formula in the FOL.

In the above query, there is no guarantee that the two electrons will be the same. If the intent is to insist that the two electrons be the same, the query must be changed by making the references to `electron` *extended references.* The following query uses extended references which forces the two electrons to be the same.

```
electron#1.px > 5 && electron#1.py > 10
```

This represents the FOL formula

$$(\exists electron)(electron.px > 5 \land electron.py > 10)$$

which means there is a single electron that satisfies both inequalities. This is the first of two reasons why one would use an extended existential reference rather than an unextended one. Extended existential references are used to force binding between multiple instances of objects with the same name in a query. To illustrate the second reason, consider the following more complex query.

```
electron#1.px+electron#1.py > 0 && electron#2.px+electron#2.py > 10
```

This represents FOL formula

$$(\exists electron_a)(\exists electron_b)(electron_a.px + electron_a.py > 0 \land electron_b.px + electron_b.py > 10)$$

which means there is an electron that satisfies the first inequality and an electron that satisfies the second.

The primary purpose of extended references is to force binding of objects with the same name. By default in the FOL, objects with different names may instantiate to the same object. In other words, the `electron#1` and `electron#2` might refer to the same electron. If the intent is to require that `electron#1` be an electron other than `electron#2`, we introduce the ***Discrimination Matrix.***

An implementation must provide a mechanism by which the user can use existentially quantified extended references to insist that references with different names do not instantiate to the same object simultaneously. The implementation can then construct a Discrimination Matrix that will be enforced during query processing.

A suggestion for such a mechanism is to allow users to explicitly specify where the differences must occur as in the following query.

```
electron#1.px+electron#1.py > 0 && electron#2.px+electron#2.py > 10
&& electron#1 != electron#2
```

This represents the FOL formula

$$(\exists electron_a)(\exists electron_b)$$
$$(electron_a.px + electron_a.py > 0 \land electron_b.px + electron_b.py > 10 \land \neg(electron_a = electron_b))$$

which means there is an electron that satisfies the first inequality and ***a different*** electron that satisfies the second.

For queries that require distinction between many objects, the implementation could provide a convenience function `diff()`. So, for example, `diff(electron#1, electron#2, electron#3)` could be expanded to

```
electron#1 != electron#2 && electron#1 != electron#3 && electron#2 != electron#3.
```

The details of how the implementation allows the user to specify such discriminations and how it uses those specifications to construct, and later utilize, the Discrimination Matrix is beyond the scope of this proposal. However, we acknowledge the necessity for such a matrix.

We believe that using extended existential references in this way, along with the Discrimination Matrix, eliminates the need for the `#ANY` token in the language.

3.2 Universal Quantification

Consider the following query.

```
electron#ALL.px > 5
```

The reference to `electron` is a ***universally quantified unextended reference.*** The above query will select all the events that satisfy the FOL formula

$$(\forall electron)(electron.px > 5)$$

which means all electrons satisfy the inequality. Now consider the following query involving two unextended universal references.

```
electron#ALL.px + electron#ALL.py > 5
```

This represents the FOL formula

$$(\forall electron_a)(\forall electron_b)(electron_a.px + electron_b.py > 5)$$

which means that every possible pair of electrons satisfy the inequality. Again, if the intent is that we are referring to a single electron, the query must be changed by making the references to `electron` extended references. The following query uses extended universal references which forces the two electrons to be the same.

```
electron#ALL1.px + electron#ALL1.py > 5
```

This represents the FOL formula

$$(\forall electron)(electron.px + electron.py > 5)$$

which means that all electrons satisfy the inequality.

Unlike existential quantifiers, there is no need for a Discrimination Matrix when using universal quantifiers. The primary reason for extending universal references is to bind references with the same name. There is one additional reason that is made clear in the next section.

3.3 Combining Existential and Universal Quantification: Skolem Functions

Consider the following query.

```
electron#ALL.px > 5 && photon.py > 10
```

This can be represented by the FOL formula

$$(\forall electron)(\exists photon)(electron.px > 5 \wedge photon.py > 10)$$

which means that all the electrons satisfy the first inequality and there is a photon satisfying the second. Because there is no interaction between the variables, this is an example where the order of the quantifiers does not matter. However, consider this query.

```
electron#ALL.px + photon.py > 10
```

This may validly be interpreted as two *very different* FOL formulas,

$$(\forall electron)(\exists photon)(electron.px + photon.py > 10)$$

and

$$(\exists photon)(\forall electron)(electron.px + photon.py > 10).$$

In the first formula we want the events where all the electrons can find some photon to satisfy the inequality. In the second we want all the events where there is a single photon that satisfies the inequality with all the electrons. In short, this is an example where the order of the quantifiers does matter. We need a mechanism to explicitly state the position of the existential quantifiers with respect to the universal quantifiers. We borrow a tool from automated reasoning called Skolem functions and exploit the extended reference of the universal quantifiers.

Stated briefly (and loosely), when an existential quantifier appears to the right of a universal quantifier, the instantiation of the existentially quantified variable is made with respect to, or in the context of, the already instantiated universal quantified variable. In a loose sense, in order to make the formula true the value the existentially quantified variable instantiates to is a function of the universal quantifier to its left. In fact, it is a function of *all* the universal quantifiers to its left.

So, when writing formulas that involve existentially and universally quantified variables that interact, the order of the quantifiers must be explicitly stated. To do this, the user can use an extended universal reference and "Skolemize" the existentially quantified variable using that reference. For example, consider the following query.

```
electron#ALL1.px + photon(1).py > 10
```

It uses `(1)` to indicate that the existentially quantified variable `photon` appears after the universally quantified variable with an extended reference of `1`. Now the above query can now unambiguously be interpreted as the FOL formula

$$(\forall electron)(\exists photon)(electron.px + photon.py > 10)$$

and the query

```
electron#ALL1.px + photon.py > 10
```

is equivalent to the query

```
electron#ALL.px + photon.py > 10
```

which can both be unambiguously interpreted as the FOL formula

$$(\exists photon)(\forall electron)(electron.px + photon.py > 10).$$

This notation can be used to disambiguate the most complex queries. For example,

```
electron#ALL.x + electron#ALL1.y + electron#ALL2.z
+ photon.x + photon#1(1,2).y + photon#2(1,2).z > 100
```

can be interpreted as the FOL formula

$$(\exists photon_a)(\forall electron_a)(\forall electron_b)(\exists photon_b)(\exists photon_c)(\forall electron_c)$$
$$(electron_c.x + electron_a.y + electron_b.z + photon_a.x + photon_b.y + photon_c.z > 100)$$

Note that the relative order of adjacent existential quantifiers and adjacent universal quantifiers does not matter.

4 Membership

The current implementation of EQL supports the notion of membership. For example, a jet3 object can have a number of electrons and photons associated with it. In this sense, we say that the electrons and photons are "members of" or "belong to" the jet3 object. Traditionally, we have denoted membership with a ".", as in the query

```
jet3.electron.px > 10
```

which represents the FOL formula

$$(\exists jet3)(\exists electron)(memberof(electron, jet3) \wedge electron.px > 10)$$

where $memberof(X, Y)$ means X is a member of Y. This query asks for all events where there is an electron that is a member of a jet3 object that satisfies the inequality. Here are some other example queries to further illustrate.

```
jet3#ALL.electron.px > 10
```

which represents the FOL formula

$$(\exists electron)(\forall jet3)(member(electron, jet3) \wedge electron.px > 10)$$

and asks that a single electron be a member of all jet3 objects and that it satisfies the inequality. To switch the order of the quantifiers, and hence the meaning, the query should be re-written as

```
jet3#ALL1.electron(1).px > 10
```

which is represented by the FOL formula

$$(\forall jet3)(\exists electron)(member(electron, jet3) \wedge electron.px > 10)$$

and means that every jet3 object has an electron, a potentially different electron for each jet3 object, that satisfies the inequality. Another example query is

```
jet3.electron#ALL.px > 10
```

which is represented by the FOL formula

$$(\exists jet3)(\forall electron)(member(electron, jet3) \wedge electron.px > 10)$$

and means there is a single jet3 object that contains all the electrons and they all satisfy the inequality. Again, changing the order of the quantifiers changes the meaning.

```
jet3(1).electron#ALL1.px > 10
```

which is represents the FOL formula

$$(\forall electron)(\exists jet3)(member(electron, jet3) \wedge electron.px > 10)$$

and means that every electron is a member of some jet3 object, potentially different jet3 objects for different electrons, and the inequality is satisfied by all of them. Finally there is the query

```
jet3#ALL.electron#ALL.px > 10
```

which is represents the FOL formula

$$(\forall jet3)(\forall electron)(member(electron, jet3) \wedge electron.px > 10)$$

and means that every electron is a member of every jet3 object and the inequality is satisfied by all electrons. To test the inequality for only those electrons belonging to one or more jet3 objects, we change the query as follows.

```
!(jet3#ALL1.electron#ALL1) || jet3#ALL1.electron#ALL1.px > 10
```

which represents the FOL formula

$$(\forall jet3)(\forall electron)(\neg member(electron, jet3) \vee electron.px > 10)$$

and means if an electron is a member of a jet3 object, then it must satisfy the inequality.

We have added a new construct into the language, a test for membership. In the query above, the test is denoted by `jet3#ALL1.electron#ALL1` (we were required to extend the universal references to bind them to the references in the inequality). The test for membership evaluates to true if and only if the instance of the electron is a member of the instance of the jet3 object.

The test for membership also provides a natural way in which to interpret queries specifying fields within objects that are not there. A query that specifies `electron.px`, is explicitly using the membership operator "." and it has the same semantics when used to specify an electron within a jet3 object, e.g., `jet3.electron.px`. So, when the user writes the query

```
electron.px > 10
```

it is *really* represents the FOL formula

$$(\exists electron)(\exists px)(member(px, electron) \wedge electron.px > 10).$$

If `px` is not field in `electron`, then this query will always fail. If the intent is to include those events where the specified field does not exist, the query can simply be modified to include a test for membership, e.g.,

```
!(electron#1.px#1) || electron#1.px#1 > 10
```

which is represented by the FOL formula

$$(\exists electron)(\exists px)(\neg member(px, electron) \vee electron.px > 10)$$

In this query those events where `px` is not field in `electron` will be retrieved.

The data values for many of the underlying objects are currently scalars. In the event that it becomes necessary or convenient to extend the data types to vectors or variable lengthed lists, the test for membership in conjunction with the rules for quantifier interpretation we have presented extend naturally to accommodate these changes. These same rules for quantification would simply apply to the new multivalued fields.

5 Summary

The following is a summary of the proposal in the context of modifications to the existing language EQL.

1. Existential Quantification

   - By default, all references are existentially quantified.
   - Extending existentially quantified references binds references with the same name.
   - By default, references with different names may instantiate to the same object.
   - Must provide a mechanism, perhaps `diff()`, that allows users to insist that existentially quantified variables with different names cannot instantiate to the same object simultaneously. Must build and implement a single Discrimination Matrix that describes this relationship between all existentially quantified variables.

   Note that the rules for existential quantification eliminate the need for the `#ANY` token.

2. Universal Quantification

   - Universally quantified references must be made explicit with the `#ALL` token.
   - Extending universally quantified references binds references with the same name.
   - Must provide a mechanism to specify the relative order of the universal and existential quantifiers. This may be done by extending the universally quantified variables and using those indentifiers to "Skolemize" the existentially quantified references.

3. Membership

   - Define "." as an infix binary membership operator such that $X.Y$ means Y is a member of X. Provide a mechanism to explicitly test for membership.

6 Conclusion

Any statement that can be represented in the FOL can be placed into PNF. Once in that form, it can be translated into a query in our proposed language. As a result, we have a language that has the full expressive power of the FOL. The converse is also true, that is, every query can be interpreted as a set of logically equivalent PNF formulas in the FOL, thus disambiguating the language.

One of our goals in designing the language was to keep the necessary changes to the current syntax and semantics to a minimum. We believe this proposal describes a language that represents a minor modification to the syntax of the existing language EQL while disambiguating the entire language and expanding its expressive power to match the FOL.

7 Acknowledgements

I thank William McCune at Argonne National Laboratory's Mathematics and Computer Science Division for our discussions. They were a valuable contribution to this work.