Existential Graphs and EGIF

John F. Sowa

Abstract. Peirce developed the algebraic notation for first-order and higher-order predicate calculus in 1885. In 1897, he developed the graphic notation for existential graphs (EGs), which have the full expressive power of his algebra. Section 1 of this article introduces existential graphs in the form he preferred in his later years (1906 to 1911). Section 2 presents the grammar for the Existential Graph Interchange Format (EGIF), which is a linear notation that has the full expressive power of the ISO standard 24707 for Common Logic (CL). The examples in Section 1 show how the FOL features of EGs are mapped to EGIF and to predicate calculus. That mapping is sufficient to determine the semantics of EGs that express FOL. For the semantics of Peirce's extensions beyond FOL, EGIF can also serve as a tool for analyzing and clarifying his many variations. This article contains graphics, excerpts, and adaptations from various publications by the author. For sources, see the references at the end.

1. Syntax of Existential Graphs

Charles Sanders Peirce was a pioneer in logic. Although Frege published the first complete system of first-order logic in 1879, no one else adopted his notation. Peirce published the algebraic version of FOL and HOL in 1885. With a change of symbols by Peano and some extensions by Whitehead and Russell, Peirce-Peano algebra is still the most widely used logic today (Putnam 1982). But as early as 1882, Peirce experimented with graph notations for "the atoms and molecules of logic." In 1897, he developed existential graphs (EGs) as a notation that expressed the full semantics of first-order predicate calculus with equality. For the first-order subset of EGs, Peirce used the same structure and rules of inference in every manuscript from 1897 to 1911. But he experimented with variations, extensions, and semantic foundations (Roberts 1973). In this article, the semantics is specified indirectly by showing how the features of the EG graphics and the EGIF linearization are mapped to corresponding features of predicate calculus. For Peirce's rules of inference and the model-theoretic semantics he called *endoporeutic*, see the article by Sowa (2011).

As an example, the EG on the left of Figure 1 asserts that there is a phoenix. The line, which he called a *line of identity*, represents existence. By itself, the line asserts "There is something." The word *phoenix* is the *name* of a relation type. The line attached to the name asserts "There exists a phoenix." For the other two EGs, Peirce (NEM 3:163) explained, "To deny that there is any phoenix, we shade that assertion which we deny as a whole [EG in the middle]. Thus what I have just scribed means 'It is false that there is a phoenix.' But the [EG on the right] only means 'There is something that is not identical with any phoenix'."

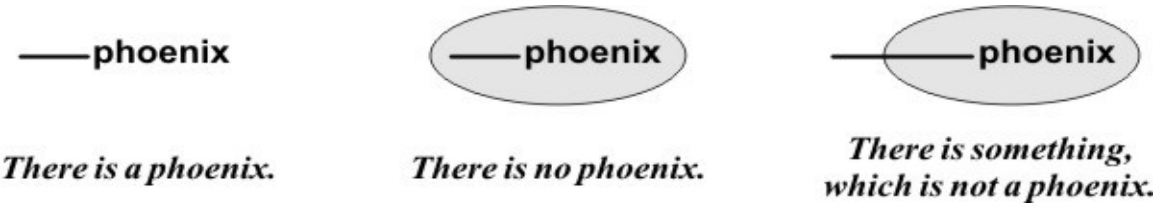

Figure 1. Three existential graphs

To indicate negation in his early EGs, Peirce used an unshaded oval enclosure, which he called a *cut* or a *sep* because it cut or separated the *sheet of assertion* into a positive (outer) area and a negative (inner) area. In the later versions, he added shading to highlight the distinction between positive and negative areas. The following table shows the Existential Graph Interchange Format (EGIF) for each of the graphs in Figure 1 and the corresponding formula in predicate calculus.

Figure 1	EGIF	Predicate Calculus
Left	**(phoenix *x)**	**∃x phoenix(x)**
Middle	**~[(phoenix *x)]**	**~∃x phoenix(x)**
Right	**[*x] ~[(phoenix x)]**	**∃x ~phoenix(x)**

In the EGIF for the graph on the left of Figure 1, the parentheses enclose the name **phoenix** of a *monad* (monadic relation) and a *defining label* ***x**. The defining label represents the beginning of a line of identity in the graphic EG. It is mapped to an existentially quantified variable ∃**x** in predicate calculus. For the EG in the middle, the shaded oval is represented in EGIF by a tilde **~** for negation and a pair of square brackets to enclose the negated subgraph: **~[ ]**. For the EG on the right, the beginning of the line of identity is outside the shaded oval. Since there is no relation outside the negation, the defining label is contained in a *coreference node* **[*x]** in front of the negation. Inside the parentheses of the relation, the label **x** without an asterisk is a *bound label* that is within the *scope* of the defining label ***x** on the outside.

When an oval is drawn inside another oval, the doubly nested area is positive (unshaded), as in Figure 2. Any area nested inside an odd number of ovals is shaded, and any area inside an even number of ovals (possibly zero) is unshaded. As Peirce said, "The graph [on the left] asserts that it thunders without lightning... a denial shades the unshaded and unshades the shaded. Consequently [the graph on the right] means 'If it thunders, it lightens'."

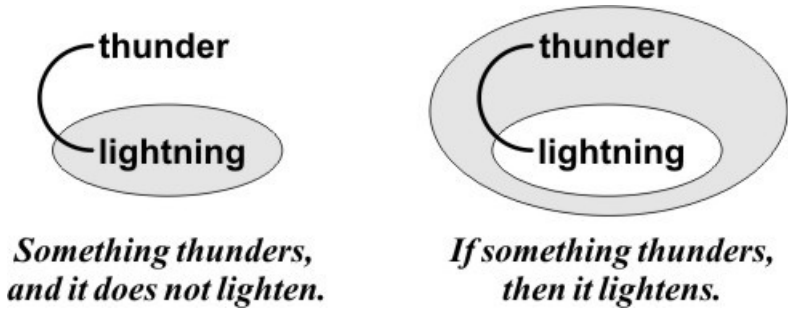

Figure 2. An EG with one negation and an EG with two negations

In the table for Figures 2, the conjunction (AND operator) is implicit in EGIF; no symbol is required. But the formula in predicate calculus requires the symbol ∧. In the last line of the table, the EGIF uses the option of replacing the two negations with the keywords **If** and **Then**; the predicate calculus uses the universal quantifier ∀ and the implication ⊃**.**

Figure 2	EGIF	Predicate Calculus
Left	**(thunder *x) ~[(lightning x)]**	**∃x (thunder(x) ∧~lightning(x))**
Right	**~[(thunder *x) ~[(lightning x)]]**	**~∃x (thunder(x) ∧~lightning(x))**
Optional	**[If (thunder *x) [Then (lightning x)]]**	**∀x (thunder(x) ⊃lightning(x))**

In EG and EGIF, a nest of two negations is used to express if-then statements in English. Peirce called that combination a *scroll*. To improve readability, the optional line in the table shows how the tildes for the two negations may be replaced by the keywords **If** and **Then**. In predicate calculus, **~∃** is equivalent to **∀~**. With that conversion, the formula for Figure 2 becomes **∀x~(thunder(*x*) ∧~lightning(x))**. By the definition of the implication operator ⊃, the formula **~(p ∧~q)** is equivalent to **p⊃q**. Therefore, the formula for the EG on the

right of Figure 2 may be rewritten as **∀x(thunder(x) ⊃ lightning(x))**. In English, this formula may be read "For every x, if x thunders, then x lightens." This example shows that the EG and EGIF for if-then, has a more direct mapping to English than the operator ⊃ in predicate calculus. In fact, EGs are isomorphic to the *discourse representation structures*, which Kamp and Reyle (1993) developed for representing natural language semantics.

Figure 3. There is a male African human

Peirce continued, "A graph may be complex or indivisible. Thus [Figure 3 shows] a graph instance composed of instances of three indivisible graphs which assert 'there is a male', 'there is something human', and 'there is something African'. The syntactic junction or point of *teridentity* asserts the identity of something denoted by all three." In EGIF, Figure 3 may be represented by three indivisible nodes followed by a *coreference node* **[x y z]**, which shows that that three lines of identity refer to the same individual. Peirce called such a junction a *ligature*. Since all three lines refer to the same thing, the defining labels ***y** and ***z** may be replaced by bound labels for ***x**. Then the coreference node is no longer needed, and it may be deleted, as shown in the following table:

Figure 3	EGIF	Predicate Calculus
Teridentity	**(male *x) (human *y) (African *z) [x y z]**	**∃x ∃y ∃z (male(x) ∧human(y) ∧African(z) ∧x=y ∧y=z)**
One label	**(male *x) (human x) (African x)**	**∃x (male(x) ∧human(x) ∧African(x))**

In modern terminology, Peirce's indivisible graphs are called *atoms*. In predicate calculus, each atom consists of a single relation followed by its list of *arguments*, which Peirce called *logical subjects*. In EGs, an N-adic relation has N *pegs*, each of which is attached to a line of identity for its logical subject. In EGIF, each atom is represented by a pair of parentheses that enclose a relation name followed by a list of defining labels or bound labels for its logical subjects..

Peirce represented a *proposition* as a relation with zero pegs. He called it a *medad*; the prefix *me-* comes from the Greek μη for *not*. In EGIF, a proposition **p** is represented as a relation with no logical subjects: **(p)**. In early versions of EGs, Peirce distinguished two subsets: Alpha for propositional logic and Beta for first-order logic. By treating medads as relations, he avoided the need to distinguish Alpha from Beta. The same rules of inference apply to both.

Peirce: "Every indivisible graph instance must be wholly contained in a single area. The line of identity may be regarded as a graph composed of any number of dyads '**—is—**' or as a single dyad." To illustrate that option, consider the graph **man—African**, which may be read "There is an African man." Replacing the line with two copies of **—is—** would break the line into three segments: **man—is—is—African**. This EG may be read, "There is a man that is something that is something African." Following is the EGIF and predicate calculus:

(man *x) (is x *y) (is y *z) (African z)

∃x ∃y ∃z (man(x) ∧x=y ∧y=z ∧African(z))

The two copies of the dyad **is** are equivalent to two coreference nodes **[x *y] [y *z]**, which may be replaced by a single coreference node **[x *y *z]**. Since all three labels are coreferent, the bound label **x** may replace all occurrences of the other defining labels and bound labels. The result is **(man *x) (African x)**.

With these examples, Peirce presented EG syntax in less than three printed pages. In another four pages, he presented the rules of inference, a brief summary of endoporeutic, and an example that shows how an Aristotelian syllogism can be translated to an EG and proved by the EG rules of inference (Sowa 2010). As

he showed, EGs have only two explicit operators: a line to represent the existential quantifier and an oval to represent negation. Conjunction is an implicit operator, expressed by drawing any number of graphs in the same area. Equality is expressed by joining lines.

All other operators of first-order logic are represented by combining these primitives. Figure 4 shows three composite operators defined in terms of nested negations.

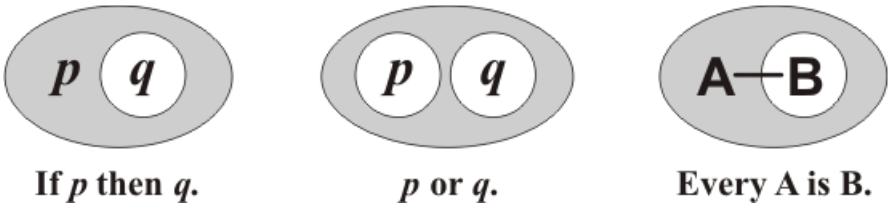

Figure 4. Three composite operators

Figure 4	EGIF	Predicate Calculus
If p, then q.	**[If (p) [Then (q)]]**	**p ⊃ q**
p or q.	**~[~[(p)] ~[(q)]]**	**p ∨ q**
Every A is B.	**[If (A *x) [Then (B x)]]**	**∀x (A(x) ⊃ B(x))**

Although these three operators are composite, their graphic patterns are just as readable as the algebraic formulas with the special symbols ⊃, ∨, and ∀. In fact, the explicit nesting of EG ovals directly shows the scope of quantifiers. But the usual Boolean operators ∧, ∨, and ⊃ look so similar that students find it hard to remember that each one has a different effect on the scope of quantifiers. In EGIF, the propositions or medads **(p)** and **(q)** are enclosed in parentheses because they are represented as relations with zero logical subjects (arguments).

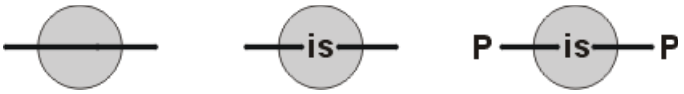

Figure 5. Stating that two things are not identical

Sometimes, the dyad **—is—** can clarify the translation of an EG to a sentence or formula. For example, Figure 5 shows three ways of saying that there exist two things. In the EG on the left, the shaded area negates the junction of the lines of identity on either side. To emphasize what is being negated, the EG in the middle replaces part of the line with the dyad **—is—**. Therefore, that EG may be read "There is something **x** that is not something **y**." The graph on the right says that there exist two different things with the property P or simply "There are two Ps."

Figure 5	EGIF	Predicate Calculus
Left	**[*x] [*y] ~[[x y]]**	**∃x ∃y ~(x=y)**
Middle	**[*x] [*y] ~[(is x y)]**	**∃x ∃y ~is(x,y)**
Right	**(P *x) (P *y) ~[(is x y)]**	**∃x ∃y (P(x) ∧ P(y) ∧ ~is(x,y))**

As these examples show, EGs state identity by joining two lines and deny identity with an oval on a line. The linear notations require named labels and symbols such as **x=y** or **x≠y**. Without labels on lines, graphic EGs

do not need special rules for replacing one label with another. But Peirce said that a complex graph with criss-crossing lines may be clearer if some connections are shown by labels.

In summary, EGIF has the same primitives and conventions as EGs, but with the adaptations necessary to linearize the graphs. An oval for negation is represented by a tilde **~** followed by a pair of square brackets to enclose the EGIF for the subgraph inside the oval. A line of identity is represented by a defining label and zero or one bound label. (See Figure 1 for examples of defining labels with no bound labels.) The graphic options for connecting and extending lines are shown by coreference nodes. A ligature is represented by a coreference node with two or more labels (defining or bound). Many coreference nodes may be eliminated by replacing several defining and bound labels with a single defining label and multiple bound labels. (See Figure 3.) The only coreference nodes that cannot be eliminated are those that are enclosed in a negation and show a junction of two or more lines whose defining labels are outside the negation. (See Figure 5.)

2. EGIF Grammar

Existential Graph Interchange Format (EGIF) is a linear notation that serves as a bridge between EGs and other notations for logic. Over the years, Peirce had written manuscripts with variant notations, terminology, and explanations (Roberts 1973). Those variations have raised some still unresolved questions about possible differences in their semantics. With its formally defined semantics, EGIF provides one precise interpretation of each graph translated to it. Whether that interpretation is the one Peirce had intended is not always clear. But the EGIF interpretation serves as a fixed reference point against which other interpretations may be compared and analyzed.

Every EGIF statement has a formally defined mapping to CGIF (Conceptual Graph Interchange Format), whose semantics is specified in ISO/IEC standard 24707 for Common Logic (CL). The semantics of the corresponding CGIF statement shall be called the *default semantics* of EGIF. To express the full CL semantics, the grammar rules in Section 2.3 specify two extensions to EGIF: (1) functions and (2) bound labels as names of relations or functions. EG relations, by themselves, can represent functions. But treating functions as a special case can simplify the inferences and make a major improvement in the speed of computation. The option of bound labels as names of relations and functions goes beyond first-order logic, and it supports some features of Peirce's Gamma graphs.

But EGIF can also be used to represent and analyze Peirce's graphs in his various manuscripts and publications. His original Alpha and Beta graphs, which represent propositional and first-order logic, are consistent with the default semantics. But with his Gamma graphs, he experimented with various extensions for higher-order logic, modal logic, and metalanguage. Section 2.4 adds grammar rules for metalanguage in a way that is consistent with the proposed IKL extensions to Common Logic. Whether the IKL extensions are compatible with Peirce's intentions is another question for further research.

Section 2.1 presents the lexical rules for the kinds of character strings used in EGIF. These rules support the features that Peirce used. They also add features that he never used in his examples, but they are compatible with them. Section 2.2 presents the phrase-structure rules. Section 2.3 states context-sensitive constraints and gives some examples. Section 2.4 discusses extensions beyond Common Logic and their use in representing Peirce's Gamma graphs. But Peirce also speculated about variations that are not supported by these rules. Anyone who wishes to extend EGIF to represent them may do so, but they should clearly state where they go beyond the version of EGIF specified here.

2.1 Lexical Rules

All EGIF grammar rules are stated as Extended Backus-Naur Form (EBNF) rules, as defined by the ISO/IEC standard 14977. The lexical rules specify names and identifiers, which exclude white space except as noted. The phrase-structure rules in Section 2.2 specify larger combinations that may have zero or more characters of white space between constituents. Each EBNF rule is preceded by an English sentence that serves as an informative description of the syntactic category. If any question arises, the EBNF rule shall be normative.

The following four lexical categories are defined formally in Section A.2 of ISO/IEC 24707, which is the normative specification for Common Logic and its standard dialects. The brief definitions here are informative summaries. White space, which is any sequence of one or more white characters, is permitted only in quoted strings and enclosed names.

- A *digit* is any of the ten decimal digits: **0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**.
- An *integer* is an optional sign (**+** or **−**) followed by a sequence of one or more digits.
- An *enclosed name* is any sequence of Unicode characters, except control characters, preceded and followed by a double quote **"**. Any double quote internal to an enclosed name shall be represented by the string **\"**. Any backslash internal to an enclosed name shall be represented by the string ****.
- A *letter* is any of the 26 upper case letters from **A** to **Z** or the 26 lower case letters from **a** to **z**.
- A *white* character is a space, a tab, a new line, a page feed, or a carriage return.

In addition to the above lexical categories, EGIF uses the following lexical category, which is specified in Section B.2 of ISO/IEC 24707:

- An *identifier* is a letter followed by zero or more letters, digits, or underscores **_**.

  identifier = letter, {letter | digit | '_'};

Identifiers and enclosed names are case sensitive: the identifier **Apple** is distinct from **apple**. But an identifier is considered identical to the enclosed name with the same case and spelling: **Apple** is identical to **"Apple"**, and **apple** is identical to **"apple"**. An integer is also considered identical to the enclosed name that contains the same sign and string of digits.

2.2 Phrase-Structure Rules

Unlike the lexical rules, the phrase-structure rules for EGIF permit an arbitrary amount of white space between constituents. White space is only necessary to separate adjacent identifiers. For example, the following two EGIF statements are semantically identical:

~[(mother*x)~[(woman x)]]

~ [(mother * x) ~ [(woman x)]]

In English, either statement may be read "If some *x* is a mother, then *x* is a woman." For better readability, a nest of two negations may be written with the first negation **~[** replaced by **[If** and the second negation replaced by **[Then**:

[If (mother *x) [Then (woman x)]]

Each phrase-structure rule is described by an informal English comment and by a formal EBNF rule. For any question, the EBNF rule shall be normative. For any derivation, the rule that defines EG shall be the starting rule. The next paragraph defines informal terms and conventions that relate EGIF to the graphic EGs.

An *area* is a space where existential graphs may be drawn or asserted. The outermost area on which EGs may be asserted is called the *sheet of assertion* (SA). The SA may be cut or fenced off by *oval enclosures*, which are areas that contain *nested* EGs. For propositional and first-order logic (Peirce's Alpha and Beta graphs), the only oval enclosures are used to represent negations. In EGIF, the area of a negation is represented as **~[EG]**, where **EG** represents an existential graph as a set of *nodes* (possibly empty). Some of the nodes in the EG may be negations whose *nested areas* may contain more deeply nested EGs. There is no limit to the depth of nesting. In an informal definition, a term is printed in italics, as in *bound label*. But the same term in EBNF is printed in boldface, as in **Bound Label**. EBNF permits spaces in an identifier, but EGIF does not.

1. A *bound label* is an identifier. The bound label shall be *bound* to a defining label with the same identifier as the bound label.

   Bound Label = identifier;

2. A *coreference node* consists of a left bracket **[**, one or more names, and a right bracket **]**. A *defining node* is a coreference node that contains exactly one defining label. An *extension node* is a coreference node that contains exactly one name that is not a defining label.

   Coreference Node = '[', Name, {Name}, ']';

3. A *defining label* consists of an asterisk ***** and an identifier.

   Defining Label = '*', identifier;

4. A *double negation* is either a negation whose enclosed EG is a negation or a scroll whose first EG is a blank.

   Double Negation = '~', '[', Negation, ']' | '[', 'If', '[' 'Then', EG, ']' ']';

5. An *existential graph (EG)* is a set of nodes. An existential graph with zero nodes is called a *blank* or an *empty existential graph*. The order of nodes in the set is semantically irrelevant; but any node that contains a bound label shall follow (occur to the right of) the node that contains its defining label.

   EG = {Node};

6. A *function* consists of a left parenthesis **(**, a type label, zero or more names, a vertical bar **|**, a name, and a right parenthesis **)**. The names to the left of the bar may be called the inputs, and the one to the right may be called the output.

   Function = '(', Type Label, {Name}, '|', Name, ')';

7. A *name* is one of a defining label, a bound label, an identifier, an enclosed name, or an integer.

   Name = Defining Label | Bound Label | identifier | enclosed name | integer;

8. A *negation* consists of a tilde **~**, a left bracket **[**, an existential graph, and a right bracket **]**.

   Negation = '~', '[', EG, ']';

9. A *node* is a coreference node, a relation, a function, a negation, or a scroll.

   Node = Coreference Node | Relation | Function | Negation | Scroll;

10. A *relation* consists of a left parenthesis **(**, a type label, zero or more names, and a right parenthesis **)**.

    Relation = '(', Type Label, {Name}, ')';

11. A *scroll* consists of a left bracket **[**, the letters **If**, an EG, a left bracket **[**, the letters **Then**, an EG, and two right brackets **]]**. Syntactically, a scroll is an optional notation for replacing the tilde in two negations with the keywords **If** and **Then**. Both notations have identical semantics.

    Scroll = '[', 'If', EG, '[', 'Then', EG, ']', ']';

12. A *type label* is any name except a defining label. If the name is a bound label, the value associated with its defining label determines the type of some relation or function.

    Type Label = Name - Defining Label;

2.3 Constraints and Examples

As the grammar rules show, an EG is represented by zero or more EGIF nodes. The only constraints on the nodes or their ordering are determined by the location of defining labels and their bound labels. The following six rules are equivalent to the rules for scope of quantifiers in predicate calculus. These rules are not needed for the graphic EGs, because the lines of identity show the scope by direct connections, not by labels.

1. No area may contain two or more defining labels with the same identifier.
2. The *scope* of a defining label shall include the area in which it occurs and any area nested directly or indirectly in this area, unless it is blocked by rule 5 below.
3. Every bound label shall be in the scope of exactly one defining label, which shall have exactly the same identifier. It is said to be *bound* to that defining label.

4. Every defining label shall precede (occur to the left of) every one of its bound labels.
5. If a defining label with some identifier *x* occurs in an area nested within the scope of a defining label in an outer area with the same identifier *x*, then the scope of the outer defining label shall be *blocked* from that area: every bound label with the identifier *x* that occurs in this inner area shall be bound to the defining label in this area.
6. In any area, all permutations of the nodes that preserve the above constraints shall be semantically equivalent.

A name enclosed in double quotes, such as **"John Q. Public"**, may contain spaces and punctuation. To avoid quotes, other stylistic options may be used, such as **John_Q_Public** or **JohnQPublic**, but these three variants are distinct. To specify multiple names as synonyms, put them in a coreference node:

["John Q. Public" John_Q_Public JohnQPublic JQPublic JQP]

Peirce treated functions as special special cases of relations. He didn't have a notation that distinguished them from ordinary relations. In EGIF, a function may be considered a kind of relation for which the value represented by the argument (or peg) after the vertical bar is uniquely determined by the values of the pegs that precede the bar. The pegs to the left of the vertical bar may be called the *inputs*, and the one to the right of the bar may be called the *output*.

- If *f* is a function with *n* inputs and if for every *i* from 1 to *n*, the *i*-th input of one instance of *f* is coreferent with the *i*-th input of another instance of *f* in the same area, then a line of identity may be drawn to connect the output pegs of both instances.

The act of drawing a line of identity between the output pegs in the graphic notation corresponds to inserting a coreference node in EGIF or an equality in other notations for logic. This rule of inference is the basis for the method of *unification* in theorem proving systems. The EGIF grammar allows functions with zero inputs; every instance of such a function would have exactly the same output value. Therefore, the output pegs of all instances of that function may be joined. These rules imply that a function with zero input pegs may be used to represent a *Skolem constant*.

Peirce did not introduce an EG notation for proper names or constants. Instead, he used monadic relations, such as **—Alexander** or **—is Alexander**. This relation would be true of anyone named Alexander. In EGIF, a name that is true of exactly one individual may be used as the name of a function with zero inputs. In the following function, the defining label ***p** would represent a line of identity for the unique person with that name:

("Philippus Aureolus Theophrastus Bombastus von Hohenheim" | *p)

A name such as Alexander, which may be unique in a specific context, could be written as a function with an input peg for the context and an output peg for the person of that name: **(Alexander c | *p)**. Although Peirce did not define a notation for functions in EGs, the output peg of a function could be distinguished by an arrowhead in the graphic form.

The EBNF rules for relation and function allow a bound label to be the type label. That option supports the feature of Common Logic that allows quantified variables to refer to functions and relations. As an example, consider the sentence "There is family relation between any two members of the same family." To translate that sentence to any version logic, restate it with explicit variables F, R, *x*, and *y*: "For any family F and any two members *x* and *y* of F, there is a family relation R that is true of *x* and *y*." That sentence may be translated to the following EGIF:

```
[If  (family *F)  (memberOf *x F)  (memberOf *y F)
   [Then  (familyRelation *R)  (R x y) ] ]
```

The expression **(familyRelation *R)** asserts that there exists a family relation R, and the next expression uses R as a type label. For his Gamma graphs, which are discussed in the next section, Peirce experimented with graphical ways of representing such options.

2.4 Extensions for Gamma Graphs

Over half a century, Peirce developed various notations for logic and experimented with semantic extensions that go beyond first-order logic. In 1885, he introduced the algebraic notation that became predicate calculus (Putnam 1982). He used the terms *first-intentional logic* for quantifiers that range over simple individuals and *second-intentional logic* for quantifiers that range over relations. In that article, he used second intentional logic to define equality $x=y$ by a statement that for every relation R, R(x) if and only if R(y). Ernst Schröder translated Peirce's terms as *erste Ordnung* and *zweite Ordnung*, which Bertrand Russell translated back to English as *first order* and *second order*. Peirce also introduced notations for three-valued logic, modal logic, and metalanguage about logic. Roberts (1973) summarized the various graphical and algebraic notations and cited the publications and manuscripts in which Peirce discussed them.

Peirce used the term *Gamma graphs* for the versions of EGs that went beyond first-order (or first-intentional) logic. As early as 1898, he used the following example of a metalevel statement in EGs:

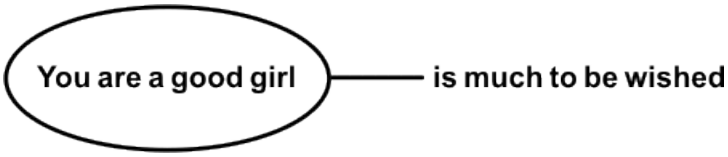

Figure 6: A metalevel existential graph by Peirce (1898)

In describing that graph, Peirce wrote "When we wish to assert something about a proposition without asserting the proposition itself, we will enclose it in a lightly drawn oval, which is supposed to fence it off from the field of assertions." Since Peirce allowed blanks to occur in the names of relations, the English sentence **"You are a good girl"** may be treated as an enclosed name of a relation with no pegs. In Common Logic, a relation with zero arguments (or zero pegs in EGs) represents a proposition. The line of identity attached to the oval asserts that there exists something that is represented by the nested EG. But Common Logic does not provide any notation or semantics for relating a CL sentence to a quantified variable.

For a proposed extension to Common Logic called IKL, Hayes and Menzel (2006) introduced an operator named *that* as a kind of function that maps an IKL sentence to a name that refers to the proposition stated by that sentence. In EGIF, the IKL that-operator could be used for a metalevel function that maps an EG to a name that refers to the proposition stated by the EG:

```
Metastatement  =  '(' 'that' EG '|' Name ')' ;
```

With the that-operator, Figure 6 could be represented by the following EGIF:

```
(that "You are a good girl" | *p)  ("is much to be wished" p)
```

In some writings, Peirce used ovals with colors or dotted boundaries to represent modality. EGIF can use identifiers such as **Possible** or **Necessary** for relations applied to propositions. The following EGIF says "That you are a good girl is possible but not necessary."

```
(that "You are a good girl" | *p)  (possible p)  ~[(necessary p)]
```

For EGs that represent first-order logic (Peirce's Alpha and Beta graphs), the semantics of Common Logic appears to be consistent with what Peirce had intended. But determining exactly what he had in mind for his many variations of Gamma graphs is still a research project. In some late manuscripts, he also hinted at a version of Delta graphs. EGIF can be a useful tool for expressing and analyzing the options.

2.5 Computer tools for processing EGIF

EGIF was originally defined as a subset of the Conceptual Graph Interchange Format (CGIF), one of the dialects of Common Logic (CL). But after some simplifications in the notation, the version of EGIF defined in this appendix still has a direct mapping to CGIF, and its semantics is still defined by its mapping to CGIF. But its syntax is simpler. As an example, "A cat is on a mat" would be translated to the following EGIF:

```
(Cat *x) (On x *y) (Mat y)
```

To translate any EGIF sentence to an equivalent CGIF sentence, put every defining label in a defining node at the beginning of the area in which it occurs. Then replace each defining label inside a relation node with a bound label with the same identifier. The result is an EGIF sentence that is logically equivalent to the original:

[*x] [*y] (Cat x) (On x y) (Mat y)

To derive a logically equivalent CGIF sentence, place a question mark **?** in front of every bound label:

[*x] [*y] (Cat ?x) (On ?x ?y) (Mat ?y)

The equivalent sentence in the Common Logic Interchange Format (CLIF) requires the keyword **exists** for the quantifier and an explicit Boolean operator **and** for the implicit conjunction:

(exists (x y) (and (Cat x) (On x y) (Mat y)))

The Heterogeneous Tool Set (HETS) provides theorem provers for Common Logic and translators to and from CL and other logics, including the logics for the Semantic Web (Mossakowski et al. 2014). Any EGIF sentence may be translated to CGIF or CLIF by the above steps and be processed by any of the HETS tools. For natural languages, discourse representation theory (Kamp et al. 2011) addresses the linguistic analysis that is prior to any representation in logic. But the base logic (DRS) is isomorphic to EGs and CGIF. For the sentence "A cat is on a mat," the DRS diagram would be linearized as

〈 **{x, y}, { (Cat x), (On x y), (Mat y) }** 〉

References

Hayes, Patrick, and Chris Menzel (2006) IKL Specification Document, http://www.ihmc.us/users/phayes/IKL/SPEC/SPEC.html (accessed 15 November 2009).

Hilpinen, Risto (1982). On C. S. Peirce's theory of the proposition: Peirce as a precursor of game-theoretical semantics. *The Monist* 65: 182-188.

ISO/IEC (1996). *Extended BNF*, (IS 14977). Geneva: International Organisation for Standardisation. http://standards.iso.org/ittf/PubliclyAvailableStandards/s026153_ISO_IEC_14977_1996(E).zip

ISO/IEC (2007). *Common Logic (CL) — A Framework for a family of Logic-Based Languages*, (IS 24707). Geneva: International Organisation for Standardisation.

Kamp, Hans, & Uwe Reyle (1993) *From Discourse to Logic*, Dordrecht: Kluwer.

Kamp, Hans, Josef van Genabith, & Uwe Reyle (2011) *Discourse Representation Theory: An Updated Survey*, in D. Gabbay (ed.), Handbook of Philosophical Logic, 2nd ed., Vol XV, pp. 125-394. http://www.ims.uni-stuttgart.de/institut/mitarbeiter/uwe/Papers/DRT.pdf

Mossakowski, Till, Mihai Codescu, Oliver Kutz, Christophe Lange, & Michael Grüninger (2014) Proof support for Common Logic, ARONL@IJCAR, http://www.iltp.de/ARQNL-2014/download/arqnl2014_paper5.pdf

Peirce, Charles Sanders (1885). On the algebra of logic. *American Journal of Mathematics* 7:180-202.

Peirce, Charles Sanders (1898). *Reasoning and the Logic of Things*, (The Cambridge Conferences Lectures of 1898), K. L. Ketner (ed) (1992). Cambridge, MA: Harvard University Press.

Peirce, Charles Sanders (NEM) *The New Elements of Mathematics*, ed. by Carolyn Eisele, 4 vols., The Hague: Mouton, 1976.

Putnam, Hilary (1982) Peirce the Logician, *Historia Mathematica* **9**:290-301, reprinted in Putnam (1990) pp. 252-260.

Roberts, Don D. (1973). *The Existential Graphs of Charles S. Peirce*. The Hague: Mouton.

Sowa, John F. (1984) *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, Reading, MA

Sowa, John F. (2000) *Knowledge Representation: Logical, Philosophical, and Computational Foundations*, Brooks/Cole Publishing Co., Pacific Grove, CA.

Sowa, John F. (2008) Conceptual graphs, in F. van Harmelen, V. Lifschitz, and B. Porter, eds., *Handbook of Knowledge Representation*, Amsterdam: Elsevier, pp. 213-237. http://jfsowa.com/cg/cg_hbook.pdf

Sowa, John F. (2009) Conceptual Graphs for Conceptual Structures, in P. Hitzler & H. Schärfe, eds., *Conceptual Structures in Practice*, Chapman & Hall/CRC Press, pp. 102-136. http://jfsowa.com/pubs/cg4cs.pdf

Sowa, John F. (2011) Peirce's tutorial on existential graphs, *Semiotica* **186:1-4**, 345-394. http://jfsowa.com/pubs/egtut.pdf

Sowa, John F. (2013) From existential graphs to conceptual graphs, *International Journal of Conceptual Structures* **1:1**, 39-72. http://jfsowa.com/pubs/eg2cg.pdf