Peirce's Tutorial on Existential Graphs

John F. Sowa

Abstract. In his formal papers on existential graphs, Peirce tended to obscure the simplicity of EGs with distracting digressions. In MS 514, however, he presented his simplest introduction to the EG syntax, semantics, and rules of inference. This article reproduces Peirce's original words and diagrams with further commentary, explanations, and examples. Unlike the syntax-based approach of most current textbooks, Peirce's method addresses the semantic issues of logic in a way that can be transferred to any notation. The concluding section shows that his rules of inference can clarify the foundations of proof theory and relate diverse methods, such as resolution and natural deduction. To relate EGs to other notations for logic, this article uses the Existential Graph Interchange Format (EGIF), which is a subset of the CGIF dialect of Common Logic. EGIF is a linear notation that can be mapped to and from the Alpha, Beta, and Gamma variants of EGs. It can also be translated to or from other formalisms, algebraic or geometrical.

This is a revised preprint of an article that appeared in *Semiotica*, **186:1-4**, 345-394.

1. Peirce's Method for Teaching Logic

Peirce's existential graphs (EGs) are the simplest, most elegant, and easiest-to-learn system of logic ever invented. Yet most logicians have never used them or even seen them. Part of the reason for their neglect is that the algebraic notation by Peirce (1880, 1885) with a change of symbols by Peano (1889) had already become the de facto standard for logic. Another reason is the complexity of Peirce's published explanations, which obscured the simplicity of the graphs with a mass of detail about important, but often distracting semiotic issues. In 1909, however, Peirce wrote Manuscript 514, which contains his clearest tutorial on existential graphs. In just 12 hand-written pages with 18 diagrams, he presented the syntax, rules of inference, and illustrative examples for first-order logic with equality. The full MS 514, however, wanders into digressions on subjects ranging from anomalies in the orbit of Mercury to a criticism of Laplace's theory of probability. This article extracts just the words and diagrams of the tutorial and adds more examples and commentary.

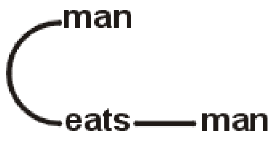
A remarkable feature of the tutorial is that Peirce does not follow the common practice of teaching the Boolean operators before introducing quantifiers. Unlike his more formal publications on EGs, this tutorial does not distinguish the Alpha graphs for propositional logic from the Beta graphs for predicate logic. Instead, Peirce returns to a notation he experimented with in 1882: the *relational graphs*, which combine the existential quantifier with an implicit conjunction. His first example, **—man**, represents the sentence "There is a man." The line **—**, which Peirce called a *line of identity*, represents an existential quantifier ($\exists x$), and the string "**man**" is the name of a monadic predicate asserted of x. With the addition of ovals to represent negation, these graphs become sufficiently general to represent full first-order logic with equality. In five pages, Peirce presents the syntax with examples; in another six pages, he presents sound and complete rules of inference illustrated with sample proofs; and in less than half a page, he adds some remarks about *endoporeutic*, which is a version of model theory that is logically equivalent to Tarski's.

Section 2 of this article reproduces Peirce's presentation of EG syntax, adds more examples, and for each diagram shows the equivalent formula in Peirce-Peano notation and in the Existential Graph Interchange Format (EGIF). Section 3 reproduces Peirce's rules of inference and illustrative proofs; the commentary adds further examples and shows that the rules are complete by using them to derive the more common rules of *modus ponens* and *universal instantiation*. Section 4 presents endoporeutic and uses it to prove that Peirce's rules are sound. Section 5 recommends Peirce's approach as a basis for teaching introductory courses on logic, and Section 6 presents theoretical principles that are simplified and clarified by EGs and Peirce's rules. The appendix presents the grammar of EGIF and discusses its relationship to the ISO/IEC standard 24707 for Common Logic and the proposed IKL extensions to Common Logic.

2. Syntax for First-Order Logic with Equality

In this article, Peirce's actual words and diagrams are presented in indented paragraphs; the commentary is not indented. Peirce numbered some of his diagrams, but not others. To avoid multiple numbering systems, Peirce's numbers are retained. Graphs that are not numbered are discussed only in the immediately preceding or following paragraphs. Peirce's graphs are nested in the indented paragraphs; all other graphs are part of the commentary. The tutorial begins at the bottom of page 10 of MS 514; the emphasis is by Peirce:

> I invented several different systems of signs to deal with relations. One of them is called the general algebra of relations, and another the algebra of dyadic relations. **I was finally led to prefer what I call a diagrammatic syntax**. It is a way of setting down on paper any assertion, however intricate, and if one so sets down any premises, and then (guided by 3 simple rules) makes **erasures** and **insertions**, he will read before his eyes a necessary conclusion from premises.
>
> This syntax is so simple that I will describe it. Every word makes an assertion. Thus, **—man** means "there is a man" in whatever universe the whole sheet offers it. The dash before "man" is the "line of identity."



> this means "Some man eats a man."

This graph has two lines of identity: the curve on the left, and the straight line on the right. In the algebraic notation, each line of identity corresponds to an existential quantifier, which Peirce represented by the Greek letter Σ. The graph could therefore be represented by the following formula in Peirce's notation of 1880 to 1885:

$$\Sigma_x \Sigma_y (\text{man}_x \bullet \text{man}_y \bullet \text{eats}_{x,y}).$$

Since Peano wanted to use logic for reasoning about mathematics, he replaced Peirce's symbols with symbols that could be freely mixed with mathematical formulas. Peano gave full credit to Peirce for the notation and declared Frege's notation unreadable. With Peano's symbols, the graph for "Some man eats a man" would be expressed by the formula

$$\exists x \exists y(\text{man}(x) \land \text{man}(y) \land \text{eats}(x,y)).$$

EGIF has a one-to-one mapping to and from each feature of the graph. Since the EG has two lines of identity and three relations, the corresponding EGIF has five components called *nodes*:

[*x] [*y] (man ?x) (man ?y) (eats ?x ?y).

The first two nodes, **[*x]** and **[*y]**, represent the two lines of identity. The character strings **x** and **y** are called *identifiers*. An identifier prefixed with an asterisk, such as ***x**, is called a *defining label*. An identifier prefixed with a question mark, such as **?x**, is called a *bound label* that is bound to the defining label with the same identifier. The three nodes in parentheses represent the three relations. Inside each relation node is the name of the relation type followed by a list of *bound labels*. The left-to-right order of the bound labels is the same as the order in which the corresponding lines of identity are attached to the relation names in the graph. A defining label and all bound labels with the same identifier are said to be *coreferent* because they refer to the same individual.

EGIF has no marker for conjunction, since there is an implicit conjunction of all nodes in the same area. Since a graph drawn on a two-dimensional sheet has no linear ordering, the only constraint on the ordering of the nodes is that a defining label must precede all coreferent bound labels. All permutations that preserve this constraint are logically equivalent. See the appendix for the full syntax of EGIF.

To deny that there is any phoenix, we shade that assertion which we deny as a whole:

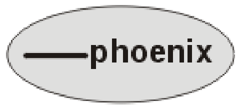


fig. 1

Thus what I have just scribed means "It is false that there is a phoenix." But the following:

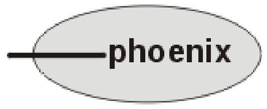


fig. 2

only means "There is something that is not identical with any phoenix."

To indicate negation in his original version of EGs, Peirce used an unshaded oval enclosure, which he called a *cut* or sometimes a *sep* because it separated the *sheet of assertion* into a positive (outer) area and a negative (inner) area. In MS 514, he added shading to highlight the distinction between positive and negative areas. Without the shaded oval, the graph **—phoenix** would assert that there exists a phoenix. In Figure 1, the entire graph is negated, but in Figure 2, part of the line is outside the negation. The following table shows the EGIF and formula for the unshaded graph, for Figure 1, and for for Figure 2:

Unshaded	**[*x] (phoenix ?x)**	$\exists x$ phoenix(x)
Figure 1	**~[[*x] (phoenix ?x)]**	$\sim\exists x$ phoenix(x)
Figure 2	**[*x] ~[(phoenix ?x)]**	$\exists x$ ~phoenix(x)

The EGIF for Figure 1 encloses the EGIF for the unshaded graph in square brackets and places the symbol **~** in front. The formula places the symbol ~ in front, and its implicit scope extends to the end of the formula. When a line crosses one or more negations, the defining label in EGIF or the corresponding existential quantifier in the algebraic formula is placed in the outermost area in which any part of the line occurs. Unlike Figure 1, which asserts the negation before the quantifier, Figure 2 asserts the quantifier before the negation.

When an oval is drawn inside another oval, the doubly nested area is positive (unshaded), as in Figure 3. Any area nested inside an odd number of ovals is shaded, and any area inside an even number of ovals (possibly zero) is unshaded.

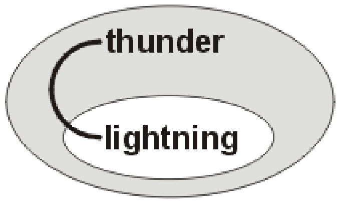


fig. 3

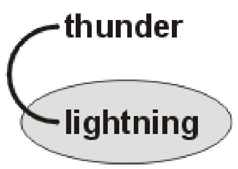


fig. 4

Fig. 3 denies fig. 4, which asserts that it thunders without lightning, for a denial shades the unshaded and unshades the shaded. Consequently fig. 3 means "If it thunders, it lightens."

The following table shows the EGIF and formulas for Figures 3 and 4:

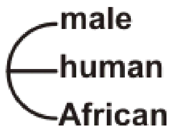
Figure 3	**~[[*x] (thunder ?x) ~[(lightning ?x)]]**	$\sim\exists x(\text{thunder}(x) \wedge \sim\text{lightning}(x))$
Figure 4	**[*x] (thunder ?x) ~[(lightning ?x)]**	$\exists x(\text{thunder}(x) \wedge \sim\text{lightning}(x))$

In the algebraic notation, $\sim\exists$ is equivalent to $\forall\sim$. With that conversion, the formula for Figure 3 becomes $\forall x\sim(\text{thunder}(x) \wedge \sim\text{lightning}(x))$. By Peirce's definition of the implication operator, $\sim(p \wedge \sim q)$ is equivalent to $p \supset q$. Therefore, the formula for Figure 3 can be rewritten as $\forall x(\text{thunder}(x) \supset \text{lightning}(x))$. This corresponds to Peirce's reading "If it thunders, it lightens." Peirce also said that a negated area can be read disjunctively. By De Morgan's law, the formula $\forall x\sim(\text{thunder}(x) \wedge \sim\text{lightning}(x))$, is converted to $\forall x(\sim\text{thunder}(x) \vee \text{lightning}(x))$, which may be read "For every x, either *x* does not thunder or *x* lightens."

For his algebraic notation, Peirce introduced the symbol Π for the universal quantifier and ⤙ for implication, but he did not use them in existential graphs. The Conceptual Graph Interchange Format (CGIF), which is a superset of EGIF defined by the ISO standard, does have symbols for those operators. But EGIF is limited to the symbols that Peirce actually used in existential graphs. The major reason why he never added the additional symbols to EGs is that they are unnecessary: the graphic notation of nested ovals, especially with shading, is already more readable than the algebraic notation even with special symbols. Furthermore, new symbols would have made his rules of inference more complicated.

> In order to make the lines of identity in their connexion with shading and its absence perfectly perspicuous, I must provide you with a bit or two of nomenclature. By an "**area**," then, I mean the whole of any continuous part of the surface on which graphs are scribed that is alike in all parts of it either shaded or unshaded. By a "**graph**" I mean **the way** in which a given assertion is scribed. It is the general kind not a single instance. For example there is in English but one single "word" that serves as definite article. It is the word "the." It will occur some twenty or more times on an average page; and when an editor asks for an article of so many thousand of "words" he means to count each of those instances as a distinct word. He speaks loosely of **instances** of words as words, which they are not.

The distinction between a graph as a general type and particular instances of it is important for the rules of inference, which are presented in Section 3 below.

> Now in like manner a **graph** is one thing, and a "**graph instance**" is another thing. Any expression of an assertion in this particular diagrammatic syntax is an **Existential Graph**, of which I use the single word "**graph**" as a common abbreviation as long as I have nothing to do with another kind of graph. A graph then may be complex or indivisible. Thus
>
> 
>
>
> is a graph instance composed of instances of three indivisible graphs which assert "there is a male" "there is something human" and "there is an African." The syntactic junction or point of teridentity asserts the identity of something denoted by all three.

A teridentity is represented by a *coreference node* **[?x ?y ?z]**, which shows that three lines of identity refer to the same individual. Following is a direct translation of the above graph to EGIF, an equivalent EGIF with a single defining node for all three lines, and the corresponding algebraic formula:

[*x] (male ?x) [*y] (human ?y) [*z] (African ?z) [?x ?y ?z]

[*x] (male ?x) (human ?x) (African ?x)

$\exists x(\text{male}(x) \wedge \text{human}(x) \wedge \text{African}(x))$

Peirce used the term *ligature* for a connection of two or more lines of identity. Each ligature can be represented by a coreference node in EGIF, and coreference nodes can often be simplified or eliminated by renaming labels, as in this example. Dau (2010) analyzed various examples of ligatures in Peirce's writings.

> Indivisible graphs usually carry "**pegs**" which are places on their periphery appropriated to denote, each of them, one of the subjects of the graph. A graph like "thunders" is called a "**medad**" as having no peg (though one might have made it mean "**some time** it thunders" when it would require a peg). A graph or graph instance having 0 peg is **medad**, having 1 peg is **monad**, having 2 pegs is **dyad**, or having 3 pegs is **triad**.

In modern terminology, Peirce's indivisible graphs are called *atoms*. In the algebraic notation, each atom consists of a single predicate or relation with its associated *arguments*, which Peirce called *logical subjects*. In EGs, each argument or logical subject of a relation is linked to its line of identity by a *peg*. A medad, which has no arguments, represents a proposition. (Peirce's term comes from the Greek μη for *not*.) In EGIF, a proposition represented by the symbol *p* is written as a relation with no bound labels: **(p)**. A monad represents a monadic predicate, which is also called a property; a dyad represents a dyadic predicate or a binary relation; and a triad represents a triadic predicate or a ternary relation.

By treating a medad as a special case of a predicate or relation, Peirce avoided the need to distinguish propositional logic from predicate logic. In other writings, he presented existential graphs in three parts: Alpha is the theory of propositional logic, which avoids any lines of identity; Beta introduces relations and lines of identity; and Gamma introduces modal and higher-order logic. In MS 514, Peirce combined Alpha and Beta in a unified presentation. Pedagogically, this approach is highly effective because it enables beginners to represent complete sentences with their earliest examples.

> Every indivisible graph instance must be wholly contained in a single area. The line of identity can be regarded as a graph composed of any number of dyads "**—is—**" or as a single dyad. But it must be wholly in one area. Yet it may abut upon another line of identity in another area.

To illustrate Peirce's point that a line of identity may be composed of any number of dyads, consider the graph **man—African**, which may be read "There is an African man." Replacing the dash with four copies of **—is—** would break the single line of identity into five separate segments:

man — is — is— is — is— African

Peirce's method of reading such graphs would produce the sentence "There is a man that is something that is something that is something that is something African." Each of the five segments corresponds to an existentially quantified variable, and each instance of the dyad **—is—** corresponds to an equal sign between two variables. Following is the EGIF and formula for the above EG:

[*x] [*y] [*z] [*u] [*v] (man ?x) (is ?x ?y) (is ?y ?z) (is ?z ?u) (is ?z ?v) (African ?v)

$\exists x \exists y \exists z \exists u \exists v\ (\text{man}(x) \land x{=}y \land y{=}z \land z{=}u \land u{=}v \land \text{African}(v))$

In EGs, equality is represented by joining lines of identity. In EGIF, a join of two lines is shown by a coreference node **[?x ?y]**, which corresponds to $x{=}y$. A coreference node may enclose any number of bound labels to show that all their lines of identity are joined:

[*x] [*y] [*z] [*u] [*v] (man ?x) [?x ?y ?z ?u ?v] (African ?v)

The rules presented in Section 3 would simplify this EGIF to **[*x] (man ?x) (African ?x)**.

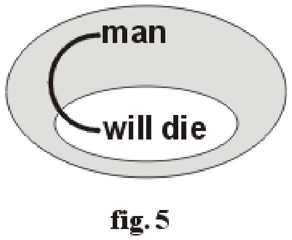


fig. 5

> Thus fig. 5 **denies** that there is a man that will not die, that is, it asserts that every man (if there be such an animal) will die. It contains two lines of identity.

Peirce considered the line of identity in the shaded area to be distinct from the line in the unshaded area. To represent that interpretation, the following EGIF has a defining label ***x** for the part in the shaded area, another defining label ***y** for the part in the unshaded area, and a coreference node **[?x ?y]** for the connection at the boundary:

~[[*x] (man ?x) ~[[*y] [?x ?y] (will_die ?y)]]

Peirce sometimes included blanks in the names of relations, but those blanks, which are not permitted in EGIF, can be replaced by underscores. Since the pure graph notation has no labels, there is never a need to relabel lines of identity. But in linear notations, the coreference node **[?x ?y]** or the corresponding equality would permit the identifier **y** to be replaced by **x**:

~[[*x] (man ?x) ~[[?x] [?x ?x] (will_die ?x)]]

The rules presented in Section 3 would then delete redundant copies of **?x** and **[?x]**:

~[[*x] (man ?x) ~[(will_die ?x)]]

$\sim\exists x(\text{man}(x) \wedge \sim\text{will\_die}(x))$.

Replacing $\sim\exists$ with $\forall\sim$ and converting the body of the formula to an implication produces the formula $\forall x(\text{man}(x) \supset \text{will\_die}(x))$, which may be read "For every *x*, if *x* is a man, then *x* will die."

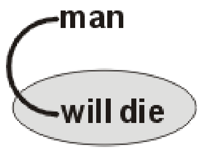


fig. 6

[Fig. 5] denies which fig. 6 asserts, "there is a man that is something that is something that is not anything that is anything unless it be something that will not die." I state the meaning in this way, to show how the identity is continuous regardless of shading; and this is **necessarily** the case. In the nature of identity that is its entire meaning. For the shading denies the **whole** of what is in its area but not each part except disjunctively.

For an introduction to logic, this explanation is more confusing than enlightening. The simpler reading of Figure 6 is "There is a man who will not die." For Peirce's disjunctive reading, the graph in the shaded area must be viewed as a conjunction of two parts. To derive his reading, two copies of the dyad **—is—** or the coreference **[?x ?y]** must be added outside the negation and two more copies inside the negation:

[*x] [*y] [*z] (man ?x) (is ?x ?y) (is ?y ?z) ~[[*u] [*v] (is ?z ?u) (is ?u ?v) (will_die ?v)]]

Before converting this EGIF to a formula, change the last copy of **?v** to the coreferent label **?z** and use parentheses to enclose the two equalities inside the negation:

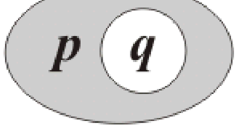
$\exists x\exists y\exists z(\text{man}(x) \wedge x{=}y \wedge y{=}z \wedge \sim(\exists u\exists v((z{=}u \wedge u{=}v) \wedge \text{will\_die}(z))))$.

Then use De Morgan's law to convert the negated conjunction to a disjunction:

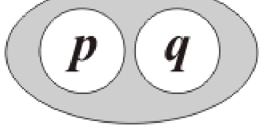
$\exists x\exists y\exists z(\text{man}(x) \wedge x{=}y \wedge y{=}z \wedge (\sim\exists u\exists v(z{=}u \wedge u{=}v) \vee \sim\text{will\_die}(z)))$.

This formula may be read "There is a man *x* that is something *y* that is something *z* that is not anything *u* that is anything *v*, or *z* is something that will not die." With practice, one can learn to "see" the disjunctive pattern without doing the algebra. Graphs enable the viewer to interpret a spatial configuration in different ways.

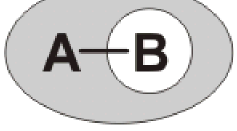
This concludes Peirce's presentation of the EG syntax. As he showed, it has only two explicit operators: a line to represent the existential quantifier and an oval to represent negation. Conjunction is an implicit operator, expressed by drawing any number of graphs in the same area. Equality is shown by joining lines. All other operators of first-order logic are represented by combining these primitives. The following diagram shows three common combinations:



If *p* then *q*.



p or *q*.

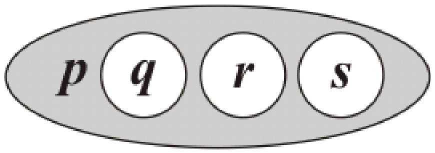


Every A is B.

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 emphasizes the effect on scope of quantifiers caused by the operators ⊃ and ∨. This effect is difficult to explain to students because the algebraic notation makes ∨ and ∧ look symmetrical. Following are the EGIF and algebraic representations. Note that the medads *p* and *q* are represented as relations with no bound labels: **(p)** and **(q)**.

If *p*, then *q*.	**~[(p) ~[(q)]]**	$p \supset q$
p or *q*.	**~[~[(p)] ~[(q)]]**	$p \lor q$
Every A is B.	**~[[*x] (A ?x) ~[(B ?x)]]**	$(\forall x)(\mathrm{A}(x) \supset \mathrm{B}(x))$

With experience, anyone who uses EGs begins to recognize many other common patterns, such as the following combination of implication and disjunction:



If *p* then *q* or *r* or *s*.

The EG has no implicit ordering of subgraphs, but some ordering is imposed by any linear notation. The following EGIF is just one of 24 equivalent permutations for representing the above EG.

~[(p) ~[(q)] ~[(r)] ~[(s)]]

Even when the four subgraphs are written in the same order, the choice of Boolean operators enables the EG to be read in many different ways as an English sentence or an algebraic formula. Following are three examples of an English reading and the corresponding algebraic notation:

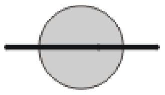
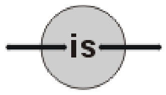
If *p*, then *q* or *r* or *s* — $p \supset (q \lor r \lor s)$.

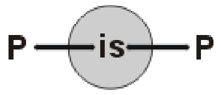
If *p* and not *q*, then *r* or *s* — $(p \land \sim q) \supset (r \lor s)$.

If *p* and not *q* and not *r*, then *s* — $(p \land \sim q \land \sim r) \supset s$.

Each of these formulas has more permutations for each choice in mapping the EG to EGIF. For that reason, EGs are a good candidate for a canonical form that can reduce the multiple variations caused by different choices of Boolean operators or the order of writing them. This reduction of equivalent variations can be especially useful for reducing the amount of search in programs for theorem proving.

Reading a line as a concatenation of dyads of the form **—is—** can sometimes clarify the translation of an EG to a sentence or formula. As examples, the next three graphs show some ways of saying that there exist two things. In the graph on the left, the shaded area negates the connection between the lines of identity on either side. To emphasize what is being negated, the graph in the middle replaces part of the line with the dyad **—is—**. Therefore, that graph may be read "There is something *x*, which 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."

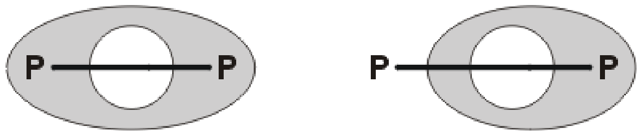





The following table shows the EGIF and formulas for these three graphs.

Left graph	**[*x] [*y] ~[[?x ?y]]**	∃*x*∃*y*~(*x*=*y*)
Middle graph	**[*x] [*y] ~[(is ?x ?y)]**	∃*x*∃*y*~is(*x*,*y*)
Right graph	**[*x] [*y] (P ?x) (P ?y) ~[is(?x ?y)]**	∃*x*∃*y*(P(*x*) ∧ P(*y*) ∧ ~is(*x*,*y*))

As these examples show, an oval with a line through it can be read as negated equality. It is equivalent to the symbol ≠ in the algebraic notation, but it is so readable that there is no need for a special symbol. With a nest of two ovals, the graph on the left below denies that there are two Ps. The graph on the right asserts that there is exactly one P.



For the graph on the right, the outer part says that there is a P, and the shaded part denies that there is another P different from the first. Therefore, there must be exactly one P. Following is the EGIF:

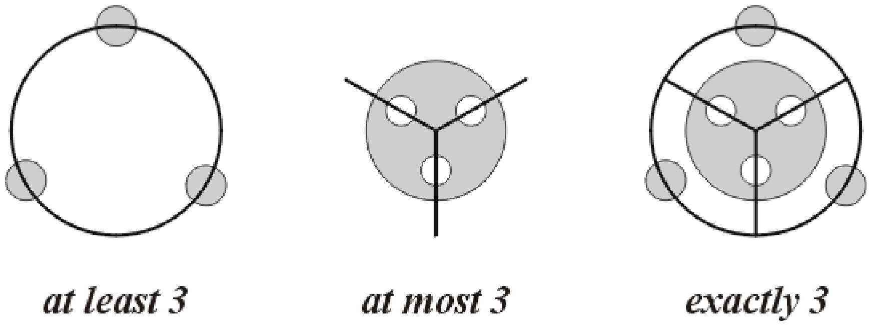
[*x] (P ?x) ~[[*y] (P ?y) ~[[?x ?y]]]

The direct translation of the EGIF to an English sentence or an algebraic formula would use two negations. But the double negation could also be read as an implication. Following are both translations:

"There is a P, and there is no other P" — ∃*x*(P(*x*) ∧ ~∃*y*(P(*y*) ∧ *x*≠*y*)).

"There's a P, and if there's any P, it's the same as the first" — ∃*x*(P(*x*) ∧ ∀*y*(P(*y*) ⊃ *x*=*y*)).

These examples can be extended with multiple lines of identity and negated equalities. The graph on the left below says there exist at least three things. The graph in the middle says there exist at most three things. The graph on the right, which combines the previous two, says there exist exactly three things:



The graph on the left has three lines of identity, which consist of the three arcs of the circle that are not enclosed in any shaded area. Each of those three lines, which may be labeled ***x**, ***y**, and ***z**, is continued into the adjacent shaded areas, which show that it is not coreferent with its neighboring lines. The graph in the middle also has three lines of identity outside the shaded area. The point in the center of the shaded area, which may be labeled ***w**, is called a teridentity because it has three branches. The graph on the right is the conjunction of the other two. The following table shows the translations of the above graphs to English, EGIF, and formulas in predicate calculus.

English	EGIF	Formulas
at least 3	**[*x] [*y] [*z] ~[[?x ?y]] ~[[?y ?z]] ~[[?z ?x]]**	$\exists x \exists y \exists z\ (x \neq y \land y \neq z \land z \neq x)$
at most 3	**[*x] [*y] [*z] ~[[*w] ~[[?w ?x]] ~[[?w ?y]] ~[[?w ?z]]]**	$\exists x \exists y \exists z {\sim} \exists w\ (w \neq y \land w \neq y \land w \neq z)$
exactly 3	**[*x] [*y] [*z] ~[[?x ?y]] ~[[?y ?z]] ~[[?z ?x]] ~[[*w] ~[[?w ?x]] ~[[?w ?y]] ~[[?w ?z]]]**	$\exists x \exists y \exists z (x \neq y \land y \neq z \land z \neq x \land {\sim} \exists w (w \neq y \land w \neq y \land w \neq z))$

Note that the graph in the middle could also be read as an implication whose conclusion is a disjunction: "There is an *x*, a *y*, and a *z*, and if there is a *w*, then either *w* is *x*, *w* is *y*, or *w* is *z*." To represent more things, EGs could be generalized to three dimensions with wires for lines of identity and bubbles for negation. As an exercise, generalize the above EGs to four things: represent each thing by a vertex of a tetrahedron; place a small shaded ball for ≠ on each of the six edges; place a large shaded ball inside the tetrahedron; place a dot to represent the nonexistent fifth thing at the center of the large ball; connect the center dot by four wires to each of the vertices; on each of those wires (but inside the large ball) place a small unshaded ball.

In the linear notations, the labels for variables or lines obscure the symmetry. The graphic form shows identity by joining two lines, but the linear versions require special notations, such as $x{=}y$ or **[?x ?y]**. Those notations then require special axioms, such as reflexivity, symmetry, and transitivity. Pure graphs have no labels, no axioms for equality, no rules for substituting values for variables, and no rules for relabeling variables. Those axioms and rules are artifacts of the notation.

3. Rules of Inference for FOL

All proofs in Peirce's system are based on "permissions" or "formal rules... by which one graph may be transformed into another without danger of passing from truth to falsity and without referring to any interpretation of the graphs" (CP 4.423). Peirce presented the permissions as three pairs of rules, one of which states conditions for inserting a graph, and the other states the inverse conditions for erasing a graph. In this commentary, the insertion rules are numbered 1i, 2i, 3i, and the inverse erasure rules are 1e, 2e, 3e.

> There are three simple rules for modifying premises when they have once been scribed in order to get any sound necessary conclusion from them.... I will now state what modifications are permissible in any graph we may have scribed.

Peirce's rules are a generalization and simplification of the rules for *natural deduction*, which Gentzen (1934) independently discovered many years later. For both Peirce and Gentzen, the rules are grouped in pairs, one of which inserts an operator, which the other erases. For both of them, the only axiom is a blank sheet of paper: anything that can be proved without any prior assumptions is a theorem. Section 6 presents a more detailed comparison with Gentzen's method.

> **1st Permission**. Any graph-instance on an unshaded area may be erased; and on a shaded area that already exists, any graph-instance may be inserted. This includes the right to cut any line of identity on an unshaded area, and to prolong one or join two on a shaded area. (The shading itself must not be erased of course, because it is not a graph-instance.)

The proof of soundness depends on the fact that erasing a graph reduces the number of options that might be false, and inserting a graph increases the number of options that might be false. Rule 1e, which permits erasures in an unshaded (positive) area, cannot make a true statement false; therefore, that area must be at least as true as it was before. Conversely, Rule 1i, which permits insertions in a shaded (negative), area cannot make a false statement true; therefore, the negation of that false area must be at least as true as it was before. A formal proof of soundness requires a version of model theory. Section 4 uses Peirce's model-theoretic semantics, which he called *endoporeutic* for "outside-in evaluation."

These rules apply equally well to propositional logic and predicate logic. Since EGs have no named variables, the algebraic rules for dealing with variables are replaced by rules for cutting or joining lines of identity (which correspond to erasing or inserting an equality or the graph **—is—**). Cutting a line in a positive area has the effect of *existential generalization*, because the newly separated ends of the line represent different existentially quantified variables. Joining two lines in a negative area has the effect of *universal instantiation*, because it replaces a universally quantified variable with an arbitrary term.

> **2nd Permission**. Any graph-instance may be **iterated** (i.e. duplicated) in the same area or in any area enclosed within that, provided the new lines of identity so introduced have identically the same connexions they had before the iteration. And if any graph-instance is already duplicated in the same area or in two areas one of which is included (whether immediately or not) within the other, their connexions being identical, then the inner of the instances (or either of them if they are in the same area) may be erased. This is called the Rule of Iteration and Deiteration.

In other writings, Peirce presented more detail about applying these rules to lines of identity. Iteration (2i) extends a line from the outside inward: any line of identity may be extended in the same area or into any enclosed area. Deiteration (2e) retracts a line from the inside outward: any line of identity that is not attached to anything may be erased, starting from the innermost area in which it occurs. Peirce also said that no graph may be copied into any area within itself; it is permissible, however, to copy a graph and then make a copy of the new graph in some area of the original graph.

The proof of soundness of iteration (2i) and deiteration (2e) shows that they are equivalence relations: they can never change the truth value of a graph. First, note that a copy of a graph p in the same area is equivalent to the conjunction $p \wedge p$; inserting a copy of p by Rule 2i or erasing it by 2e cannot change the truth value. For a copy of a subgraph into a nested area, the proof in Section 4 shows that the subgraph makes its contribution to the truth value of the whole graph at its first occurrence. The presence or absence of a more deeply nested copy is irrelevant.

> **3rd Permission**. Any ring-shaped area which is entirely vacant may be suppressed by extending the areas within and without it so that they form one. And a vacant ring shaped area may be created in any area by shading or by obliterating shading so as to separate two parts of any area by the new ring shaped area.

A vacant ring-shaped area corresponds to a *double negation*: two negations with nothing between them. The third permission says that a double negation may be erased (3e) or inserted (3i) around any graph on any area, shaded or unshaded. Note that Peirce considered the *empty graph*, represented by a blank sheet of paper, as a valid existential graph; therefore, a double negation may be drawn or erased around a blank. An important qualification, which Peirce discussed elsewhere, is that such a ring is considered vacant even if it contains lines of identity, provided that the lines begin outside the ring and continue to the area enclosed by the ring without having any connections to one another or to anything else in the area of the ring. Both rules 3e and 3i are equivalence rules, as the method of endoporeutic shows in Section 4.

> It is evident that neither of these three principles will ever permit one to assert more than he has already asserted. I will give examples the consideration of which will suffice to convince you of this.

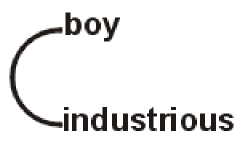


fig. 7

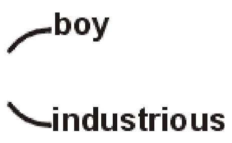


fig. 8

> Fig. 7 asserts that some boy is industrious. By the 1st permission it can be changed to fig. 8, which asserts that there is a boy and that there is an industrious person. This was asserted as fig. 7, together with the identity of some case.

Erasing a graph or subgraph usually simplifies a statement, but erasing part of a line of identity replaces one line of identity with two. To emphasize what is being erased, the EGIF or formula that corresponds to Figure 7 can be written as if it contained the dyad **—is—**, the coreference **[?x ?y]**, or the equation x=y. The following table shows the EGIF or the formula for Figure 7, an intermediate form with the coreference or the equation, and the EGIF or formula for Figure 8.

Figure 7	**[*x] (boy ?x) (industrious ?y)**	$\exists x(\text{boy}(x) \wedge \text{industrious}(x))$
Intermediate	**[*x] [*y] (boy ?x) (industrious ?y) [?x ?y]**	$\exists x \exists y(\text{boy}(x) \wedge \text{industrious}(y) \wedge x{=}y)$
Figure 8	**[*x] [*y] (boy ?x) (industrious ?y)**	$\exists x \exists y(\text{boy}(x) \wedge \text{industrious}(y))$

Rule 1e, which allows any graph to be erased in a positive area, has the effect of erasing the equation from the intermediate form to derive the EGIF and formula for Figure 8. The result leaves open the question whether the boy and the industrious person are the same or different.

> Fig. 9 asserts either there is nothing known for certain or else there is no communication with anybody. By the same permission this can be changed to fig. 10 which asserts that no communication with anybody deceased is known for certain. But this is fully included in the state of things asserted in fig. 9.

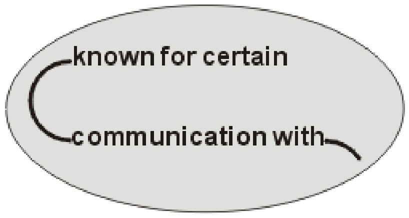


fig. 9

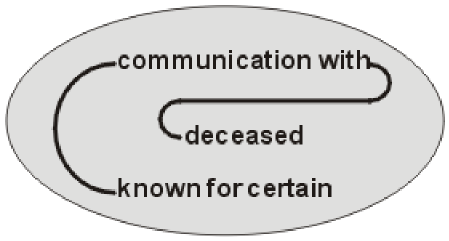


fig. 10

The simplest reading of Figure 9 is to append "It is false that" in front of the reading of the positive graph: "It is false that some x is known for certain that is a communication with some y." A more natural English reading would translate a negated line of identity as "nothing" or "no," but then the second line of identity must be read as "any" in order to avoid a double negation. That would lead to the reading "No communication with anybody is known for certain." The same procedure would lead to Peirce's reading for Figure 10: "No communication with anybody deceased is known for certain."

Peirce's reading for Figure 9 is another disjunctive statement, which is more confusing than enlightening in a tutorial for beginners. To derive that reading, start with the direct translation of Figure 9 to an algebraic formula:

$$\sim\exists x \exists y(\text{knownForCertain}(x) \wedge \text{communicationWith}(x,y))$$

Then move the negation inward to convert the existential quantifiers to universals and apply De Morgan's laws to convert a negated conjunction to a disjunction of negated relations:

$$\forall x \forall y(\sim\text{knownForCertain}(x) \vee \sim\text{communicationWith}(x,y))$$

Literally, this formula says "For every x and y, either x is not known for certain, or x is not a communication with y." Such a complex reading is distracting in an example intended to illustrate rules of inference.

In another passage of MS 514, Peirce said that the purpose of his rules is "to dissect the reasoning into the greatest possible number of distinct steps and so to force attention to every requisite of the reasoning." To illustrate that point, he proved the syllogism named Barbara as a derived rule of inference. He starts with the premises "Any M is P" and "Any S is M" in Figure 11 and concludes "Any S is P" in Figure 16.

I will now, by way of an example of the way of working with this syntax, show how by successive steps of inference to pass from the premises of a simple syllogism to its conclusion.

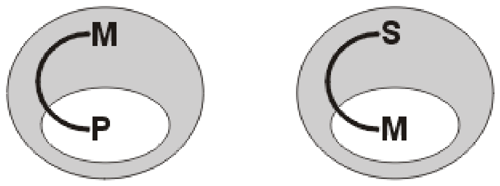


fig. 11

The first step consists in passing to fig. 12 by the 2nd Permission [which allows the graph on the left of fig. 11 to be iterated (copied) into the graph on the right].

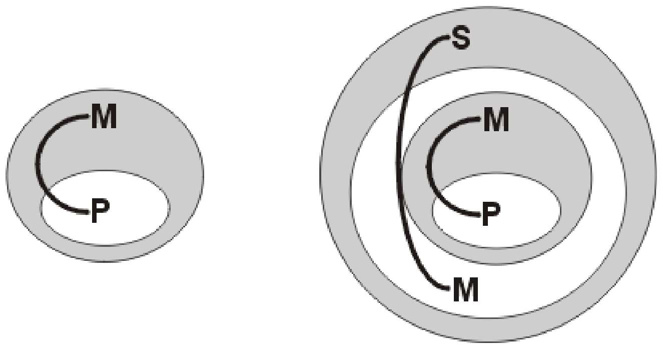


fig. 12

The second step is simply to erase "Any M is P" by the 1st Permission. The third step is to join the two ligatures by the 1st Permission as shown in fig. 13:

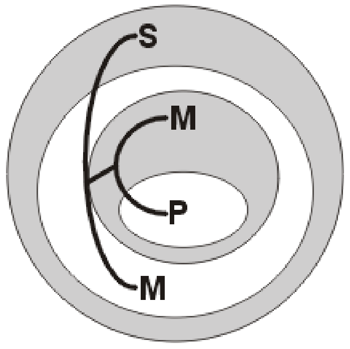


fig. 13

It will be observed that in iterating the major premise, I had a right to put the new graph instance at any part of the area into which I put it; and I took care to have the ligature of the minor premise **touch** the shaded area of iterated graph instance. Now by the 1st Permission I have a right to insert what I please into a shaded area, and without making the new line of junction leave the shaded area, I make it touch the unshaded line of identity of the major premise.

Peirce's explanations of lines that touch a boundary are sometimes confusing. If he had not made the boundary of the iterated graph instance touch the line of identity, the join would take two steps: an extension of the outer line into the shaded area by Rule 2i, and a join of the two lines by Rule 1i.

Before the two lines were joined, the inner copy of M could not be erased by deiteration because the two copies of M were attached to different ligatures. But after the join, both copies of M are attached to the same ligature, and the inner copy of M can be erased by Rule 2e.

This gives me a right in the fourth step to deiterate M so as to give fig. 14 by the second permission.

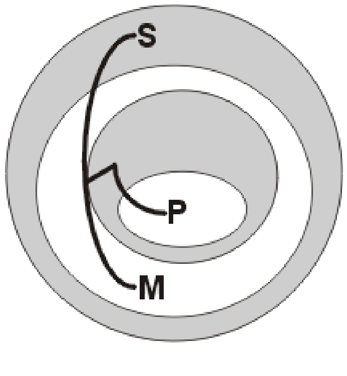


fig. 14

The fifth step is to delete the M on an unshaded field giving fig. 15

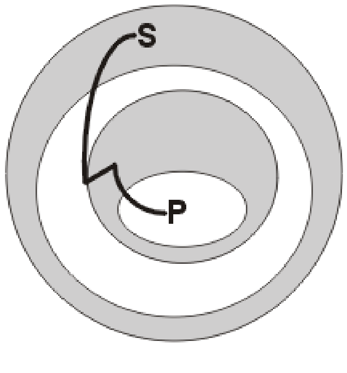


fig. 15

while the Sixth step authorized by permission the third consists in getting rid of the empty ring shaped shaded area round the P, giving fig. 16.

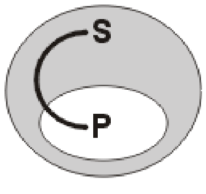


fig. 16

In the EGIF version of Peirce's proof, the major differences result from using labels instead of lines of identity. The lines that cross borders between areas raise questions that Peirce answered in different ways in different writings. The labels in EGIF simplify the issues about lines crossing borders, but they add complications when lines of identity are joined. Following are the EGIF statements for the starting graphs in Figure 11 and the six steps that produce Figure 16:

0. Starting graphs: **~[[*x] (M ?x) ~[(P ?x)]]   ~[[*y] (S ?y) ~[(M ?y)]]**
1. By rule 2i, iterate (copy) the graph on the left into the innermost area of the graph on the right:
   ~[[*x] (M ?x) ~[(P ?x)]]   ~[[*y] (S ?y) ~[(M ?y) ~[[*x] (M ?x) ~[(P ?x)]]]]
2. By rule 1e, erase the graph on the left:
   ~[[*y] (S ?y) ~[(M ?y) ~[[*x] (M ?x) ~[(P ?x)]]]]
3. By rule 1i, join the line of identity with the label **y** to the line with the label **x**:
   ~[[*y] (S ?y) ~[(M ?y) ~[[?y] (M ?y) ~[(P ?y)]]]]

   Note that the join causes the defining label ***x** and its bound labels **?x** to become **?y**. This operation is the equivalent of substituting one variable for another in the algebraic notation.
4. Since there are two copies of **(M ?y)**, the more deeply nested copy may be deiterated (erased) by rule 2e. The node **[?y]**, which is more deeply nested than the defining node **[*y]**, can also be deiterated:
   ~[[*y] (S ?y) ~[(M ?y) ~[~[(P ?y)]]]]

5. By rule 1e, erase the node **(M ?y)**, which is in a positive area nested two levels deep:
   ~[[*y] (S ?y) ~[~[~[(P ?y)]]]]
6. By rule 3e, erase one of the double negations. The result is the EGIF for Figure 16:
   ~[[*y] (S ?y) ~[(P ?y)]]

Peirce's rules are fundamentally semantic: each one inserts or erases one or more meaningful units. For propositional logic (Alpha graphs), those units are relations (medads) and negations. First-order logic with equality (Beta graphs) adds lines of identity and connections between lines. In EGIF, those four kinds of meaningful units are expressed by *nodes*: relations, negations, defining nodes, and coreference nodes. Peirce treated functions as a special case of relations, and he used the same notation for both. The EGIF grammar presented in the appendix adds nodes called *functions* to support the full semantics of Common Logic. The rules for propositional logic are a subset of the FOL rules because they do not mention lines of identity:

1. (i) In a negative (shaded) area, one or more nodes may be inserted.

   (e) In a positive (unshaded) area, one or more nodes may be erased.
2. (i) One or more nodes in any area *a* may be iterated (copied) in the same area *a* or into any area nested in *a*.

   (e) Any node that could have been derived by rule 2i may be erased. (Whether or not a node had previously been derived by 2i is irrelevant.)
3. (i) A double negation may be drawn around any collection of zero or more nodes in any area.

   (e) Any double negation in any area may be erased.

Since EGIF does not use shading, a positive area is defined by an even number of negations, and a negative area by an odd number of negations. The conditions for first-order logic with equality include all the above conditions with further constraints and operations on the labels for lines of identity. Before stating those conditions, some definitions are necessary: the *scope* of a defining label; the EGIF equivalents for Peirce's verbs *prolong*, *join*, and *cut*; and the verb *simplify* when applied to coreference nodes.

- If a defining node *d* occurs in an area *a*, the *scope* of *d* and its defining label is the area *a* and any area directly or indirectly enclosed by any negation in *a*. Any bound label in the scope of *d* that has the same identifier as *d* is said to be *bound* to *d* and to the defining label of *d*. In EGIF, the node *d* must precede (occur to the left of) all nodes that directly or indirectly contain labels bound to *d*.
- To *prolong* a line of identity with a defining node *d* into any area *a* in the scope of *d* is to insert a coreference node in *a* that has a single bound label that is bound to *d*.
- To *join* two lines of identity with defining nodes *d* and *e* in any area *a* that is in the scope of both *d* and *e* is to insert a coreference node in *a* that contains bound labels for *d* and *e* and no others.
- To *cut* a line of identity with defining node *d* in an area *a* in the scope of *d* is to insert a defining node *e* whose label is distinct from all other labels whose scope includes *a* and to replace some or all of the labels bound to *d* that are in the scope of *e* with labels bound to *e*.
- To *simplify* the coreference nodes in an area *a* is to perform the following operations as often as they are applicable:
  1. If a coreference node in *a* contains two or more bound labels with the same identifier, erase all but one of them.
  2. If two coreference nodes in *a* each contain a bound label with the same identifier, they are *merged*: erase one of the coreference nodes, move all its bound labels to the other, and remove any duplicates according to operation 1.
  3. If a defining node *d* in *a* has a bound label in a coreference node *c* in *a* and *c* also contains a bound label *b* that is not bound to *d*, then the defining node *d* is erased, the label in *c* that was bound to *d* is erased, and every other label that was bound to *d* is replaced with a copy of *b*.

For an example of simplification, see the discussion of the EGs for an African man in the paragraph just before Figure 5. For an example of a join followed by a simplification, see Peirce's conversion of Figure 12 to Figure 13 and its translation to EGIF in step 3 of the proof following Figure 16. Any coreference node that contains a single bound label may be erased by the rule of deiteration, because it is a copy of a defining node. That erasure, which is performed by a rule of inference, is not considered one of the simplification rules.

The basic constraint on erasing or inserting nodes is that no operation may leave or insert a bound label outside the scope of its defining label. Following are further conditions for Peirce's first and second permissions; no more conditions are needed for the third permission:

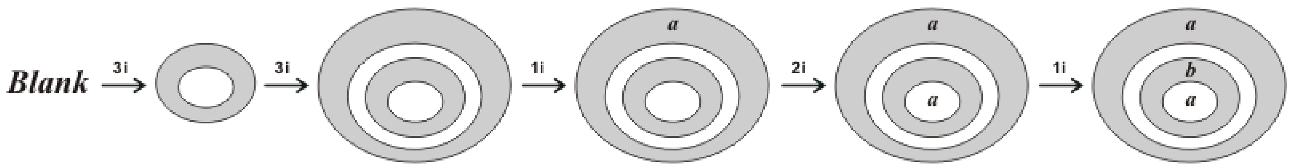
1. (i) In a negative area, no node that contains a bound label may be inserted in an area that is not in the scope of its defining label. No defining node may be inserted in an area that is in the scope of another defining label that has the same identifier.

   (e) In a positive area, no defining node that has one or more bound labels may be erased, unless all the nodes that contain those bound labels are erased in the same operation.

2. (i) If a defining node is iterated, the copy must be converted to a coreference node that contains a single bound label with the same identifier. A defining node that is enclosed in a negation, however, may remain unchanged when the negation is copied; but to avoid possible conflicts with future operations, the identifier of that defining label and all its bound labels should be replaced with an identifier that is not otherwise used.

   (e) Any nodes that could have been derived by rule 2i may be erased. (Whether or not a node had previously been derived by 2i is irrelevant.)

Peirce's meticulous attention to the smallest steps of reasoning enabled him to prove the soundness of every rule. Fr ege (1879) assumed nine unprovable and nonobvious axioms. Whitehead and Russell (1910) assumed five axiom schemata, of which one was redundant, but nobody discovered the redundancy until 1926. For EGs, only one axiom is necessary: a blank sheet of assertion, from which all the axioms and rules of inference by Frege, Whitehead, and Russell can be proved by Peirce's rules. As an example, Frege's first axiom, $a \supset (b \supset a)$, can be proved in five steps by Peirce's rules.

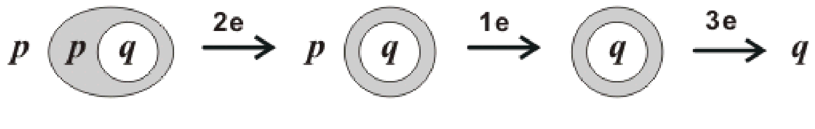


In EGIF, the propositions *a* and *b* are represented as relations with zero arguments: **(a)** and **(b)**. Following are the five steps of the proof:

1. By rule 3i, Insert a double negation around the blank: **~[~[ ]]**
2. By 3i, insert a double negation around the previous one: **~[~[~[~[ ]]]]**
3. By 1i, insert **(a)**: **~[(a) ~[~[~[ ]]]]**
4. By 2i, iterate **(a)**: **~[(a) ~[~[~[(a)]]]]**
5. By 1i, insert **(b)**: **~[(a) ~[~[(b) ~[(a)]]]]**

The theorem to be proved contains five symbols, and each step of the proof inserts one symbol into its proper place in the final result.

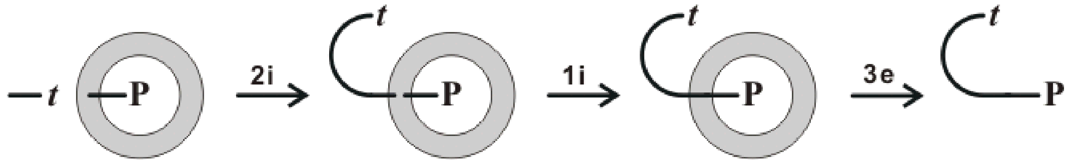
Frege's two rules of inference were *modus ponens* and *universal instantiation*. Following is a proof of *modus ponens*, which derives *q* from a statement *p* and an implication $p \supset q$:



Proof in EGIF:

0. Starting graphs: **(p) ~[(p) ~[(q)]]**
1. By 2e, erase the nested copy of **(p)**: **(p) ~[~[(q)]]**
2. By 1e, erase **(p)**: **~[~[(q)]]**
3. By 3e, erase the double negation: **(q)**

The rule of *universal instantiation* allows any term *t* to be substituted for a universally quantified variable in a statement of the form (∀*x*)P(*x*) to derive P(*t*). In EGs, the term *t* would be represented by a graph of the form **—t**, in which **t** represents some monadic relation defined by a graph attached to the line of identity. The universal quantifier ∀ corresponds to ~∃~, which is represented by a line whose outermost part occurs in a negative area. Since a graph has no variables, there is no notion of substitution. Instead, the following proof performs the equivalent operation by connecting the two lines.



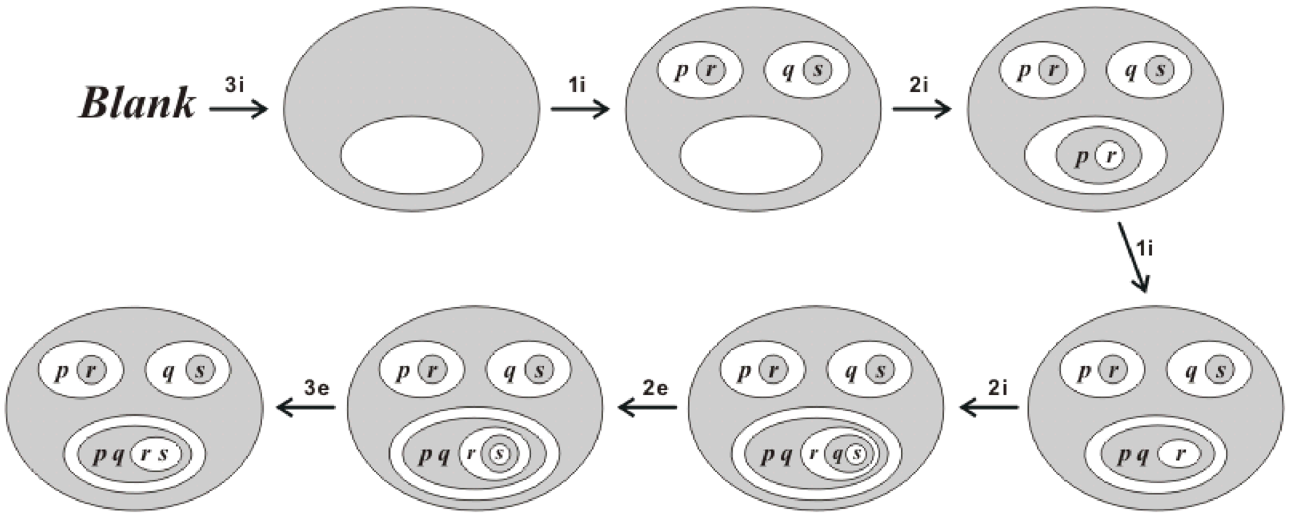
The proof in EGIF is more complex because of the need to relabel identifiers:

0. Starting graphs: **[*y] (t ?y) ~[[*x] ~[(P ?x)]]**
1. By 2i, iterate **[*y]** and change the defining label ***y** to a bound label **?y** in the copy:
   [*y] (t ?y) ~[[?y] [*x] ~[(P ?x)]]
2. By 1i, insert the coreference **[?x ?y]**:
   [*y] (t ?y) ~[[?y] [*x] [?x ?y] ~[(P ?x)]]
3. Relabel ***x** and **?x** to **?y**, simplify **[?y ?y]** to **[?y]**, and deiterate copies of **[?y]**:
   [*y] (t ?y) ~[~[(P ?y)]]
4. By 3e, erase the double negation: **[*y] (t ?y) (P ?y)**

In the *Principia Mathematica*, Whitehead and Russell proved the following theorem, which Leibniz called the *Praeclarum Theorema* (Splendid Theorem). It is one of the last and most complex theorems in propositional logic in the *Principia*, and the proof required a total of 43 steps:

$$((p \supset r) \land (q \supset s)) \supset ((p \land q) \supset (r \land s))$$

With Peirce's rules, this theorem can be proved in just seven steps starting with a blank sheet of paper. Each step inserts or erases one graph, and the final graph is the statement of the theorem.



After only four steps, the graph looks almost like the desired conclusion, except for a missing copy of *s* in the innermost area. Since that area is positive, the only way to get *s* in there is by iterating some graph that contains *s* and erasing the parts that are not needed. Following is the EGIF version of the proof:

1. By 3i, draw a double negation around the blank: **~[~[ ]]**
2. By 1i, insert the hypothesis in the negative area:
   ~[~[(p) ~[(r)]] ~[(q) ~[(s)]] ~[ ]]
3. By 2i, iterate the left part of the hypothesis into the conclusion:
   ~[~[(p) ~[(r)]] ~[(q) ~[(s)]] ~[~[(p) ~[(r)]]]]
4. By 1i, insert **(q)**:
   ~[~[(p) ~[(r)]] ~[(q) ~[(s)]] ~[~[(p) (q) ~[(r)]]]]
5. By 2i, iterate the right part of the hypothesis into the innermost area:
   ~[~[(p) ~[(r)]] ~[(q) ~[(s)]] ~[~[(p) (q) ~[(r)] ~[(q) ~[(s)]]]]]
6. By 2e, deiterate **(q)**:
   ~[~[(p) ~[(r)]] ~[(q) ~[(s)]] ~[~[(p) (q) ~[(r)] ~[~[(s)]]]]]
7. By 3e, erase the double negation to generate the conclusion:
   ~[~[(p) ~[(r)]] ~[(q) ~[(s)]] ~[~[(p) (q) ~[(r)] (s)]]]

This proof illustrates the usual pattern for deriving a theorem by Peirce's rules. The first step draws a double negation around the blank. The second step inserts the hypothesis into the negative area. The remaining steps copy parts of the hypothesis into the conclusion, or they perform other operations on the parts that have been copied. When people prove theorems, they look ahead to the desired conclusion and try to make the current graph look as similar to the conclusion as possible. With a method for measuring graph similarity, that heuristic can also be implemented in computer systems.

4. Peirce's Model Theory

In his first publication on model theory, Tarski (1933) quoted Aristotle as a precedent. The medieval Scholastics developed Aristotle's insights further, and Ockham (1323) presented a model-theoretic analysis of Latin semantics. Although Ockham wasn't as formal as Tarski, he covered the Latin equivalents of the Boolean connectives, the existential quantifier (*aliquis*), the universal quantifier (*omnis*), and even modal, temporal, and causal terms. Peirce owned a copy of Ockham's book and lectured on it at Harvard. He used model-theoretic arguments to justify the rules of inference for all his versions of logic, including truth tables for Boolean logic. But his most explicit development of model theory was the method of *endoporeutic*, which is an "outside-in" method for determining the truth value of an existential graph. In the following excerpt from MS 514, Peirce mentions endoporeutic:

> The rule of interpretation which necessarily follows from the diagrammatization is that the interpretation is "endoporeutic" (or proceeds inwardly) that is to say a ligature denotes "something" or "anything not" according as its **outermost part** lies on an unshaded or a shaded area respectively.

Peirce had defined logic as "the formal science of the conditions of the truth of representations" (CP 2.220). To see the similarity to Tarski's approach, compare the following quotations:

- Peirce (1869): "All that the formal logician has to say is, that if facts capable of expression in such and such forms of words are true, another fact whose expression is related in a certain way to the expression of these others is also true.... The proposition 'If A, then B' may conveniently be regarded as equivalent to 'Every case of the truth of A is a case of the truth of B.'"
- Tarski (1936): "In terms of these concepts [of model], we can define the concept of logical consequence as follows: *The sentence X follows logically from the sentences of the class K if and only if every model of the class K is also a model of the class X.*"

Although Peirce's method is logically equivalent to Tarski's, the proof of equivalence was not discovered until Hilpinen (1982) showed that Peirce's endoporeutic is a variation of *game-theoretical semantics* (Hintikka 1973). For their introduction to model theory, Barwise and Etchemendy (1993) chose the title *Tarski's World*, but what they presented was a version of the game-theoretical method.

In modern terminology, endoporeutic can be defined as a *two-person zero-sum perfect-information game*, of the same genre as board games like chess, checkers, and tic-tac-toe. Unlike those games, which frequently end in a draw, every finite EG determines a game that must end in a win for one of the players in a finite number of moves. In fact, Henkin (1961), the first modern logician to rediscover the game-theoretical method, showed that it could evaluate the denotation of some infinitely long formulas in a finite number of steps. Peirce also considered the possibility of infinite EGs: "A graph with an endless nest of seps [ovals] is essentially of doubtful meaning, except in special cases" (CP 4.494). Although Peirce left no record of those special cases, they are undoubtedly the ones for which endoporeutic terminates in a finite number of steps. The version of endoporeutic presented here is based on Peirce's writings, supplemented with ideas adapted from Hintikka (1973), Hilpinen (1982), and Pietarinen (2006).

In the game of endoporeutic, one player, whom Peirce called Graphist, proposes a graph *g* that is claimed to be true, and another player, called Grapheus, is a skeptic or devil's advocate who tries to show that *g* is false. The game begins with some state of affairs, which could be represented by a Tarski-style model M=(D,R), in which D is a set of individuals and R is a collection of relations defined over D. The model M, like any Tarski-style model, can be represented as a relational graph (an EG with no negations): each individual in D is represented by a line of identity; each *n*-tuple of each relation *r* in R is represented by a copy of the string that names *r* with *n* pegs attached to *n* lines of identity. The game proceeds according to the following recursive procedure with Graphist as the initial proposer and Grapheus as the initial skeptic. During the game, they switch sides as they peel off each negation.

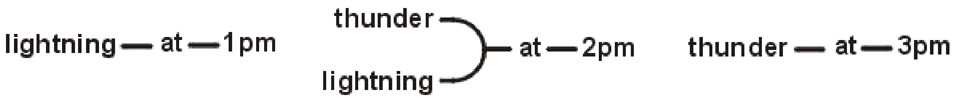
1. If *g* contains no negations, then the game is over, and the winner is determined by one of the following three conditions:
   - If *g* is an empty graph, it is true because it says nothing false. The current proposer wins.
   - Else if there is a mapping (graph homomorphism) of *g* to M, the current proposer wins. The mapping need not be an isomorphism, since multiple lines of identity in *g* may map to the same line in M. Each relation node in *g* must map to a relation node in M with the same name.
   - Else the current skeptic wins.
2. Else if *g* consists of just a single negation, the graph inside the oval becomes the new *g*, and the shading of each area in *g* is reversed. Then the two players switch sides: the proposer becomes the new skeptic, the skeptic becomes the new proposer, and the game continues with the new *g* and the new roles for both players.
3. Else if *g* consists of two or more negations, the skeptic chooses any one of the negations as the new *g*, and the game continues.
4. Else *g* consists of two or more parts: a subgraph g_0, which is a relational graph with no negations, and one or more negations: $g_1,...,g_n$. The graph g_0 may contain some lines of identity that are joined to lines inside one or more of the negations. There are now two possibilities:
   - If there is no mapping of g_0 to M, the game is over, and the skeptic wins.
   - Else one or more mappings are possible, and the proposer may choose any one. That choice maps every line of identity *x* in g_0 to some line *y* in M. If *x* is joined to any line *z* in any negation g_i, then *z* must remain mapped to *y* for the duration of the game. Then the subgraph g_0 is erased, leaving *g* as a conjunction of one or more negations, and the game continues.

Since each pass through this procedure reduces the size of *g*, any game that starts with a finite graph must terminate in a finite number of moves. Since none of the ending conditions results in a draw, either Graphist, the original proposer, or Grapheus, the original skeptic, must have a winning strategy for any *g* and model M.

If Graphist has a winning strategy, *g* is true of M. If Grapheus has a winning strategy, *g* is false of M.

These rules can evaluate Alpha graphs (propositional logic) in exactly the same way as Beta graphs. A model M for propositional logic would be a set of medads, such as {**(p)**, **(q)**, **(t)**}. A graph *g* with no negations is true of M if the set of medads in *g* is a subset of the medads in M. As an exercise, evaluate the EG for the Praeclarum Theorema in terms of some set of medads. Since it is a theorem, it should be true of every model, including the empty set.

To illustrate these rules for Beta graphs, consider the following relational graph as a model M of a state of affairs in which there is lightning at 1 pm, thunder and lightning at 2 pm, and thunder at 3pm.



As the first example, let the graph *g* be **lightning—at—2pm**. Since *g* has no negations, Graphist, the original proposer, wins because there is a mapping from *g* to the middle part of M. But if *g* happened to be **lightning—at—3pm**, then Grapheus would win, because this *g* cannot be mapped to M.

As an example that requires more than one pass through the procedure, consider Peirce's graph in Figure 4, which represents the sentence "There is thunder, and not lightning." That graph meets the conditions for Step 4 of the endoporeutic procedure: the subgraph g_0 for the part outside the negation is **—thunder**. Graphist, the proposer, can choose any of two possible mappings of g_0 to M: one for thunder at 2 pm and the other for thunder at 3 pm; this choice also determines the mapping of the line inside the negation. If Graphist chooses 3 pm, the subgraph g_0 is erased, and the game continues. At Step 2, the negation is erased, causing the new *g* to become **—lightning**, but with the line forced to be mapped to the subgraph of M at 3 pm. Then the two players change sides, making Graphist the new skeptic. The game continues at Step 1, where *g* cannot be mapped to M because lightning occurred at 2 pm, not 3 pm. Therefore, the skeptic wins because this subgraph with this mapping is false. But the current skeptic is Graphist, who had been the original proposer for Figure 4, which is therefore true. Note that if Graphist had made a mistake in the original choice of mapping, then Graphist would have lost. The truth of a graph, however, is not determined by mistakes; it depends only on the existence of a winning strategy if both players make the best choice at each option.

As a final example, consider Figure 3, which adds one more negation around Figure 4. This graph meets the condition for Step 2 of the game, which removes the negation and causes the two players to change sides. That makes Grapheus the proposer for a graph that is now identical to Figure 3, which has a winning strategy. Therefore, Grapheus wins the game that started with Figure 4, which must therefore be false. Since Graphist and Grapheus may change sides during the game, a graph is true if and only if Graphist wins, independent of which role Graphist happens to be playing at the end.

To use endoporeutic to prove that Peirce's rules of inference are sound, it is necessary to show that each rule preserves truth — i.e., if Graphist has a winning strategy with a given EG, no rule of inference can transform it to a graph that allows Grapheus to win. Therefore, the rules must monotonically increase the winning options for Graphist and monotonically decrease the winning options for Grapheus. Since the two players switch sides when a negation is removed (Step 2), Graphist is the proposer in every positive area and the skeptic in every negative area. The two steps of endoporeutic that depend on the number of options are Step 3, where the skeptic chooses one of the negations, and Step 4, where the proposer chooses one of the possible mappings of g_0 to M.

1. Rule 1e erases an arbitrary subgraph in a positive area. Erasing a negation decreases the number of options for Grapheus, who is the skeptic in a positive area. Erasing all or part of a relational graph reduces the constraints on the mapping to the model M and thereby increases the options for Graphist.

   Rule 1i, which inserts an arbitrary subgraph in a negative area, has the opposite effects: If negations are added, Graphist, who is the skeptic, has more options to choose. Adding a subgraph to a relational graph adds more constraints on the mapping to M and thereby reduces the options for Grapheus.

2. Rules 2e (deiteration) and 2i (iteration) have no effect on the truth value of any graph because any subgraph *g* that could be iterated or deiterated has its effect on the evaluation at its first occurrence. There are two possibilities to consider: *g* is true, or *g* is false. If *g* is true, a copy of *g* inserted or erased from any area has no effect on the truth value of that area. If *g* is false, the original area in which *g* occurs is false; the truth value of any nested area is irrelevant.
3. The only effect of evaluating a double negation is to cause the proposer and skeptic to reverse their roles twice. Therefore, Rules 3e (erasing a double negation) and 3i (inserting a double negation) can have no effect on the truth value of any graph.

In summary, rules 1e and 1i monotonically increase the options for Graphist and monotonically decrease the options for Grapheus. Rules 2e, 2i, 3e, and 3i have no effect on the winning strategies for either side.

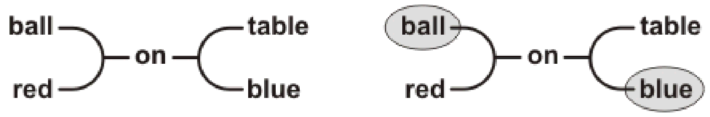
What distinguishes the game-theoretical method from Tarski's approach is its procedural nature. One reason why Peirce had such difficulty in explaining it is that he and his readers lacked the vocabulary of the game-playing algorithms of artificial intelligence. Peirce had written many pages about endoporeutic, but no one clearly deciphered them until Hilpinen noticed the similarity to game-theoretical semantics. The discussion in this section is a reconstruction and clarification in modern terminology of Peirce's often cryptic notes.

Unlike Tarski's definition, which maps all variables in a formula to individuals in a model M, endoporeutic is a "lazy" method that avoids mapping a line to M until the proposer chooses it in Step 4. Any subgraph that is not chosen is never evaluated; some very large or even infinite subgraphs can often be ignored. Lazy methods similar to endoporeutic are used in computational systems, such as evaluating an SQL query in terms of a relational database. Game-playing programs reduce the computation by a lazy method called the α-β algorithm. An application of α-β to endoporeutic could order the skeptic's options in Step 3 with the smaller graphs first. If the best strategy for the skeptic is to choose one of the smaller graphs, the larger, possibly infinite graphs, might be ignored. Another optimization is to index the names of relations in M in order to speed up Step 4, where the proposer searches for a mapping from a subgraph to M. With these optimizations, the computational complexity of endoporeutic is the same as evaluating an SQL query in terms of a relational database.

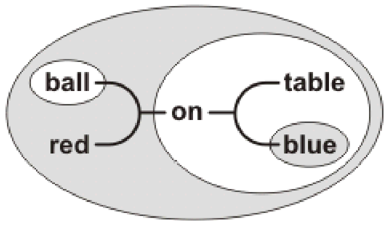
5. How to Teach Logic

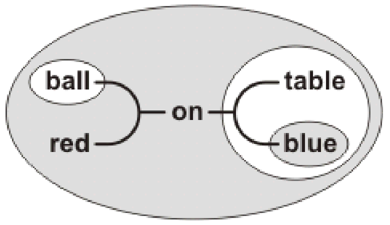
Peirce's approach in MS 514 is an excellent basis for teaching logic. The relational subset consisting of just conjunction and the existential quantifier is easy to understand, and it is widely used, even by people who have no idea that it is a subset of logic. Without any additional features, relational graphs can describe anything that exists. They are general enough to express the content of a relational database or anything expressed in the Resource Description Framework (RDF). Peirce (1882) tried to generalize relational graphs by adding other operators, but he couldn't state everything expressible in his algebraic notation.

As an example, the sentence "A red ball is on a blue table" can be expressed by the relational graph on the left below. With small ovals that negate one relation at a time, the graph on the right says "Something red that is not a ball is on a table that is not blue."



Peirce had tried to add more features to relational graphs, but he couldn't find a way to extend the scope of a negation or quantifier beyond such local modifications. In the 1960s, that same limitation plagued the development of *semantic networks* by the artificial intelligence community (Sowa 1987). Peirce's breakthrough of 1897 was merely to use larger ovals, as in the next two graphs.





Since each of these graphs has four ovals, there are several different ways of reading them. They make good classroom exercises for soliciting different readings from the students. The graph on the left can be read as a simple *if-then* sentence, as an *if* sentence with a disjunctive conclusion, or as a universally quantified sentence with a disjunctive body:

- "If there is a red thing that is not a ball, then it's on a table that is not blue."
- "If there is something red, then either it's a ball or it's on a table that is not blue."
- "Every red thing is a ball or is on a table that is not blue."

Since the graph on the right has two lines of identity (quantifiers) inside the shaded oval, the English pronouns must be supplemented with other words to distinguish them:

- "If a red thing that is not a ball is on something, then the latter is a table that is not blue."
- "If a red thing is on something, then either the former is a ball or the latter is a table that is not blue."
- "If a red thing *x* is on something *y*, then either *x* is a ball or *y* is a table that is not blue."
- "For every red thing *x* on something *y*, either *x* is a ball or *y* is a table that is not blue."

Letters or variables are used in the algebraic notation for logic or to supplement the pronouns of a natural language. Such supplements are necessary for a linear notation, but they are not needed in a graph notation that shows identity by direct connections.

These diagrams illustrate some fundamental principles of logic, which are true of any notation, but which are especially clear when expressed in EGs:

- Without negations, the operators of conjunction and existence are sufficient to describe anything that exists. That includes all the experimental data of any branch of science, since negations can only be inferred, never observed directly.
- Local negations, which negate an indivisible graph or atom, deny that a particular property or relation is true of the individual(s) designated by the pegs attached to the relation name.
- To deny an entire configuration of entities and relations, negations with a larger scope are necessary. The scope can be demarcated by a larger oval in EGs, by a parenthesized expression in a formula, or by a phrase or sentence in natural languages.
- To state generalizations, implications, and disjunctions, a nest of two negations is required. In some notations, the nest of negations may be implicit in a single symbol, such as $\forall$, $\supset$, and $\vee$.

Franz Brentano, who was born a few months before Peirce, also maintained the primacy of conjunction, negation, and the existential quantifier as the basic logical operators. Although Brentano's notation was limited to expressing Aristotle's syllogisms, it used the same three operators as EGs in logically equivalent ways (Simons 2004).

A good way to illustrate these principles is to start with a relational graph on any subject, overlay ovals of various sizes, and ask the students to translate them to their native language. The graphs can be drawn in any medium ranging from chalk on a blackboard to animated computer graphics. In classroom exercises, colored balls, blocks, and tables can be used to build structures. Then the students can compete to see who can be the first to describe a structure in a correct EG, translate an EG to English, or modify a structure when the EG is changed. Simple relational graphs are sufficient to describe any structure that can be built, but the teacher can ask the students to draw EGs that express generalizations or options that are true of several different structures. Another possibility is to draw a general EG and ask the students to find many different structures that make the EG true or false. These exercises serve two purposes: they teach logic, and they teach the students how to express themselves clearly and precisely in their native language.

Logicians sometimes object to teaching EGs because the algebraic notation is the one that students must use in order to get their papers published. But Don Roberts showed that EGs are a superior notation for learning logic, even when the students later use other notations. At the philosophy department of the University of Waterloo, Roberts and another professor were scheduled to teach two sections of an introductory course on logic. Roberts proposed a direct comparison: he would teach his section using EGs, and the other professor would use a traditional textbook based on the algebraic notation. At the end of the course, the students in both sections would take the same final exam, which would use only the algebraic notation. The other professor agreed. For almost the entire course, Roberts used EGs to introduce all the concepts of logic including Peirce's method of proofs. In the last two weeks, he showed how to translate EGs to and from the algebraic notation, including some exercises for adapting Peirce's proof methods to the algebraic notation. On the final exam, the section that began with EGs had a higher average score, and their scores on problems that involved proofs were much higher.

The comparison by Roberts and his colleague was not a controlled study, but it's typical of the way students respond to existential graphs. For evidence of why they prefer EGs, look at the usual exercises in courses, textbooks, and exams. Following is a problem from an exam that was posted on the Web:

> Premises: $r \lor u$. $\sim q \supset \sim r$. $\sim s$. $q \supset s$.
>
> Use the argument forms modus tollens, modus ponens, disjunctive syllogism, and conjunctive addition to deduce $\sim r \land u$. Justify each step.

As an exercise, solve this problem in two ways: by the teacher's method and by applying Peirce's rules to the equivalent EGs. Note that the teacher had stated exactly which derived rules of inference to use and in which order to apply them (probably because too many students had failed previous exams). With EGs, the three permissions without any derived rules are simple and efficient: by 2e, derive $\sim q$; by 2e, derive $\sim\sim\sim r$; by 3e, derive $\sim r$; by 2e, derive $\sim\sim u$; by 3e, derive u; by 1e, erase everything but the desired conclusion, $\sim r \land u$. Some teachers who would like to adopt EGs complain that there is no suitable textbook. But they can use a traditional textbook: explain the text in terms of EGs and ask the students to use EGs to do the exercises. The students learn to think in EGs and to translate other notations to the more natural EGs.

In my own experience, I had taught logic in the traditional way long before I discovered Peirce's EGs. After I learned EGs, I still used the traditional methods and textbooks in teaching, but I added two hours on EGs at the end of the course: one hour on the syntax and one hour on proofs. The students' major complaint was that I "had saved the good wine for last." Since then, I found that students at all levels master the fundamental concepts of logic faster and more thoroughly with the EG notation. Later, they can transfer their skills to any notation, including their native language. Transfer is the ultimate goal. As Peirce (1878) said, "The very first lesson that we have a right to demand that logic shall teach us is, how to make our ideas clear."

6. Advanced Topics

Students find the EG graphic notation easy to learn and use, and professional logicians enjoy a novel way of viewing familiar topics. But the symmetry of the EG structure and the simplicity of the rules of inference also support a proof theory that is simpler and more powerful than traditional methods. All the common proof procedures can be derived from Peirce's rules, but the converse does not hold: some proofs by Peirce's rules can be orders of magnitude shorter than proofs by other methods. The symmetry properties, for example, imply the *reversibility* theorem.

- **Reversibility Theorem.** Any proof of $p \vdash q$ by Peirce's rules of inference for existential graphs can be converted to a proof of $\sim q \vdash \sim p$ by negating the EGs for each step and reversing their order.
- **Proof.** Let $s_0, ..., s_n$ be the steps of the proof with s_0 as the EG that represents p, s_n as the EG that represents q, and each r_i from 1 to n as the rule that converts $s_{i\text{-}1}$ to s_i. For the reverse sequence of

negated EGs, note that ~[s_n] represents ~*q*, and ~[s_0] represents ~*p*. Furthermore, each ~[s_i] is converted to ~[s_{i-1}] by the inverse of rule r_i. The only question is whether the inverse conversions are permissible. Since the rules 2e, 2i, 3e, and 3i are equivalence operations, their inverses are permissible in any area, positive or negative. But 1e can only be performed in a negative area, and its inverse 1i can only be performed in a positive area. Since each step of the reverse proof is negated, the polarity (negative or positive) of every nested area in each ~[s_i] is reversed. Therefore, any application of 1i or 1e in the forward direction can be undone by its inverse (1e or 1i) in the reverse direction.

In most versions of logic, a proof of $p \vdash q$ can be converted to a proof of $\sim q \vdash \sim p$. But the conversions are more complex, because traditional rules of inference don't have exact inverses and they can't be applied inside a negated area. For some kinds of proofs, Peirce's method is much shorter because of a property that is not shared by other common proof procedures: his rules can perform surgical operations (insertions or erasures) in an area of a graph or formula that is nested arbitrarily deep. Furthermore, the rules depend only on whether the area is positive or negative. Those properties imply the *cut-and-paste* theorem (Sowa 2000), which can be adapted to any notation for first-order logic:

- **Cut-and-Paste Theorem.** If a proof $p \vdash q$ is possible on a blank sheet sheet of assertion, then in any positive area of an EG where *p* occurs, *q* may be substituted for *p*.
- **Proof.** Since the nested area in which *p* occurs is positive, every step of a proof on a blank sheet can be carried out in that area. Therefore, it is permissible to "cut out" the steps of a proof from *p* to *q* in the outer area and "paste" them into the nested area. After *q* has been derived, Rule 1e can be used to erase the original *p* and any remaining steps of the proof other than *q*.

From the cut-and-paste theorem (CAPT), other important theorems can be proved as corollaries. One example is the *deduction theorem*:

- **Deduction Theorem**. If a proof $p \vdash q$ is possible, then $p \supset q$ is provable.
- **Proof**. To show that $p \supset q$ is provable, use CAPT to derive **~[(p) ~[(q)]]** from a blank:
    1. By 3i, draw a double negation around a blank: **~[~[ ]]**
    2. By 1i, insert **(p)** in the negative area: **~[(p) ~[ ]]**
    3. By 2i, iterate **(p)** to derive the equivalent of $p \supset p$: **~[(p) ~[(p)]]**
    4. By CAPT, replace the inner **(p)** with **(q)**: **~[(p) ~[(q)]]**

The *constructive dilemma* is another another corollary of CAPT that is difficult to prove in many systems:

- **Constructive Dilemma**. If $p_1 \vdash q$ and $p_2 \vdash q$, then $p_1 \vee p_2 \vdash q$.
- **Proof**: Use two applications of CAPT.
    0. Starting graph: **~[~[(p1)] ~[(p2)]]**
    1. By CAPT, replace **(p1)** with **q**: **~[~[(q)] ~[(p2)]]**
    2. By CAPT, replace **(p2)** with **q**: **~[~[(q)] ~[(q)]]**
    3. By 2e, deiterate one copy of **~[(q)]**: **~[~[(q)]]**
    4. By 3e, erase the double negation: **(q)**

Converting various proof procedures to EG form provides fundamental insights into the nature of the proofs and their interrelationships. Gentzen (1934) developed two proof procedures: *natural deduction* and the *sequent calculus*. A proof by either one can be systematically converted to a proof by Peirce's rules. The converse, however, does not hold because Gentzen's rules cannot operate on deeply nested expressions. For some proofs, many steps are needed to bring an expression to the surface of a formula before those rules can be applied. An example is the *cut-free* version of the sequent calculus, in which proofs can sometimes be exponentially longer than proofs in the usual version. Dau (2006) showed that with Peirce's rules, the corresponding cut-free proofs are longer by just a polynomial factor.

Like Peirce's rules, Gentzen's rules for natural deduction also come in pairs, one of which inserts an operator, which the other erases. Gentzen, however, required six pairs of rules for each of his six operators: ∧, ∨, ⊃, ~, ∀, and ∃. Peirce had only three operators and three pairs of rules. Even when limited to the same three operators, Gentzen's rules are highly irregular. They lack the symmetry of paired rules that are exact inverses of one another. Four of the twelve rules even require a provability test, expressed by the operator ⊢. Yet Gentzen's rules can all be proved as derived rules of inference in terms of Peirce's rules. In fact, two of them were just proved above: the rule for ⊃-introduction is the deduction theorem, and the rule for ∨-elimination is the constructive dilemma. The following table shows Gentzen's six pairs of rules.

	Introduction Rules	**Elimination Rules**
$\wedge$	$\frac{A,\ B}{A \wedge B}$	$\frac{A \wedge B}{A}$ $\quad$ $\frac{A \wedge B}{B}$
$\vee$	$\frac{A}{A \vee B}$ $\quad$ $\frac{B}{A \vee B}$	$\frac{A \vee B,\ A \vdash C,\ B \vdash C}{C}$
$\supset$	$\frac{A \vdash B}{A \supset B}$	$\frac{A,\ A \supset B}{B}$
$\sim$	$\frac{A \vdash \bot}{\sim A}$ $\quad$ $\frac{\bot}{A}$	$\frac{A,\ \sim A}{\bot}$ $\quad$ $\frac{\sim\sim A}{A}$
$\forall$	$\frac{A(a)}{(\forall x)A(x)}$	$\frac{(\forall x)A(x)}{A(t)}$
$\exists$	$\frac{A(t)}{(\exists x)A(x)}$	$\frac{(\exists x)A(x),\ A(a) \vdash B}{B}$

All the rules in Gentzen's table can be proved as derived rules of inference in terms of Peirce's rules, but some of them require further explanation. The symbol ⊥ in the rules for introducing or eliminating the negation symbol ~ represents a proposition that is always false. In EGs, a single oval with nothing inside or **~[ ]** represents ⊥. Peirce called **~[ ]** the *pseudograph* because it is always false. In the resolution method for theorem proving, it is called the *empty clause*. From the pseudograph, any proposition A can be derived: start with **~[ ]**; by 1e, insert **~[(A)]** to derive **~[~[(A)]]**; by 3e, erase the double negation to derive **(A)**.

Two other special symbols are the arbitrary term *t* in the rules for ∃-introduction and ∀-elimination and the arbitrary value or free variable *a* in the rules for ∀-introduction and ∃-elimination. The arbitrary term *t* is an ordinary expression with no free variables, and it can be represented as **—t** or **[*x] (t ?x)**, where **t** is any relation or graph with one peg designated to represent the value of the term. The rule for ∀-elimination, also called universal instantiation, was proved in Section 3. The proof for ∃-introduction takes one step:

0. Starting graph: **[*x] (t ?x) (A ?x)**
1. By 1e, erase **(t ?x)**: **[*x] (A ?x)**

Since EGs don't have variables, they don't have free variables. But proof procedures that use arbitrary values or free variables have been a source of controversy for years. Fine (1985) interpreted Gentzen's rule of ∀-introduction as an assumption that A(*a*) had been previously derived from some statement φ and the existence of some arbitrary value *a*, which could be represented by **[*a]** in EGIF. Following is a proof that uses the cut-and-paste theorem:

0. Starting assumption: **φ, [*a] ⊢ (A ?a)**
1. By 3i, insert a double negation around a blank: **~[~[ ]]**
2. By 1i, insert **[*x]** into the negative area: **~[[*x] ~[ ]]**

3. By 2i, iterate **φ** from step 0 into the doubly nested area: **~[[*x] ~[φ]]**
4. By 2i, iterate **[?x]** into the doubly nested area: **~[[*x] ~[φ [?x]]]**
5. By CAPT, replace **φ [?x]** with **(A ?x)**: **~[[*x] ~[(A ?x)]]**

For Gentzen's rule of ∃-elimination, A(*a*) is true of some arbitrary values, but not all. The conclusion B is provable from A(*a*) only for values of *a* for which A(*a*) is true.

0. Starting assumptions: **[*x] (A ?x); [*a] (A ?a) ⊢ (B)**
1. Let **[*a]** be some **[*x]** that makes **(A ?x)** true. Then: **(B)**

Although each of Gentzen's rules can be derived from Peirce's rules, proofs in the two systems take different paths. EG proofs proceed in a straight line from a blank sheet to the conclusion: each step inserts or erases one subgraph in the immediately preceding graph. No bookkeeping is required to keep track of information from any previous step. But Gentzen's proofs interrupt the flow of a proof at every rule that contains the pattern $p\vdash q$. When deriving *q* from *p*, those rules use a method of bookkeeping called *discharging an assumption*:

1. Whenever the symbol ⊢ appears in a rule, some proposition *p* is assumed.
2. The current proof is suspended, and the state of the current rule is recorded.
3. A side proof of *q* from *p* is initiated.
4. When the conclusion *q* is derived, the assumption *p* is *discharged* by resuming the previous proof and using *q* in the rule of inference that had been interrupted.

Such side proofs might be invoked recursively to an arbitrary depth.

EG proofs avoid that bookkeeping by an option that most notations for logic can't represent: drawing a double negation around a blank. In the proof of the Praeclarum Theorema, for example, the first step is to draw a double negation on a blank sheet, and the second step is to insert the hypothesis into the shaded area. The result is a well-formed EG, and no bookkeeping is necessary to keep track of how that EG had been derived. In Gentzen's method, the first step is to assume the hypothesis, and a record of that assumption must be retained until the very end of the proof, when it is finally discharged to form the conclusion. This observation is the basis for converting a proof by Gentzen's method of natural deduction to a proof by Peirce's method:

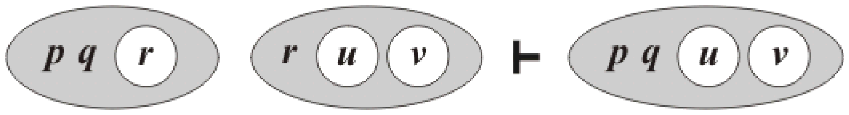
- Replace each of Gentzen's rules that does not contain the symbol ⊢ with one or more of Peirce's rules that produce an equivalent result.
- Whenever one of Gentzen's rules containing ⊢ is invoked, apply rule 3i to insert a double negation around the blank, copy the hypothesis into the negative area by rule 1i, and continue.

An EG proof generated by this procedure will be longer than a direct proof that starts with Peirce's rules. As an exercise, use Gentzen's table of rules and his method of discharging assumptions to prove the Praeclarum Theorema. To avoid the bookkeeping, introduce a nest of two ovals when the symbol ⊢ appears in a rule. Insert the hypothesis into the shaded area, but continue to use the algebraic notation for the formulas. Finally, translate each formula to an EG, and add any intermediate steps needed to complete the proof according to Peirce's rules. Proofs by Gentzen's other method, the sequent calculus, can be converted to EG proofs more directly because they don't require extra steps to discharge assumptions.

In computational systems, the widely used method of *resolution* applies a single rule of inference to a set of *clauses* (Robinson 1965). Each clause is a disjunction of positive or negative *literals*, which are single atoms or their negations. In the following example, the two clauses on the left of ⊢ are combined by resolution to derive the clause on the right:

$$\sim p \lor \sim q \lor r,\quad \sim r \lor u \lor v \quad \vdash \quad \sim p \lor \sim q \lor u \lor v$$

The leftmost clause has a positive literal *r*, and the middle clause has a negated copy *~r*. Resolution is a cancellation method that erases both *r* and *~r* and merges the remaining literals to form the clause on the right. Following are the equivalent EGs for the three clauses:



With EGs, resolution takes four steps: by 2i, iterate the middle EG into the doubly nested (unshaded) area of the EG on the left; by 2e, erase the innermost copy of *r*; by 1e, erase the remaining copy of *r*; by 3e, erase the double negation. The result is the EG on the right. Resolution is commonly used in a refutation procedure, which attempts to prove a statement *s* by deriving a contradiction from ~*s*:

1. Negate the statement to be proved: ~*s*.
2. Convert ~*s* to clause form.
3. Use repeated steps of resolution to derive the empty clause, **~[ ]**.
4. Since the empty clause is false, the original *s* must be true.

Many logicians have observed that proofs by resolution are "almost" the inverse of proofs by natural deduction. They aren't exact inverses, however, because the rule of resolution is simple and regular, but Gentzen's six pairs of insertion and elimination rules are highly irregular. But the proof that resolution is the inverse of Peirce's form of natural deduction is a variant of the reversibility theorem. For propositional logic, start by negating each step. The negation of the empty clause in step 4 is a double negation of a blank sheet, which is always true. Each application of the resolution rule corresponds to four EG rules, which can be reversed inside a negation. A conversion to clause form is an equivalence that can be performed in any context; and the double negation ~~*s* is equivalent to *s*.

With quantifiers, two additional procedures must also be checked for reversibility: *skolemization* in step 2 and *unification* in step 3. In clause form, all variables are governed by universal quantifiers: $\forall x_1,...,\forall x_n$. Skolemization replaces each existential quantifier with a term *t*, which is a constant or a function applied to one or more of the universally quantified variables. Unification, which is discussed in Section A.3, involves universal instantiations followed by joining the pegs of functions or relations that are being unified. After the pegs are joined, rule 2e allows redundant copies of relation or function symbols to be erased.

To show that skolemization and unification are reversible with EG rules, note that the universal quantifiers of an algebraic formula correspond to lines of identity or defining nodes **[*x1]**,...,**[*xn]** just inside the shaded area of the corresponding EG. To show that skolemization is reversible inside a negation, consider the following example, in which the skolem function *s*(*x*) replaces the variable *y*:

$$\forall x\exists y\mathrm{R}(x,y) \quad\Rightarrow\quad \exists f\,\forall x\mathrm{R}(x,f(x)) \quad\Rightarrow\quad \forall x\mathrm{R}(x,s(x))$$

If the left formula is true, the middle one must be true for at least one function *f*, and the right formula names such a function *s*. When negated, these implications are reversible: if the right formula is false, no function *f* exists for the middle one, and the left one is also false. This argument can be generalized to any number of universal quantifiers, including zero for a skolem constant.

Unification is an application of ∀-elimination followed by merging identical functions or relations applied to identical values. In EGs, it is performed by joining a subgraph that represents a constant or functional term to a line of identity followed by joining pegs according to the rule of inference discussed in Section A.3. Its inverse inside a negation corresponds to Gentzen's ∃-introduction. In EGs, it is performed by cutting the line and erasing the subgraph. The operation of joining lines whose values are known to be identical is an equivalence that can be reversed in any area.

On broader questions about graph representations, many theoretical and computational issues remain to be explored. Stewart (1996) showed that a theorem prover based on EGs with Peirce's rules of inference was comparable in performance to a resolution theorem prover. But the world's fastest theorem provers depend on efficient methods for storing intermediate results and recognizing which are the most relevant at each step. Transforming graphs to clause form can be counterproductive because it may obscure features that are important for pattern recognition. Peirce's original inspiration for graph logic came from the graphical notations in organic chemistry, and chemists are still in the forefront of research on finding patterns in large graphs and large collections of graphs. The innovations in processing chemical graphs have led to powerful new techniques for logical graphs (Majumdar and Sowa 2009). Interdisciplinary research that combines techniques from chemistry, logic, and mathematics may lead to further breakthroughs.

As these discussions indicate, the structure of EGs and Peirce's rules of inference can help generalize and clarify the foundations of logic and proof theory. Since each rule inserts or erases a semantic unit, those rules can be applied to any notation just by defining negative areas, positive areas, and semantic units in terms of the syntax. The rules can also be adapted to modal, intuitionistic, relevance, and nonmonotonic logics by varying the constraints on insertions and erasures. Higher-order logic uses Cantor's hierarchy of infinities to represent a hierarchy of domains for quantifiers. Although Peirce had studied and commented on Cantor's theories, he put his relations in the same "universe of discourse" as the individuals. In that regard, Peirce's semantics is closer to some systems currently used in computer science, such as the model theory for Common Logic, which supports EGIF. Peirce's voluminous manuscripts about logic are a gold mine of insights and innovations. Their relative neglect during the 20th century makes them a largely unexplored treasure for the 21st.

Appendix: EGIF Grammar

The 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 many variants of notation, terminology, and explanations of existential graphs. 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 can be compared and analyzed.

EGIF is a proper subset of the Conceptual Graph Interchange Format (CGIF), which is one of the dialects of the ISO/IEC standard 24707 for Common Logic. Any statement in EGIF can be automatically processed by computer systems designed for CGIF. In this appendix, Section A.1 presents the lexical rules for the names and identifiers used in EGIF. Section A.2 presents the phrase-structure rules. Section A.3 states context-sensitive constraints and gives some examples. Section A.4 discusses the IKL extensions beyond Common Logic and their use in representing Peirce's Gamma graphs.

A.1 Lexical Rules

All EGIF syntax 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 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. In case of ambiguity, the EBNF rule is normative.

The following four lexical categories are defined formally in Section A.2 of ISO/IEC 24707, which is the normative specification. 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 from '0' to '9'.
- An *enclosedname* is any sequence of Unicode characters, except control characters, preceded and followed by a double quote '"'. Any double quote internal to an enclosedname must be represented by the string '\"'. Any backslash internal to an enclosedname must 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 specified in Section B.2 of ISO/IEC 24707:

- An *identifier* consists of 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 enclosedname with the same case and spelling: **Apple** names the same relation type as **"Apple"**, and **apple** names the same type as **"apple"**.

A.2 Phrase-Structure Rules

The EGIF phrase structure rules define a proper subset of CGIF. Unlike the lexical rules, the phrase-structure rules permit an arbitrary amount of white space between constituents. Following are two ways of writing the EGIF for Peirce's Figure 4:

```
~[[*x](thunder?x)~[(lightning?x)]]

~ [ [ * x ] ( thunder ? x ) ~ [ ( lightning ? x ) ] ]
```

Every EGIF statement is also a syntactically correct CGIF statement with the same semantics. But the EGIF category names reflect the EG structure, their definitions have fewer options than the rules for CGIF, and their names begin with an upper case letter to distinguish them from the category names for CGIF. Following are the EBNF rules. The start symbol is EG.

- A *bound label* consists of a question mark '?' and an identifier.

  BoundLabel = '?', identifier;

- A *coreference node* consists of a left bracket '[', one or more bound labels, and a right bracket ']'.

  CoreferenceNode = '[', BoundLabel, {BoundLabel}, ']';

- A *defining label* consists of an asterisk '*' and an identifier.

  DefiningLabel = '*', identifier;

- A *defining node* consists of a left bracket '[', a defining label, and a right bracket ']'.

  DefiningNode = '[', DefiningLabel, ']';

- A *double negation* is a negation that encloses a negation and no other node. This rule is not necessary to define the syntax of EGIF, but the term is used in the inference rules 3e and 3i.

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

- An *existential graph (EG)* consists of zero or more nodes.

  EG = {Node};

- A *function* consists of a left parenthesis '(', a type, zero or more bound labels, a vertical bar '|', a bound label, and a right parenthesis ')'.

  Function = '(', Type, {BoundLabel}, '|', BoundLabel, ')';

- A *negation* consists of a symbol '~', a left bracket '[', an existential graph, and a right bracket ']'.

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

- A *node* is one of a defining node, a coreference node, a relation, a function, or a negation.

  Node = DefiningNode | CoreferenceNode | Relation | Function | Negation;

- A *relation* consists of a left parenthesis '(', a type, zero or more bound labels, and a right parenthesis ')'.

  Relation = '(', Type, {BoundLabel}, ')';

- A *type* is one of an identifier, an enclosed name, or a hash mark '#' followed by a bound label.

  Type = identifier | enclosedname | '#', BoundLabel;

A.3 Constraints and Examples

As the grammar rule for EG shows, an existential graph is represented by zero or more nodes in EGIF. The only constraints on the nodes or their ordering are determined by the scope of defining nodes and their bound nodes:

1. No defining node may be in the scope of another defining node with the same identifier.
2. Every bound label must be in the scope of its defining node.
3. Every defining node must precede (occur to the left of) any node that directly or indirectly contains one of its bound labels.

In any area, all permutations of the nodes that preserve these three constraints are equivalent.

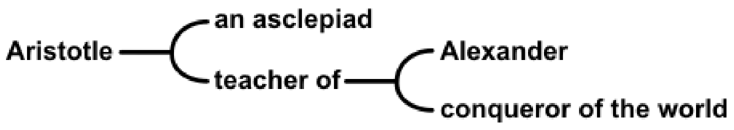
A name enclosed in double quotes, such as **"John Q. Public"**, can contain spaces and punctuation. In the EGIF for Peirce's Figure 5, the identifier **will_die** used an underscore to represent the space, but Peirce's choice of name could also be represented as **"will die"**. Either option is permissible in EGIF, but they're not interchangeable, since an underscore is not identical to a space.

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 can be considered a kind of relation for which the value represented by the last argument (or peg in Peirce's terms) is uniquely determined by the values of the pegs that precede the vertical bar. The pegs to the left of the vertical bar may be called the *input pegs*, and the one to the right of the bar may be called the *output peg*. That constraint implies the following rule of inference for functions:

- 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*, then the output pegs of both instances may be joined.

This rule of inference is the basis for the method of *unification* for reasoning about combinations of functions. The EGIF grammar also allows functions with zero inputs. Every instance of such a function would have exactly the same output value, and this rule of inference would imply that the output pegs of all instances of that function may be joined.

Peirce did not introduce an EG notation for proper names or constants. Instead, he used monadic relations, such as **—Alexander**, which would be true of anyone named Alexander. Following is a subgraph from one of his examples (CP 4.445, Figure 80).



In EGIF, a name that is true of exactly one individual could be used as the name of a function with zero inputs. In the function **("Philippus Aureolus Theophrastus Bombastus von Hohenheim" | ?p)**, the bound label **?p** would represent a line of identity for the unique person with that name. 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: **—Alexander→**.

The grammar rule for Type includes an option with a hash symbol **#** followed by a bound label. That option supports the feature of Common Logic that allows quantified variables to range over functions and relations. As an example, consider the sentence "There is family relation between any two members of the same family." Following are two interpretations of that sentence and their representations in EGIF:

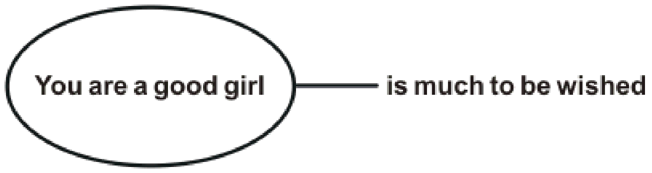
1. “R is a relation, and if *x* and *y* are members of a family F, then *x* and *y* are related by R.”
   ```
   [*R] (relation ?R)
       ~[[*F] [*x] [*y] (family ?F) (memberOf ?x ?F) (memberOf ?y ?F)
           ~[ (#?R ?x ?y] ]
   ```
2. “If F is a family, then R is a relation, and if *x* and *y* are members of F, then *x* and *y* are related by R.”
   ```
   ~[ [*F] (family ?F) ~[ [*R] (relation ?R)
       ~[[*x] [*y] (memberOf ?x ?F) (memberOf ?y ?F)
           ~[ (#?R ?x ?y] ] ] ]
   ```

The first line of the second EGIF could also be read “For any family F, there is a relation R...” Note that EGIF allows bound labels that refer to relations to occur in the argument position (peg) of another relation or with the prefix **#** in the type position. For Gamma graphs, which are discussed in the next section, Peirce experimented with graphical ways of representing such options.

A.4 Extensions for Gamma Graphs

Throughout his long career, Peirce experimented with a variety of notations for logic and a wide range of semantic extensions that went far beyond ordinary first-order logic. In his article of 1885, in which he presented his most complete version of the algebraic notation, 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 to *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) summarizes the various graphical and algebraic notations and cites the publications and manuscripts in which Peirce discussed them.

Peirce used the term *Gamma graphs* for the many variations 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:



The sentence inside the oval could be expressed in EGIF as a proposition or medad with a name enclosed in double quotes. The line of identity and phrase outside the oval could be expressed by a defining node and a monadic relation with an enclosed name. But neither EGIF nor CGIF as defined by ISO/IEC 24707 can represent a line of identity linked to an oval. A proposed extension to Common Logic called IKL (Hayes and Menzel 2006) can support such constructs. An extension to the CLIF dialect with the IKL semantics uses an operator *that* followed by a CLIF sentence to denote the proposition stated by the sentence:

("is much to be wished" (that ("You are a good girl")))

An equivalent extension to CGIF or EGIF would use the following notation:

```
[*x ("You are a good girl")] ("is much to be wished" ?x)
```

Either the EG or its translation to CLIF, CGIF, or EGIF could be read *That you are a good girl is much to be wished*. A syntactic extension to EGIF for such expressions could be represented with an optional EG in the rule for DefiningNode:

```
DefiningNode = '[', DefiningLabel, [EG] ']';
```

When a line of identity represents a proposition, the bound label prefixed with **#** could be used in the type position of a medad to assert the proposition: **(#?x)**. It could then be negated in the usual way, **~[(#?x)]**, to say that it is false that you are a good girl. 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 first line of the following EGIF defines *x* as the proposition that you are a good girl and *y* as its negation. The second line says that if *x* is necessary, then *y* is not possible. With the double negation erased by rule 3e, that line would say it's false that *x* is necessary and *y* is possible.

```
[*x ("You are a good girl")] [*y ~[(#?x)]]
~[(Necessary ?x) ~[ ~[(Possible ?y)] ]]
```

The semantics of EGIF is formally defined by the model theory of Common Logic or the IKL extensions. For Alpha and Beta graphs, the EGIF semantics seems to be consistent with what Peirce had intended. Determining exactly what he had intended for his many variations of Gamma graphs is still a research project, for which EGIF can be a useful tool.

References

Barwise, Jon, and John Etchemendy (1993). *Tarski's World*. Stanford, CA: CSLI Publications.

Dau, Frithjof (2006). Some notes on proofs with Alpha graphs. In *Conceptual Structures: Inspiration and Application*, (LNAI 4068), H. Schärfe, P. Hitzler, and P. Øhrstrom (eds.), 172-188. Berlin: Springer.

Dau, Frithjof (2010). Ligatures in Peirce's Existential Graphs. *Semiotica* [This issue].

Fine, Kit (1985). *Reasoning with Arbitrary Objects*. Oxford: Basil Blackwood.

Frege, Gottlob (1879). *Begriffsschrift*. In *From Frege to Gödel*, J. van Heijenoort (ed.) (1967), 1-82. Cambridge, MA: Harvard University Press.

Gentzen, Gerhard (1934). Untersuchungen über das logische Schließen [Investigations into logical deduction]. In *The Collected Papers of Gerhard Gentzen*, M. E. Szabo (ed. and translator), (1969), 68-131. Amsterdam: North-Holland Publishing Co.

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

Henkin, Leon (1961). Some remarks on infinitely long formulas. In *Infinitistic Methods*, (Proceedings of symposium on foundations of mathematics), 176-183. London: Pergamon Press.

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.

Hintikka, Jaakko (1973). *Logic, Language Games, and Information*. Oxford: Clarendon Press.

ISO/IEC (1996). *Extended BNF*, (IS 14977). Geneva: International Organisation for Standardisation.

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

Majumdar, Arun K., and John F. Sowa (2009). Two paradigms are better than one, and multiple paradigms are even better. In *Proceedings of ICCS 2009*, (LNAI 5662), S. Rudolph, F. Dau, and S.O. Kuznetsov (eds.), 32-47. Berlin: Springer.

Ockham, William of (1488 [1323]). *Summa Logicae*. Paris: Johannes Higman. (The edition owned by C. S. Peirce.)

Peano, Giuseppe (1889). *Aritmetices principia nova methoda exposita* [Principles of arithmetic presented by a new method]. Torino: Bocca.

Peirce, Charles Sanders (1869). Grounds of validity of the laws of logic. *Journal of Speculative Philosophy* 2: 193-208. http://www.peirce.org/writings/p41.html (accessed 15 November 2009).

Peirce, Charles Sanders (1878). How to Make Our Ideas Clear. *Popular Science Monthly* 12: 286-302. http://www.peirce.org/writings/p119.html (accessed 15 November 2009).

Peirce, Charles Sanders (1880). On the algebra of logic. *American Journal of Mathematics* 3: 15-57.

Peirce, Charles Sanders (1882). Letter to O. H. Mitchell. In *Writings of Charles S. Peirce*, (1982-1993), 4: 394-399. Bloomington: Indiana University Press.

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 (1909). Manuscript 514. Transcribed by Michel Balat with commentary by J. F. Sowa. http://www.jfsowa.com/peirce/ms514.htm (accessed 15 November 2009).

Peirce, Charles Sanders (1931-1958 CP). *Collected Papers of C. S. Peirce*, C. Hartshorne, P. Weiss, and A. Burks (eds.), 8 vols., 1931-1958. Cambridge, MA: Harvard University Press.

Pietarinen, Ahti-Veikko (2006). *Signs of Logic: Peircean Themes on the Philosophy of Language, Games, and Communication*. Dordrecht: Springer. *Semiotica*

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

Robinson, J. Alan (1965). A machine oriented logic based on the resolution principle. *Journal of the ACM* 12: 23-41.

Simons, Peter (2004). Judging correctly: Brentano and the reform of elementary logic. In *The Cambridge Companion to Brentano*, D. Jacquette (ed.), 45-65. Cambridge: Cambridge University Press, Cambridge.

Sowa, John F. (1992 [1987]). Semantic networks. In *Encyclopedia of Artificial Intelligence,*, S. C. Shapiro (ed.). New York: Wiley.

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

Stewart, John (1996). Theorem Proving Using Existential Graphs, MS Thesis, Computer and Information Science. Santa Cruz: University of California.

Tarski, Alfred (1933). Pojęcie prawdy w językach nauk dedukcynych [The concept of truth in formalized languages]. In *Logic, Semantics, Metamathematics*, A. Tarski (1982), 2nd ed, 152-278. Indianapolis: Hackett.

Tarski, Alfred (1936). Über den Begriff der logischen Folgerung [On the concept of logical consequence]. In *Logic, Semantics, Metamathematics*, A. Tarski (1982), 2nd ed, 409-420. Indianapolis: Hackett.

Whitehead, Alfred North, and Bertrand Russell (1925 [1910]) *Principia Mathematica*, 2nd ed. Cambridge: Cambridge University Press.