EUGENIA BAHIT

HYPOTHETICAL SYLLOGISMS TO SECURE INFORMATION

A brief introduction to formal logic applied to information security

EBRC™ by B&B Group Publisher

WHAT ARE SYLLOGISMS?

A *syllogism* is a logical reasoning procedure. The most classic types of syllogism are **categorical** and **hypothetical** syllogisms. Both types have three components:

1. The first statement, called the *premise*.
2. A second statement, called the *proposition*.
3. And a final statement, called the *conclusion*.

The difference between categorical and hypothetical syllogisms lies in the first statement, the premise. While the premise of a hypothetical syllogism is a conditional statement (also called the hypothetical statement), the premise of a categorical syllogism is categorical. Hence, a hypothetical syllogism can be defined as a logical reasoning process where the premise (the first component) is a conditional statement.

In science, the hypothetical syllogism is an essential reasoning procedure because it forms the basis of the ***hypothetico-deductive method*** — also called the H-D method — used to construct the hypotheses of a scientific theory.

THE HYPOTHETICAL SYLLOGISM AND ITS LOGIC RULES

A conditional statement has the form: if *p*, then *q*, where *p* is called the *antecedent* and *q is called* the *consequent*.

In the following example, *John is tall* is the antecedent and *Jane is short*, the consequent:

If John is tall, then Jane is short.

To form a hypothetical syllogism, it is necessary to have a *conditional* statement followed by a *categorical premise*, and finished by another categorical statement called the *conclusion*.

The categorical premise can either be the antecedent or the denial of the consequent. Thus, in the previous example, the possibilities are:

1. John is tall, or
2. Jane is not short.

This is because there are two rules or ways to form a hypothetical syllogism:

1. **Affirming the antecedent,** also called *modus ponens*.
2. **Denying the consequent**, also called *modus tollens*.

When the antecedent is affirmed, the consequent must be affirmed in the conclusion:

If John is tall, then Jane is short.
John ***is*** *tall.*
Then, Jane ***is*** *short.*

And when the consequent is denied, the antecedent must be denied in the conclusion, such that:

If John is tall, then Jane is short.
Jane ***is not*** *short.*
Then, John ***is not*** *tall.*

AVOIDING FALLACIES

The evident question might be why it is not possible to deny the antecedent or affirm the consequent in the proposition. The short answer is to avoid fallacies. The more complex answer is that logic and reasoning methods have rules, and in the realm of logic, rules are necessary to prevent *reasoning errors*.

In psychology, reasoning errors are known as *thinking errors*. This is due to the difference between *thinking* and *reasoning*. While thinking involves everything a person can think of (and, for the most part, believe in), reasoning has a limit. In that sense, thinking is infinite while reasoning is finite. Even though it is possible to use the conclusion of one argument as a premise for another, each reasoning process requires an endpoint. A logical argument must always have a conclusion.

Thinking errors arise from *cognitive biases*, which are beliefs that people hold based on previous experiences, political ideologies, and social and cultural factors, among other influences.

Cognitive biases interfere with the natural reasoning process, resulting in a thinking error.

Precisely, a *fallacy* is a psychologically persuasive argument that uses general thinking as if it were logical reasoning. There are two types of formal fallacies related to hypothetical syllogisms: denying the antecedent (or fallacy of the inverse) and affirming the consequent (or fallacy of the converse).

It is simple to demonstrate the reasoning error with an example. When the cost of petroleum increases, it also tends to increase the price of public transport tickets, so that:

If the cost of petroleum increases, then the price of public transport tickets increases.

By **forgetting the reasoning rules,** it is possible to commit the following **reasoning error:**

If the cost of petroleum increases, then the price of public transport tickets increases too.
⊗ The cost of petroleum does not increase.
⊗ Then, the price of public transport tickets does not increase.

However, what happens if the price of public transport tickets increases, but the cost of petroleum does not? The conclusion would be false because the premise only addresses the consequences of increasing petroleum costs, not the causes of the increase in public transport ticket prices.

Thus, the hypothetical syllogism is limited. It only allows us to know two things:

1. What consequences will occur if the antecedent is true.
2. What cause can be rejected if the supposed consequence is not true.

A general principle of logical reasoning is that **it is not possible to affirm more than what has been affirmed in the premise.** Thus, by following two simple rules (affirming the antecedent to affirm the consequent and denying the consequent to deny the antecedent), fallacies are avoided.

THE HYPOTHETICAL SYLLOGISM IN SOFTWARE SECURITY

These concepts (hypothetical syllogism and reasoning rules) are of interest to software security systems because making categorical definitions of information makes it possible to establish a hypothesis for each one to validate the data. For example, a categorical definition of a product price could be like this:

A price is a real number greater or equal than 0.01 and lower or equal than 9999.99.

The previous sentence is categorical because its form is "*S is P*". It has a subject (formally called the *subject term*), followed by a "to be" verb (linguistically called a linking verb), and finished by a predicate (formally called the *predicate term*). There are four forms of categorical propositions:

General forms:

(A) Every S is P.
(E) No S is P.

Particular forms:

(I) Some S is P.
(O) Some S is not P.

Forms *A* and *E* are general forms, while forms *I* and *O* are particular forms. **For software security, the A-form is chosen.**

A categorical definition can be formally described using mathematical logic. For example, for the sentence: *A price is a real number greater than or equal to 0.01 and less than or equal to 9999.99*, the formal definition would be:

A price is a real number greater than or equal to 0.01 and less than or equal to 9999.99, such that:

$((p \in R) \wedge (p \geq 0.01 \wedge p \leq 9999.99))$

where *p* is the price.

In this way, it is possible to establish a hypothesis to validate a variable by stating that:

If p is a valid price, then p is a real number greater than or equal to 0.01 and lower than or equal to 9999.99.

There are only two ways to validate *p*:

1. Affirming that *p is a valid price*, but this is a paradox because it is trying to validate *p* precisely.
2. Denying that *p is a real number greater than or equal to 0.01 and lower than or equal to 9999.99*. And this is the unique feasible option.

So, by following the second rule, it is possible to infer that:

$$\neg((p \in R) \wedge (p \geq 0.01 \wedge p \leq 9999.99)) \Rightarrow \neg p$$

In order to validate *p*, it is necessary to have a function *f* of *p* that receives *p*, validates *p*, and returns a boolean value, either *true* or *false*. This function would be as follows:

$$f(p) \rightarrow \neg((p \in R) \wedge (p \geq 0.01 \wedge p \leq 9999.99)) \Rightarrow \neg p$$

The above statement can be translated into source code literally.

Some examples are shown below.

```
def is_a_price(p):
    term_1 = (type(p) == float)
    statement = (term_1 and (p >= 0.01 and p <= 9999.99))
    if not statement: return False
    return True
```

PYTHON CODE

```
function is_a_price($p) {
    $term_1 = (gettype($p) == 'double');
    $statement = ($term_1 and ($p >= 0.01 and $p <= 9999.99));
    if(!$statement) return False;
    return True;
}
```

PHP CODE

The output of the above function executed in the Python shell would look like:

```
>>> is_a_price(1)
False
>>> is_a_price(1.1)
True
>>> is_a_price('1.1')
False
>>> is_a_price({1.1})
False

                PYTHON OUTPUT
```

The output of the above function executed in the PHP shell would show:

```
php > print is_a_price(1);
php > print is_a_price(1.1);
1
php > print is_a_price('1.1');
php > print is_a_price([1.1]);

                   PHP OUTPUT
```

It is possible to observe that when the variable *p* does not have a suitable data type, the AND operation is not entirely executed. This is because, in such cases, it would throw an exception, as ‘1.1’ (and the set containing the number 1.1) cannot be managed as a number. Therefore, the language must obtain the data type to ensure the rest of the conditional can be executed without risk.