6.1.4. Abstracts

When we recognize an equation, we know immediately that it is a two-place predication; but, in other cases, identifying the places of a predicate is one of the chief tasks in analyzing a predication. A full analysis of a predication will identify one place in the predicate for each individual term appearing in the predication (more precisely, for each occurrence of an individual term satisfying the requirement referential transparency). When giving an analysis of an English sentence, it is natural also to regard the order of the places of the predicate we identify as being the same as the order in which the terms filling them appear in the English sentences. But this way of associating places of a predicate with individual terms in its English output is not required by the concept of a predicate. And, even though we will not make use of the freedom to depart from it in our analyses, we will want to allow intentional interpretations of symbolic predications to have full freedom in this regard.

For this reason, we need notation for predicates that will allow us to specify an order for the places of a predicate that is different from the order of blanks they correspond to and that will allow us to associate a given place with more than one blank. What we will use is an extension of the ordinary algebraic use of variables. It is a simple idea that was used by Frege but it was first studied extensively by the American logician Alonzo Church (1903-1995) in the 1930s, and it is his notation for it that has become standard. The usual form of definition for a function—for example,

$$f(x, y) = x^2 + 3xy + 1$$

gives a name to the function and uses a variable or variables to indicate the input values, with the output specified by an algebraic formula. Such definition might be read *f is the function which, when given input x and y, yields the output* $x^2 + 3xy + 1$. Church's notation, the notation of ***lambda abstraction***, provides a symbolic version of the definite description following *is* in the English definition above. Using this notation, the symbolic definition could be written as

$$f = \lambda xy\ (x^2 + 3xy + 1)$$

That is, $\lambda xy\ (x^2 + 3xy + 1)$ can be read as *the function which, when given input x and y, yields the output* $x^2 + 3xy + 1$. The notation of lambda abstraction thus describes a function without introducing a name for it. This idea has been important in the development of computer programming languages and, in that context, the right-hand side of the second equation taken by itself would now often be described as an "anonymous function"; so, when it is expressed in the notation of lambda abstraction, the defining equation specifies a function anonymously and then assigns it the name "f."

Since the variables *x* and *y* might appear in any order and any number of times in the expression specifying the output, this sort of notation provides the kind of flexibility we want in specifying predicates. For example, we can write

λxy (y *told* x *about* y)

for a predicate that, when given the input *Ann* and *Bill* yields the output *Bill told Ann about Bill*—or, more idiomatically, *Bill told Ann about himself*. As another example, note that the fullest and most natural analysis of this output sentence would instead see it as the output of the predicate

λxyz (x *told* y *about* z)

when it is given as its input the names *Bill*, *Ann*, and *Bill* again (where the second "Bill" is the same name again and not merely the same letters; that is, it is not the name of another person).

We will refer to such an abstract as an ***abstract*** and, more specifically as a ***predicate abstract*** when its output is a sentence. The general form of an abstract with *n* places is

$$\lambda x_1 \ldots x_n \quad (\ldots x_1 \ldots x_n \ldots)$$

λ-operator *body*

It has of two parts, a ***lambda operator*** consisting of the letter λ followed by a list of *n* variables and, as its ***body***, an expression formed drawing on these variables and other vocabulary. The variables need not actually appear in the body (to allow for lambda abstracts that achieve the effect of definitions like *f(x) = 2*); when they do appear, their occurrences in the body are said to be ***bound*** to the lambda operator. When the abstract is a predicate abstract, it might be read as

the attribute that $x_1 \ldots x_n$ *have when it is true that* $\ldots x_1 \ldots x_n \ldots$

We might take an abbreviated version of this reading as English notation for predicate abstracts:

the attribute of $x_1 \ldots x_n$ that $\ldots x_1 \ldots x_n \ldots$

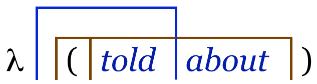
In case of the first predicate abstract above, we would have the following symbolic form, English reading, and English notation:

λxy (y *told* x *about* y)
the attribute that x *and* y *have when it is true that* y *told* x *about* y
the attribute of x and y that y *told* x *about* y

The English notation for abstracts is further from standard English than was the English notation we used for connectives and we will use it less often.

Variables have the grammatical status of individual terms but have no reference values. Until it is bound to a lambda operator, a variable is little more than a different way of marking a blank in a sentence. It does more than a simple line only because it indicates that the blank is one that could be linked to the place of a predicate. When it is bound, a variable functions more actively, but its function is merely to mark a correspondence between the place of a predicate and a blank in a sentence. Because of this, an older terminology referred to bound variables as “apparent variables.” And a less convenient but clearer notation would replace these variables by a more direct indication of the correspondence that they mark—e.g., by lines linking places after the initial λ with locations in the body, as in the following alternative to the first example above:

λxy (y *told* x *about* y)



In the latter diagram, lines show where occurrences of the terms serving as input should be placed in the body in order to form the output. In the lambda abstract, the same thing is indicated by the correspondence between the variables in the lambda operator and the variables marking blanks in the body.

Because bound variables only mark a correspondence between locations in the λ-operator and the body of the abstract, the bound variables of different abstracts have no connection with one another. This means that, for example, the following abstracts express the same predicate:

λxy (y *told* x *about* y)
λyz (z *told* y *about* z)

Each says that for any input terms τ and υ (in that order), the output sentence should be υ *told* τ *about* υ.

We will refer to as ***alphabetic variants*** expressions, like the two abstracts above, that differ only in the particular variables they use to link a lambda operator to places in the body of an abstract. Notice that, although the variable y appears in both, it would be replaced by a different one of the input terms in each case. Because the identity and difference of variables matters only for establishing links within an abstract, there is no connection between occurrences of a variable in different abstracts.

English has devices which function like bound variables. In the general case, an abstract might be stated fully in English as follows:

$$\lambda x_1 \ldots x_n (\ldots x_1 \ldots x_n \ldots)$$

the attribute that n things have when it is true that … the first … the nth …

In this form of words, the "*n* things" are understood to be given as a list of (not necessarily distinct) things of length *n* and expressions like *the first*, *the second*, and so on, refer back to locations in this list. The description of the attribute tells when it is possessed by any such list of things, so no definite list of things is in question; and the expressions *the first*, etc., that refer back to list locations make no definite reference outside the sentence. In short, expressions like *the first* function here like pronouns. This may be clearer if we consider the case of a one-place predicate abstract along with a comparable English expression:

λx (*Tom bought* x)

the property that a thing has when it is true that Tom bought it

The English fills the blank marked by x in the body of the abstract with a pronoun *it* that has *a thing* as its antecedent. Since *a thing* makes no definite reference, neither does the pronoun; the pronoun "refers back" to its antecedent only in the sense that their references are linked in their indefiniteness: they are not indefinite in independent ways. The general moral is that the variables used in abstracts are like pronouns, and the lambda operators are like their antecedents. You should not expect variables in the scope of different lambda operators to be linked in their reference any more than you would expect this of pronouns with different antecedents.

It is sometimes useful to consider the body of an abstract by itself. Just as a variable is grammatically like an individual term but does not have a reference value (not even the nil value), an expression like

x *told* y *about* z

that contains unbound variables is grammatically like a sentence but does not say anything. It is merely a sentence with blanks that might correspond to places of a predicate. The term ***formula*** is used for any expression that is grammatically like a sentence, with the term "***sentence***" reserved for formulas all of whose variables are bound (to abstracts within the sentence). Since all formulas are grammatically like sentences, the grammatical vocabulary applied to "sentences" in previous chapters applies to all formulas. In particular, formulas can be built from formulas by use of connectives, so formulas can be compound and have components.

In particular, we can speak of an ***atomic formula***. Now that we analyze sentences and other formulas into components like predicates and individual terms, the atomic formulas will no longer be simply the unanalyzed sentences though those will still count as atomic. We will also count as atomic any predication. Although predications are compound and can even have formulas as components (though not as immediate components), their role in derivations is sufficiently analogous to that of unanalyzed sentences for it to make sense to put them both in the same category. There is a way of building the analogy into our syntactic categories: an unanalyzed sentence can be thought of as a ***zero-place predicate***, one that requires no input to yield a sentence as output.