Chapter 1

CLASSICAL LOGIC—SYNTAX

1. Terms and Formulas

The formulation of a higher-order logic allows some freedom—there are certain places where choices can be made. Several of these choices produce equivalent results. Before getting to the formal machinery, I informally set out my decisions on these matters. Other treatments may make different choices, but ultimately it is largely a matter of convenience that is involved.

Often classical first-order logic is formulated with a rich variety of terms, built up from constant symbols and variables using function symbols. Since higher-order constructs are already complicated, I have decided to have constant symbols but not function symbols. If necessary for some purpose, it is not a major issue to add them—doing so yields a conservative extension.

Higher-order logic can be formulated with or without explicit abstraction machinery. Speaking informally, one wants to make sure that every formula specifies a class, but there are two ways of making this happen. One is to assume *comprehension axioms*, formulas of the general form:

$$(\exists X)[X(x_1, \ldots, x_n) \equiv \varphi(x_1, \ldots, x_n)].$$

where $\varphi(x_1, \ldots, x_n)$ is a formula with free variables as indicated. Such axioms ensure that to each formula corresponds an 'object.' The other approach is to elaborate the term-forming machinery, so that there is an explicit *name* for the object specified by a formula φ. This involves *predicate abstraction*, or *λ-abstraction*:

$$\langle \lambda x_1, \ldots, x_n . \varphi(x_1, \ldots, x_n) \rangle.$$

The two approaches are equivalent in a direct way. I have chosen to use explicit abstracts for several reasons. First, axioms are not as natural when tableau systems are the proof machinery of choice. And second, predicate abstraction has already played a major role in earlier investigations of modal logic [FM98], and makes discussion of major issues considerably easier here.

Finally, one can characterize higher-order formulas more-or-less the way it is done in the first-order setting, taking quantifiers and connectives as "logical constants." This is the approach of [Sch60]. Alternatively, following [Chu40], one can think of quantifiers and connectives as constants *of* the language, which itself is formulated in lambda-calculus style. In this book I take the first approach, though one can make arguments for the second on grounds of elegance and economy. My justification is that doing things the way that has become standard for first-order logic will be less confusing to the reader.

Recently one further alternative has become available. In [Gil99, Gil01], Paul Gilmore has shown that by a relatively simple change, a system of classical higher-order logic can be developed allowing a controlled degree of *impredicativity*—typing rules can be relaxed to permit the formation of certain useful sentences that are not "legal" in the approach presented here. This, in turn, allows a more natural development of arithmetic in the higher-order setting. I do not follow Gilmore's approach here, but I recommend it for study. Much of what I develop carries over quite directly.

So these are my choices: no function symbols, explicit predicate abstraction, quantifiers and connectives as in the first-order setting, and no impredicativity. With this out of the way I can begin presenting the formal syntactical machinery.

In first-order logic, relation symbols have an *arity*—some are one-place, some are two-place, and so on. In higher-order logic this simple idea gets replaced by a *typing* mechanism, which is considerably more complex. Terms, and certain other items, are assigned types, and rules of formation make use of these types to ensure that things fit together properly. I begin by saying what the types are.

DEFINITION 1.1 (TYPE) *0 is a type. If $t_1, \ldots, t_n$ are types, $\langle t_1, \ldots, t_n \rangle$ is a type. I generally use t, t_1, t_2, t', etc. to represent types.*

An object of type 0 is intended to be a ground-level object—it corresponds to the designation of a constant symbol or variable in standard first-order logic. An object of type $\langle t_1, \ldots, t_n \rangle$ is a predicate that takes n arguments, of types $t_1, \ldots, t_n$ respectively. Thus a constant symbol of type $\langle 0, 0, 0 \rangle$, say, would be called a three-place relation symbol in

standard first-order logic—it applies to three ground-level arguments. But now we can have relation symbols of types such as $\langle\langle 0\rangle, \langle 0, 0\rangle, 0\rangle$, to which nothing in first-order logic corresponds.

DEFINITION 1.2 ($L(C)$) *Let C be a set of constant symbols with a type associated to each, containing at least an equality symbol $=^{\langle t,t\rangle}$ for each type t. I denote the classical higher-order language built up from C by $L(C)$. The rest of this section amounts to the formal characterization of $L(C)$.*

For each type t I assume there are infinitely many variable symbols of that type. I generally use letters from the beginning of the Greek alphabet to represent variables, with the type written as a superscript: α^t, β^t, γ^t, ... Likewise I generally use letters from the uppercase Latin alphabet as constant symbols, again with the type written as a superscript: A^t, B^t, C^t, D^t, ... As noted, equality is primitive, so for each type t there is a constant symbol $=^{\langle t,t\rangle}$ of type $\langle t, t\rangle$. Often types can be inferred from context, and so superscripts will be omitted where possible, in the interests of uncluttered notation.

Sometimes it is helpful to refer to the *order* of a term or formula—first-order, second-order, and so on. It is simplest to define this terminology first for types themselves.

DEFINITION 1.3 (ORDER) *Type 0 is of* order *0. Type $\langle t_1, \ldots, t_n\rangle$ has as its* order *the maximum of the orders of t_1, ..., t_n, plus one.*

Thus $\langle 0, 0\rangle$ is of order 1, or *first-order*. Likewise $\langle 0, \langle 0, 0\rangle\rangle$ is of order 2, or *second-order*. Types will play the fundamental role, but order provides a convenient way of referring to the maximum complexity of some construct. When I talk about the order of a constant or variable, I mean the order of its type. Likewise once formulas are defined, I may refer to the order of the formula, by which I mean the highest order of a typed part of it.

Next I define the class of formulas, and their free variables. This definition is more complex than the corresponding first-order version because the notion of term cannot be defined first; both term and formula must be defined together. And to define both, I need the auxiliary notion of predicate abstract which is, itself, part of a mutual recursion involving Definitions 1.4, 1.5, and 1.6.

DEFINITION 1.4 (PREDICATE ABSTRACT OF $L(C)$) *Let Φ be a formula of $L(C)$ and α_1, ..., α_n be a sequence of distinct variables of types t_1, ..., t_n respectively. $\langle\lambda\alpha_1, \ldots, \alpha_n.\Phi\rangle$ is a* predicate abstract *of*

$L(C)$. Its type is $\langle t_1, \ldots, t_n \rangle$, and its free variable occurrences are the free variable occurrences in the formula Φ, except for occurrences of the variables $\alpha_1, \ldots, \alpha_n$.

DEFINITION 1.5 (TERM OF $L(C)$) *Terms of each type are characterized as follows.*

1 *A constant symbol of $L(C)$ or variable is a term of $L(C)$. If it is a constant symbol, it has no free variable occurrences. If it is a variable, it has one free variable occurrence, itself.*

2 *A predicate abstract of $L(C)$ is a term of $L(C)$. Its free variable occurrences were defined above.*

τ is used, with and without subscripts, to stand for terms.

DEFINITION 1.6 (FORMULA OF $L(C)$) *The notion of* formula *is given as follows.*

1 *If τ is a term of type $\langle t_1, \ldots, t_n \rangle$, and $\tau_1, \ldots, \tau_n$ is a sequence of terms of types $t_1, \ldots, t_n$ respectively, then $\tau(\tau_1, \ldots, \tau_n)$ is a formula (atomic) of $L(C)$. The free variable occurrences in it are the free variable occurrences of $\tau, \tau_1, \ldots, \tau_n$.*

2 *If Φ is a formula of $L(C)$ so is $\neg\Phi$. The free variable occurrences of $\neg\Phi$ are those of Φ.*

3 *If Φ and Ψ are formulas of $L(C)$ so is $(\Phi \wedge \Psi)$. The free variable occurrences of $(\Phi \wedge \Psi)$ are those of Φ together with those of Ψ.*

4 *If Φ is a formula of $L(C)$ and α is a variable then $(\forall\alpha)\Phi$ is a formula of $L(C)$. The free variable occurrences of $(\forall\alpha)\Phi$ are those of Φ, except for occurrences of α.*

EXAMPLE 1.7 Suppose $\alpha^{\langle 0,0\rangle}$ is a variable of type $\langle 0,0\rangle$ (and so first-order), β^0 is a variable of type 0, and $\gamma^{\langle\langle 0,0\rangle,0\rangle}$ is a variable of type $\langle\langle 0,0\rangle,0\rangle$ (second-order). Both β^0 and $\gamma^{\langle\langle 0,0\rangle,0\rangle}$ are terms. Then the expression $\gamma^{\langle\langle 0,0\rangle,0\rangle}(\alpha^{\langle 0,0\rangle}, \beta^0)$ is an atomic formula. Generally I will write the simpler looking $\gamma(\alpha, \beta)$, and give the information contained in the superscripts in a separate description. Since this atomic formula contains a variable γ of order 2, it is referred to as a second-order atomic formula.

DEFINITION 1.8 (SENTENCE) *A formula with no free variables is a* sentence.

One can think of $\vee$, $\supset$, $\equiv$, and $\exists$ as defined symbols, with their usual definitions. But sometimes it is convenient to take them as primitive—I do whatever is most useful at the time. Also square and curly parentheses are used, as well as the official round ones, to aid readability. And finally, I write the equality symbol in infix position, following standard convention. Thus, for example, I write $(\alpha^t =^{\langle t,t\rangle} \beta^t)$ in place of $=^{\langle t,t\rangle}(\alpha^t, \beta^t)$.

Several examples involving just first and second-order notions will be considered, so a few special alphabets are introduced informally, to make reading the examples a little easier.

Order	Constants	Variables
0	$a, b, c, \ldots$	$x, y, z, \ldots$
1	$A, B, C, \ldots$	$X, Y, Z, \ldots$
2	$\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$	$\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \ldots$

EXAMPLE 1.9 For this example I give explicit type information (in superscripts), until the end of the example. After this I omit the superscripts, and say in English what is needed to restore them.

Suppose x^0, $X^{\langle 0\rangle}$, and $\mathcal{X}^{\langle\langle 0\rangle\rangle}$ are variables (the first is of order 0, the second is of order 1, and the third is of order 2). Also suppose $\mathcal{P}^{\langle\langle 0\rangle\rangle}$ and g^0 are constant symbols of $L(C)$ (the first is of order 2 and the second is of order 0).

1 Both $\mathcal{X}^{\langle\langle 0\rangle\rangle}(X^{\langle 0\rangle})$ and $X^{\langle 0\rangle}(x^0)$ are atomic formulas. All variables present have free occurrences.

2 $\langle\lambda\mathcal{X}^{\langle\langle 0\rangle\rangle}.\mathcal{X}^{\langle\langle 0\rangle\rangle}(X^{\langle 0\rangle})\rangle$ is a predicate abstract, of type $\langle\langle\langle 0\rangle\rangle\rangle$. Only the occurrence of $X^{\langle 0\rangle}$ is free.

3 Since $\mathcal{P}^{\langle\langle 0\rangle\rangle}$ is of type $\langle\langle 0\rangle\rangle$, $\langle\lambda\mathcal{X}^{\langle\langle 0\rangle\rangle}.\mathcal{X}^{\langle\langle 0\rangle\rangle}(X^{\langle 0\rangle})\rangle(\mathcal{P}^{\langle\langle 0\rangle\rangle})$ is a formula. Only $X^{\langle 0\rangle}$ is free.

4 $[\langle\lambda\mathcal{X}^{\langle\langle 0\rangle\rangle}.\mathcal{X}^{\langle\langle 0\rangle\rangle}(X^{\langle 0\rangle})\rangle(\mathcal{P}^{\langle\langle 0\rangle\rangle}) \supset X^{\langle 0\rangle}(x^0)]$ is a formula. The only free variable occurrences are those of $X^{\langle 0\rangle}$ and x^0.

5 $(\forall X^{\langle 0\rangle})[\langle\lambda\mathcal{X}^{\langle\langle 0\rangle\rangle}.\mathcal{X}^{\langle\langle 0\rangle\rangle}(X^{\langle 0\rangle})\rangle(\mathcal{P}^{\langle\langle 0\rangle\rangle}) \supset X^{\langle 0\rangle}(x^0)]$ is a formula. The only free variable occurrence is that of x^0.

6 $\langle\lambda x^0.(\forall X^{\langle 0\rangle})[\langle\lambda\mathcal{X}^{\langle\langle 0\rangle\rangle}.\mathcal{X}^{\langle\langle 0\rangle\rangle}(X^{\langle 0\rangle})\rangle(\mathcal{P}^{\langle\langle 0\rangle\rangle}) \supset X^{\langle 0\rangle}(x^0)]\rangle$ is a predicate abstract. It has no free variable occurrences, and is of type $\langle 0\rangle$.

The type machinery is needed to guarantee that what is written is well-formed. Now that the exercise above has been gone through, I will display the predicate abstract without superscripts, as

$$\langle\lambda x.(\forall X)[\langle\lambda\mathcal{X}.\mathcal{X}(X)\rangle(\mathcal{P}) \supset X(x)]\rangle,$$

http://www.springer.com/978-1-4020-0604-3

Types, Tableaus, and Gödel's God
Fitting, M.
2002, XV, 181 p., Hardcover
ISBN: 978-1-4020-0604-3