2 Formal Languages – 2

Part 1 - Zero-Order Languages

1. Introduction

In the current part we develop a general account of a very simple class of formal languages – namely, zero-order languages. For these purposes, a zero-order language (ZOL) is a formal language in which every grammatical expression is either a sentence (category S) or a sentential connective (category Sk➔S). See Appendix A3 for a general description of categorial grammar.

2. The Meta-Language

In the current part, the meta-language (English+) is specified informally as follows.

(1) **Metalinguistic variables:** lower case Greek letters (with or without subscripts) are constants and variables that range over strings of symbols.

(2) **Quote-Notation:** the official name of a string of symbols of the object language is obtained by enclosing that string in single quotes.

(3) **String concatenation:** this operation is explicitly indicated – by the plus sign '+'. Algebraically speaking, the plus operation is associative, so parentheses are dropped [e.g., 'α+β+γ'].

(4) **Metaphysics:** As it turns out, our notation cannot distinguish between the letter 'a' and the word 'a'. Furthermore, our metaphysics supports this "confusion" — we treat strings *mereologically*; in particular, we regard every symbol as a string – an atomic string. Slogan: every "letter" is also a "word". Note: this is technically at odds with the usual set-theoretic approach to strings, according to which a string of symbols is a function from the natural numbers into the set of symbols, and so a symbol σ is technically distinct from its unit string $\langle\sigma\rangle$.

3. Prefix, Infix, and Postfix

In natural languages, and in mathematics, grammatical functors are implemented in a variety of ways. Among these, we can identify three *simple* formatting techniques.

1. Prefix Format

The functor goes in front of its argument(s).

Examples:
the mother of α
the square root of α
the sum of α (and) β
it is not true that α

2. Postfix Format

The functor goes in back of its argument(s).

Examples:
α is tall
α's mother
α's square [α^2]
α … not! [colloquial]

3. Infix Format

The functor goes between its arguments.

Examples:
α is taller than β
α respects β
$\alpha + \beta$
α and β

4. Prefix (Polish) Formatted ZOL's

In the prefix (Polish) implementation of zero-order languages: all connectives are placed in prefix notation. The following is the general account.

1. General Vocabulary

a. atomic formulas; a countable set;
b. connectives; every one is n-place for some n or other.

2. General Rules Of Formation

a. every atomic formula is a formula;
b. if c is an n-place connective, and $\phi_1,...,\phi_n$ are formulas, then $c+\phi_1+...+\phi_n$ is a formula;
c. nothing else is a formula.

5. Abstract Example of a Polish Formatted ZOL

The example presented here is abstract, in the sense that the actual symbols are not displayed, but only mentioned. The exact orthographic nature of the symbols is left completely unspecified.

1. Vocabulary (lexicon)

a. denumerable sequence $\langle\alpha_1,\alpha_2,...\rangle$ of atomic formulas
b. one 1-place connective, μ
c. one 2-place connective, κ

2. Rules of Formation

a. every atomic formula is a formula;
b. if ϕ is a formula, then $\mu+\phi$ is a formula;
c. if ϕ_1 and ϕ_2 are formulas, then $\kappa+\phi_1+\phi_2$ is a formula;
d. nothing else is a formula.

6. Concrete Example of a Polish Formatted ZOL

Here, the example is concrete; the orthographic symbols are specified exactly [notice all the single quotes].

1. vocabulary

'p', '♯', 'K', 'N'

2. rules of formation:

a. atomic formulas

1. 'p' is an atomic formula;
2. if ϕ is an atomic formula, then so is ϕ+'♯';
3. nothing else is an atomic formula.

b. connectives

1. 'N' is a 1-place connective;
2. 'K' is a 2-place connective.

c. formulas

1. every atomic formula is a formula;
2. if ϕ is a formula, then so is 'N'+ϕ;
3. if ϕ_1 and ϕ_2 are formulas, then so is 'K' + ϕ_1 + ϕ_2;
4. nothing else is a formula.

7. Postfix Formatted ZOL's

Prefix notation has a natural twin – postfix notation, also called 'Reverse Polish Notation'; the latter is in fact a registered trademark of Hewlett-Packard, who use this formatting scheme in their calculators.

Perhaps the best-known example of a postfix language is Postscript, which is a programming language employed by every printer that supports Postscript, which includes every "serious" laser printer, as well as every Macintosh compatible laser printer.

Theoretically speaking there is no significant difference between prefix and postfix languages. Accordingly, we will not be particularly concerned with postfix formatted languages.

8. Infix (Algebraic) Formatted ZOL's

Finally we consider infix notation, also called algebraic notation. [For example, the term 'Algebraic Logic' is used to refer to the formatting scheme used in calculators built by Texas Instruments.]

Now, prefix formatted languages have two advantages over infix formatted languages. (1) Prefix format works no matter what degree a connective is; infix format works only for 2-place connectives. (2) Prefix format does not require any punctuation [there are no parenthesis keys on an HP calculator!]; infix format requires parentheses for parsing. On the other hand, infix notation has a major advantage in readability.

The following is the general account of algebraically formatted zero-order languages.

1. Vocabulary

a. atomic formulas; a countable set;
b. connectives; each one has degree less than or equal to 2;
c. punctuation marks: '(', ')'

2. Rules of formation

a. every atomic formula is a formula;
b. if χ is a connective of degree 1, and ϕ is a formula, then so is $\chi+\phi$
c. if χ is a connective of degree 2, and ϕ_1, ϕ_2 are formulas, then so is '(' + ϕ_1 + χ + ϕ_2 + ')';
d. nothing else is a formula.

9. Abstract Example of an Algebraically Formatted ZOL

Once again, our first example is abstract, in the sense that no actual symbol is orthographically characterized, except for the parentheses; the symbols are mentioned, but not displayed.

1. vocabulary (lexicon)

a. denumerable sequence $\langle\alpha_1,\alpha_2,...\rangle$ of atomic formulas
b. one 1-place connective, μ
c. one 2-place connective, κ

2. rules of formation

a. every atomic formula is a formula;
b. if ϕ is a formula, then $\mu+\phi$ is a formula;
c. if ϕ_1 and ϕ_2 are formulas, then ‘(’ + ϕ_1 + κ + ϕ_2 + ‘)’ is a formula;
d. nothing else is a formula.

10. Concrete Example of an Algebraically Formatted ZOL

The following is a concrete example of a ZOL; the symbols are orthographically specified.

1. vocabulary

‘P’, ‘♯’, ‘→’, ‘∼’, ‘(’ , ‘)’

2. rules of formation

a. atomic formulas:

1. ‘P’ is an atomic formula;
2. if ϕ is an atomic formula, then so is ϕ+‘♯’;
3. nothing else is an atomic formula.

b. connectives:

1. ‘∼’ is a 1-place connective;
2. ‘→’ is a 2-place connective.

c. formulas:

1. every atomic formula is a formula;
2. if ϕ is a formula, then so is ‘∼’+ϕ;
3. if ϕ_1 and ϕ_2 are formulas, then so is ‘(’ + ϕ_1 + ‘→’ + ϕ_2 + ‘)’;
4. nothing else is a formula.

Part 2 - First-Order Languages and First-Order Theories

11. Introduction

In the present chapter, we examine a class of formal languages that has been studied in great detail over the last century – first-order languages.

Perhaps, we should call these formal languages *standard first-order languages*, since we adopt a considerably wider definition of first-order languages elsewhere in this book.

Even with this clarification, there is still room for disagreement; some logicians regard definite descriptions as officially part of first-order logic and languages; others do not. We do.

12. First-Order Languages

In Part 1 of this chapter, we use quote-plus notation. In this part, we instead employ neither quotes nor plus. In particular, we use the logical symbols ambiguously (in effect as names of themselves), and we use the implicit juxtaposition operator. [The reader is invited to translate our description into quote-plus notation.]

a. **logical vocabulary** (the same for every FOL)

variables
constants (also called parameters; these are optional)
quantifiers: $\forall$, $\exists$
connectives: $\sim$, $\rightarrow$, $\leftrightarrow$, &, $\vee$
description operator: ι
identity sign: =
parentheses: (,), [,]

b. **non-logical vocabulary** (different for different FOL's)

proper nouns
function signs; 0-place, 1-place, 2-place, etc.
predicates; 0-place, 1-place, 2-place, etc.

First-order languages form a kind, if you will. To specify a *particular* first-order language, one must specify its non-logical vocabulary — what are its proper nouns, what are its one-place function signs, etc. In this regard, notice that each (admissible) grammatical category can have any number of representatives (including zero!). See the many examples later in this part.

13. The Rules of Formation for First-Order Languages

These rules are schematic; to obtain the rules of formation for a specific first-order language, one needs to specify its non-logical vocabulary.

The rules are presented in terms of two primitive grammatical categories – singular terms and formulas, which are defined inductively as follows. Note once again that we use quote-less-plus-less notation.

a. singular terms

every variable is a singular term;
every constant (parameter) is a singular term;
every proper noun is a singular term;
if ϕ is an n-place function sign, and $\tau_1,...,\tau_n$ are singular terms,
then $\phi(\tau_1,...,\tau_n)$ is a singular term;
if $\mathbb{F}$ is a formula, and ν is a variable, then $\iota\nu\mathbb{F}$ is a singular term;
nothing else is a singular term.

b. atomic formulas

if $\mathbb{P}$ is an n-place predicate and $\tau_1,...,\tau_n$ are singular terms,
then $\mathbb{P}[\tau_1,...,\tau_n]$ is an atomic formula;
if τ_1 and τ_2 are singular terms,
then $[\tau_1=\tau_2]$ is an atomic formula;
nothing else is an atomic formula.

c. formulas

every atomic formula is a formula;
if $\mathbb{F}$ is a formula then so is: $\sim\mathbb{F}$
if $\mathbb{F}$ and $\mathbb{G}$ are formulas, then so are $(\mathbb{F}\rightarrow\mathbb{G})$, $(\mathbb{F}\vee\mathbb{G})$, $(\mathbb{F}\&\mathbb{G})$, $(\mathbb{F}\leftrightarrow\mathbb{G})$;
if $\mathbb{F}$ is a formula, and ν is a variable, then $\forall\nu\mathbb{F}$ and $\exists\nu\mathbb{F}$ are formulas;
nothing else is a formula.

Note all the punctuation marks (parentheses and square brackets) are officially part of first-order languages **as specified here**. Note, however, that most logicians characterize first-order languages without the square brackets for predicate application, and the round parentheses for function sign application.

14. First-Order Theories

In order to discuss examples of first-order languages, it is a good idea to consider them in context. Specifically, first-order languages are constructed to formalize first-order theories, which are constructed to formalize "naturally occurring" theories. For this reason, we discuss several examples of first-order theories, which involves discussing the underlying first-order languages.

Briefly, a formal theory $\mathcal{T}$ is specified in three stages.

1. One formally specifies the formal language $\mathcal{L}$ on which $\mathcal{T}$ is based.
2. One specifies the logical system S that governs $\mathcal{T}$.
3. One formally specifies which formulas of $\mathcal{L}$ are theorems/theses of $\mathcal{T}$.

These are inter-related. The language $\mathcal{L}$ must be a species of the kind of language that S governs. The underlying principle of all formal theories is then:

> The theses of $\mathcal{T}$ form a subset of formulas of $\mathcal{L}$ that is *logically closed* with respect to S.

What 'logically closed' means here is summarized as follows.

> if ϕ_1, ..., ϕ_m are all theses of $\mathcal{T}$,
> and α *follows from* ϕ_1, ..., ϕ_m, according to logical system S
> then α is also a thesis of $\mathcal{T}$.

Of particular interest in the current sub-chapter, of course, are first-order theories, which are formulated in first-order languages, and logically governed by first-order logic.

Here, another point of contention arises: what do we mean by 'first-order logic'. In this connection there are two plausible candidates – Classical First-Order Logic and Free First-Order Logic. The difference concerns how they handle referentially improper singular terms. In the case of Classical FOL, every singular term refers, even referentially improper singular terms. To quarantine these goof balls, we choose (more or less arbitrarily) an individual in the domain to which all referentially improper terms refer. In the case of Free FOL, referentially improper singular terms refer to nothing whatsoever. The advantage is referential clarity. The disadvantage is that extra axioms are often required to guarantee existence.

A first-order theory consists of a first-order language together with a specification of which formulas are theses (theorems). Ordinarily, the theorems of a formal theory are generated *axiomatically*. Specifically, first, a set of primitive theorems [called axioms] are presented; then, from these primitive theorems, all theorems are deduced using the deductive techniques of first-order logic. In order to illustrate these ideas, we consider a number of well-known examples.

15. Example 1: Peano Arithmetic

Arithmetic (i.e., the theory of natural numbers) was given its classical formulation by Peano (and Dedekind). Peano Arithmetic can be formulated in a first-order language with just three (primitive) non-logical symbols, which are the following.

Non-Logical Vocabulary

Symbol:	**Category:**	**Reading:**
0	proper noun	zero
N	one-place predicate	N[α] : α is a number
s	one-place function sign	s(α) : the (immediate) successor of α

Axioms

P1. N[0]
P2. $\forall x(N[x] \rightarrow N[s(x)]$
P3. $\sim\exists x[s(x)=0]$
P4. $\forall x\forall y(s(x)=s(y) \rightarrow x=y)$
P5. $\mathbb{F}[0/\nu] \,\&\, \forall x(N[x] \rightarrow .\, \mathbb{F}[x/\nu] \rightarrow \mathbb{F}[s(x)/\nu]) .\rightarrow \forall x(N[x] \rightarrow \mathbb{F}[x/\nu])$

Note: P5 is not an axiom, but rather an *axiom schema*; it is short for infinitely many axioms, one for each formula $\mathbb{F}$ and variable ν; $\mathbb{F}[\tau/\nu]$ is the formula that results when variable ν is replaced by term τ in all its free occurrences in $\mathbb{F}$ [all remaining free variables are understood as universally quantified]. '$\mathbb{F}$' is not part of the language of Peano Arithmetic — the object language; rather, it is a metalinguistic expression which stands for any formula of the (object) language of Peano Arithmetic. [[Axiom P5, which is the Principle of Mathematical Induction, is exceedingly important, and will be discussed in detail in Chapter 3.]]

The theorems (theses) of Peano Arithmetic are then all the logical consequences of the Peano Axioms. The usual laws of arithmetic are obtained by logic from the axioms, together with various definitions. These include definitions of the various numbers (1,2, etc.), the arithmetic functions (addition, multiplication, etc.), as well as the arithmetic predicates (even, odd, less than, greater than, etc.). [[The logical reduction of a theory consists of two components; on the one hand, one reduces the various concepts of the theory to a subset of concepts — the primitive concepts; on the other hand, one reduces the theses to a subset of theses — the axioms.]]

16. Example 2: Pure First-Order Logic

Before continuing with examples of specialized first-order languages, it is useful to note that the language used in elementary logic is also a first-order language, which might be called the *generic* first-order language. The associated "theory" is sometimes referred to as Pure First-Order Logic.

There are two prominent differences between Pure First-Order Logic, which is a *generic* first-order theory, and any *specific* theory, such as Arithmetic.

At the syntactic level, whereas Arithmetic only has three (primitive) non-logical symbols, the generic first-order language has infinitely many symbols of every grammatical category – there are infinitely many proper nouns, infinitely many one-place predicates, etc.

Although Arithmetic is syntactically poor, it is semantically rich, compared to pure first-order logic. On the one hand, pure first-order logic has no non-logical theses, but only logical theses (not surprisingly!). On the other hand, Arithmetic has non-logical theses, in addition to logical theses. For example, '1+2=1+2' is a logical thesis of Arithmetic, whereas '1+2=2+1' is a non-logical thesis of Arithmetic.

17. Example 3: Elementary Group Theory

There are many theories in mathematics, some more abstract than others. One of the abstract theories is Group Theory, the elementary component of which is a first-order theory.

Aside: Mathematicians sometimes refer to the first-order fragment of a theory as the elementary theory. The morpheme 'element' is suggestive of the intent; in the *element*ary theory, one "talks" exclusively about *elements*, rather than *sets* of elements. Formally speaking, this amounts to quantifying exclusively over elements. The more general theory quantifies over sets of elements, sets of sets of elements, and so forth.

The "additive formulation" of Elementary Group Theory may be formally specified as follows.

Non-Logical Vocabulary

Symbol:	**Category:**	**Reading:**
0	proper noun	zero
–	one-place function sign	$-\alpha$: the negative of α
+	two-place function sign	$\alpha+\beta$: α plus β

Axioms

G1. $\forall x \forall y \exists z[z = x+y]$
G2. $\forall x \exists y[y = -x]$
G3. $\forall x[x + 0 = x]$
G4. $\forall x[0 + x = x]$
G5. $\forall x[x + -x = 0]$
G6. $\forall x[-x + x = 0]$
G7. $\forall x \forall y \forall z[x + (y + z) = (x + y) + z]$

Note: g1 and g2 are required if the logical system is Free First-Order Logic; if the logical system is Classical First-Order Logic, then they are logical theses.

18. Example 4: Elementary Boolean Algebra

Another example of a first-order theory is Elementary Boolean Algebra, which may be formally specified as follows.

Non-Logical Vocabulary

Symbol:	**Category:**	**Reading:**
0	proper noun	zero
1	proper noun	one
–	one-place function sign	$-\alpha$: the complement of α
$\wedge$	two-place function sign	$\alpha \wedge \beta$: α meet β
$\vee$	two-place function sign	$\alpha \vee \beta$: α join β

Axioms

B1. $\exists y[y = -x]$
B2. $\exists z[z = x \wedge y]$
B3. $\exists z[z = x \vee y]$
B4. $x \wedge x = x$
B5. $x \wedge y = y \wedge x$
B6. $x \wedge (y \wedge z) = (x \wedge y) \wedge z$
B7. $x \vee x = x$
B8. $x \vee y = y \vee x$
B9. $x \vee (y \vee z) = (x \vee y) \vee z$
B10. $x \wedge (x \vee y) = x$
B11. $x \vee (x \wedge y) = x$
B12. $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
B13. $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
B14. $-(-x) = x$
B15. $x \wedge -x = 0$
B16. $x \vee -x = 1$
B17. $-(x \wedge y) = -x \vee -y$
B18. $-(x \vee y) = -x \wedge -y$

Note: b1-b3 are required if the logical system is Free First-Order Logic; if the logical system is Classical First-Order Logic, then they are logical theses. Also, b4-b18 are understood to be universally quantified over their free variables. Compare this with our presentation of Group Theory in the previous section.

19. Example 5: Classical Set Theory

Classical Set Theory, which is originally due to Cantor and Frege, may be very simply formalized as a first-order theory as follows.

Non-Logical Vocabulary

Symbol:	Category:	Reading:
$\in$	2-place predicate	$\alpha \in \beta$: α is an element of β

Axioms

A1. $\forall x \forall y[\forall z(z \in x \leftrightarrow z \in y) \rightarrow x=y]$
A2. $\exists x \forall y[y \in x \leftrightarrow \mathbb{F}]$
(An axiom schema; $\mathbb{F}$ is any formula in which y is not free)

Classical Set Theory is very powerful. Unfortunately, as first shown by Russell, it is inconsistent! For this reason, Modern Set Theory was formulated.

20. Example 6: Modern Set Theory

Modern Set Theory can also be formulated as a first-order language. The following are the principal axioms of what is usually called ZF Set Theory (after Zermelo and Fraenkel). Note that this is *pure* set theory, so the universe of discourse (domain) consists exclusively of sets, and so the quantifiers range over sets.

Non-Logical Vocabulary

$\in$	2-place predicate	$\alpha \in \beta$: α is an element of β

Axioms

A1. $\forall x \forall y[\forall z(z \in x \leftrightarrow z \in y) \rightarrow x=y]$
A2. $\exists x \sim \exists y[y \in x]$
*A3. $\forall x \exists y \forall z\{z \in y \leftrightarrow (z \in x \,\&\, \mathbb{F})\}$
A4. $\forall x \forall y \exists z \forall w(w \in z \leftrightarrow . w=x \vee w=y)$
A5. $\forall x \exists y \forall z[z \in y \leftrightarrow \exists w(w \in x \,\&\, z \in w)]$
A6. $\forall x \exists y \forall z[z \in y \leftrightarrow \forall w(w \in z \rightarrow w \in x)]$
*A7. $\forall \nu \exists! \upsilon \mathbb{F} \rightarrow \forall x \exists y \forall z\{z \in y \leftrightarrow \exists w(w \in x \,\&\, \mathbb{F}[w/\nu, z/\upsilon])\}$

* An axiom schema; $\mathbb{F}$ is any formula in which y is not free.

[Note: '$\exists!\upsilon$' means 'there is a unique υ such that']

Part 3 – Second-Order Languages

21. Introduction

Although this book is principally interested in first-order logic, it is useful to see examples of languages that are not first-order. For this reason, in the current part, we examine simple second-order

languages. These languages are "simple" in comparison to *general* second-order languages. In particular, these languages involve the following simplifications.

(1) they do not countenance lambda-abstraction;
(2) they do not countenance higher-order predicates.

22. The Syntax of Simple Second-Order Languages

First, the official syntax.

1. Logical Vocabulary

Name	**Instances**	**Category**
individual variables	x, y, z, etc.	V_0 [= N(v)]
individual constants	a, b, c, etc.	N
predicate variables	X^0, Y^0, Z^0, etc.	(∅➔S)(v) [≡ S(v)]
	X^1, Y^1, Z^1, etc.	(N➔S)(v)
	X^2, Y^2, Z^2, etc.	[(N➔S)2 ➔ S](v)
	X^3, Y^3, Z^3, etc.	[(N➔S)3 ➔ S](v)
	etc.	
predicate constants (optional)	A^0, B^0, B^0, etc.	∅➔S [≡ S]
	A^1, B^1, B^1, etc.	N ➔ S
	A^2, B^2, B^2, etc.	(N➔S)2 ➔ S
	A^3, B^3, B^3, etc.	(N➔S)3 ➔ S
	etc.	

SL connectives	∼, →, ↔, &, ∨	Sk ➔ S
first-order quantifiers	∀, ∃	V_0+S ➔ S
second-order quantifiers	∀, ∃	V_1+S ➔ S
description operator	ι	V_0+S ➔ N
identity sign	=	N2 ➔ S
parentheses	(,)	none

Note that the quantifiers are categorially ambiguous, as indicated in the category column. Note also the presence of subcategories V_0 (individual variables) and V^*_1 (predicate variables).

2. Non-Logical Vocabulary

proper nouns	N
first-order function signs (for each $k \geqslant 0$):	$N^k \rightarrow N$
first-order predicates (for each $k \geqslant 0$):	$N^k \rightarrow S$

Note: the *non-logical* vocabulary of a simple second-order language is restricted to first-order notions. There are no second-order function signs or predicates.

As with all formal languages, the non-logical vocabulary will the theory-specific. For example, the non-logical vocabulary of second-order arithmetic differs from the non-logical vocabulary of second-order set theory. Second-order languages form a kind. To specify a *particular* second-order language, one must specify its non-logical vocabulary — what specifically are the proper nouns, function signs, and predicates. In this regard, each (admissible) grammatical category can have any number of representatives (including zero).

3. Rules of Formation

The formation rules are schematic; to obtain the rules of formation for a specific second-order language, one must specify its non-logical vocabulary.

There are several categories of well-formed expressions, which are defined inductively as follows.

a. Singular Terms:

every individual variable/constant is a singular term;
every proper noun is a singular term;
if ϕ is an n-place function sign, and $\tau_1,...,\tau_n$ are singular terms, then $\phi(\tau_1,...,\tau_n)$ is a singular term;
if $\mathbb{F}$ is a formula, and ν is an individual variable, then $\iota\nu\mathbb{F}$ is a singular term;
nothing else is a singular term.

b. First-Order Predicates:

'=' is a first-order predicate of degree 2 [written in infix notation];
for each $n \geqslant 0$:
    every predicate variable/constant of degree n is a first-order predicate of degree n;
    every non-logical first-order predicate of degree n is a first-order predicate of degree n;
nothing else is a first-order predicate of degree n.

c. Atomic Formulas:

if $\mathbb{P}$ is a first-order n-place predicate and $\tau_1,...,\tau_n$ are singular terms,
    then $\mathbb{P}[\tau_1,...,\tau_n]$ is an atomic formula;
if τ_1 and τ_2 are singular terms,
    then $[\tau_1=\tau_2]$ is an atomic formula;
nothing else is an atomic formula.

d. Formulas:

every atomic formula is a formula;
if $\mathbb{F}$ is a formula, then so is: $\sim\mathbb{F}$;
if $\mathbb{F}$ and $\mathbb{G}$ are formulas, then so are: $(\mathbb{F}\rightarrow\mathbb{G})$, $(\mathbb{F}\vee\mathbb{G})$, $(\mathbb{F}\&\mathbb{G})$, $(\mathbb{F}\leftrightarrow\mathbb{G})$;
if $\mathbb{F}$ is a formula, and ν is an individual variable, then $\forall\nu\mathbb{F}$ and $\exists\nu\mathbb{F}$ are formulas;
if $\mathbb{F}$ is a formula, and V is a predicate variable, then $\forall V\mathbb{F}$ and $\exists V\mathbb{F}$ are formulas;
nothing else is a formula.

23. Example 1: Second-Order Arithmetic

Having examined the class of simple second-order languages, we consider two examples of theories that can be profitably formulated in such languages.

Probably the most famous second-order theory is Second-Order Peano Arithmetic. We have already seen the first-order formulation of Peano Arithmetic. The second-order formulation is quite similar.

First, the underlying language is based on precisely the same three primitive non-logical symbols.

Non-Logical Vocabulary

Symbol:	**Category:**	**Reading:**
0	proper noun	zero
N	one-place predicate	N[α] : α is a number
s	one-place function sign	s(α) : the (immediate) successor of α

On the other hand, the axioms of Second-Order Peano Arithmetic are given as follows.

P1. N[0]
P2. $\forall x(N[x] \rightarrow N[s(x)]$
P3. $\sim\exists x[s(x)=0]$
P4. $\forall x\forall y(s(x)=s(y) \rightarrow x=y)$
P5. $\forall X\{X[0] \,\&\, \forall y(N[y] \rightarrow .\, X[y] \rightarrow X[s(y)]) \;.\rightarrow \forall y(N[y] \rightarrow X[y])\}$

There is in fact not a great deal of difference between first-order and second-order arithmetic. The difference pertains exclusively to the Axiom of Induction (P5). In first-order arithmetic, P5 must be formulated as an axiom schema. This is not necessary in second-order arithmetic, because it can utilize second-order quantification. Notice that the quantifier '$\forall$X' quantifies into predicate position, something that is grammatically forbidden in a first-order theory.

24. Example 2: Second-Order Set Theory

Set theory can be formulated in a simple second-order language. First, the non-logical vocabulary is quite simple.

Symbol:	**Category:**	**Reading:**
$\in$	2-place predicate	$\alpha \in \beta$: α is an element of β

We consider two versions of set theory. Note that we are discussing *pure* set theory, so the universe of discourse (domain) consists exclusively of sets, and so the quantifiers range exclusively over sets.

1. Classical Set Theory

Classical Set Theory, which is originally due to Cantor and Frege, may be very simply formalized as a second-order theory based on the following two axioms.

A1. $\forall x \forall y[\forall z(z \in x \leftrightarrow z \in y) \rightarrow x=y]$
A2. $\forall X \exists y \forall z[z \in y \leftrightarrow Xz]$

As mentioned in the section on first-order set theory, classical set theory is inconsistent, which is why modern set theory was formulated.

2. Modern Set Theory

Modern Set Theory is usually formulated as a first-order theory, but it can be more succinctly formulated as a second-order theory. The following are the principal axioms of ZF Set Theory.

A1. $\forall x \forall y[\forall z(z \in x \leftrightarrow z \in y) \rightarrow x=y]$
A2. $\exists x \sim \exists y[y \in x]$
A3. $\forall X \forall x \exists y \forall z\{z \in y \leftrightarrow (z \in x \,\&\, Xz)\}$
A4. $\forall x \forall y \exists z \forall w(w \in z \leftrightarrow . \; w=x \vee w=y)$
A5. $\forall x \exists y \forall z[z \in y \leftrightarrow \exists w(w \in x \,\&\, z \in w)]$
A6. $\forall x \exists y \forall z[z \in y \leftrightarrow \forall w(w \in z \rightarrow w \in x)]$
A7. $\forall X\{\forall y \exists !z Xyz \rightarrow \forall x \exists y \forall z\{z \in y \leftrightarrow \exists w(w \in x \,\&\, Xwz)\}\}$
[Note: '$\exists !z$' means 'there is a unique z such that']

Part 4 – Categorial Languages

25. Introduction

So far we have described each formal language in a largely ad hoc manner. An alternative to this procedure is to employ general categorial grammar. In the previous sections, we have mentioned the various grammatical categories, but we have not really put them to use.

In the current part, we briefly describe the categorial approach to formal languages. Not only does this approach afford more generality, it enables us to describe formal languages for which no ad hoc procedure will work.

A general and detailed account of categorial grammar may be found in Appendix 3.

26. The Fundamental Categorial Rule

We have already seen examples of grammatical categories, in reference to the various functors. For example, a one-place predicate is a functor of category

$$N \rightarrow S$$

which means that it takes a single noun phrase (N) as input, and yields a sentence (S) as output. Similarly, a two-place function sign is a functor of category

$$N+N \rightarrow N \qquad \text{or:} \qquad N2 \rightarrow N$$

which means that it takes two noun phrases as input, and yields a noun phrase as output.

The two previous examples are first-order functors. There are higher-order functors also. For example, a predicate adverb is a functor of category

$$(N \rightarrow S) \rightarrow (N \rightarrow S)$$

which means it takes a predicate as input, and generates a predicate as output.

The examples so far are specific examples. The general case is given by the following general rule of formation for languages that are categorially specified.

Generic Formation Rule for Categorial Languages

if Φ is an expression of category $(K_1+\ldots+K_m) \rightarrow K_0$,
and $\varepsilon_1, \ldots, \varepsilon_m$ are expressions of category $K_1, \ldots, K_m$, respectively,
then $\Phi(\varepsilon_1, \ldots, \varepsilon_m)$ is an expression of category K_0.

Here, $\Phi(\varepsilon_1, \ldots, \varepsilon_m)$ is the result of "applying" functor Φ to expressions $\varepsilon_1, \ldots, \varepsilon_m$.

The above rule of formation is generic (general) in two senses.

(1) It doesn't specify which categories are in fact instantiated.
(2) It does not specify the orthographic details of functor application.

By 'orthographic details' we mean, for example, whether the functor is written in prefix, postfix, infix, or some other notation. etc.). Both of these will depend upon the specific language in question.

Given the general rule of formation, in order to specify a formal language under this rubric, we need merely specify the atomic symbols together with their categories. If we need orthographic specifics, then the details of functor application are also required.

In the next few sections, we examine how various (kinds of) languages can be specified categorially. We concentrate on languages not covered previously. [Exercise: Go back and rewrite the descriptions of the previous languages in pure categorial form.]

27. Sentential Modal Logic

Sentential Modal Logic provides an example of a zero-order language not already discussed. It is obtained from ordinary SL by adding modal operators – $\Box$ (necessarily) and $\Diamond$ (possibly).

Logical Vocabulary:

$\sim$, $\Box$, $\Diamond$	S➔S
$\rightarrow$, $\leftrightarrow$, &, $\vee$	S2➔S
Infinite list of the following:	
Sentential constants: P, Q, R, etc.	S

Examples of Formulas:

The actual formulas will depend upon functor implementation; if we pursue the usual implementation – infix (algebraic) format – we have the following sorts of formulas.

$\Box(P\rightarrow Q)\rightarrow(\Box P\rightarrow\Box Q)$
$\Box P \rightarrow (\Diamond Q \rightarrow \Diamond(P\&Q))$
$\Diamond\Box P \rightarrow \Box\Diamond P$

On the other hand, if we pursue a pure prefix implementation ("Polish notation"), then we have the following sorts of formulas.

$\rightarrow\Box\rightarrow PQ\rightarrow\Box P\Box Q$
$\rightarrow\Box P\rightarrow\Diamond Q\Diamond\&PQ$
$\rightarrow\Diamond\Box P\Box\Diamond P$

28. Quantified Sentential Logic

Quantified Sentential Logic is an odd little logic that has quantification, but no predication! One might call it second-order sentential logic, or one might call it absolutely minimal second-order logic. On the other hand, in an important sense it is not second-order (see Appendix 3 for definition of order).

Logical Vocabulary:

$\sim$	S➔S
$\rightarrow$, $\leftrightarrow$, &, $\vee$	S+S➔S
$\forall$, $\exists$	S(v)+S➔S

Infinite list of each of the following:

Sentential variables: X, Y, Z, etc.	S(v)
Sentential constants: P, Q, R, etc.	S

Examples of Formulas:

If we pursue the usual syntactic implementation of functor application, we have the following sorts of formulas.

$\forall X(X \rightarrow X)$
$\forall X(X \rightarrow \forall Y(Y \rightarrow X))$
$\exists X \forall Y(X \rightarrow Y)$

29. An Abstract Example

This example is abstract in the sense that English readings are not provided for any of the various grammatical expressions.

t	:	proper noun	:	N;
f( )	:	a one-place function sign	:	N➔N;
p[ ]	:	a two-place predicate	:	N2➔S;
c⟨ ⟩	:	a two-place connective	:	S2➔S;
s{ }	:	a one-place subnective	:	S➔N;

It is furthermore understand that every instance of functor application employs prefix format in addition the indicated category markers (and commas as necessary). The following are examples of well-formed expressions.

t	N
f(t)	N
p[t, t]	S
p[t, f(t)]	S
c⟨p[t, t], p[t, t]⟩	S
s{p[t, t]}	N

30. Generalized First-Order Languages

Ordinary First-Order Logic is severely limited grammatically; in particular, the only non-logical symbols it allows are proper nouns (N), function signs (Nk➔N), and predicates (Nk➔S). A more general logic can be obtained by enlarging the class of admissible categories to include every first-order category.

The general concept of *order* is presented in Appendix 3. For our current purposes, we simply offer the following special case – the definition of *zero-order*, and *first-order*.

> N is zero-order;
> S is zero-order;
> nothing else is zero-order;
>
> if K_1, …, K_m are zero-order, and K_0 is zero-order or first-order, then
> $(K_1+\ldots+K_m)\rightarrow K_0$ is first-order;
> nothing else is first-order.

A corresponding rough-and-ready account of first-order functor is given as follows.

> A *first-order* functor is a functor that takes no functor as input, and whose output is either a first-order functor or a primitive category (N or S).

For example, a functor of either of the following categories

$(N\rightarrow S)\rightarrow S$
$(N\rightarrow S)\rightarrow N$

is *not* first-order because in each case the input is a one-place predicate, which is a species of functor.

Examples of first-order functors are plentiful. For example, the simple functors are first-order functors.

$Sk\rightarrow S$	k-place sentential operators (connectives)
$Nk\rightarrow S$	k-place predicates
$Nk\rightarrow N$	k-place function signs
$Sk\rightarrow N$	k-place subnectives

So are the following inhomogeneous (mixed input) functors

$N+S\rightarrow S$	(example, α believes that $\mathbb{S}$)
$N+S\rightarrow N$	(examples?)

Furthermore, since these are all first-order, any functor that takes combinations of N and S as input, and generates one of these as output is also a first-order functor, including the following.

$N\rightarrow(N\rightarrow S)$
$N\rightarrow(S\rightarrow S)$
$N\rightarrow(S\rightarrow N)$

Indeed, given the inductive nature of the definition of *first-order*, the process of constructing first-order categories is never-ending. For example, the following is an infinite list of categories.

$N\rightarrow S$
$N\rightarrow(N\rightarrow S)$
$N\rightarrow(N\rightarrow(N\rightarrow S))$
$N\rightarrow(N\rightarrow(N\rightarrow(N\rightarrow S)))$
etc.

What this entails is that we cannot give explicit rules of formation for a generic generalized first-order language, because there are too many categories to consider. Instead, we have to rely on the categorial method to present the rules of formation.

31. The Syntax of Generalized First-Order Languages

1. Logical Vocabulary:

individual variables: x, y, z, etc.	N(v)
individual constants: a, b, c, etc.	N
sentential variables: X, Y, Z, etc.	S(v)
sentential constants: P, Q, R, etc.	N
individual quantifiers: $\forall$, $\exists$	N(v)+S➔S [or: N(v)➔(S➔S)]
sentential quantifiers: $\forall$, $\exists$	S(v)+S➔S [or: S(v)➔(S➔S)]

connectives:

$\sim$	S➔S
$\rightarrow$, $\leftrightarrow$, &, $\vee$	S+S➔S

general description operator: ι
for each category K, such that order(K) = 0: K(v)+S➔K [or: K(v)➔(S➔N)]

general identity operator: =
for each category K, such that order(K) = 0: K+K➔S

Optional Logical Items

general lambda-abstraction operator λ:
for each sequence K_1, ..., K_m of categories, such that order(K_i) = 0:

1 (predicate):	$[K_1(v)+...+K_m(v)+S]\rightarrow[K_1+...+K_m\rightarrow S]$
2 (function sign):	$[K_1(v)+...+K_m(v)+N]\rightarrow[K_1+...+K_m\rightarrow N]$

2. Non-Logical Vocabulary:

for each category K, such that order(K) $\leqslant$ 1:
zero or more non-logical expressions
of category K. K

3. Rules of Formation

Primitive Expressions:

Every expression of category N is a singular term;
Every expression of category S is a formula;

Derivative Expressions:

if Φ is an expression of category $(K_1, ..., K_m \rightarrow K_0)$,
and $\eta_1, ..., \eta_m$ are expressions of category $K_1, ..., K_m$, respectively,
then $\Phi(\eta_1, ..., \eta_m)$ is an expression of category K_0.
[As usual, the various functors are implemented in a language-specific manner].

32. An Example of A Generalized First-Order Theory

As before, a theory specifies three things (the first one often being presupposed).

(1) the underlying logico-linguistic system (first-order logic, second-order logic, etc.)
(2) the non-logical vocabulary (concepts)
(3) the non-logical axioms (principles)

The following is a fairly simple example of a theory underwritten by generalized first-order logic.

Non-Logical Vocabulary

Symbol:	**Category:**	**Reading:**
$\Box$	one-place connective	$\Box\mathbb{S}$: necessarily, $\mathbb{S}$
$\Diamond$	one-place connective	$\Diamond\mathbb{S}$: possibly, $\mathbb{S}$

Axioms:

(m1) $\forall X\{\Box X \rightarrow X\}$
(m2) $\forall X\{X \rightarrow \Diamond X\}$
(m3) $\forall X(\sim\Box X \leftrightarrow \Diamond \sim X\}$
(m4) $\forall X\{\sim\Diamond X \leftrightarrow \Box \sim X\}$
(m5) $\forall X\{\forall \nu \Box X \leftrightarrow \Box \forall \nu X\}$
(m6) $\forall X\{\exists \nu \Diamond X \leftrightarrow \Diamond \exists \nu X\}$

Notice in the axioms the presence of sentential quantifiers, which are not countenanced in ordinary first-order logic. These are logical symbols, and must accordingly be provided with corresponding rules of inference. The symbols $\Box$ and $\Diamond$, by contrast, are *not* part of the logic properly so called. According to this presentation, they are non-logical. So, instead of calling the resulting deductive system a modal *logic*, we would refer to it as a modal *theory*. Similarly, according to this rendition of modality, the infinite variety of modal systems – S5, S4, B, T, etc. – are thought of as competing *theories of modality*, rather than as competing *logics*.

33. Exercises for Chapter 2

1. Zero-Order Languages

Consider the ZOL described in Section 6 of Part 1.

1. For each of the strings displayed below, say whether it is well-formed (grammatical).

p♯; N♯p; Kp♯♯p♯; NKp♯p; KKp♯pp♯; NKpNp♯p; pKp♯; KpKpp; KKKpppp

2. For each of the grammatical strings in Part 1, prove that it is grammatical.

The following is an example of such a proof, using quoteless notation.

By clause a1, p is an atomic formula, so by clause a2, p♯♯ is an atomic formula, so by clause c1, p♯♯ is a formula, so by clause c2, Np♯♯ is a formula, so by clause c3, KpNp♯♯ is a formula.

2. Converting between Prefix Notation and Infix Notation

1. Using the following correspondence,

N	⇔	~
K	⇔	&
D	⇔	∨
C	⇔	→
B	⇔	↔
p,q,r	⇔	P,Q,R

convert each of the following prefix formatted formulas into the corresponding infix-formatted formulas; include all parentheses.

KpKqr ; KKpqr ; CpCqp ; CCNqNpCpq ; CCpCqrCCpqCpr ; BKpDqrDKpqKpr

2. Using the above correspondence, convert each of the following infix-formatted formulas into Polish notation.

((P∨Q)&R) ; ~(P→Q) ; (P→((Q→P)→R)) ; (((P↔Q)↔R)↔(P↔(Q↔R)))

3. First-Order Languages

Consider a first-order language $\mathcal{L}$ whose non-logical vocabulary is the following:

0	:	a proper noun;
s	:	a two-place function sign;
R	:	a two-place predicate;

1. Specify the rules of formation for $\mathcal{L}$. For these purposes, suppose that $\mathcal{L}$ does not employ category markers for function signs and predicates; in particular, the only parentheses that officially appear are in connection with two-place connectives

2. For each of the following strings, say whether it is grammatical in $\mathcal{L}$, and if it is grammatical, specify its grammatical category.

   s0 ; s00; s000; ss000; sss0000; s0s0s0; s0s0s00
   R00; R000; RR000; Rs00; Rs000; sR000; Rss00s000; sR00R00
   ∀xR00; ∃xRx0; ∃x∀yRx0; ∃x∃x∃xR00; R∀xRx; (R00 → Rs00s00); ∀xsR00
   ιxsxx; ιxRx0; sιxRxx; RιxRxxx; RιxRxxιx∀yRxy; ιxιyRxy;

3. For each of the strings in Part 2, rewrite it using bracket/parenthesis/comma notation.

4. For each of the strings in Part 2, rewrite it using infix notation.

5. For each of the grammatical strings in Part 2, prove that it is grammatical.

The following is an example of such a proof (in quoteless notation).

0 is a proper noun, so by clause a3, 0 is a singular term, so by clause a4, s00 is a singular term, so by clause a3, s0s00 is a singular term, so by clause c1, Rs0s00 is a formula. The clause numbers refer to the rules of formation for a FOL.

4. Categorial Languages

Consider a formal language $\mathcal{L}$ categorially specified as follows, where it is understand that every instance of functor application employs prefix format in addition the indicated category markers (and commas as necessary).

t	:	proper noun	:	N;
f( )	:	a one-place function sign	:	N➔N;
p[ ]	:	a two-place predicate	:	N2➔S;
c⟨ ⟩	:	a two-place connective	:	S2➔S;
s{ }	:	a one-place subnective	:	S➔N;

1. Write down the rules of formation for $\mathcal{L}$ in explicit form.
2. Give examples of formulas that involve all five symbols (with their punctuation markers).
3. Give examples of singular terms that involve all five symbols.
4. Give examples of ill-formed expressions that involve all five symbols.

34. Answers to Selected Exercises

1.1.

p♯	:	grammatical
N♯p	:	ungrammatical
Kp♯♯p♯	:	grammatical
NKp♯p	:	grammatical
KKp♯pp♯	:	grammatical
NKpNp♯p	:	ungrammatical
pKp♯	:	ungrammatical
KpKpp	:	grammatical
KKKpppp	:	grammatical

2.1.

(P & (Q & R))
((P & Q) & R)
(P → (Q → P))
((∼Q → ∼P) → (P → Q))
((P → (Q → R)) → ((P → Q) → (P → R))
((P & (Q ∨ R)) ↔ ((P & Q) ∨ (P & R)))

2.2.

KDpqr
NCpq
CpCCqpr
BBBpqrBpBqr

3.1.

a. singular terms

every variable is a singular term;
every constant (parameter) is a singular term;
0 is a singular term;
if τ_1 and τ_2 are singular terms, then $s\tau_1\tau_2$ is a singular term;
if $\mathbb{F}$ is a formula, and ν is a variable, then $\iota\nu\mathbb{F}$ is a singular term;
nothing else is a singular term.

b. atomic formulas

if τ_1 and τ_2 are singular terms, then $R\tau_1\tau_2$ is an atomic formula;
if τ_1 and τ_2 are singular terms, then $[\tau_1{=}\tau_2]$ is an atomic formula;
nothing else is an atomic formula.

c. formulas

every atomic formula is a formula;
if $\mathbb{F}$ is a formula then so is: $\sim\mathbb{F}$
if $\mathbb{F}$ and $\mathbb{G}$ are formulas, then so are $(\mathbb{F}\rightarrow\mathbb{G})$, $(\mathbb{F}\vee\mathbb{G})$, $(\mathbb{F}\&\mathbb{G})$, $(\mathbb{F}\leftrightarrow\mathbb{G})$;
if $\mathbb{F}$ is a formula, and ν is a variable, then $\forall\nu\mathbb{F}$ and $\exists\nu\mathbb{F}$ are formulas;
nothing else is a formula.

3.2.

s0	:	ill-formed
s00	:	N
s000	:	ill-formed
ss000	:	N
sss0000	:	N
s0s0s0	:	ill-formed
s0s0s00	:	N
R00	:	S
R000	:	ill-formed
RR000	:	ill-formed
Rs00	:	ill-formed
Rs000	:	S
sR000	:	ill-formed

Rss00s000	:	S
sR00R00	:	ill-formed
∀xR00	:	S
∃xRx0	:	S
∃x∀yRx0	:	S
∃x∃x∃xR00	:	S
R∀xRx	:	ill-formed
(R00 → Rs00s00)	:	S
∀xsR00	:	ill-formed
ιxsxx	:	ill-formed
ιxRx0	:	N
sιxRxx	:	ill-formed
Rιx Rxxx	:	S
RιxRxxιx∀yRxy	:	S
ιxιyRxy	:	ill-formed

4.1

a. singular terms

every variable is a singular term;
every constant (parameter) is a singular term;
t is a singular term;
if τ is a singular term, then f(τ) is a singular term;
if $\mathbb{F}$ is a formula, then s{$\mathbb{F}$} is a singular term;
nothing else is a singular term.

b. atomic formulas

if τ_1 and τ_2 are singular terms, then p[τ_1,τ_2] is an atomic formula;
nothing else is an atomic formula.

c. formulas

every atomic formula is a formula;
if $\mathbb{F}$ and $\mathbb{G}$ are formulas, then c$\langle\mathbb{F},\mathbb{G}\rangle$ is a formula;
nothing else is a formula.