

Logic - Notes

Dom Hutchinson

February 3, 2020

Contents

1	Introduction	2
1.1	Alphabets & Strings	2
1.2	Countable Sets	2
2	First-Order Languages	3
2.1	Induction of Terms & Formulae	6

1 Introduction

1.1 Alphabets & Strings

Definition 1.1 - Alphabet

An *Alphabet* is a set of symbols from which *Strings* can be created.

Definition 1.2 - String

A *String* over a set \mathcal{A} is any sequence $\alpha := \langle a_1, \dots, a_n \rangle$ where $a_1, \dots, a_n \in \mathcal{A}$.
N.B. Here we say α has *length* n and $\alpha \in \mathcal{A}^n$.

Definition 1.3 - Power Set

Let \mathcal{A} be an alphabet. We define

$$\mathcal{A}^* := \bigcup_{n \in \mathbb{N}} \mathcal{A}^n = \{ \langle a_1, \dots, a_n \rangle : n \in \mathbb{N}; a_1, \dots, a_n \in \mathcal{A} \}$$

This means \mathcal{A}^* is the set of all possible strings over alphabet \mathcal{A} .

Remark 1.1 - Concatenating Strings

Define *Strings* $\alpha := \langle a_1, \dots, a_n \rangle \in \mathcal{A}^n$ and $\beta := \langle b_1, \dots, b_m \rangle \in \mathcal{A}^m$.

We define *Concatenation* of α & β as $\alpha\beta := \langle a_1, \dots, a_n, b_1, \dots, b_m \rangle$ Note that

$$\alpha\beta \neq \langle \alpha, \beta \rangle = \langle \langle a_1, \dots, a_n \rangle, \langle b_1, \dots, b_m \rangle \rangle$$

N.B. Sometimes the following notation is used $\alpha * \beta$.

Example 1.1 - English Alphabet

If we define an alphabet $\mathcal{A} := \{ 'a', \dots, 'z' \}$ then $\langle 't', 'h', 'i', 's' \rangle$ is a *String* of \mathcal{A} .

Remark 1.2 - Ambiguity when using multiple Alphabets

Consider the *Alphabets* $\mathcal{A}_1 := \{0, 1, \dots, 9\}$ & $\mathcal{A}_2 := \mathbb{N}$.

Then we are unsure which of the following definitions of 123 is valid

$$\langle 123 \rangle, \langle 12, 3 \rangle, \langle 1, 23 \rangle, \langle 1, 2, 3 \rangle$$

Remark 1.3 - $\mathcal{A} := \{0, 1\}$ is sufficient to describe any language - binary

Remark 1.4 - Describing Formal Languages

When describing a *Formal Language* we need to provide two things

- i) An *Alphabet* which defines what symbols are allowed.
- ii) A *Grammar* which defines what combinations of symbols are allowed.

1.2 Countable Sets

Definition 1.1 - Countable Set

A set X is said to be *Countable* if

$$\begin{aligned} &\exists \text{ a surjection } f : \mathbb{N} \rightarrow X \\ &\exists \text{ an injection } f : X \rightarrow \mathbb{N} \end{aligned}$$

Definition 1.2 - Countably Infinite Set

A set X is said to be *Countably Infinite* if \exists a bijection $f : X \rightarrow \mathbb{N}$.

Theorem 1.1 - Power set is Countable

If set \mathcal{A} is *countable* then \mathcal{A}^* is *countable*.

Proof 1.1 - Theorem 1.1

Let $f : \mathcal{A} \rightarrow \mathbb{N}$ (This function exists trivially since we define \mathcal{A} to be countable).

Define the following function $g(\cdot) : \mathcal{A}^* \rightarrow \mathbb{N}$

$$g(\langle a_1, \dots, a_n \rangle) := p_1^{f(a_1)+1} \cdot \dots \cdot p_n^{f(a_n)+1}$$

where p_i is the i^{th} prime.

Since each natural number can be described by a unique composition of primes and since $f(\cdot)$ is injective, then $g(\cdot)$ is injective.

Thus there exists an injection from \mathcal{A}^* to \mathbb{N} , making \mathcal{A}^* countable.

Theorem 1.2 - If \mathcal{A} is countable, then so are $\mathcal{A}^*, (\mathcal{A}^*)^*, \dots$

2 First-Order Languages

Definition 2.1 - First-Order Language, \mathcal{L}

The *Alphabet* of a *First-Order Language*, comprises of the following, pairwise disjoint, categories (and nothing else)

- i) Negation, \neg , and implication, \rightarrow .
- ii) For all, \forall .
- iii) Infinitely many variables, $\{v_0, v_1, \dots\}$.
- iv) Parentheses, (\cdot) , and comma $,$.
- v) Equality, \equiv , which is the only logical predicate symbol with 2-arity.
- vi) A set of constant symbols, $\{c_1, c_2, \dots\}$. (Possibly empty)
- vii) For each $n \geq 1$, a set of n -arity function symbols $\{f_1^n, f_2^n, \dots\}$. (Possibly empty)
- viii) For each $n \geq 1$, a set of n -arity non-logical predicate symbols $\{P_1^n, P_2^n, \dots\}$. (Possibly empty)

N.B. We denote the set of variables by $Var := \{v_0, v_1, \dots\}$; denote a language as \mathcal{L} and the alphabet of \mathcal{L} as $\mathcal{A}_{\mathcal{L}}$.

N.B. In this course *Alphabets* are restricted to being *Countable*.

Definition 2.2 - Negation, \neg

Negation returns in the inverse of a predicate (DO I MEAN PREDICATE)

P	$\neg P$
T	F
F	T

Definition 2.3 - Implication, \rightarrow

Implication returns whether one predicate being true necessarily implies a second predicate being true

P	Q	$P \rightarrow Q$
T	T	T
T	F	F
F	T	T
F	F	T

Remark 2.1 - *First-Order Languages don't have \wedge , \vee , \exists*

Alphabets for *First-Order Languages* do not contain propositional connectives for AND, \wedge , OR, \vee , or EXISTS, \exists since they can be expressed as a combination of negation & implication.

$$\begin{aligned}
 P \vee Q &\iff \neg P \rightarrow Q \\
 P \wedge Q &\iff \neg(P \rightarrow \neg Q) \\
 \exists x \text{ st } P(x) \text{ is true} &\iff \neg(\forall x, \neg P(x))
 \end{aligned}$$

P	Q	$\neg P$	$\neg P \rightarrow Q$	P	Q	$\neg Q$	$P \rightarrow \neg Q$	$\neg(P \rightarrow \neg Q)$
T	T	F	T	T	T	F	F	T
T	F	F	T	T	F	T	T	F
F	T	F	T	F	T	F	T	F
F	F	F	F	F	F	T	T	F

Example 2.1 - *Recursive Definition*

Consider the following, normal, definition

$$x \text{ is a multiple of } 5 \iff \exists y \in \mathbb{Z} \text{ st } y \cdot 5 = x$$

We can instead use the recursive definition

- i) 0 is a multiple of 5.
- ii) If n is a multiple of 5 then $n + 5$ is a multiple of 5.

Definition 2.4 - *\mathcal{L} -Term & Complexity*

Let \mathcal{L} be a *First-Order Language*.

We define \mathcal{L} -Terms & Complexity, $cp(\cdot)$, together using the following *recursive definition*

- i) If s is a variable or a constant symbol, then s is an \mathcal{L} -Term with $cp(s) = 0$.
N.B. Terms with $cp(\cdot) = 0$ are called *Atomic Terms*.
- ii) If f is a function symbol with k -arity & if a_1, \dots, a_k are \mathcal{L} -Terms then $f(a_1, \dots, a_k)$ is an \mathcal{L} -Term with complexity

$$cp(f(a_1, \dots, a_k)) := \max\{cp(a_1), \dots, cp(a_k)\} + 1$$

N.B. Terms with $cp(\cdot) \geq 1$ are called *Compound Terms*.

- iii) Nothing else is an \mathcal{L} -Term

N.B. We denote the set of \mathcal{L} - Terms by $T_{\mathcal{M}_{\mathcal{L}}}$.

Example 2.2 - *Complexity*

Let $\{c, d, f, g, h, p\} \subseteq \mathcal{L}$ with c, d being constants, g, p being unary functions & f, h being binary functions.

Show that the following is an \mathcal{L} -Term & find its Complexity

$$h(g(f(x, c)), p(d))$$

- i) x is an \mathcal{L} -Term with $cp(x) = 0$ by (i).

- ii) c & d are \mathcal{L} -Terms with $cp(c) = 0 = cp(d)$ by (i).
- iii) $f(x, c)$ is an \mathcal{L} -Term with $cp(f) = \max 0, 0 + 1 = 1$ by (ii).
- iv) $p(d)$ is an \mathcal{L} -Term with $cp(f) = \max 0 + 1 = 1$ by (ii).
- v) $g(f(x, c), p(d))$ is an \mathcal{L} -Term with $cp(g) = \max 1, 1 + 1 = 2$ by (ii).
- vi) $h(g(f(x, c), p(d)))$ is an \mathcal{L} -Term with $cp(h) = \max 2 + 1 = 3$ by (ii).

Thus $h(g(f(x, c), p(d)))$ is an \mathcal{L} -Term with Complexity 3.

Notation 2.1 - More readable Functions

WE often write $x \circ y$ instead of $\circ(x, y)$ as it is more readable (even though the later is technically the only correct notation). Similarly, $x + y$ instead of $+(x, y)$.

Definition 2.5 - Atomic Formulae

Let \mathcal{L} be a First-Order Language.

The atomic \mathcal{L} -Formulae are those strings over $\mathcal{A}_{\mathcal{L}}$ of the form

$$R(t_1, \dots, t_n) \text{ for } n \in \mathbb{N}$$

where R is a predicate symbol of \mathcal{L} with n -arity and t_1, \dots, t_n are \mathcal{L} -terms.

N.B. $\equiv (t_1, t_2)$ is an Atomic \mathcal{L} -Formula for each \mathcal{L} terms t_1, t_2 .

Definition 2.6 - \mathcal{L} -Formulae & Complexity

We define \mathcal{L} -Formulae & Complexity, $cp(\cdot)$, together using the following recursive definition

- i) If $\phi \in \mathcal{A}_{\mathcal{L}}^*$ is an Atomic \mathcal{L} -Formula then ϕ is an \mathcal{L} -Formula with $cp(\phi) = 0$.
- ii) If ϕ is an \mathcal{L} -Formula with $cp(\phi) = n$ then $\neg\phi$ is an \mathcal{L} -Formula with $cp(\neg\phi) = n + 1$.
- iii) If ϕ & ψ are \mathcal{L} -Formulae then $\phi \rightarrow \psi$ is an \mathcal{L} -Formula with $cp(\phi \rightarrow \psi) = \max\{cp(\phi), cp(\psi)\} + 1$.
- iv) if ϕ is an \mathcal{L} -Formula then $\forall x\phi$ is an \mathcal{L} -Formula with $cp(\forall x\phi) = cp(\phi) + 1$, where x is a variable.

Remark 2.2 - Formulae are uniquely readable & parsable

Example 2.3 - \mathcal{L} -Formulae Complexity

Let $\{R, f\} \subset \mathcal{L}$ be binary operations.

Show that the following is an \mathcal{L} -Formula

$$\forall v_0 (\neg R(f(v_0, v_2), v_2) \longrightarrow \underbrace{\equiv (v_0, v_2)}_{v_0 \equiv v_2})$$

- i) v_0, v_2 are \mathcal{L} -Terms.
- ii) $f(v_0, v_2)$ is an \mathcal{L} -Term.
- iii) $R(f(v_0, v_2), v_2)$ is an \mathcal{L} -Formula with $cp(\cdot) = 0$.
- iv) $\neg R(f(v_0, v_2), v_2)$ is an \mathcal{L} -Formula with $cp(\cdot) = 0 + 1 = 1$.
- v) $\equiv (v_0, v_2)$ is an \mathcal{L} -Formula with $cp(\cdot) = 0$.
- vi) $\neg R(f(v_0, v_2), v_2) \longrightarrow \equiv (v_0, v_2)$ is an \mathcal{L} -Formula with $cp(\cdot) = \max\{0, 1\} + 1 = 2$.

vii) $\forall v_0 (\neg R(f(v_0, v_2), v_2) \longrightarrow \equiv (v_0, v_2))$ is an \mathcal{L} -Formula with $cp(\cdot) = 2 + 1 = 3$.

Notation 2.2 - Convention for common operators

To make formulae more readable we generally make the following allowances in notation

$$\begin{aligned} t_1 \equiv t_2 & \text{ for } \equiv (t_1, t_2) \\ t_1 \not\equiv t_2 & \text{ for } \neg \equiv (t_1, t_2) \\ t_1 < t_2 & \text{ for } < (t_1, t_2) \\ t_1 \not< t_2 & \text{ for } \neg < (t_1, t_2) \end{aligned}$$

Further, when a formula is encapsulated by parentheses then we will often suppress the outermost parentheses (only), as they do not affect anything.

$$\phi \longrightarrow (\psi \longrightarrow \theta) \text{ for } (\phi \longrightarrow (\psi \longrightarrow \theta))$$

Definition 2.7 - More complex operators

- AND, $(\phi \wedge \psi) := \neg(\phi \longrightarrow \neg\psi)$.
- OR, $(\phi \vee \psi) := (\neg\phi \longrightarrow \psi)$.
- IFF, $(\phi \longleftrightarrow \psi) := (\phi \longrightarrow \psi) \wedge (\psi \longrightarrow \phi)$.
- EXISTS, $(\exists x\phi) := \neg\forall x \neg\phi$.

Notation 2.3 - Sets of \mathcal{L} Features

- $T_{\mathcal{M}_{\mathcal{L}}} :=$ Set of \mathcal{L} -Terms.
- $F_{\mathcal{M}_{\mathcal{L}}} :=$ Set of \mathcal{L} -Formulae.
- $\text{Var} :=$ Set of Variables.

Proposition 2.1 - $T_{\mathcal{M}_{\mathcal{L}}}$ & $F_{\mathcal{M}_{\mathcal{L}}}$ are always countable in this course since we assume \mathcal{L} to be finite.

2.1 Induction of Terms & Formulae

Theorem 2.1 - Inheritance of a Property - \mathcal{L} -Terms

Let P be a property of \mathcal{L} -Terms.

Suppose the following to be true

- i) All Atomic \mathcal{L} -Terms have property P .
- ii) $\forall k \in \mathbb{N}, \forall$ function symbols f with k -arity: If \mathcal{L} -Terms t_1, \dots, t_k have property P then $f(t_1, \dots, t_k)$ has P .

Then every \mathcal{L} -Term has property P .

Proof 2.1 - Theorem 2.1

This is a proof by contradiction. Suppose that i) & ii) are true but there exists some \mathcal{L} -Term which does not have P .

Let t be an \mathcal{L} -Term with minimum complexity st t does not have P .

Then $cp(t) \neq 0$ otherwise i) would be untrue.

Thus $t \equiv f(t_1, \dots, t_k)$ by the minimality of $cp(t)$.

We know that t_1, \dots, t_k have P .

Thus $f(t_1, \dots, t_k)$ has P . This is a contradiction.

Theorem 2.2 - Inheritance of a Property - \mathcal{L} -Formulae

Let P be a property of \mathcal{L} -Formula.

Suppose the following to be true

- i) All *Atomic \mathcal{L} -Formulae* have property P .
- ii) If $\phi, \psi \in F_{\mathcal{M}_{\mathcal{L}}}$ have P then $\neg\phi, (\phi \rightarrow \psi)$ & $\forall x\phi$ have P to.

Then every \mathcal{L} -Formulae has property P .