Logic - Notes

Dom Hutchinson

February 6, 2020

Contents

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

$$\begin{array}{c} \exists \text{ a surjection } f : \mathbb{N} \to X \\ \exists \text{ an injection } f : X \to \mathbb{N} \end{array}$$

Definition 1.2 - *Countably Infinite Set*
A set X is said to be *Countably Infinite* if $\exists$ a bijection $f : X \to \mathbb{N}$.

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 $f : \mathcal{A} \longrightarrow \mathbb{N}$ (This function exists trivally since we define $\mathcal{A}$ to be countable).
Define the following function $g(\cdot) : \mathcal{A}^* \longrightarrow \mathbb{N}$

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

where p_i is the i^{th} prime.
Since each natural number can be described by a unique composition of primes and since $f(\dot{})$ 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, $\longrightarrow$.

ii) For all, $\forall$.

iii) Infinitely many variables, $\{v_0, v_1, \dots\}$.

iv) Parentheses, '(' ')', and comman ','.

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, $\longrightarrow$*
Implication returns whether one predicate being true necessarily implies a second predicate being true

P	Q	$P \to Q$
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.

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

P	Q	$\neg P$	$\neg P \to Q$
T	T	F	T
T	F	F	T
F	T	F	T
F	F	F	F

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

Example 2.1 - *Recursive Defintion*
Consider the following, normal, deifition

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

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 fucntion symbol with k-arity & if $a_1, \dots, a_k$ are *$\mathcal{L}$-Terms* then $f(a_1, \dots, f_k)$ 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 - *$\mathcal{L}$-Term & Complexity*
Let $\{c, d, f, g, h, p\} \subseteq$ Ł with c, d being constants, g, p being uniary 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 readble 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 $\phi \to \psi$ is an $\mathcal{L}$-*Formula* with $cp(\phi \to \psi) = \max\{cp(\phi), cp(\psi)\} + 1$.

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.

N.B. Complexity is just a measure of the syntactic complexity, not semantic. Notice how $cp(\neg\neg\phi) = cp(\phi) + 2$.

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) $\neq R(f(v_0, v_2), v_2) \longrightarrow \equiv (v_0, v_2)$ 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 $cp(\dot{)} = 2 + 1 = 3$.

Notation 2.2 - *Convention for common operators*
To make formulae more readble we general make the following allowances in notation

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

Further, when a formula is encapsulated by parantheses then we will often surpress 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.
- 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 Proeprty - $\mathcal{L}$-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 minimlaity 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 Proeprty - $\mathcal{L}$-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 $\neg\phi, (\phi \to \psi)$ & $\forall\ x\phi$ have P to.

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

Theorem 2.3 - *Number of Parenthese*
Ever *$\mathcal{L}$-Formula* has as many left parentheses as right parentheses.
Ever *$\mathcal{L}$-Term* has as many left parentheses as right parentheses.

Proof 2.2 - *Theorem 2.3*
This is a proof by induction.
Let P be the property *"Has as many left parenthese as right"*.
Base Case - When ϕ is an *Atomic $\mathcal{L}$ Formula* it trivially has equal number of parenthese.
Inductive Case
Let ϕ & ψ be arbitrary $\mathcal{L}$-Formulae.
Assume that $P(\phi)$ & $P(\psi)$ hold.
We need to show that $P(\neg\phi),\ P(\phi \to \psi)$ & $P(\forall\ x\phi)$ all hold.
We do not need to show $P(\neg\psi),\ P(\psi \to \phi)$ & $P(\forall\ x\psi)$ hold as ϕ & ψ are arbitrary.
We have that $\neg\phi$ and $\forall\ x\phi$ don't add any brackets, so P holds.
We have that $(\phi \to \psi)$ add one left & one right parentheses (although they are often surpressed), thus P holds.
Thus by the process of mathematical induction P holds forall $\mathcal{L}$-Formulae.
N.B. The proof for $\mathcal{L}$-Terms is very similar.

2.2 Free Variables

Definition 2.1 - *Variable Function, $var(\cdot)$*
Define $var : {\mathcal{A}_\mathcal{L}}^* \to 2^{\text{Var}}$ st $var(s)$ is the set of all variables in string s.

Example 2.4 - Var$(\cdot)$

$$\begin{array}{rcl} var(f(x, f(y, c)) & = & \{x, y\} \\ var(f(c, f(c, c)) & = & \emptyset \\ var(\equiv, \equiv, \equiv) & = & \emptyset \text{ nonsense strins are acceptable} \end{array}$$

Definition 2.2 - *Free Variables*
Free Variables are variables whose value are ambiguous in an *$\mathcal{L}$-Formula.*

Definition 2.3 - *Free Variable Function, $FV(\cdot)$*
We recursively define $FV(\phi)$ for *$\mathcal{L}$-Formulae* as ϕ as follows

i) $FV(\phi) = var(\phi)$ if ϕ is an *Atomic $\mathcal{L}$-Formula.*

ii) $FV(\neg\phi) = FV(\phi)$.

iii) $FV((\phi \to \psi)) = FV(\phi) \cup FV(\psi)$.

iv) $FV(\forall\ x\phi) = FV(\phi)\backslash\{x\}$.

Example 2.5 - *Free Variable Function*

$$\begin{array}{rcl}FV(\forall x(P(y)\to Q(x))) & = & FV(P(y)\to Q(x))\backslash\{x\}\\ & = & [FV(P(y))\cup FV(Q(x))]\backslash\{x\}\\ & = & [\{y\}\cup\{x\}]\backslash\{x\}\\ & = & \{y\}\end{array}$$

Proposition 2.2 - *Free Variable Function for more complex operators*

$$\begin{array}{rcl}FV(\phi\wedge\psi) & = & FV(\neg(\phi\to\neg\psi))\text{ by definition of }\wedge\\ & = & FV(\phi)\cup FV(\psi)\\ FV(\phi\vee\psi) & = & FV(\neg\phi\to\psi)\text{ by definition of }\vee\\ & = & FV(\phi)\cup FV(\psi)\\ FV(\exists x\phi) & = & FV(\neg\forall x\neg\phi)\text{ by definition of }\exists\\ & = & FV(\phi)\backslash\{x\}\end{array}$$

Definition 2.4 - *Closed $\mathcal{L}$-Term*
Let t be an $\mathcal{L}$-Term.
If $var(t)=\emptyset$ then t is called a *Closed $\mathcal{L}$-Term.*

Definition 2.5 - *$\mathcal{L}$-Sentence*
Let ϕ be an $\mathcal{L}$-Formula.
If $FV(\phi)=\emptyset$ then ϕ is called an *$\mathcal{L}$-Sentence.*

Example 2.6 - *$\mathcal{L}$-Sentence*

$$\begin{array}{rcl}FV(\forall x(P(x)\to\exists y\ R(y,x)) & = & FV((P(x)\to\exists y\ R(y,x))\backslash\{x\}\\ & = & FV(P(x))\cup FC(\exists y\ R(y,x))\backslash\{x\}\\ & = & \{x\}\cup(FV(R(y,x)\backslash\{y\})\backslash\{x\}\\ & = & \{x\}\cup(\{y,x\}\backslash\{y\})\backslash\{x\}\\ & = & \{x\}\cup\{x\}\backslash\{x\}\\ & = & \emptyset\end{array}$$

Remark 2.3 - *$\mathcal{L}$-Sentences have no Free Variables and thus no ambiguity in meaning.*

3 Semantics of First-Order Languages

Definition 3.1 - *$\mathcal{L}$-Structure*
Let $\mathcal{L}$ be a first-order language.
An *$\mathcal{L}$-Structure* is an ordered pair $\mathfrak{M}=\langle D,\mathfrak{I}\rangle$

i) D is a non-empty set.

ii) $\mathfrak{I}$ is a function on the non-logical symbols of $\mathcal{L}$ st

- For each predicate symbol $P\in\mathcal{L}$ with n-arity.

$$\mathfrak{I}(P)\subset D^n$$

- For each function symbol f of $\mathcal{L}$ with n-arity

$$\mathfrak{I}(f):D^n\to D$$

- For each constant symbol c of $\mathcal{L}$
$$\mathfrak{I}(c) \in D$$

N.B. D is the domain, $\mathfrak{I}$ is the interpretation.

Notation 3.1 - *$\mathcal{L}$-Structure*
For ease we use the following notation wrt *$\mathcal{L}$-Structure*
$$|\mathfrak{M}| := D \quad f^{\mathfrak{M}} := \mathfrak{I}(f) \quad c^{\mathfrak{M}} := \mathfrak{I}(c) \quad p^{\mathfrak{M}} = \mathfrak{I}(p)$$

Example 3.1 - *$\mathcal{L}$-Structure*
Let $\mathcal{L}_{\text{Rng}} := \{\bar{0}, \bar{1}\bar{+}, \bar{\cdot}\}$ where $\bar{+}$ & $\bar{\cdot}$ are binary functions and $\bar{0}$ & $\bar{1}$ are constants.
(This is the language for ring theory)
We use the overline to distringuish language symbols from standard symbols.
Define
$$\begin{array}{rcl} D & := & \mathbb{R} \\ \mathfrak{I}(\bar{0}) & = & 0 \in \mathbb{R} \\ \mathfrak{I}(\bar{1}) & = & 1 \in \mathbb{R} \\ \mathfrak{I}(\bar{+}) & : & \mathbb{R} \times \mathbb{R} \to \mathbb{R} \text{ with}(a, b) \mapsto (a + b) \\ \mathfrak{I}(\bar{\cdot}) & : & \mathbb{R} \times \mathbb{R} \to \mathbb{R} \text{ with}(a, b) \mapsto (a \cdot b) \end{array}$$
We recall $\langle D, \imath\rangle$ is the standard model of the real field.
N.B. We can alternatively write $\langle D, \mathfrak{I}\rangle = \langle \mathbb{R}, 0, 1, +, \cdot\rangle$ for neatness.
Definition 3.2 - *Variable Assignment*
A *Variable Assignment* over an $\mathcal{L}$-Structure is a function which maps from the set of variables to the domain of the $\mathcal{L}$-Structure.
$$s : \text{Var} \to |\mathfrak{M}|$$

Definition 3.3 - *Extension of Variable Assignment*
Let $\mathfrak{M}$ be an $\mathcal{L}$-Structure & s be a variable assignment over $\mathfrak{M}$.
The function $\bar{s} : T_{\mathfrak{M}_{\mathcal{L}}} \to |\mathfrak{M}|$ is defined using the following recursion

i) $\bar{s}(t) = s(t)$ if $t \in \text{Var}$.

ii) $\bar{s}(t) = t^{\mathfrak{M}}$ if t is a constant symbol.

iii) $\bar{s}(f(t_1, \dots, t_k)) = f^{\mathfrak{M}}(\bar{s}(t_1), \dots, \bar{s}(t_k))$.

Example 3.2 - *Variable Assignment*
Let $\mathfrak{M}$ be the standard model of the real field.
Let s bea variable assignment over $\mathfrak{M}$ st $s(x) = s(y) = \pi$. Then
$$\begin{array}{rcl} \bar{s}(x\bar{+}y) & = & \bar{+}^{\mathfrak{m}}(\bar{s}(x), \bar{s}(y)) \\ & = & \bar{+}^{\mathfrak{m}}(s(x), s(y)) \\ & = & \bar{+}^{\mathfrak{m}}(\pi, \pi) \\ & = & \pi + \pi \\ & = & 2\pi \end{array}$$

Theorem 3.1 -
Let s be a variable assignment over $\mathfrak{M}$, $x \in \text{Var}$ & $d \in |\mathfrak{M}|$.
A new variable assignment $\frac{sd}{x}$ over $\mathfrak{M}$ is defined as
$$\frac{sd}{x}(y) = \begin{cases} d & \text{if } y = x \\ s(y) & \text{otherwise} \end{cases}$$

Definition 3.4 - *Satisfaction Relation*
Let $\mathfrak{M}$ be an $\mathcal{L}$-Structure & s be a variable assignment over $\mathfrak{M}$.
The *Satiscation Relation*, $\mathfrak{M}, s \vDash \phi$ between $\mathfrak{M}, s$ and $\mathcal{L}$-Formula ϕ is recursively defined as

i) $\mathfrak{M}, s \vDash t_1 \equiv t_2$ iff $\bar{s}(t_1) = \bar{s}(t_2)$.

ii) $\mathfrak{M}, s \vDash P(t_1, \dots, t_k)$ iff $\langle \bar{s}(t_1), \dots, \bar{s}(t_k) \rangle \in P^{\mathfrak{M}} \subset D^k$.

iii) $\mathfrak{M}, s \vDash \neg\phi$ iff $\mathfrak{M}, s \not\vDash \phi$.

iv) $\mathfrak{M}, s \vDash \phi \to \psi$ iff if $\mathfrak{M}, s \vDash \phi$ then $\mathfrak{M}, s \vDash \psi$.

v) $\mathfrak{M}, s \vDash \forall x \phi$ iff for all $d \in |\mathfrak{M}|$, $\mathfrak{M}\frac{sd}{x} \vDash \phi$.