Algebraic Logic
for Deep Learning

by

YKY

A Thesis Submitted to

The Hong Kong University of Science and Technology

in Partial Fulfilment of the Requirements for

the Degree of Master of Philosophy

in Applied Mathematics

August 2024, Hong Kong

Authorization

I hereby declare that I am the sole author of the thesis.

I authorize the Hong Kong University of Science and Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the Hong Kong University of Science and Technology to reproduce the thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

_____

YKY

13 August 2024

Algebraic Logic
for Deep Learning

by

YKY

This is to certify that I have examined the above MPhil thesis
and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by
the thesis examination committee have been made.

_____

Prof. Who, Thesis Supervisor

_____

Prof. Who Else, Thesis Co-supervisor

_____

Prof. Ikari Someone, Head of Department

Department of Maths

13 August 2024

ACKNOWLEDGEMENTS

Thank you, all the Evangelion.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

PREFACE

Our main contribution is the re-formulation of old-fashioned logic-based AI based on the new technique of deep learning.

Deep learning is truly revolutionary for a single reason: It learns much faster. This allows to handle huge datasets which were perviously out of consideration, and crucially, common-sense reasoning by its nature has to be training on massive datasets. The reason why we're now seeing machines with human-comparable intelligence owes to this feat.

Why is deep learning so highly efficient? It seems to achieve this by operating in a "continuous" domain (by gradient descent) in very high-dimensional spaces, such that the problem of **local minima** seems to be miraculously avoided. To accurately explain this phenomenon may involve the mysteries of the $P \overset{?}{=} NP$ problem and should not be expected any time soon.

Deep learning also has the characteristic of hierarchically many layers of parameters, but this may not be a necessary requirement, as recent thinking on "**shallow learning**" suggests. Our algorithm also challenges this view.

Our key contribution is to make logic rules **differentiable**, so that gradient descent can be applied to find an optimal set of logic rules optimizing some objective function. This solves the age-old problem of logic-based AI's lack of an efficient learning algorithm, that is responsible for the so-called "**AI Winter**" around the 1970-80s.

The Transformer, based on Self-Attention, is the current state-of-the-art module for building LLMs (Large Language Models). It is based on a **differentiable memory-retrieval mechanism** that originated in **Neural Turing Machines**, and later shown to be equivalent to **Hopfield Networks**, also a kind of associative memory. It suffers from the drawback that it is not easy to interpret what the "tokens" stand for, that hinders further attempts to improve it. The logic-based approach offers a much better understanding of the internal workings of AGI.

Moreover, our model, which may be called "Logic Transformer," is significantly more structured than the traditional Transformer. According to the **No Free Lunch theorem**, machines with more restrictive structures have more inductive bias and learn faster.

The most important theoretical background that inspired this research is Paul Halmos' **algebraic logic**. In retrospect, I just needed to make traditional logic algorithms differentiable. But with a great mathematician as Halmos standing behind my back and pointing to what needs to be done, I felt much more confident to work through some tedious details. One cannot exactly

pinpoint what is the importance of "beautiful" mathematics, but it really made a big difference.

At this point I do not know the experimental outcome yet. I hope my work can inspire the reader to find better improvements (eg. by rewriting the neural string diagram) or explore entirely different directions.

Good luck.

<div align="right">

YKY

2024, Hong Kong

</div>

Algebraic Logic
for Deep Learning

by YKY

Department of Maths

The Hong Kong University of Science and Technology
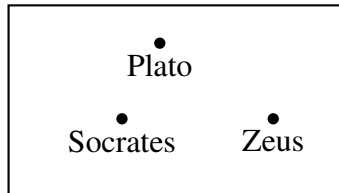
Abstract

Some text.

CHAPTER 0

INTRODUCTION

Our goal is to make logic rules **differentiable**. The **gradient descent** $\nabla_\Theta \mathcal{L}$ will find the best set of rules, as long as we can express an overall loss function $\mathcal{L}$ in terms of all the parameters Θ of the rules.

0.1 Predicates

The trickiest part of algebraizing a logic (for symbolic AI) is how to handle **predicates**. The common practice is to treat predicates as mathematical **functions** that send elements from a space X to the set of truth values $\Omega = \{\top, \bot\}$.

I coined the term **Subject Space** X as the space of subjects that predicates refer to:

$$\text{Subject Space } X \quad \boxed{\begin{array}{c} \bullet \\ \text{Plato} \\ \bullet \qquad \bullet \\ \text{Socrates} \quad \text{Zeus} \end{array}} \tag{1}$$

Think of it as the "embedding space" of atomic concepts (or words or tokens in an LLM, prepared by the **Word2Vec** algorithm). Keep this picture in mind for the sequel. This is usually a **high-dimensional** space, eg. 512-dim. A separate **Predicate Space** is needed, as required by predicate logic.

In our learning algorithm, each predicate function is implemented as a **neural network** associated to that predicate (input = vector in X, output truth value). This is convenient as neural networks can carve out regions of irregular shapes (not necessarily simply connected) in a high dimensional space.

0.2 Logic rules

For simplicity we restrict to **Horn rules**, which are of the form:

$$Q_1 \wedge Q_2 \wedge Q_3 \wedge ... \rightarrow Q_0 \tag{2}$$

where each Q_i is an **atomic proposition** that may be optionally negated. This is the form of rules used by the logic programming language **PROLOG**, so we assume it is powerful enough to express knowledge for AGI.

If rules are to be differentiable, this means every rule may potentially "morph" into any other rule in the space of rules. The only solution I can think of is to maintain a fixed number, M, of rules, each containing a fixed number, K, of predicates[1]. The parameters would look like a rectangular matrix:

$$\text{rule 1:} \quad \boxed{P_1^1 X_{11}^1 ... X_{1I}^1} \wedge \boxed{P_2^1 X_{21}^1 ... X_{2I}^1} \wedge ... \boxed{P_K^1 X_{K1}^1 ... X_{KI}^1} \rightarrow \boxed{P_0^1 X_{01}^1 ... X_{0I}^1}$$

$$\text{rule 2:} \quad \boxed{P_1^2 X_{11}^2 ... X_{1I}^2} \wedge \boxed{P_2^2 X_{21}^2 ... X_{2I}^2} \wedge ... \boxed{P_K^2 X_{K1}^2 ... X_{KI}^2} \rightarrow \boxed{P_0^2 X_{01}^2 ... X_{0I}^2} \tag{3}$$

$$.... \quad ....$$

where P are predicates and X are arguments (that can be constants or variables). Each predicate has I arguments, also fixed.

0.3 Constants

A constant argument in a predicate can morph into a variable. For example, the constant vector $\vec{c} = \text{John}$ in $\text{loves}(\text{John}, \text{Mary})$ can become $\forall X.\text{loves}(X, \text{Mary})$, "everyone loves Mary."

This is achieved by assigned to each argument (be it a constant or a variable), a **cylindrification factor** $\gamma \in [0, 1]$ such that when $\gamma = 0$, the constant John would be exactly John, but when $\gamma \rightarrow 1$, John would become the entire Subject Space X. We define the function $\gamma(\vec{c})$ which is a **radial basis function** centered at the constant $\vec{c}$.

---

[1]In my mind I think of "millions of rules" and "tens of thousands of predicates" so the letters M and K are easy to remember. The number of predicates may be comparable to human vocabulary size. My estimate for rules is just an arbitrary guess.

Cylindrification is illustration as follows:

$$(4)$$
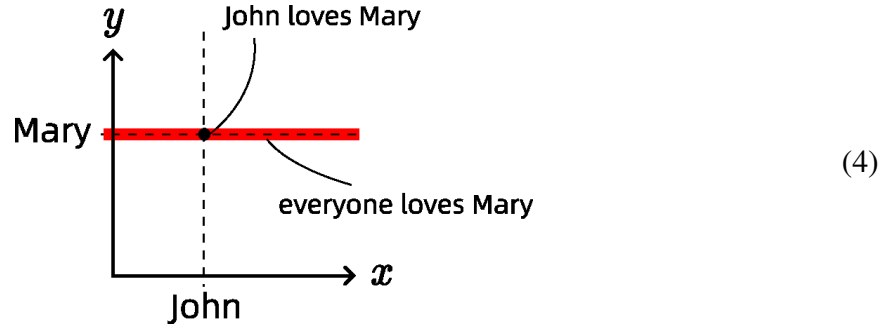
The Subject Space X is 1-dimensional here. The relation love lives in $X \times X$.

0.4 Variables

The treatment of variables and their bindings is another tricky problem. Our solution follows Paul Halmos' approach developed in the 1950s [1]. In his **monadic logic**, where predicates are functions $P : X \to \Omega$, there could only be one variable living in the Subject Space X. In his **polyadic logic**, predicates can take on more than one arguments, which involves multiple copies of the Subject Space X. He constructs X^I where I is an index set of "variables," even though they themselves don't really change. For example, $I = \{1, 2, 3, 4\}$ provides 4 variable "slots".

The implementation can be illustrated as follows:

$$(5)$$
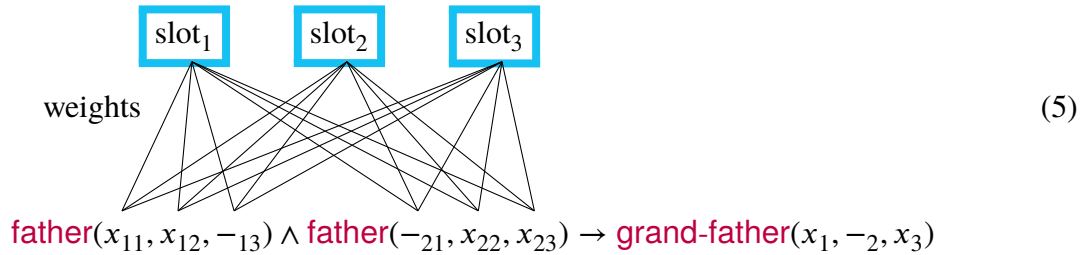
Every argument in the lower row has the potential to end up in any slot on the upper row. For each slot, we take a **Softmax** to "select" one argument as the "winner". Such a parallel use of Softmax has essentially the same structure as – surprise? – **Self Attention**.

The rest of my thesis will explain these constructions in more details, but once the reader understands the principles behind them, the tedious details kind of fall into place naturally.

CHAPTER 1

BACKGROUND: THE MIND AS A DYNAMICAL SYSTEM

1.1 The set-up

The set of equations F defines an algebraic set = **the world**:

$$F(x) = 0. \tag{1.1}$$

The objective of an intelligent agent is to learn F.

We have the function f performing **prediction** of the immediate future:

$$\boxed{\text{current state}} \quad x_t \overset{F}{\mapsto} x_{t+1} \quad \boxed{\text{next state}}. \tag{1.2}$$

In an infinitesimal sense, we can see f as a **differential equation** describing the **world trajectory**:

$$\dot{x} = f(x). \tag{1.3}$$

So F is the **solution** to this differential equation.

It seems that F and f are more or less equivalent ways to describe the world.

Logic can be turned into some form of algebra, and this algebra can be used to express either F or f. Perhaps both ways are feasible, or even mixing the two.

What does it mean to use logic to express F or f?

Go back to a physics example, the parabolic trajectory of a canon ball:

$$\tag{1.4}$$
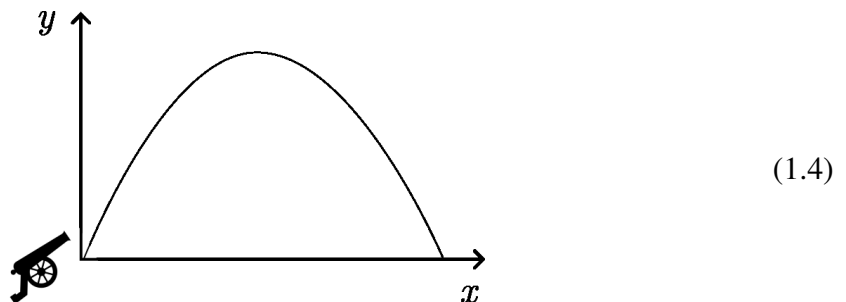
The parabola is given by the quadratic equation from high school:

$$F(x) = 0 \qquad F(x) = ax^2 + bx + c \tag{1.5}$$

but the trajectory can also be described by the physics equation parameterized by time t:

$$\dot{\mathbf{x}} = f(\mathbf{x}) = (v_x, v_y) = (v_x^0, -gt + v_y^0). \tag{1.6}$$

This parametric form is not unique. For example, another way is for the point $\mathbf{x}$ to move with uniform speed along the trajectory.

Note that the trajectory above is qualitatively the same as a "thought trajectory" in cognitive space. One intuitive way to visualize cognitive trajectories is via the example of a chess game tree (where the opponent's move is regarded as how the "world" reacts to the agent's actions):
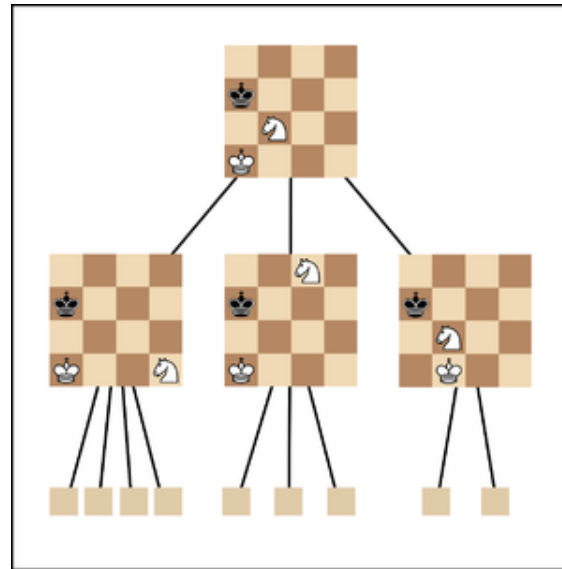
$$\tag{1.7}$$

$f(x)$ is the functional form we want, as it contains information about the "interestingness" of deduced conclusions.

Are our equations in Table 4.5 describing F or f?

An intuitive idea: state = facts = grounded equations, rules = quantified equations. So the equations are modeling f. This exposed a problem of classical AI that I have not paid too much attention to: the selection of interesting conclusions. It's hard to **enumerate conclusions**, let alone to rate their interestingness.

CHAPTER 2

BACKGROUND: CATEGORICAL LOGIC

Lambek posited this "trinity":

$$\begin{array}{c}
\text{type}\\
\text{theory}
\end{array}$$

$$\text{logic} \xrightleftharpoons[\text{internal language}]{\text{classifying topos}} \begin{array}{c}\text{category}\\\text{theory}\end{array}$$

with *Curry–Howard* and *obvious* labels. (2.1)
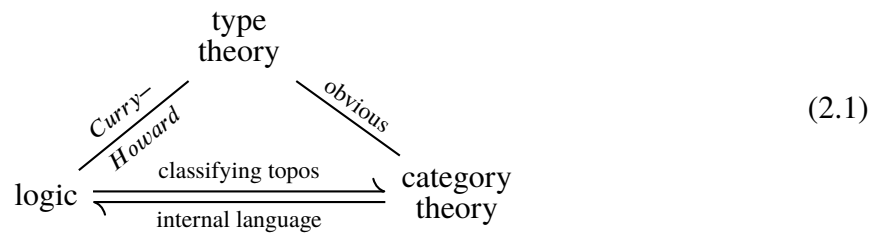
where the double arrows at the base can be understood thusly:

I extended some nodes to better see their relations:

$$\begin{array}{c}\text{type}\\\text{theory}\end{array}$$

$$\begin{array}{c}\text{algebraic}\\\text{logic}\end{array} — \text{logic} — \begin{array}{c}\text{category}\\\text{theory}\end{array} — \text{topos} — \begin{array}{c}\text{algebraic}\\\text{geometry}\end{array}$$

? (2.2)
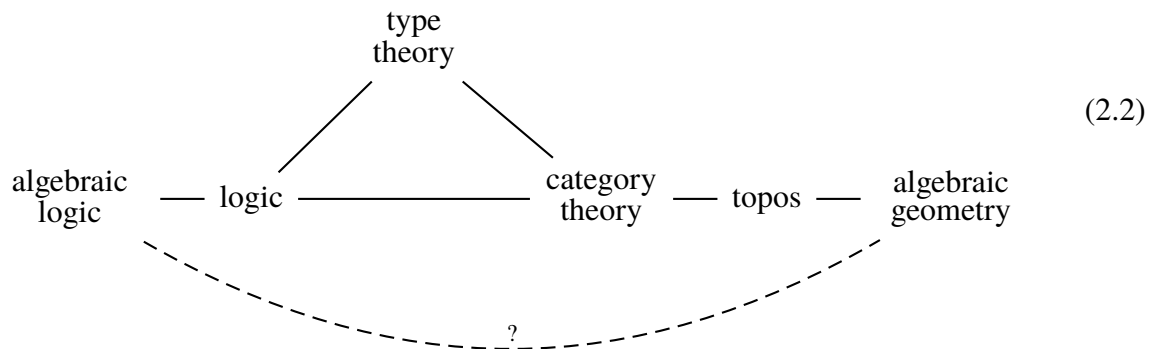
I am curious if the two algebras on the left and right are identical?

2.1 Topos and internal language

CHAPTER 3

BACKGROUND: ALGEBRAIC LOGIC

3.1 Paul Halmos' algebraic logic

The essence of Halmos' approach to algebraic logic may be explained by this diagram:

$$(3.1)$$
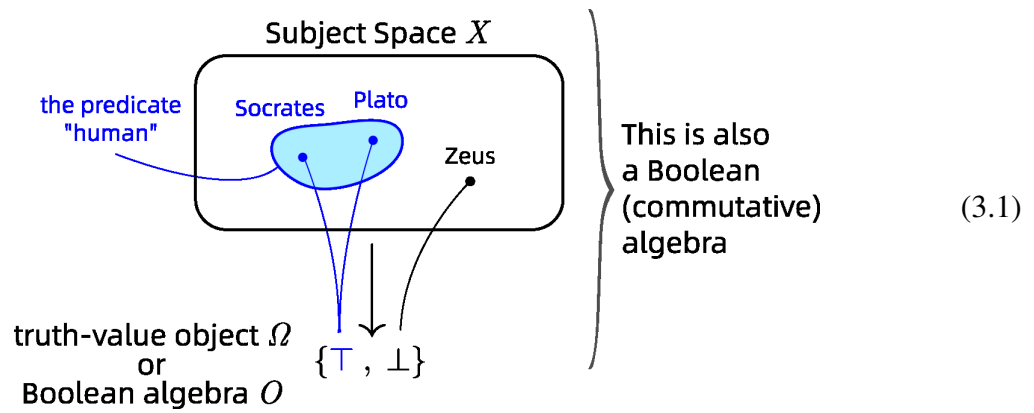
I coined the term "Subject Space" to refer to the space of **subjects** referred to by predicates. For example the predicate mortal(_) applies to subjects such as Socrates and Plato but does not apply to the subject Zeus.

We all know that propositional logic is isomorphic to the Boolean algebra of sets (Stone's duality theorem). This is the basis of treating logic as algebra. But the most sticky point of algebraizing a logic is the treatment of **predicates**. Halmos observed that one can form **propositional functions** from a Subject Space X to a Boolean algebra, and the resulting set of functions would still be a Boolean algebra. In the above diagram we can see a function mapping subjects to the set $\{\top, \bot\}$ which is the minimal Boolean algebra O, containing only 2 elements, true and false.

The target domain can be a more complex Boolean algebra, as in $X \to A$. For example, if Achilles was Zeus's son, he would also be immortal. So in the proposition space A, there would be an implication arrow immortal(Zeus) $\to$ immortal(Achilles). Note that the space A contains points which are just propositions, we cannot access their internal structure. Anyway, I find that $O = \{\top, \bot\}$ suffices for our purposes, and we don't need the richer structure of A.

More technically, we have the following duality. Every Boolean algebra $\mathbb{A}$ is isomorphic to

the set of all continuous functions from X into $\mathbb{O}$, where X is the dual space of the algebra $\mathbb{A}$, and $\mathbb{O}$ is the Boolean algebra with 2 elements. If there is a homomorphism f between Boolean algebras $\mathbb{A} \to \mathbb{B}$ then there is a dual morphism f^* between their dual spaces $Y \to X$:

$$
\begin{array}{ccc}
\mathbb{A} & \xrightarrow{\quad f \quad} & \mathbb{B} \\
\wr\wr & & \wr\wr \\
\overline{X} & \xleftarrow{\quad f^* \quad} & \overline{Y} \\
\downarrow & & \downarrow \\
\mathbb{O} & & \mathbb{O}
\end{array}
\tag{3.2}
$$

3.2 An "algebraic geometry" idea from Yuri Manin and Russians

Please see the next 4 pages excerpted from the book *Geometry I: Basic Ideas and Concepts of Differential Geometry*, by some Russian authors from 1991 [2]. They attributed the idea to Yuri Manin.

My understanding is that **Boolean algebra** and **Boolean rings** are essentially synonymous. Both can be regarded as a form of propositional logic.

Stone duality provides the link between Boolean algebra and the algebra of sets (in a topological space). But whether this link can be extended to predicate logic is unclear.

| Analysis | → | Differential topology | → | Homological algebra |

↓

| initial "algebra" | "geometry" directing it | secondary "algebra" regulating this "geometry" |
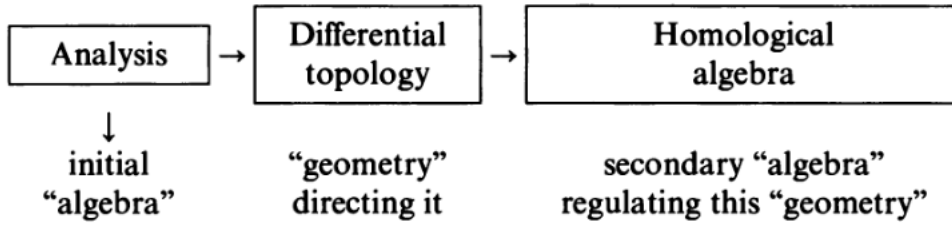
is an example of a large-scale wave in modern mathematics.

§2. Two Examples: Algebraic Geometry, Propositional Logic and Set Theory

2.1. In ancient manuscripts of geometrical content, instead of a proof of a theorem there is often displayed a convenient diagram together with the instruction "look". Thereby it is implicitly assumed that the contemplation of the diagram is capable of arranging the thoughts of the observer into a conclusive chain of deductions. Modern algebraic geometry is an approach to the solution of problems of commutative algebra by way of an intelligent and systematic construction of the necessary visual images. The source of this approach is an idea that goes back to Descartes: it is possible to obtain a visual representation of the set of solutions of a system of polynomial equations, interpreting it by introducing coordinates as an "algebraic subvariety" of affine space. In modern algebraic geometry the main way of visualizing is to interpret an arbitrary commutative algebra as the algebra of functions on some set. Its description in general outline reduces to the following (the definitions of the simplest concepts of commuatative algebra used below can be found, for example, in Volume 11 of the present series).

Let A be a commuatative algebra with unity over a field k. Consider the set Spec A of all prime ideals of this algebra. An element $a \in A$ can be thought of as a "function" on this set, whose value at a point $\mathfrak{p} \in$ Spec A is equal to the image of a under the natural homomorphism $A \to A/\mathfrak{p}$. As a result we obtain the following picture: to each point $\mathfrak{p}$ of Spec A we "attach" an algebra $A/\mathfrak{p}$ without divisors of zero (since $\mathfrak{p}$ is a prime ideal) and the "function" mentioned above, associated with the element $a \in A$, is the map $\mathfrak{p} \mapsto [a] \in A/\mathfrak{p}$. The algebra $A/\mathfrak{p}$, generally speaking, depends on $\mathfrak{p}$. If this dependence did not exist, that is, all the algebras $A/\mathfrak{p}$ were the same, then the construction we have described would lead us to the usual $A/\mathfrak{p}$-valued functions on Spec A.

Example. Let $k = \mathbb{C}$ (the field of complex numbers) and let $A = \mathbb{C}[x]$ be the algebra of polynomials with complex coefficients in the variable x. Any non-zero prime ideal $\mathfrak{p} \subset A$ consists of polynomials divisible by $x - c$ (the number $c \in \mathbb{C}$ is fixed). Dividing a given polynomial $p(x) \in A$ by $x - c$ and taking the remainder, we obtain a unique representation in the form $p(x) = p_0 + (x - c)h(x)$, where $p_0 \in \mathbb{C}$. Then $A/\mathfrak{p} = \mathbb{C}$ if $\mathfrak{p} \neq \{0\}$, and we can represent the operation of factoriza-

tion $A \to A/\mathfrak{p}$ as the operation $p(x) \mapsto p_0$, that is, the value of the polynomial $p(x)$ in the ideal $\mathfrak{p} \in \operatorname{Spec} A$ is equal to p_0. If we now identify the ideal $\mathfrak{p}$ with the number c that determines it uniquely, we see, in view of the equality $p_0 = p(c)$, that the usual meaning of the words "value of a function" and the one described above coincide.

Example. If $k = \mathbb{R}$ (the field of real numbers), then in the algebra $A = \mathbb{R}[x]$ of polynomials with real coefficients there are two types of nonzero prime ideals. An ideal of the first type consists of polynomials divisible by the binomial $x - c$ (the number $c \in \mathbb{R}$ is fixed). Ideals of the second type consist of polynomials divisible by a fixed trinomial $x^2 + px + q$, where $p, q \in \mathbb{R}$ and $4q > p^2$. If $\mathfrak{p}$ is an ideal of the first type, then $A/\mathfrak{p} = \mathbb{R}$, and if it is of the second type, then $A/\mathfrak{p} = \mathbb{C}$. In particular, the set of prime ideals of the first type can be identified with the "real" line $\mathbb{R}$. If in the usual way we consider a polynomial $p(x) \in A$ as a a function on $\mathbb{R}$, then this function coincides with the restriction of the "function" associated with it by means of the construction carried out above to the set $\mathbb{R} \subset \operatorname{Spec} A$ of ideals of the first type.

The latter example shows that we can diminish the dependence of the algebra $A/\mathfrak{p}$ on $\mathfrak{p}$ by considering not the whole set $\operatorname{Spec} A$, but only a suitable part of it. We illustrate this by an example that is fundamental for differential geometry.

Example. Let $\Omega \subset \mathbb{R}^n$ be a domain, let $k = \mathbb{R}$, and let $A = C^\infty(\Omega)$ be the algebra of all infinitely differentiable functions defined on Ω. With any point $a \in \Omega$ we can associate an ideal μ_a of A by putting

$$\mu_a = \{ f(x) \in A \mid f(a) = 0 \}.$$

It is not difficult to see that μ_a is a maximal ideal, and so $\mu_a \in \operatorname{Spec} A$ and $A/\mu_a = \mathbb{R}$. The value of f on μ_a, according to the basic construction, is equal to its usual value $f(a)$ at the point a. For this reason any function f as an element of A is uniquely determined by its values on the subset $\operatorname{Specm}^{\mathbb{R}} A \subset \operatorname{Spec} A$ consisting of all maximal ideals $\mu \subset A$ such that $A/\mu = \mathbb{R}$. In fact, we have the fundamental equality

$$\Omega = \operatorname{Specm}^{\mathbb{R}} C^\infty(\Omega).$$

This equality remains true if Ω is replaced by an arbitrary smooth manifold (see below).

The set $\operatorname{Spec} A$, and also the subsets of it that it is expedient to consider, are equipped in a natural way with additional geometrical structures, for example a topology, that visually reflect the properties of the original algebra A. Working on this basis, we can construct visual models for the basic concepts of commutative algebra. For example, an ideal in $A \Rightarrow$ "submanifolds" in $\operatorname{Spec} A$, a module over $A \Rightarrow a$ "bundle" (see p. 59 below) over $\operatorname{Spec} A$, and so on. Below we give some visual interpretations of the "differential" aspects of commutative algebra. A systematic use of such models enables us to construct visual interpretations of problems of commutative algebra, which play the same role in the proof as the diagram in classical geometry.

The spirit of the interaction of the algebraic and geometrical origins in algebraic geometry was expressively conveyed by A. Weil, who said that geometrical intuition is "invaluable so long as we recognize its limitations".

2.2. It is very instructive to apply the principle of visualization in algebraic geometry to the analysis of propositional logic.

The simplest method of organizing a certain set of propositions into a system is based on the following rule for combining them. Firstly, from two propositions P and Q we can form new ones: $P \vee Q$ (conjunction) and $P \wedge Q$ (disjunction). Secondly, with any proposition P we can associate its negation $\bar{P}$. If a certain set of propositions $\mathcal{A}$ is closed under the operations of conjunction, disjunction, and negation, then we can define on it the structure of a commuatative ring, introducing the operations of addition "+" and multiplication "·" according to the rule

$$P + Q = (P \vee Q) \wedge (\overline{P \wedge Q}), \qquad P \cdot Q = P \wedge Q.$$

'+' = XOR

The empty proposition is the zero of this ring, and its negation is the unity. Obviously,

$$P^2 = P.$$

A ring K with unity in which $a^2 = a$ for all $a \in K$ is called a *Boolean algebra*. If $a, b \in K$, then $a + b = (a + b)^2 = a^2 + ab + ba + b^2 = a + b + ab + ba$. Hence $ab + ba = 0$. In particular, if $a = b$, then it follows that $2a = a + a = a^2 + a^2 = 0 \Leftrightarrow a = -a$.

These identities show that any Boolean algebra is commutative and is an algebra over the field $\mathbb{F}_2$ of residues mod 2.

Thus, any closed universe of propositions is a Boolean algebra. Since any Boolean algebra is commutative, we can visualize it by the method described above.

It is not difficult to verify that any prime ideal μ of a Boolean algebra $\mathcal{A}$ is maximal and $\mathcal{A}/\mu = \mathbb{F}_2$. For this reason any element $P \in \mathcal{A}$ can be understood as an $\mathbb{F}_2$-valued (that is, taking values 0 and 1) function on the set Spec A. The subset $M_P \subset$ Spec A consisting of those points where this function is equal to unity uniquely determines this function, and hence the element P itself. We have

$$M_{P+Q} = (M_P \cup M_Q) \backslash (M_P \cap M_Q),$$

$$M_{P \cdot Q} = M_P \cap M_Q,$$

$$M_{P \vee Q} = M_P \cup M_Q, \qquad M_{P \wedge Q} = M_P \cap M_Q.$$

These relations show that any Boolean algebra $\mathcal{A}$ is isomorphic to a certain algebra of subsets of Spec A with respect to the operations of symmetric union (or symmetric difference) and intersection (product). (The symmetric union of subsets M_1 and M_2 is the subset $(M_1 \cup M_2) \backslash (M_1 \cap M_2)$.) This isomorphism is carried out by the map $P \to M_P$. The operation $(P, Q) \to P + Q + PQ$ in a Boolean algebra corresponds to the more usual operation of union for subsets. If A is finite, then any subset of Spec A has the form M_P for some $P \in A$. If A is

3.3   "Term rewriting and all that"

From the book of that title [3]. In Chapter 8, *Gröbner bases and Buchberger's Algorithm*, a correspondence is made between Huet's **higher-order unification** algorithm and Buchberger's algorithm for finding Gröbner bases.

An interesting question is whether this correspondence fits into our framework of algebraic logic. I need more time to determine this...

AI logic is more general than this

Equational logic, with equality predicate "="
able to decide A =? B questions (Ch 4)
also known as the "word problem"

Gröbner bases (calculated by Buchberger algorithm) (Ch 8)   (3.3)

can efficeintly decide the **ideal membership** problem:
given $f, f_1, ..., f_k \in K[X_1, ..., X_n]$
ask $f \in \langle f_1, ..., f_k \rangle$?

the ideal congruence problem can be regarded as a word problem
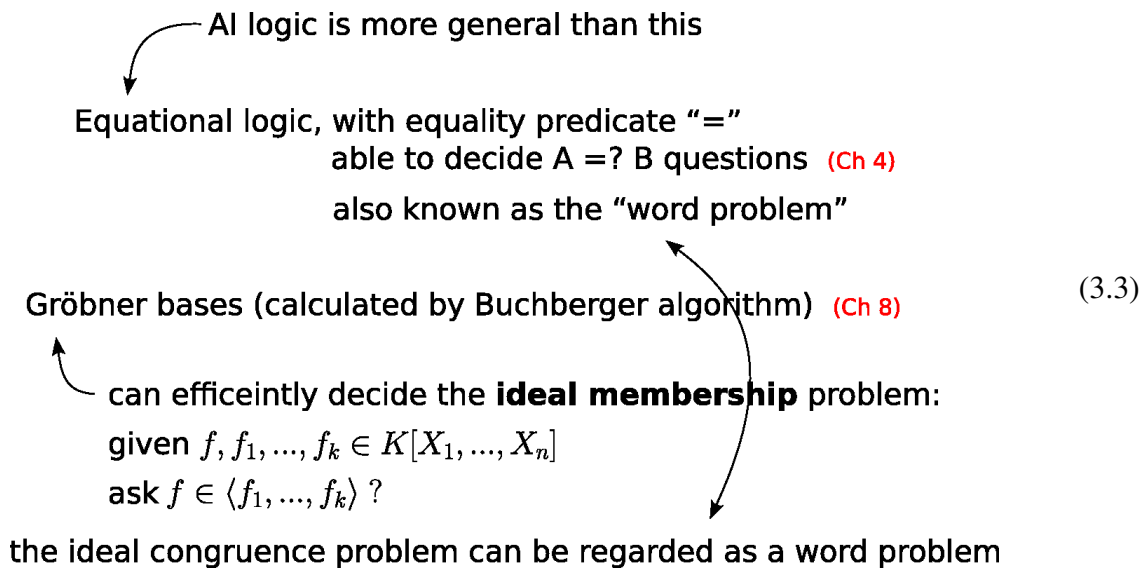
- What is the **word problem**?

  is defined for an equational theory E.

  is the problem of deciding whether $s = t$

- why is Gröbner basis equivalent to the word problem

  to ask ideal congruence $f = ?g$ means $f - g \in ?J$

  which is ideal membership problem

  a polynomial can be regarded as a rewrite rule

  because $f = 0$, we can take the "largest monomial" in f as the LHS, and the rest of f as RHS.

  In other words: ideal = set of rules

  We ask if a polynomial can be rewritten by the set of rules to another form.

  This is similar to logic deduction.

- Here an important question is: polynomial reduction seems unable to handle **logic variables**, it seems only capable of **simple symbolic rewriting**.

- logic is equivalent to what form of polynomials?

  taking the cue that Huet's higher-order unification = Buchberger algorithm, ...

15

CHAPTER 4

DESIGN OF ALGORITHM

The following table depicts the main correspondences relevant to our research:

LOGIC	facts	rules
	human(socrates)	$\forall x.\text{human}(x) \rightarrow \text{mortal}(x)$
ALGEBRA	**element**	**element**
	$p \in \mathbb{A}$	$(p \rightarrow q) \in \mathbb{A}$
WORLD	**states**	**state transitions**
	x_t	$x_t \overset{F}{\mapsto} x_{t+1}$

(4.1)

The relation between LOGIC and WORLD has been elucidated quite thoroughly in the AI literature. Note that the state x_t is made up of a set of facts (logic propositions). A single step of logic inference results in a new conclusion δx which is *added* (as a set element) to the current state x_t to form a new state x_{t+1}. Here t refers to "mental time" which does not necessarily coincide with real time.

4.1 From abstract algebraic logic to concrete computations

There are two main routes to make abstract algebraic logic concrete:

- Find **matrix representations** of the logical algebra
- Implement the logical algebra as the commutative algebra of (classical) **polynomials**

4.2 What does it mean to train the AI?

From the previous section,

$$F(x) = 0 \quad \text{is the solution to} \quad \dot{x} = f(x) \tag{4.2}$$

and the two descriptions (by F or by f) are equivalent.

The discrete version of f is $\mathcal{f}$:

$$\mathcal{f}(x_t) = x_{t+1} - x_t = \delta x \tag{4.3}$$

The sensory data from the AI are a set of "world" points $\{x_i\}$ and we require either:

$$F(x_t) = 0 \quad \text{or} \quad \mathcal{f}(x_t) = \delta x = x_{t+1} - x_t \tag{4.4}$$

and F or f can be trained by gradient descent to eliminate errors in the above conditions (equations).

- the x_t's are represented as **logic facts**
- F or f is represented as **logic rules**

and we need to **evaluate** $F(x_t)$ or $f(x_t)$.

Let's do some examples:

Logic formula	Algebraic form
human(socrates)	h(s) = 1
human(socrates) $\wedge$ human(plato)	h(s) $\cdot$ h(p) = 1
human(socrates) $\rightarrow$ mortal(socrates)	1 + h(s) + h(s) $\cdot$ m(s) = 1
$\forall x.\, \text{human}(x)$	$h(x)$ is a propositional function $\forall_x h(x)$ is a constant function mapping to 1 or 0
$\forall x.\, \text{human}(x) \rightarrow \text{mortal}(x)$	$\forall_x \big(1 + h(x) + h(x) \cdot m(x)\big)$ is a constant function mapping to 1 or 0
$\forall x, y, z.\, \text{father}(x, y) \wedge \text{father}(y, z) \rightarrow$ grandfather(x, z)	$\forall_x \forall_y \forall_z \big(1 + f(x, y) \cdot f(y, z) + f(x, y) \cdot f(y, z) \cdot g(x, z)\big)$ $\mapsto 0$ or 1
general Horn formula: $\forall_{x...} P \wedge Q \wedge R... \rightarrow Z$	$\forall_{x...} \big(1 + P \cdot Q \cdot R.... + P \cdot Q \cdot R... \cdot Z\big)$ $\mapsto 0$ or 1

$$\tag{4.5}$$

Now imagine there are millions of such rules. Number of predicates obviously increases.

Does each equation require new variables, or can variables be re-used? Seems yes, can be re-used.

The **loss function** would be the sum of squared errors over all equations:

$$\mathcal{L} = \sum_{\text{eqns}} \epsilon^2 = \sum_i \big(\phi_i(x...) - 1\big)^2. \tag{4.6}$$

18

Learning means to perform the **gradient descent** via $\nabla_\Phi \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \Phi}$ where Φ is the set of parameters for the equations.

Potential problems:

- How to represent the set of equations efficiently? Matrix of coefficients seems wasteful.
- We have lost the "**deepness**" of deep learning, but there is recent research showing that **shallow learning** may work well too.
- Need to iterate logical inference multiple times using the same set of equations.

How are new conclusions added to the state? What is the state? State = set of facts = set of **grounded** equations.

Inference: how to get from current state to next state? Big problem!! New ground facts have to be read off from satisfaction of all equations. Rather intractable...

See Chapter 1.

How to enumerate conclusions? The loss function (4.6) can be trained on any data with correlations. But what we have is data in the form of time series. Need extra measures to ensure that equations only model $x_t \rightarrow x_{t+\Delta t}$. Now how to enumerate conclusions? From current state, iterate over all equations to generate new states. This is getting very close to the classical logic-based AI inference algorithm.

The **interestingness function** gives a probability distribution over conclusions given the current "context" (which we identify as the current state x_t): $\mathrm{Intg}(\delta x) = \mathbb{P}(\delta x | x_t)$. This function has to be learned. It has an equivariant structure due to the state as a set of propositions with permutation invariance.

One more efficiency problem: iterating through all equations is inefficient, which brings back the necessity of the classical **rete algorithm**: instead of matching rules against the state, we should match the current state against rules. In other words, instead of $\delta x := \bigcup_i \mathrm{rule}_i(x)$, perform $\delta x := \text{compiled-rules}(\Delta x)$, where Δx is the change of the current state from the previous state, and δx is the change from the current state to the next state.

But we may also avoid the rete algorithm by stacking logical equations into **layers**, thus getting an efficiency advantage similar to deep learning. A "single" step of logic inference would mean going through multiple layers of logic rules (equations).

4.3 "Geometric" logic inference algorithm

$$K \text{ trained predicates}$$

(4.7)

$$P \wedge Q \wedge R \wedge \ldots \rightarrow Z$$
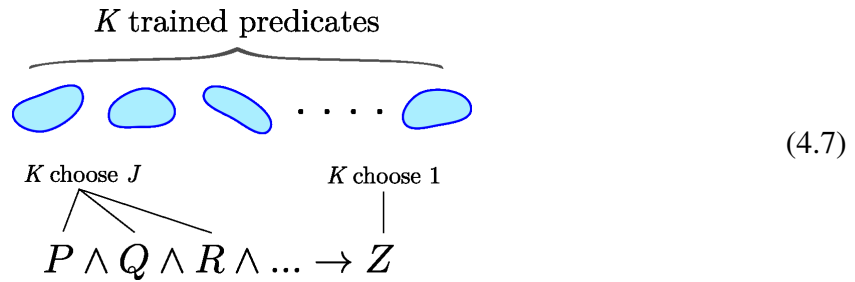
- What is a logic fact within a state?
- How does a rule generate a single new fact?

A fact consists of a point (in subject space) and a predicate that contains it. The point itself does not suffice because it can belong to various predicates.

4.3.1 How to determine if a rule is satisfied

To apply a rule, each **atomic term** in the rule has to be satisfied. For each predicate $Q()$, this is verified by testing if we have any points among the facts contained in Q.

If we have father(john, pete) as a fact then we certainly can satisfy father(x, y). But we already have the point (john, pete) which may satisfy other predicates $Q(x, y)$. So our method is slightly more permissive (and thus more powerful) in rules matching.

How does the rule's RHS generate a new fact? It should also be a (point, predicate) pair. The point has to **match** the premise. How could this be ensured?

Secondly, the output predicate may not cover the point.

The matching process: Syntactically, we look at each literal in the rule and see if any fact unifies with the literal. Geometrically, it means taking a point and checking if it lies inside a predicate. If the fact is a (point, predicate) pair then it is given that the point belongs to the predicate, so it is not necessary to check for membership. The result is simply taking the point when the predicates match. But we still need to keep track of which variables the point coordinates are binding to.

The matching of the second literal will also return a point, but its coordinates would be bound to different dimensions.

The binding of coordinates would be the basis of verifying the rule LHS.

RHS: The bound variables (in various dimensions) need to be projected to the output space. It may or may not be covered by the output predicate. If not, the rule does *not* apply. This is in accord with the principle that rules should not change during inference.

4.3.2 Handling variables in rules

Again drawing inspiration from Halmos' algebraic logic. A monadic logic deals with predicates as $X \to \Omega$, whereas in polyadic logic one has $X^I \to \Omega$ where I is an index set. The elements of I are **variables** even though they do not really change. X^I creates I copies of the Subject Space X. Note that $(I \to X) \to \Omega$ is not isomorphic to $I \to (X \to \Omega)$.

For example $I = \{1, 2, 3, 4\}$ would provide 4 variables or "slots" for a rule to use. A rule such as:

$$\text{father}(X_1, X_2) \wedge \text{father}(X_2, X_3) \to \text{grand-father}(X_1, X_3) \tag{4.8}$$

uses 3 variables: X_1, X_2, X_3. I have used the symbol X with subscripts to denote variables, as a reminder that each variable is really a **copy** of our Subject Space X.

It is perhaps illuminating to rewrite the logic rule this way:

$$\text{father}(X^I) \wedge \text{father}(X^I) \to \text{grand-father}(X^I) \tag{4.9}$$
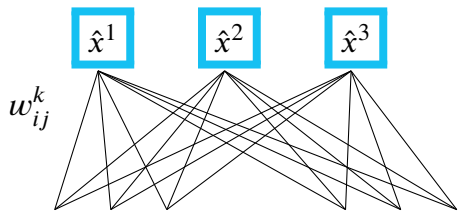
where I is at least $\{1, 2, 3\}$.

In my mind, I had this mental picture:

$$\text{father}(\;\rule[0pt]{2pt}{12pt}\rule[0pt]{2pt}{14pt}\rule[0pt]{2pt}{6pt}\;) \wedge \text{father}(\;\rule[0pt]{2pt}{6pt}\rule[0pt]{2pt}{14pt}\rule[0pt]{2pt}{12pt}\;) \to \text{grand-father}(\;\rule[0pt]{2pt}{12pt}\rule[0pt]{2pt}{6pt}\rule[0pt]{2pt}{12pt}\;) \tag{4.10}$$

where the blue bars represent some kind of relative "proportion" of variables, such that the overall construction could be **differentiable**.

The eventual implementation is quite natural:

$$w_{ij}^k \tag{4.11}$$

$$\text{father}(x_{11}, x_{12}, \bullet_{13}) \wedge \text{father}(\bullet_{21}, x_{22}, x_{23}) \to \text{grand-father}(x_1, \bullet_2, x_3)$$

Each weight represents how "strong" an argument (lower row) occupies a variable slot (upper row). **Softmax** normalizes the weights so that one of them would be most prominent, ie. the selected one. Every argument can potentially appear in every variable slot, thus the bipartite graph is fully connected.

The resulting formula is this: Each output slot $\hat{x}^k \in \hat{X}^I$ contains a **sum** of vectors from all

arguments of all predicates on the left hand side of a rule:

$$\hat{x}^k := \forall i \in_{\text{predicates}} \forall j \in_{\text{arguments}} \left\langle \operatorname{soft\,max}_{ij} w_{ij}^k , \ x_{ij} \right\rangle. \tag{4.12}$$

Note that ij is treated as one set of indices, so $x_{ij} = (x_{11}, x_{12}, x_{13}, x_{21}, x_{22}, x_{23})$ say, and $x_{ij}^\top$ is a simple column vector.

In the above method, the copying of a vector is not "pure" but may be contaminated by residuals of other vectors. This is tolerable, as the vector embedding of tokens can allow for small perturbations.

The structure of our formula is reminiscent of the Transformer's **Self-Attention**:

$$\operatorname{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \operatorname{soft\,max} \frac{\langle \mathbf{Q}, \mathbf{K} \rangle}{\sqrt{d_k}} \mathbf{V} \tag{4.13}$$

What we're doing here is to move variables around; this suggests that the Transformer is also capable of moving **tokens** around. The syntactic manipulation ability of Transformers has been corroborated by other researchers [cite].

For comparison, this is the **neural string diagram** for the traditional Self-Attention:
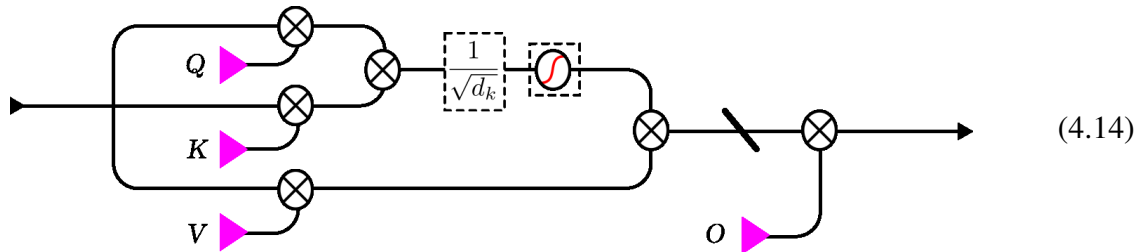
The training of the neural network for predicate P occurs in X^2 if the arity of the predicate is 2. This has nothing to do with X^I as the set I of variables are only involved in logic rules.

To evaluate a rule: Each predicate, such as father(_, _), is matched against some points, such as (a, b). The neural network returns a TV (truth value) independent of the variable space X^I. But we need to put the point (a, b) into X^I via the **Softmax** selector. This is repeated for all predicates on the LHS or "head" of the rule. Then the truth values will determine if the rule is satisfied or not. If yes, we will output a predicate such as grand-father(X_1, X_3). Now we go back to X^I to retrieve the point corresponding to the variables (X_1, X_3).

4.3.3 String diagram

K predicate vectors
[M rules, K predicates, |Ψ|]

select 1 predicate in RHS of rules
[M rules, K predicates]

select J out of K predicates in LHS of rules
[M rules, K predicates]

constants in rules
[M rules, K predicates, I constants, |X|]

cylindrification factors
[M rules, K predicates, I factors]

propositions from working memory {
predicates
[N propositions, |Ψ|]

points in Subject Space
[N propositions, |X|^I]

truth values
[N propositions]

neural nets for predicates
[K predicates, L layers = |X|, h..., 1]

substitutions
[M rules, K predicates, I slots, I slots]

RHS substitutions
[M rules, I slots, I slots]

$[M,K,|\Psi|]$ $[M,K]$ $[M,|\Psi|]$
output predicate

logits of $[M]$ conclusions

$[M,K]$

$[M,K]$

output args

$[M,|X|^I]$

$[M,K,|X|^I]$

I
① ② ③ ? $[M,K]$ $[M]$
TVs of rules

× L

$[N,|X|^I]$ $[K]$
TVs of predicates

$[N]$

I^2 $[M,|X|^I]$ a point for each output predicate

points in variable slots

$[M,|X|^I]$

(4.15)

Remarks:

- The **grayed-out** parts of the circuit tries to match the **predicates** of Working-Memory propositions to those in the rules. For example, the predicate loves where loves(John, Mary) matches loves(X_1, X_2). But I take the more liberal approach where matching succeeds whenever a point in Working Memory falls within the target predicate's region of definition, regardless of whether the source and target predicates match or not. For example, assume human(Socrates) is in Working Memory, and we want to match against mortal(X_1). This will succeed if the human region lies strictly within the mortal region in Subject Space. If the predicate regions are learned properly, this would be the case. Using this approach, rules are more easily satisfied, and the code seems to be simpler.

① We want to compare arguments of predicates in rules against propositions from Working Memory, ie, $\gamma(C) \stackrel{?}{=} c$, where γ is a cylindrification factor (turned into a radial basis function), C is a **constant** in a predicate in a rule, such as Mary in loves(John, Mary), to be matched against c, another constant argument of a predicate of a proposition from

23

Working Memory. In the above example, (John, Mary) is a **point** in the Subject Space X^2. John and Mary are also known as "coordinates" in X^2.

(2) On each step, we're not just comparing two elements, but two multi-dimensional matrices of elements. On the left side is a matrix of dimensions $[M, K, |X|^I]$, for M rules, each with K predicates, each with I argument slots, each containing an element of the Subject Space X. On the right side (from Working Memory), we have a matrix of dimensions $[N, |X|^I]$, for N propositions, each with I argument slots, each containing an element of the Subject Space X.

The operation $\overset{?}{=}$ is performed on the axis I, common to both sides. For each rule $m \in M$, for each predicate $k \in K$, for each slot $i \in I$, we compare C_{mki} against c_{ni} for each proposition $n \in N$, for each argument slot $i \in I$. This totals to $MKNI$ comparisons (the index i counts only once). This means trillions of comparisons, by our naive estimation of sizes.

(3) Each proposition in Working Memory has a truth value. This TV has to be multiplied to the result in (2).

4.4 Computer representation of rules

Each logic rule may vary in the following ways:

- number of literals and their polarity
- number of arguments
- each argument may be a logic variable or logic constant
- which specific logic variable or constant.

The numbers in #1 and #2 can be fixed.

For #3 and #4, each variable or constant may be represented by a pair (p, γ) where p is a point (or coordinates) in the Subject Space, and γ is the **cylindrification factor** representing the degree to which this argument is like a point or like a "for all" variable.

Each predicate has as its arguments n copies of the Subject Space, ie, X^n. One or more of

the copies may be **cylindrified**.

$$(4.16)$$
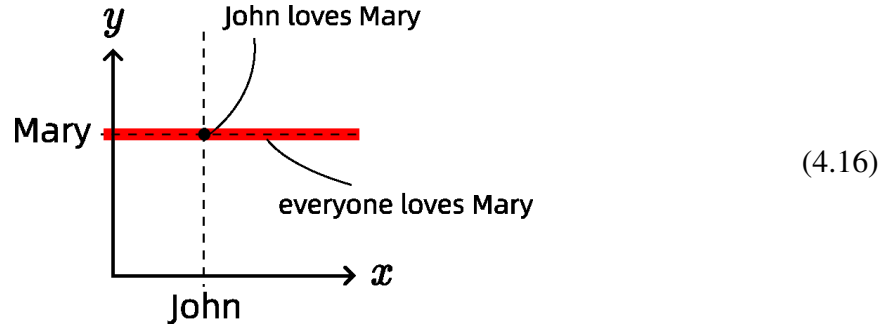
In the above example, we have two copies of the Subject Space X, each is 1-dimensional.

During rule matching, suppose a point $p = (a, b)$ is presented from a proposition $P(p) = P(a, b)$ in the state X. This is matched against a term $Q(X, Y)$ in the rule. The rule-matching process seems costly, which suggests we may need to re-formulate a neural-network version of the **Rete algorithm** – but such an algorithm would not be differentiable. In fact, differentiability dictates that all rules must participate in matching at all times (or at least probabilistically with non-zero probabilities).

Anyway, the coordinate a needs to be matched against the argument X_1, which may be a variable or a constant. Denote the constant part of X_1 as C_1, so we try the matching $a \overset{?}{=} \gamma(C_1)$, where γ is the cylindrification, a radial basis function. This returns the truth value of the match. At the same time, a would be copied into the variable slot X_1 for substitution processing, independently. This is OK, as the result will have a low TV if matching fails.

We suggest the following values (for a general human-level intelligent agent):

Meaning	Symbol	Est. typical values
# of rules	M	millions
total # of predicates	K	10,000's
# of predicates per rule	J	2-10, typical 3,4
# of arguments per predicate	I	2, 3, 4
Subject Space dimension	$\|X\|$	100-1000, typical 512, 768
Predicate Space dimension	$\|\Psi\|$	same as above
Hidden layer size	h	similar to $\|X\|$
# of propositions in working memory	N	100

$$(4.17)$$

Total number of parameters (in descending order of size) $= MKIX + MK\Psi + MKI^2 +$ $MKI + 2MK + MI^2 + NIX + N\Psi + N + K(|X| + (L-2)h) \approx 20$ trillion. Perhaps M and K have to be reduced.

4.5 Rules recommender

The rules recommender would be a set function:

$$\mathsf{Ru} : \{\text{current state}\} \rightarrow \{\text{set of rules}\} \tag{4.18}$$

which has equivariant structure on both its input and output. This suggests the **Transformer** architecture is suitable for learning this function.

But this is clearly **non-differentiable**!

4.5.1 Differentiability

In binary logic a rule either applies or does not apply. To make the entire set of rules differentiable, rule application must be "graded".

So we propose a **probabilistic** rule-matching mechanism:

$$\tag{4.19}$$
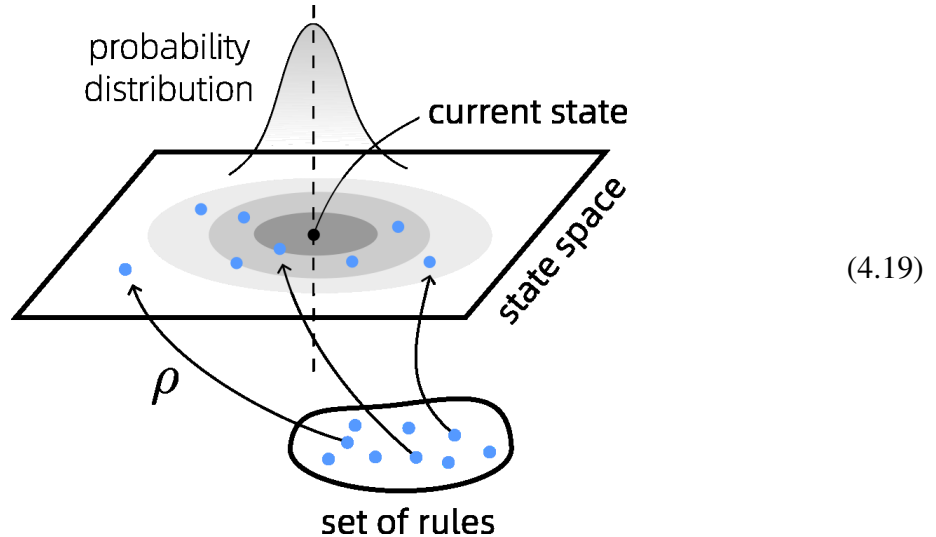
Rules are mapped by a function ρ to the state space, such that rules are located close to the states they are most likely to match (this is a multi-objective optimization problem). Given a current state x, denote the ball around it as $B(x, d)$. The rules we predict most likely to match x are given by $\rho^{-1}(B(x, d))$ for some distance d. We can assign a Gaussian probability distribution centered around x over such rules, so that rules are selected **probabilistically** for update.

It seems that doing so would not affect the convergence (of the learning algorithm) of the rules, but it would be much more efficient as the probability distribution is *more* concentrated around *x*, so that large numbers of rules need *not* be evaluated (for each state, at each time step).

$$x_{t+1} = F(x_t; \Theta) \tag{4.20}$$

where Θ are all the parameters that determine the rules, F is the total function aggregating all the rules. If we vary any one rule parameter, does it change abruptly? The parameters Θ include the rules-sorting function ρ.

For gradient descent we need to calculate $\nabla_\Theta \mathcal{L}$ where Θ are all the parameters of rules, and the loss function is defined as the sum of errors over all rules, $\mathcal{L} = \sum_{\text{all rules}} \epsilon^2$. Each error usually is given by the difference as compared against a reference value (supervised learning), but in our case such a value is unavailable, instead the objective comes from reinforcement learning, ie, the Hamilton-Jacobi-Bellman equation: $J(x_{t+1}) = \max_a [\gamma J(x_t) + R(x_t, a)]$, where *a* means **action**. In the logic-based setting, an action means applying a logic rule to change the state.

Policy function:

$$\pi : X \times A \to [0, 1] \tag{4.21}$$

By the **Policy Gradient Theorem**, calculation of $\nabla_\Theta J$ translates to calculating $\nabla_\Theta \pi$.

In reinforcement learning, in particular the **Policy Gradient** algorithm, we need to calculate the gradient $\nabla_\phi \mathcal{L}(\phi)$ of a loss function of the form:

$$\mathcal{L}(\phi) = \mathbb{E}_{z \sim q_\phi(z)}[f(z)]. \tag{4.22}$$

The expectation gives an integral $\nabla_\phi \mathcal{L}(\phi) = \nabla_\phi \int dz \, q_\phi(z) f(z)$ which removes the "randomness" and evaluates to a real number. This allows to be handled by traditional differential calculus.

At each state, the application of all rules results in a list of all available actions. The differentiability problem may arise from probabilistic rule-matching.

4.6   Interestingness

This can be implemented implicitly by making the inference algorithm output a probability distribution over all deduced conclusions and then picking the most probable one.

4.7 Combining logic and reinforcement learning

4.7.1 Logical policy function

The policy function $\pi : X \times A \rightarrow [0,1]$ should be implemented with logic structure. The current state would be matched against rules selected by the rules recommender, and then each rule would be applied, resulting in conclusions Z_i each associated with a probabilistic strength $\mathbb{P}(Z_i)$. These are the available actions and their probabilities as given by the policy.

This section concerns how to establish the "link" from logic to reinforcement learning.

For reinforcement learning, I find it easier to consider Q-learning, ie, learning the utility function $Q(s,a)$, where s = current state and a = action taken in current state.

During forward inference, each logic rule of the form (also known as Horn form):

$$P \wedge Q \wedge R \wedge ... \rightarrow Z \tag{4.23}$$

yields a conclusion Z. But the "truths" of the premises $P, Q, R, ..$ are not binary but fuzzy, because the rules have to be differentiable (this can be implemented by softmax). Thus each conclusion Z is also fuzzy. The application of all the rules yields a set of conclusions $\{Z_i\}$, with their associated fuzzy truth values, which we can normalize as a probability distribution over all available next actions. This can be regarded as the **policy function** $\pi(a|s)$, which gives the probability of an action given the current state, $\mathbb{P}(a|s)$.

In Q-learning we need to learn the values $Q(s,a)$. Given the reward R, $Q(s,a)$ satisfies the Bellman equation:

$$Q(s,a) \mathrel{+}= \eta\left(R + \gamma \max_{a'} Q(s',a') - Q(s,a)\right) \tag{4.24}$$

where η is the learning rate, γ is the discount rate of values.

The policy function $\pi(a|s)$ is not exactly the same as $Q(s,a)$, but there is a trick that can relate them, that is the basis of **Soft-Q learning** (Soft Actor-Critic, SAC):

$$\pi(a|s) \propto \exp Q(s,a) \tag{4.25}$$

The situation can be illustrated by the following figure (not a commutative diagram):

$$
\begin{array}{c}
\text{Bellman} \\
\text{update} \\
\downarrow \\
\text{logic} \atop \text{rules} \longrightarrow \pi(a|s) \underset{\exp}{\overset{\log}{\rightleftarrows}} Q(s,a)
\end{array}
\tag{4.26}
$$

Logic rules give us $\pi(a|s)$, which in turn gives us $Q(s,a)$, but $Q(s,a)$ is also determined by Bellman update.

My solution is: Logic outputs a finite set of actions, to each action is associated a weight, which we identify as Q. Use Bellman update to train these Q values. To choose an action, calculate $\pi(a|s)$ as suggested by the Soft-Q method. This has the effect of **maximum-entropy** exploration.

4.8   Basics of DQN (Deep Q Learning)

The **Bellman optimality condition** says that:

$$
Q_t^* = \max_a \{ R + \gamma Q_{t+1}^* \}
\tag{4.27}
$$

where ()* denotes optimal values.

In the **Bellman update**,

$$
Q_t \mathrel{+}= \eta \cdot \text{TD-error}
\tag{4.28}
$$

where TD = **temporal difference** is defined as:

$$
\text{TD-error} = \overbrace{R + \gamma \max_{a'} Q(s_{t+1}, a')}^{\text{ideal value}} \overbrace{- \ Q(s,a)}^{\text{current value}} .
\tag{4.29}
$$

So obviously the Bellman update causes the current value of Q to **converge** to that of the ideal value Q^*.

Remember that, in the simplest **Q-table** method, we apply the Bellman update over Q-table entries. The **DQN** approach differs in that, instead of a Q-table, we use a deep neural network in its place. So the updating of the Q-table now becomes updating the DQN via **gradient descent**.
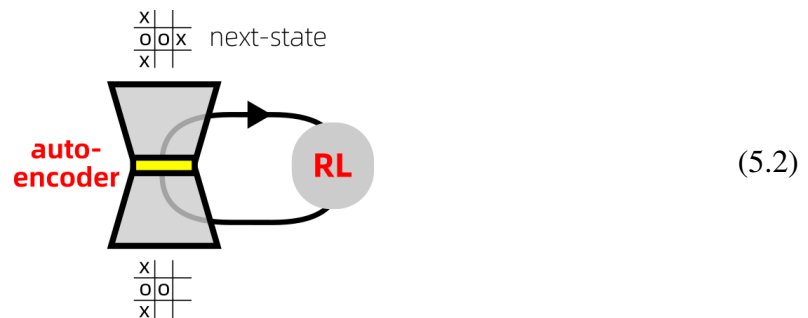
CHAPTER 5

COMBINING RL AND AUTO-REGRESSION

How can we combine reinforcement learning with auto-regressive learning?

(5.1)
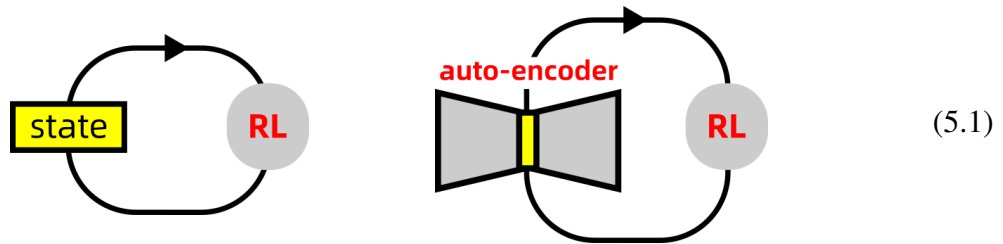
If we use RL to model internal mental states, then each action is an internal next thought. In this setting, what does auto-regression, or "prediction of the next state," mean?

Perhaps think about the example of Tic-Tac-Toe:

(5.2)

What we call "prediction" should be about external world-states.

Think about it: it does not make too much sense for auto-regression to predict the next *internal* thought; that would be highly redundant.

(5.3)

CHAPTER 6

EXPERIMENTS

6.1 Representation of states and rules

At any time the state is a set of facts, ie, pairs of (point $\in$ predicate).

There will be K predicates and M rules.

Each rule is a conjunction of all predicates. If K is large, the rules would be cumbersome.

Having a large number of rules, M, seems not to have a deleterious effect, if the rules recommender is good at its job.

Each literal in the rule may be negated, how to handle this?

Each literal contains a predicate and its arguments, which can be constants or variables.

It seems that genetic algorithms would be best suited for this kind of search for rules... or unless the rules are represented in such a way that they can continuously vary in a differentiable manifold.

For TicTacToe, let's say number of rules = 50.

CHAPTER 7

CONCLUSIONS

Some conclusion text.

REFERENCES

[1]   Halmos, *Algebraic logic*. (Dover 2016), 1962 (cit. on p. 3).

[2]   Alekseevskij, Vinogradov, and Lychagin, *Geometry I: Basic Ideas and Concepts of Differential Geometry*. Springer-Verlag, 1991 (cit. on p. 10).

[3]   Baader and Nipkow, *Term rewriting and all that*. 1998 (cit. on p. 14).

APPENDIX A

LIST OF PUBLICATIONS

APPENDIX B

FYTGS REQUIREMENTS

The requirements are from the RPG Handbook.

B.1 Components

B.1.1 Order

A thesis should contain the following parts in the order shown:

1. Title page, containing in this order:

    a. Thesis title

    b. Full name of the candidate

    c. Degree for which the thesis is submitted

    d. Name of the University, *i.e.* The Hong Kong University of Science and Technology

    e. Month and year of submission

2. Authorization page

3. Signature page

4. Acknowledgments

5. Table of contents

6. Lists of figures and tables

7. Abstract ($\leq$ 300 words.)

8. Thesis body

9. Bibliography

10. Appendices and other addenda, if any.

B.1.2 Authorization page

On this page, students authorize the University to lend or reproduce the thesis.

1. The copyright of the thesis as a literary work vests in its author (the student).

2. The authorization gives HKUST Library a non-exclusive right to make it available for scholarly research.

B.1.3 Signature page

This page provides signatures of the thesis supervisor(s) and Department Head confirming that the thesis is satisfactory.

B.1.4 Acknowledgments

The student is required to declare, in this section, the extent to which assistance has been given by his/her faculty and staff, fellow students, external bodies or others in the collection of materials and data, the design and construction of apparatus, the performance of experiments, the analysis of data, and the preparation of the thesis (including editorial help). In addition, it is appropriate to recognize the supervision and advice given by the thesis supervisor(s) and members of TSC.

B.1.5 Abstract

Every copy of the thesis must have an English abstract, being a concise summary of the thesis, in 300 words or less.

B.1.6 Bibliography

The list of sources and references used should be presented in a standard format appropriate to the discipline; formatting should be consistent throughout.

Sample pages of both MPhil and PhD theses are provided here (MPhil / PhD), with specific instructions for formatting page content (centering, spacing, etc.).

B.2 Language, Style and Format

B.2.1 Language

Theses should be written in English.

Students in the School of Humanities and Social Science who are pursuing research work in the areas of Chinese Studies, and who can demonstrate a need to use Chinese to write their theses should seek prior approval from the School via their thesis supervisor and the divisional head.

If approval is granted, students are also required to produce a translation of the title page, authorization page, signature page, table of contents and the abstract in English.

B.2.2   Pagination

1. All pages, starting with the Title page should be numbered.
2. All page numbers should be centered, at the bottom of each page.
3. Page numbers of materials preceding the body of the text should be in small Roman numerals.
4. Page numbers of the text, beginning with the first page of the first chapter and continuing through the bibliography, including any pages with tables, maps, figures, photographs, etc., and any subsequent appendices, should be in Arabic numerals.
5. Start a new page after each chapter or section but not after a sub-section.

Note: That means the Title page will be page i; the first page of the first chapter will be page 1.

B.2.3   Format

1. A conventional font, size 12-point, 10 to 12 characters per inch must be used.
2. One-and-a-half line spacing should be used throughout the thesis, except for abstracts, indented quotations or footnotes where single line spacing may be used.
3. All margins—top, bottom, sides—should be consistently 25mm (or no more than 30mm) in width. The same margin should be used throughout a thesis. Exceptionally, margins of a different size may be used when the nature of the thesis requires it.

B.2.4   Footnotes

1. Footnotes may be placed at the bottom of the page, at the end of each chapter or after the end of the thesis body.
2. Like references, footnotes should be presented in a standard format appropriate to the discipline.
3. Both the position and format of footnotes should be consistent throughout the thesis.

B.2.5 Appendices

The format of each appended item should be consistent with the nature of that item, whether text, diagram, figure, etc., and should follow the guidelines for that item as listed here.

B.2.6 Figures, Tables and Illustrations

Figures, tables, graphs, etc., should be positioned according to the scientific publication conventions of the discipline, e.g., interspersed in text or collected at the end of chapters. Charts, graphs, maps, and tables that are larger than a standard page should be provided as appendices.

B.2.7 Photographs/Images

1. High contrast photos should be used because they reproduce well. Photographs with a glossy finish and those with dark backgrounds should be avoided.
2. Images should be dense enough to provide 300 ppi for printing and 72 dpi for viewing.

B.2.8 Additional Materials

Raw files, datasets, media files, and high resolution photographs/images of any format can be included.

Note: Students should get approval from their department head before deviating from any of the above requirements concerning paper size, font, margins, etc.