Automated Logical Reasoning
Decision of theory

KAIST SoC & Mathmatics 17 Kim Tae Young

August 2, 2019

1 Abstract

현재의 전산학에서 언급하지 않고 넘어갈 수 없는 주제를 단 하나만 고르라면, 십중팔구 AI, 즉 인공지능을 고를 것이다. 그리고 대부분의 일반인들이 AI에 가지는 궁금증 중 하나로, "AI가 제 일자리를 빼앗을까요?"이다. 그래서 이번 교류전 세미나는 수학자들의 일자리는 AI에 의해 빼앗길 것인가? 에 대하여 이야기해보려한다. 수학자들이 하는 일이 무엇인가를 생각해보면, 정리의 증명이다. 즉 만약 AI가 수학자들의 일자리를 빼앗는다면 그 이름은 Automated Theorem Prover가 될 것이다. 이 개념이 그렇게 새로운 개념은 아니다. 이 분야의 가장 기본이라고 할만한 SAT-Solver의 일종인 DPLL algorithm은 1962년에 처음 등장하였기 때문에 수학사에서 본다면 최신이지만, 전산학사에서 본다면 상당히 오래된 개념이다.
먼저 '컴퓨터가 증명한 정리' 중 가장 유명한 정리를 생각해보자. 아마 많은 사람들이 4색 정리를 고를 것이다. 4색 정리에서 사용된 '컴퓨터를 이용한 증명'은 완벽하지 않다. 즉, 이 증명 자체를 컴퓨터로만 재현하는 것은 불가능한 것이다. 하지만 상당히 중요한 점을 서사해준다. 즉, 컴퓨터가 우리가 하는 증명을 도와줄 수 있다는 것이다.

2 Background

2.1 Proof-Assistant

그렇게 등장하는 개념이 Proof Assistant이다. 마치 논문 제출을 할 때 peer review를 하듯이 컴퓨터가 우리의 증명이 제대로되었는지 확인하게 하는 프로그램이다. 대표적으로 Coq, HOL이 있는데 결론적으로 이러한 프로그램도 수학자의 일자리를 빼앗지는 못한다. 이들을 Interactive Theorem Prover라고 하는데 자기 스스로는 증명할 수 없으나 사람이 일련의 명령어를 입력하여 자신의 증명이 맞는지 확인할 수 있는 프로그램이다. 이후에 이들을 발전시켜 Mizar, Isabelle, HOL4, Lean 같은 다양한 Theorem proving enviroment들이 주어졌다. 그리고 최근의 ML 열풍에 힘입어 HOLIST라는 강화학습을 통한 정리 증명 AI도 등장한 바가 있다. 다만 다행히도 아직까지 증명 성공율은 40%를 넘지 않았으며, 새로운 증명을 한 것도 아니다. HOLIST가 거의 최초의 Deep Reinforcement Learning based

Automated Theorem Prover라는걸 감안하면 괜찮은 편이긴하다. (물론 MNIST가 linear classifier로 에러율이 7%이었던걸 생각하면 굉장히 낮은 수치이다.)

결국 Coq, HOL 등은 증명을 확인할 수 있었지만 결국 사람이 일련의 택틱을 입력해주었어야하며, HOLIST같은 경우는 스스로 증명을 할 수 있었으나 40%의 낮은 성공율과 Intellectual Debt라고 하는, 원리가 이해된 상태로 작동하지 않는 딥러닝의 큰 약점에 빠지고 만다. 즉, 명제를 주었을때 그 명제를 증명하는 프로그램으로는 둘 모두 부족하다.

2.2 Mathmatical Logic

이제 다시 처음의 문제로 돌아가보자. 우리의 목적은 Automated Theorem Prover를 만드는 것이기 때문에 이것이 무엇인가를 생각해보도록하자. 먼저 Theorem, 그리고 Prover라는 말을 명확하게 하기 위해 수리논리학을 간단하게 짚고 넘어가려고 한다.

먼저 일차 논리는 주어진 constant, function, predicate symbol들과 이들을 이용한 boolean logic, 그리고 이 모델 상의 모든 원소에 대한 existential quantifier, universal quantifier로 만들어진다. 이후 이차 논리는 모델 상의 원소들의 집합, 함수, 명제에 대한 quantifier를 포함한다. 예를 들어 다음과 같은 식들이 일차 논리로 나타낼 수 있다.

$$\forall x.\exists y.(x = y + y \vee x = y + y + 1)$$

$$\forall n.(n > 2) \to \neg(\exists x.\exists y.\exists z.x^n + y^n = z^n)$$

$$\forall n.\exists p.(n < p) \wedge \forall q.(q < p) \to \neg\exists r.(p = qr)$$

즉, 정리란 n차 논리로 작성된 식 중 참인 것을 말합니다. 이번 세미나에서는 일차 논리에 한정하여 다루려고 하지만, 실제 Automated Logical Reasoning에서는 Monadic Second Order Logic이라는 set of element에 대한 quantifier를 포함하는 이론을 다루는 경우도 있습니다.

결론적으로 Automated Theorem Prover라는 것은, 식이 주어졌을때 이 식이 참인지 거짓인지 판별하는 알고리즘을 말하게 됩니다.

2.3 Decision of Theory

먼저, Theory를 명확히 정의하려면 signature와 structure가 필요합니다. 프로그래밍 언어에 익숙하신 분들은 각각 syntax와 semantic 문법을 이야기한다고 생각하여도 좋습니다.

Definition 2.1. *A signature is 3-Tuple* (C, F, P) *where* C *is set of constant symbols,* F *is set of function symbols,* P *is set of predicate symbols, and we have countably many variables* $\{x_1, \ldots\}$
Term over signature is either constant symbol, variable, or $f(t_1, \ldots, t_k)$ *for k-ary function symbol f, terms* $t_1, \ldots t_k$ *(First order logic) Formula over signature is either* $p(t_1, \ldots, t_k)$ *for k-ary predicate symbol p and terms* $t_1, \ldots, t_k$*, negation* $(\neg G)$*, conjunction* $(G \wedge H)$*, disjunction* $(G \vee H)$*, existential quantified formula* $(\exists x.G)$*, universal quantified formula* $(\forall x.G)$*.*

이제 structure는 signature 상에서 정의가 되는데, 간단하게는 signature의 symbol들, constant, function, predicate symbol들에 의미를 부여해준다고 보면 쉽습니다.

Definition 2.2. *A structure over signature is defined as universe U, mapping each constant symbols to element in U, mapping each function symbols to function on U, and mapping each k-ary predicate symbols to k-ary relation over universe, so that each predicate is true if value of terms are in image.*

이제 여러 이론을 생각해보자. 먼저, 가장 유명한 이론으로 Peano Arithmetic, $Th(\mathbb{N}, 0, 1, +, \times, =, <)$이 있다. 비슷하게 Presburger Arithmetic은 Peano Arithmetic에서 곱셈을 제외한, 즉 $Th(\mathbb{N}, 0, 1, +, =, <)$이다.

어느 이론이 Complete하다는 것은 signature로 생성되는 모든 식이 True거나 False임을 말합니다. 어느 이론이 Decidable하다는 것은 signature 상의 식을 입력받았을 때 그 식이 True거나 False임을 출력하는 알고리즘이 존재한다는 것을 말합니다. 즉, Decision of Theory라는 말은 이 알고리즘의 존재성에 관한 문제입니다.

3 Decidable Theory

이제 실제로 우리의 직업을 위협할 수 있는 AI를 만들어보고자 합니다. 여기서 사용할 이론은 위에서 소개한 Presburger Arithmetic보다 약간 강력한, $Th(\mathbb{N}, 0, 1, +, <, \{c|\cdot\}_{c>0})$을 이용합니다. 이 이론이 강력한 편은 아니지만 상당히 유용하기 때문에 Coq에서는 지금 증명할 알고리즘을 이용한 택틱(Omega)이 존재합니다. 즉, 덧셈으로만 이루어진 자연수 상에서의 연산이라면 자동으로 증명해줍니다.

먼저 좀 더 일반적인 이론을 사용하려고 합니다.

Definition 3.1. *A theory admits Quantifier Elimination(QE) if for every quantifier free formula F there exists quantifier free formula G such that $\exists x.F \equiv G$. If such G is computable, we say theory has quantifier elimination precedure.*

Theorem 3.1. *A theory is decidable if quantifier free formula of theory is decidable, and theory has quantifier elimination procedure.*

Theorem 3.2. *Extended Presburger Arithmetic $Th(\mathbb{N}, 0, 1, +, <, \{c : \cdot\}_{c>0})$ has quantifier elimination procedure.*

Corollary 3.1. *Extended Presburger Arithmetic is decidable. In particular, Presburger Arithmetic is decidable.*

Note that QF is trivially decidable.

Theorem 3.3. *Presburger Arithmetic is decidable in time $2^{2^{2^{O(n)}}}$, Triple Exp.*

이 외에도 Decidable한 Theory는 생각보다 많습니다. 예를 들어 실수 상에서의 First Order Logic Theory(Tarski, 1949), 또는 Presburger Arithmetic과 비슷한 Skolem Arithmetic($Th(\mathbb{N}, 0, 1, \times, =)$)은 Decidable합니다.

이번 세미나에서는 Decision of Theory에서 가장 강력한 툴 중 하나인 Quantifier Elimination을 사용하였지만, 모든 Deicision of Theory가 Quantifier Elimination

을 사용하지는 않습니다. 대표적으로 실수 상에서 polynomial과 equality만으로 생겨진 이론은 Quantifier Elimination을 허용하지 않습니다. $\exists x.x^2 + y^2 = 1$이 $-1 \leq y \leq 1$로 나타나는 것을 보면 알 수 있습니다. 다만 실대수기하의 기본 정리 중 하나로 Semi-algebraic Set, 즉 polynomial과 등호와 부등호들의 boolean combination으로 나타난 식의 projection은 다시 Semi-algebraic set이라는 정리가 존재하며, 이 식이 computable하다는 정리 또한 존재합니다. 그리고 이 알고리즘은 Cylindrical Algebraic Decomposition이라고 부릅니다.

이 정리가 어디에 사용될 수 있는가를 생각해보면, 당연하지만 Presburger Arithmetic으로 나타낼 수 있는 모든 정리는 증명할 수 있습니다. 위에서 작성한 $\forall x.\exists y.(x = y + y \vee x = y + y + 1)$, 이나 Chicken McNugger Problem으로 잘 알려진 $\exists n.(c > n) \rightarrow (\exists x.\exists y.c = \overbrace{x + \ldots + x}^{a} + \overbrace{y + \ldots + y}^{b})$ 같은 것들을 말이죠. 두번째로, Abelian Group이나 Boolean Algebra에 대해서도 decidability가 많이 증명되어있는데, 이 중 전산학에서 Automata를 사용하여 나타낼 수 있는 Theory에 대해서 두 theory가 isomorphic함이 decidable임이 증명되어있습니다. 다른 말로 하면, isomorphism theorem을 부분적으로 풀 수 있습니다. 마지막으로 실제 적용에 대해 잠깐 이야기해보자면 소프트웨어 테스팅에서 버그 발생 조건을 First Order Logic으로 작성한다면 그 프로그램이 버그를 발생하는지 아닌지를 판별할 수 있습니다. 또는 Type System에 대한 증명 역시 비슷하게 증명하는 경우가 많습니다.

4 Undecidable Theory

이제 세미나의 두번째 섹션으로 넘어가겠습니다. 첫번째 섹션에서 증명한 것과 같이 수학자들의 직업을 위협하는 AI는 실제로 존재합니다. 그리고 세미나에서 따로 증명하지는 않겠지만 위와 같이 컴퓨터를 이용한 증명으로 모든 자연수는 4개의 binary palindrome의 합으로 나타내어진다, 라는 정리가 있습니다. 즉, 이미 컴퓨터는 수학을 하고 있습니다.

이제 다시 처음으로 돌아가보겠습니다. 저희의 질문은 "컴퓨터가 수학을 할 줄 아는가?" 가 아니라, "컴퓨터가 수학자들의 일자리를 없앨 수 있는가?" 였으니까요. 결론부터 말씀드리면 아닙니다. 계산이론적으로 말한다면, General Theorem Proving은 Uncomputable합니다.

먼저, Peano Arithmetic, $Th(\mathbb{N}, 0, 1, +, \times, >)$은 Uncomputable합니다. 실제로 더욱 강력한 정리로, diophantine equation over Peano Arithmetic is uncomputable.

4.1 Computability Theory

먼저 Uncomputable이 무슨 이야기인지 생각해봅시다. 일반적으로 어떤 함수에 대하여 이야기하는 개념인데, 우선 자연수에 대하여 생각해보겠습니다.

Definition 4.1. *A function $f : \mathbb{N} \rightarrow \mathbb{N}$ is uncomputable if there is no algorithm computing f.*

여기서는 자연수에 한하여 정의하였지만, 대부분의 Countable 집합에서는 비슷하게 정의할 수 있습니다. 예를 들면 모든 finite string $(= \Sigma^*)$이나, $\mathbb{Q}$, 심지어는

모든 알고리즘에 대해서도 정의할 수 있습니다.

여기서 알고리즘들을 domain으로 하는 함수에 대해 생각해보면, 알고리즘이라는 단어가 수학적 정의가 아니기 때문에 좀 더 제대로 된 정의가 필요합니다. 여기서 나타나는 것이 Church-Turing Thesis입니다.

Definition 4.2 (Church-Turing Thesis)**.** *A partial function $f \subset \mathbb{N} \to \mathbb{N}$ is computable if there is turing machine that computes f.*

Definition 4.3. *A Turing machine is 6-Tuple $(Q, q_0, Q_F, \Sigma, \Gamma, \delta)$ where*

- *Q is finite set of all states.*
- *q_0 is initial state, element of Q.*
- *Q_F is set of halting states, subset of Q.*
- *Σ is finite set of alphabets allowed for input.*
- *Γ is finite set of alphabets allowed to written on tape. Contains Σ.*
- *$\delta : Q \setminus Q_F \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$ is transistion function.*

Then run of turing machine is finite sequence of 'snapshot' of turing machine, which is state: q^i, tape: $tape^i$, and position of head on tape: h^i (for $i = 0 \dots n$), where $q^0 = q_0, tape^i = x, h^0 = 0$ and for every n $\delta(q^i, tape^i[h^i]) = (q^{i+1}, tape^{i+1}[h^i], N)$, where $h^i = h^{i+1}$ if N is S, $h^i = h^{i+1} - 1$ if N is R, $h^i = h^{i+1} + 1$ if N is L.
Then a run is accepted if $q^n \in Q_F$.

즉, 계산이론에서는 알고리즘의 정의로 튜링 머신을 사용합니다. 실제로 양자 컴퓨터를 포함한 현재의 컴퓨터들은 전부 튜링 머신으로 시뮬레이션 가능하다는 것이 증명되어있습니다.

Theorem 4.1. *Halting Problem is problem that given turing machine and its input, deciding whether the machine halts on input or not.*
Halting Problem is uncomputable.

Theorem 4.2. *Post Correspondence Problem is problem that decides whether given collection of pair of string $(w_1, v_1), \dots, (w_n, v_n)$ whether there exists indices $i_1, \dots, i_k$ where $w_{i_1} \dots w_{i_k} = v_{i_1} \dots v_{i_k}$.*
Post Correspondence Problem is uncomputable.

Theorem 4.3. *A formula for given signature is said to be valid if that formula is true for every structure of given signature.*
Validity for General First Order Logic is undecidable.

Corollary 4.1. *Satisfiability for General First Order Logic is undecidable. Even stronger, it is not semidecidable.*

Proof. Formula F is valid if and only if $\neg F$ is unsatisfiable. So it is equivalent that validity is undecidable and satisfiability is decidable. Unsatisfiability for general first order logic is semidecidable. So if satisfiability is semidecidable, satisfiability is decidable. □