

이 강의는 **오토마타¹⁾이론**과 **언어²⁾이론**을 다룬다. **오토마타이론**은 **컴퓨터이론** 혹은 **계산이론** (computability)으로 발전하고, **언어이론**은 **문제(problem)**와 **집합론(set theory)**으로 발전한다. 오토마타 이론에서 Finite State Automata(FA)³⁾, Pushdown Automata(PDA)⁴⁾, Turing Machine(TM)⁵⁾ 등 다양한 **계급**의 오토마타와 문법(grammar)을 소개한다. 오토마타이론과 언어이론은 **같은 것**으로 이 두 가지 이론의 관계를 공부하고, **계산 가능하다**는 것 (computable)과 **계산 불가능하다**는 것(incomputable)을(Turing-Church's Thesis) 공부하고 끝으로 NP-complete 문제를 간단히 소개한다.

1.1.1 언어이론과 오토마타이론의 발전사

1900년대 초반에 논리 또는 수학으로 표현할 수 없는 명제(또는 집합; 또는 문제)가 있는가에 관한 논의가 활발하였다. 이 논의에 시작은 2000여 년 전 그리스 수학자 **피타고라스**가 **무리수** $\sqrt{2}$ 를 발견하면서 시작되었는데, 1890년대 Cantor가 자연수 부분집합의 수가 **셀 수 없이 무한(uncountable infinite)**함을 증명하고, 1900년대 초 Russel이 **정의할 수 없는 집합(Russel's paradox)**을 보였고, 1931년 Gödel에 불확정성 정리(Incompleteness Theorem)가 알려지며 논의에 핵심이 분명하여졌다.

논리로 **정의할 수 있는** 문제와 **정의할 수 없는** 문제의 구분은 **computer(또는 program)**가 할 수 있는 일과 할 수 없는 일을 구분하는 것과 같은 것으로 받아들여져서 program이 수학(논리)이 같다는 주장의 근거가 되기도 하고, 거꾸로 이 구분이 **computer의 정의**(Turing Machine; Chap. 8)로 받아들여진다. 교과서에서는 컴퓨터 혹은 수학이 할 수 있는 일과 할 수 없는 일을, recursively enumerable⁶⁾(**RE**; computable, programmable)과 non-recursively enumerable(**non-RE**; non-computable, non-programmable)이라는 용어로 구분한다. 이에 관한 자세한 논의는 강의 후반부(Chap. 8)에서 다룬다.

Non-RE 문제들에 증명은 모두 자신에 대한 부정에 기초하고 있다. Cantor에 diagonal argument(1891), Russel에 paradox(1901), Hilbert 프로그램이 존재하지 않는다는 사실, Gödel에 불확정성 정리, Halting 프로그램이 없다는 사실은 모두 **같은 문제**이고 **같은 증명**이다.

한편 1940-50년대에는 **finite state automata**라는 간단한 기계가 나타난다. Finite state automata(FA)는 **regular language(RL)**, **regular expression(RE)**, **regular grammar(RG)**와 모두 동급⁷⁾이고, 이는 강의 전반부(Chap. 2, 3, 4)에서 다룬다. 특히 **한글 모아쓰기** 문제는 **deterministic** finite state automata로 해결되므로, 프로그램 프로젝트 1-1에서 실제로 프로그램해 볼 것이다.

-
- 1) 오토마타(automata, an automaton)는 기계 혹은 자동기계로 번역된다.
 - 2) 언어(Language)는 문자열(strung)의 집합으로 정의한다.
 - 3) Chomsky의 오토마타, 문법, 언어의 계급(hierarchy) 중 type 3.
 - 4) Chomsky의 오토마타, 문법, 언어의 계급(hierarchy) 중 type 2.
 - 5) Chomsky의 오토마타, 문법, 언어의 계급(hierarchy) 중 type 0.
 - 6) "Recursively enumerable"을 "무한하게 자연수로 열거한다."라고 직역할 수 있다.
 - 7) 오토마타(FA), 언어(RL), 식(RE), 문법(RG)이 서로 변환 가능하므로 모두 같은 계급(hierarchy)이고, 이 계급을 Chomsky는 가장 낮은 계급인 type 3로 정의하였다.

또 1950년대 말 N. Chomsky는 **grammar**(**문법**; context-free grammar)에 관한 논의를 시작하고 이는 finite automata에 **stack**이라는 **간단한** 저장소(memory)가 추가된 pushdown automata로 정의하였다. Chomsky의 생각은 언어학과 전산학 모두에 큰 영향을 주었다. 이 부분은 강의 중반부(Chap. 5, 6, 7)에서 다루어지며, 특히 context-free grammar의 **deterministic** parsing은 프로그래밍 언어 컴파일러의 구문분석(syntax analysis)에서 이용되므로 실용적으로 매우 중요⁸⁾하다. 프로그램 프로젝트 1-2에서 이를 실제로 프로그램해 볼 것이다.

RE(recursively enumerable) 문제는 컴퓨터 프로그램이 **끝나는**(**terminate**; 혹은 **recursive, decidable**) 경우와 **끝나지 않는**(**non-terminate** 혹은 **undecidable; RE but not recursive**) 경우도 나눈다. 우리는 끝나는(recursive) 문제에 관심(**algorithm**)이 많으며, 끝나는 문제를 **빠르게 풀 수 있는**(**tractable; polynomial**) 경우와 **빠르게 풀 수 없는**(**intractable; exponential**) 경우로 1969년 Cook이 구분하였으며, 빠르게 풀기 어려운 **NP** 중에 **가장 어려운 문제를 NP-complete**라는 용어로 **가정**하며 강의의 제일 마지막 부분에서 다루어진다.

이 모든 문제는, non-RE, RE, recursive, context-free, regular의 다섯 개 언어 계급으로 구분된다. 상위언어 계층이 하위언어 계층을 적절히 포함(proper inclusion)하는 **전형적인 계층구조**를 가지며, non-RE를 제외하고 위에서부터 차례로 RE는 type 0으로 recursive는 type 1로, context-free는 type 2로 regular는 type 3으로 Chomsky가 불렀으며 이를 **Chomsky's languages hierarchy**라고 부른다.

문법	언어	automata
non-type 0(type -1)	Non Recursively Enumerable No grammar	No automata Non-computable non-programmable
type 0 (unrestricted) grammar	Recursively Enumerable Computable, Programmable	Turing Machine (FA + memory (tape))
type 1 context-sensitive	Recursive, decidable(algorithm) ???	
type 1.H NP-complete	intractable(Exponential?)	
type 1.L	tractable(Polynomial)	
type 2 context-free	context-free	Pushdown Automata (FA + stack)
type 3 regular regular expression	regular	Finite Automata (FA + no memory)

8) Top-down parsing 방법인 LL과 Bottom-up 방법인 LR로 분류되며, LR의 변형인 LALR이 널리 사용된다(yacc).

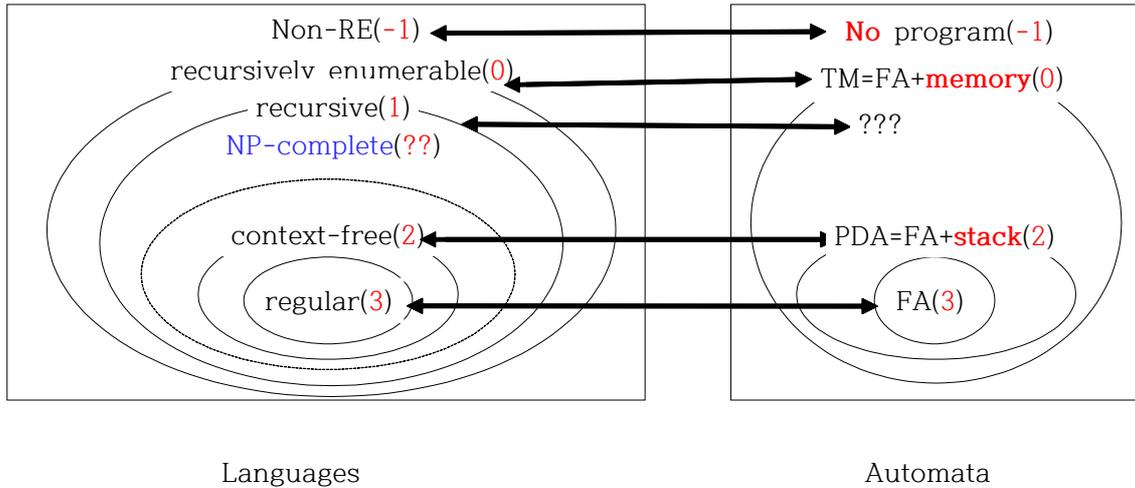


표 1.1 Chomsky's hierarchy

1.1.2 강의순서

강의는 역사에 흐름과는 다르게, 쉬운 것부터 어려운 순서로 강의할 것이다. (I)강의에 처음 2~3주는 오토마타와 언어이론을 다루기 위하여 수학 기초를 소개할 것이다. 여기서는 집합, 관계(relation), 그래프(graph), 함수(function), 집합 크기(cardinality of set), 집합에 같음 (set isomorphism), 유한집합과 무한집합, 셀 수 있는 무한(countably infinite)과 셀 수 없는 무한(uncountably infinite)과 Cantor's diagonal argument 또는 Russel's paradox를 다룬다. 2학년 때 배운 이산수학(Discrete Math.) 중 오토마타이론과 관련된 복습이다. 끝으로 언어이론에 쓰이는 **4개에 전문용어(terminology)**인 기본문자(vocabulary), 문자(symbol), 문자열(string), 언어(language)를 소개할 것이다.

(II)강의 두 번째로 언어(language) 계층 중 가장 낮은 계급인 type 3 정규언어(regular language)를 4~5주에 걸쳐 다룬다. Deterministic finite state automata(DFA)로 정규언어를 정의하고, 여러 방법으로 비결정성(non-deterministic)을 허용하는 다양한 형태⁹⁾에 Finite automata(FA)들을 차례로 소개하며, 정규언어의 또 다른 표현 방식인 정규식(regular expression, RE)을 소개하고, FA들과 RE가 모두 같은(equivalent) 정규언어를 표현하는 방법임을 보인다. 언어가 non-regular임을 증명하는 pumping lemma로 context-free(III)를 준비하고, 마지막으로 상태(state)가 가장 작은 minimal state DFA(m-DFA)를 소개하고, 프로그램 프로젝트 1-1에서 한글 모아쓰기 오토마타(mDFA)를 프로그램과 표(table)로 **쉽게** 구현해 본다.

(III)세 번째로 context-free 문법(CFG)으로 context-free 언어(CFL)를 정의한다. 쓸모 있는 (useful) 문법만 가려내는 법을 배우고, 이 과정에서 loop invariance와 loop termination이라는 개념을 소개한다¹⁰⁾. CFL을 위한 pushdown 오토마타(PDA)를 정의하고, CFL을 위한

9) (1) DFA, (2) 부분함수를 허용하는 DFA, (3) Non deterministic FA(NFA), (4) ε-move를 허용하는 ε-NFA, (5) 확장된 FA(XFA; eXtended FA)의 5가지이다.

pumping lemma와 Chomsky-Normal Form(CNF)을 소개하며, 임의의 문자열이 주어진 언어에 속하는지 하는 문제를¹¹⁾ (1) 일반적이고 deterministic 하지만 **많이** 느린($O(n^3)$) CYK(Coke, Younger, Kasami) 알고리즘과 (2) CFG를 위한 non-deterministic 파서(parser 혹은 PDA)의 두 가지 대표적인 방법인 좌/우 파서(Left/Right parser)를 소개하고 좌/우 파서의 deterministic version으로 **빠른**($O(n)$) LL과 LR 파서를 간단히 소개한다¹²⁾. 끝으로 II에서 배운 정규식을 mDFA로 고치는 알고리즘을 **도구**(tool: lex와 yacc¹³⁾)를 이용하여 프로그램 프로젝트 1-2에서 **쉽게** 프로그램해본다.

(IV)네 번째로 Turing machine(TM)과 TM의 다양한 변형, 컴퓨터를 소개하고, TM 혹은 프로그램으로 해결할 수 있고(RE), 없는 문제(non-RE)를 공부하고, Church에 생각인 primitive recursive 함수와 μ (mu)-recursive 부분함수를 소개¹⁴⁾하며 Turing과 Church에 생각이 같다는 **Turing-Church's Thesis**에 증명까지를 3주 만에 마치고, (V)마지막 주에는 NP-complete를 소개하는 Cook's Theorem을 증명하며 강의를 마친다.

1.1.3 강의 일정

I. Mathematical Preview			
Chap. 1 and handout ¹⁵⁾			2주
II. DFA와 FA의 확장, Regular expression, Pumping Lemma for RL, mDFA			
Chap. 2, 3, 4 and handout	type 3		4주
III. CFG, Pushdown Automata, Left/Right Parser , and CNF과 CYK 알고리즘			
Chap. 5, 6, 7 and handout	type 2		4주
IV. Turing Machine, non-RE and Church's thesis			
Chap. 8, 9 and handout	type 0		3주
V. NP-complete			
Chap. 10	type 1		1주

뱀 다리

강의성적은 숙제 및 프로젝트 25%, 중간고사 25%, 기말고사 25%, 출석 25%로 관리됩니다. 출석은 엄밀히 확인되며 3번까지 결석은 불이익을 안 받고, 4번부터는 25% 범위 안에서 점수에 불이익을 받게 됩니다. 자신의 출, 결석은 자신이 관리하시기 바랍니다.

학생(學生)은 **모르기** 때문에 **배우는** 사람입니다. **몰라서** 배우는 학생이 모르는 것을 **질문하는** 것은 학생의 신성한 **권리**인 동시에 심지어는 **의무**이기까지 합니다. 강의시간 중에 어떤 **질문**도 적극 환영합니다. 질문이 많으면 여러분 전체 학점이 A+ 쪽으로 올라가고, 질문이 거의 없으면 학점이 F 쪽으로 내려갈 것입니다¹⁶⁾.

10) 교과서에는 없으나 매우 중요한 문제이므로 handout을 참조하십시오.

11) Membership problem이라고 한다.

12) LL은 left-recursive한 문법은 불가능하지만, LR은 대부분에 모호하지 않은(unambiguous) 문법을 $O(n)$ 에 파싱할 수 있다.

13) 교과서에는 없으므로 UNIX manual을 참조하십시오.

14) 교과서에는 없으므로 handout을 참조하십시오.

15) **handout**은 현 교과서에는 없으므로 다른 책이나 논문에서 가져온 강의 자료입니다.

(A) 이 강의노트의 내용을 다 이해하는 학생은 이 강의를 들을 필요가 없습니다. 그는 모르지 않기 때문입니다. (B) 그러나 강의 전에는 몰랐지만, 해당 부분의 강의를 끝난 뒤에 이 강의노트 앞부분의 내용을 이해한다면, 그 학생은 이 과목에서 좋은 성적을 받을 것입니다. 특히 이 강의노트는 시험 전에 꼭 읽어보시기 바랍니다. 좋은 정리와 시험 문제를 예상할 수 있을 것입니다. (C) 그러나 강의 뒤에도 이해하지 못한다면, 이 강의에서 배운 것이 없다는 뜻입니다. 그것은 교사인 저의 잘못이 큼니다. 제가 나쁜 교사로 남지 않게 도와주시기 바랍니다. 강의 중에 계속 **질문**¹⁷⁾하여 저의 강의의 부족한 점을 저에게 알려주시고, 여러분의 이해도 확인하시기 바랍니다. 절대로 **모르는 것을** 부끄러워하지 말고, **모르는데 그냥 가만 앉아있는**¹⁸⁾ 자신을 부끄러워하시기 바랍니다.

공부는 이름 짓는 일이다.

(學而之銘名)

공자(孔子) 논어, 학이(學而) 편 의 패러디

이름을 지으면, 그 이름은 이미 본래 이름이 아니다.

(名可名, 非常名)

노자(老子) 도덕경

16) 이 강의는 상대평가가 아니고 절대평가입니다.

17) 학생이 모르면 학생의 잘못보다는 교사의 잘못이 더 큼니다. 교사가 잘못 가르쳤다는 뜻이죠.

18) 가장 나쁜 학생은 모르면서도 아는 척하는 사기꾼 같은^^; 학생입니다.