LANGUAGE AND AUTOMATA THEORY AND APPLICATIONS

Carlos Martín-Vide

Characterization

- It deals with the description of properties of sequences of symbols
- Such an abstract characterization explains the interdisciplinary flavour of the field
- The theory grew with the need of formalizing and describing the processes linked with the use of computers and communication devices, but its origins are within mathematical logic and linguistics

A bit of history

- Early roots in the work of logicians at the beginning of the XXth century: Emil Post, Alonzo Church, Alan Turing
  - Developments motivated by the search for the foundations of the notion of proof in mathematics (Hilbert)
- After the II World War: Claude Shannon, Stephen Kleene, John von Neumann
  - Development of computers and telecommunications
  - Interest in exploring the functions of the human brain
- Late 50s XXth century: Noam Chomsky
  - Formal methods to describe natural languages
- Last decades
  - Molecular biology considers the sequences of molecules formed by genomes as sequences of symbols on the alphabet of basic elements
  - Interest in describing properties like repetitions of occurrences or similarity between sequences

Chomsky hierarchy of languages

- Finite-state or regular
- Context-free
- Context-sensitive
- Recursively enumerable

$$\text{REG} \subset \text{CF} \subset \text{CS} \subset \text{RE}$$

Finite automata: origins

- Warren McCulloch & Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115-133, 1943

- Stephen C. Kleene. Representation of events in nerve nets and finite automata. In C.E. Shannon & J. McCarthy, eds., *Automata Studies*: 3-42. Princeton University Press, 1956

Kleene’s theorem

- The simplest model of computation: a discrete control + a finite memory
- Equivalence between the model of finite automata and the description of sequences of symbols using the three logical primitives
  - set union
  - set product
  - iteration
- The expressions constructed this way are called **rational expressions** and a language described by a rational expression is called a **rational language**
- **Kleene's theorem**: A language is rational iff it can be recognized by a finite automaton

Circuits

- The early papers on finite automata also have a link with the theory of circuits
- A sequential circuit, in which the output depends on an input signal, is appropriately modeled by a finite automaton
- Example: Figure represents a finite automaton. It has two states called 1 and 2. State 1 is initial and both states 1 and 2 are final. Its edges are labeled by the symbols a and b.
- According to Kleene's theorem, this set can also be described by the rational expression $(ab + b)^*(\lambda + a)$, where λ denotes the empty word, + the union and * the iteration

Star-height

- The **star-height** of a rational language X is the minimal number (over all possible regular expressions describing X) of nested stars in the expression
- Example: The star-height is 0 if and only if the language is finite (i.e. there is no star at all in the expression)
- Example: The expressions (a*b)*a* and (a+b)* both describe the set of all words on a,b. The first one has star-height 2 but the second one only 1. Therefore, the star-height of the language is 1
- The problem of computing the star-height of a given rational language was raised since the beginnings of automata theory and was solved by Kosaburo Hashiguchi. Algorithms for determining relative star heigth and star height. *Information and Computation*, 78:124-169, 1987. The star-height is recursively computable
- Open problem: What is the minimal value of the star-height of extended rational expressions, namely those allowing the additional use of the complementation?

Krohn-Rhodes theorem

- Two finite automata may be composed to form a single one
- Kenneth Krohn & John L. Rhodes. Algebraic theory of machines, I: Prime decomposition theorem for finite semigroups and machines. *Transactions of the American Mathematical Society*, 116:450-464, 1965
- Any finite automaton can be obtained by a composition of automata of two sorts:
  - **group automata**, in which the actions of the symbols on the states are one-to-one
  - **reset automata**, in which the automaton just keeps the memory of the last symbol read
- The result applies also to finite semigroups and gives an algebraic decomposition theorem for semigroups
- <u>Open problem</u>: The computation of the complexity of a finite semigroup as the minimal number of groups appearing in a decomposition

Syntactic semigroup

- It was soon recognized that finite automata are closely linked with finite semigroups, thus giving an algebraic counterpart of the definition of recognizability by finite automata
- One may characterize the rational languages as those which are recognized by a morphism on a finite semigroup, i.e. of the form $X = \varphi^{-1}(Y)$ for some morphism $\varphi : A^* \to S$ on a finite semigroup S and $Y \subset S$
- There is also a minimal possible semigroup S for each X, called the **syntactic semigroup** of X

Schützenberger theorem

- A **star-free language** is one that can be obtained with a restricted use of the rational operations, namely without the star but allowing all Boolean operations including complement
- Example: The set of all strings with alternating 0's and 1's is a star-free language since it can be written as the complement of the set of strings with some block 00 or 11
- A finite semigroup S is called **aperiodic** if there is an integer $n \geq 1$ such that for any $x \in S$, one has $x^{n+1} = x^n$
- Theorem: A rational language is star-free iff it can be recognized by an aperiodic semigroup

Varieties of rational languages

- Samuel Eilenberg. *Automata, Languages and Machines*, vols. A-B. Academic Press, 1974-1976
- A **variety** of semigroups is a family of finite semigroups closed by morphism, direct product of two elements and by taking subsemigroups
- A main example is the variety of aperiodic semigroups
- Theorem: There is a correspondence between varieties of semigroups and families of rational languages, also called varieties of languages: the semigroups are the syntactic semigroups of the corresponding languages
- Example: The variety of aperiodic semigroups corresponds to the variety of star-free languages

Locally testable languages

- A language X is **locally testable** if there is an integer k such that the property $w \in X$ only depends on the set of blocks of length k in w
- A semigroup is **idempotent** and **commutative** if $x = x^2$ and $xy = yx$, respectively
- McNaughton & Brzozowski theorem: A language is locally testable iff its syntactic monoid is locally idempotent and commutative

Finite automata and logic

- To use finite automata in the context of mathematical logic was the idea of Richard Büchi
- It was known since the work of Gödel in the 1930s that the logical theory of integers with the operations + and × is undecidable
- This opened the search for decidable subtheories
- Mojzesz Presburger proved that the theory of integers with + as the unique operation is decidable
- Büchi proved in the 1960s that the monadic second order theory of the integers with the successor is decidable: reduction to finite automata through considering a set of integers as a binary infinite sequence (example: the set of even integers corresponds to the sequence 1010101...)

Büchi's work

- Extension of the theory of finite automata to infinite words
- Interconnections between automata theory and mathematical logic
- Examples:
  - Rational languages are exactly those which can be defined in a logical language allowing to compare positions of letters in words and use quantifiers over sets of positions (monadic second order theory of the integers with $<$)
  - One of the original motivations to study star-free languages was the fact observed by McNaughton that they correspond to the first-order part (i.e. without set variables) of the above theory

Infinite words

- The work of Büchi has produced a theory of languages, called **ω-languages**, where the elements are infinite words instead of finite ones
- All notions in the finite case translate to the case of infinite words, where the proofs are substantially more difficult
- Examples:
  - The basic facts concerning rational languages like their closure under all Boolean operations become, in the infinite case, delicate results (Büchi)
  - The basic determinization algorithm of finite automata becomes, in the infinite case, a difficult theorem (McNaughton)

Automata on trees

- Automata on trees and possibly infinite trees can also be defined
- Main idea: processing a tree top-down consists in duplicating copies of the automaton at each son of the current vertex
- The state reached at the leaves (or the infinitely repeated states if the tree is infinite) determines whether or not the automaton accepts

Other theoretical connections of finite automata

- With symbolic dynamical systems
- With code theory
- With computational group theory
- With the theory of automatic groups
- With hyperbolic geometry
- ...

Origins of context-free grammars

- Main references:
  - Emil L. Post. Finite combinatory processes-Formulation 1. *Journal of Symbolic Logic*, 1:103-105, 1936
  - Alan Turing. On computable numbers with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230-265, 1936-1937
  - Zellig Harris. From morpheme to utterance. *Language*, 22:161-183, 1946
  - Noam Chomsky. Three models for the description of language. *IRE Trans.Inf.Th.*, IT-2, 1956
- They are founded on the concept of formal system (Thue, Post)
- A similar concept was developed by the early inventors of programming languages (for example, Backus for ALGOL)

Context-free languages

- A **context-free grammar** is a set of rewriting rules of the form $x \to w$ where x is a non-terminal symbol (or variable) and w is a word on the joint alphabet of terminal and non-terminal symbols
- A **derivation** consists in substituting a number of times a variable by application of a rule
- The **language** generated by the grammar is the set of words on the terminal alphabet which can be derived from an initial symbol called the axiom
- A language is context-free iff it can be generated by some context-free grammar
- Context-free languages are closed by a number of operations (including intersection with a rational language) but not under complement

Pushdown automata

- A pushdown automaton is a non-deterministic machine with a memory which may be of unbounded size but accessible only through a restricted mode called a stack
- It consists in giving access only to the last element in a first-in/last-out mode
- A word is accepted by a pushdown automaton if there is a computation which leads to empty the stack after reading the input.
- A language is context-free iff it can be accepted by some pushdown automaton

Equivalence of pushdown automata

- The equivalence of general (non deterministic) pushdown automata is an undecidable problem
- The equivalence of deterministic pushdown automata is decidable: Géraud Sénizergues. The equivalence problem for deterministic pushdown automata is decidable. In *Automata, languages and programming* (ICALP'1997), vol. 1256 of Lecture Notes in Computer Science: 671-681. Springer, 1997

Dyck language

- It is the language of well-formed expressions using n types of parenthesis
- It is generated by the grammar with rules $S \rightarrow a_n S \bar{a}_n$ for n = 1,..., n and $S \rightarrow \lambda$
- A more symmetric version also uses all rules $S \rightarrow \bar{a}_n S a_n$
- It is actually the set of words on the alphabet $A_n \cup \bar{A}_n$ equivalent to the neutral element in the free group on the set A_n

Context-free groups

- A group is context-free if it admits a presentation as $G = \langle A | R \rangle$ such that the set $L(G)$ of words on $A \cup \bar{A}$ which cancel modulo the relations from R is a context-free language
- Free groups are context-free since the Dyck language is context-free
- Finite groups are context-free since, for any presentation, the language $L(G)$ is rational
- A group is context-free iff it is an extension of a free group by a finite group

Language equations

- It is possible to give a completely algebraic definition of context-free languages based upon the idea that grammars can be seen as systems of equations
- The characterization uses the **left quotient** of a language X by a word u defined as $u^{-1}X = \{v \mid uv \in X\}$
- A language is rational iff the set of its left quotients is finite
- A family F of subsets of A is called **stable** if $u^{-1}X \in F$ for any $u \in A^*$ and $X \in F$
- Context-free languages on A are the elements of some finitely generated stable subalgebra of the algebra of subsets of A*

Computability

- The larger class containing all languages recognizable by some machine can be approached by
  - Turing machines
  - Recursive functions
- Recursive functions and Turing machines (and several other formalisms) define the same class of computable objects
- **Church thesis**: Everything computable is computable by a Turing machine
- This notion of computability does not take into account the time or space needed by a computation

Turing machines

- A Turing machine works with an infinite memory (a word on a fixed alphabet, called the tape) in which it can both read and write
- It has a finite set of states and it is said to recognize the input word w if, after starting with w on its tape, it stops in some final state
- It might never stop (like a program entering an infinite loop)

Recursively enumerable and computable

- A language L is said to be **recursively enumerable** if it can be recognized by some Turing machine M
- It is called **recursively computable** (or simply **computable**, or **decidable**) if it is recursively enumerable the same as its complement
- L is computable if it can be recognized by a Turing machine which always halts
- A typical undecidable language is the set (<M>,x) of pairs of a Turing machine (suitably coded by a word) and a word x such that M halts on x

Complexity classes

- Inside the class of recursive languages, natural subclasses appear which depend on the amount of resources needed for the recognition of a word of size n
- The limitation can be either on the time or on the space used by the Turing machine
- Important classes:
  - Class P of polynomial languages (or algorithms): limits the computation time of a deterministic Turing machine by a polynomial function
  - Class NP: same as P by allowing only non-deterministic Turing machines

Class NP

- A language is in NP if there is a search in a binary tree of polynomial height producing the solution
- A typical problem in NP is the satisfiability of Boolean formulas: $(x \vee \neg y) \wedge (\neg x \vee z) \wedge \ldots$
- For each choice of values {true, false} of the variables, it is easy to check whether the formula is true or false: therefore, the problem of finding a set of values for which the formula is valid is in NP
- It can even be shown to be NP-complete, in the sense that any problem in NP can be reduced to this one in polynomial time

P *vs* NP problem

- Suppose that solutions to a problem can be verified quickly. Then, can the solutions themselves also be computed quickly?

Classes defined by restrictions on the space used

- PSPACE is defined as the class of languages which can be recognized by a Turing machine working in space of size bounded by a polynomial in the length of the input
- NP $\subset$ PSPACE
- A typical problem in PSPACE is the satisfiability of quantified Boolean formulas: $\forall x \forall y(x \wedge \neg y \vee \exists z((x \wedge z) \vee (y \wedge z)))$

Quantum computing

- Richard Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21:467-488, 1982
- David Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. R. Soc. London A*, 400:97-117, 1985

Formal series

- Instead of considering sets of words, it is mathematically natural to consider functions from a set of words into a set of numerical values: such functions are called **formal series**
- This approach can capture several important notions such as multiplicities (when the values are integers) or probabilities (when the values are real numbers)

Rational series

- A formal series on the alphabet A is said to be **rational** if there is a morphism μ from A^* in the monoid of $n \times n$ matrices such that $(S,w) = \delta\mu(w)\gamma$ for some vectors δ,γ
- Rational languages correspond to solutions of systems of linear equations
- <u>Example</u>: The rational language $X = (ab + b)^*$ is the solution of the equation $X = (ab + b)X+1$

Algebraic series

- Context-free languages correspond to solutions of algebraic equations
- <u>Example</u>: The Dyck language generated by the grammar $D \to aDbD\lambda$ is just the solution of the equation $D = aDbD + 1$

Combinatorics on words

- Axel Thue. Über unendliche Zeichenreihen. *Norske Vid. Selsk. Skr. I Math-Nat. Kl.*, 7:1-22, 1906
- Axel Thue. Über die gegenseitige Loge gleicher Teile gewisser Zeichenreihen. *Norske Vid. Selsk. Skr. I Math-Nat. Kl. Chris.*, 1:1-67, 1912
- The most classical result is the existence of infinite square-free words, originally due to Thue

Square-free words

- A **square** in a word is a factor of the form ww
- The simplest way to obtain a square-free word is the following:
  - Start with the Thue-Morse word t = abbabaab... defined as follows:
    - Let $\beta(n)$ denote the number of 1 in the binary expansion of n. Then t_n = a if $\beta(n)$ is even and t_n = b if it is odd
  - Form the word m = abcacbabcbac..., which is the inverse image of t under the substitution a $\rightarrow$ abb, b $\rightarrow$ ab, c $\rightarrow$ a
- It can be shown that m is square-free
- The Thue-Morse word is not square-free, since it is on a binary alphabet and every long enough binary word has a square. However, it it is cube-free and even more: it does not contain an overlap, i.e. a factor of the form uvuvu with u non-empty

Sturmian words

- A **Sturmian word** is an infinite word x such that for each n, the number p(n) of distinct factors of length n appearing in x is n + 1 (it can be shown that if p(n) ≤ n, then it is actually constant and the word x is ultimately periodic)
- The simplest example of a Sturmian word is the Fibonacci word f = 01001010
- Let x_0 be 0 and x_1 be 01. Now $x_n = x_{n-1}x_{n-2}$ (the concatenation of the previous sequence and the one before)
- The rules for construction are: a → ab, b → a

Domains of application I: Compilers

- The lexical part of a compiler dealing with low-level notions such as format of the input is described by finite automata. Several software tools exist to facilitate the implementation of this part, known as **lexical analysis**
- The syntax of a programming language is often described using a context-free grammar (or an equivalent formalism). The process of checking the syntactic correctness (and computing the syntactic structure) is known as **syntax analysis**. It is performed by methods which implement a form of pushdown automaton
- The translation from the source language to the object language (a kind of low-level machine language) is a third part of the process implementing a tree traversal which can be described by attribute grammars

Domains of application II: Pattern matching

- The problems involved with text processing are relatively low-level but of everyday importance
- A domain of active research has been the study of pattern matching algorithms
- One of the most famous of these algorithms is Donald E. Knuth, James H. Morris, Jr. & Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal of Computing*, 6(2):323-350, 1977
- It allows to locate a word w in a text t in time proportional to the sum of the lengths of w and t (and not their product as in the naive algorithm looking for all possible positions of w in t at every index)
- This algorithm is actually closely linked with the computation of the minimal automaton recognizing the set of words ending with w

Domains of application III: Text compression

- A great number of algorithms have been devised to perform the compression of texts
- This is important to speed-up the transmission as well as to reduce the size of the files
- One of the most famous is the **Ziv-Lempel method** which builds a factorization of the input in blocks $x_1x_2...x_n$, where x_n is the shortest word which is not in the list $(x_1,x_2,...,x_{n-1})$

Domains of application IV: Genomes

- The progress of molecular biology, and in particular the discovery of the genetic code, has opened a field called **computational biology** dealing with biological sequences as computational objects
- Many algorithms have been applied to the analysis of biological sequences and some of them have been specifically designed for such a purpose
- One of the most famous of these algorithms is the sequence comparison based on the search of a **longest common subsequence**: a technique called dynamic programming allows to find the longest common subsequence of two sequences in time proportional to the product of the lengths of the sequences

The broad relevance of language and automata I: Mathematics

- theoretical computer science
- algebraic methods in computer science
- combinatorics on words
- computational logic
- codes
- probabilistic machines
- computability and complexity
- circuit theory
- text and image compression
- cryptography

The broad relevance of language and automata II: Language technologies

- mathematical linguistics
- parsing
- finite-state techniques
- mildly context-sensitive grammatical formalisms
- unification
- categorial logic
- mathematical foundations of natural language processing

The broad relevance of language and automata III: Artificial intelligence

- processing architectures
- parallelism
- grammar systems
- concurrency
- networks of evolutionary processors
- models of artificial life
- pattern recognition
- grammatical inference
- machine learning
- programme verification

The broad relevance of language and automata IV: Bioinformatics

- computational biology
- sequential methods in theoretical biology
- linguistics of DNA
- combinatorial algorithms for genome analysis
- mathematical evolutionary genomics
- text retrieval and pattern matching

The broad relevance of language and automata V: Nature-inspired computing

- biomolecular computing
- DNA computing
- splicing systems
- genetic algorithms
- evolutionary computing
- cellular automata
- symbolic neural networks
- quantum computing
- biomolecular nanotechnology
- unconventional computing

Anyway…

Science is like sex: it may well have practical outcomes, but this is not why we do it.

(*popular wisdom*)

Thank you !