Recognition Complexity

Eric Sven Ristad

Final version

In mathematical linguistics, grammars model linguistic theories, and the sets of strings that grammars generate model natural languages. More precisely, a language L(G) is a set of strings generated by a grammar G. The power of a grammar is the difficulty of characterizing its output. The corresponding formal problem is the *recognition problem* (RP): is a given string in a given language or not? Alternately, defining the output of a grammar to be a set of structural descriptions results in the *parsing problem*: what structural descriptions are assigned by a given grammar to a given string?

A language may be characterized in extension, by all the grammars that generate it, or constructively, by a particular grammar that generates it. In the first case, the *fixed language RP* (FLRP) is posed: given an input string x, is x in L for some fixed language L? It does not matter which grammar generates L: both grammar and language are fixed (ignored) in the problem statement. In the second case, the grammar is of interest, and the *universal RP* (URP) is posed: Given a grammar G and an input string x, is x in L(G)? Because the URP determines membership with respect to a particular grammar, it more closely models the parsing problem, which uses a grammar to assign structural descriptions.

Problems are solved by algorithms; algorithms run on machines; machines consume computational resources, such as time or space. Therefore, the complexity of a problem is given indirectly by the algorithms that solve it. Some problems cannot be solved by any algorithm: they are UNDECIDABLE (not recursive) → *Automata Theory*.

The complexity of an algorithm is the rate at which it consumes the computational resources of time and space, expressed as the order of growth of a function in the size of the problem input. Orders of growth are an upper bound on the resource requirements of an algorithm. They are useful because they abstract from many irrelevant details of the machine.

Algorithm complexity provides an upper bound on problem complexity: the most efficient known algorithm for a problem gives the tightest upper bound. Problem complexity can also be bounded from below by a reduction, according to the theory of computational complexity.

Because the complexity of a problem is a function of its input parameters, and because the URP includes the grammar in its input while the FLRP does not, the complexities of the URP and FLRP can differ. For example, the URP for generalized phrase structure grammars can require more than exponential time, while the FLRP requires less than cubic time. The parsing problem must be at least as hard as the URP, and the URP at least as hard as the FLRP, by definition.

Computational complexity theory is a mathematical theory of the intrinsic lower-bound difficulty of obtaining the solution to a problem no matter how the solution is obtained. It classifies problems according to the amount of computational resources (in our case, time or space) needed to solve them on a given abstract machine. Four important complexity classes are $\mathcal{P}$, $\mathcal{NP}$, PSPACE, and EXPPOLY.

$\mathcal{P}$ is the natural and important class of problems solvable in deterministic $\mathcal{P}$olynomial time, that is, on a deterministic Turing machine in time n^j for some integer j, where n denotes the size of the problem to be solved. $\mathcal{P}$ is considered to be the class of problems that can be solved efficiently. For example, sorting takes $n \cdot \log n$ time in the worst case using a variety of algorithms, and therefore is efficiently solvable. The URP for both context-free and regular grammars is in $\mathcal{P}$.

$\mathcal{NP}$ is the class of all problems solvable in $\mathcal{N}$ondeterministic $\mathcal{P}$olynomial time. Informally, a problem is in $\mathcal{NP}$ if one can guess an answer to the problem and then verify its correctness in polynomial time. For example, the problem of deciding whether a whole number i is composite is in $\mathcal{NP}$ because it can be solved by guessing a pair of potential divisors, each less than $\lceil \sqrt{i} \rceil$, and then quickly checking if their product equals i.

PSPACE is the class of problems solvable in deterministic polynomial space. PSPACE contains $\mathcal{NP}$ because polynomial space allows us to simulate an entire $\mathcal{NP}$ computation, but it is not known if the inclusion is proper. Intuitively, PSPACE is the class of combinatorial two-person games: it includes the problems of winning generalized versions of Checkers, Go, and Parker Brothers' Instant Insanity$^{(\mathrm{TM})}$. Many problems in formal language theory are known to be PSPACE-complete, such as finite state automaton inequivalence and intersection and the FLRP for context-sensitive languages.

NSPACE[s(n)] is the class of problems solvable in nondeterministic space that grows with s(n). In particular, the URP for Context-Sensitive Grammars is NSPACE[n] (requires linear nondeterministic space, see Kuroda 1964). Therefore the closure of CSLs under complementation follows from the theorem that NSPACE[s(n)]=co-NSPACE[s(n)] for s(n)$\geq$ log(n), see Immerman (1988), Szelepcsényi (1987) .

Finally, EXPPOLY is the class of problems solvable in deterministic time $c^{f(n)}$ for any constant c and polynomial $f(n)$ in n. This class includes PSPACE, and EXP (that is, all exponential time problems), and so includes problems that are provably intractable.

We say a problem T is $\mathcal{C}$-hard if T is at least as hard computationally as any problem in the complexity class $\mathcal{C}$. Note that T need not be in $\mathcal{C}$ to be $\mathcal{C}$-hard. A problem is $\mathcal{C}$-complete if it is both $\mathcal{C}$-hard and included in $\mathcal{C}$.

NP-complete problems can be solved only by methods too slow for even the fastest computers. Since it is widely believed, though not proved, that no faster methods of solution can ever be found for these problems, NP-complete problems are considered the easiest computationally intractable problems. The URP for many formal linguistic theories is intractable (see chart).

Complexity classifications are established with the proof technique of reduction. A reduction converts instances of a problem T of known complexity into instances of a problem S whose complexity we wish to determine. The reduction operates in polynomial time. Therefore, if we had a polynomial time algorithm for solving S, then we could also solve T in polynomial time, simply by converting instances of T into S. (This follows because the composition of two polynomial time functions is also polynomial time.) Formally, if we choose T to be NP-complete, then a polynomial time reduction from T to S shows that S is at least as hard as T, or NP-hard. If we were also to prove that S was in $\mathcal{NP}$, then S would be NP-complete.

Linguistic Model	**Complexity of URP (proof)**
Aspects transformational grammar model	Undecidable (Peters-Ritchie, 1973)
Restricted Aspects model	EXP-hard (Rounds, 1975)
Lexical-functional grammar	NP-hard (Berwick, 1982)
Generalized phrase structure grammar	EXPPOLY-hard (Ristad, 1986)
Revised generalized phrase structure grammar	NP-complete (Ristad, 1986)
Context-free grammar	P-complete (Jones-Laaser, 1974)

References Barton, Edward, Robert Berwick, and Eric Ristad. 1987. Computational Complexity and Natural Language. Cambridge, MA: MIT Press.

Berwick, Robert. 1982. Computational complexity and lexical-functional grammar. American Journal of Computational Linguistics 8(3–4):97–109.

Hopcroft, John, and Jeffrey Ullman, 1979. Introduction to Automata Theory, Languages, and Computation. Reading, MA: Addison-Wesley.

Immerman, Neil. 1988. Nondeterministic space is closed under complementation. SIAM Journal of Computing 17, 935–938

Jones, Neil, and William Laaser. 1974. Complete problems for deterministic polynomial time. Proceedings of the Sixth ACM Symposium on Theory of Computing, 40–46. Seattle: Association for Computing Machinery.

Kuroda, Sige-Yuki. 1964. Classes of languages and linear-bounded automata. Information and Control 7, 207–223.

Lewis, Harry and Christos Papadimitriou. 1978. The efficiency of algorithms. Scientific American 238:96–109.

Peters, Stanley, and R.W. Ritchie. 1973. On the generative power of transformational grammars. Information Science 6:49–83.

Rounds, William. 1975. A grammatical characterization of the exponential time languages. Proceedings of the 16th Annual Symposium on Switching Theory and Automata, 135–143. New York: IEEE Computer Society.

Szelepcsényi, Róbert. 1987. The method of forcing for nondeterministic automata. Bulletin of the European Association for Theoretical Computer Science 33, 96–100