comes up in the context of *multiprecision arithmetic*. Our computer has word size k and we wish to add and multiply numbers of length nk (n-word numbers). We take as base the number 2^k, so that $0 \leq \alpha_i, \beta_i < 2^k$, and use algorithms for finding $a + b$, $a * b$.

2.4 Parsing Expressions in Context-Free Languages

The scope of complexity theory is by no means limited to algebraic or arithmetical calculations. Let us consider *context-free grammars* of which the following is an example. The *alphabet* of G consists of the symbols t, x, y, z, (,), $+$, $*$. Of these symbols t is a *nonterminal* and all the other symbols are *terminals*. The *productions* (or rewrite rules) of G are

1. $t \to (t + t)$, 2. $t \to t * t$, 3. $t \to x$,
4. $t \to y$, 5. $t \to z$.

Starting from t, we can successively rewrite words by use of the productions. For example,

$$\bar{t} \to^1 (\bar{t} + t) \to^3 (x + \bar{t}) \to^2 (x + \bar{t} * t) \to^4 (x + y * \bar{t}) \to^5 (x + y * z). \quad (3)$$

The number above each arrow indicates the production used, and $\bar{t}$ stands for the nonterminal to be rewritten. A sequence such as (3) is called a *derivation*, and we say that $(x + y * z)$ is derivable from t. The set of all words u derivable from t and containing only terminals, is called the *language generated* by G and is denoted by $L(G)$. The above G is just an example, and the generalization to arbitrary context-free grammars is obvious.

Context-free grammars and languages commonly appear in programming languages and, of course, also in the analysis of natural languages. Two computational problems immediately come up. Given a grammar G and a word W (i.e. string of symbols) on the alphabet of G, is $W \in L(G)$? This is the *membership* problem.

The *parsing problem* is the following. Given a word $W \in L(G)$, find a derivation sequence by productions of G, similar to (3), of W from the initial symbol of G. Alternatively, we want a *parse tree*, of W. Finding a parse tree of an algebraic expression, for example, is an essential step in the compilation process.

2.5 Storing of Files

A file of records $R_1, R_2, \ldots, R_n$ is stored in either secondary or main memory. The index i of the record R_i indicates its location in memory. Each record R has a key (e.g. the social-security number in a income-tax file) $k(R)$. The computational task is to rearrange the file in memory into a sequence $R_{i_1}, \ldots, R_{i_n}$ so that the keys are in ascending order

$$k(R_{i_1}) < k(R_{i_2}) < k(R_{i_n}).$$

We emphasize both the distinction between the key and the record, which may be considerably larger than the key, and the requirement of actually rearranging the records. These features make the problem more realistic and somewhat harder than the mere sorting of numbers.

2.6 Theorem Proving by Machine

Ever since the advent of computers, trying to endow them with some genuine powers of reasoning was an understandable ambition resulting in considerable efforts being expended in this direction. In particular, attempts were made to enable the computer to carry out logical and mathematical reasoning, and this by proving theorems of pure logic or by deriving theorems of mathematical theories. We consider the important example of the theory of addition of natural numbers.

Consider the system $\mathcal{N} = \langle N, + \rangle$ consisting of the natural numbers $N = \{0, 1, \ldots\}$ and the operation $+$ of addition. The formal language L employed for discussing properties of $\mathcal{N}$ is a so-called first-order predicate language. It has variables x, y, z, ... ranging over natural numbers, the operation symbol $+$, equality $=$, the usual propositional connectives, and the quantifiers $\forall$ ("for all") and $\exists$ ("there exists").

A sentence such as $\exists x \forall y [x + y = y]$ is a formal transcription of "there exists a number x so that for all numbers y, $x + y = y$." This sentence is in fact true in $\mathcal{N}$.

The set of all sentences of L true in $\mathcal{N}$ will be called the *theory* of $\mathcal{N}$ ($Th(\mathcal{N})$) and will be denoted by $PA = Th(\mathcal{N})$. For example,

$$\forall x \forall y \exists z [x + z = y \vee y + z = x] \in PA.$$

We shall also use the name "Presburger's arithmetic," honoring Presburger, who has proved important results about $Th(\mathcal{N})$.

The *decision problem* for PA is to find an algorithm, if indeed such an algorithm exists, for determining for every given sentence F of the language L whether $F \in PA$ or not.

Presburger [12] has constructed such an algorithm for PA. Since his work, several researchers have attempted to devise efficient algorithms for this problem and to implement them by programs. These efforts were often within the framework of projects in the area of automated programming and program verification. This is because the properties of programs that one tries to establish are sometimes reducible to statements about the addition of natural numbers.

3. Central Issues and Methodology of Computational Complexity

In the previous section we listed some typical computational tasks. Later we shall present results which were obtained with respect to these problems. We shall now describe in general terms the main questions that are raised, and the central concepts that play a role in complexity theory.

3.1 Basic Concepts

A class of similar computational tasks will be called a *problem*. The individual cases of a problem P are