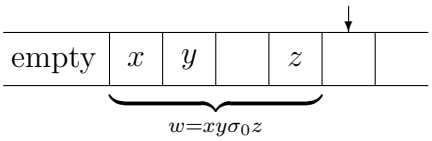
if $C = \{\sigma_1, \ldots, \sigma_k\}$ and M has set of states $\{Q_0, \ldots, Q_m\}$. Now consider a tape snapshot wQ_iav, which indicates that the head is in state Q_i, scans the symbol a (which could be the empty square, i.e. $a = \sigma_0$), and to the left of it is the word $w \in \{\sigma_0, \sigma_1, \ldots, \sigma_k\}^*$ and to the right of it the word $v \in \{\sigma_0, \sigma_1, \ldots, \sigma_k\}^*$. Notice that at each stage of the computation of the machine M there will only be finitely many non-empty squares. Here w is the word which represents from left-to-right the contents of the tape starting from the left-most non-empty square to the square just to the left of the square scanned by the head. In case every square to the left of the head is empty, $w = \theta$. Similarly for v.

Example.



We can represent this tape snapshot in the register machine N by putting w in R1, Q_ia in R2, and v in R3. Then for each entry $(Q_i, a, -, -, -)$ in the table of M we can write a set of instructions (subroutine) for N such that the effect of this entry on wQ_iav is the same as the effect of the corresponding set of instructions on the corresponding register values. In this way, we can show that the notions of TM-computability and RM-computability are equivalent.

3.4.B The Church-Turing Thesis

As mentioned earlier, many alternative models of computation $\mathcal{M}$ have been proposed over the years, and for each one we have a corresponding notion of an $\mathcal{M}$-computable function. Two common ones which we will not discuss, but which you may encounter later, are the Lambda Calculus and the theory of Recursive Functions. Another one is the theory of cellular automata, an specific example of which is Conway's Game of Life. It turns out that in all these cases, one can again prove that $\mathcal{M}$-computable = TM-computable, so all these models are, in that sense, equivalent to the Turing machine model.

It is accepted today that the informal notion of algorithmic computability is represented correctly by the precise notion of TM-computability (= RM-computability = ...), in a way similar to the assertion that the formal ϵ-δ definition of continuity of functions of a real variable represents accurately

the intuitive concept of continuity. This (extramathematical) assertion that "computability = TM-computability" is referred to as the *Church-Turing Thesis*.

More explicitly, consider a partial function $f : A^* \to B^*$ (A, B finite alphabets). We call it *intuitively computable* if there is an algorithm which for each input $w \in A^*$ terminates (in finitely many steps) exactly when w is in the domain of f and in that case produces the output $f(w)$. Then the Church-Turing Thesis is the assertion:

$$\text{intuitively computable} = \text{TM-computable}.$$

We will simply say *computable* instead of TM-computable from now on.

Remark. Any decision problem (A, P) may be identified with the (total) function $f : A^* \to \{Y, N\}^*$ given by $f(w) = Y$ if $w \in P$ and $f(w) = N$ if $w \notin P$. Thus (A, P) is decidable iff f is computable.

3.5 Universal Machines

So far we have been discussing machines which represent the operation of a single computer program. But what about a computer itself, i.e. a single machine which can execute any program? It turns out that our existing formalization is sufficient to deal with this more general case as well. After a bit of thought, this should come as no surprise, since after all the inside of a computer is just a program which sequentially reads and executes the instructions of other programs. We will now describe how to formalize this notion.

To start with, we can effectively code the words in an arbitrary finite alphabet $A = \{\sigma_1, \ldots, \sigma_k\}$ by using words in the binary alphabet $\{0, 1\}$. For example, if $k = 22$ we can use five letter words from $\{0, 1\}$ to represent each individual symbol in A, and thus every word of length n from A can be encoded as a word of length $5n$ in the binary alphabet. (We could even use the unary alphabet $\{1\}$ instead of the binary alphabet, but this would require exponential increase in the length of the word if A has more than one symbol. This does not affect computability issues, but it can certainly affect the complexity issues discussed later.)

So let's assume that we have fixed some simple method for encoding words in a given alphabet A by binary words and refer to the binary word

$b(w)$ encoding the word $w \in A^*$ as the *binary* code of w. It is now not hard to see that given a TM M on A we can construct a TM M^* on $\{0,1\}$ such that for each $w \in A^*$ the input-output behavior of M on w is exactly the same on the input-output behavior of M^* on $b(w)$.

Thus, without any loss of generality, we can restrict ourselves to discussing TMs on the binary alphabet. Any such machine M is simply given by a finite table of 5-tuples which again can be easily coded as a word in the binary alphabet. We call this the *binary code* of M and denote it by $b(M)$. By carrying out a somewhat complicated programming project, one can now construct a *universal Turing Machine*. This is a TM U on $\{0,1\}$ such that for any TM M on $\{0,1\}$ and any input $w = \{0,1\}^*$, if we put $b(M) * w$ (where $* =$ empty square) as input in U (so that in the beginning the head scans the first symbol of $b(M)$), then U terminates on this input iff M terminates on w, and in this case their outputs are the same. So we have:

Theorem 3.5.1 (Existence of universal TM) *We can construct a TM U on $\{0,1\}$ such that for any TM M on $\{0,1\}$ with binary code $b(M)$, the input-output behavior of U on $b(M) * w$ ($w \in \{0,1\}^*$) is the same as the input-output behavior of M on w.*

Remark. The universal TM was conceived theoretically by Turing in the 1930's. Any personal computer today is essentially a practical implementation of this idea.

3.6 The Halting Problem

We will now use theorem 3.5.1 to show that the so-called *halting problem* for TM is undecidable. This is precisely formulated as:

$$\begin{aligned} A &= \{0,1\} \\ P &= \{b(M) : M \text{ is a TM on } \{0,1\} \text{ and } M \\ &\qquad \text{terminates on the empty input}\}. \end{aligned} \qquad \text{(HALTING)}$$

Theorem 3.6.1 (Turing) HALTING *is undecidable.*

Proof. Consider the universal TM U. For each $x \in \{0,1\}^*$ consider the TM U_x which on the empty input first prints $x * x$, moves the head over

the first symbol of the first x, and then follows U. The table of U_x is easily obtained from the table of U by adding some further entries depending on x.

Now, if HALTING were decidable, one could easily build a TM N_0 on $\{0,1\}$ such that for any $x \in \{0,1\}^*$, N_0 on input x would terminate with output 0 if U_x terminated on the empty input, and N_0 on input x would terminate with output 1 if U_x did not terminate on the empty input. By changing N_0 a little, we could then construct a TM M_0 on $\{0,1\}$ such that if N_0 on input x terminates with output 1, so does M_0, but if N_0 on input x terminates with output 0, then M_0 does *not* terminate in that input. To simplify the notation, let us write

$$M(w) \downarrow$$

if a TM M terminates on input w and

$$M(w) \uparrow$$

if it does not. Then we have for $x \in \{0,1\}^*$

$$\begin{aligned} M_0(x) \downarrow &\Leftrightarrow U_x(\theta) \uparrow \\ &\Leftrightarrow U(x * x) \uparrow . \end{aligned}$$

In particular, for any TM M on $\{0,1\}$,

$$\begin{aligned} M_0(b(M)) \downarrow &\Leftrightarrow U(b(M) * b(M)) \uparrow \\ &\Leftrightarrow M(b(M)) \uparrow \end{aligned}$$

Putting $M = M_0$ we get a contradiction, since it cannot be the case that both $M_0(b(M_0)) \uparrow$ and $M_0(b(M_0)) \downarrow$. ⊣

Similarly one can of course prove that the corresponding halting problem for RM is undecidable. More generally, given any programming language, it is impossible to design an algorithm that will decide whether an arbitrary program will terminate on a given input or not.

3.7 Undecidability of the Validity Problem

We have succeeded in showing that one problem—the halting problem—is undecidable, but the prospects are not good for applying the method we

used to other problems, since it depends on the halting problem being a sort of "meta-problem" *about* computability. Fortunately, we can leverage this success to show the undecidability of other problems, by using the method of computable reductions, as follows.

Definition 3.7.1 Suppose (A, P) and (B, Q) are two decision problems. A (total) function $f : A^* \to B^*$ is a *(computable) reduction* of P to Q if f is computable and for any $w \in A^*$,

$$w \in P \Leftrightarrow f(w) \in Q.$$

Notice that if P is reduced to Q via f and Q is decidable, so that there is a total computable function $g : B^* \to \{Y, N\}^*$ such that $v \in Q \Rightarrow g(v) = Y$, $v \notin Q \Rightarrow g(v) = N$, then $g \circ f = h$ is computable and $w \in P \Rightarrow h(w) = Y$, $w \notin P \Rightarrow h(w) = N$. Thus P is decidable as well.

So if P can be reduced to Q and Q is decidable, then P is decidable. This observation provides a general method for showing that decision problems are *un*decidable. Specifically, to show that (B, Q) is undecidable, choose an appropriate problem (A, P) that is known to be undecidable, and find a reduction of P to Q. For then if (B, Q) were decidable, so would (A, P) be, which we know is not the case. We will apply this method to show the undecidability of the validity problem for an appropriate language L.

Theorem 3.7.2 (Church) *There is a finite language L such that the decision problem* VALIDITY$_L$ *is undecidable.*

Proof. First consider a variation of the halting problem, HALTING$_U$. This is simply the halting problem for the universal TM U. It consists of all $w \in \{0, 1\}^*$ for which $U(w) \downarrow$. Since for any TM M on $\{0, 1\}^*$, $M(\theta) \downarrow$ iff $U(b(M) * \theta) \downarrow$ iff $U(b(M)) \downarrow$, clearly HALTING$_U$ is also undecidable.

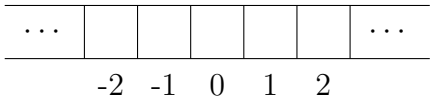
Say the states of U are $\{Q_0, Q_1, \ldots, Q_m\}$. We take L to consist of the following symbols:

$$\begin{array}{rl} 0 & : \text{constant} \\ S & : \text{unary function} \\ < & : \text{binary relation} \\ H & : \text{binary relation} \\ T_*,\ T_0,\ T_1 & : \text{binary relations} \\ R_0, \ldots, R_m & : \text{unary relations.} \end{array}$$

Intuitively, we have in mind the following interpretation for these:

(i) $0, S, <$ will be the 0, successor and order on $\langle \mathbb{Z}, 0, S, < \rangle$.

(ii) Variables will vary over $\mathbb{Z}$. They will represent both the time (i.e. stage in the computation), when restricted to ≥ 0 values (so 0 will be the beginning, 1 the next step, 2 the next one, etc.), and also the location of a square on the tape.



(iii) $H(t, x)$ will be true iff $t, x \in \mathbb{Z}$, $t \geq 0$ and the head at time t scans the square x.

(iv) $T_*(t, x)$ will be true iff $t, x \in \mathbb{Z}$, $t \geq 0$ and the tape at time t contains $*$ (i.e. is empty) at the square x, and similarly for T_0 and T_1.

(v) $R_i(t)$ will be true iff $t \in \mathbb{Z}$, $t \geq 0$, and at time t the machine is in state Q_i.

We will now assign to each $w \in \{0,1\}^*$ a sentence σ_w in L such that

$$U(w) \downarrow \quad \text{iff} \quad \sigma_w \text{ is valid.}$$

It will be clear from our construction that the map $w \mapsto \sigma_w$ is computable. Thus it will follow that HALTING_U can be reduced to VALIDITY_L, so VALIDITY_L must be undecidable.

We will first construct some preliminary sentences in L:

(i) Z is the conjunction of the following sentences:

$$\forall x(\neg x < x)$$
$$\forall x \forall y \forall z(x < y \wedge y < z \Rightarrow x < z)$$
$$\forall x \forall y(x < y \vee x = y \vee y < x)$$
$$\forall x \exists y(S(y) = x)$$
$$\forall x(x < S(x))$$
$$\forall x \forall y(x < y \Rightarrow y = S(x) \vee S(x) < y)$$

(expressing the usual properties of $\langle \mathbb{Z}, 0, S, < \rangle$).

(ii) NOCONFLICT is the conjunction of the following sentences:

$$\forall t(0 = t \vee 0 < t \Rightarrow R_0(t) \vee \cdots \vee R_m(t))$$
$$\forall t(0 = t \vee 0 < t \Rightarrow \neg R_i(t) \vee \neg R_{i'}(t)),\ 0 \leq i < i' \leq m$$
$$\forall t(0 = t \vee 0 < t \Rightarrow \exists x H(t, x))$$
$$\forall t \forall x \forall x'((0 = t \vee 0 < t) \wedge x \neq x' \Rightarrow \neg H(t, x) \vee \neg H(t, x'))$$
$$\forall t \forall x(0 = t \vee 0 < t \Rightarrow T_*(t, x) \vee T_0(t, x) \vee T_1(t, x))$$
$$\forall t \forall x(0 = t \vee 0 < t \Rightarrow (\neg T_*(t, x) \vee \neg T_0(t, x)))$$
$$\forall t \forall x(0 = t \vee 0 < t \Rightarrow (\neg T_0(t, x) \vee \neg T_1(t_1, x))$$
$$\forall t \forall x(0 = t \vee 0 < t \Rightarrow (\neg T_1(t, x) \vee \neg T_*(t, x))$$

expressing that at each time t, M is in exactly one state, and scans exactly one square, and for each time t and each square x there is exactly one symbol on that square (possibly empty).

(iii) START_w is the conjunction of the following sentences, where $w = w_0 \dots w_{n-1}$ with $w_i \in \{0, 1\}$, and we use the abbreviation

$$\bar{i} = S(S(\dots S(0)) \dots)$$

(i times) for all $0 \leq i \in \mathbb{Z}$:

$$R_0(0)$$
$$H(0, 0)$$
$$T_{w(i)}(0, \bar{i}),\ 0 \leq i < n$$
$$\forall x(\bar{n} < x \vee \bar{n} = x \Rightarrow T_*(x))$$
$$\forall x(x < 0 \Rightarrow T_*(x))$$

(expressing that the machine starts in state Q_0 with the head on the 0th square and on the input w).

(iv) NONSTOP is the sentence

$$\forall t(0 = t \vee 0 < t \Rightarrow \neg R_m(t))$$

expressing that the machine does not stop.

(v) STEP is the sentence constructed as follows:

Let (Q_i, a, b, s, Q_j) be an entry in the table of U. Say $s = +1$. (Appropriate changes have to be made in the formula below in case $s = 0$ or $s = -1$.) Assign to this entry the following sentence:

$$\forall t \forall x [0 = t \vee 0 < t \Rightarrow (R_i(t) \wedge H(t,x) \wedge T_a(t,x) \Rightarrow R_j(S(t)) \wedge H(S(t), S(x)) \wedge T_b(S(t), x))]$$

expressing the action of the machine following this entry. Then STEP is the conjunction of all these sentences for all entries in the table.

Now let ρ_w be the conjunction of all the sentences Z, NOCONFLICT START$_w$, NONSTOP, STEP, and put $\sigma_w = \neg\rho_w$. Then we have

$$U(w) \downarrow \quad \text{iff} \quad \sigma_w \text{ is valid.}$$

To see this, first notice that if $U(w) \uparrow$, then σ_w is not valid, i.e. ρ_w has a model. For we can take as such a model

$$\mathcal{M} = \langle \mathbb{Z}, 0, S, <, H^{\mathcal{M}}, T_*^{\mathcal{M}}, T_0^{\mathcal{M}}, T_1^{\mathcal{M}}, R_0^{\mathcal{M}}, \ldots, R_m^{\mathcal{M}} \rangle,$$

where $0, S, <$ have their usual meaning and $H^{\mathcal{M}}, \ldots, R_m^{\mathcal{M}}$ are interpreted as in (3)-(5) before.

Conversely, assume that $U(w) \downarrow$, say the computation on input w terminates at time $N \geq 0$. If σ_w failed to be valid (working towards a contradiction), ρ_w would have some model

$$\mathcal{A} = \langle A, 0^{\mathcal{A}}, S^{\mathcal{A}}, <^{\mathcal{A}}, H^{\mathcal{A}}, T_*^{\mathcal{A}}, T_0^{\mathcal{A}}, T_1^{\mathcal{A}}, R_0^{\mathcal{A}}, \ldots, R_m^{\mathcal{A}} \rangle.$$

Since $\mathcal{A} \models Z$, clearly A contains a copy of $\mathbb{Z}$, with $0^{\mathcal{A}}$ corresponding to 0, $S^{\mathcal{A}}(0^{\mathcal{A}})$ to 1, etc. Denote by $\mathbf{n}$ the element of A corresponding to $n \in \mathbb{Z}$. (In general, A contains many more other elements than these $\mathbf{n}$.) Then it is easy to see that for $t \in \mathbb{Z}$ with $t \geq 0$ and $x \in \mathbb{Z}$, $H^{\mathcal{A}}(\mathbf{t}, \mathbf{x})$ will be true iff the head at time t scans the square x, and similarly for $T_*^{\mathcal{A}}, \ldots, R_m^{\mathcal{A}}$. This can be proved, for example, by induction on t. Thus $R_m^{\mathcal{A}}(\mathbf{N})$ is true and this contradicts that $\mathcal{A} \models$ NONSTOP. $\dashv$

3.8 The Hilbert Tenth Problem

Using the method of reduction it has been also shown that the Hilbert 10th Problem (example 3.1.4) is undecidable. More precisely, consider the alphabet

$$A = \{x, 0, 1, \cdot, +, -, (,), \hat{}\}.$$

Encoding the variable x_n by $x(n$ written in binary), natural numbers by their binary notation, and using $\hat{}$ for exponentiation, any polynomial in several variables with integer coefficients can be encoded by a word in this alphabet.

Example 3.8.1 $x_1^2 - 7x_2^2 - 1$ will be encoded by the word

$$(x(1))\hat{}(10) + (-111)(x(10))\hat{}(10) + (-1).$$

Let now DIOPHANTINE EQUATIONS be the decision problem (A, P) where

$$P = \{w \in A^* : w \text{ encodes a polynomial (with integer coefficients in several variables) which has an integer solution}\}.$$

We now have

Theorem 3.8.2 (Matyasevich) DIOPHANTINE EQUATIONS *is undecidable.*

So this gives a negative answer to the Hilbert 10th Problem.

3.9 Decidable Problems

We will now discuss some examples of decidable problems. We will present them informally, by giving the instances together with the question defining the set of instances that constitutes the decision problem. It will be assumed that then these can be encoded in some reasonable way as formal decision problems (A, P) as in section 3.2.

Example 3.9.1 ELEMENTARY ALGEBRA is the following decision problem:

Instance. A sentence σ in the first-order language $L = \{0, 1, +, \cdot\}$.

Question. Is σ true in $\langle \mathbb{R}, 0, 1, +, \cdot \rangle$, i.e. is $\sigma \in \mathrm{Th}(\mathbb{R}, 0, 1, +, \cdot)$?

Tarski in 1949 has shown that ELEMENTARY ALGEBRA is decidable.

Remark. On the other hand, if we consider the corresponding problem ELEMENTARY ARITHMETIC for $\mathrm{Th}(\mathcal{N}, 0, S, +, \cdot, <)$, then Church has shown in the 1930's that it is undecidable (this also follows easily from 3.8.2).

Example 3.9.2 SATISFIABILITY is the following decision problem:

Instance. A finite set $U = \{u_1, \ldots, u_k\}$ of propositional variables and $C = \{c_1, \ldots, c_m\}$ a finite set of clauses $c_i = \{\ell_{i,1}, \ldots, \ell_{i,n_i}\}$, where each $\ell_{i,j}$ is a literal from U, i.e. a variable in U or its negation. (As usual we write $\bar{u}_i$ instead of $\neg u_i$ in this context.)

Question. Is C satisfiable (i.e. is there a valuation $\nu : U \to \{T, F\}$ which makes every clause c_i true, recalling that the clause c_i as above represents the disjunction $\ell_{i,1} \vee \cdots \vee \ell_{i,n_i}$)?

Using truth tables it is clear that SATISFIABILITY is decidable.

Example 3.9.3 TRAVELING SALESMAN is the following decision problem:

Instance. A finite set $\{c_1, \ldots, c_m\}$ of "cities", a distance function $d(c_i, c_j) \in \{1, 2, \ldots\}$ for $i \neq j$, and a bound $B \in \{1, 2, \ldots\}$.

Question. Is there a tour of all the cities with total length $\leq B$, i.e. an ordering $c_{\pi(1)}, c_{\pi(2)}, \ldots, c_{\pi(m)}$ of $\{c_1, \ldots, c_m\}$ such that

$$\sum_{i=1}^{m-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(m)}, c_{\pi(1)}) \leq B?$$

Again, by listing all possible orderings and calculating the above sum for each one of these, it is trivial to see that this problem is decidable.

Example 3.9.4 PRIMES is the following decision problem:

Instance. An integer $n \geq 2$.

Question. Is n prime?

This is also decidable as we can go through all the integers between 2 and $n - 1$ and check whether they divide n.

3.10 The Class $\mathcal{P}$

We have so far been discussing what problems are computable *in principle*. However, a question of more practical interest is the following: given a problem which *is* computable, can we compute it *efficiently*? The study of efficiency is also called *computational complexity*. We will concentrate here on time complexity (how long it takes to solve the problem), but one can also discuss, for example, space complexity (how much "memory" storage is used in solving it). A natural measure of time complexity for us is the number of steps in the computation of a TM that decides a given problem.

Since different problems require different amounts of input, in order to compare the complexities of different algorithms and problems, we must measure complexity as a function of the input size. In addition, with computer speeds increasing rapidly, small instances of a problem can usually be solved no matter how difficult the problem is. For this reason, and also to eliminate the effect of "overhead" time, we will consider only the *asymptotic* complexity as the size of the input increases. First we recall the "big-O notation," which gives a rigorous way of comparing asymptotic growth rates.

Definition 3.10.1 Given two functions $f, g : \mathbb{N} \to \mathbb{N}$ we define

$$f = O(g) \text{ iff } \exists n_0 \, \exists C \, \forall n \geq n_0 (f(n) \leq Cg(n)).$$

For example, if $p(n)$ is a polynomial, then $p(n) = O(n^d)$ for large enough d.

Definition 3.10.2 Given any $T : \mathbb{N} \to \mathbb{N}$, we let TIME($T$) consist of all decision problems (A, P) which can be decided by a TM in time t for some $t = O(T)$.

More precisely, a decision problem (A, P) is in TIME(T) if there is $t = O(T)$ and a TM M on some alphabet $B \supseteq A \cup \{Y, N\}$ such that for $w \in A^*$:

(i) $w \in P \Rightarrow$ on input w, M halts in $\leq t(|w|)$ steps with output Y.

(ii) $w \notin P \Rightarrow$ on input w, M halts in $\leq t(|w|)$ steps with output N

(here $|w|$ = length of the word w).

What sort of growth rate should we require of an algorithm to consider it manageable? After all, we must expect the time to increase somewhat with the input size. It turns out that the major distinctions in complexity are between "polynomial time" algorithms and faster-growing ones, such as exponentials.

Definition 3.10.3 A decision problem is in the class $\mathcal{P}$ (or it is *polynomially decidable*) if it is TIME(n^d) for some $d \geq 0$, i.e. it can be decided in polynomial time.

Problems in the class $\mathcal{P}$ are considered *tractable* (efficiently decidable) and the others *intractable*. It is clear that the class $\mathcal{P}$ provides an upper limit for problems that can be algorithmically solved in realistic terms. If a problem is in $\mathcal{P}$, however, it does not necessarily mean that an algorithm for it can be practically implemented, for example, it could have time complexity of the order of $n^{1,000,000}$ or of the order n^3 but with enormous coefficients. However, most natural problems that have been shown to be polynomially decidable have been found to have efficient (e.g., very low degree) algorithms. Moreover, the class of problems in $\mathcal{P}$ behaves well mathematically, and is independent of the model of computation, since any two formal models of computation can be mutually simulated within polynomial time.

Remark. Time complexity as explained here is a worst case analysis. If a problem P is intractable, then there is no polynomial time algorithm which for all n and all inputs of length n will decide P. But one can still search for approximate algorithms that work well on the average or for most practical (e.g., small) instances of the problem or give the correct answer with high probability, etc.

Example 3.10.4 ELEMENTARY ALGEBRA is intractable (Fisher-Rabin 1974). In fact it is in TIME($2^{2^{cn}}$) for some $c > 0$ but not in TIME($2^{2^{dn}}$) for any $d > 0$; that is, it is *super-exponential*.

Example 3.10.5 LINEAR PROGRAMMING is the decision problem given by:

Instance. An integer matrix (v_{ij}) for $1 \leq i \leq m$ and $1 \leq j \leq n$, along with integer vectors $D = (d_i)_{i=1}^m$ and $C = (c_j)_{j=1}^n$, and an integer B.

Question. Is there a rational vector $X = (x_j)_{j=1}^n$ such that $\sum_{j=1}^n v_{ij}x_j \leq d_i$, for $1 \leq i \leq m$, and $\sum_{j=1}^n c_j x_j \geq B$?

LINEAR PROGRAMMING turns out to be in $\mathcal{P}$ (Khachian, 1979).

3.11 The Class $\mathcal{NP}$ and the $\mathcal{P} = \mathcal{NP}$ Problem

Although a large class of problems have been classified as tractable or intractable, there is a vast collection of decision problems, many of them of great practical importance, that are widely assumed to be intractable but no one until now has been able to demonstrate it. These are the so-called *$\mathcal{NP}$-complete problems*. In order to introduce these problems, we must first define the class $\mathcal{NP}$ of *non-deterministic polynomial* decision problems.

Definition 3.11.1 Let (A, P) be a decision problem. We say that (A, P) is in the class $\mathcal{NP}$ if there is a TM M on an alphabet $B \supseteq A \cup \{Y, N\}$ and a polynomial $p(n)$ such that for any $w \in A^*$:

$$w \in P \Leftrightarrow \begin{array}{l}\exists v \in B^* \text{ such that } |v| \leq p(|w|) \text{ and on input } w * v,\ M \\ \text{stops with output } Y \text{ after at most } p(|w|) \text{ many steps.}\end{array}$$

In other words, $w \in P$ iff there is a "guess" v, of length bounded by a polynomial in the length of w, such that w together with v pass a polynomial acceptance test. (This can be also viewed as a non-deterministic polynomial time algorithm.)

Example 3.11.2 SATISFIABILITY is in $\mathcal{NP}$, since if we can guess a truth assignment, we can verify that it satisfies the given set of clauses in polynomial time.

Example 3.11.3 Similarly, TRAVELING SALESMAN is in $\mathcal{NP}$, since if we guess the order of the set of cities, we can calculate whether the length of the tour is $\leq B$ in polynomial time.

In fact a vast number of problems like these, which involve some kind of search, are in $\mathcal{NP}$.

Remark. Clearly $\mathcal{P} \subseteq \mathcal{NP} \subseteq \bigcup_d \text{TIME}(2^{n^d})$.

Problem. A famous problem in theoretical computer science is whether

$$\mathcal{P} = \mathcal{NP},$$

that is, whether any problem with an efficient nondeterministic algorithm also has an efficient deterministic algorithm. This is known, unsurprisingly, as the *$\mathcal{P} = \mathcal{NP}$ Problem*. Recently the Clay Mathematics Institute included the $\mathcal{P} = \mathcal{NP}$ Problem as one of its seven Millenium Prize Problems, offering $1,000,000 for its solution! The prevailing assumption today is that $\mathcal{P} \neq \mathcal{NP}$.

3.12 $\mathcal{NP}$-Complete Problems

We can get a better understanding of the $\mathcal{P} = \mathcal{NP}$ problem by discussing the notion of an $\mathcal{NP}$-complete problem. The $\mathcal{NP}$-complete problems form sort of a "core" of the "most difficult" problems in $\mathcal{NP}$. Recall the concept of a computable reduction (definition 3.7.1). We adapt this to the setting of complexity as follows:

Definition 3.12.1 A (total) function $f : A^* \to B^*$, where A, B are finite alphabets, is *polynomial-time computable* if there is a TM M on a finite alphabet $C \supseteq A \cup B$, and a polynomial p, such that for every input $w \in A^*$, M terminates on at most $p(|w|)$ steps with output $f(w)$.

Unsurprisingly, a *polynomial-time reduction* is a computable reduction which is in addition polynomial-time computable. We now say that

Definition 3.12.2 A decision problem (B, Q) is *$\mathcal{NP}$-complete* if it is in $\mathcal{NP}$, and for every $\mathcal{NP}$ problem (A, P) there is a polynomial-time reduction of P to Q. (That is, there is a polynomial-time computable function $f : A^* \to B^*$ such that $w \in P$ if and only if $f(w) \in Q$.)

An $\mathcal{NP}$-complete problem is in some sense a hardest possible problem in $\mathcal{NP}$. For example, it is clear that if (B, Q) is in $\mathcal{P}$ and (A, P) can be polynomial-time reduced to Q, then also (A, P) is in $\mathcal{P}$. It therefore follows that if *any* $\mathcal{NP}$-complete problem is in $\mathcal{P}$, then $\mathcal{P} = \mathcal{NP}$. Thus the $\mathcal{P} = \mathcal{NP}$ question is equivalent to the question of whether any given $\mathcal{NP}$-complete problem is in $\mathcal{P}$.

It is clear from the definition that all $\mathcal{NP}$-complete problems are "equivalent" in some sense, since each one can be reduced to the other by a polynomial-time computable function. We have not yet established however that there are *any* $\mathcal{NP}$-complete problems. This was first shown by Cook and Levin in 1971.

Theorem 3.12.3 (Cook, Levin) SATISFIABILITY *is $\mathcal{NP}$-complete.*

Karp in 1972 has shown that TRAVELING SALESMAN and many other combinatorial problems are $\mathcal{NP}$-complete, and since that time hundreds of others have been discovered in many areas of mathematics and computer science.

We will give a proof of this theorem shortly, but first let us say a few things about the complexity of PRIMES, because it occupies a special place among "difficult" problems. Given a decision problem (A, P), its *complement* is the decision problem $(A, \sim P)$, where $\sim P = A^* \setminus P$. It is clear that a problem is decidable (or in $\mathcal{P}$) iff its complement is decidable (or in $\mathcal{P}$), but it is unknown whether a problem is in $\mathcal{NP}$ iff its complement is $\mathcal{NP}$.

We denote by "co-$\mathcal{NP}$" the class of problems whose complements are in $\mathcal{NP}$. Thus $\mathcal{P} \subseteq \mathcal{NP} \cap$co-$\mathcal{NP}$, but whether $\mathcal{NP} =$ co-$\mathcal{NP}$ is unknown. Now it is easy to see that PRIMES in co-$\mathcal{NP}$. (In precisely formulating PRIMES we assume that every positive integer is represented by its binary notation. If numbers were represented in unary notation, then clearly PRIMES would be in $\mathcal{P}$.) With some work, using some elementary number theory, it can be shown that PRIMES is also in $\mathcal{NP}$. Thus

$$\text{PRIMES} \in \mathcal{NP} \cap \text{co-}\mathcal{NP}.$$

Miller in 1976 has shown that if one assumes a widely believed but still unproven hypothesis, the so-called Generalized Riemann Hypothesis, then actually PRIMES is in $\mathcal{P}$, but it is still unknown whether PRIMES is actually in $\mathcal{P}$. (*Addendum*: In 2002 Agrawal, Kayal and Saxena proved that PRIMES is indeed in $\mathcal{P}$.)

Proof of 3.12.3. The proof is a variation of that of 3.7.2.

To fix a formal encoding of the satisfiability problem, we simply view each instance as a formula in propositional logic in conjunctive normal form (cnf), and which can thus be viewed as a word in the alphabet

$$C = \{p, 0, 1, \neg, \wedge, \vee, (,)\},$$

where we write pn, with n in binary, instead of the propositional variable p_n.

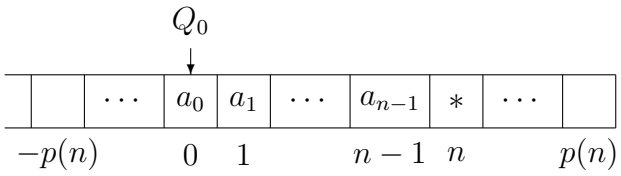
Consider now an arbitrary decision problem (A, P) which is in the class $\mathcal{NP}$. We will find a polynomial-time computable map $f : A^* \to C^*$ such that, for each $w \in A^*$, $f(w)$ is a formula in cnf and

$$w \in P \Leftrightarrow f(w) \text{ is satisfiable.}$$

By a slight reformulation of the definition of what it means to be in the class $\mathcal{NP}$, we can find an alphabet $B \supseteq A$, a polynomial $p(n) > n$ and a TM M on B with states $\{Q_0, \ldots, Q_m\}$ (Q_0 is the start and Q_m is the stop

state), such that all the entries in the table of M that begin with Q_m are of the form $(Q_m, b, b, 0, Q_m)$, and we have for any $w = a_0 a_1 \dots a_{n-1} \in A^*$:

$w \in P$ iff there is an input



in which a_i is in square i, $0 \leq i \leq n-1$, square n is empty, as are all squares $j < 0$ and $j > p(n)$ (but some symbols in B are possibly occupying some squares between $n+1$ and $p(n)$), and on which M terminates in at most $p(n)$ many steps (and thus at each step of the computation the machine only visits squares with index $-p(n) \leq j \leq p(n)$).

Let $A = \{\sigma_1, \dots, \sigma_K\}$, $B = \{\sigma_1, \dots, \sigma_K, \ \sigma_{K+1}, \dots, \sigma_L\}$, and finally let σ_0 denote the empty square.

Associate with M and each $n = 0, 1, 2, \dots$ the following list of propositional variables:

(i) $R_{i,k}$, $0 \leq i \leq p(n)$, $0 \leq k \leq m$,

(ii) $H_{i,j}$, $0 \leq i \leq p(n)$, $-p(n) \leq j \leq p(n)$,

(iii) $T_{i,j,\ell}$, $0 \leq i \leq p(n)$, $-p(n) \leq j \leq p(n)$, $0 \leq \ell \leq L$.

The intended meanings of these variables are as follows:

(i) $R_{i,k}$ being true means that at time i the machine is in state Q_k.

(ii) $H_{i,j}$ being true means that at time i the head scans square j.

(iii) $T_{i,j,\ell}$ being true means that at time i the tape contains σ_ℓ at square j.

(We can of course view these propositional variables as part of our standard list $p_0, p_1, p_2, \dots$.)

Consider now the following clauses in these variables, associated to each $w = \sigma_{k_0} \sigma_{k_1} \dots \sigma_{k_{n-1}} \in A^*$, $1 \leq k_i \leq K$:

(a)

$$\begin{array}{cl}
R_{i,0} \vee \cdots \vee R_{i,m} & \\
\neg R_{i,k} \vee \neg R_{i,k'} & (0 \leq i \leq p(n),\ 0 \leq k < k' \leq m) \\
H_{i,-p(n)} \vee \cdots \vee H_{i,p(n)} & \\
\neg H_{i,j} \vee \neg H_{i,j'} & (0 \leq i \leq p(n),\ -p(n) \leq j < j' \leq p(n)) \\
T_{i,j,0} \vee \cdots \vee T_{i,j,L} & \\
\neg T_{i,j,\ell} \vee \neg T_{i,j,\ell'} & (0 \leq i \leq p(n),\ -p(n) \leq j \leq p(n), \\
 & \ 0 \leq \ell < \ell' \leq L)
\end{array}$$

(expressing that at each time i the machine is in exactly one state and scans exactly one square and for each time i and square j there is exactly one symbol on j, possibly $*$).

(b)

$$\begin{array}{cll}
R_{0,0};\ H_{0,0};\ T_{0,j,k_j} & (0 \leq j \leq n-1) \\
T_{0,n,0} & \\
T_{0,j,0} & (j < 0)
\end{array}$$

(these express that the machine starts at time 0 at state Q_0 with the head at the 0th square and on input as in the preceding picture)

(c)

$$R_{p(n),m}$$

(this expresses that the machine terminates at time $\leq p(n)$)

(d) Let $(Q_k, \sigma_\ell, \sigma_{\ell'}, s, Q_{k'})$ be an entry in the table of M. Say $s = +1$. (Appropriate changes in the formula below have to be made if $s = 0$ or $s = -1$.) Assign to this entry the following clauses for each $0 \leq i < p(n)$, $-p(n) \leq j < p(n)$:

$$\begin{array}{c}
\neg R_{i,k} \vee \neg H_{i,j} \vee \neg T_{i,j,\ell} \vee R_{i+1,k'} \\
\neg R_{i,k} \vee \neg H_{i,j} \vee \neg T_{i,j,\ell} \vee H_{i+1,j+1} \\
\neg R_{i,k} \vee \neg H_{i,j} \vee \neg T_{i,j,\ell} \vee T_{i+1,j,\ell'}
\end{array}$$

(these express the action of the machine, when at time i it is at state Q_k and scans σ_ℓ on the jth square). Note here that $\neg R_{i,k} \vee \neg H_{i,j} \vee \neg T_{i,j,\ell} \vee R_{i+1,k'}$ is equivalent to $R_{i,k} \wedge H_{i,j} \wedge T_{i,j,\ell} \Rightarrow R_{i+1,k'}$.

Finally, let $f(w)$ be the conjunction of all the formulas (clauses) given in (a), (b), (c), (d). It is easy to check that f is a polynomial-time computable function. We verify that $w \in P \Leftrightarrow f(w)$ is satisfiable.

If $w \in P$, then we can find an input as in the previous picture with the properties stated there and we assign truth values to the variables according to their intended meaning in this computation associated with that input. This valuation clearly satisfies all the clauses of $f(w)$.

Conversely, assume a valuation ν satisfies all the clauses in $f(w)$. Because ν satisfies all the clauses in (a), (b), for each $-p(n) \leq j \leq p(n)$ there is a unique $\sigma_{k(j)}$ $(0 \leq k(j) \leq L)$ with $\nu(T_{0,j,k(j)}) = T$. Consider the input for M that has $\sigma_{k(j)}$ in the jth square. Because $k_j = k(j)$ for $0 \leq j \leq n-1$ and $k(n) = 0$, this is exactly an input as in the preceding picture. If we start M on that input then it is easy to see, since ν satisfies all the clauses in (a), (d), that for all $0 \leq i \leq p(n)$, $\nu(R_{i,k}) = T$ iff at time i in this computation the machine is in state Q_k, and similarly for $H_{i,j}$ and $T_{i,j,\ell}$. This is proved by induction on i. But then, since ν satisfies (c), we must have that the computation of M on that inputs stops in at most $p(n)$ steps, so $w \in P$. $\dashv$