in time and space $O(f(n))$, respectively. Moreover, the complexity class NTIMETM$(f(n))$ *is the set of decision problems P whose corresponding language L_P can be solved in time $O(f(n))$ by a nondeterministic Turing machine.*

According to the above definition and because the Pascal program and Turing machine models are polynomially related, we have that

$$\text{P} = \cup_{k=0}^{\infty} \text{TIMETM}(n^k),$$

$$\text{PSPACE} = \cup_{k=0}^{\infty} \text{SPACETM}(n^k),$$

and

$$\text{NP} = \cup_{k=0}^{\infty} \text{NTIMETM}(n^k).$$

Observe that, according to the above definitions, classes P, PSPACE, and NP contain decision problems. In the following, however, we will, in some cases, identify a decision problem with its corresponding language and we will view these classes as sets of languages.

Before concluding this section, we observe that Turing machines are in practice not used to design algorithms. However, defining the complexity classes using Turing machines is useful when proving hardness results. The Turing machine is such a simple computation model that it is relatively easy to express any computation on it using a simple structure like a Boolean formula. We will soon use this to show a hardness result for SATISFIABILITY.

6.1.5 NP-completeness and Cook-Levin theorem

Now that we have defined P and NP using the Turing machine computation model we will proceed to define NP-completeness in this model. Recall from Chap. 1 that a decision problem P is NP-complete if it is included in NP and if every problem in NP is polynomial-time Karp-reducible to P. The Karp-reducibility defined in Sect. 1.3 is stated using a Pascal program that transforms an instance of one problem into an instance of another problem. Instead of a polynomial-time Pascal program we can use a polynomial-time Turing machine that simulates the Pascal program in polynomial time. Observe that such a Turing machine does not behave as an acceptor machine as defined in Def. 6.2, but it must return an output value and is usually called a *transducer*: the formal definition of a transducer is left to the reader (see also the end of Sect. 6.1.1).

As stated in Chap. 1, the standard way to show the NP-completeness of a decision problem is to find a polynomial-time Karp-reduction from some

other problem that is already known to be NP-complete. Clearly, such a process needs some *initial* NP-complete problem, whose NP-completeness must be proved by using some other technique.

We now see an important result, known as *Cook-Levin theorem*, which shows that SATISFIABILITY is such an initial NP-complete problem.

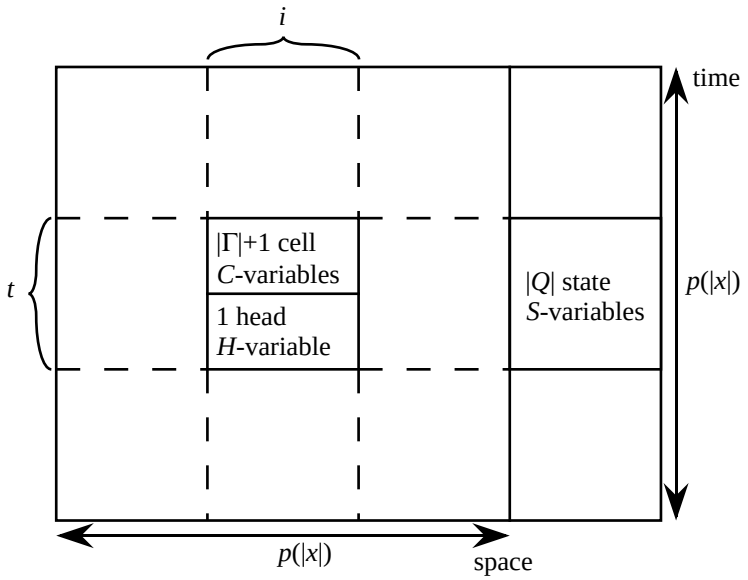

Figure 6.6
The Boolean variables of Cook-Levin theorem

SATISFIABILITY *is* NP*-complete.* ◄ Theorem 6.1

PROOF

SATISFIABILITY has already been shown to belong to NP by providing a nondeterministic polynomial-time algorithm that solves it (see Algorithm 1.3). We have then only to show that, given any decision problem $P \in$ NP, we can (Karp-) reduce P to SATISFIABILITY in polynomial time.

Let M be a nondeterministic Turing machine that recognizes L_P in time $p(n)$, for a suitable polynomial p. Given $x \in \Sigma^*$, we will construct a formula w in conjunctive normal form such that w is satisfiable if and only if M accepts x in time $p(|x|)$, that is, if and only if $x \in L$. Intuitively, w will be constructed in such a way that any satisfying truth assignment to its variables will *encode* an accepting computation path of M on input x.

In order to simplify the proof, and without loss of generality, we make the assumption on M that its tape is semi-infinite and that, for any input, there is no transition trying to overpass the leftmost tape cell: it can be shown (see Exercise 6.8) that this assumption is justified. Since M on input x runs in time $p(|x|)$, only the first $p(|x|)$ tape cells may be accessed

during the computation. Finally, we will denote with $a \Rightarrow b$ (respectively, $a \equiv b$) the Boolean formula $\overline{a} \vee b$ (respectively, $(\overline{a} \vee b) \wedge (a \vee \overline{b})$).

The formula w we are going to derive is the conjunction of several formulas, that is, $w = w_M \wedge w_I \wedge w_A \wedge w_T$, where:

1. w_M specifies the general properties of Turing machines.
2. w_I specifies that M has been given string x as input on the tape.
3. w_A specifies that M has eventually entered an accepting configuration.
4. w_T specifies the particular transition function of M.

The formula w contains the following variables with the corresponding interpretations (see Fig. 6.6):

1. $S(t,k)$ $(0 \leq t < p(|x|),\ 1 \leq k \leq |Q|)$ where Q is the set of states of M: $S(t,k)$ takes value TRUE if and only if M is in state q_k at time t (we let q_1 be the initial state q_S and $q_{|Q|}$ be the accepting state q_A).
2. $H(t,i)$ $(0 \leq t < p(|x|),\ 0 \leq i < p(|x|))$: $H(t,i)$ takes value TRUE if and only if M's head scans cell i at time t.
3. $C(t,i,h)$ $(0 \leq t < p(|x|),\ 0 \leq i < p(|x|),\ 0 \leq h \leq |\Gamma|)$ where Γ is the tape alphabet of M: $C(t,i,h)$ takes value TRUE if and only if cell i at time t contains the symbol σ_h (we let $\sigma_0 = \square$).

Let us now see how the various subformulas of w are built:

1. w_M is the conjunction of four formulas w_{MS}, w_{MH}, w_{MC}, and w_{MT} that specify the following properties, respectively:[5]
   - At any time t, M must be in exactly one state, that is, w_{MS} is the formula
   $$\bigwedge_t \left(\bigvee_k S(t,k) \wedge \bigwedge_{k_1 \neq k_2} (S(t,k_1) \Rightarrow \overline{S}(t,k_2)) \right).$$
   - At any time t, the head must be on exactly one tape cell, that is, w_{MH} is the formula
   $$\bigwedge_t \left(\bigvee_i H(t,i) \wedge \bigwedge_{i_1 \neq i_2} (H(t,i_1) \Rightarrow \overline{H}(t,i_2)) \right).$$

[5] Note that here the quantifiers $\bigwedge$ and $\bigvee$ are used just as abbreviations: for example, $\bigwedge_t F(\ldots t \ldots)$ stands for $F(\ldots 0 \ldots) \wedge F(\ldots 1 \ldots) \wedge \ldots \wedge F(\ldots p(|x|)-1 \ldots)$.

- At any time t, each cell must contain exactly one character, that is, w_{MC} is the formula

$$\bigwedge_{t,i}\left(\bigvee_{h} C(t,i,h) \wedge \bigwedge_{h\neq h'} (C(t,i,h) \Rightarrow \overline{C}(t,i,h'))\right).$$

- At any two successive times t and $t+1$, the tape contents are the same, except possibly for the cell scanned at time t. That is, w_{MT} is the formula

$$\bigwedge_{t,i,h} (\overline{H}(t,i) \Rightarrow (C(t,i,h) \equiv C(t+1,i,h))).$$

  Notice that w_{MT} is not in conjunctive normal form: however, it can be easily put in conjunctive normal form with a linear increase of its length (see Exercise 6.9).

Taking into account the ranges of the values assumed by t, i, h, and k, the overall length of w_M is $O(p^3(|x|))$.

2. If $x = \sigma_{h_0}\sigma_{h_1}\ldots\sigma_{h_{|x|-1}}$, then w_I is the formula

$$\begin{aligned} &S(0,1) \wedge H(0,0) \wedge C(0,0,h_0) \wedge C(0,1,h_1) \wedge \cdots \\ &\cdots \wedge C(0,|x|-1,h_{|x|-1}) \wedge C(0,|x|,0) \wedge \cdots \\ &\cdots \wedge C(0,p(|x|)-1,0). \end{aligned}$$

3. $w_A = S(0,|Q|) \vee S(1,|Q|) \vee \cdots \vee S(p(|x|)-1,|Q|)$.

4. w_T encodes the set of transition rules for M. It is structured as the conjunction of $p^2(|x|)|Q|(|\Gamma|+1)$ formulas $w_{t,i,k,h}$:

$$\bigwedge_{t=0}^{p(|x|)-1} \bigwedge_{i=0}^{p(|x|)-1} \bigwedge_{k=1}^{|Q|} \bigwedge_{h=0}^{|\Gamma|} w_{t,i,k,h}.$$

For any t,i,k,h, if

$$\delta(q_k,\sigma_h) = \{(q_{k_1},\sigma_{h_1},r_{k_1}),\ldots,(q_{k_d},\sigma_{h_d},r_{k_d})\},$$

then (assuming $\mu(\mathrm{R}) = 1$, $\mu(\mathrm{L}) = -1$, and $\mu(\mathrm{N}) = 0$)

$$w_{t,i,k,h} = (S(t,k) \wedge H(t,i) \wedge C(t,i,h)) \Rightarrow \bigvee_{j=1}^{d} w^j_{t,i,k,h}$$

where

$$w^j_{t,i,k,h} = H(t+1, i+\mu(r_{k_j})) \wedge C(t+1,i,h_j) \wedge S(t+1,k_j).$$

If $\delta(q_k,\sigma_h) = \emptyset$, then $w_{t,i,k,h}$ is equal to TRUE, for any t and i. The length of formula w_T can be seen to be $O(p^2(|x|))$. As in the case of w_{MT}, w_T is not in conjunctive normal form but it can be put in conjunctive normal form with a linear increase of its length.

Taking into consideration the whole formula w, it can easily be checked that its length is $O(p^3(|x|))$ and that it can be derived from x (and from M and p) in time proportional to its length. Moreover, it is not difficult to verify that the formula w is satisfiable if and only if M accepts the string x in time $p(|x|)$.

QED

We have thus shown that SATISFIABILITY is NP-complete. Proving the NP-completeness of other problems is, usually, an easier task since we can use a decision problem that is already known to be NP-complete.

Example 6.5 ▶ Let us consider E3-SATISFIABILITY which is the restriction of SATISFIABILITY to instances with exactly three literals per clause. It is clear that such a problem belongs to NP. To prove that it is NP-complete, we will define a polynomial-time reduction from SATISFIABILITY to E3-SATISFIABILITY. The reduction will transform the clauses of the instance of SATISFIABILITY into a set of "equivalent" clauses containing exactly three (different) literals. More precisely, let C_i be any clause of the instance of SATISFIABILITY. Then C_i is tranformed into the following subformula C'_i, where the y variables are new ones:

1. If $C_i = l_{i_1}$, then $C'_i = (l_{i_1} \vee y_{i,1} \vee y_{i,2}) \wedge (l_{i_1} \vee y_{i,1} \vee \overline{y_{i,2}}) \wedge (l_{i_1} \vee \overline{y_{i,1}} \vee y_{i,2}) \wedge (l_{i_1} \vee \overline{y_{i,1}} \vee \overline{y_{i,2}})$.
2. If $C_i = l_{i_1} \vee l_{i_2}$, then $C'_i = (l_{i_1} \vee l_{i_2} \vee y_i) \wedge (l_{i_1} \vee l_{i_2} \vee \overline{y_i})$.
3. If $C_i = l_{i_1} \vee l_{i_2} \vee l_{i_3}$, then $C'_i = l_{i_1} \vee l_{i_2} \vee l_{i_3}$.
4. If $C_i = l_{i_1} \vee l_{i_2} \vee \ldots \vee l_{i_k}$ with $k > 3$, then $C'_i = (l_{i_1} \vee l_{i_2} \vee y_{i,1}) \wedge (\overline{y_{i,1}} \vee l_{i_3} \vee y_{i,2}) \wedge \ldots \wedge (\overline{y_{i,k-4}} \vee l_{i_{k-2}} \vee y_{i,k-3}) \wedge (\overline{y_{i,k-3}} \vee l_{i_{k-1}} \vee l_{i_k})$.

Clearly, such a reduction can be done in polynomial time. Moreover, it is easy to prove that the original formula is satisfiable if and only if the transformed formula is satisfiable (see Exercise 6.12). That is, E3-SATISFIABILITY is NP-complete.

6.2 Oracles

ORACLES WERE mentioned already in Chap. 1: we will here define them more formally. Oracles will later be used for defining probabilistically checkable proofs, a basic tool for proving non-approximability results (see Sect. 6.3).