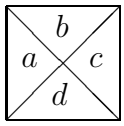
- $3SAT \in \mathbf{NP}$. We know from Theorem 6.5.7 that this is true.
- $A \leq_P 3SAT$, for *every* language $A \in \mathbf{NP}$. Hence, we have to show this for languages A such as $kColor$, HC, SOS, $NPrim$, KS, $Clique$, and for *infinitely* many other languages.

In 1971, Cook has exactly done this: He showed that the language $3SAT$ is **NP**-complete. Since his proof is rather technical, we will prove the **NP**-completeness of another language.

6.5.3 An NP-complete domino game

We are given a finite collection of *tile types*. For each such type, there are arbitrarily many tiles of this type. A *tile* is a square that is partitioned into four triangles. Each of these triangles contains a symbol that belongs to a finite alphabet Σ. Hence, a tile looks as follows:



We are also given a square *frame*, consisting of cells. Each cell has the same size as a tile, and contains a symbol of Σ.

The problem is to decide whether or not this *domino game* has a solution. That is, can we completely fill the frame with tiles such that

- for any two neighboring tiles s and s', the two triangles of s and s' that touch each other contain the same symbol, and
- each triangle that touches the frame contains the same symbol as the cell of the frame that is touched by this triangle.

There is one final restriction: The orientation of the tiles is fixed, they cannot be rotated.

Let us give a formal definition of this problem. We assume that the symbols belong to the finite alphabet $\Sigma = \{0,1\}^m$, i.e., each symbol is encoded as a bit-string of length m. Then, a tile type can be encoded as a tuple of four bit-strings, i.e., as an element of Σ^4. A frame consisting of t rows and t columns can be encoded as a string in Σ^{4t}.

We denote the language of all solvable domino games by *Domino*:

$$\begin{aligned} \mathit{Domino} \;:=\; & \{\langle m, k, t, R, T_1, \ldots, T_k\rangle : \\ & m \geq 1, k \geq 1, t \geq 1, R \in \Sigma^{4t}, T_i \in \Sigma^4, 1 \leq i \leq k, \\ & \text{frame } R \text{ can be filled using tiles of types} \\ & T_1, \ldots, T_k.\} \end{aligned}$$

We will prove the following theorem.

Theorem 6.5.12 *The language Domino is* **NP***-complete.*

Proof. It is clear that $\mathit{Domino} \in \mathbf{NP}$: A solution consists of a $t \times t$ matrix, in which the (i, j)-entry indicates the type of the tile that occupies position (i, j) in the frame. The number of bits needed to specify such a solution is polynomial in the length of the input. Moreover, we can verify in polynomial time whether or not any given "solution" is correct.

It remains to show that

$$A \leq_P \mathit{Domino}, \text{ for every language } A \text{ in } \mathbf{NP}.$$

Let A be an arbitrary language in **NP**. Then there exist a polynomial p and a non-deterministic Turing machine M, that decides the language A in time p. We may assume that this Turing machine has only one tape.

On input $w = a_1 a_2 \ldots a_n$, the Turing machine M starts in the start state z_0, with its tape head on the cell containing the symbol a_1. We may assume that during the entire computation, the tape head never moves to the left of this initial cell. Hence, the entire computation "takes place" in and to the right of the initial cell. We know that

$$\begin{aligned} w \in A \quad \Longleftrightarrow \quad & \text{on input } w, \text{ there exists an accepting computation} \\ & \text{that makes at most } p(n) \text{ computation steps.} \end{aligned}$$

At the end of such an accepting computation, the tape only contains the symbol 1, which we may assume to be in the initial cell, and M is in the final state z_1. In this case, we may assume that the accepting computation makes exactly $p(n)$ computation steps. (If this is not the case, then we extend the computation using the instruction $z_1 1 \to z_1 1 N$.)

We need one more technical detail: We may assume that $za \to z'bR$ and $za' \to z''b'L$ are not both instructions of M. Hence, the state of the Turing machine uniquely determines the direction in which the tape head moves.

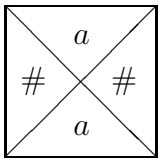
We have to define a domino game, that depends on the input string w and the Turing machine M, such that

$$w \in A \Longleftrightarrow \text{this domino game is solvable.}$$

The idea is to encode an accepting computation of the Turing machine M as a solution of the domino game. In order to do this, we use a frame in which each row corresponds to one computation step. This frame consists of $p(n)$ rows. Since an accepting computation makes exactly $p(n)$ computation steps, and since the tape head never moves to the left of the initial cell, this tape head can visit only $p(n)$ cells. Therefore, our frame will have $p(n)$ columns.

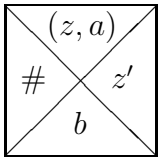
The domino game will use the following tile types:

1. For each symbol a in the alphabet of the Turing machine M:



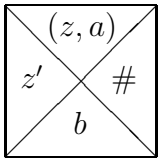
   Intuition: Before and after the computation step, the tape head is not on this cell.

2. For each instruction $za \to z'bR$ of the Turing machine M:



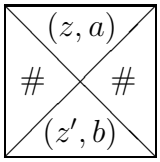
   Intuition: Before the computation step, the tape head is on this cell; the tape head makes one step to the right.

3. For each instruction $za \to z'bL$ of the Turing machine M:



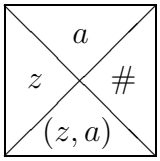
Intuition: Before the computation step, the tape head is on this cell; the tape head makes one step to the left.

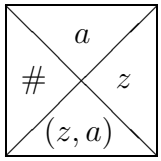
4. For each instruction $za \to z'bN$ of the Turing machine M:



Intuition: Before and after the computation step, the tape head is on this cell.

5. For each state z and for each symbol a in the alphabet of the Turing machine M, there are two tile types:





Intuition: The leftmost tile indicates that the tape head enters this cell from the left; the righmost tile indicates that the tape head enters this cell from the right.

This specifies all tile types. The $p(n) \times p(n)$ frame is given in Figure 6.5. The top row corresponds to the start of the computation, whereas the bottom row corresponds to the end of the computation. The left and right columns correspond to the part of the tape in which the tape head can move.

The encodings of these tile types and the frame can be computed in polynomial time.

It can be shown that, for any input string w, any accepting computation of length $p(n)$ of the Turing machine M can be encoded as a solution of this domino game. Conversely, any solution of this domino game can be "translated" to an accepting computation of length $p(n)$ of M, on input string w. Hence, the following holds.

$$\begin{aligned} w \in A \iff & \text{ there exists an accepting computation that makes} \\ & p(n) \text{ computation steps} \\ \iff & \text{ the domino game is solvable.} \end{aligned}$$

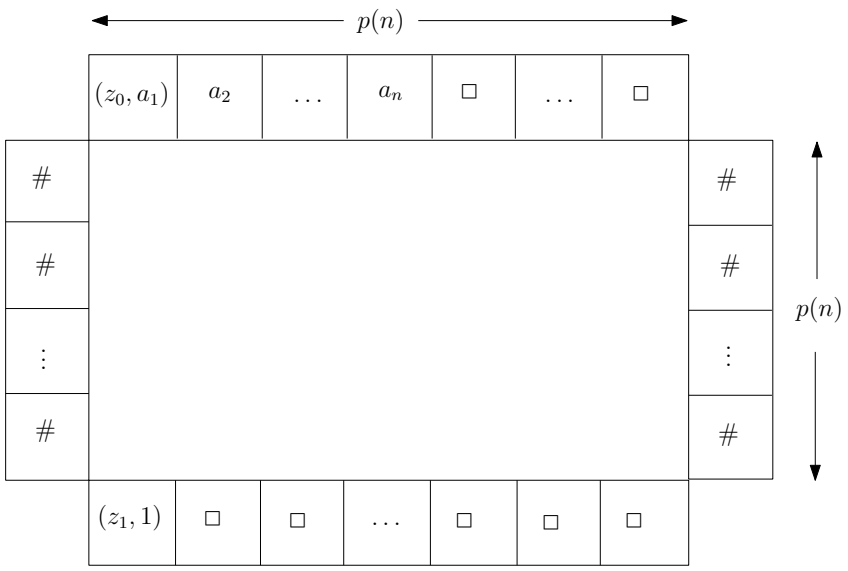


Figure 6.5: *The $p(n) \times p(n)$ frame for the domino game.*

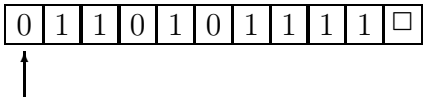
Therefore, we have $A \leq_P$ *Domino*. Hence, the language *Domino* is **NP**-complete. ∎

An example of a domino game

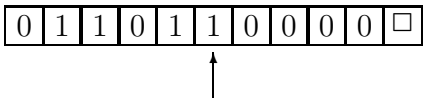
We have defined the domino game corresponding to a Turing machine that solves a decision problem. Of course, we can also do this for Turing machines that compute functions. In this section, we will exactly do this for a Turing machine that computes the successor function $x \to x + 1$.

We will design a Turing machine with one tape, that gets as input the binary representation of a natural number x, and that computes the binary representation of $x + 1$.

Start of the computation: The tape contains a 0 followed by the binary representation of the integer $x \in \mathbb{N}_0$. The tape head is on the leftmost bit (which is 0), and the Turing machine is in the start state z_0. Here is an example, where $x = 431$:



End of the computation: The tape contains the binary representation of the number $x + 1$. The tape head is on the rightmost 1, and the Turing machine is in the final state z_1. For our example, the tape looks as follows:



Our Turing machine will use the following states:

z_0 : start state; tape head moves to the right
z_1 : final state
z_2 : tape head moves to the left; on its way to the left, it has not read 0

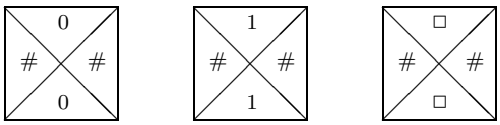
The Turing machine has the following instructions:

$$z_0 0 \to z_0 0R \qquad z_2 1 \to z_2 0L$$
$$z_0 1 \to z_0 1R \qquad z_2 0 \to z_1 1N$$
$$z_0 \square \to z_2 \square L$$

In Figure 6.6, you can see the sequence of states and tape contents of this Turing machine on input $x = 11$.

We now construct the domino game that corresponds to the computation of this Turing machine on input $x = 11$. Following the general construction in Section 6.5.3, we obtain the following tile types:

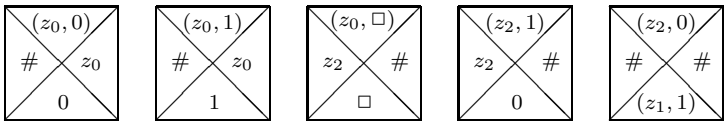
1. The three symbols of the alphabet yield three tile types:



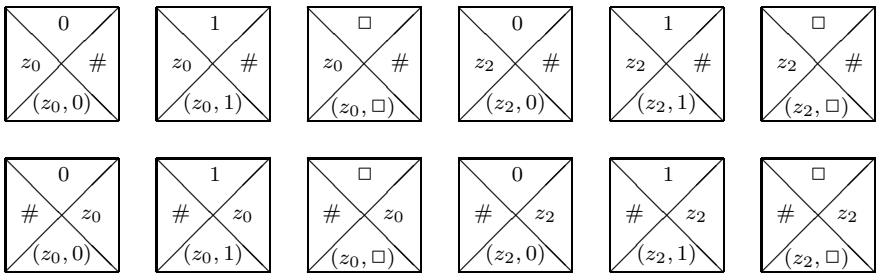
2. The five instructions of the Turing machine yield five tile types:

$(z_0, 0)$	1	0	1	1	$\square$
0	$(z_0, 1)$	0	1	1	$\square$
0	1	$(z_0, 0)$	1	1	$\square$
0	1	0	$(z_0, 1)$	1	$\square$
0	1	0	1	$(z_0, 1)$	$\square$
0	1	0	1	1	$(z_0, \square)$
0	1	0	1	$(z_2, 1)$	$\square$
0	1	0	$(z_2, 1)$	0	$\square$
0	1	$(z_2, 0)$	0	0	$\square$
0	1	$(z_1, 1)$	0	0	$\square$

Figure 6.6: *The computation of the Turing machine on input $x = 11$. The pair (state,symbol) indicates the position of the tape head.*



3. The states z_0 and z_2, and the three symbols of the alphabet yield twelve tile types:



The computation of the Turing machine on input $x = 11$ consists of nine computation steps. During this computation, the tape head visits exactly six cells. Therefore, the frame for the domino game has nine rows and six columns. This frame is given in Figure 6.7. In Figure 6.8, you find the solution of the domino game. Observe that this solution is nothing but an equivalent way of writing the computation of Figure 6.6. Hence, the computation of the Turing machine corresponds to a solution of the domino game; in fact, the converse also holds.

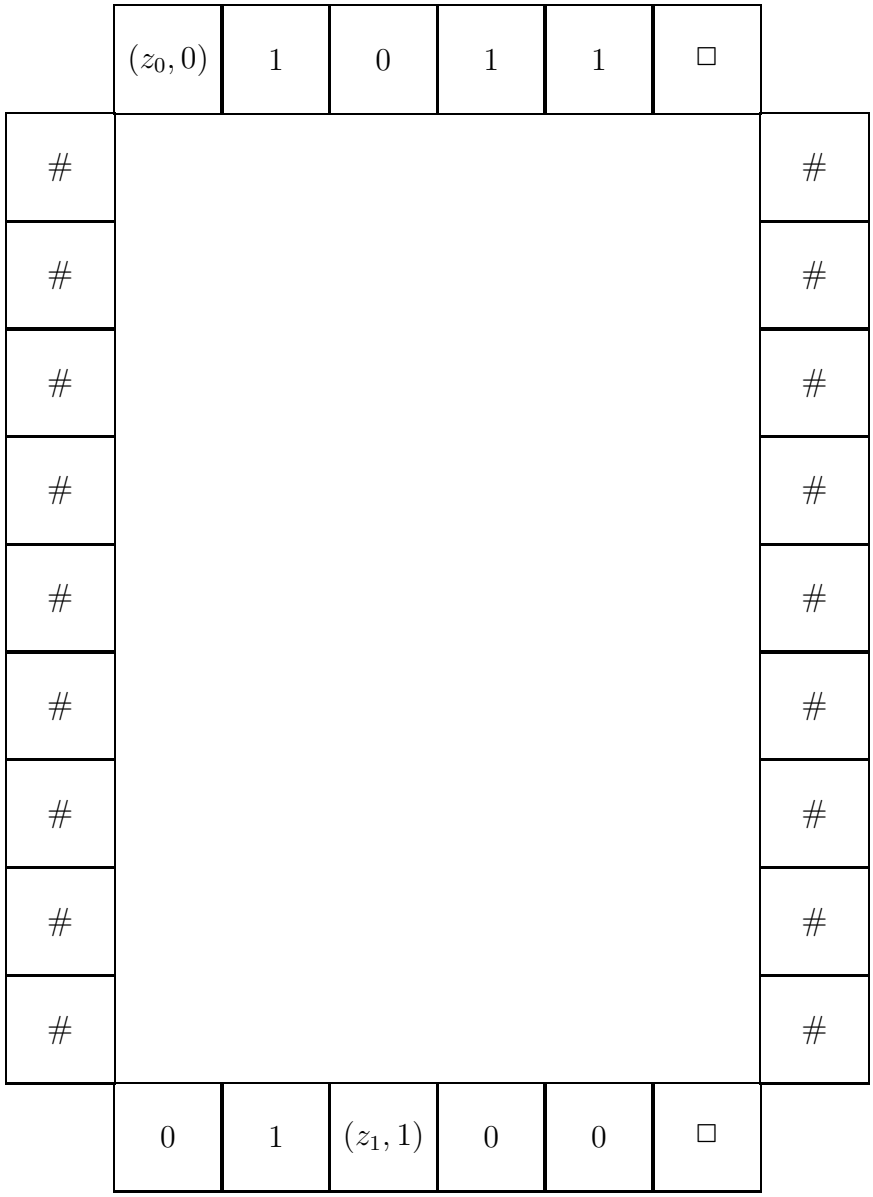


Figure 6.7: *The frame for the domino game for input* $x = 11$.

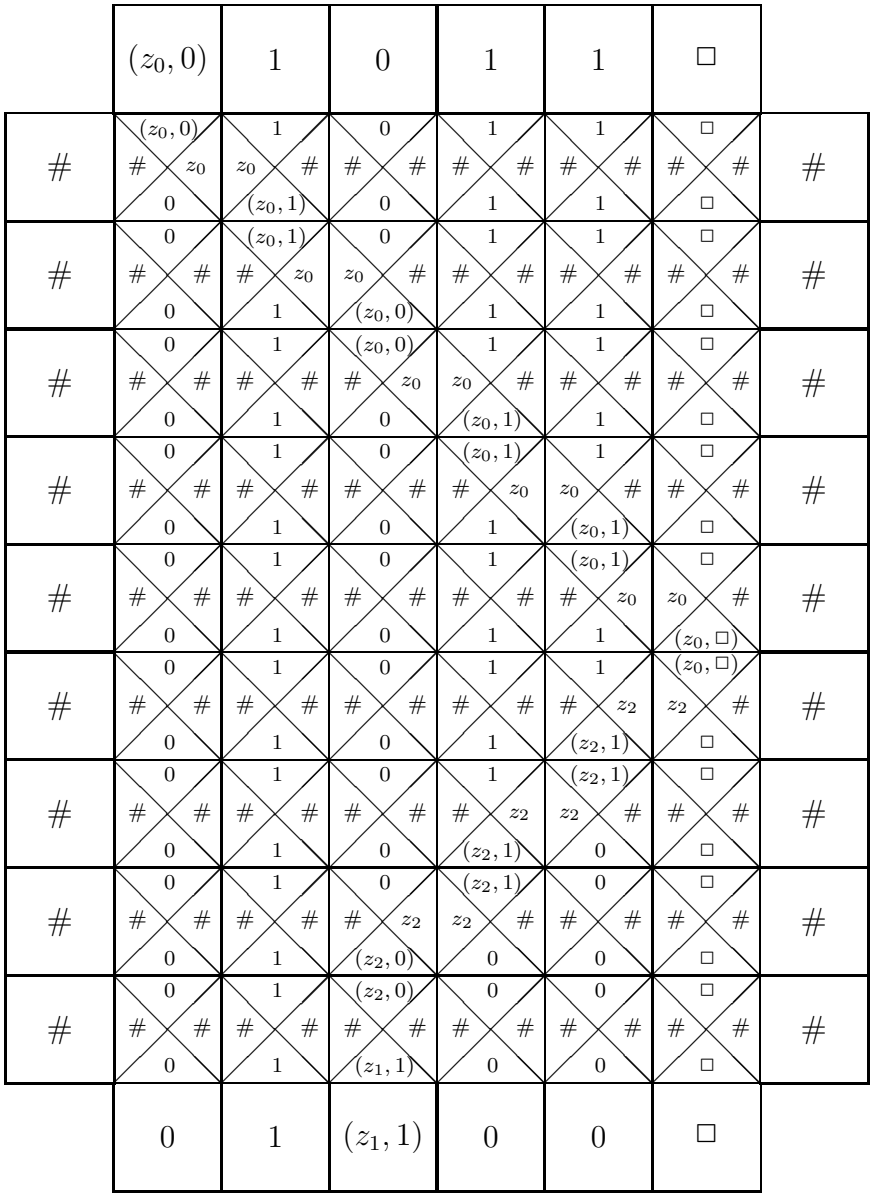


Figure 6.8: *The solution for the domino game for input* $x = 11$.

6.5.4 Examples of NP-complete languages

In Section 6.5.3, we have shown that *Domino* is **NP**-complete. Using this result, we will apply Theorem 6.5.11 to prove the **NP**-completeness of some other languages.

Satisfiability

We consider Boolean formulas φ, in the variables $x_1, x_2, \ldots, x_m$, having the form

$$\varphi = C_1 \wedge C_2 \wedge \ldots \wedge C_k, \tag{6.9}$$

where each C_i, $1 \leq i \leq k$, is of the following form:

$$C_i = \ell_1^i \vee \ell_2^i \vee \ldots \vee \ell_{k_i}^i.$$

Each ℓ_j^i is either a variable or the negation of a variable. Such a formula φ is said to be *satisfiable*, if there exists a truth-value in $\{0, 1\}$ for each of the variables $x_1, x_2, \ldots, x_m$, such that the entire formula φ is true. We define the following language:

$$SAT := \{\langle\varphi\rangle : \varphi \text{ is of the form (6.9) and is satisfiable}\}.$$

We will prove that SAT is **NP**-complete.

It is clear that $SAT \in \mathbf{NP}$. If we can show that

$$Domino \leq_P SAT,$$

then it follows from Theorem 6.5.11 that SAT is **NP**-complete. (In Theorem 6.5.11, take $B := Domino$ and $C := SAT$.)

Hence, we need a function $f \in \mathbf{FP}$, that maps input strings for *Domino* to input strings for SAT, in such a way that for every domino game D, the following holds:

$$\text{domino game } D \text{ is solvable} \iff \begin{array}{l}\text{the formula encoded by the} \\ \text{string } f(\langle D\rangle) \text{ is satisfiable.}\end{array} \tag{6.10}$$

Let us consider an arbitrary domino game D. Let k be the number of tile types, and let the frame have t rows and t columns. We denote the tile types by $T_1, T_2, \ldots, T_k$.

We map this domino game D to a Boolean formula φ, such that (6.10) holds. The formula φ will have variables

$$x_{ij\ell}, 1 \leq i \leq t, 1 \leq j \leq t, 1 \leq \ell \leq k.$$

These variables can be interpretated as follows:

$$x_{ij\ell} = 1 \Longleftrightarrow \text{there is a tile of type } T_\ell \text{ at position } (i, j) \text{ of the frame.}$$

We define:

- For all i and j with $1 \leq i \leq t$ and $1 \leq j \leq t$:

  $$C^1_{ij} := x_{ij1} \vee x_{ij2} \vee \ldots \vee x_{ijk}.$$

  This formula expresses the condition that there is at least one tile at position (i, j).

- For all i, j, ℓ and ℓ' with $1 \leq i \leq t$, $1 \leq j \leq t$, and $1 \leq \ell < \ell' \leq k$:

  $$C^2_{ij\ell\ell'} := \neg x_{ij\ell} \vee \neg x_{ij\ell'}.$$

  This formula expresses the condition that there is at most one tile at position (i, j).

- For all i, j, ℓ and ℓ' with $1 \leq i \leq t$, $1 \leq j < t$, $1 \leq \ell \leq k$ and $1 \leq \ell' \leq k$, such that $i < t$ and the right symbol on a tile of type T_ℓ is not equal to the left symbol on a tile of type $T_{\ell'}$:

  $$C^3_{ij\ell\ell'} := \neg x_{ij\ell} \vee \neg x_{i,j+1,\ell'}.$$

  This formula expresses the condition that neighboring tiles in the same row "fit" together. There are symmetric formulas for neighboring tiles in the same column.

- For all j and ℓ with $1 \leq j \leq t$ and $1 \leq \ell \leq k$, such that the top symbol on a tile of type T_ℓ is not equal to the symbol at position j of the upper boundary of the frame:

  $$C^4_{j\ell} := \neg x_{1j\ell}.$$

  This formula expresses the condition that tiles that touch the upper boundary of the frame "fit" there. There are symmetric formulas for the lower, left, and right boundaries of the frame.

The formula φ is the conjunction of all these formulas C^1_{ij}, $C^2_{ij\ell\ell'}$, $C^3_{ij\ell\ell'}$, and $C^4_{j\ell}$. The complete formula φ consists of

$$O(t^2k + t^2k^2 + t^2k^2 + tk) = O(t^2k^2)$$

terms, i.e., its length is polynomial in the length of the domino game. This implies that φ can be constructed in polynomial time. Hence, the function f that maps the domino game D to the Boolean formula φ, is in the class **FP**. It is not difficult to see that (6.10) holds for this function f. Therefore, we have proved the following result.

Theorem 6.5.13 *The language SAT is* **NP***-complete.*

In Section 6.5.1, we have defined the language $3SAT$.

Theorem 6.5.14 *The language* $3SAT$ *is* **NP***-complete.*

Proof. It is clear that $3SAT \in \mathbf{NP}$. If we can show that

$$SAT \leq_P 3SAT,$$

then the claim follows from Theorem 6.5.11. Let

$$\varphi = C_1 \wedge C_2 \wedge \ldots \wedge C_k$$

be an input for SAT, in the variables $x_1, x_2, \ldots, x_m$. We map φ, in polynomial time, to an input φ' for $3SAT$, such that

$$\varphi \text{ is satisfiable} \iff \varphi' \text{ is satisfiable.} \tag{6.11}$$

For each i with $1 \leq i \leq k$, we do the following. Consider

$$C_i = \ell^i_1 \vee \ell^i_2 \vee \ldots \vee \ell^i_{k_i}.$$

- If $k_i = 1$, then we define

$$C'_i := \ell^i_1 \vee \ell^i_1 \vee \ell^i_1.$$

- If $k_i = 2$, then we define

$$C'_i := \ell^i_1 \vee \ell^i_2 \vee \ell^i_2.$$