Examples of Proofs of NP-completeness

NP-completeness of SAT problem

Recall the SAT problem:

SAT (satisfiability of boolean formulas)

Input: Boolean formula φ.

Question: Is φ satisfiable?

It is easy to show that SAT belongs to NPTIME:

A nondeterministic algorithm solving SAT in polynomial times works as follows:

- It nondeterministically chooses truth valuation ν that assigns boolean value to each variable occurring in formula φ.
- It evaluates φ is valuation ν, i.e., computes value $[\varphi]_\nu$.
- If $[\varphi]_\nu = 1$, the algorithm returns answer YES.
  Otherwise, it returns answer NO.

NP-completeness of SAT problem

To show that SAT is NP-hard is more difficult.

It is necessary to show that for every problem $P \in$ NPTIME there exists a polynomial reduction from problem P to SAT,
i.e., to show that there exists an algorithm that:

- obtains as an input an (arbitraty) instance of problem P,
- for this instance, it constructs a boolean formula φ such that φ is satisfiable iff for the given instance of problem P the answer is YES,
- it will be of polynomial time complexity.

NP-completeness of SAT problem

If $P \in$ NPTIME, there must exist **nondeterministic** Turing machine $\mathcal{M}$ and a polynomial $p(n)$ such that:

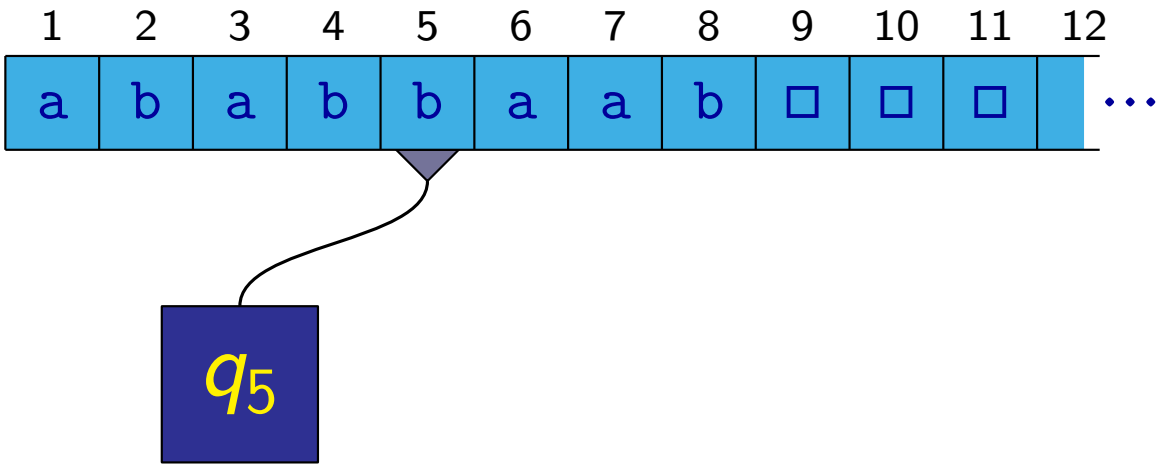
- It holds for every instance w of problem P (represented as a word over some alphavet Σ) that:
  - If the answer for w is YES, then there exists at least one computation of machine $\mathcal{M}$ on word w where machine $\mathcal{M}$ gives answer YES.
  - If the answer for w is NO, then all computations of machine $\mathcal{M}$ on word w end with answer NO.
- The machine $\mathcal{M}$ executes in every computation on word w at most $p(|w|)$ steps.

NP-completeness of SAT problem

We will show how for a given nondeterministic Turing machine $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, F)$, a polynomial $p(n)$, and word $w \in \Sigma^*$ to construct a boolean formula φ such that:

- φ will be satisfiable iff there exists a computation of the machine $\mathcal{M}$ on the word w where $\mathcal{M}$ executes $p(|w|)$ steps and gives answer YES.
- The formula φ could be constructed in polynomial time with respect to the size of the word w.

NP-completeness of SAT problem

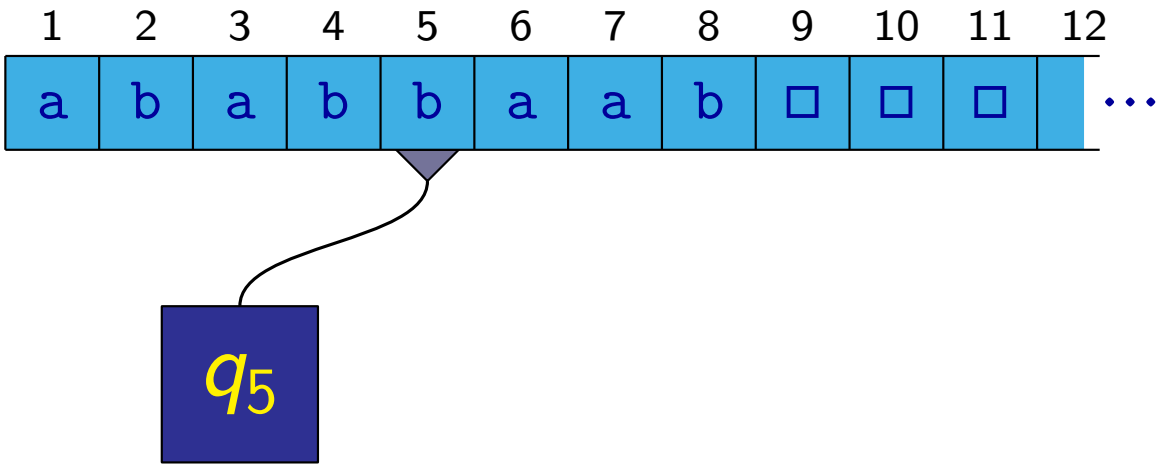


We will assume that the Turing machine uses a one-sided tape.

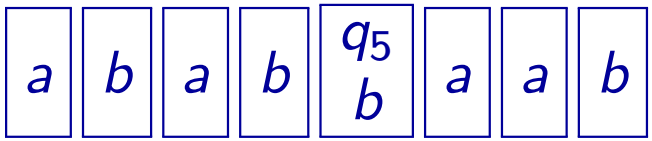
The cells of the tape can be numbered $1, 2, 3, \ldots$.

We can also assume that the final states are $F = \{q_{acc}, q_{rej}\}$ (q_{acc} — accepts input, q_{rej} — rejects input).

NP-completeness of SAT problem



Configurations can be written as words over alphabet $\Gamma \cup (Q \times \Gamma)$:



This word always contains exactly one symbol from $(Q \times \Gamma)$ that denotes a state of the control unit and a position of the head.

NP-completeness of SAT problem

We also assume that the time complexity of the machine $\mathcal{M}$ is bounded from above by a function $p(n)$ where $p(n)$ is a polynomial.

(Without loss of generality we can assume that for all n we have $p(n) \geq n$.)

If the machine $\mathcal{M}$ obtains a word w of length n as an input, it will execute at most $p(n)$ steps during a computation.

Because the machine starts with the head on cell number 1, the head can reach during this computation at most the cell number $p(n) + 1$ (in every step it moves by one cell).

NP-completeness of SAT problem

So if the time complexity of the machine $\mathcal{M}$ is bounded from above by function $p(n)$, all configurations in a computation on an input of size n can be written as words of length $p(n) + 1$.

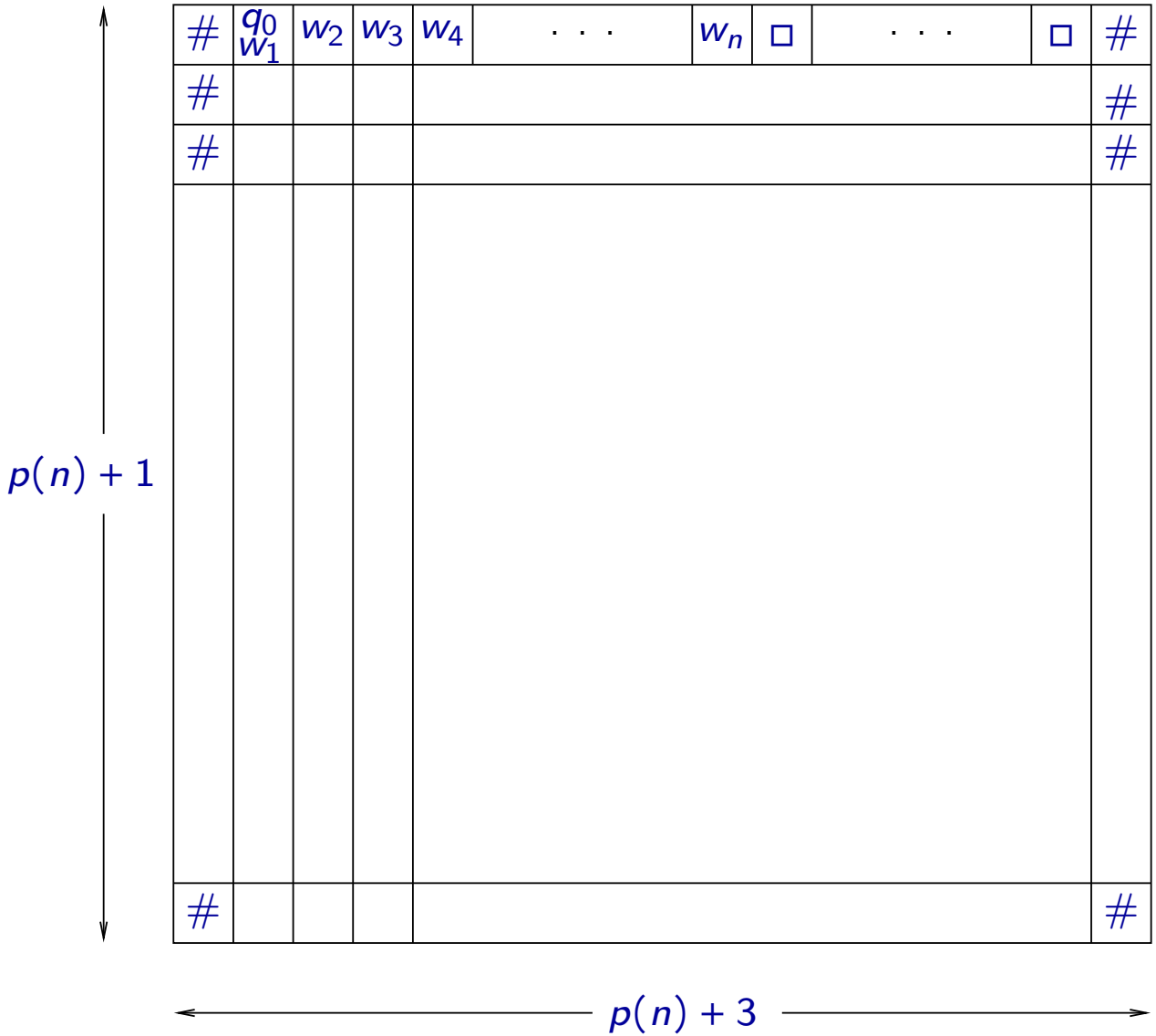
So cells with numbers greater than $p(n) + 1$ are not reached during a computation and they will contain symbol $\square$ (recall that we assume that $p(n) \geq n$).

NP-completeness of SAT problem

Words representing configuration in a computation of the machine $\mathcal{M}$ on the word $w = w_1 w_2 \cdots w_n$ can be written to a table where:

- Rows correspond to configurations (written as words over alphabet $\Gamma \cup (Q \times \Gamma)$).
- Columns correspond to the cells of the tape with numbers $1, 2, \ldots, p(n) + 1$.
- For technical reasons we also add two columns from the left and from the right containg only specicial separating symbols $\#$ (where $\# \notin Q \cup \Gamma$).

NP-completeness of SAT problem



NP-completeness of SAT problem

- So individual cells of the table will contain symbols from alphabet
$$\Delta = \Gamma \cup (Q \times \Gamma) \cup \{\#\}$$
- Rows will be numbered from 0 to $p(n) + 1$.
(Row 0 will contain the initial configuration.)
- Columns will be numbered from 0 to $p(n) + 2$.
(Columns 0 and $p(n) + 2$ will contain symbols $\#$.)

NP-completeness of SAT problem

Remark: A computation can be shorter than $p(n)$ steps, and in such case, some rows of the table would not be filled.

To ensure that the table is always completely filled, we can repeat the last configuration where computation halts, and copy it to all remaining rows of the table.

NP-completeness of SAT problem

So if we have a (nondeterministic) Turing machine $\mathcal{M}$ with a time complexity bounded from above by a polynomial $p(n)$ solving problem P and an instance of this problem written as a word w, the answer for this instance is YES iff there exists an accepting computation of the machine $\mathcal{M}$ on word w.

Such accepting computation can be written in the way described above into a table with $p(n) + 1$ rows and $p(n) + 3$ columns.

Remark: Note that that the size of the table is polynomial with respect to n.

NP-completeness of SAT problem

For the given machine $\mathcal{M}$, the polynomial $p(n)$, and the word w, a formula φ is created such that:

- Different valuations ν of boolean variables in the formula φ will represent all possible (including nonsensical) contents of the table.
- $[\varphi]_\nu = 1$ will hold for exactly those valuations ν that represent such a content of the table that is a representations of an accepting computation of the machine $\mathcal{M}$ on the input w.

So the formula φ will be satisfiable iff there exists an accepting computation of the machine $\mathcal{M}$ on the input w.

NP-completeness of SAT problem

The formula φ will be composed using logical connectives from atomic propositions of the form:

$$\text{“Cell } (i,j) \text{ contains symbol } a\text{.”}$$

where i, j, a will be particular constants, as in:

$$\text{“Cell } (9,4) \text{ contains symbol } \boxed{\begin{matrix} q_5 \\ b \end{matrix}}\text{.”}$$

Remark: When we talk about cell (i,j), we mean the cell on i-th row and j-th column of the table.

NP-completeness of SAT problem

So the formula φ will contain boolean variables $x_{i,j}^{a}$ where:

- $0 \leq i \leq p(n) + 1$
- $0 \leq j \leq p(n) + 2$
- $a \in \Delta$ (where $\Delta = \Gamma \cup (Q \times \Gamma) \cup \{\#\}$)

with the intended meaning that $\nu(x_{i,j}^{a}) = 1$ holds if ν represents a content of the table where cell (i, j) contains symbol a,

and $\nu(x_{i,j}^{a}) = 0$ means that ν represents a content of the table where cell (i, j) does not contain symbol a.

NP-completeness of SAT problem

Example: Variable $x_{9,4}^{q_5,\mathrm{b}}$ represents proposition:

"*Cell* $(9,4)$ *contains symbol* $\boxed{\begin{matrix} q_5 \\ \mathrm{b} \end{matrix}}$."

Remark: Values from $(Q \times \Gamma)$ in indexes will be written as q, a instead of $\boxed{\begin{matrix} q \\ a \end{matrix}}$.

So if $\nu(x_{9,4}^{q_5,\mathrm{b}}) = 1$, that in the content of the table represented by ν, the cell $(9,4)$ contains the symbol $\boxed{\begin{matrix} q_5 \\ \mathrm{b} \end{matrix}}$,

and if $\nu(x_{9,4}^{q_5,\mathrm{b}}) = 0$, then the cell $(9,4)$ does not contain $\boxed{\begin{matrix} q_5 \\ \mathrm{b} \end{matrix}}$.

NP-completeness of SAT problem

The whole formula φ will basically say:

> *The table contains an accepting computation of the machine* $\mathcal{M}$ *on the word* w*.*

It will consist of many subformulas where each of these subformulas will describe some simple condition that must by satisfied for an accepting computation of the machine $\mathcal{M}$ on the word w.

These subformulas will be connected by conjunctions.

So if for a given valuation ν some of these conditions is violated, the whole formula φ will have value 0, i.e., $[\varphi]_\nu = 0$.

NP-completeness of SAT problem

In the following description, the individual subformulas will be described.

When we say about a formula ψ that

"ψ is added to φ,"

we mean that ψ will be connected using conjunction ($\wedge$) with the the part of the formula φ that was created so far.

NP-completeness of SAT problem

To ensure that the table really contains an accepting computation of the machine $\mathcal{M}$ of the word w, the following conditions must be satisfied:

1. Every cell of the table contains exactly one symbol from Δ.
2. The row 0 contains the initial configuration on the word w.
3. Every row of the table (except the row 0) contains either:
   - a configuration that is reachable by one step (according to the transition function δ) from a configuration written in the previous row, or
   - a final configuration that is a copy of the configuration in the previous line.
4. The last row of the table contains a configuration with the state q_{acc}.

NP-completeness of SAT problem

It is obvious that if the table contains an accepting computation, then these four conditions are satisfied.

On the other hand, it is also obvious that if these four conditions are satisfied, then the table really contains an accepting computation of the machine $\mathcal{M}$ on the word w.

NP-completeness of SAT problem

Take a look at the first condition:

Every cell in the table contains exactly one symbol from Δ.

This will be ensured by adding, for every cell (i, j), a subformula to φ that will say:

The cell (i, j) *exactly one symbol from* Δ,

which can be also formulated as follows:

Exactly one variable from variables $x_{i,j}^{a_1}$, $x_{i,j}^{a_2}$, $\ldots$, $x_{i,j}^{a_k}$ *has value* 1,

where $\{a_1, a_2, \ldots, a_k\}$ is the set of all symbols from Δ.

NP-completeness of SAT problem

To express the proposition that exactly one of variables $x_1, x_2, \ldots, x_k$ has value 1 (where $x_1, x_2, \ldots, x_k$ are some arbitrary boolean variables) is not difficult.

We will demostrate on an example where for simplicity we have only four boolean variables A, B, C, D:

$$\begin{aligned}
&(\ \ A \wedge \neg B \wedge \neg C \wedge \neg D) \ \vee \\
&(\neg A \wedge \ \ B \wedge \neg C \wedge \neg D) \ \vee \\
&(\neg A \wedge \neg B \wedge \ \ C \wedge \neg D) \ \vee \\
&(\neg A \wedge \neg B \wedge \neg C \wedge \ \ D) \ \vee
\end{aligned}$$

It is not difficult to check that this formula has value 1 in exactly those valuations where exactly one of variables A, B, C, D has value 1.

NP-completeness of SAT problem

In general, for a set of variables $X = \{x_1, x_2, \ldots, x_k\}$, this condition can be written as follows:

$$\bigvee_{x_i \in X} \left(x_i \ \wedge \bigwedge_{x_j \in X - \{x_i\}} \neg x_j \right)$$

Note that for k variables, this formula is of size $\mathcal{O}(k^2)$.

In our case $k = |\Delta|$, and so the size of each subformula added for each cell is $\mathcal{O}(|\Delta|^2)$ and so it is a constant that does not depend on the size of the input w.

Remark: There is a way how to represent the given condition using a formula of size $\mathcal{O}(k \log k)$ but we do need it here.

NP-completeness of SAT problem

The next condition that must be satisfied, is:

The row 0 contains the initial configuration with the word w.

So if $w_1 w_2 \dots w_n$ are the symbols of the word w while $n \geq 1$, the following must be true:

- Cell $(0, 1)$ contains symbol (q_0, w_1) where q_0 is the initial state.
- Cells $(0, 2)$, $(0, 3)$, $\dots$, $(0, n)$ contain symbols $w_2, w_3, \dots, w_n$.
- Cells $(0, n+1)$, $(0, n+2)$, $\dots$, $(0, p(n+1))$ contain symbol $\square$.
- Cells $(0, 0)$ and $(0, p(n)+2)$ contain symbol $\#$.

NP-completeness of SAT problem

So this condition can be represented by the following formula that is added to φ:

$$x_{0,1}^{q_0,w_1} \wedge \left(\bigwedge_{i=2}^{n} x_{0,i}^{w_i} \right) \wedge \left(\bigwedge_{i=n+1}^{p(n)+1} x_{0,i}^{\square} \right) \wedge x_{0,0}^{\#} \wedge x_{0,p(n)+2}^{\#}$$

The size of this formula is $\mathcal{O}(p(n))$.

Remark: In the case when $n = 0$, the only difference would be that instead of $x_{0,1}^{q_0,w_1}$ the formula would contain $x_{0,1}^{q_0,\square}$.

NP-completeness of SAT problem

The most complicated is ensuring of the third condition:

> *Every row (except the row 0) contains a configuration that is obtained from the previous configuration by executing a single step (or it is a copy of the previous final configuration).*

Consider two consecutive configurations.

A difference between these two configurations is always in at most two positions:

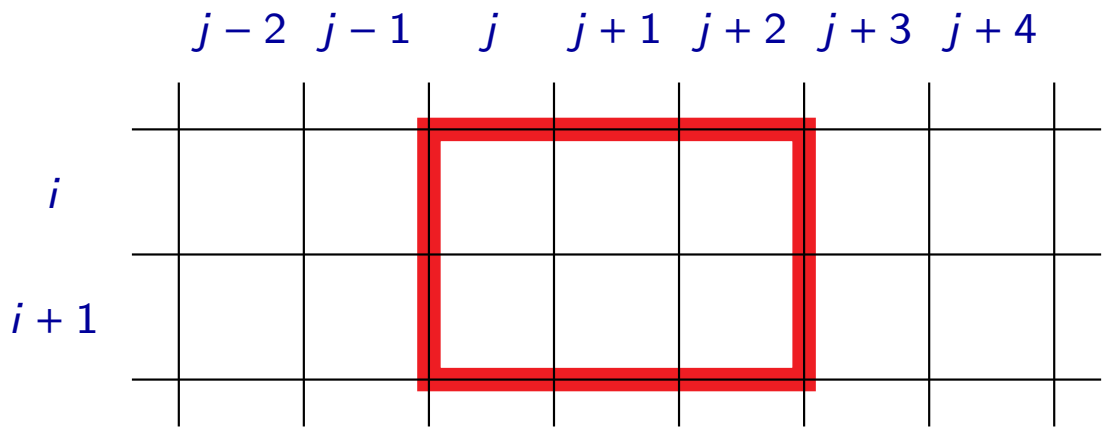
- at the position where the head occurs in the first of these configurations,
- and at the neighbouring position where the head moves.

So the content of rows i and $i + 1$ in the table is always closely connected to each other.

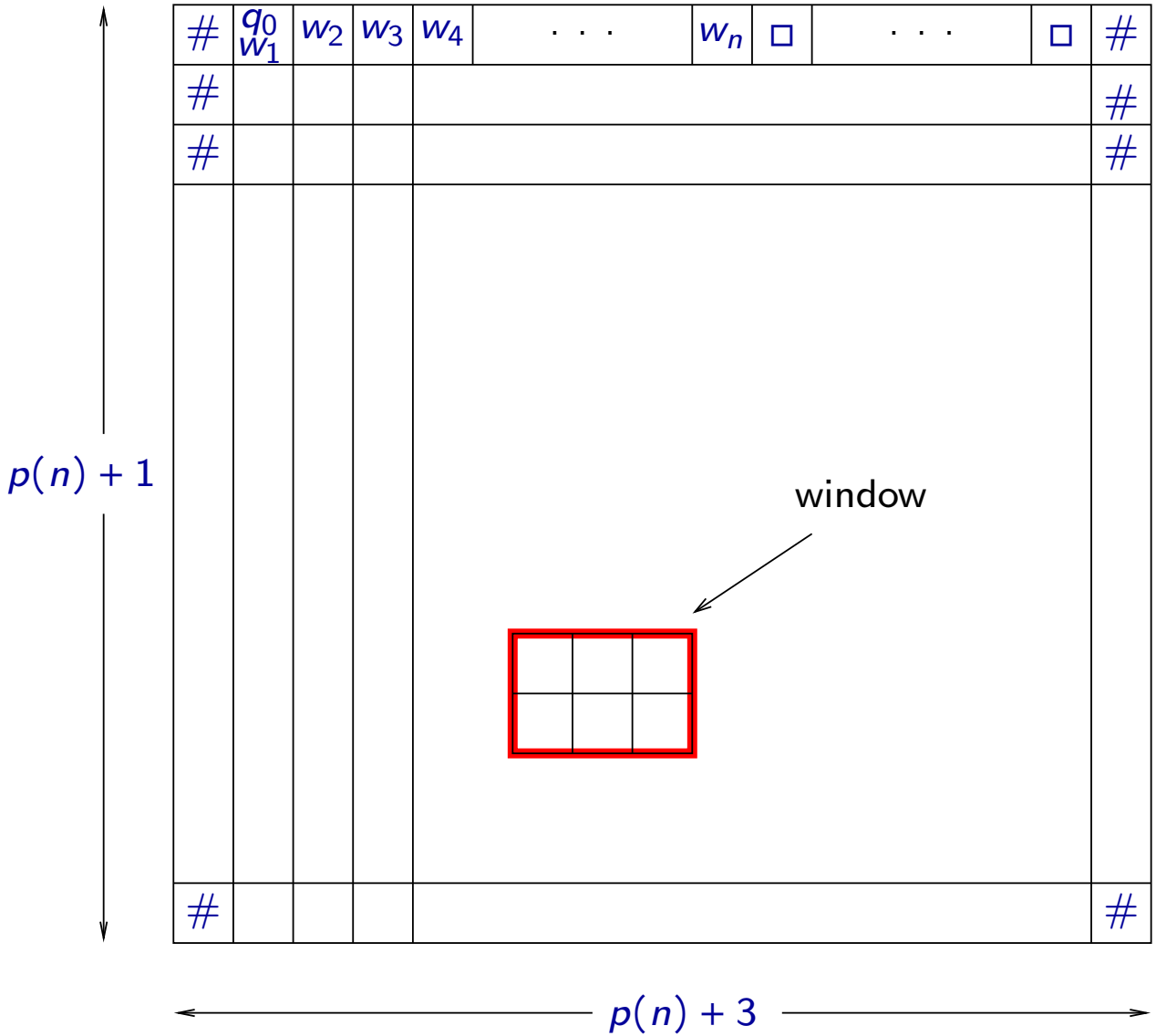
NP-completeness of SAT problem

If a content of row $i+1$ does not correspond to a configuration reachable by one step from a configuration on row i, we can check this by finding a particular position where these configurations do not "fit".

It is not hard to see that in such case we can always find a "window" of size 2×3 for these two configurations such that to check that these two rows do not contain consecutive configurations, it is sufficient to consider the content of this window (i.e., without considering the content of the other cells).



NP-completeness of SAT problem



NP-completeness of SAT problem

Those contents of windows that can occur in two consecutive configurations will be called **correct**, and those can not occur in two consecutive configurations (and so witness that these two lines do not represent consecutive configurations) are called **incorrect**.

We will not describe here the exact rules that a correct contents of a window need to satisfy .

Instead we will see examples of correct and incorrect windows.

But you can try (after seeing these example) to formulate these conditions yourself.

NP-completeness of SAT problem

Examples of **incorrect** windows where we assume that $\delta(q_5, \mathtt{a}) = \{(q_8, \mathtt{b}, -1), (q_3, \mathtt{a}, +1)\}$:

a	b	a
b	#	a

a	q_5 a	a
b	a	a

a	q_5 a	b
q_4 □	b	q_7 a

q_5 a	b	a
b	q_6 b	a

q_{acc} a	a	b
a	q_8 a	b

a	b	b
a	b	a

NP-completeness of SAT problem

Examples of **correct** windows where we assume $\delta(q_5, \mathtt{a}) = \{(q_8, \mathtt{b}, -1), (q_3, \mathtt{a}, +1)\}$:

a	b	a
a	b	a

a	q_5 a	b
a	a	q_3 b

b	a	□
q_3 b	a	□

q_5 a	b	a
b	b	a

a	a	q_{acc} b
a	a	q_{acc} b

□	□	#
□	□	#

NP-completeness of SAT problem

We denote the set of all tuples of six symbols that form a correct contents of windows (for the given particular machine $\mathcal{M}$) as *Corr*.

I.e., $(a, b, c, d, e, f) \in Corr$ iff

a	b	c
d	e	f

is a correct content of a window.

The total number of all possible contents of a window is $|\Delta|^6$, which is a constant independent of the size the input w, and so also the number of elements of the set *Corr* is a constant independent of the size of the input.

NP-completeness of SAT problem

For each window in the table we add a subformula to φ that claims that the content of the given window is correct (i.e., that it contains one of the correct tuples of six symbols).

I.e., for each i such that $0 \leq i < p(n)$, and each j such that $0 \leq j \leq p(n)$, we add the following subformula to φ:

$$\bigvee_{\substack{(a,b,c,d,e,f) \\ \in Corr}} \left(x^a_{i,j} \wedge x^b_{i,j+1} \wedge x^c_{i,j+2} \wedge x^d_{i+1,j} \wedge x^e_{i+1,j+1} \wedge x^f_{i+1,j+2} \right)$$

Each of these subformulas is of the size at most $\mathcal{O}(|\Delta|^6)$, i.e., bounded by some constant independent of the size of the input w.

The total number of these subformulas is $\mathcal{O}(p(n)^2)$

NP-completeness of SAT problem

Now it remains to ensure the last condition:

> *The last line of the table contains a configuration where the state of the control unit is* q_{acc}.

Again, this is simple — it is sufficient to add a subformula to φ that claims that on some position in the last line (i.e., on the line $p(n)$) there is a pair (q_{acc}, a) where a a symbol from Γ.

This subformula looks as follows:

$$\bigvee_{j=1}^{p(n)+1} \bigvee_{a \in \Gamma} x_{p(n),j}^{(q_{acc},a)}$$

The size of this formula is $\mathcal{O}(p(n))$.

NP-completeness of SAT problem

We can see from the previous description that the size of the formula φ created for the given input w of the size n is in $\mathcal{O}(p(n)^2)$.

If $p(n)$ is a polynomial then also $p(n)^2$ is a polynomial, and so the size of φ is polynomial with respect to n.

Because the structure of the formula φ is simple and regular, it is also obvious that the time complexity of the algorithm that creates the formula for the given word w basically corresponds to the size of the formula φ, and it is also $\mathcal{O}(p(n)^2)$.

NP-completeness of SAT problem

So we have seen that the construction is polynomial; now we shortly describe why it is correct:

- Let us assume that the answer for w in the problem P is $\textsc{Yes}$, which means that there exists a computation of the nondeterministic machine $\mathcal{M}$ (that solves the problem P) on the word w that gives the answer $\textsc{Yes}$.

  This computation can be written to the table, and the variables in the formula φ can be assigned boolean values according to the content of this table.

  It is obvious that in this assignment φ will have the value 1 becuase all conditions tested in the fomula φ are satisfied.

NP-completeness of SAT problem

- Let us assume now that the formula φ is satisfiable, i.e., for some assignment ν is $[\varphi]_\nu = 1$.

  Now, according to the valuation ν we can fill in the table.

  Because all conditions described in the formula φ must be satisfied for the valuation ν, it follows that the table filled this way contains a description of an computation of the machine $\mathcal{M}$ on the word w where this machine gives the answer YES, and so that such computation exists.

We can see that the formula φ is satisfiable iff there exists a computation of the machine $\mathcal{M}$ that accepts the word w, i.e., iff the answer is YES.

Reduction from SAT to 3-SAT

We have seen that the SAT problem is NP-complete.

We will now show that it is still NP-complete even if it is restricted to formulas of a certain specific form:

3-SAT

Input: A boolean formula φ in conjunctive normal form where every clause contains exactly 3 literals.

Question: Is φ satisfiable?

Reduction from SAT to 3-SAT

We will describe an algorithm that for a given formula φ constructs a formula φ' such that:

- φ' will be in CNF and each of its clauses will contain exactly 3 literals,
- φ' will be satisfiable iff φ will be satisfiable.

Remark: The simple idea — to transform φ to CNF — does not work. The problem is that the resulting formula can be exponentially bigger than φ (and so it can not be constructed in a polynomial time).

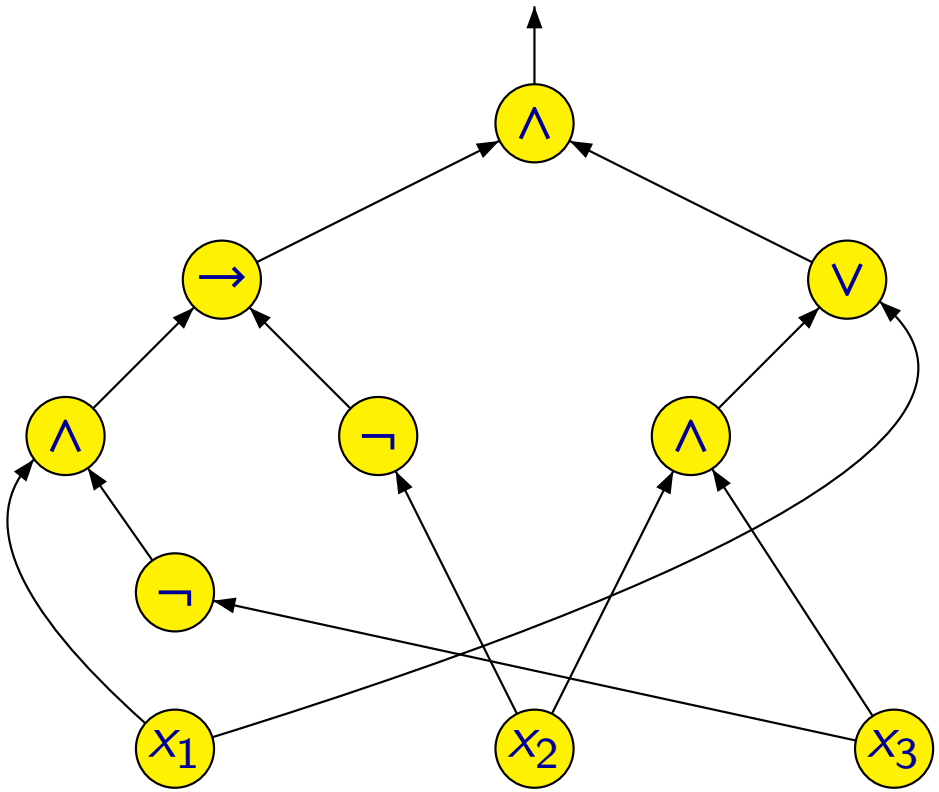
Reduction from SAT to 3-SAT

The algorithm will be divided into two parts:

- At first we construct a formula φ_1, which will be in CNF and will contain **at most** 3 literals in every clause
(and that will be satisfiable iff φ is satisfiable).
- Then we will construct formula φ' from φ_1 that will be in CNF and that will contain **exactly** 3 literals in every clause
(and that will be satisfiable iff φ_1 is satisfiable).

Reduction from SAT to 3-SAT

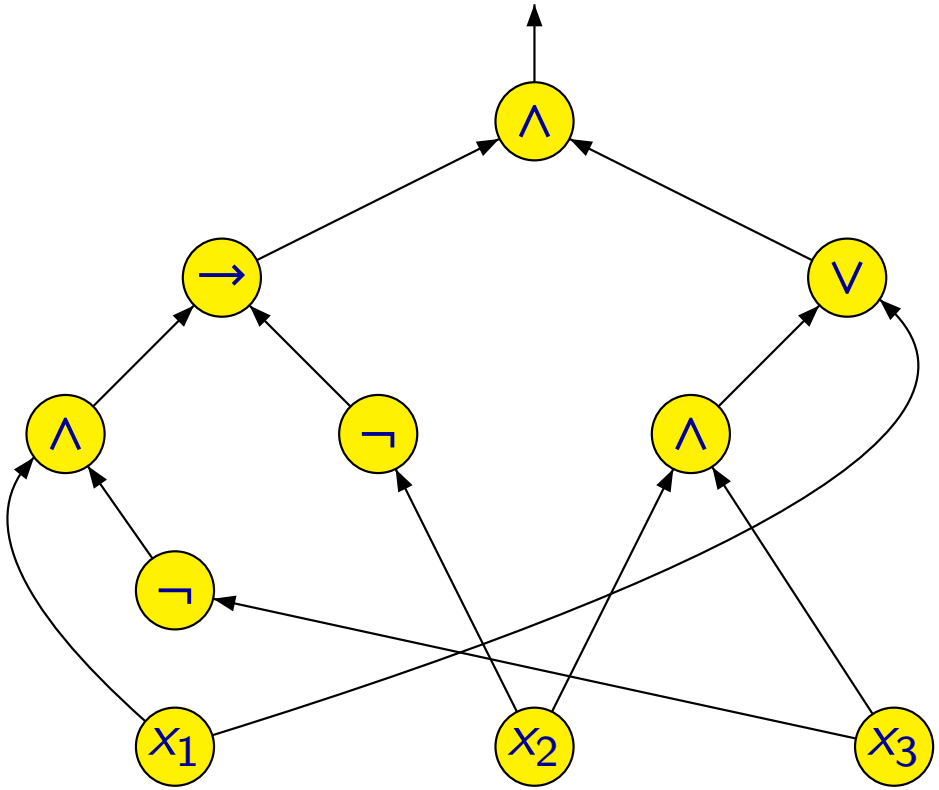
Formula φ can be represented as a boolean circuit whose structure is given by an (abstract) syntax tree of the given formula:



$$((x_1 \wedge \neg x_3) \rightarrow \neg x_2) \wedge ((x_2 \wedge x_3) \vee x_1)$$

Reduction from SAT to 3-SAT

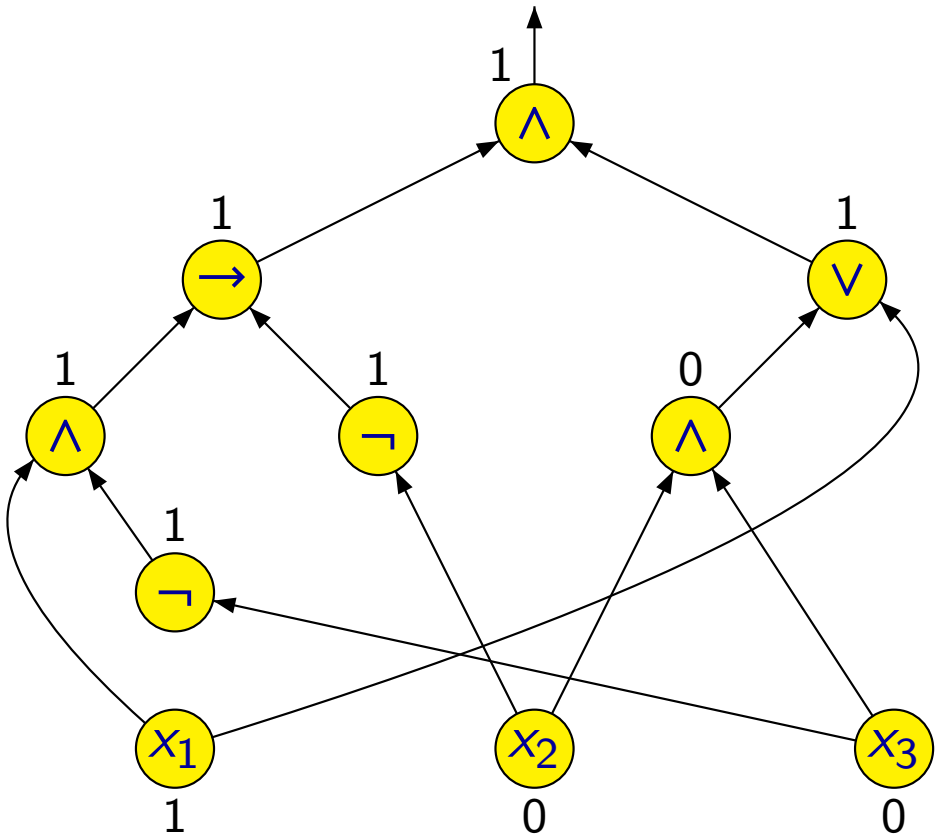
The formula φ is satisfiable iff there exists some input, for which we obtain 1 on the output.



$$((x_1 \wedge \neg x_3) \rightarrow \neg x_2) \wedge ((x_2 \wedge x_3) \vee x_1)$$

Reduction from SAT to 3-SAT

The formula φ is satisfiable iff there exists some input, for which we obtain 1 on the output.



$$((x_1 \wedge \neg x_3) \rightarrow \neg x_2) \wedge ((x_2 \wedge x_3) \vee x_1)$$

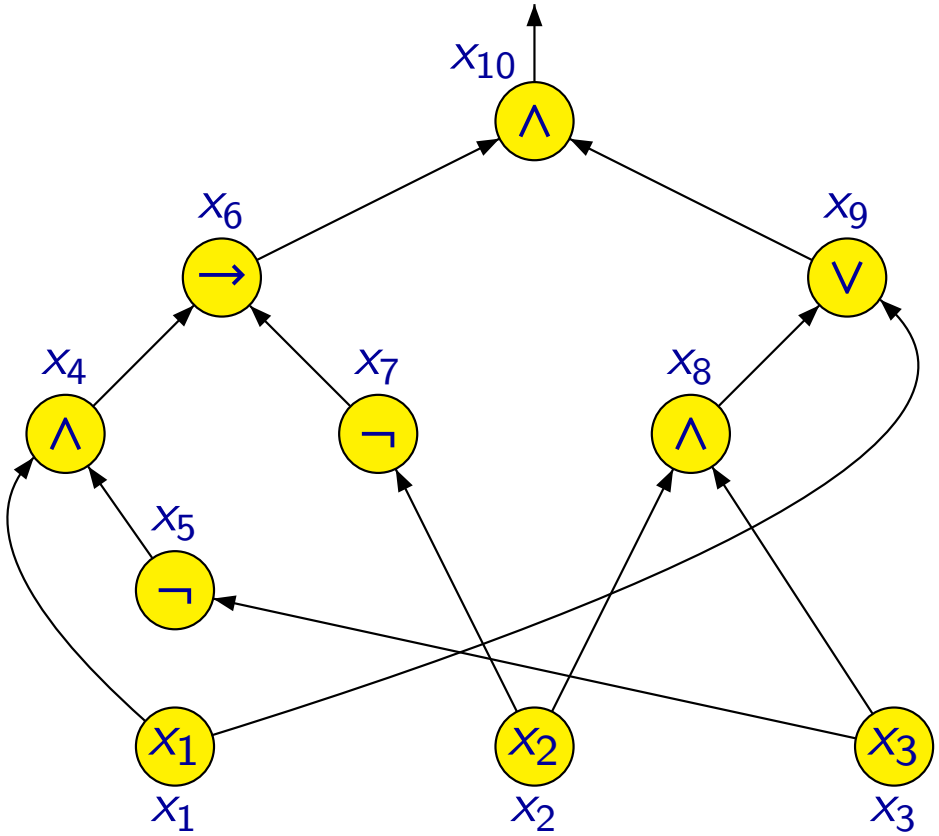
Reduction from SAT to 3-SAT

The formula φ_1 that will be constructd for a given formula φ will contain the following variables:

- all variables that occur in the original formula φ
  (i.e., one variable for every input of the circuit),
- one variable for every occurrence of a boolean operator in φ
  (i.e., one variable for every gate of the circuit).

Reduction from SAT to 3-SAT

Example: Formula φ_1 will contain variables $x_1, x_2, \ldots, x_{10}$.



$$((x_1 \wedge \neg x_3) \rightarrow \neg x_2) \wedge ((x_2 \wedge x_3) \vee x_1)$$

Reduction from SAT to 3-SAT

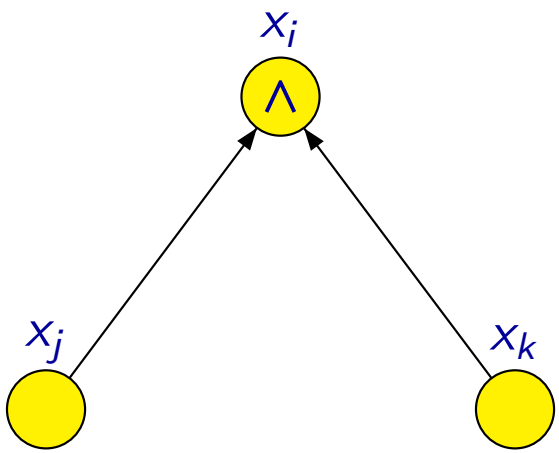
The formula φ_1 will be constructed in such a way that for each valuation ν it will hold $[\varphi_1]_\nu = 1$ iff:

- ν represents a correct assignment of boolean values to all inputs and outputs of all gates, and
- there is the value 1 on the output of the circuit.

(If some of these conditions is violated then $[\varphi_1]_\nu = 0$.)

Reduction from SAT to 3-SAT

Let us concentrate now on a single gate (e.g., of type $\wedge$) whose output is represented by a variable x_i and whose inputs are represented by variables x_j and x_k.



Reduction from SAT to 3-SAT

Possible (correct) assignments of values on inputs and on the output of the given gate (of type $\wedge$) are described by the following table:

x_j	x_k	x_i
0	0	0
0	1	0
1	0	0
1	1	1

The content of this table can represented by the following formula ψ:

$$
\begin{array}{lllllll}
(& \neg x_j & \wedge & \neg x_k & \rightarrow & \neg x_i &) \wedge \\
(& \neg x_j & \wedge & x_k & \rightarrow & \neg x_i &) \wedge \\
(& x_j & \wedge & \neg x_k & \rightarrow & \neg x_i &) \wedge \\
(& x_j & \wedge & x_k & \rightarrow & x_i &)
\end{array}
$$

Reduction from SAT to 3-SAT

The formula ψ represents the table given above in the sense that ψ has the value 1 for exactly those assignments that occur in this table (and for those that do not occur there, it has the value 0).

Reduction from SAT to 3-SAT

An arbitrary formula of the form

$$A \wedge B \rightarrow C$$

can be transformed to the formula of the form

$$\neg(A \wedge B) \vee C$$

and this can be rewritten to the equivalent formula of the form

$$\neg A \vee \neg B \vee C$$

Reduction from SAT to 3-SAT

So formula

$$
\begin{array}{lllllll}
(& \neg x_j & \wedge & \neg x_k & \rightarrow & \neg x_i &) \wedge \\
(& \neg x_j & \wedge & x_k & \rightarrow & \neg x_i &) \wedge \\
(& x_j & \wedge & \neg x_k & \rightarrow & \neg x_i &) \wedge \\
(& x_j & \wedge & x_k & \rightarrow & x_i &)
\end{array}
$$

can be rewritten to the equivalent formula

$$
\begin{array}{lllllll}
(& x_j & \vee & x_k & \vee & \neg x_i &) \wedge \\
(& x_j & \vee & \neg x_k & \vee & \neg x_i &) \wedge \\
(& \neg x_j & \vee & x_k & \vee & \neg x_i &) \wedge \\
(& \neg x_j & \vee & \neg x_k & \vee & x_i &)
\end{array}
$$

Reduction from SAT to 3-SAT

For a gate of the type $\vee$, we can use similar approach.

For the table

x_j	x_k	x_i
0	0	0
0	1	1
1	0	1
1	1	1

we can construct formula

$$
\begin{array}{lllllllll}
(& x_j & \vee & x_k & \vee & \neg x_i &) & \wedge \\
(& x_j & \vee & \neg x_k & \vee & x_i &) & \wedge \\
(& \neg x_j & \vee & x_k & \vee & x_i &) & \wedge \\
(& \neg x_j & \vee & \neg x_k & \vee & x_i &) &
\end{array}
$$

Reduction from SAT to 3-SAT

In a similar way, we can represent also other boolean operations ($\rightarrow$, $\leftrightarrow$, ...).

As an example, here we have also a construction for a gate of the type $\neg$ (there is now just one input x_j):

For the table

x_j	x_i
0	1
1	0

the corresponding formula is

$$(\neg x_j \rightarrow x_i) \ \wedge \ (x_j \rightarrow \neg x_i)$$

that can be rewritten to the form

$$(x_j \vee x_i) \ \wedge \ (\neg x_j \vee \neg x_i)$$

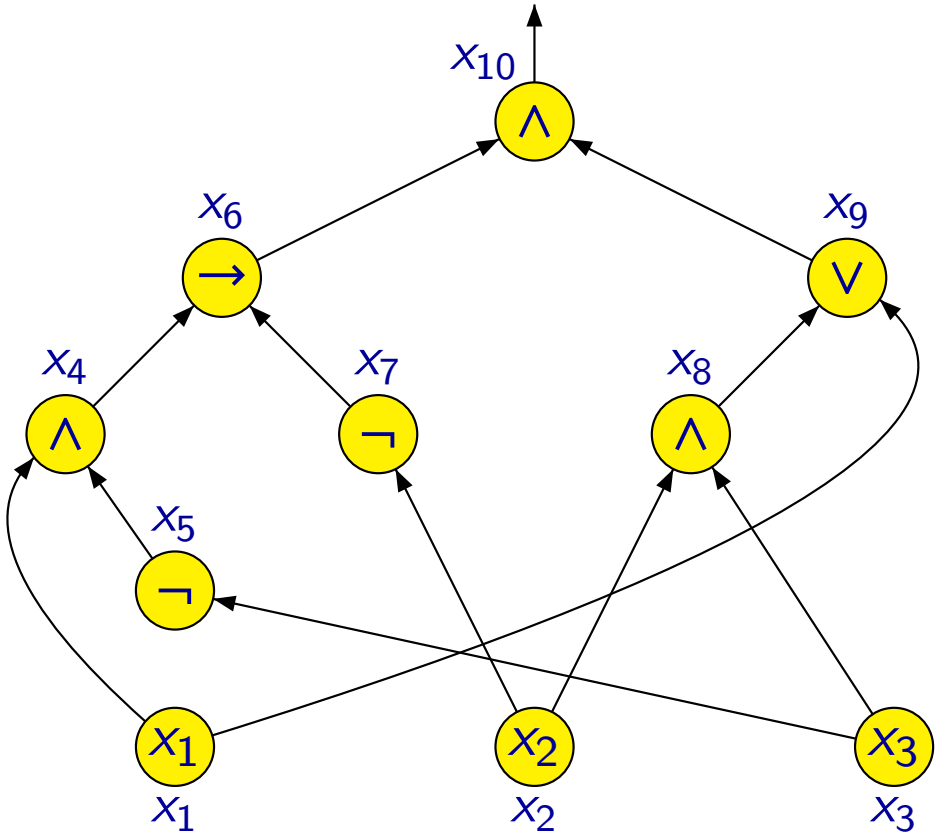
Reduction from SAT to 3-SAT

Now we go to the construction of the fomula φ_1 that can be created as a conjunction of the following formulas:

- For each gate, we add one corresponding formula constructed as discussed above.
- We add the formula x_{out} where x_{out} is a variable representing the output of the circuit.

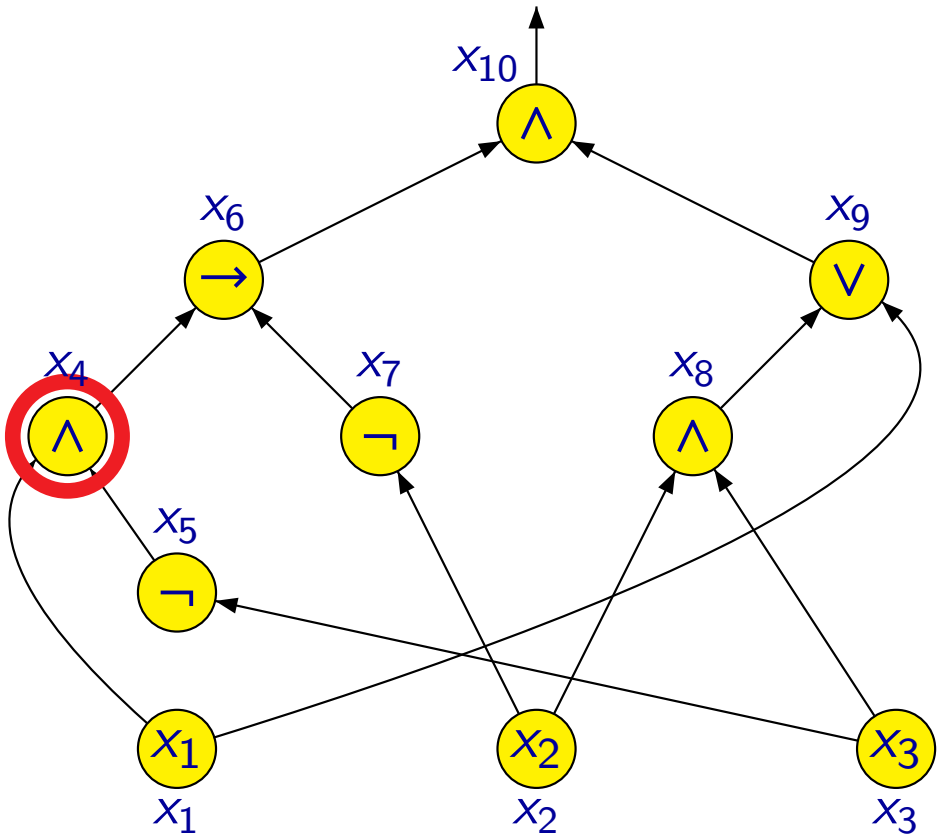
Reduction from SAT to 3-SAT

Example:



Reduction from SAT to 3-SAT

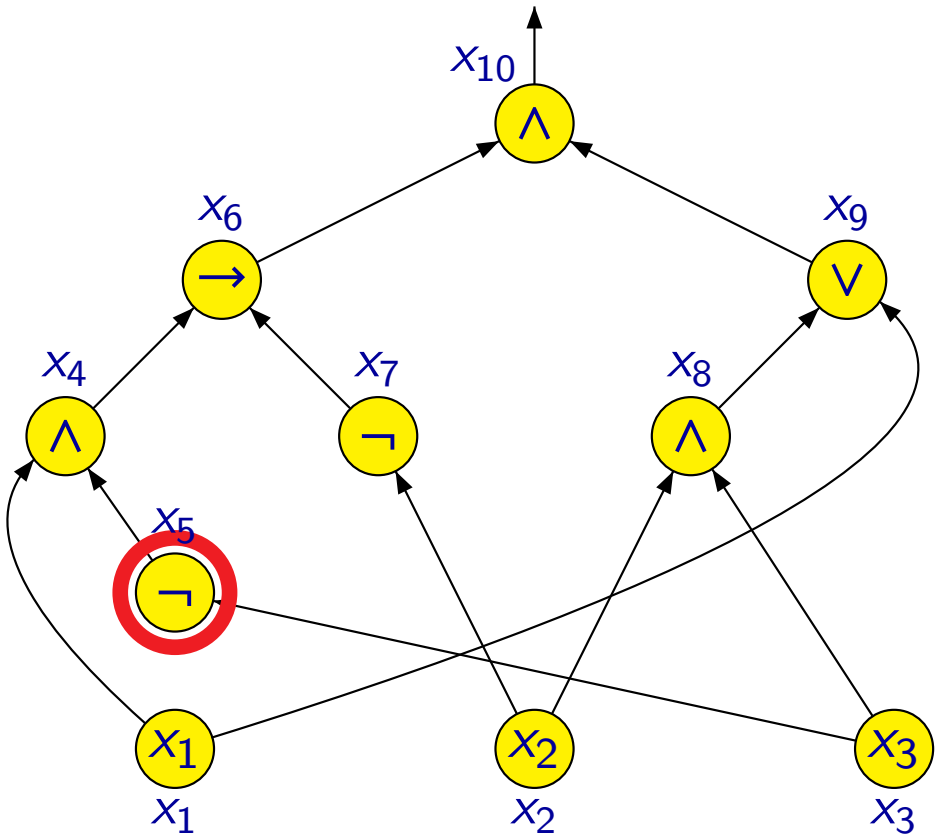
Example:



For x_4 we add the following clauses to the formula φ_1:
$(x_1 \vee x_5 \vee \neg x_4)$, $(x_1 \vee \neg x_5 \vee \neg x_4)$, $(\neg x_1 \vee x_5 \vee \neg x_4)$, $(\neg x_1 \vee \neg x_5 \vee x_4)$

Reduction from SAT to 3-SAT

Example:

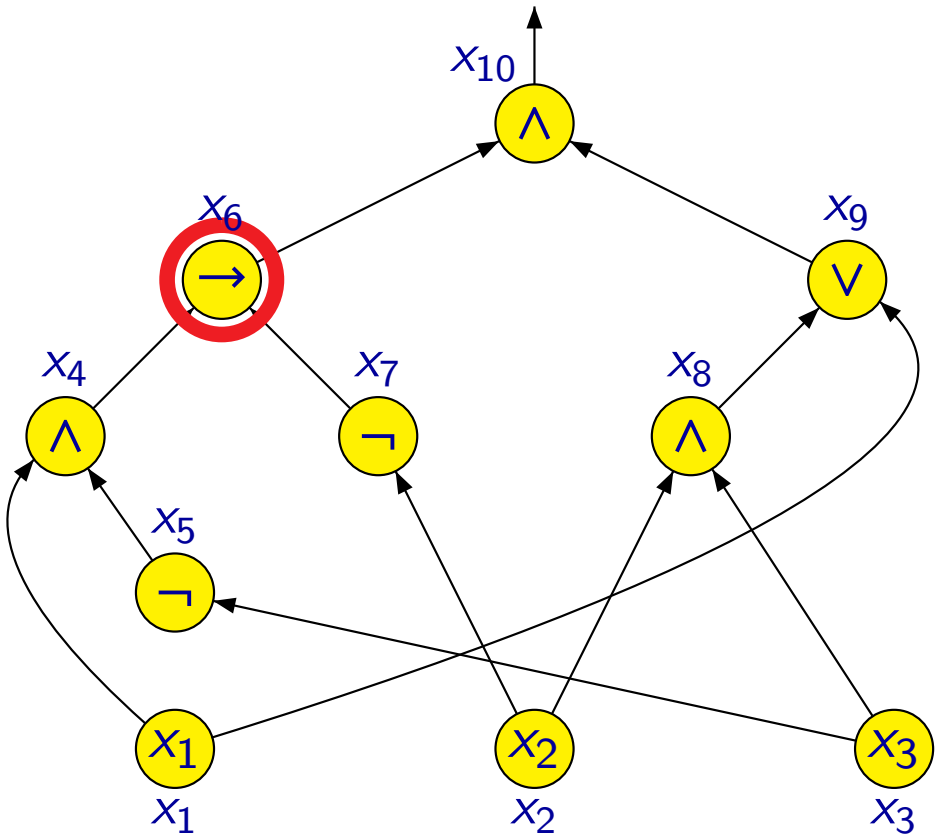


For x_5 the following clauses will be added to the formula φ_1:

$$(x_3 \vee x_5), (\neg x_3 \vee \neg x_5)$$

Reduction from SAT to 3-SAT

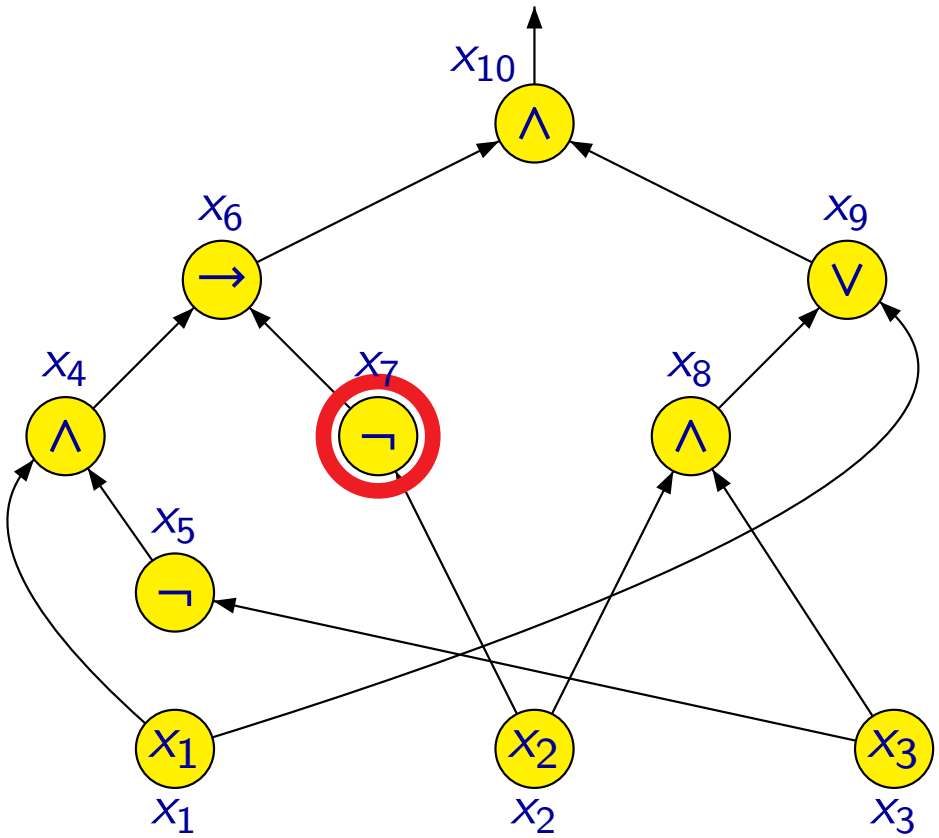
Example:



For x_6 we add the following clauses to the formula φ_1:
$(x_4 \vee x_7 \vee x_6)$, $(x_4 \vee \neg x_7 \vee x_6)$, $(\neg x_4 \vee x_7 \vee \neg x_6)$, $(\neg x_4 \vee \neg x_7 \vee x_6)$

Reduction from SAT to 3-SAT

Example:

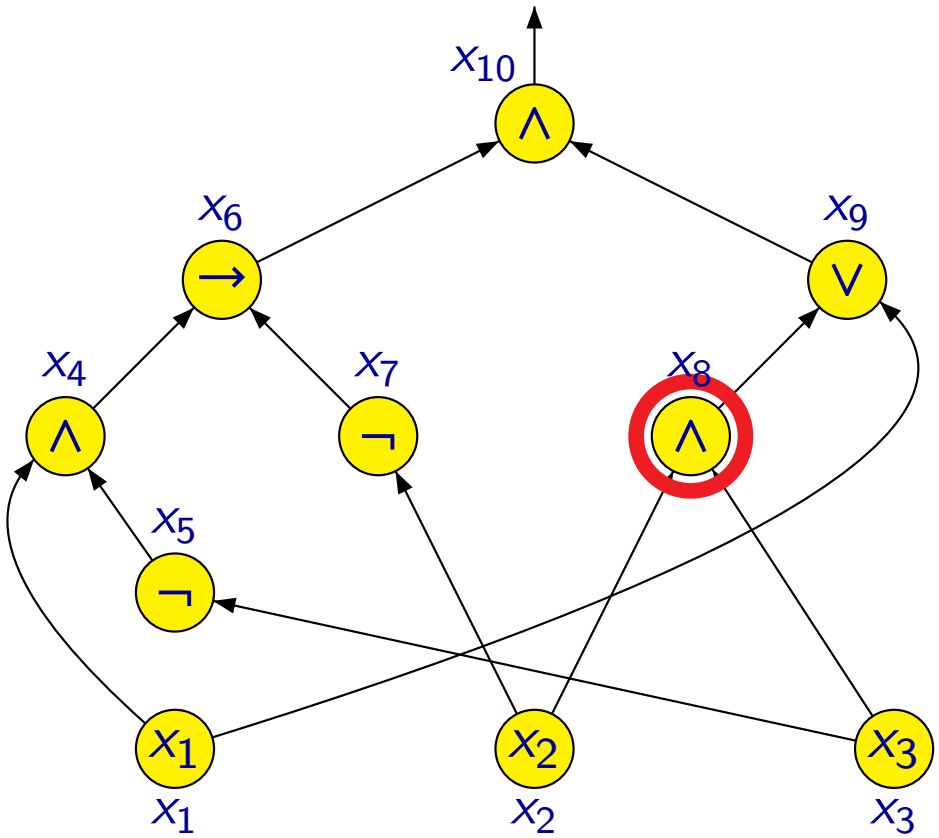


For x_7 the following clauses will be added to the formula φ_1:

$$(x_2 \vee x_7), (\neg x_2 \vee \neg x_7)$$

Reduction from SAT to 3-SAT

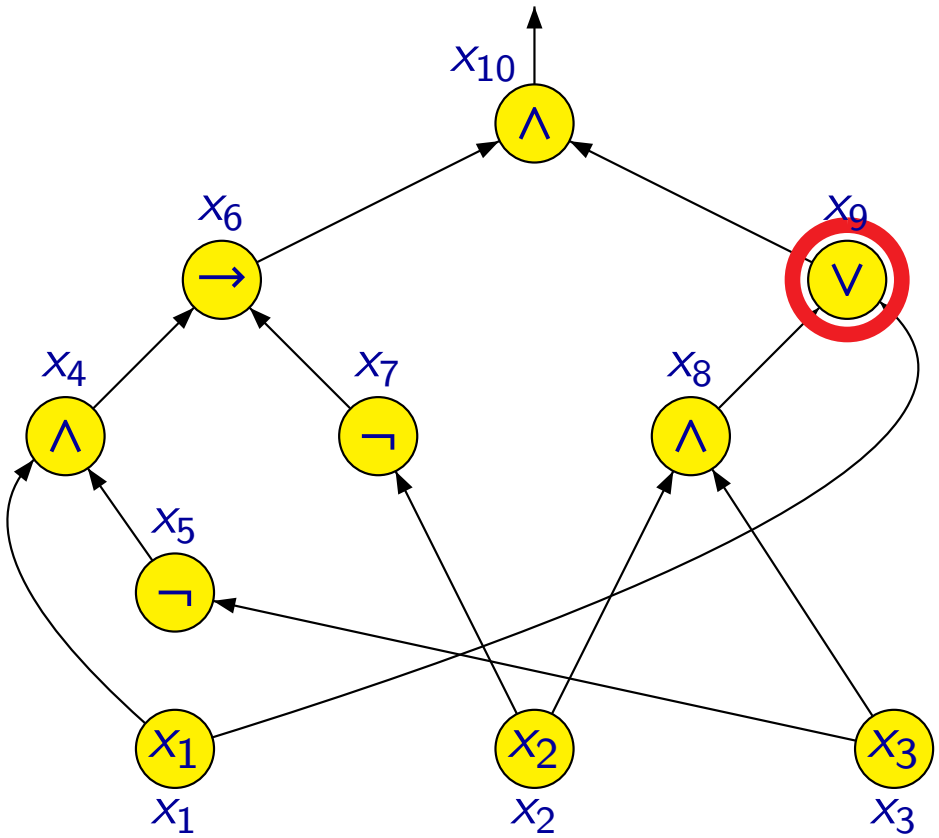
Example:



For x_8 we add the following clauses to the formula φ_1:
$(x_2 \vee x_3 \vee \neg x_8)$, $(x_2 \vee \neg x_3 \vee \neg x_8)$, $(\neg x_2 \vee x_3 \vee \neg x_8)$, $(\neg x_2 \vee \neg x_3 \vee x_8)$

Reduction from SAT to 3-SAT

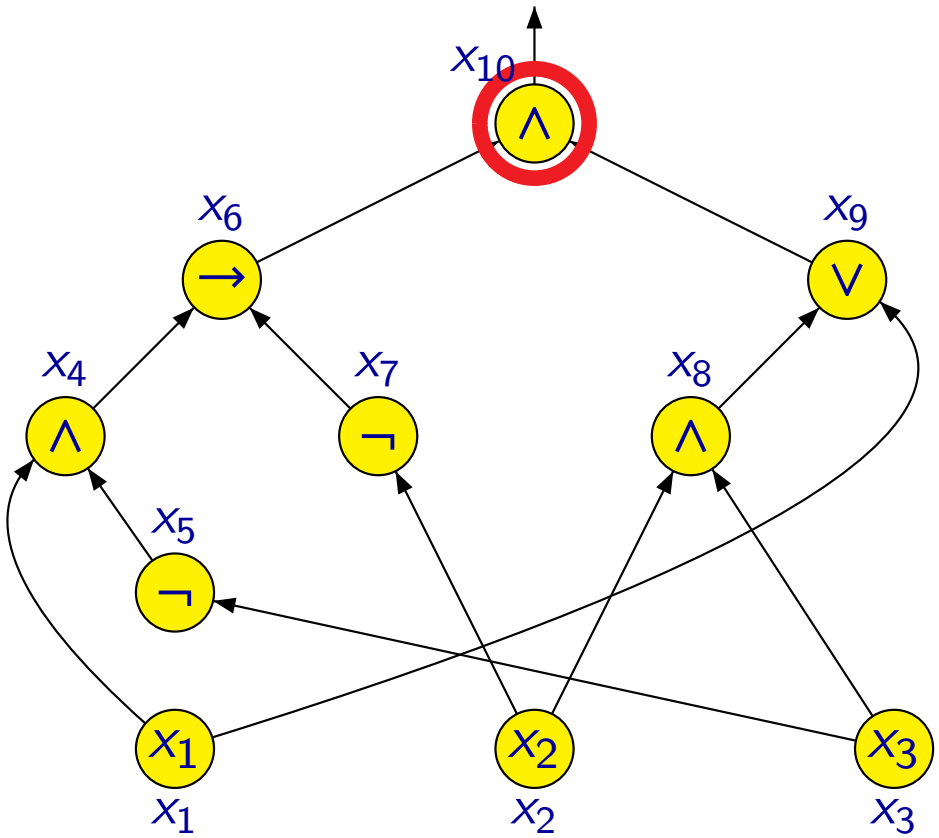
Example:



For x_9 we add the following clauses to the formula φ_1:
$(x_8 \vee x_1 \vee \neg x_9)$, $(x_8 \vee \neg x_1 \vee x_9)$, $(\neg x_8 \vee x_1 \vee x_9)$, $(\neg x_8 \vee \neg x_1 \vee x_9)$

Reduction from SAT to 3-SAT

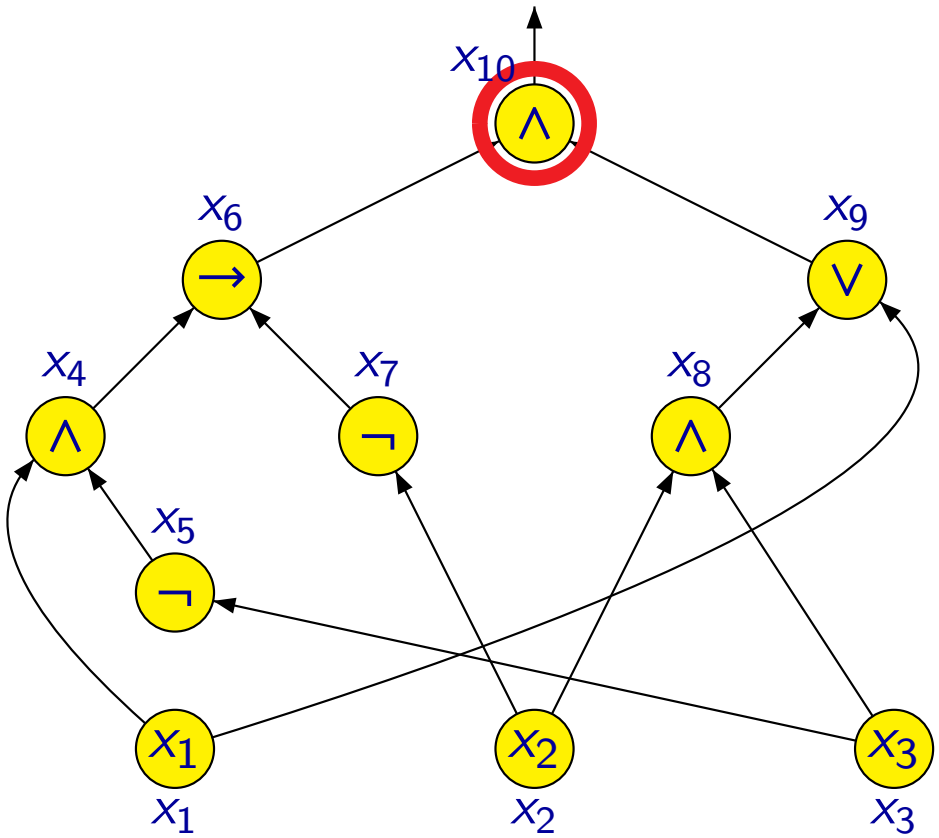
Example:



For x_{10} we add the following clauses to the formula φ_1:
$(x_6 \vee x_9 \vee \neg x_{10})$, $(x_6 \vee \neg x_9 \vee \neg x_{10})$, $(\neg x_6 \vee x_9 \vee \neg x_{10})$, $(\neg x_6 \vee \neg x_9 \vee x_{10})$

Reduction from SAT to 3-SAT

Example:



At the end, we add a clause representing the value on the output:

$$(x_{10})$$

Reduction from SAT to 3-SAT

The whole formula φ_1 now looks as follows:

$(x_1 \vee x_5 \vee \neg x_4) \wedge (x_1 \vee \neg x_5 \vee \neg x_4) \wedge (\neg x_1 \vee x_5 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_5 \vee x_4) \wedge$
$(x_3 \vee x_5) \wedge (\neg x_3 \vee \neg x_5) \wedge$
$(x_4 \vee x_7 \vee x_6) \wedge (x_4 \vee \neg x_7 \vee x_6) \wedge (\neg x_4 \vee x_7 \vee \neg x_6) \wedge (\neg x_4 \vee \neg x_7 \vee x_6) \wedge$
$(x_2 \vee x_7) \wedge (\neg x_2 \vee \neg x_7) \wedge$
$(x_2 \vee x_3 \vee \neg x_8) \wedge (x_2 \vee \neg x_3 \vee \neg x_8) \wedge (\neg x_2 \vee x_3 \vee \neg x_8) \wedge (\neg x_2 \vee \neg x_3 \vee x_8) \wedge$
$(x_8 \vee x_1 \vee \neg x_9) \wedge (x_8 \vee \neg x_1 \vee x_9) \wedge (\neg x_8 \vee x_1 \vee x_9) \wedge (\neg x_8 \vee \neg x_1 \vee x_9) \wedge$
$(x_6 \vee x_9 \vee \neg x_{10}) \wedge (x_6 \vee \neg x_9 \vee \neg x_{10}) \wedge (\neg x_6 \vee x_9 \vee \neg x_{10}) \wedge (\neg x_6 \vee \neg x_9 \vee x_{10}) \wedge$
(x_{10})

Reduction from SAT to 3-SAT

Now we will check that φ_1 is satisfiable iff φ is satisfiable.

Reduction from SAT to 3-SAT

At first, let us assume that φ is satisfiable.

So there exists a valuation ν such that $[\varphi]_\nu = 1$. Let us define valuation ν' as follows:

- $\nu'(x_i) = \nu(x_i)$ if x_i is a variable in formula φ
- If x_i represent an output of a gate, then we set $\nu'(x_i)$ to the value that will be on this output in valuation ν.

Since $[\varphi]_\nu = 1$, there must be the case that $\nu'(x_{out}) = 1$.

So it is obvious that $[\varphi_1]_{\nu'} = 1$ because x_{out} and all other clauses corresponding to individual gates will have the value 1 in the valuation ν'.

Reduction from SAT to 3-SAT

Let us assume now that φ_1 is satisfiable,
i.e., $[\varphi_1]_{\nu'} = 1$ for some valuation ν'.

It is easy to verify that $[\varphi]_{\nu'} = 1$ because ν' must correspond to some assignment of values on the outputs of individual gates where on the output of the whole circuit is the value 1.

By this we have verified that the described construction of the formula φ_1 is correct.

Reduction from SAT to 3-SAT

Now we construct, for a formula φ_1, a formula φ' such that:

- φ' will be in CNF,
- every clause of formule φ' will contain exactly 3 literals,
- no variable will occcur in some clause of formula φ' more than once,
- φ' will be satisfiable iff φ_1 is satisfiable.

Reduction from SAT to 3-SAT

At first, we get rid of redundant literals and clauses:

- If some literal occurs in some clause more than once, we can remove all its occurrences except one.
- If some clause contains at the same time literals x_i and $\neg x_i$ (where x_i is a variable), we can remove the whole clause
(such clause will have the value 1 in every valuation).

It is obvious that the modified formula is equivalent to the original formula.

Reduction from SAT to 3-SAT

We add two new variables y and z.

- Clauses with three literals will be left as they are.
- Every clause of the form $(A \vee B)$ (i.e., a clause with two literals A and B) will be replaced with the following pair of clauses:

$$(A \vee B \vee y) \ \wedge \ (A \vee B \vee \neg y)$$

- Every clause of the form (A) (i.e., a clause with one literal) will be replace the four following clauses:

$$(A \vee y \vee z) \ \wedge \ (A \vee y \vee \neg z) \ \wedge \ (A \vee \neg y \vee z) \ \wedge \ (A \vee \neg y \vee \neg z)$$

It is not difficult to check that the resulting formula φ' is satisfiable iff the original formula was satisfiable.

Reduction from SAT to 3-SAT

If the size of formula φ is n, the sizes of formulas φ_1 and φ' will be in $\mathcal{O}(n)$.

The formulas φ_1 and φ' can be easily construct in time $\mathcal{O}(n)$.

So the described reduction is a polynomial reduction.