Computational complexity

lecture 3

Announcement

Mid-term exam:

12.12.2017, during the lecture (Tuesday, 12:15)

Universal machines

Theorem:

There exists a universal Turing machine U (an “interpreter”), such that $U(\langle M\rangle,w)=M(w)$. If M works in time $T(|w|)$ and space $S(|w|)$, then U works in time $O(T(|w|)\cdot log(T(|w|)))$ and space $O(S(|w|))$.

Universal machines

Theorem:

There exists a universal Turing machine U (an “interpreter”), such that $U(\langle M\rangle,w)=M(w)$. If M works in time $T(|w|)$ and space $S(|w|)$, then U works in time $O(T(|w|)\cdot log(T(|w|)))$ and space $O(S(|w|))$.

Two possible definitions of time / space complexity:

- T_1/S_1 using machines (“there exists a machine...”)
- T_2/S_2 using programs for the universal machine (“there exists a program...”)

Relation between them:

- $T_1 \leq T_2 \leq T_1 \cdot log\ T_1$
- $S_1=S_2$

only small difference!
we use the definition with machines

Hierarchy theorems

Are there problems, which require very large time / space to be solved?
(Maybe every problem can be solved e.g. in polynomial time?)

Hierarchy theorems

Are there problems, which require very large time / space to be solved?
(Maybe every problem can be solved e.g. in polynomial time?)

Space hierarchy theorem:

If:

- function $g(n)$ is space-constructible, and
- $f(n)=o(g(n))$

then DSPACE($f(n)$)≠DSPACE($g(n)$)

Time hierarchy theorem – similar

definition: $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

Hierarchy theorems

Space hierarchy theorem:

If:

- function $g(n)$ is space-constructible, and
- $f(n)=o(g(n))$

then DSPACE($f(n)$)≠DSPACE($g(n)$)

Proof:

- Consider the language

$$L = \{(\langle M\rangle,w) \mid \text{tape alphabet of } M \text{ is } \{0,1,\triangleright,\perp\}, \text{ and } |\langle M\rangle| \leq g(|(\langle M\rangle,w)|), \text{ and } M \text{ rejects } (\langle M\rangle,w) \text{ in space } g(|(\langle M\rangle,w)|)\}$$

Hierarchy theorems

$L = \{(\langle M\rangle,w)$ | tape alphabet of M is $\{0,1,\triangleright,\perp\}$, and $|\langle M\rangle|\leq g(|(\langle M\rangle,w)|)$, and M rejects $(\langle M\rangle,w)$ in space $g(|(\langle M\rangle,w)|)\}$

<u>Part 1</u> – $L\notin$DSPACE($f(n)$)

Suppose that $L\in$DSPACE($f(n)$). Then there is M with tape alphabet $\{0,1,\triangleright,\perp\}$, which recognizes L in space $O(f(n))$.

Because $f(n)=o(g(n))$, for some long word w machine M works on $(\langle M\rangle,w)$ in space $g(|(\langle M\rangle,w)|)$, and $|\langle M\rangle|\leq g(|(\langle M\rangle,w)|)$

We have a contradiction:

(M accepts $(\langle M\rangle,w)$) $\Leftrightarrow$ $(\langle M\rangle,w)\in L$ $\Leftrightarrow$ (M rejects $(\langle M\rangle,w)$)

Remark – for the language

$L' = \{((\langle M\rangle,w)$ | M rejects $(\langle M\rangle,w)\}$

the same argument gives undecidability.

Hierarchy theorems

$L = \{(\langle M\rangle,w)$ | tape alphabet of M is $\{0,1,\triangleright,\perp\}$, and $|\langle M\rangle| \leq g(|(\langle M\rangle,w)|)$, and M rejects $(\langle M\rangle,w)$ in space $g(|(\langle M\rangle,w)|)\}$

Part 2: $L \in$ DSPACE($g(n)$) – i.e., L can be recognized in space $O(g(n))$.

- Generally: simulate the run of M on $(\langle M\rangle,w)$

Hierarchy theorems

$L = \{(\langle M \rangle, w)$ | tape alphabet of M is $\{0,1,\triangleright,\perp\}$, and $|\langle M \rangle| \leq g(|(\langle M \rangle, w)|)$, and M rejects $(\langle M \rangle, w)$ in space $g(|(\langle M \rangle, w)|)\}$

Part 2: $L \in$ DSPACE($g(n)$) – i.e., L can be recognized in space $O(g(n))$.

- Generally: simulate the run of M on $(\langle M \rangle, w)$
- Reserve working space $g(n)$ (where n = length of input)
  - space $O(g(n))$ is enough (by assumption g is space-constructible)

Hierarchy theorems

$L = \{(\langle M\rangle,w)$ | tape alphabet of M is $\{0,1,\triangleright,\perp\}$, and $|\langle M\rangle|\leq g(|(\langle M\rangle,w)|)$, and M rejects $(\langle M\rangle,w)$ in space $g(|(\langle M\rangle,w)|)\}$

<u>Part 2:</u> $L\in$DSPACE($g(n)$) – i.e., L can be recognized in space $O(g(n))$.

- Generally: simulate the run of M on $(\langle M\rangle,w)$
- Reserve working space $g(n)$ (where n = length of input)
  - space $O(g(n))$ is enough (by assumption g is space-constructible)
- Check that the input is of the form $(\langle M\rangle,w)$, that the alphabet is $\{0,1,\triangleright,\perp\}$, and that $|\langle M\rangle|\leq g(n)$
  - space $O(g(n))$ is enough

Hierarchy theorems

$L = \{(\langle M\rangle,w)$ | tape alphabet of M is $\{0,1,\triangleright,\perp\}$, and $|\langle M\rangle|\leq g(|(\langle M\rangle,w)|)$, and M rejects $(\langle M\rangle,w)$ in space $g(|(\langle M\rangle,w)|)\}$

<u>Part 2:</u> $L\in$DSPACE($g(n)$) – i.e., L can be recognized in space $O(g(n))$.

- Generally: simulate the run of M on $(\langle M\rangle,w)$
- Reserve working space $g(n)$ (where n = length of input)
  - space $O(g(n))$ is enough (by assumption g is space-constructible)
- Check that the input is of the form $(\langle M\rangle,w)$, that the alphabet is $\{0,1,\triangleright,\perp\}$, and that $|\langle M\rangle|\leq g(n)$
  - space $O(g(n))$ is enough
- Use the Sipser's theorem (or assume that $g(n)=\Omega(log(n))$, and use the approach with a counter), and check whether M rejects $(\langle M\rangle,w)$ in reserved space $g(n)$.
  - when M rejects → we accept
  - when M accepts or loops or exceeds space → we reject
  - space $O(g(n))$ is enough

Hierarchy theorems

Space hierarchy theorem:

If:

- function $g(n)$ is space-constructible, and
- $f(n)=o(g(n))$

then DSPACE($f(n)$)≠DSPACE($g(n)$)

Time hierarchy theorem:

If:

- function $g(n)$ is time-constructible,
- $f(n)=o(g(n))$

then DTIME($f(n)$)≠DTIME($g(n)log(g(n))$)

Hierarchy theorems

Time hierarchy theorem:

If:

- function $g(n)$ is time-constructible,
- $f(n)=o(g(n))$

then DTIME($f(n)$)$\neq$DTIME($g(n)log(g(n))$)

Proof

- Consider the language

$$L = \{(\langle M\rangle,w) \mid \text{ tape alphabet of } M \text{ is } \{0,1,\triangleright,\perp\}\text{, and } |\langle M\rangle| \leq log(|(\langle M\rangle,w)|)$$
$$\text{and } M \text{ rejects } (\langle M\rangle,w) \text{ in time } g(|(\langle M\rangle,w)|)\}$$

- Part 1 – $L \notin$DTIME($f(n)$) → exactly as previously

Hierarchy theorems

$L = \{(\langle M\rangle,w) \mid$ tape alphabet of M is $\{0,1,\triangleright,\perp\}$, and $|\langle M\rangle| \leq log(|(\langle M\rangle,w)|)$ and M rejects $(\langle M\rangle,w)$ in time $g(|(\langle M\rangle,w)|)\}$

<u>Part 2</u> – $L\in$DTIME($g(n)log(g(n))$) – i.e., L can be recognized in time $O(g(n)log(g(n)))$

- Generally: simulate the run of M on $(\langle M\rangle,w)$
- Check that the input is of the form $(\langle M\rangle,w)$, that the alphabet is $\{0,1,\triangleright,\perp\}$, and that $|\langle M\rangle| \leq log(n)$ (where n = length of input)
  - running time: $O(n)$
- Reserve a unary counter of length $g(n)$, on a separate tape
  - g is time constructible
  - running time: $O(g(n))$
- Simulate M on word $(\langle M\rangle,w)$, like the universal machine; increase the counter after every step.
  - running time: $O(g(n)\cdot(log\ g(n)+|\langle M\rangle|)) = O(g(n)log(g(n)))$

simulating tapes (↑ $log\ g(n)$)

reading the description of M, modifying state (↑ $|\langle M\rangle|$)

Hierarchy theorems

$L = \{(\langle M\rangle,w) \mid$ tape alphabet of M is $\{0,1,\triangleright,\perp\}$, and $|\langle M\rangle| \leq log(|(\langle M\rangle,w)|)$ and M rejects $(\langle M\rangle,w)$ in time $g(|(\langle M\rangle,w)|)\}$

<u>Part 2</u> – $L \in$ DTIME($g(n)log(g(n))$) – i.e., L can be recognized in time $O(g(n)log(g(n)))$

- Generally: simulate the run of M on $(\langle M\rangle,w)$
- Check that the input is of the form $(\langle M\rangle,w)$, that the alphabet is $\{0,1,\triangleright,\perp\}$, and that $|\langle M\rangle| \leq log(n)$ (where n = length of input)
  - running time: $O(n)$
- Reserve a unary counter of length $g(n)$, on a separate tape
  - g is time constructible
  - running time: $O(g(n))$
- Simulate M on word $(\langle M\rangle,w)$, like the universal machine; increase the counter after every step.
  - running time: $O(g(n) \cdot (log\ g(n) + |\langle M\rangle|)) = O(g(n)log(g(n)))$
  - when M rejects → we accept
  - when M accepts or exceeds time → we reject

Hierarchy theorems

Are there problems, which require very large time / space to be solved?
(Maybe every problem can be solved e.g. in polynomial time?)

Corollary from hierarchy theorems

- DTIME(n^k)≠DTIME(n^{k+1}), DSPACE(n^k)≠DSPACE(n^{k+1})
- L≠PSPACE, P≠EXPTIME

because P⊆DTIME(2^n)≠DTIME(4^n)⊆EXPTIME

Hierarchy theorems

Are there problems, which require very large time / space to be solved?
(Maybe every problem can be solved e.g. in polynomial time?)

Corollary from hierarchy theorems

- DTIME(n^k)≠DTIME(n^{k+1}), DSPACE(n^k)≠DSPACE(n^{k+1})
- L≠PSPACE, P≠EXPTIME

because P⊆DTIME(2^n)≠DTIME(4^n)⊆EXPTIME

If a machine M works in time / space precisely $f(n)$, then there exists a problem requiring more time / space to be solved

- e.g. $2^{f(n)}$ *or* $f(n)^2$ – for time & space
- e.g. $f(n) \cdot log(log(n))$ – for space
- Moreover, functions being complexities of problems are distributed "quite densely", especially for space

Gap theorems

- Functions being complexities of problems are distributed "quite densely"
- Simultaneously, we have the following gap theorems:

There is a computable function $f(n) \geq n$ such that DTIME($f(n)$)=DTIME($2^{f(n)}$).

There is a computable function $f(n)$ such that DSPACE($f(n)$)
=DSPACE($2^{f(n)}$).

A contradiction with hierarchy theorems?

No – the function f will not be constructible (it can be computed, but in a larger time / space)

At the same time: we see that in the hierarchy theorems the assumption about constructability is really needed

Gap theorems (✶)

<u>Gap theorem</u> – time

There is a computable function $f(n) \geq n$ such that DTIME($f(n)$)=DTIME($2^{f(n)}$).

<u>Proof</u>

Fix an input alphabet $\Sigma = \{0,1\}$ (another alphabet → time multiplied by a constant)

We construct a function $f(n)$ such that no machine stops between $f(n)$ and $2^{f(n)}$ steps:

- Assign numbers to Turing machines (in a computable way)

Gap theorems (∗)

Gap theorem – time

There is a computable function $f(n) \geq n$ such that DTIME($f(n)$)=DTIME($2^{f(n)}$).

Proof

Fix an input alphabet $\Sigma=\{0,1\}$ (another alphabet → time multiplied by a constant)

We construct a function $f(n)$ such that no machine stops between $f(n)$ and $2^{f(n)}$ steps:

- Assign numbers to Turing machines (in a computable way)
- We say that $P(i,k)$ is satisfied iff none among the first i machines on none among inputs of length i stops between k and $i \cdot 2^k$ steps (they stop earlier than k or later than $i \cdot 2^k$ or loop forever)

Gap theorems (∗)

Gap theorem – time

There is a computable function $f(n) \geq n$ such that DTIME($f(n)$)=DTIME($2^{f(n)}$).

Proof

Fix an input alphabet $\Sigma=\{0,1\}$ (another alphabet → time multiplied by a constant)

We construct a function $f(n)$ such that no machine stops between $f(n)$ and $2^{f(n)}$ steps:

- Assign numbers to Turing machines (in a computable way)
- We say that $P(i,k)$ is satisfied iff none among the first i machines on none among inputs of length i stops between k and $i \cdot 2^k$ steps (they stop earlier than k or later than $i \cdot 2^k$ or loop forever)
- Let $k_1(i)=i$ and $k_{j+1}(i)=i \cdot 2^{k_j(i)}$
- For a fixed i, every pair (input_of_length_i, machine_with_number_$\leq i$) can falsify $P(i,k_j(i))$ for at most one j,

  Thus there exists some $j \leq i \cdot 2^i$ such that $P(i,k_j(i))$ is true.

Gap theorems (∗)

Gap theorem – time

There is a computable function $f(n) \geq n$ such that DTIME($f(n)$)=DTIME($2^{f(n)}$).

Proof

Fix an input alphabet $\Sigma = \{0,1\}$ (another alphabet → time multiplied by a constant)

We construct a function $f(n)$ such that no machine stops between $f(n)$ and $2^{f(n)}$ steps:

- Assign numbers to Turing machines (in a computable way)
- We say that $P(i,k)$ is satisfied iff none among the first i machines on none among inputs of length i stops between k and $i \cdot 2^k$ steps (they stop earlier than k or later than $i \cdot 2^k$ or loop forever)
- Let $k_1(i)=i$ and $k_{j+1}(i)=i \cdot 2^{k_j(i)}$
- For a fixed i, every pair (input_of_length_i, machine_with_number_$\leq i$) can falsify $P(i,k_j(i))$ for at most one j,

  Thus there exists some $j \leq i \cdot 2^i$ such that $P(i,k_j(i))$ is true.
- We put $f(i)=k_j(i)$. This function is computable.

Gap theorems (✶)

Gap theorem – time

There is a computable function $f(n) \geq n$ such that DTIME($f(n)$)=DTIME($2^{f(n)}$).

Proof

- For every n, none among the first n machines on none among inputs of length n stops between $f(n)$ and $n \cdot 2^{f(n)}$ steps.
- Take any machine M with number m running in time $c \cdot 2^{f(n)}$
- For every input of length $n \geq max(m,c)$ the machine stops in $\leq c \cdot 2^{f(n)}$ steps, but not between $f(n)$ and $n \cdot 2^{f(n)}$ steps, hence in $\leq f(n)$ steps

Gap theorems (∗)

Gap theorem – time

There is a computable function $f(n) \geq n$ such that DTIME($f(n)$)=DTIME($2^{f(n)}$).

Proof

- For every n, none among the first n machines on none among inputs of length n stops between $f(n)$ and $n \cdot 2^{f(n)}$ steps.
- Take any machine M with number m running in time $c \cdot 2^{f(n)}$
- For every input of length $n \geq max(m,c)$ the machine stops in $\leq c \cdot 2^{f(n)}$ steps, but not between $f(n)$ and $n \cdot 2^{f(n)}$ steps, hence in $\leq f(n)$ steps
- There are only constantly many inputs of length $< max(m,c)$
- Thus the language can be recognized in time $O(f(n))$

Gap theorems

<u>Remarks</u>

- In the same way we can construct a function f such that DSPACE($f(n)$)=DSPACE($2^{f(n)}$).
- Actually, for every function g such that $g(n)\geq n$ (instead of $g(n)=2^n$) we can find f a such that DTIME($f(n)$)=DTIME($g(f(n))$) or DSPACE($f(n)$)=DSPACE($g(f(n))$).
- The functions f grow very quickly.
- They are not time/space-constructible.
- But they are computable.

Just finished:

Deterministic Turing machines – basic facts

Next topic:

Boolean circuits

Later:

- Nondeterministic Turing machines, reductions
- Probabilistic computations
- Fixed parameter tractability (FPT)
- Interactive proofs
- Alternating Turing machines
- Probabilistically checkable proofs (PCP)
- ...

Nonuniform computation models

- Suppose that P≠NP. Then there is no algorithm which quickly solves all instances of the SAT problem.
- But maybe for every n there is a separate algorithm, which quickly solves all instances of size n?
- Even if these algorithms are difficult to find, this would mean that SAT can be solved in practice.

Nonuniform computation models

- Suppose that P≠NP. Then there is no algorithm which quickly solves all instances of the SAT problem.
- But maybe for every n there is a separate algorithm, which quickly solves all instances of size n?
- Even if these algorithms are difficult to find, this would mean that SAT can be solved in practice.
- A similar example: breaking the cryptographic algorithm RSA. If there is an algorithm, which quickly breaks the RSA encoding for a fixed (being currently used) key length, in practice we can treat the RSA code as insecure (even if the algorithm works only for one fixed n, not for all n).

Nonuniform computation models

- Suppose that P≠NP. Then there is no algorithm which quickly solves all instances of the SAT problem.
- But maybe for every n there is a separate algorithm, which quickly solves all instances of size n?
- Even if these algorithms are difficult to find, this would mean that SAT can be solved in practice.
- A similar example: breaking the cryptographic algorithm RSA. If there is an algorithm, which quickly breaks the RSA encoding for a fixed (being currently used) key length, in practice we can treat the RSA code as insecure (even if the algorithm works only for one fixed n, not for all n).

Hence, it makes sense to consider computation models in which for every n we apply a different algorithm.

One has to be careful, though: for every n, the language of instances of size n is regular.

Models of parallel computations

What if we have plenty of processors?

Example: matrix multiplication

- 1 processor: time $O(n^3)$ (the standard algorithm)
- n^2 processors: time $O(n)$
- n^3 processors: time $O(log(n))$ – an exponential speed up!

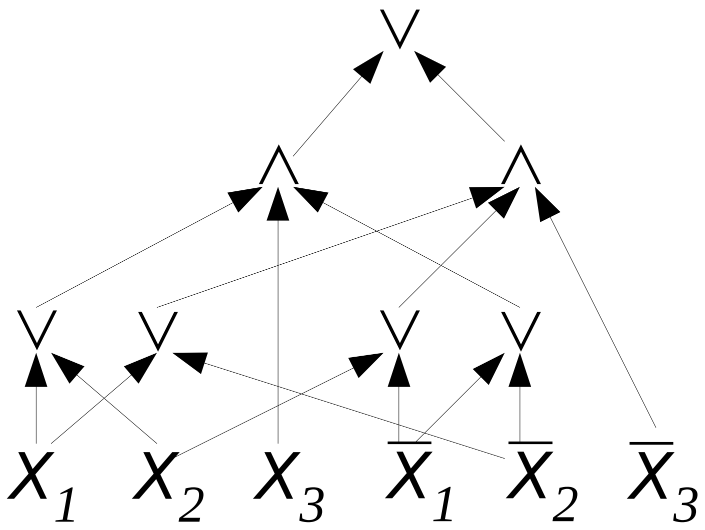
Question: Which algorithms do parallelize well, and which do not?

Boolean circuits

Another computational model: boolean circuits

idea: computing boolean functions using logical gates

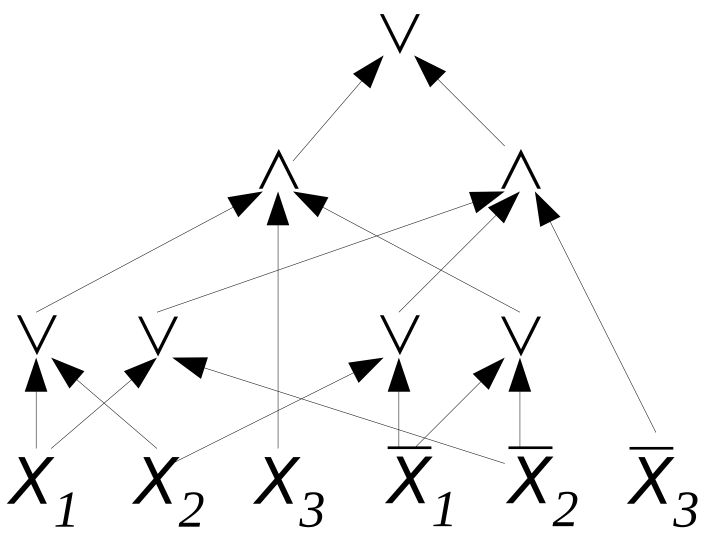
intuition: every gate represents a very simple processor



Boolean circuits

Definition: a boolean circuit having input of size n is given by an acyclic directed graph, in which:

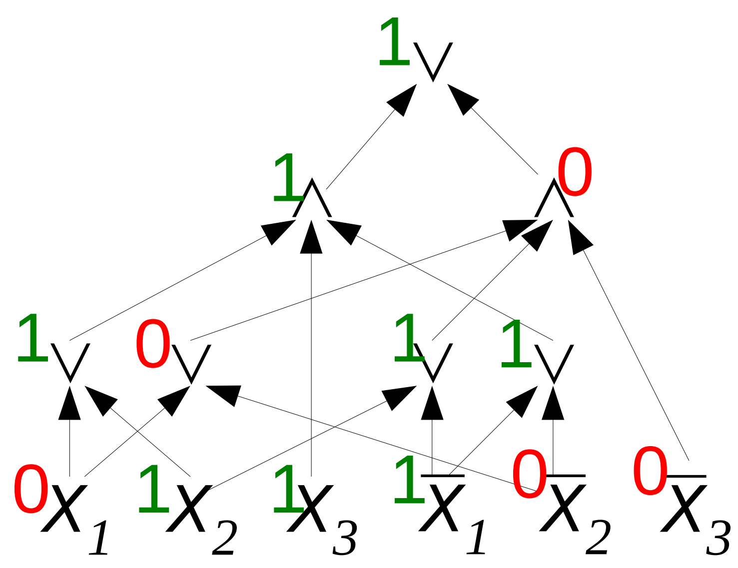
- there are $2n$ gates (nodes) of in-degree 0, denoted $X_1, \overline{X}_1, \ldots, X_n, \overline{X}_n$ (input gates)
- all other gates (having in-degree ≥ 0) are marked by one of the symbols $\wedge$ or $\vee$
- one of the gates (having out-degree 0) is marked as the output gate [another version: multiple outputs – when we compute a function]



Boolean circuits

For a fixed valuation $v:\{X_1,\ldots,X_n\}\rightarrow\{0,1\}$ we define:

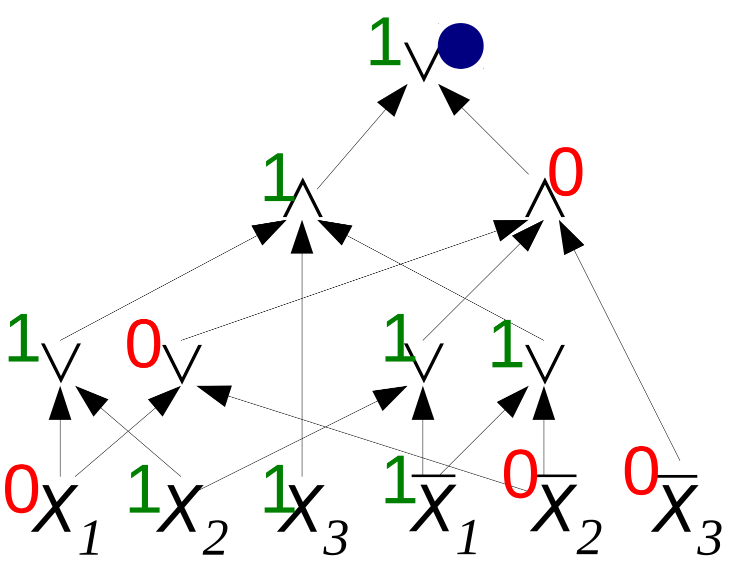
- the gate labeled by X_i gets value $v(X_i)$
- the gate labeled by $\overline{X}_i$ gets value $\neg v(X_i)$
- the value of an OR (AND) gate is computed as the disjunction (conjunction) of values of predecessors of the gate
- the value of the circuit = the value of the output gate
- the definition makes sense, because the graph is acyclic



Boolean circuits

An equivalent definition – a circuit as a game:

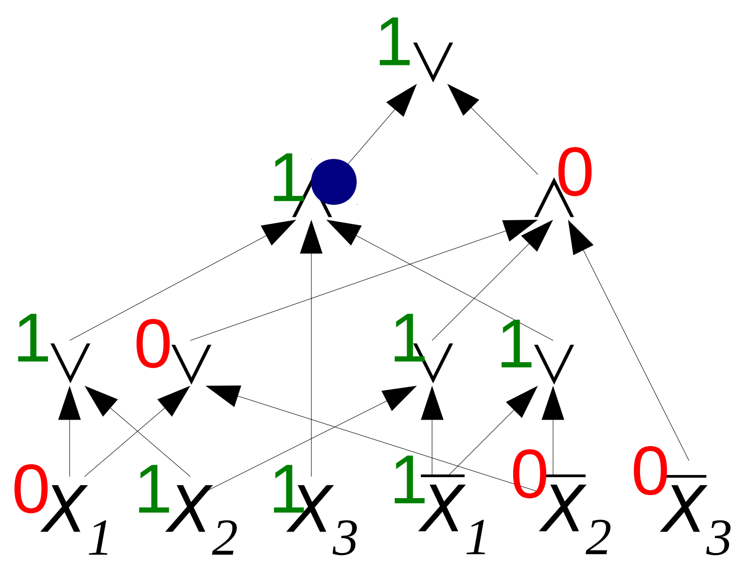
- two players (AND and OR) move a pawn over the graph, going back from the output gate
- AND (OR) decides in $\wedge$ nodes ($\vee$ nodes, respectively)
- OR wins, if the game finishes in X_i and $v(X_i)=1$, or in $\overline{X}_i$ and $v(X_i)=0$
- the value of the circuit is 1 if OR has a winning strategy



Boolean circuits

An equivalent definition – a circuit as a game:

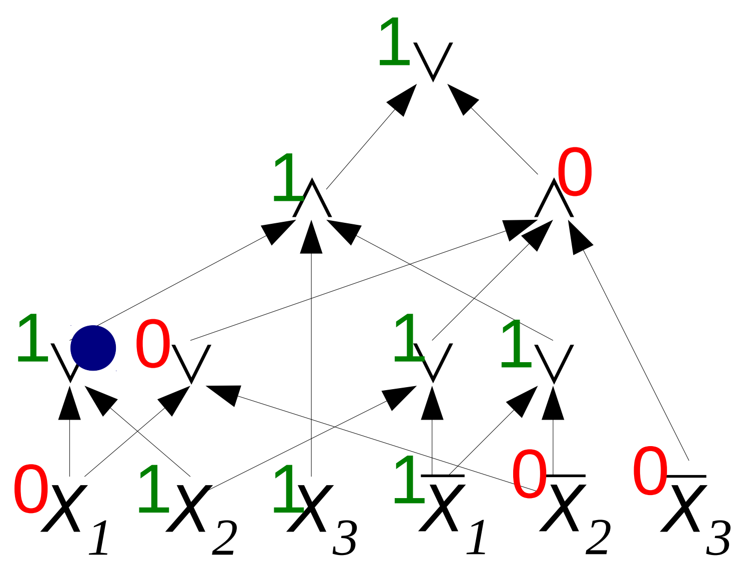
- two players (AND and OR) move a pawn over the graph, going back from the output gate
- AND (OR) decides in ∧ nodes (∨ nodes, respectively)
- OR wins, if the game finishes in X_i and $v(X_i)=1$, or in $\overline{X}_i$ and $v(X_i)=0$
- the value of the circuit is 1 if OR has a winning strategy



Boolean circuits

An equivalent definition – a circuit as a game:

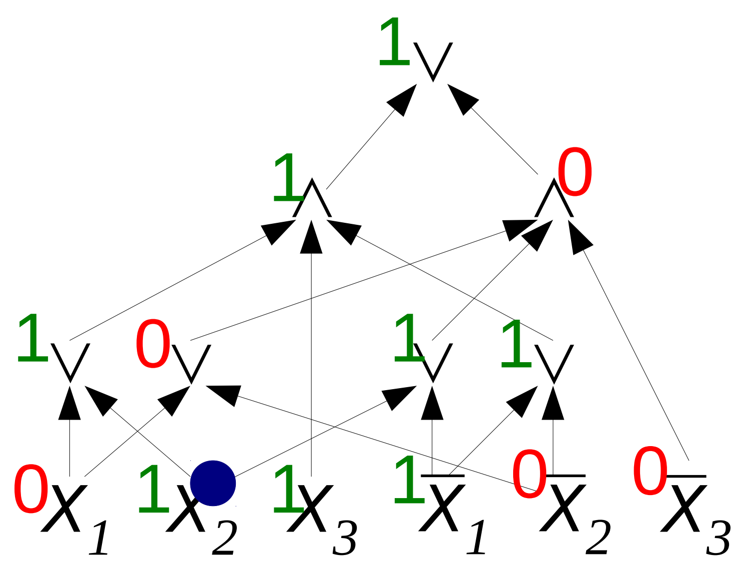
- two players (AND and OR) move a pawn over the graph, going back from the output gate
- AND (OR) decides in ∧ nodes (∨ nodes, respectively)
- OR wins, if the game finishes in X_i and $v(X_i)=1$, or in $\overline{X}_i$ and $v(X_i)=0$
- the value of the circuit is 1 if OR has a winning strategy



Boolean circuits

An equivalent definition – a circuit as a game:

- two players (AND and OR) move a pawn over the graph, going back from the output gate
- AND (OR) decides in $\wedge$ nodes ($\vee$ nodes, respectively)
- OR wins, if the game finishes in X_i and $v(X_i)=1$, or in $\overline{X}_i$ and $v(X_i)=0$
- the value of the circuit is 1 if OR has a winning strategy



Boolean circuits

Equivalence of the two definitions:

- if the output has value 1, we have a strategy for OR: descend always to a node labeled by 1
- if the output has value 0, we have a strategy for AND: descend always to a node labeled by 0

Boolean circuits

- For a fixed valuation $v:\{X_1,\ldots,X_n\}\to\{0,1\}$ we have defined the value of a circuit
- The input amounts to a word $v\in\{0,1\}^n$
- A circuit C computes a function $\{0,1\}^n\to\{0,1\}$, i.e., it recognizes a subset of $\{0,1\}^n$

Boolean circuits

<u>Size?</u>

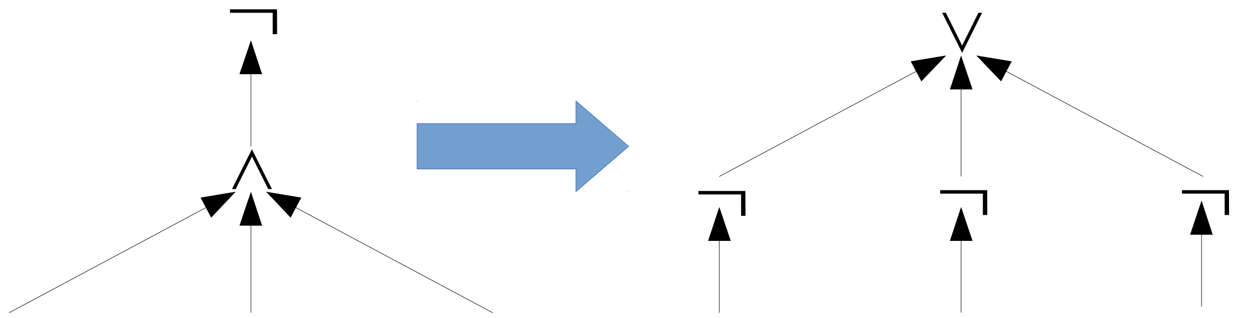
We have several parameters:

- the length of an input n
- the depth of a circuit (the length of the longest path)
- the number of gates B, the number of edges K
- the length of a representation of a circuit: $(B+K)\cdot log(B)$ (because numbers of gates have $log(B)$ bits)
- in-degree of gates (fan-in) – we consider circuits
  - ➔ with arbitrary fan-in
  - ➔ with fan-in ≤ 2

Boolean circuits

Negations?

- in our definition there are no NOT gates, but we have negated input gates
- this does not change anything: negations can be easily moved to leaves (De Morgan laws)



Boolean circuits

Recognizing languages by sequences of circuits:

- A circuit C_n having input of size n recognizes $L(C_n)$ – a subset of $\{0,1\}^n$ [in particular C_0 has no inputs, returns always 1 or always 0]
- Having a sequence of circuits $C_0,C_1,C_2,...$ we can recognize a language containing words of any length:

$$L((C_n)_{n\in\mathbb{N}})=L(C_0)\cup L(C_1)\cup L(C_2)\cup ...$$

- Which languages can be recognized using boolean circuits?

Boolean circuits

Recognizing languages by sequences of circuits:

- A circuit C_n having input of size n recognizes $L(C_n)$ – a subset of $\{0,1\}^n$ [in particular C_0 has no inputs, returns always 1 or always 0]
- Having a sequence of circuits $C_0,C_1,C_2,\ldots$ we can recognize a language containing words of any length:

$$L((C_n)_{n\in\mathbb{N}})=L(C_0)\cup L(C_1)\cup L(C_2)\cup\ldots$$

- Which languages can be recognized using boolean circuits?

Fact.
Every laguage can be recognized by some sequence of boolean circuits (having depth 2 and exponential size)

i.e., the size of C_n is exponential in n

Boolean circuits

<u>Recognizing languages by sequences of circuits:</u>

- A circuit C_n having input of size n recognizes $L(C_n)$ – a subset of $\{0,1\}^n$ [in particular C_0 has no inputs, returns always 1 or always 0]
- Having a sequence of circuits $C_0,C_1,C_2,...$ we can recognize a language containing words of any length:

$$L((C_n)_{n\in\mathbb{N}})=L(C_0)\cup L(C_1)\cup L(C_2)\cup ...$$

- Which languages can be recognized using boolean circuits?

<u>Fact</u>.
Every laguage can be recognized by some sequence of boolean circuits (having depth 2 and exponential size)

A more interesting question: Which languages can be recognized by a sequence of circuits of polynomial size?