Digital Logic Design: a rigorous approach ©

Chapter 12: Trees

part 2

Guy Even Moti Medina

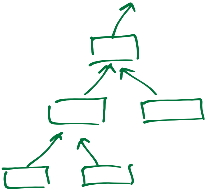
School of Electrical Engineering Tel-Aviv Univ.

May 3, 2020

Book Homepage:
http://www.eng.tau.ac.il/~guy/Even-Medina

Preliminary questions:

1. Which Boolean functions are suited for implementation by tree-like combinational circuits?
2. In what sense are tree-like implementations optimal?



cost?
delay?

Reminder: Binary Boolean Functions

Definition

A binary Boolean function is a function $f : \{0,1\}^2 \to \{0,1\}$.

A binary function is often denoted by a dyadic operator, say $*$. So instead of writing $f(a,b)$, we write $a * b$.

examples : OR, AND, XOR

$*$ - can be any bin. Boolean func.

Reminder: Associative Boolean functions

Definition

A binary Boolean function $*: \{0,1\}^2 \to \{0,1\}$ is associative if

$$(x_1 * x_2) * x_3 = x_1 * (x_2 * x_3) \ ,$$

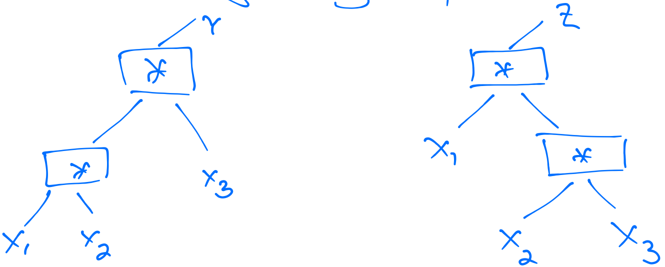
for every $x_1, x_2, x_3 \in \{0,1\}$.

One may omit parenthesis: $x_1 * x_2 * x_3$ is well defined.
Consider the function $f_n : \{0,1\}^n \to \{0,1\}$ defined by

$$f_n(x_1, \ldots, x_n) \triangleq x_1 * \cdots * x_n$$

example : $OR_n(x_1, \ldots, x_n) = x_1 + \cdots + x_n$

associativity by picture



$(x_1 * x_2) * x_3$ $\qquad\qquad$ $x_1 * (x_2 * x_3)$

$* \text{ assoc.} \iff Y = Z$

$(\forall\, x_1, x_2, x_3)$

Extension of associative function

Definition

Let $f : \{0,1\}^2 \rightarrow \{0,1\}$ denote a Boolean function. The function $f_n : \{0,1\}^n \rightarrow \{0,1\}$, for $n \geq 1$, is defined recursively as follows.

1. If $n = 1$, then $f_1(x) = x$.
2. If $n = 2$, then $f_2 = f$.
3. If $n > 2$, then f_n is defined based on f_{n-1} as follows:

$$f_n(x_1, x_2, \ldots x_n) \triangleq f(f_{n-1}(x_1, \ldots, x_{n-1}), x_n).$$

Claim

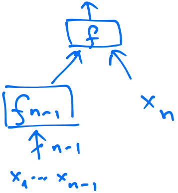
If $f : \{0,1\}^2 \rightarrow \{0,1\}$ is an associative Boolean function, then

$$f_n(x_1, x_2, \ldots x_n) = f(f_{n-k}(x_1, \ldots, x_{n-k}), f_k(x_{n-k+1}, \ldots, x_n)),$$

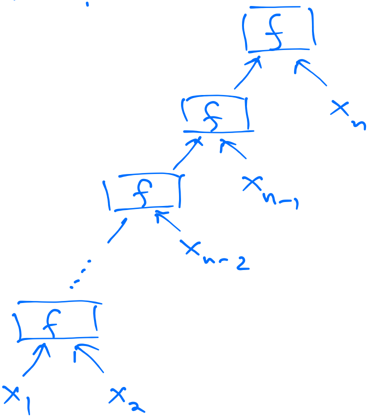
for every $n \geq 2$ and $k \in [1, n-1]$.

$$f_n(x_1,\dots,x_n)$$

$$= f(f_{n-1}(x_1,\dots,x_{n-1}), x_n)$$



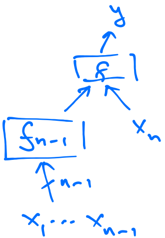
expand recursion:

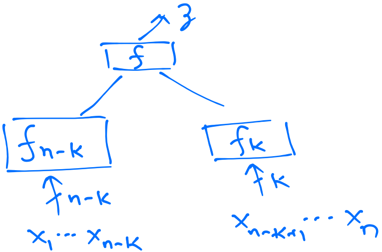


want to prove that $\forall n \geq 2 \ \forall k \in [1, n-1]$:

$$f_n(x_1, \dots, x_n) = f\left(f_{n-k}(x_1, \dots, x_{n-k}), f_k(x_{n-k+1}, \dots, x_n)\right)$$

meaning: $y = z$ where





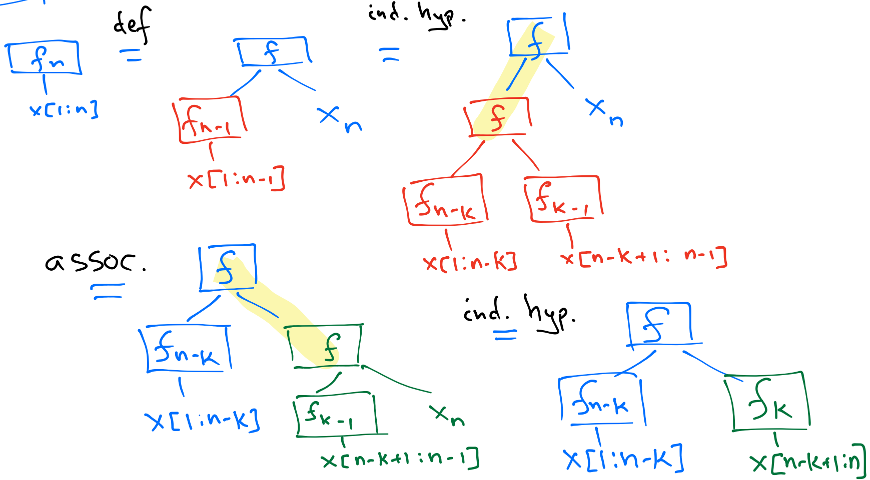
* note: for $k=1$, equality holds because LHS is the same as RHS

proof by comp. ind. on n.

basis: $n=2$ true because $k=1$.

hyp: LHS = RHS $\forall n' < n$ $k \in [1, n'-1]$

step: (by pictures!)



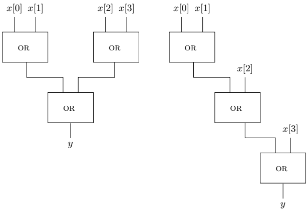
Trees of associative Boolean gates

To simplify the presentation, consider the Boolean function OR_n.

Definition

A combinational circuit $H = (V, E, \pi)$ that satisfies the following conditions is called an OR-tree(n).

1. The graph $DG(H)$ is a rooted tree with n sources.
2. Each vertex v in V that is not a source or a sink is labeled $\pi(v) = \mathrm{OR}$.
3. The set of labels of leaves of H is $\{x_0, \ldots, x_{n-1}\}$.



Correctness of OR-tree(n)

Definition

A combinational circuit $H = (V, E, \pi)$ that satisfies the following conditions is called an OR-tree(n).

1. *Topology.* The graph $DG(H)$ is a rooted tree with n sources.
2. Each vertex v in V that is not a source or a sink is labeled $\pi(v) = \text{OR}$.
3. The set of labels of leaves of H is $\{x_0, \ldots, x_{n-1}\}$.

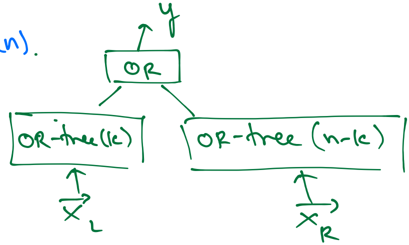
Claim

Every OR*-tree*(n) *implements the Boolean function* OR_n.

$\forall$ OR-tree(n) implements OR_n

<u>proof</u>: by comp. ind. on n. (basis, hyp exercise)

decompose OR-tree(n).



where

$$\vec{x}_L \cap \vec{x}_R = \emptyset$$

& $x_L \cup x_R = \{x_1, \dots, x_n\}$

<u>ind. hyp</u>:

$$\underset{\uparrow\, \vec{x}_L}{\boxed{\text{OR-tree}(k)}} = OR_k(\vec{x}_L)$$

$$\underset{\uparrow\, \vec{x}_R}{\boxed{\text{OR-tree}(n-k)}} = OR_{n-k}(\vec{x}_R)$$

$$y = OR(OR_k(\vec{x}_L), OR_{n-k}(\vec{x}_R)) \overset{\text{claim}}{=} OR_n(\vec{x}_L \cup \vec{x}_R) \overset{\text{comm.}}{=} OR_n(\vec{x}) \quad \boxtimes$$

Relation to Boolean Formulas

Definition

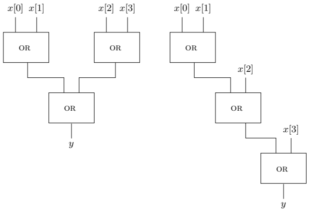
A Boolean formula φ is an $\textsc{or}(n)$ formula if it satisfies three conditions: (i) it is over the variables $X_0, \ldots, X_{n-1}$, (ii) every variable X_i appears exactly once in φ, and (iii) the only connective in φ is the $\textsc{or}$ connective.

Claim

A Boolean circuit C is an $\textsc{or}(n)$-tree if and only if its graph (without the input/output gates) is a parse tree of an $\textsc{or}(n)$-formula.

exercise (hint: complete ind.)

Cost of OR-tree(n)



Claim

The cost of every OR*-tree*(n) *is* $(n-1) \cdot c(\text{OR})$.

Lemma

Let $G = (V, E)$ *denote a rooted tree in which the in-degree of each vertex is at most two. Then*

$$|\{v \in V \mid deg_{in}(v) = 2\}| \;=\; |\{v \in V \mid deg_{in}(v) = 0\}| - 1.$$

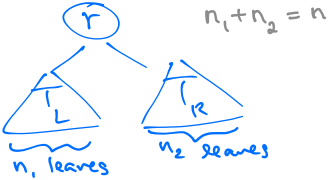
$$|\{v \mid \deg_{in}(v) = 2\}| = |\{v \mid \deg_{in}(v) = 0\}| - 1$$

proof: comp. ind. on $|V|$

basis: $|V| = 1$. LHS $= 0$ RHS $= 1 - 1 = 0$

hyp: $\forall$ rooted tree with #leaves $< n$ ~~$|V| < n$~~ Lemma holds

step: decompose tree



$$|\{v \in \text{tree} \mid \deg_{in}(v) = 2\}|$$

$$= |\{v \in T_L \mid \deg_{in}(v) = 2\}|$$

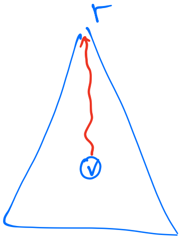
$$+ |\{v \in T_R \mid \deg_{in}(v) = 2\}| + 1$$

$$\overset{\text{ind. hyp.}}{=} (n_1 - 1) + (n_2 - 1) + 1 = n - 1$$

$\boxtimes$

exercise: what if $\deg_{in}(r) = 1$?

depth of vertex in a rooted tree



depth(v) = length of path from v to root.

but we will use a non-standard definition of depth.

confusing, so be aware...

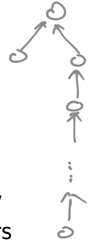
delay of an OR tree = number of OR-gates along the longest path from an input to an output.

Definition (depth - nonstandard definition)

The depth of a rooted tree T is the maximum number of vertices with in-degree greater than one in a path in T. We denote the depth of T by $depth(T)$.

Why is this nonstandard?

- Usually, depth is simply the length of the longest path.
- Here we count only vertices with in-degree ≥ 2.
- Why?
  - Input and output gates have zero delay (no computation)
  - Assume inverters are free and have zero delay (we will show that for $\text{OR}(n)$ cost & delay are not reduced even if inverters are free and without delay)



depth = 1

Binary rooted trees

Definition

A rooted tree is a binary tree if the maximum in-degree is two.

A rooted tree is a minimum depth tree if its depth is minimum among all the rooted trees with the same number of leaves.
All binary trees with n leaves have the same cost. But, which have minimum depth?

1. if n that is a power of 2, then there is a unique minimum depth tree, namely, the perfect binary tree with $\log_2 n$ levels.
2. if n is not a power of 2, then there is more than one minimum depth tree... (balanced trees)

Example: Delay analysis

Are these minimum depth trees?

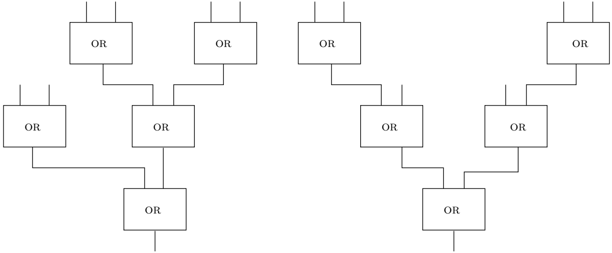


Figure: Two trees with six inputs.

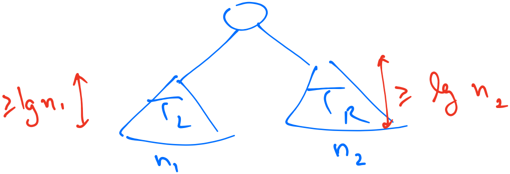
Depth: lower bound

Claim

If T_n is a rooted binary tree with n leaves, then the depth of T_n is at least $\lceil \log_2 n \rceil$.

1. Suffice to prove depth $\geq \log_2 n$.
2. Complete induction on n.

sketch: (comp. ind.)



$*$ $n_1 + n_2 = n$

$\Rightarrow \max\{n_1, n_2\} \geq \frac{n}{2}$

$$\mathrm{depth}(T) = 1 + \max\{\mathrm{depth}(T_L), \mathrm{depth}(T_R)\}$$

$$\geq 1 + \max\{\lg n_1, \lg n_2\}$$

$$\overset{*}{\geq} 1 + \log_2\left(\frac{n}{2}\right) = \log_2 n$$

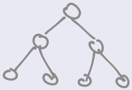
$\boxtimes$

Min Depth: the case $n = 2^k$ (perfect binary trees)

The distance of a vertex v to the root r in a rooted tree is the length of the path from v to r.

Definition

A rooted binary tree is perfect if:



- The in-degree of every non-leaf is 2, and
- All leaves have the same distance to the root.

Note that the depth of a perfect tree equals the distance from the leaves to the root (no vertices with in-degree 1).

Claim

The number of leaves in a perfect tree is 2^k, where k is the distance of the leaves to the root.

Claim

Let n denote the number of leaves in a perfect tree. Then, the distance from every leaf to the root is $\log_2 n$.

Minimum depth trees

We now show that for every n, we can construct a minimum depth tree T_n^* of depth $\lceil \log_2 n \rceil$. In fact, if n is not a power of 2, then there are many such trees.

Balanced partitions

Definition

Two positive integers a, b are a balanced partition of n if:

1. $a + b = n$, and
2. $\max\{\lceil \log_2 a \rceil, \lceil \log_2 b \rceil\} \leq \lceil \log_2 n \rceil - 1$.

Claim

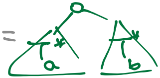
If $n = 2^k - r$, where $0 \leq r < 2^{k-1}$, then the set of balanced partitions is

$$P \triangleq \{(a, b) \mid 2^{k-1} - r \leq a \leq 2^{k-1} \text{ and } b = n - a\}.$$

$n = 13 = 2^4 - 3$

$P = \{(8,5), (7,6), (6,7), (5,8)\}$

Construction of a balanced tree

$T_n^* =$ 

Algorithm 1 Balanced-Tree(n) - a recursive algorithm for constructing a binary tree T_n^* with $n \geq 1$ leaves.

1. The case that $n = 1$ is trivial (an isolated root).
2. If $n \geq 2$, then let a, b be balanced partition of n.
3. Compute trees T_a^* and T_b^*. Connect their roots to a new root to obtain T_n^*.

Definition

A rooted binary tree T_n is a balanced tree if it is a valid output of Algorithm Balanced-Tree(n).

Def: balanced tree

Algorithm 2 Balanced-Tree(n) - a recursive algorithm for constructing a binary tree T_n^* with $n \geq 1$ leaves.

1. The case that $n = 1$ is trivial (an isolated root).
2. If $n \geq 2$, then let a, b be balanced partition of n.
3. Compute trees T_a^* and T_b^*. Connect their roots to a new root to obtain T_n^*.

Claim

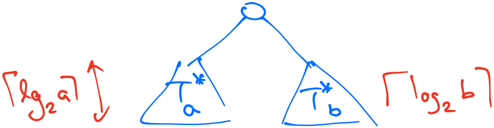
The depth of a binary tree T_n^* constructed by Algorithm Balanced-Tree(n) is $\lceil \log_2 n \rceil$.

Corollary

The propagation delay of a balanced $\textsc{or}$*-tree(n) is* $\lceil \log_2 n \rceil \cdot t_{pd}(\textsc{or})$.

$$\operatorname{depth}(T_n^*) = \lceil \log_2 n \rceil$$

proof sketch: comp. ind. on n



$$\operatorname{depth}(T_n^*) = 1 + \max\{\lceil \log_2 a \rceil, \lceil \log_2 b \rceil\}$$

$$\leq 1 + \lceil \log_2 n \rceil - 1 = \lceil \log_2 n \rceil$$

but T_n^* has n leaves

$$\Longrightarrow \operatorname{depth}(T_n^*) \geq \lceil \log_2 n \rceil$$

$\boxtimes$

Summary

* defined OR_n
* defined OR-tree(n)

can be extended to any binary Boolean assoc. func.

* cost OR-tree(n)
* balanced trees
* delay of balanced trees

CIRCUIT LOWER BOUNDS

given $f_n : \{0,1\}^n \to \{0,1\}$.

find $c : \mathbb{N} \to \mathbb{N}$ such that

$\forall$ comb. circuit COMB_n that computes f_n : $\text{cost}(\text{COMB}_n) \geq c(n)$.

If a comb. circuit H has $\text{cost}(H) < c(n)$,

then H does not compute f_n !

example: $f_n = OR_n$, $f_{2n}(a,b) =$ n'th bit of $\langle a \rangle + \langle b \rangle$

Lower bound does not restrict COMB_n !

Optimality of trees

$\forall$ gate : cost(gate) = 1

Goals: prove optimality of a balanced $\textsc{or}$-tree(n).

Theorem

Let C_n denote a combinational circuit that implements $\textsc{or}_n$. Then,

$$c(C_n) \geq n - 1.$$

Theorem

Let C_n denote a combinational circuit that implements $\textsc{or}_n$. Let k denote the maximum fan-in of a gate in C_n. Then

$$t_{pd}(C_n) \geq \lceil \log_k n \rceil .$$

Flipping bits

Definition

Let $flip_i : \{0,1\}^n \to \{0,1\}^n$ be the Boolean function defined by $flip_i(\vec{x}) \triangleq \vec{y}$, where

$$y_j \triangleq \begin{cases} x_j & \text{if } j \neq i \\ \text{NOT}(x_j) & \text{if } i = j. \end{cases}$$

$flip_1(110) = 101$

Definition (Cone of a Boolean function)

The cone of a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is defined by

$$cone(f) \triangleq \{i : \exists \vec{v} \text{ such that } f(\vec{v}) \neq f(flip_i(\vec{v}))\}$$

Example

$cone(\text{XOR}) = \{1, 2\}$.

We say that f depends on x_i if $i \in cone(f)$.

$\text{XOR}(0,0) \neq \text{XOR}(1,0)$ $\quad 2 \in cone(\text{XOR})$

$\text{XOR}(0,0) \neq \text{XOR}(0,1)$ $\quad 1 \in cone(\text{XOR})$

Example

Consider the following Boolean function:

$$f(\vec{x}) = \begin{cases} 0 & \text{if } \sum_i x_i < 3 \\ 1 & \text{otherwise.} \end{cases}$$

Suppose that one reveals the input bits one by one. As soon as 3 ones are revealed, one can determine the value of $f(\vec{x})$. Nevertheless, the function $f(\vec{x})$ depends on all its inputs, and hence, $cone(f) = \{1, \dots, n\}$.

$f(1, 1, 1, \cdots) = 1$

$f(0, 0 \cdots 0, 1, 1, x) = x$

Constant Functions

Claim

$\text{cone}(f) = \emptyset \iff f$ *is a constant Boolean function.*

$\mathrm{cone}(f) = \emptyset \iff f$ constant

$(\Leftarrow)$ if f constant, then

$\forall v \ \forall i : f(v) = f(\mathrm{flip}_i(v))$

$\Rightarrow \forall i : i \notin \mathrm{cone}(f)$

$\Rightarrow \mathrm{cone}(f) = \emptyset$

$\mathrm{cone}(f) = \phi \iff f$ constant

($\Longrightarrow$) need to prove that (counter-positive)

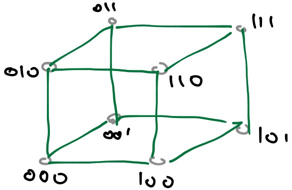
$\mathrm{Cone}(f) \neq \phi \Longleftarrow f$ not const.

suppose : $f : \{0,1\}^n \longrightarrow \{0,1\}$

consider Hypercube $(\{0,1\}^n, E)$ where

$(u,v) \in E$ iff $\exists i : u = \mathrm{flip}_i(v)$

Hypercube over $\{0,1\}^3$



$f \neq$ const $\Rightarrow \exists u \exists v :$

$f(u) = 0$ & $f(v) = 1$

$f(u)=0$, $f(v)=1$.

Consider path from u to v in Hypercube.

[path exists: flip bits in which u & v disagree one at a time.

$000 \to 100 \to 110 \to 111$.]

path: $u = x_0 - x_1 - \dots - x_\ell = v$

$$f(x_0)=0 \quad , \quad f(x_\ell)=1$$

$$\Rightarrow \quad \exists i : f(x_i)=0 \ \& \ f(x_{i+1})=1$$

but $(x_i, x_{i+1}) \in E$, so $\exists j: x_{i+1} = \mathrm{flip}_j(x_i)$

$$\Rightarrow \quad j \in \mathrm{cone}(f).$$

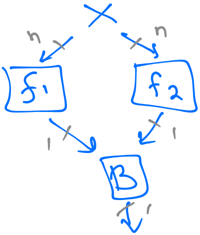
$\boxed{X}$

Composition of Functions

Claim

If $g(\vec{x}) \triangleq B(f_1(\vec{x}), f_2(\vec{x}))$, *then*

$$\text{cone}(g) \subseteq \text{cone}(f_1) \cup \text{cone}(f_2)\ .$$



$$g(x) \triangleq B(f_1(x), f_2(x))$$

$$\mathrm{cone}(g) \subseteq \mathrm{cone}(f_1) \cup \mathrm{cone}(f_2)$$

proof by counter-positive, suffice:

$$i \notin \mathrm{cone}(f_1) \cup \mathrm{cone}(f_2) \Longrightarrow i \notin \mathrm{cone}(g).$$

$$g(x) = B(f_1(x), f_2(x))$$

$$= B(f_1(\mathrm{flip}_i(x)), f_2(\mathrm{flip}_i(x)))$$

$$= g(\mathrm{flip}_i(x))$$

$\boxtimes$

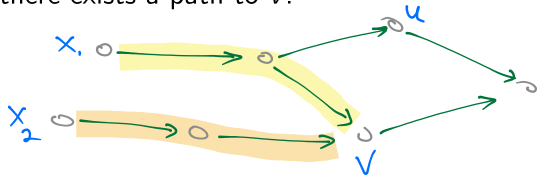
Graphical Cone

Definition

Let $G = (V, E)$ denote a DAG. The graphical cone of a vertex $v \in V$ is defined by

$$cone_G(v) \triangleq \{u \in V : deg_{in}(u) = 0 \text{ and } \exists \text{path from } u \text{ to } v\}.$$

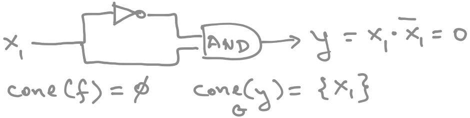
In a combinational circuit, every source is an input gate. This means that the graphical cone of v equals the set of input gates from which there exists a path to v.



$cone_G(v) = \{x_1, x_2\}$

$cone_G(u) = \{x_1\}$

Functional Cone $\subseteq$ Graphical Cone

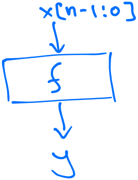


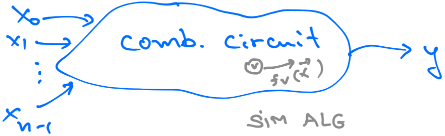
Claim

Let $H = (V, E, \pi)$ denote a combinational circuit. Let $G = DG(H)$. For every vertex $v \in V$, the following holds:

$$\text{cone}(f_v) \subseteq \text{cone}_G(v) \; .$$

Namely, if f_v depends on x_i, then the input gate u that feeds the input x_i must be in the graphical cone of v.





$\mathrm{cone}(f_v) \subseteq \mathrm{cone}_G(v)$

SIM ALG

<u>proof</u> topo. sort of V

$(v_1, v_2, \ldots, v_n)$

prove (by comp. ind on i):

$\forall i: \ \mathrm{cone}(f_{v_i}) \subseteq \mathrm{cone}_G(v_i)$

<u>basis:</u> $i = 1$ (or sources)

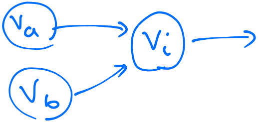
$\mathrm{cone}(f_{v_1}) = \{V_1\} \qquad \mathrm{cone}_G(v_1) = \{v_1\}$

<u>hyp</u>: claim holds for $j < i$.

<u>step</u>:

3 cases: $\deg_{in}(v_i) \in \{0, 1, 2\}$

exercise: $\deg_{in}(v) = 1$



$f_{v_i}(x) = B_{\pi(v_i)}(f_{v_a}(x), f_{v_b}(x))$

$\Rightarrow \mathrm{cone}(f_{v_i}) \subseteq \mathrm{cone}(f_{v_a}) \cup \mathrm{cone}(f_{v_b})$

$\subseteq \mathrm{cone}_G(v_a) \cup \mathrm{cone}_G(v_b)$

$= \mathrm{cone}_G(v_i)$

"Hidden" Rooted Trees

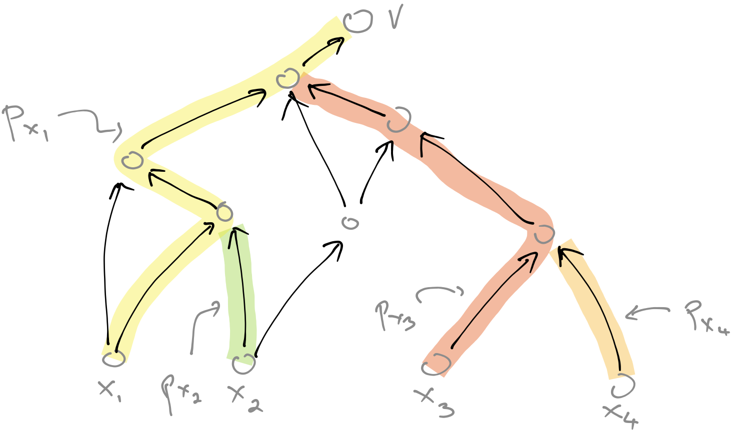
Claim

Let $G = (V, E)$ denote a DAG. For every $v \in V$, there exist $U \subseteq V$ and $F \subseteq E$ such that:

1. *$T = (U, F)$ is a rooted tree;*
2. *v is the root of T;*
3. $\mathrm{cone}_G(v)$ *equals the set of leaves of (U, F).*

The sets U and F are constructed as follows.

1. Initialize $F = \emptyset$ and $U = \emptyset$.
2. For every source u in $cone_G(v)$ do
   (a) Find a path p_u from u to v.
   (b) Let q_u denote the prefix of p_u, the vertices and edges of which are not contained in U or F.
   (c) Add the edges of q_v to F, and add the vertices of q_v to U.



v
p_{x_1}
p_{x_2}
p_{x_3}
p_{x_4}
x_1
x_2
x_3
x_4

Lower Bound on Cost

Theorem (Linear Cost Lower Bound Theorem)

Let $H = (V, E, \pi)$ denote a combinational circuit. If the fan-in of every gate in H is at most 2, then

$$c(H) \geq \max_{v \in V} |\text{cone}(f_v)| - 1.$$

Corollary

Let C_n denote a combinational circuit that implements OR_n. *Then*

$$c(C_n) \geq n - 1.$$

$|\text{cone}(\text{OR}_n)| = n$

$\text{OR}_n(\vec{0}) = 0$

$\text{OR}_n(\text{flip}_i(\vec{0})) = 1$

$$\mathrm{cost}(H) \geq \max_v |\mathrm{cone}(f_v)| - 1$$

<u>proof</u>: consider $DG(H)$. fix vertex v.
construct tree T_v rooted at v with

$$\mathrm{leaves}(T_v) = \mathrm{cone}_{DG(H)}(v)$$

$$|\{u \in T_v : \deg_{in}^{T_v}(u) = 2\}| = |\mathrm{Leaves}(T_v)| - 1$$

$$= |\mathrm{cone}_{DG(H)}(v)| - 1$$

$$\geq |\mathrm{cone}(f_v)| - 1$$

but: $\mathrm{cost}(H) \geq |\{u \in T_v : \deg_{in}^{T_v}(u) = 2\}|$.

note: did not count inv's.

$\boxtimes$

Lower Bound on Delay

Theorem (Logarithmic Delay Lower Bound Theorem)

Let $H = (V, E, \pi)$ denote a combinational circuit. If the fan-in of every gate in H is at most 2, then

$$t_{pd}(H) \geq \max_{v \in V} \log_2 |\mathrm{cone}(f_v)|.$$

Corollary

Let C_n denote a combinational circuit that implements OR_n. Let 2 denote the maximum fan-in of a gate in C_n. Then

$$t_{pd}(C_n) \geq \lceil \log_2 n \rceil .$$

$$\mathrm{tpd}(H) \geq \max_v \log_2 |\mathrm{cone}(f_v)|$$

proof: fix v. let T_v denote tree rooted at v with $\mathrm{Leaves}(T_v) = \mathrm{cone}_{DG(H)}(v)$

$$\begin{aligned}
\mathrm{depth}(T_v) &\geq \log_2 |\mathrm{Leaves}(T_v)| \\
&= \log_2 |\mathrm{cone}_{DG(H)}(v)| \\
&\geq \log_2 |\mathrm{cone}(f_v)|
\end{aligned}$$

$\mathrm{tpd}(H) \geq \mathrm{depth}(T_v)$.

$\boxtimes$

What is the effect of increasing the fan-in on the delay?

Theorem (Logarithmic Delay Lower Bound Theorem)

Let $H = (V, E, \pi)$ denote a combinational circuit. If the fan-in of every gate in H is at most k, then

$$t_{pd}(H) \geq \max_{v \in V} \log_k |\text{cone}(f_v)|.$$

Corollary

Let C_n denote a combinational circuit that implements OR_n. *Let k denote the maximum fan-in of a gate in C_n. Then*

$$t_{pd}(C_n) \geq \lceil \log_k n \rceil .$$

exercise

Summary

- Focus on combinational circuits that have a topology of a tree with identical gates.
- Trees are especially suited for computing associative Boolean functions.
- Defined an $\textsc{or}$-tree(n) to be a combinational circuit that implements $\textsc{or}_n$ using a topology of a tree.
- Proved that $\textsc{or}$-tree(n) are asymptotically optimal (cost).
- Balance conditions to obtain good delay.
- General lower bounds based on $cone(f)$.
    - $\#$ gates in a combinational circuit implementing a Boolean function f must be at least $|cone(f)| - 1$.
    - the propagation delay of a combinational circuit implementing a Boolean function f is at least $\log_2 |cone(f)|$.