Chapter 3: Trees

Computer Structure - Spring 2004

©Dr. Guy Even

Tel-Aviv Univ.

Goals

- define associative Boolean functions (and classify them).
- trees - combinational circuits that implement associative Boolean funcs.
- analyze delay & cost of trees.
- prove optimality.

Associative dyadic boolean functions

Def: A Boolean function $f : \{0,1\}^2 \rightarrow \{0,1\}$ is associative if

$$f(f(\sigma_1, \sigma_2), \sigma_3) = f(\sigma_1, f(\sigma_2, \sigma_3)),$$

for every $\sigma_1, \sigma_2, \sigma_3 \in \{0,1\}$.

Q: List all the associative Boolean functions

$$f : \{0,1\}^2 \rightarrow \{0,1\}.$$

"A": There are 16 dyadic Boolean functions, only need to list them and check...

f_n : repeating $f : \{0,1\}^2 \rightarrow \{0,1\}$

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

1. If $n = 2$ then $f_2 \equiv f$.
2. If $n > 2$, then f_n is defined based on f_{n-1} as follows:

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

Example:

$$\text{NOR}_4(x_1, x_2, x_3, x_4) = \text{NOR}(\text{NOR}(\text{NOR}(x_1, x_2), x_3), x_4).$$

Note that NOR is not associative!

f_n : the associative case

If $f(x_1, x_2)$ is associative, then parenthesis are not important.

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

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

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

Q: Show that the set of functions $f_n(x_1, \ldots, x_n)$ that are induced by associative dyadic Boolean functions is

$$\{\text{constant } 0, \text{constant } 1, x_1, x_n, \text{AND}, \text{OR}, \text{XOR}, \text{NXOR}\}.$$

note: only last 4 functions are "interesting". We focus on OR.

Definition of OR-trees

Def: A combinational circuit $C = \langle \mathcal{G}, \mathcal{N} \rangle$ that satisfies the following conditions is called an OR-tree(n).

1. **Input:** $x[n-1:0]$.
2. **Output:** $y \in \{0,1\}$
3. **Functionality:** $y = \text{OR}(x[0], x[1], \cdots, x[n-1])$.
4. **Gates:** All the gates in $\mathcal{G}$ are OR-gates.
5. **Topology:** The underlying graph of $DG(C)$ (i.e. undirected graph obtained by ignoring edge directions) is a tree.

   Note that in the tree:
   - leaves correspond to the inputs $x[n-1:0]$ and the output y.
   - interior nodes - OR-gates.
   - Could root the tree, and then the root is the output.

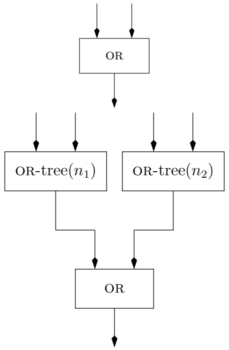
Recursive definition of OR-trees

Def: an OR-tree(n) is defined recursively as follows:

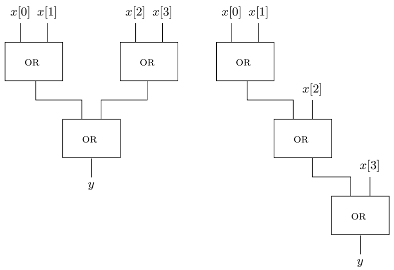
basis: a single OR-gate is an OR-tree(2).

step: an OR(n)-tree is a circuit in which

1. the output is computed by an OR-gate.
2. the inputs of this OR-gate are the outputs of OR-tree(n_1) & OR-tree(n_2), where $n = n_1 + n_2$.



Example: OR-tree(4)



Cost - both trees have 3 gates.

Delay - 2 gates vs. 3.

Cost of OR-trees

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

Proof: By induction on n.

Induction basis: $n = 2$. In this case, OR-tree(2) contains a single OR-gate.

Cost of OR-trees - Induction step

- let C denote an OR-tree(n).
- let g denote the OR-gate that outputs the output of C.
- g is fed by two wires e_1 and e_2.
- e_1 is the output of C_1 - an OR-tree(n_1)
- e_2 is the output of C_2 - an OR-tree(n_2)
- $n_1 + n_2 = n$
- Ind. Hyp. $\Rightarrow c(C_1) = (n_1 - 1) \cdot c(\text{OR})$ & $c(C_2) = (n_2 - 1) \cdot c(\text{OR})$.
-

$$\begin{aligned} c(C) &= c(g) + c(C_1) + c(C_2) \\ &= (1 + n_1 - 1 + n_2 - 1) \cdot c(\text{OR}) \\ &= (n-1) \cdot c(\text{OR}). \end{aligned}$$

QED

Delay of OR-trees

Claim: The delay of a balanced OR-tree(n) is

$$\lceil \log_2 n \rceil \cdot t_{pd}(\text{OR}).$$

Proof: homework. Note that the term “balanced tree” can be interpreted in more than one way especially if n is not a power of 2...

Are balanced OR-trees optimal?

- What is the best (min. cost & delay) choice of a topology for a combinational circuit that implements the Boolean function OR_n? Is a tree indeed the best topology?
- Could one do better if another implementation is used?

Optimality of balanced OR-trees

Would like to prove that every combinational circuit C that implements OR_n satisfies:

$$c(C) \geq n - 1$$
$$t_{pd}(C) \geq \log_2 n.$$

We need to be more accurate about the model:

Q: what is the cost/delay of an n-input OR-gate?

assumption 1: the fan-in of every gate ≤ 2, so we have to build big gates from basic gates.

assumption 2: the cost of every basic gate is ≥ 1. (input/output gates are free)

Optimality of balanced OR-trees

Would like to prove that every combinational circuit C that implements OR_n satisfies:

$$c(C) \geq n - 1$$
$$t_{pd}(C) \geq \log_2 n.$$

Looking for proof also for the case that $DG(C)$ is not a tree!

Restriction of a Boolean function

Def: Let $f : \{0,1\}^n \to \{0,1\}$ denote a Boolean function. Let $\sigma \in \{0,1\}$. The Boolean function $g : \{0,1\}^{n-1} \to \{0,1\}$ defined by

$$g(w_0, \ldots, w_{n-2}) \triangleq f(w_0, \ldots, w_{i-1}, \sigma, w_i, \ldots, w_{n-2})$$

is called the restriction of f with $x_i = \sigma$. We denote it by $f_{\restriction x_i=\sigma}$.

Examples:
$$\text{XOR}_{\restriction x_2=1}(x_1) \triangleq \text{XOR}(x_1, 1)$$

$$\text{MAJORITY}_{\restriction x_n=1}(x_1, \ldots, x_{n-1}) \triangleq \begin{cases} 1 & \text{if } \sum_{i=1}^{n-1} x_i + 1 > n/2 \\ 0 & \text{otherwise.} \end{cases}$$

Cone of a Boolean function

A boolean function $f : \{0,1\}^n \to \{0,1\}$ depends on its ith input if

$$f_{\restriction x_i=0} \not\equiv f_{\restriction x_i=1}.$$

Def: The cone of a Boolean function f is defined by

$$\textit{cone}(f) \triangleq \{i : f \text{ depends on its } i\text{th input}\}.$$

Claim: The Boolean function OR_n depends on all its inputs, namely

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

Input-Output reachability

Claim: If a combinational circuit C implements a Boolean function f, then there must be a path in $DG(C)$ from every input in *cone*(f) to the output of f.

Proof: by contradiction,

- assume $i \in$ *cone*(f).
- let $g_i \in \mathcal{G}$ denote the input gate that feeds the ith input.
- assume that in $DG(C)$ there is no path from g_i to the output y.
- show that C does not implement f.

Input-Output reachability - cont.

Find vectors $w', w'' \in \{0,1\}^n$ such that

$$f(w') \neq f(w'')$$
$$w'[i] \neq w''[i].$$

Proof of Simulation Theorem
$\Rightarrow C$ outputs the same value in y when input w' and w''.

$\Rightarrow C$ errs either with w' or with w''. QED

Linear Cost Lower Bound Theorem

assumptions:

- fan-in of every gate at most 2.
- cost of trivial gates (i.e. input/output gates) is zero.
- cost of non-trivial gate is at least 1.

Theorem: If C is a combinational circuit that implements a Boolean function f, then

$$c(C) \geq |cone(f)| - 1.$$

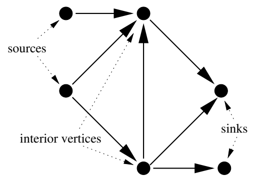
Corollary: If C_n is a combinational circuit that implements OR_n, then $c(C_n) \geq n - 1$.

Easy to prove theorem for trees, but what about arbitrary DAGs?

DAG terminology

Consider the directed acyclic graph (DAG) $DG(C)$.

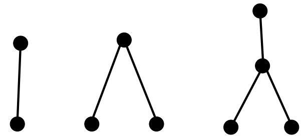
- $\deg_{in}(v)$: in-degree of a vertex v is the number of edges that enter the vertex v.
- $\deg_{out}(v)$: out-degree of a vertex v : is the number of edges that emanate from the vertex v.
- source - a vertex with in-degree zero.
- sink - a vertex with out-degree zero.
- interior vertex - a vertex that is neither a source or a sink.



Leaves and interior vertices in trees

- Let $T = (V, E)$ denote a tree with at least two vertices.
- A leaf is a vertex of degree 1.
- An interior vertex is a vertex that is not a leaf.
- leaves(V) - set of leaves in V.
- interior(V) - set of interior vertices in V.
- Claim:
  If the degree of every vertex in T is at most three, then

$$|\text{interior}(V)| \geq |\text{leaves}(V)| - 2.$$



Underlying graph of $DG(C)$

- C - a combinational circuit & $DG(C) = (V, A)$ - a DAG
- underlying graph of $DG(C)$ - undirected graph $G = (V, E)$, where $(u, v) \in E \;\Leftrightarrow\; (u \to v) \in A$.
- If fan-in of every gate in C is at most 2, then degree of every vertex in G is at most 3.
- Leaves in G correspond to input and output gates in C.
- Interior vertices in G correspond to non-trivial gates in C.
- Case of a tree:
  Claim: Assume C has n inputs and a single output. Assume fan-in of all gates is at most 2. If G is a tree, then

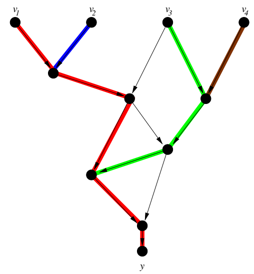
$$c(C) \geq n - 1.$$

Proof of linear cost lower bound theorem

- If underlying graph of $DG(C)$ is a tree, then previous claim proves the theorem.
- If $DG(C) = (V, E)$ is not a tree, then construct a directed "binary tree" $T = (V', E')$ such that
  - $V' \subseteq V$ & $E' \subseteq E$
  - sources$(T') = cone(f)$
  - output gate $\in V'$.
- in T' we have |interior nodes| $\geq$ |sources| $- 1$.
- But interior nodes of T are also interior in $DG(C)$, and number of sources in T equals $|cone(f)|$. QED.

Left to show how T is constructed...

Construction of T



Logarithmic Delay Lower Bound Theorem

Theorem: Let $C = \langle \mathcal{G}, \mathcal{N} \rangle$ denote a combinational circuit that implements a non-constant Boolean function $f : \{0,1\}^n \to \{0,1\}$. If the fan-in of every gate in $\mathcal{G}$ is at most c, then the delay of C is at least $\log_c |cone(f)|$.

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

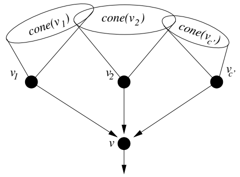
$$t_{pd}(C_n) \geq \lceil \log_c n \rceil .$$

Proof of logarithmic lower bound

- deal only with the graph $DG(C)$.
- show that exists a path with at least $\log_c |cone(f)|$ interior vertices in $DG(C)$.
- why interior?
- input/output gates and constants have zero delay $\Rightarrow$ should not be counted.
- only sources & sinks have zero delay $\Rightarrow$ count interior vertices.
- $cone(v)$ - set of sources from which v is reachable. Note that $|cone(output)| = |cone(f)|$.
- $d(v)$ - max number of interior vertices along a path from a source in $cone(v)$ to v (including v).
- suffice to prove that $d(v) \geq \log_c |cone(v)|$.

Proof: $d(v) \geq \log_c |cone(v)|$

- Proof by induction on $d(v)$.
- Basis: $d(v) = 0$. In this case v is a source, $|cone(v)| = 1$.
- Step: $d(v) = i + 1$. Edges entering v are $v_1 \to v, \ldots, v_{c'} \to v$, for $c' \leq c$.
- by def: $d(v) = \max\{d(v_i)\}_{i=1}^{c'} + 1$.
- $cone(v) = \bigcup_{i=1}^{c'} cone(v_i)$.



$$|cone(v)| \leq \sum_{i=1}^{c'} |cone(v_i)|$$
$$\leq c' \cdot \max\{|cone(v_i)|\}_{i=1}^{c'}.$$

Cont. proof: $d(v) \geq \log_c |cone(v)|$

Let v' denote a predecessor of v that satisfies

$$|cone(v')| = \max\{|cone(v_i)|\}_{i=1}^{c'} \geq |cone(v)|/c'.$$

The induction hypothesis implies that

$$d(v') \geq \log_c |cone(v')|.$$

But,

$$\begin{aligned} d(v) &\geq 1 + d(v') \\ &\geq 1 + \log_c |cone(v')| \\ &\geq \log_c c + \log_c |cone(v)|/c' \\ &\geq \log_c |cone(v)|. \end{aligned}$$

Cont. proof: $d(v) \geq \log_c |cone(v)|$

- Finally, we deal with the case that v is a sink.
- v has a unique predecessor v'.
- we have $d(v) = d(v')$ and $cone(v) = cone(v')$.
- induction step applies to v', and hence we have

$$d(v) \geq \log_c |cone(v)|,$$

as required.

Summary

- associative Boolean functions.
- extend dyadic functions to functions with n arguments.
- only four non-trivial associative Boolean functions.
- OR-tree(n) - combinational circuits that implement OR_n using a topology of a tree.
- cost(OR-tree) $= n - 1$.
- t_{pd}(balanced OR-tree) $= \log_2 n$.
- Balanced OR-trees optimal cost & delay.
- two lower bounds:
  - cost $\geq |cone(f)| - 1$.
  - $t_{pd} \geq \log_c |cone(f)|$.