Advanced Digital Logic Design – EECS 303

http://ziyang.eecs.northwestern.edu/eecs303/

Teacher:	Robert Dick
Office:	L477 Tech
Email:	dickrp@northwestern.edu
Phone:	847–467–2298

NORTHWESTERN
UNIVERSITY

Outline

1. Heuristic two-level minimization

2. Homework

There be Dragons here

Today's material might at first appear difficult

Perhaps even a bit dry

. . . but follow closely

Trust me, if you really get it, there is great depth and beauty here

Section outline

1. Heuristic two-level minimization
    - Review of logic minimization
    - Definitions
    - Espresso algorithm
    - Espresso phases
    - Tautology checking

Two-level logic minimization

- Goal: two-level logic realizations with fewest gates and fewest number of gate inputs
- Algebraic
- Karnaugh map
- Quine–McCluskey
- Espresso heuristic

Optimal two-level logic synthesis is $\mathcal{NP}$-complete

Upper bound on number of prime implicants grows $3^n/n$ where n is the number of inputs

Given > 16 inputs, can be intractable

However, there have been advances in complete solvers for many functions

- Optimal solutions are possible for some large functions

Logic minimization methods

For difficult and large functions, solve by heuristic search

Multi-level logic minimization is also best solved by search

The general search problem can be introduced via two-level minimization

- Examine simplified version of the algorithms in Espresso

Section outline

1. Heuristic two-level minimization
    - Review of logic minimization
    - Definitions
    - Espresso algorithm
    - Espresso phases
    - Tautology checking

Espresso two-level logic minimization heuristic

- Generate only a subset of prime implicants
- Carefully select prime implicants in this subset covering on-set
- Guaranteed to be correct
- May not be minimal
- Usually high-quality in practice

Espresso

Start with a potentially optimal algorithm

Add numerous techniques for constraining the search space

Use efficient move order to allow pruning

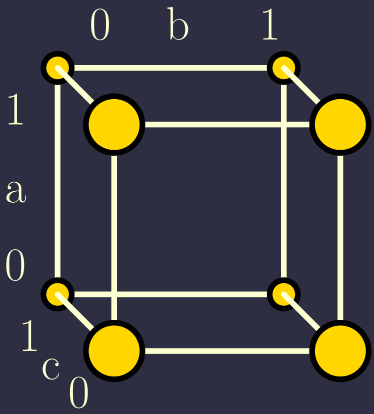
Disable backtracking to arrive at a heuristic solver

Widely used in industry

Still has room for improvement

- E.g., early recursion termination

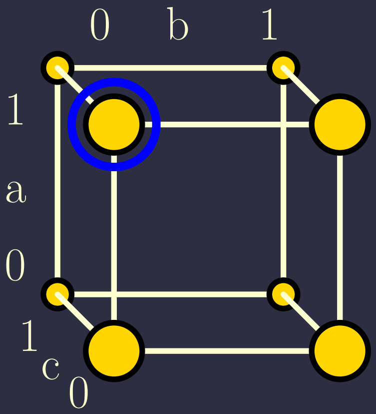
Boolean space



If g and h are two Boolean functions s.t. the on-set of g is a subset of the on-set of h then

- h *covers* g or…
- …$g \subseteq h$

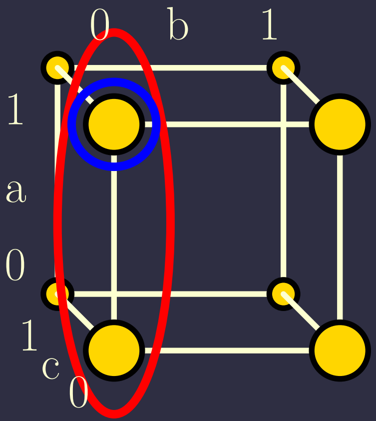
Boolean space



If g and h are two Boolean functions s.t. the on-set of g is a subset of the on-set of h then

- h *covers* g or. . .
- . . . $g \subseteq h$

Boolean space



If g and h are two Boolean functions s.t. the on-set of g is a subset of the on-set of h then

- h *covers* g or. . .
- . . . $g \subseteq h$

Redundancy in Boolean space

If a formula contains $A\overline{B}$ and $\overline{B}$, $A\overline{B} \subseteq \overline{B} \Rightarrow A\overline{B}$ is *redundant*

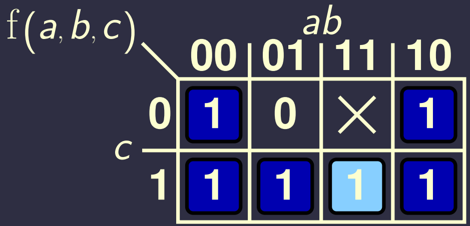
Sometimes redundancy is difficult to observe

- If $f = BC + AB + A\overline{C}$, then AB is redundant

Section outline

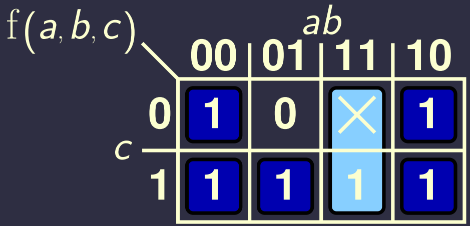
1. Heuristic two-level minimization
    - Review of logic minimization
    - Definitions
    - Espresso algorithm
    - Espresso phases
    - Tautology checking

Espresso moves



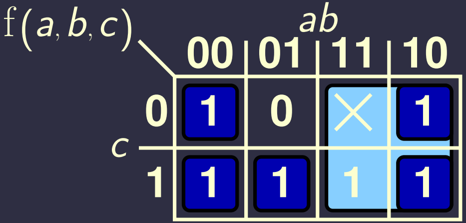
- Add a literal to a cube (reduce)
- Remove a literal from a cube (expansion)
- Remove redundant cubes (irredundant cover)

Espresso moves



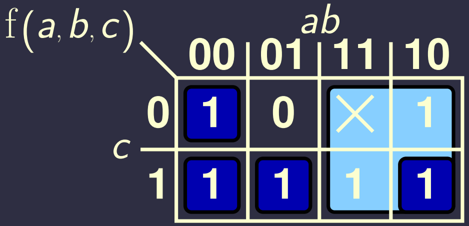
- Add a literal to a cube (reduce)
- Remove a literal from a cube (expansion)
- Remove redundant cubes (irredundant cover)

Espresso moves



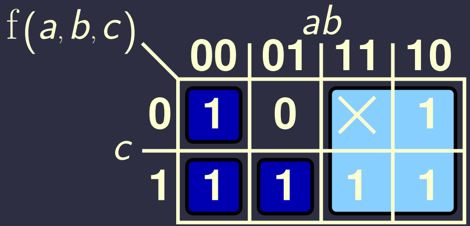
- Add a literal to a cube (reduce)
- Remove a literal from a cube (expansion)
- Remove redundant cubes (irredundant cover)

Espresso moves



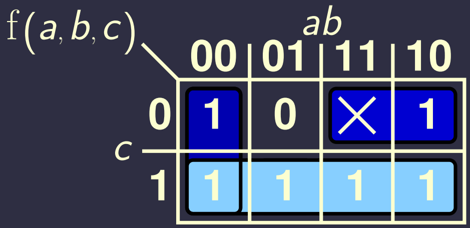
- Add a literal to a cube (reduce)
- Remove a literal from a cube (expansion)
- Remove redundant cubes (irredundant cover)

Espresso moves



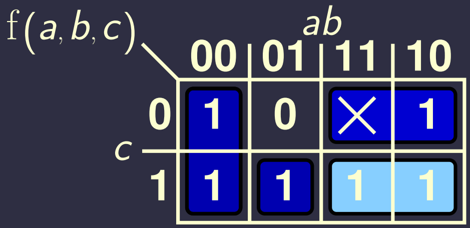
- Add a literal to a cube (reduce)
- Remove a literal from a cube (expansion)
- Remove redundant cubes (irredundant cover)

Espresso moves



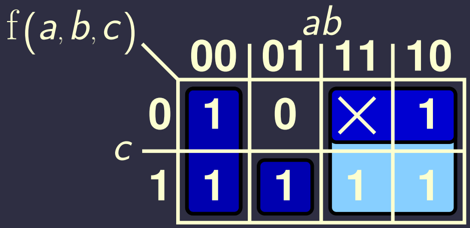
- Sometimes necessary to increase cost to escape local minima
- Add a literal to a cube (reduction)

Espresso moves



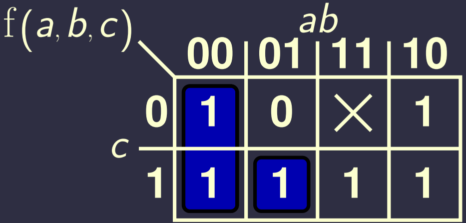
- Sometimes necessary to increase cost to escape local minima
- Add a literal to a cube (reduction)
- To later allow expansion in another dimension

Espresso moves



- Sometimes necessary to increase cost to escape local minima
- Add a literal to a cube (reduction)

Espresso moves

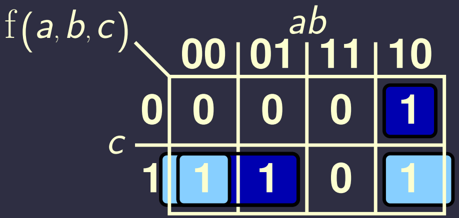


- Sometimes necessary to increase cost to escape local minima
- Add a literal to a cube (reduction)

Irredundant functions need not be minimal

$f(a,b,c)$ — c \ ab	00	01	11	10
0	0	0	0	1
1	1	1	0	1

Irredundant functions need not be minimal



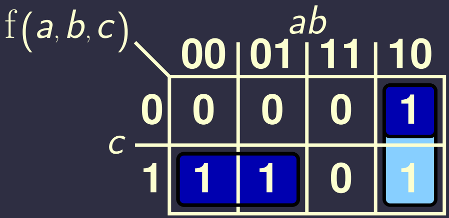
- $\overline{B}\,C + \overline{A}\,C + A\overline{B}\,\overline{C}$

Irredundant functions need not be minimal

$f(a,b,c)$	ab = 00	01	11	10
c = 0	0	0	0	1
1	1	1	0	1

- $\overline{B}\,C + \overline{A}\,C + A\overline{B}\,\overline{C}$
- Reduce: $A\overline{B}\,C + \overline{A}\,C + A\overline{B}\,\overline{C}$

Irredundant functions need not be minimal



- $\overline{B}\,C + \overline{A}\,C + A\overline{B}\,\overline{C}$
- Reduce: $A\overline{B}\,C + \overline{A}\,C + A\overline{B}\,\overline{C}$
- Expand: $A\overline{B} + \overline{A}\,C + A\overline{B}\,\overline{C}$

Irredundant functions need not be minimal

$f(a, b, c)$

c \ ab	00	01	11	10
0	0	0	0	1
1	1	1	0	1

- $\overline{B}\,C + \overline{A}\,C + A\overline{B}\,\overline{C}$
- Reduce: $A\overline{B}\,C + \overline{A}\,C + A\overline{B}\,\overline{C}$
- Expand: $A\overline{B} + \overline{A}\,C + A\overline{B}\,\overline{C}$
- Irredundant cover: $A\overline{B} + \overline{A}\,C$

Espresso algorithm

Repeat the following

1. REDUCE sometimes necessary to contain cubes within others
2. An IRREDUNDANT COVER is extracted from the expanded primes
3. EXPAND implicants to their maximum size

Espresso algorithm

Repeat the following

1. REDUCE sometimes necessary to contain cubes within others
   - Another cover with fewer terms or fewer literals might exist
   - Shrink prime implicants to allow expansion in another variable
2. An IRREDUNDANT COVER is extracted from the expanded primes
3. EXPAND implicants to their maximum size

Espresso algorithm

Repeat the following

1. REDUCE sometimes necessary to contain cubes within others
2. An IRREDUNDANT COVER is extracted from the expanded primes
   - Similar goals to the Quine-McCluskey prime implicant chart
   - Good performance requires a few tricks
3. EXPAND implicants to their maximum size

Espresso algorithm

Repeat the following

1. REDUCE sometimes necessary to contain cubes within others
2. An IRREDUNDANT COVER is extracted from the expanded primes
3. EXPAND implicants to their maximum size
    - Implicants covered by an expanded implicant are removed from further consideration
    - Quality of result depends on order of implicant expansion
    - Heuristic methods used to determine order

Espresso algorithm

Repeat sequence REDUCE, EXPAND, IRREDUNDANT COVER to find alternative prime implicants

Keep doing this as long as new covers improve on last solution

A number of optimizations are tried, e.g., identify and remove essential primes early in the process

Espresso pseudocode

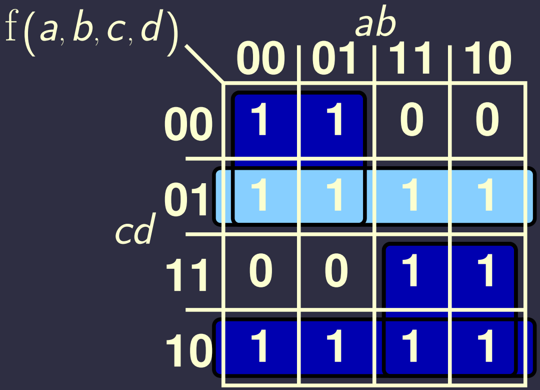
```
Procedure ESPRESSO(F, D, R)
/* F is ON set, D is don't care, R OFF */
R = COMPLIMENT(F+D); /* Compute complement */
F = EXPAND(F, R); /* Initial expansion */
F = IRREDUNDANT(F,D); /* Initial irredundant cover */
E = ESSENTIAL(F,D) /* Detecting essential primes */
F = F - E; /* Remove essential primes from F */
D = D + E; /* Add essential primes to D */
while Cost(F) keeps decreasing do
    F = REDUCE(F,D); /* Perform reduction, heuristic which cubes */
    F = EXPAND(F,R); /* Perform expansion, heuristic which cubes */
    F = IRREDUNDANT(F,D); /* Perform irredundant cover */
end while
F = F + E;
return F;
```

Espresso example

$f(a, b, c, d)$

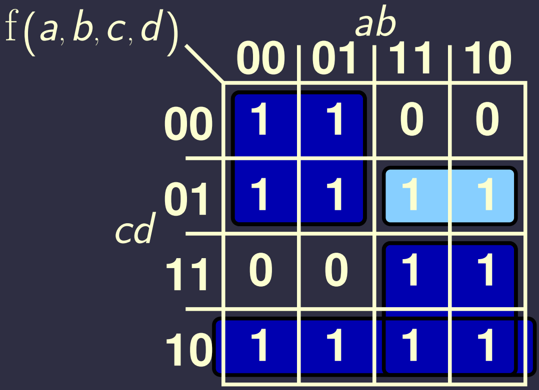
cd \ ab	00	01	11	10
00	1	1	0	0
01	1	1	1	1
11	0	0	1	1
10	1	1	1	1

Espresso example



Irredundant but not minimal

Espresso example



Reduce

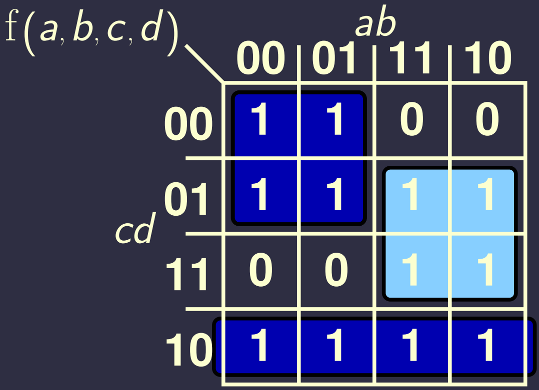
Espresso example

$f(a, b, c, d)$

cd \ ab	00	01	11	10
00	1	1	0	0
01	1	1	1	1
11	0	0	1	1
10	1	1	1	1

Expand

Espresso example



Irredundant Cover

Espresso input

$$f(A,B,C,D) = \sum(4,5,6,8,9,10,13) + d(0,7,15)$$

Input	Meaning
.i 4	# inputs
.o 1	# outputs
.ilb a b c d	input names
.ob f	output name
.p 10	number of product terms
0100 1	$\overline{A}\,B\,\overline{C}\,\overline{D} = 1$
0101 1	$\overline{A}\,B\,\overline{C}\,D = 1$
0110 1	$\overline{A}\,B\,C\,\overline{D} = 1$
1000 1	$A\,\overline{B}\,\overline{C}\,\overline{D} = 1$
1001 1	$A\,\overline{B}\,\overline{C}\,D = 1$
1010 1	$A\,\overline{B}\,C\overline{D} = 1$
1101 1	$A\,B\,\overline{C}\,D = 1$
0000 -	$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D} = X$
0111 -	$\overline{A}\,B\,C\,D = X$
1111 -	$A\,B\,C\,D = X$
.e	end

Espresso output

$$f(A, B, C, D) = \sum(4, 5, 6, 8, 9, 10, 13) + d(0, 7, 15)$$

Output	Meaning
.i 4	# inputs
.o 1	# outputs
.ilb a b c d	input names
.ob f	output name
.p 3	number of product terms
1-01 1	$A\,\overline{C}\,D = 1$
10-0 1	$A\,\overline{B}\,\overline{D} = 1$
01– 1	$\overline{A}\,B = 1$
.e	end

$$g(A, B, C, D) = A\overline{C}\,D + A\overline{B}\,\overline{D} + \overline{A}\,B$$

Two-level heuristic minimization summary

- Generating all prime implicants can be too expensive
- Make incremental changes: EXPAND, REDUCE, AND IRREDUNDANT COVER to improve cover
- Determining whether incremental change represents same function is difficult
  - Need to use clever algorithms to speed it up

Section outline

1. Heuristic two-level minimization
    - Review of logic minimization
    - Definitions
    - Espresso algorithm
    - Espresso phases
    - Tautology checking

Irredundant cover

- After expansion, it's necessary to remove redundant cubes to reach a local minimum
- First, find the *relatively essential cubes*
- For each other cube, check to see whether it is covered by relatively essential cubes or don't-cares
- If so, it's *totally redundant*
- If not, it's *partially redundant*

Irredundant cover

- Relatively essential cubes must be kept
- Totally redundant cubes can clearly be eliminated
- A subset of the partially redundant cubes need to be kept
- Formulate as a unate covering problem
    - We'll come back to this in a moment

Irredundant cover

- After expansion, it's necessary to remove redundant cubes to reach a local minimum
- First, find the *relatively essential cubes*
- For each other cube, check to see whether it is covered by relatively essential cubes or don't-cares
- If so, it's *totally redundant*
- If not, it's *partially redundant*

Section outline

1. Heuristic two-level minimization
   - Review of logic minimization
   - Definitions
   - Espresso algorithm
   - Espresso phases
   - Tautology checking

Tautology check for relatively essential cubes

- c is a 1-cube
- Check to see whether the union of 1-cubes and don't-care cubes minus c, cofactored by c, is a tautology
- Let A be the set of 1-cubes
- Let D be the set of don't-care cubes
- $((A \cup D) - c)_c \neq 1 \Leftrightarrow$ c is relatively essential
- That's it: You can use tautology checking to determine whether a cube is relatively essential
- Of course, an example would make it clearer

Terminology example

a	b	c	f
0	X	X	1
X	0	X	1
X	X	1	1
1	X	1	1
1	0	0	X

- Find the relatively essential cubes
- Find totally redundant cubes
- Find partially redundant cubes

Detecting relatively essential cubes

- How to determine whether a cube is fully covered by other 1 and don't-care cubes?
- Could decompose everything to minterm canonical form

Detecting relatively essential cubes

- How to determine whether a cube is fully covered by other 1 and don't-care cubes?
- Could decompose everything to minterm canonical form
- Recall that there may be 2^n minterms, given n variables
- Decomposition is a bad idea
  - Exponential

Recursive pivoting?

- Could also recursively pivot on variables if inclusion fails

$$0XX|1$$
$$\downarrow$$
$$00X|1,\ 01X|1$$
$$\downarrow$$
$$000|1,\ 001|1,\ 010|1,\ 011|1$$

- Lets us terminate recursion as soon as cube is covered by single other cube, e.g., 01X|X
- However, even with pruning, this is still slow in practice
- Worst-case time complexity?

Definition: Cofactor by variable

$$f_{x_1} = f(1, x_2, \ldots, x_n)$$
$$f_{\overline{x_1}} = f(0, x_2, \ldots, x_n)$$

Note that it's commutative,

$$(f_{x_1})_{x_2} = (f_{x_2})_{x_1}$$

Definition: Cofactor by cube, usage

Given that c is a cube, and literals $l_1, l_2, \ldots l_n \in c$, cofactoring the function by the cube is equivalent to sequentially cofactoring by all cube literals, i.e.,

$$\mathrm{f}_c = \mathrm{f}_{l_1, l_2, \ldots l_n}$$

$$c \subseteq f \iff f_c = 1$$

A tautology is a function that is always true

A cube is less than or equal to a function, i.e., is fully covered by the function, if and only if the function cofactored by the cube is a tautology

Problem conversion

Thus, we have taken the problem

Determine whether a cube, c, is covered by a set of 1-cubes, A, or don't-care, D, cubes.

and converted it to

Determine whether a set of 1-cubes, A, and don't-care cubes, D, cofactored by cube c is a tautology.

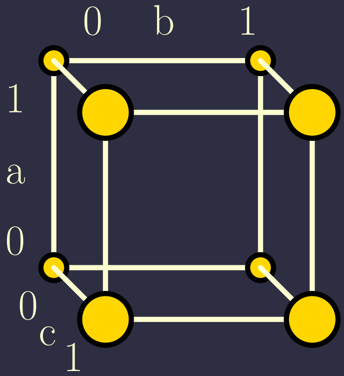
Conversion benefits

- Cofactoring eliminates variables, speeding analysis
- Tautology is a straight-forward and well-understood problem
- However, tautology checking is not easy
  - Could pivot on all variables…
  - …but this is too slow
  - Example?

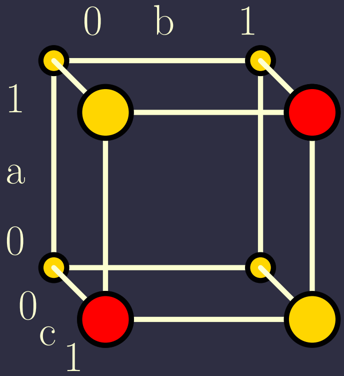
Unate functions

- $f(x_1, x_2, \ldots, x_n)$ is monotonically increasing in x_1 if and only if $\forall x_2, \ldots, x_n : f(0, x_2, \ldots, x_n) \leq f(1, x_2, \ldots, x_n)$
- $f(x_1, x_2, \ldots, x_n)$ is monotonically decreasing in x_1 if and only if $\forall x_2, \ldots, x_n : f(0, x_2, \ldots, x_n) \geq f(1, x_2, \ldots, x_n)$
- A function that is neither monotonically increasing or monotonically decreasing in x_1 is non-monotonic in x_1
- A function that is monotonically increasing or monotonically decreasing in x_1 is unate in x_1
- A function that is unate in all its variables is unate

Unate functions

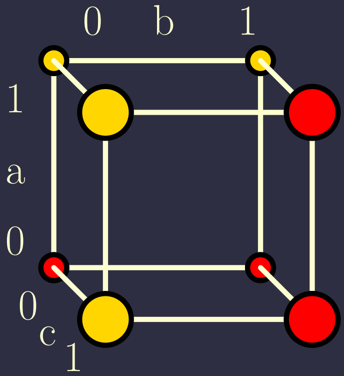


Unate functions



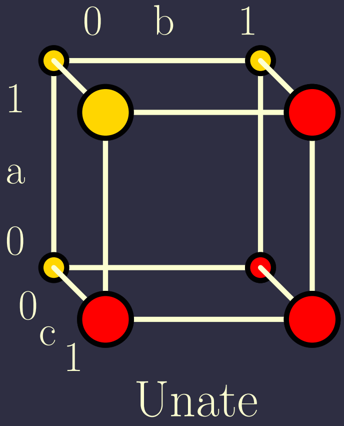
Unate in c, not in a or b

Unate functions

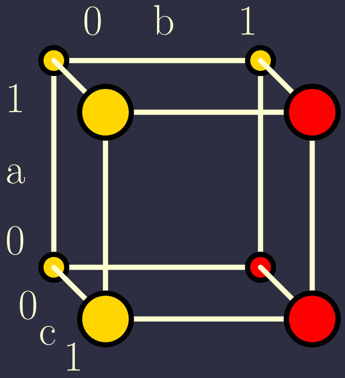


Unate in a and b, not in c

Unate functions



Unate functions



Unate

Unate covers

- Unate functions are difficult to identify
- A cover is unate as long as the complemented and uncomplimented literals for the same variable do not both appear
- Identifying unate covers is easy

Identifying unate covers is easy

a	b	c	f
0	0	X	1
X	1	1	1
X	1	0	1
0	1	1	X

Identifying unate covers is easy

a	b	c	f
0	0	X	1
X	1	1	1
X	1	0	1
0	1	1	X

- Scan the columns for the presence of a 0 and 1

Identifying unate covers is easy

a	b	c	f
0	0	X	1
X	1	1	1
X	1	0	1
0	1	1	X

- Scan the columns for the presence of a 0 and 1
- Note, some unate functions can have non-unate covers
  - Unate covers always express a unate function

Unate cover tautology checking

- A unate cover is a tautology if and only if it contains a 1, i.e., $XXX|1$
- Think of it this way: There is some point or cube in the input space of the function at which all cubes intersect
  - Thus, the only way to have a tautology is for one of the cubes to be a tautology
- Thus, it's trivial to check unate covers for tautology
  - Search for a tautology cube

Fast tautology checking using unate covers

Given that C is a cover containing cubes composed of variables $x_1, x_2, \ldots, x_n$

$\textsc{Tautology}(C)$

if C is unate **then**
  if C contains a 1-row **then**
    Return *true*
  end if
else
  Return $\textsc{Tautology}(C_{x_1}) \wedge \textsc{Tautology}(C_{\overline{x_1}})$
end if

Fast tautology checking using unate covers

What if the cover isn't unate?

- Can still accelerate

If cover C is unate in a variable, x_1, then factor out x_1

$$C = x_1 \cdot \mathrm{F}_1(x_2, \ldots, x_n) + \mathrm{F}_2(x_2, \ldots, x_n)$$

or

$$C = \overline{x_1} \cdot \mathrm{F}_1(x_2, \ldots, x_n) + \mathrm{F}_2(x_2, \ldots, x_n)$$

Example of unate cofactoring

$$\overline{a}\,c + bc + \overline{a}\,\overline{b}\,\overline{c}$$

Cover unate only in a

$$\left(\overline{a}\,c + bc + \overline{a}\,\overline{b}\,\overline{c}\right)_a = bc$$

$$\left(\overline{a}\,c + bc + \overline{a}\,\overline{b}\,\overline{c}\right)_{\overline{a}} = c + bc + \overline{b}\,\overline{c}$$

Notice anything nice about this?

Example of unate cofactoring

$$\overline{a}\,c + bc + \overline{a}\,\overline{b}\,\overline{c}$$

Cover unate only in a

$$\left(\overline{a}\,c + bc + \overline{a}\,\overline{b}\,\overline{c}\right)_a = bc$$

$$\left(\overline{a}\,c + bc + \overline{a}\,\overline{b}\,\overline{c}\right)_{\overline{a}} = c + bc + \overline{b}\,\overline{c}$$

Notice anything nice about this?

Fast tautology checking using unate covers

Assume

$$C = x_1 \cdot \mathrm{F}_1(x_2, \ldots, x_n) + \mathrm{F}_2(x_2, \ldots, x_n)$$

Then the C_{x_1} cofactor is

$$\mathrm{F}_1(x_2, \ldots, x_n) + \mathrm{F}_2(x_2, \ldots, x_n)$$

and the $C_{\overline{x_1}}$ cofactor is

$$\mathrm{F}_2(x_2, \ldots, x_n)$$

Example of unate cofactoring

$$\overline{a}\,c + bc + \overline{a}\,\overline{b}\,\overline{c}$$

Cover unate only in a

$$\left(\overline{a}\,c + bc + \overline{a}\,\overline{b}\,\overline{c}\right)_a = bc$$
$$\left(\overline{a}\,c + bc + \overline{a}\,\overline{b}\,\overline{c}\right)_{\overline{a}} = c + bc + \overline{b}\,\overline{c}$$
$$\mathrm{F}_1 = c + \overline{b}\,\overline{c}$$
$$\mathrm{F}_2 = bc$$

Fast tautology checking using unate covers

$$C_{x_1} = \mathrm{F}_1(x_2, \ldots, x_n) + \mathrm{F}_2(x_2, \ldots, x_n)$$
$$C_{\overline{x_1}} = \mathrm{F}_2(x_2, \ldots, x_n)$$

Clearly,

$$C_{\overline{x_1}} \subseteq C_{x_1}$$

Therefore we need only consider $C_{\overline{x_1}}$ for tautology checking, significantly simplifying the problem, i.e., if $C_{\overline{x_1}}$ is a tautology, then C_{x_1} is obviously also a tautology.

Tautology checking final version

Given that C is a cover containing cubes composed of variables $x_1, x_2, \ldots, x_n$

$\textsc{Tautology}(C)$

if C is unate **then**
  if C contains a 1-row **then**
    Return *true*
  end if
else
  Return $\textsc{Tautology}(C_{\overline{x_1}})$
end if

Tautology checking final version

Given that C is a cover containing cubes composed of variables $x_1, x_2, \ldots, x_n$

$\textsc{Tautology}(C)$

if C is unate **then**
  if C contains a 1-row **then**
    Return *true*
  end if
else
  Return $\textsc{Tautology}(C_{\overline{x_1}})$
end if

Summary: Fast tautology checking

- Identify unate covers
  - No columns with 1s and zeros
- If unate, scan for an $XXX|1$ row
- If not unate, cofactor on (preferable) unate variable
- Only need to consider uncomplemented or uncomplemented cofactor
  - Why? $F_2 \leq F_1 + F_2$

More complicated example

a	b	c	f
0	X	0	1
1	0	X	1
X	1	1	1
1	X	1	1
X	0	0	1
1	0	0	X

Relatively essential check for $0X0|1$?

Full check on c.

Two-level heuristic minimization summary

- Generating all prime implicants can be too expensive
- Make incremental changes: EXPAND, REDUCE, AND IRREDUNDANT COVER
- Determining whether incremental change represents same function is too expensive
- Use cofactoring to convert it to a tautology check
- Use unateness to make the tautology check fast in most cases

Espresso summary

Reduce: Allows expansion in another direction, get out of local minima

Expand: Decreases complexity, in practice blocking matrix used for expansion. Search with would also work but would be slower in most cases.

Irredundant cover: Remove redundant cubes

Tautology check used in many places, gave example of use in Irredundant cover use

We have only scratched the surface!

CAD

Computer–Aided Design (of Integrated Circuits and Systems)

Also called Electronics Design Automation (EDA)

Without it, computers wouldn't work

Questions

What is the unate covering problem?

Where have we seen it used?

What can tautology checking be used for?

How do we make it fast?

Next lecture: Implementation technologies

- PALs, PLAs
- MUX, DEMUX review
- Steering logic

Outline

1. Heuristic two-level minimization
2. Homework

Reading assignment

- M. Morris Mano and Charles R. Kime. *Logic and Computer Design Fundamentals*. Prentice-Hall, NJ, fourth edition, 2008
- Chapter 4
- M. Morris Mano and Charles R. Kime. *Web supplements to Logic and Computer Design Fundamentals*. Prentice-Hall, NJ. http://www.writphotec.com/mano4/Supplements
- VLSI Programmable Logic Devices, document 1