# Logic Synthesis and Verification 

## SOPs and Incompletely Specified Functions

Reading:<br>Logic Synthesis in a Nutshell<br>Section 2

most of the following slides are by courtesy of Andreas Kuehlmann

## Boolean Function Representation

## Sum of Products

$\square$ A function can be represented by a sum of cubes (products):

- E.g., $f=a b+a c+b c$

Since each cube is a product of literals, this is a "sum of products" (SOP) representation

- An SOP can be thought of as a set of cubes $F$
- E.g., $F=\{a b, a c, b c\}$
$\square$ A set of cubes that represents $f$ is called a cover of $f$
- E.g.,
$F_{1}=\{a b, a c, b c\}$ and $F_{2}=\left\{a b c, a b c^{\prime}, a b b^{\prime} c, a^{\prime} b c\right\}$ are covers of $f=a b+a c+b c$.


## List of Cubes (Cover Matrix)

$\square$ We often use a matrix notation to represent a cover:

- Example

$$
\mathrm{F}=\mathrm{ac}+\mathrm{c}^{\prime} \mathrm{d}=
$$

> a b c d abcd
> ac $\rightarrow 1212$ or $1-1$ $c^{\prime} d \rightarrow 2201 \quad-01$
$\square$ Each row represents a cube
$\square 1$ means that the positive literal appears in the cube $\square 0$ means that the negative literal appears in the cube $\square 2$ (or -) means that the variable does not appear in the cube. It implicitly represents both 0 and 1 values.

## PLA

$\square$ A PLA is a (multiple-output) function $\mathrm{f}: \mathrm{B}^{n} \rightarrow \mathrm{~B}^{m}$ represented in SOP form
$n=3, m=3$
cover matrix


| abc | $f_{1} f_{2} f_{3}$ |
| :--- | :--- |
| $10-1--$ |  |

-11 1 - -
0-0 - 1 -
$111-11$
00- - - 1
$\square$ Each distinct cube appears just once in the ANDplane, and can be shared by (multiple) outputs in the OR-plane, e.g., cube (abc)Extensions from single-output to multiple-output minimization theory are straightforward

## SOP

$\square$ The cover (set of SOPs) can efficiently represent many practical logic functions (i.e., for many practical functions, there exist small covers)

ㅁ Two-level minimization seeks the cover of minimum size (least number of cubes)
= onset minterm
Note that each onset minterm is "covered" by at least one of the cubes!
None of the offset minterms is covered

## Irredundant Cube

$\square$ Let $F=\left\{c_{1}, c_{2}, \ldots, c_{k}\right\}$ be a cover for $f$, i.e., $f=\sum_{i=1}^{k} C_{i}$
A cube $c_{i} \in F$ is irredundant if $F \backslash\left\{c_{i}\right\} \neq f$

- Example

$$
f=a b+a c+b c
$$




Not covered
$\mathrm{F} \backslash\{\mathrm{ab}\} \neq \mathrm{f}$

## Prime Cube

## Prime and Irredundant Cover

$\square$ A literal $x$ (a variable or its negation) of cube $c \in F$ (cover of $f$ ) is prime if $(F \backslash\{c\}) \cup\left\{C_{x}\right\} \neq f$,
where $c_{x}$ (cofactor w.r.t. $x$ ) is $c$ with literal $x$ of $c$ deleted
$\square$ A cube of $F$ is prime if all its literals are prime
$\square$ Example

$$
\begin{aligned}
& f=x y+x z+y z \\
& \left.c=x y ; c_{y}=x \text { (literal } y \text { deleted }\right) \\
& F \backslash\{c\} \cup\left\{c_{y}\right\}=x+x z+y z
\end{aligned}
$$

inequivalent to f since offset vertex is covered

$\square$ Definition 1. A cover is prime (resp. irredundant) if all its cubes are prime (resp. irredundant)

- Definition 2. A prime (cube) of $f$ is essential (essential prime) if there is a onset minterm (essential vertex) in that prime but not in any other prime
$\square$ Definition 3. Two cubes are orthogonal if they do not have any minterm in common
- E.g.

$$
c_{1}=x^{\prime} y
$$

$c_{2}=y^{\prime} z$ are orthogonal
$c_{2}=y z$ are not orthogonal

## Prime and Irredundant Cover

## Incompletely Specified Function

- Example
$f=a b c+b^{\prime} d+c^{\prime} d$ is prime and irredundant.
abc is essential since abcd' $\in a b c$, but not in b'd or c'd or ad


Why is abcd not an essential vertex of abc?
What is an essential vertex of abc?
What other cube is essential? What prime is not essential? represents "don't care"

- $\mathrm{f}=$ onset function
$f(x)=1 \leftrightarrow F(x)=1$
- $r=$ offset function $r(x)=1 \leftrightarrow F(x)=0$
- d = don't care function $d(x)=1 \leftrightarrow F(x)=*$
$\square(f, d, r)$ forms a partition of $B^{n}$, i.e,
$-f+d+r=B^{n}$
$\square(f \cdot d)=(f \cdot r)=(d \cdot r)=\varnothing($ pairwise disjoint)
(Here we don't distinguish characteristic functions and the sets they represent)


## Incompletely Specified Function

A completely specified function g is a cover for $F=(f, d, r)$ if$\mathrm{f} \subseteq \mathrm{g} \subseteq \mathrm{f}+\mathrm{d}$

- $\mathrm{g} \cdot \mathrm{r}=\varnothing$
- if $x \in d$ (i.e. $d(x)=1$ ), then $g(x)$ can be 0 or 1;
if $x \in f$, then $g(x)=1$; if $x \in r$, then $g(x)=0$
$\square$ We "don't care" which value $g$ has at $x \in d$


## Prime of Incompletely Specified Function

$\square$ Definition. A cube $c$ is a prime of $F=(f, d, r)$ if $c \subseteq$ $\mathrm{f}+\mathrm{d}$ (an implicant of $\mathrm{f}+\mathrm{d}$ ), and no other implicant ( of $f+d$ ) contains c (i.e., it is simply a prime of $\mathrm{f}+\mathrm{d}$ )
$\square$ Definition. Cube $c_{j}$ of cover $G=\left\{c_{i}\right\}$ of $F=(f, d, r)$ is redundant if $f \subseteq G \backslash\left\{c_{j}\right\}$; otherwise it is
irredundant
$\square$ Note that $\mathrm{c} \subseteq \mathrm{f}+\mathrm{d} \leftrightarrow \mathrm{c} \cdot \mathrm{r}=\varnothing$

## Prime of Incompletely Specified Function

$\square$ Example
Consider logic minimization of $F(a, b, c)=(f, d, r)$ with
$f=a^{\prime} b c^{\prime}+a b^{\prime} c+a b c$ and $d=a b c^{\prime}+a b^{\prime} c^{\prime}$
$F_{1}=\left\{a^{\prime} b c^{\prime}, a b^{\prime} c, a b c\right\}$

- on
- off
- don't care
$F_{2}=\left\{a, a^{\prime} b c^{\prime}, a b \prime c\right\}$ $a b$ 'c is redundant $a$ is prime
$F_{3}=\left\{a, a^{\prime} b c^{\prime}\right\}$
Expand $a^{\prime} b c^{\prime} \rightarrow b c$

$$
F_{4}=\left\{a, b c^{\prime}\right\}
$$



## Checking of Prime and Irredundancy

Let $G$ be a cover of $F=(f, d, r)$, and $D$ be a cover for $d$

- $c_{i} \in G$ is redundant iff
$c_{i} \subseteq\left(G \backslash\left\{c_{i}\right\}\right) \cup D$
(Let $G^{i} \equiv G \backslash\left\{c_{i}\right\} \cup D$. Since $c_{i} \subseteq G^{i}$ and $f \subseteq G \subseteq f+d$, then $c_{i} \subseteq c_{i} f+c_{i} d$ and $c_{i} f$ $\subseteq G \backslash\left\{c_{i}\right\}$. Thus $f \subseteq G \backslash\left\{c_{i}\right\}$.)
$\square$ A literal $I \in c_{i}$ is prime if $\left(c_{i} \backslash\{I\}\right)\left(=\left(c_{i}\right)_{I}\right)$ is not an implicant of $F$
$\square$ A cube $c_{i}$ is a prime of $F$ iff all literals $I \in c_{i}$ are prime Literal $I \in c_{i}$ is not prime $\Leftrightarrow\left(c_{i}\right)_{\perp} \subseteq f+d$

Note: Both tests (1) and (2) can be checked by tautology (to be explained):
$\begin{array}{ll}\square & \left(\mathrm{G}^{\mathrm{i}}\right)_{\mathrm{c}_{\mathrm{i}}} \equiv 1 \\ \square & (\mathrm{f} \cup \mathrm{d})_{\left(\mathrm{c}_{\mathrm{i}}\right)} \equiv 1\end{array} \quad$ (implies $\mathrm{c}_{\mathrm{i}}$ redundant)
The above two cofactors are with respect to cubes instead of literals

## (Literal) Cofactor

$\square$ Let $\mathrm{f}: \mathrm{B}^{\mathrm{n}} \rightarrow \mathrm{B}$ be a Boolean function, and $\mathrm{x}=\left(\mathrm{x}_{1}, \mathrm{x}_{2}, \ldots, \mathrm{x}_{\mathrm{n}}\right)$ the variables in the support of $f$; the cofactor $f_{a}$ of $f^{\prime} b y$ a literal $a=x_{i}$ or $a=\neg x_{i}$ is
$\square f_{x_{i}}\left(x_{1}, x_{2}, \ldots, x_{n}\right)=f\left(x_{1}, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_{n}\right)$
$\square f_{-x_{i}}\left(x_{1}, x_{2}, \ldots, x_{n}\right)=f\left(x_{1}, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_{n}\right)$
The computation of the cofactor is a fundamental operation in Boolean reasoning!

- Example



## (Literal) Cofactor

$\square$ The cofactor $C_{x_{j}}$ of a cube $C$ (representing some Boolean function) with respect to a literal $x_{j}$ is

- C
if $x_{i}$ and $x_{j}$ ' do not appear in C
$\square C \backslash\left\{x_{j}\right\} \quad$ if $x_{j}$ appears positively in C, i.e., $x_{j} \in C$
$\square \varnothing \quad$ if $x_{i}$ appears negatively in $C$, i.e., $x_{j}{ }^{\prime} \in C$
$\square$ Example
$\begin{array}{ll}C=x_{1} x_{4}^{\prime} x_{6}, & \\ C_{x_{2}}=C & \left(x_{2} \text { and } x_{2}^{\prime} \text { do not appear in } C\right) \\ C_{x_{1}}=x_{4}^{\prime} x_{6} & \left(x_{1} \text { appears positively in } C\right) \\ C_{x_{4}}=\varnothing & \left(x_{4} \text { appears negatively in } C\right)\end{array}$


## (Literal) Cofactor

$\square$ Example
$F=a b c^{\prime}+b^{\prime} d+c d$
$\mathrm{F}_{\mathrm{b}}=\mathrm{ac}{ }^{\prime}+\mathrm{cd}$
(J ust drop b everywhere and throw away cubes containing literal b')

Cofactor and disjunction commute!

## Shannon Expansion

Let $f: \mathrm{B}^{\mathrm{n}} \rightarrow \mathrm{B}$
Shannon Expansion:

$$
f=x_{i} f_{x_{i}}+x_{i}^{\prime} f_{x_{i}}{ }^{\prime}
$$

Theorem: $F$ is a cover of $f$. Then

$$
F=x_{i} F_{x i}+x_{i}^{\prime} F_{x_{i}}{ }^{\prime}
$$

We say that $f$ and $F$ are expanded about $x_{i}$, and $x_{i}$ is called the splitting variable

## Shannon Expansion

- Example
$F=a b+a c+b c$
$F=a F_{a}+a^{\prime} F_{a}$,
$=a(b+c+b c)+a^{\prime}(b c)$
$=a b+a c+a b c+a^{\prime} b c$
Cube bc got split into two cubes

a

a


## (Cube) Cofactor

$\square$ The cofactor $f_{C}$ of $f$ by a cube $C$ is $f$ with the fixed values indicated by the literals of $C$
$\square$ E.g., if $C=x_{i} x_{j}$, then $x_{i}=1$ and $x_{j}=0$
$\square$ For $C=x_{1} x_{4}{ }^{\prime} x_{6}, f_{C}$ is just the function $f$ restricted to the subspace where $x_{1}=x_{6}=1$ and $x_{4}=0$
$\square$ Note that $f_{C}$ does not depend on $x_{1}, x_{4}$ or $x_{6}$ anymore
(However, we still consider $f_{c}$ as a function of all $n$
variables, it just happens to be independent of $x_{1}, x_{4}$ and $x_{6}$ )
$\square \mathrm{x}_{1} \mathrm{f} \neq \mathrm{f}_{\mathrm{x}_{1}}$
ロE.g., for $f=a c+a \prime c, a \cdot f_{a}=a \cdot f=a \cdot c$ and $f_{a}=c$

## Containment vs. Tautology

$\square$ A fundamental theorem that connects functional containment and tautology:

Theorem. Let c be a cube and f a function. Then $\mathrm{c} \subseteq \mathrm{f} \Leftrightarrow \mathrm{f}_{\mathrm{c}} \equiv 1$.
Proof.
We use the fact that $\mathrm{xf}_{\mathrm{x}}=\mathrm{xf}$, and $\mathrm{f}_{\mathrm{x}}$ is independent of x .
$(\Leftarrow)$
Suppose $f_{c} \equiv 1$. Then $c f=f_{c} c=c$. Thus, $c \subseteq f$. ( $\Rightarrow$ )
Suppose $\mathrm{c} \subseteq \mathrm{f}$. Then $\mathrm{f}+\mathrm{c}=\mathrm{f}$. In addition, $\mathrm{f}_{\mathrm{c}}=(\mathrm{f}+\mathrm{c})_{\mathrm{c}}=\mathrm{f}_{\mathrm{c}}+1=1$. Thus, $\mathrm{f}_{\mathrm{c}}=1$.

## Checking of Prime and Irredundancy (Revisited)

Let $G$ be a cover of $F=(f, d, r)$. Let $D$ be a cover for $d$
$\square c_{i} \in G$ is redundant iff
$c_{i} \subseteq\left(G \backslash\left\{c_{i}\right\}\right) \cup D$
(Let $G^{i} \equiv G \backslash\left\{c_{i}\right\} \cup D$. Since $c_{i} \subseteq G^{i}$ and $f \subseteq G \subseteq f+d$, then $c_{i} \subseteq c_{i} f+c_{i} d$ and $c_{i} f$ $\subseteq G \backslash\left\{c_{i}\right\}$. Thus $f \subset G \backslash\left\{c_{i}\right\}$.)
$\square$ A literal $I \in c_{i}$ is prime if $\left(c_{i} \backslash\{I\}\right)\left(=\left(c_{i}\right)_{1}\right)$ is not an implicant of $F$
$\square$ A cube $c_{i}$ is a prime of $F$ iff all literals $I \in C_{i}$ are prime
Literal $I \in c_{i}$ is not prime $\Leftrightarrow\left(c_{i}\right)_{I} \subseteq f+d$
Note: Both tests (1) and (2) can be checked by tautology (explained):
$\square\left(\mathrm{G}^{\mathrm{i}}\right)_{\mathrm{c}_{\mathrm{i}}} \equiv 1 \quad$ (implies $\mathrm{c}_{\mathrm{i}}$ redundant)
$\square \quad(\mathrm{f} \cup \mathrm{d})_{\left(\mathrm{c}_{\mathrm{i})},\right.} \equiv 1 \quad$ (implies I not prime)
The above two cofactors are with respect to cubes instead of literals

## Generalized Cofactor

$\square$ Definition. Let $f, g$ be completely specified functions. The generalized cofactor of $f$ with respect to g is the incompletely specified function:

$$
\operatorname{co}(f, g)=(f \cdot g, \bar{g}, \bar{f} \cdot g)
$$

$\square$ Definition. Let $\mathfrak{I}=(f, d, r)$ and $g$ be given. Then

$$
\operatorname{co}(\mathfrak{I}, g)=(f \cdot g, d+\bar{g}, r \cdot g)
$$

## Shannon vs. Generalized Cofactor

$\square$ Let $\mathrm{g}=\mathrm{x}_{\mathrm{i}}$. Shannon cofactor is

$$
f_{x_{i}}\left(x_{1}, x_{2}, \ldots, x_{n}\right)=f\left(x_{1}, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_{n}\right)
$$

$\square$ Generalized cofactor with respect to $\mathrm{g}=\mathrm{x}_{\mathrm{i}}$ is

$$
\operatorname{co}\left(f, x_{i}\right)=\left(f \cdot x_{i}, \bar{x}_{i}, \bar{f} \cdot x_{i}\right)
$$Note that

$$
f \cdot x_{i} \subseteq f_{x_{i}} \subseteq f \cdot x_{i}+\bar{x}_{i}=f+\bar{x}_{i}
$$

In fact $f_{x}$, is the unique cover of $\operatorname{co}\left(f, x_{i}\right)$ independent of the variable $\mathbf{x}_{\mathbf{i}}$.

Shannon vs. Generalized Cofactor


## Shannon vs. Generalized Cofactor



$$
\operatorname{co}(f, a)=(f \cdot a, \bar{a}, \bar{f} \cdot a)
$$



So $f \cdot a \subseteq f_{a} \subseteq f+\bar{a}$

Shannon vs. Generalized Cofactor

## Shannon Cofactor

$x \cdot f_{x}+\bar{x} \cdot f_{\bar{x}}=f$
$\left(f_{x}\right)_{y}=f_{x y}$
$(f \cdot g)_{y}=f_{y} \cdot g_{y}$
$(\bar{f})_{x}=\overline{\left(f_{x}\right)}$

## Generalized Cofactor

$f=g \cdot \operatorname{co}(f, g)+\bar{g} \cdot \operatorname{co}(f, \bar{g})$
$\operatorname{co}(\operatorname{co}(f, g), h)=\operatorname{co}(f, g h)$
$\operatorname{co}(f \cdot g, h)=\operatorname{co}(f, h) \cdot \operatorname{co}(g, h)$
$\operatorname{co}(\bar{f}, g)=\overline{\operatorname{co(f,g)}}$

We will get back to the use of generalized cofactor later

## Operation on Cube Lists

## Data Structure for SOP Manipulation

$\square$ AND operation.

- take two lists of cubes
- take two lists of cubes
new list
- represent cubes in computer words
- implement set operations as bit-vector operations

Algorithm AND(List_of_Cubes C1,List_of_Cubes C2) \{
$C=\varnothing$
foreach c1 $\in$ C1 \{ foreach c2 $\in$ C2 \{ $c=c 1 \& c 2$
$c=C \cup c$
\} ${ }^{\}}$
\}
return C

## Operation on Cube Lists

## Operation on Cube Lists

- OR operation:
- take two lists of cubes
- computes union of both lists

ㅁ Naive implementation:
Algorithm OR(List_of_Cubes C1, List_of_Cubes C2) \{ return C1 $\cup$ C2
\}
ㅁ On-the-fly optimizations:

- remove cubes that are completely covered by other cubes $\square$ complexity is $\mathrm{O}\left(\mathrm{m}^{2}\right)$; m is length of list
- conjoin adjacent cubes (consensus operation)
- remove redundant cubes?
- coNP-complete
$\square$ too expensive for non-orthogonal lists of cubes


## $\square$ Simple trick:

■ keep cubes in lists orthogonal
$\square$ check for redundancy becomes $\mathrm{O}\left(\mathrm{m}^{2}\right)$
$\square$ but lists become significantly larger (worst case: exponential)

■ Example
01-0

$1-01$ OR $\quad 0-1-\quad=$| $01-0$ |
| :--- |
| $1-11$ |
| $001-$ |
| 0111 |
| $1-11$ |

## Operation on Cube Lists

ㅁ Adding cubes to orthogonal list:

```
Algorithm ADD_CUBE(List_of_Cubes C, Cube c) {
    if(C = \varnothing) return {c}
    Cres = c-c' /* chopping off minterms may result in multiple cubes */
    foreach cres \in Cres {
        C = ADD_CUBE(C\{c'},cres) \cup {c'}
    }
    }
}
```

- How can the above procedure be further improved?

What about the AND operation, does it gain from orthogonal cube lists?

## Operation on Cube Lists

ㅁ Naive implementation of COMPLEMENT operation
■ apply De'Morgan's law to SOP

- complement each cube and use AND operation

Example

| Input |  |  |
| :--- | :--- | :--- |
| $01-10 \Rightarrow$ | non-orth. |  |
| $1----$ | orthogonal |  |
| $-0---$ | $1----$ |  |
|  | $--0-0--$ |  |
|  | ----1 | $01-0-$ |
|  |  | $01-11$ |

ㅁ Naive implementation of TAUTOLOGY check

- complement function using the COMPLEMENT operator and check for emptiness
- We will show that we can do better than that!


## Tautology Checking

$\square$ Let A be an orthogonal cover matrix, and all cubes of A be pair- wise distinguished by at least two literals (this can be achieved by an on-the-fly merge of cube pairs that are distinguished by only one literal)

Does the following conjecture hold?

$$
A \equiv 1 \Leftrightarrow A \text { has a row of all "-"s? }
$$

This would dramatically simplify the tautology check!

## Tautology Checking

```
```

Algorithm CHECK_TAUTOLOGY(List_of_Cubes C) {

```
```

Algorithm CHECK_TAUTOLOGY(List_of_Cubes C) {
if(C == \varnothing) return FALSE;
if(C == \varnothing) return FALSE;
if(C == {-...-})return TRUE; // cube with all '.'
if(C == {-...-})return TRUE; // cube with all '.'
xi = SELECT_VARIABLE(C)
xi = SELECT_VARIABLE(C)
C0 = COFACTOR(C,\negXi)
C0 = COFACTOR(C,\negXi)
if(CHECK_TAUTOLOGY(C0) == FALSE) {
if(CHECK_TAUTOLOGY(C0) == FALSE) {
print xi = 0
print xi = 0
return FALSE;
return FALSE;
}
}
C1 = COFACTOR(C,Xi)
C1 = COFACTOR(C,Xi)
if(CHECK_TAUTOLOGY(C1) == FALSE) {
if(CHECK_TAUTOLOGY(C1) == FALSE) {
print xi = 1
print xi = 1
return FALSE
return FALSE
}
}
return TRUE;
return TRUE;
}

```
}
```

```
        return FALSE
```

```
        return FALSE
```


## Tautology Checking

- Implementation tricks
- Variable ordering:
$\square$ pick variable that minimizes the two sub-cases ("-"s get replicated into both cases)

■ Quick decision at leaf:
-return TRUE if C contains at least one complete "-" cube among others (case 1)
$\square r e t u r n$ FALSE if number of minterms in onset is $<2^{n}$ (case 2)
$\square r e t u r n ~ F A L S E$ if C contains same literal in every cube (case 3)

## Tautology Checking



## Special Functions

$\square$ Definition. A function $f: B^{n} \rightarrow B$ is symmetric with respect to variables $x_{i}$ and $x_{i}$ iff
$f\left(x_{1}, \ldots, x_{i}, \ldots, x_{j}, \ldots, x_{n}\right)=f\left(x_{1}, \ldots, x_{j}, \ldots, x_{i}, \ldots, x_{n}\right)$
$\square$ Definition. A function $\mathrm{f}: \mathrm{B}^{\mathrm{n}} \rightarrow \mathrm{B}$ is totally symmetric iff any permutation of the variables in $f$ does not change the function

Symmetry can be exploited in searching BDD since

$$
f_{x_{i} \bar{x}_{j}}=f_{\bar{x}_{i} x_{j}}
$$

- can skip one of four sub-cases
- used in automatic variable ordering for BDDs


## Special Functions

$\square$ Definition. A function $f: B^{n} \rightarrow B$ is positive unate in variable $x_{i}$ iff

$$
f_{\bar{x}_{i}} \subseteq f_{x_{i}}
$$

■ This is equivalent to monotone increasing in $\mathrm{x}_{\mathrm{i}}$ :

$$
f\left(m^{-}\right) \leq f\left(m^{+}\right)
$$

for all min-term pairs $\left(\mathrm{m}^{-}, \mathrm{m}^{+}\right)$where

$$
\begin{aligned}
m_{j}^{-} & =m_{j}^{+}, j \neq i \\
m_{i}^{-} & =0 \\
m_{i}^{+} & =1
\end{aligned}
$$

- Example
(1001, 1011) with $i=3$


## Special Functions

$\square$ Similarly for negative unate $f_{x_{i}} \subseteq f_{\bar{x}_{i}}$
monotone decreasing $f\left(m^{-}\right) \geq f\left(m^{+}\right)$
$\square$ A function is unate in $x_{i}$ if it is positive unate or negative unate in $\mathrm{x}_{\mathrm{i}}$

Definition. A function is unate if it is unate in each variable
$\square$ Definition. A cover $F$ is positive unate in $x_{i}$ iff $\bar{x}_{i} \notin C_{j}$ for all cubes $c_{j} \in F$

ㅁ Note that a cover of a unate function is not necessarily unate! (However, there exists a unate cover for a unate function.)

## Special Functions

$\square$ Example


## Unate Recursive Paradigm

$\square$ Key pruning technique is based on exploiting the properties of unate functions
$\square$ based on the fact that unate leaf cases can be solved efficiently
$\square$ New case splitting heuristic
$\square$ splitting variable is chosen so that the functions at lower nodes of the recursion tree become unate

## Unate Recursive Paradigm

$\square$ Unate covers F have many extraordinary properties:

- If a prime cover $F$ is minimal with respect to singlecube containment, all of its cubes are essential primes $\square$ In this case $F$ is the unique minimum cube representation of its logic function
- A unate cover represents a tautology iff it contains a cube with no literals, i.e., a single tautologous cube
$\square$ This type of implicit enumeration applies to many subproblems (prime generation, reduction, complementation, etc.). Hence, we refer to it as the Unate Recursive Paradigm.


## Unate Recursive Paradigm

1. Create cofactoring tree stopping at unate covers

- choose, at each node, the "most binate" variable for splitting
- iterate until no binate variable left (unate leaf)

2. "Operate" on the unate cover at each leaf to obtain the result for that leaf . Return the result
3. At each non-leaf node, merge (appropriately) the results of the two children.

[^0]
## Unate Recursive Paradigm

$\square$ Binate select heuristic
$\square$ Tautology checking and other programs based on the unate recursive paradigm use a heuristic called BINATE SELECT to choose the splitting variable in recursive Shannon expansion
-The idea is, for a given cover $F$, choose the variable which occurs, both positively and negatively, most often in the cubes of $F$

## Unate Recursive Paradigm

## Unate Recursive Paradigm

$\square$ Binate select heuristic

- Example

Unate and non-unate covers.


- Binate variables of a cover are those with both 1's and 0's in the corresponding column
- In the unate recursive paradigm, the BINATE_SELECT heuristic chooses a (most) binate variable for splitting, which is thus eliminated from the sub-covers


## Unate Recursive Paradigm <br> Unate Reduction

$\square$ Let $F(x)$ be a cover. Let $(a, c)$ be a partition of the variables $x$, and let

$$
F=\left[\begin{array}{c:c}
A & C \\
\hdashline T & F^{*}
\end{array}\right]
$$

where

1. the columns of $A$ (a unate submatrix) correspond to variables $a$ of $x$
2. T is a matrix of all "-"s
$\square$ Theorem. Assume $A \neq 1$. Then $F \equiv 1 \Leftrightarrow F^{*} \equiv 1$

## -Example


$f=a c \bar{e}+\bar{c} d+b c \bar{d} e$


50

## Unate Recursive Paradigm <br> Unate Reduction

-Example

$$
F=\left[\begin{array}{c:c}
A & C \\
\hline T & F^{*}
\end{array}\right]
$$



We pick for the partitioning unate variables because it is easy to decide that $A \neq 1$

## Unate Recursive Paradigm Unate Reduction

- Example

- Assume $A_{1}$ and $A_{2}$ are unate and have no row of all "-"s.
- Note that $A_{3}$ and $A_{4}$ are unate (single-row sub-matrices)
- Consequently only have to look at $D_{1}$ to test if this is a tautology


## Unate Recursive Paradigm <br> Unate Reduction

Theorem:$$
F=\left[\begin{array}{c:c}
A & C \\
\hdashline T & F^{*}
\end{array}\right]
$$

Let $A$ be a non-tautological unate matrix $(A \neq 1)$ and T is a matrix of all -'s. Then $F \equiv 1 \Leftrightarrow F^{*} \equiv 1$.

Proof:
If part: Assume $\mathrm{F}^{*} \equiv 1$. Then we can replace $\mathrm{F}^{*}$ by all -'s. Then last row of F becomes a row of all "-"s, so tautology.

## Unate Recursive Paradigm <br> Unate Reduction

$\boldsymbol{\square}$ Proof (cont'd):
Only if part: Assume $F^{*} \neq 1$. Then there is a minterm $\mathrm{m}_{2}$ (in $c$ variables) such that $F^{*}{ }_{m_{2}}=0$ (cofactor in cube), i.e. $\mathrm{m}_{2}$ is not covered by $F^{*}$.
Similarly, $\mathrm{m}_{1}$ (in a variables) exists where $A_{\mathrm{m}_{1}}=0$,
i.e. $m_{1}$ is not covered by $A$. Now the minterm
$\mathrm{m}_{1} \mathrm{~m}_{2}$ (in the full variable set) satisfies $F_{\mathrm{m}_{1} \mathrm{~m}_{2}}=0$.
Since $\mathrm{m}_{1} \mathrm{~m}_{2}$ is not covered by $F, F \neq 1$.

## Unate Recursive Paradigm

Application - Tautology Checking

```
\square Improved tautology check
Algorithm CHECK_TAUTOLOGY(List_of_Cubes C) {
    if(C == \varnothing) return FALSE;
    if(C == {-..-}) return TRUE; // cube with all '-'
    C = UNATE_REDUCTION(C)
    xi = BINATE_SELECT(C)
    C0 = COFACTOR(C,\negxi)
    if(CHECK_TAUTOLOGY(C0) == FALSE) {
        return FALSE
    }
    C1 = COFACTOR(C,xi)
    if(CHECK_TAUTOLOGY(C1) == FALSE) {
        return FALSE;
    }
    return TRUE;
}
```


## Unate Recursive Paradigm <br> Application - Tautology Checking

$\square$ Example


## Unate Recursive Paradigm <br> Application - Complement

$\square$ We have shown how tautology check (SAT check) can be implemented recursively using the Binary Decision Tree
$\square$ Similarly, we can implement Boolean operations recursively, e.g. the COMPLEMENT operation:
$\square$ Theorem.

$$
\bar{f}=x \cdot \bar{f}_{x}+\bar{x} \cdot \bar{f}_{\bar{x}}
$$

- Proof.

$$
\begin{aligned}
& g=x \cdot \bar{f}_{x}+\bar{x} \cdot \bar{f}_{\bar{x}} \\
& f=x \cdot f_{x}+\bar{x} \cdot f_{\bar{x}} \\
& \left.\begin{array}{l}
f \cdot g=0 \\
f+g=1
\end{array}\right\} \Rightarrow g=\bar{f}
\end{aligned}
$$

## Unate Recursive Paradigm <br> Application - Complement

$\square$ Efficient complement of a unate cover

```
```

Algorithm COMPLEMENT(List_of_Cubes C) {

```
```

Algorithm COMPLEMENT(List_of_Cubes C) {
if(C contains single cube c) {
if(C contains single cube c) {
Cres = complement_cube(c) // generate one cube per
Cres = complement_cube(c) // generate one cube per
return Cres // literal l in c with \negl
return Cres // literal l in c with \negl
}
}
else {
else {
xi = SELECT_VARIABLE(C)
xi = SELECT_VARIABLE(C)
C0 = COMPLEMENT(COFACTOR(C,\negxi)) ^ \negxi
C0 = COMPLEMENT(COFACTOR(C,\negxi)) ^ \negxi
C1 = COMPLEMENT(COFACTOR(C,xi)) ^ xi
C1 = COMPLEMENT(COFACTOR(C,xi)) ^ xi
return OR(C0,C1)
return OR(C0,C1)
}
}
}

```
```

}

```
```


## Unate Recursive Paradigm <br> Application - Complement

- Complement operation on cube list
$\square$ Idea:
- variables appear only in one polarity on the original cover $(a b+b c+a c)^{\prime}=\left(a^{\prime}+b^{\prime}\right)\left(b^{\prime}+c^{\prime}\right)\left(a^{\prime}+c^{\prime}\right)$

■ when multiplied out, a number of products are redundant $a^{\prime} b^{\prime} a^{\prime}+a^{\prime} b^{\prime} c^{\prime}+a^{\prime} c^{\prime} a^{\prime}+a^{\prime} c^{\prime} c^{\prime}+b^{\prime} b^{\prime} a^{\prime}+b^{\prime} b^{\prime} c^{\prime}+b^{\prime} c^{\prime} a^{\prime}+b^{\prime} c^{\prime} c^{\prime}=$ $a^{\prime} b^{\prime}+a^{\prime} c^{\prime}+b^{\prime} c^{\prime}$

- we just need to look at the combinations for which the variables cover all original cubes (see the following example)
$\square$ this works independent of the polarity of the variables because of symmetry to the ( $1,1,1, \ldots, 1$ ) case (see the following example)


## Unate Recursive Paradigm <br> Application - Complement

- Map (unate) cover matrix F into Boolean matrix B

convert: " 0 "," " " in F to " 1 " in B (literal is present)

$$
\text { "-" in F to " } 0 \text { " in } \mathrm{B} \quad \text { (literal is not present) }
$$

## Unate Recursive Paradigm <br> Application - Complement

$\square$ Find all minimal column covers of B.

- A column cover is a set of columns J such that for each row $i$, $\exists \mathrm{j} \in \mathrm{J}$ such that $\mathrm{B}_{\mathrm{ij}}=1$
- Example
$\{1,4\}$ is a minimal column cover for matrix $B$


All rows "covered" by at least one 1

## Unate Recursive Paradigm <br> Application - Complement

$\square$ For each minimal column cover create a cube with opposite column literal from F

- Example

By selecting a column cover $\{1,4\}$, a'd is a cube of $\mathrm{f}^{\prime}$

| 1 | 2 | 3 | 4 | 5 |
| :---: | :---: | :---: | :---: | :---: |
| $a$ | $b$ | $c$ | $d$ | $e$ |
| - | 1 | - | 0 | - |
| - | - | 0 | 0 | 1 |
| 1 | 1 | - | - | 1 |
| 1 | - | 0 | - | 1 |

## Unate Recursive Paradigm Application - Complement

$\square$ The set of all minimal column covers $=$ cover of $\bar{f}$
$\square$ Example

| $a$ | $b$ | $c$ | $d$ | $e$ |
| :---: | :---: | :---: | :---: | :---: |
| - | 1 | - | 0 | - |
| - | - | 0 | 0 | 1 |
| 1 | 1 | - | - | 1 |
| 1 | - | 0 | - | 1 |$\quad \longrightarrow$| $a$ | $b$ | $c$ | $d$ | $e$ |
| :---: | :---: | :---: | :---: | :---: |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |

- $\{(1,4),(2,3),(2,5),(4,5)\}$ is the set of all minimal covers. This translates into:

$$
\bar{f}=\bar{a} d+\bar{b} c+\overline{b e}+d \bar{e}
$$

## Unate Recursive Paradigm <br> Application - Complement

ㅁ Theorem (unate complement theorem):
Let $F$ be a unate cover of $f$. The set of cubes associated with the minimal column covers of $B$ is a cube cover of $f$.

- Proof:

We first show that any such cube c generated is in the offset of $f$, by showing that the cube $c$ is orthogonal with any cube of $F$.

- Note, the literals of $c$ are the complemented literals of $F$. $\square$ Since $F$ is a unate cover, the literals of $F$ are just the union of the literals of each cube of F).
- For each cube $m_{i} \in F, \exists j \in J$ such that $B_{i j}=1$.
$\square \mathrm{J}$ is the column cover associated with c .
- Thus, $\left(m_{i}\right)_{j}=x_{j} \Rightarrow c_{j}=\bar{x}_{j}$ and $\left(m_{i}\right)_{j}=\bar{x}_{j} \Rightarrow c_{j}=x_{j}$. Thus $\mathrm{m}_{\mathrm{i}} \mathrm{c}=\varnothing$. Thus $\mathrm{c} \subseteq \mathrm{f}$.


## Unate Recursive Paradigm <br> Application - Complement

ㅁ Proof (cont'd)
We now show that any minterm $m \in \overline{\mathrm{f}}$ is contained in some cube c generated:

- First, $m$ must be orthogonal to each cube of $F$
$\square$ For each row of $F$, there is at least one literal of $m$ that conflicts with that row.
- The union of all columns (literals) where this happens is a column cover of B
- Hence this union contains at least one minimal cover and the associated cube contains m.


## Unate Recursive Paradigm <br> Application - Complement

$\square$ The unate covering problem finds a minimum column cover for a given Boolean matrix B

- Unate complementation is one application based on the unate covering problem
- Unate Covering Problem:

Given a matrix $B$, with $B_{i i} \in\{0,1\}$, find $x$, with $\mathrm{x}_{\mathrm{i}} \in\{0,1\}$, such that $\mathrm{Bx} \geq 1$ (componentwise inequality) and $\Sigma_{j} \mathrm{x}_{\mathrm{j}}$ is minimized

- Sometimes we want to minimize

$$
\Sigma_{\mathrm{j}} \mathrm{c}_{\mathrm{j}} \mathrm{x}_{\mathrm{j}}
$$

where $c_{j}$ is a cost associated with column $j$


[^0]:    $\square$ Main idea: "Operation" on unate leaf is computationally less complex
    $\square$ Operations: complement, simplify, tautology, prime generation,

