VLSI CAD: logic to layout

This is a note for a lesson as the tittle: VLSI CAD.

Lecture 1 to lecture ?? are logic part,

Lec 1 Computational Boolean Algebra

1. Boolean Algebra Basement

（1）计算需求：少数变量的布尔代数式可以利用卡诺图化简，但对于多变量的布尔代数式，无法有效化简，需要有效的计算策略去优化计算：分解、计算、应用

> 数学上对于复杂非线性函数，可以进行泰勒展开、傅里叶变换（周期函数）等。

（2）解决方法：香农展开定理

设$F(A,B,C)$是一个布尔函数，A,B,C是二值变量，则F可以展开为：

$$F(A,B,C) = A \cdot F(A=1,B,C) + \bar{A} \cdot F(A=0,B,C)$$
$$= A \cdot F_A + \bar{A} \cdot F_{\bar{A}}$$

> 定理应用：
>
> 1. 简化复杂函数式，以便于简化电路设计模式
> 2. 逻辑隔离：当电路故障时，通过检查$F_A, F_{A'}$，可以确定是B,C出错还是A出错。
> 3. 展出电路：分离出的 A和 A'与简化的 $F(A / A', B, C)$是与逻辑
> 4. 逻辑优化：可以帮助找出多变量函数的最小化逻辑式

例：对四元布尔逻辑式进行香农展开得到：

$$F(x,y,z,w) = x \cdot y \cdot F_{xy} + x \cdot y' \cdot F_{xy'} + x' \cdot y \cdot F_{x'y} + x' \cdot y' \cdot F_{x'y'}$$

其中F_{xy}等二级逻辑式是z,w的函数，也被称为$cofactors$。

2. Boolean Difference

> 也叫布尔差分、布尔导数。

（1）辅助因子cofactor的性质和作用：

1. 问题引入：复合函数的cofactors与其组成函数的cofactors的关系是什么？

$$Q: if\ H(x)\ is\ a\ function, F(x)、G(x)\ also\ are.$$

$$(1)H=\bar{F} \qquad (2)H=F\cdot G$$

$$(3)H=F+G \quad (4)H=F\oplus G$$

$$What's\ the\ relationship\ between\ H's\ cofactors\ and\ F、G's?$$

$$\star\ cofactor\ 性质\ 1：(F')_x=(F_x)'=\bar{F}_x$$

$$if\ H=F\oplus G,\ H'=(F\oplus G)'=F'\oplus G'$$

（2）布尔导数/差分：

$$Boolean\ derivatives\ (布尔导数)：$$

$$\frac{\partial F}{\partial x}=F_x\oplus F_{x'}$$

布尔导数的性质：

$$(1)\frac{\partial F}{\partial x\partial y}=\frac{\partial F}{\partial y\partial x} \quad (2)\frac{\partial(F\oplus G)}{\partial x}=\frac{\partial F}{\partial x}\oplus\frac{\partial G}{\partial x}$$

$$(3)if\ F=constant,\ for\ any\ x,\ \frac{\partial F}{\partial x}=0$$

$$but\ for\ AND\ and\ OR\ logic,\ it\ doesn't\ work\ the\ same.$$

$$because\ bool\ logic\ for\ and\ \&\ or\ not\ always\ act\ as\ real\ numbers.$$

（3）Gate-level View：

$$(1)NOT\ Gate:\ f=\bar{x}$$

$$f_x=1,\ f_{x'}=0\ \Rightarrow\ \frac{\partial f}{\partial x}=1\oplus 0=1$$

$$(2)AND\ Gate:\ f=x\cdot y$$

$$f_x=y,\ f_{x'}=0\ \Rightarrow\ \frac{\partial f}{\partial x}=y\oplus 0=y$$

$$(3)OR\ Gate:\ f=x+y$$

$$f_x=1,\ f_{x'}=y\ \Rightarrow\ \frac{\partial f}{\partial x}=1\oplus y=\bar{y}$$

$$(4)NOR\ Gate:f=x\oplus y$$

$$f_x=\bar{y},\ f_{x'}=y\ \Rightarrow\ \frac{\partial f}{\partial x}=1\oplus 0=1$$

$$When\ \partial f/\partial x=1, it\ means\ if\ x\ changes, y\ changes.$$

（4）意义：有效解释了数字系统输入的变化如何引起输出的变化，得以求出哪些输入变量会引起输出的变化。

对一个数字系统加入一个输入 X，使得

$$\partial f/\partial X=1$$

那么输入 X 的任何变化都会强制使输出函数 f 发生变化。

3. Quantification Operators

> 计算布尔代数中一些 *Cofactor*的组合可以做一些有趣的事情：量化操作符。量化操作符用于处理布尔表达式中的变量，根据这些变量的所有可能值来评估整个表达式的真值。主要分两种：全称量化（*Universal Quantification*）和存在量化（*Existential Quantification*）。

（1）全称量化（Universal Quantification）：用符号 ∀ 表示，读作“ for all ”。在布尔代数中，全称量化用于声明一个属性对所有变量都成立。例如，全称量化 ∀x F(x) 的意思是“对于所有的 x，F(x) 都为真”。

在布尔代数中，全称量化通常与逻辑与（AND）操作符结合使用。如果我们要表达“ x，P(x) 都为真”，我们可以通过逻辑与操作符将 P(x) 对于所有 x 的值结合起来。

$$AND\ Quantification\ Operator: F_{x_i}\cdot F_{x_i'}$$

$$\forall x_i\ F(x_1,x_2,\dots,x_{i-1},x_{i+1},\dots,x_n)$$

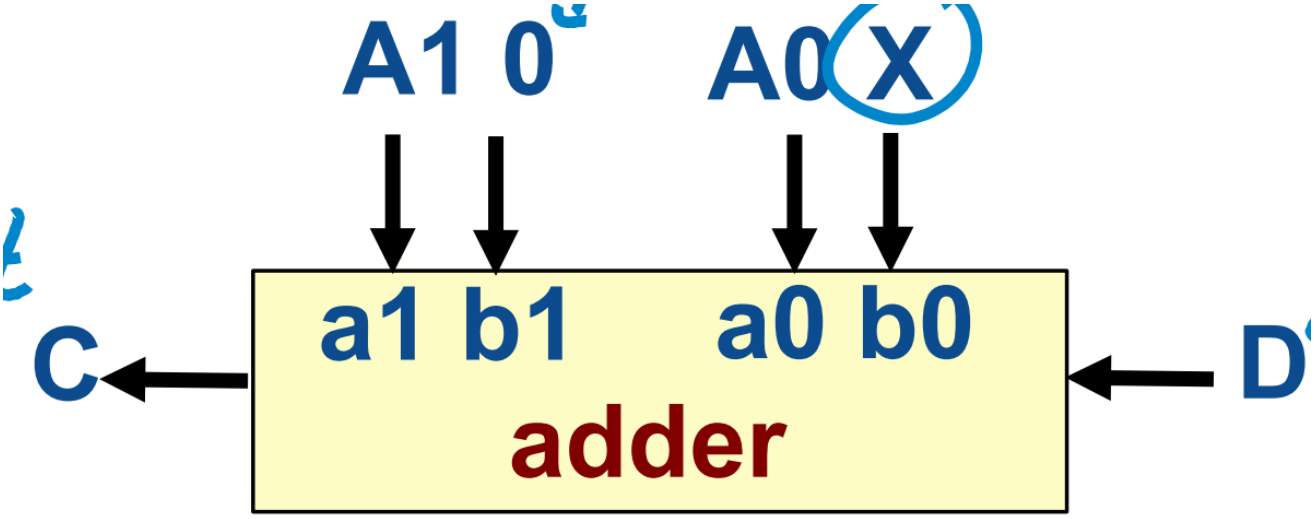
$$read\ as\ 〃\ for\ all\ x_i,\ F(\dots,x_{i-1},x_{i+1},\dots)\ is\ true.〃$$

（2）存在量化（Existential Quantification）： 用符号 ∃ 表示，读作"exist"。在布尔代数中，存在量化于声明至少存在一个变量使得属性成立。例如，存在量化 ∃x P(x) 的意思是“存在至少一个 x 使得 P(x) 为真”。

在布尔代数中，存在量化通常与逻辑或（OR）操作符结合使用。如果我们要表达“存在至少一个 x 使得 P(x) 为真”，我们可以通过逻辑或操作符将 P(x) 对于至少一个 x 的值结合起来。

$$OR\ Quantification\ Operator: F_{x_i} + F_{x_i'}$$
$$\exists x_i\ F(x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$$
$$read\ as\ "exist\ x_i,\ F(\ldots, x_{i-1}, x_{i+1}, \ldots)\ is\ true."$$

（3）量化举例：如图是一个2-bit加法器，进位输入D，2-bit的加数A_1A_0、B_1B_0，进位输出C。其中B_1为0，B_0为未知数X（0/1），即未知数只有X和D。



C的输出表达式：$C = A_1A_0X + A_1(A_0 + X)D$

$$C_{A_1A_0} = X + D \quad C_{A_1A_0'} = X \cdot D \quad C_{A_1'A_0} = 0 \quad C_{A_1'A_0'} = 0$$

1. 全称量化：

$$\forall A_1, A_0\ C(X, D) = C_{A_1A_0} \cdot C_{A_1A_0'} \cdot C_{A_1'A_0} \cdot C_{A_1'A_0'} = 0$$
$$So\ for\ all\ A_1、A_0, NO\ value\ of\ X, D\ could\ make\ C = 1$$
$$(there\ must\ be\ a\ A_1, A_0\ at\ least\ could\ make\ C = 0)$$

2. 存在量化：

$$\exists A_1, A_0\ C(X, D) = C_{A_1A_0} + C_{A_1A_0'} + C_{A_1'A_0} + C_{A_1'A_0'} = X + D$$
$$So\ for\ some\ A_1、A_0, at\ least\ X\ or D\ is\ 1\ could\ make\ C = 1$$
$$(X或D其中一个为1时，\exists A_1, A_0\ C(X, D) = 1，C可以为1)$$

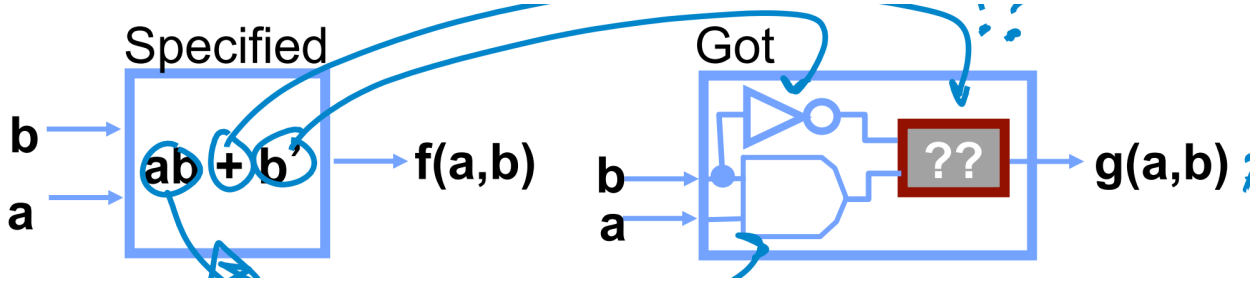
（4）总结：量化操作符就是一个逻辑推理问题。当出现问题时，可以通过对单一变量进行量化，从而检查是否此变量出现问题。

4. Application to Logic Network Repair

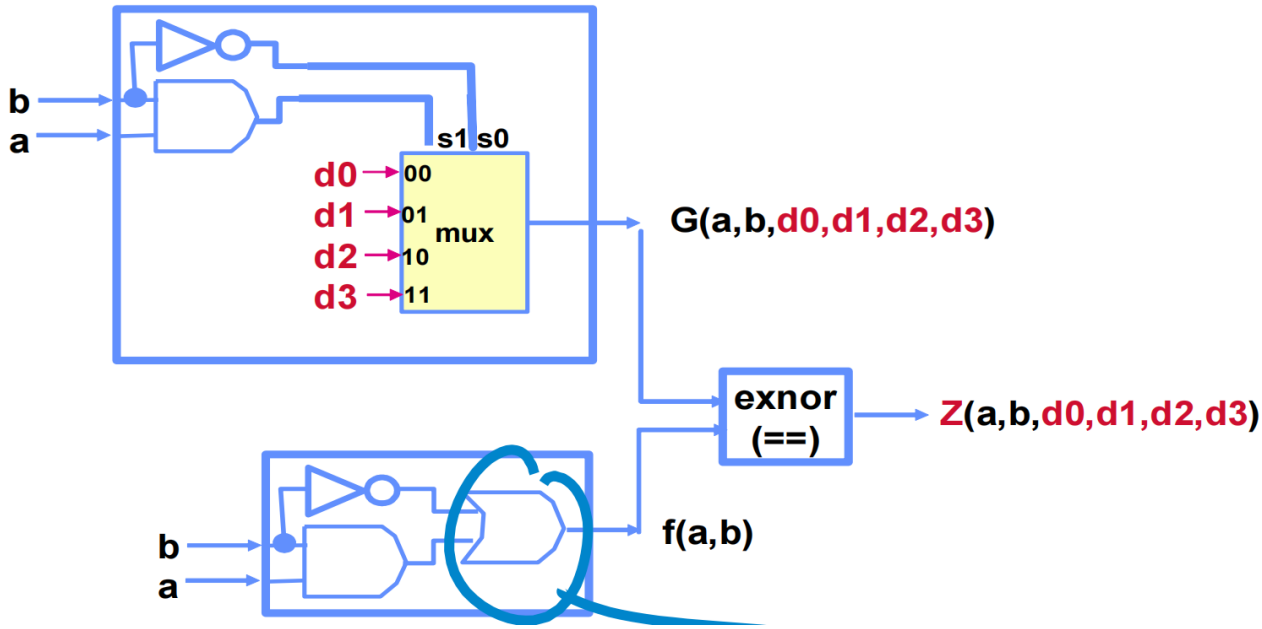
（1）逻辑网络修复思想：对原有变量进行量化。

对于下图的简单逻辑网络有：

$$f(a,b) = ab + b'$$



假如已知其第三个逻辑门发生损坏，且不知道是什么门。同时我们还有另一个逻辑正确的门电路。以4选1多路选择器MUX替代损坏逻辑门，再将输出函数G与正确的函数f(a,b)进行异或非操作：



> *EX-NOR*（异或非）：当输入一致时输出*1*，不同时输出*0*.

$$Goal: find\ out\ inputs(d_0,d_1,d_2,d_3)\ to\ make$$
$$\forall a,b\ Z(d_0,d_1,d_2,d_3)==1$$
$$(Z==1\ means\ G=f(a,b),so\ the\ logic\ network\ is\ fixed)$$
$$For\ MUX: G=d_0s_1's_0'+d_1s_1's_0+d_2s_1s_0'+d_3s_1s_0$$

$$Z=G\bar{\oplus}f\ \Rightarrow\ Z_{ab}=(G\bar{\oplus}f)_{ab}=G_{ab}\bar{\oplus}f_{ab}$$
$$Z_{ab}=d_2\quad Z_{ab'}=d_1\quad Z_{a'b}=\bar{d_0}\quad Z_{a'b'}=d_1$$
$$\forall a,b\ Z(d_0,d_1,d_2,d_3)=Z_{ab}\cdot Z_{ab'}\cdot Z_{a'b}\cdot Z_{a'b'}=\bar{d_0}d_1d_2$$
$$if\ (d_0,d_1,d_2,d_3)=(0,1,1,X),\ Z==1,\ the\ network\ repaired.$$
$$So\ the\ broken\ gate\ might\ be\ an\ OR\ gate.$$

5. Recursive Tautology

> 递归永真式（恒真命题）

（1）构建一个数据结构：对所有输入，当 **f( )==1** 时，算法输出yes，反之输出no

对于一个多变量输入的函数，这显然不是一个简单的任务。

（2）boolean cube and PCN representation：对于变量可视化表示的一种方法。

$1.Boolean\ Cube$：对于一个三变量函数$f(A,B,C)$，其输入不同组合有2^3种

即$000,001,010,011,100,101,110,110$

可构建一个正方体，以A,B,C为三维直角坐标系。各顶点即为输入的不同组合.

在各顶点处写上对应值输入$f(A,B,C)$的输出值，得到对应的$Boolean\ Cube$.

$2.Positional\ Cube\ Notation(PCN)$：对于三变量函数$f(A,B,C)$

分别以$01\Rightarrow x,10\Rightarrow x',11\Rightarrow no\ x\ or\ x'$表示函数中各项的组成部分

例如：$f(A,B,C)=AB'+A'C$：

$$A\Rightarrow 01,B'\Rightarrow 10,no\ C\Rightarrow 11,so\ AB'=[01\ 10\ 11]$$
$$so\ A'C=[10\ 11\ 01]$$
$$f(A,B,C)=[01\ 10\ 11],[10\ 11\ 01]$$

（3）永真式的检查方法：

$$Great\ result：当且仅当f_x和f'_x都恒真时，f(x)为恒真.$$
$$If\ f()=1, cofactors\ both=1$$
$$If\ both\ cofactors=1,\ f=x\cdot F_x+x'\cdot F_{x'}=x\cdot 1+x'\cdot 1=1$$
$$So\ if\ you\ can't\ tell\ f==1, you\ can\ try\ to\ see\ each\ cofactor==1.$$

（4）URP 实现（Unate Recursive Paradigm）：<u>没学懂</u>

$$1.Cofactor分解规则：将要分解的对象的Cube\ list值设为11$$
$$例如：对于f(a,b,c)=ab+b'c=[01,01,11],[11,10,01]$$
$$f_a:[01,01,11]\Rightarrow[11,01,11](set\ the\ first\ 01\ to\ 11)$$
$$[11,10,01]\ don't\ need\ to\ set$$
$$f_b:[01,01,11]\Rightarrow[01,11,11](set\ the\ second\ 01\ to\ 11)$$
$$[11,10,01]\ need\ to\ set\ as\ [11,11,01]$$

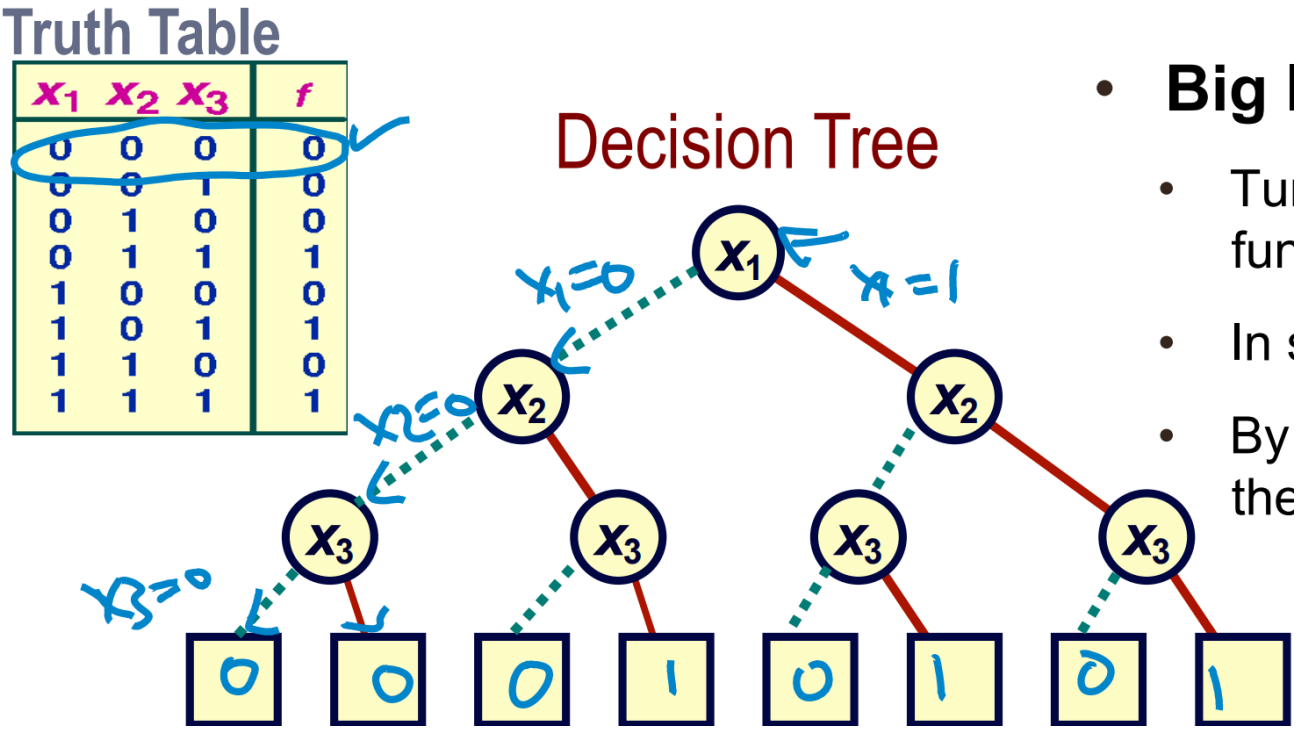
$$2.Unate\ function\ rules：单调函数规则$$
$$(1)positive\ unate(单调不减)：变量0\Rightarrow 1, f\ 0\Rightarrow 1\ or\ constant\ be\ 0/1.$$
$$(2)negative\ unate(单调不增)：变量0\Rightarrow 1, f\ 1\Rightarrow 0\ or\ constant\ be\ 0/1.$$
$$f(a,b,c)=a+bc+ac:for\ a\ var\ is\ unate$$
$$f(a,b,c)=a+b'c+bc:is\ not$$
$$3.An\ important\ rule：(1)如果Cube\ list中全是11，那么单调函数f恒真.$$
$$(2)f=\{无关元素+x+x'+无关元素\}==1$$

Lec 2 计算布尔代数的表示: BDD

> ***Binary Decision Diagrams(BDD)***：二进制决策图。
>
> *URP*是递归永真式中常用的思考方法，但是现代人类通常使用一种新的数据结构——二进制决策图，来有效描述计算布尔代数中的一类问题。

1. Binary Decision Diagrams Basics

（1）BDD: 决策树、决策图



通过设置权重，如 $x_1>x_2>x_3$来进行有向的决策,最终得到$f(x_1,x_2,x_3)$的值。

如上图：节点$(vertex)$表示变量,每条线代表一次决策

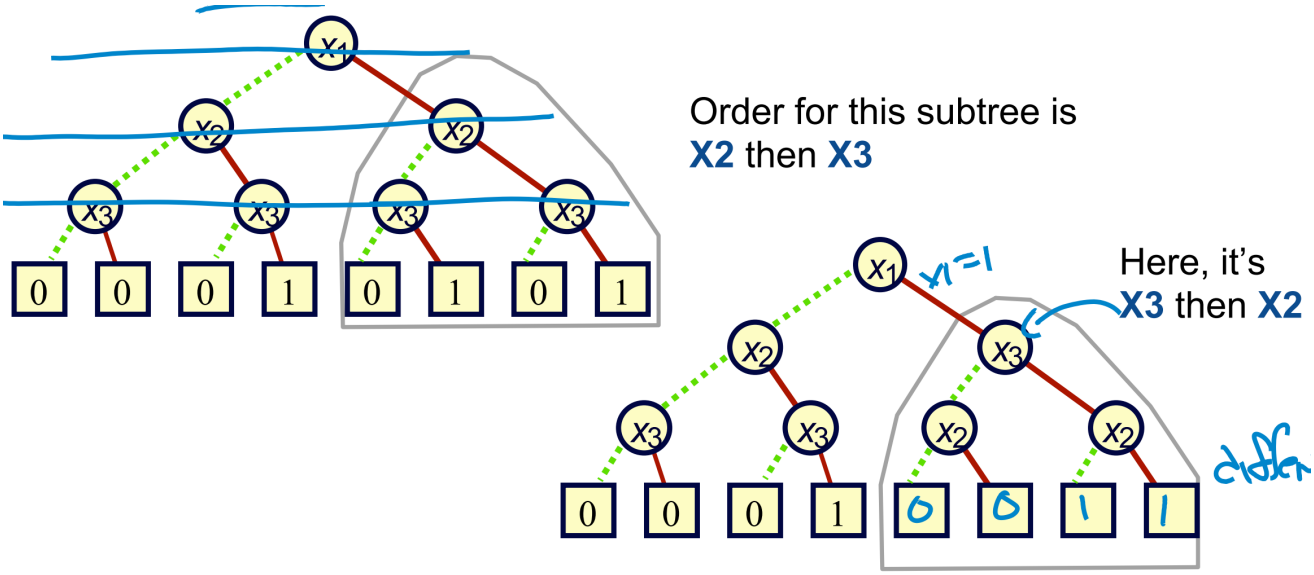
绿色虚线表示节点上的变量为0,红色实线表示为1,即二元决策

无需加上箭头，因为默认由上层往下层进行决策。

1.$variable\ ordering$：不同的决策顺序是允许的，如 $x_1>x_3>x_2$

2.$path$：最顶层的那个节点称为$root$,最底层的节点称为$leaf$,是固定的函数值,由$root$到$leaf$的路径即为$path$.

3.决策图不唯一：不同的决策顺序、如下图所示的逻辑简化都有可能改变决策图



（2）Canonical form：范式

> 普通决策图并不规范，与真值表一样大，实用意义小。

为了使决策图变得更$useful$：

1.规定所有变量的决策顺序：每个路径上的变量都是同一顺序;

2.对于某些不重要的变量可以直接省略：每条路径上，同一变量至多出现一次。

但实际上可省略的部分很少，决策图依旧还不够规范。

3.$Reduction$：删除不必要的节点和冗余的线。

三要素：$Decision$、$Ordering$、$Reduction$

$Why?$①想要得到尽可能小、不能再精简的结果$(Graph)$.

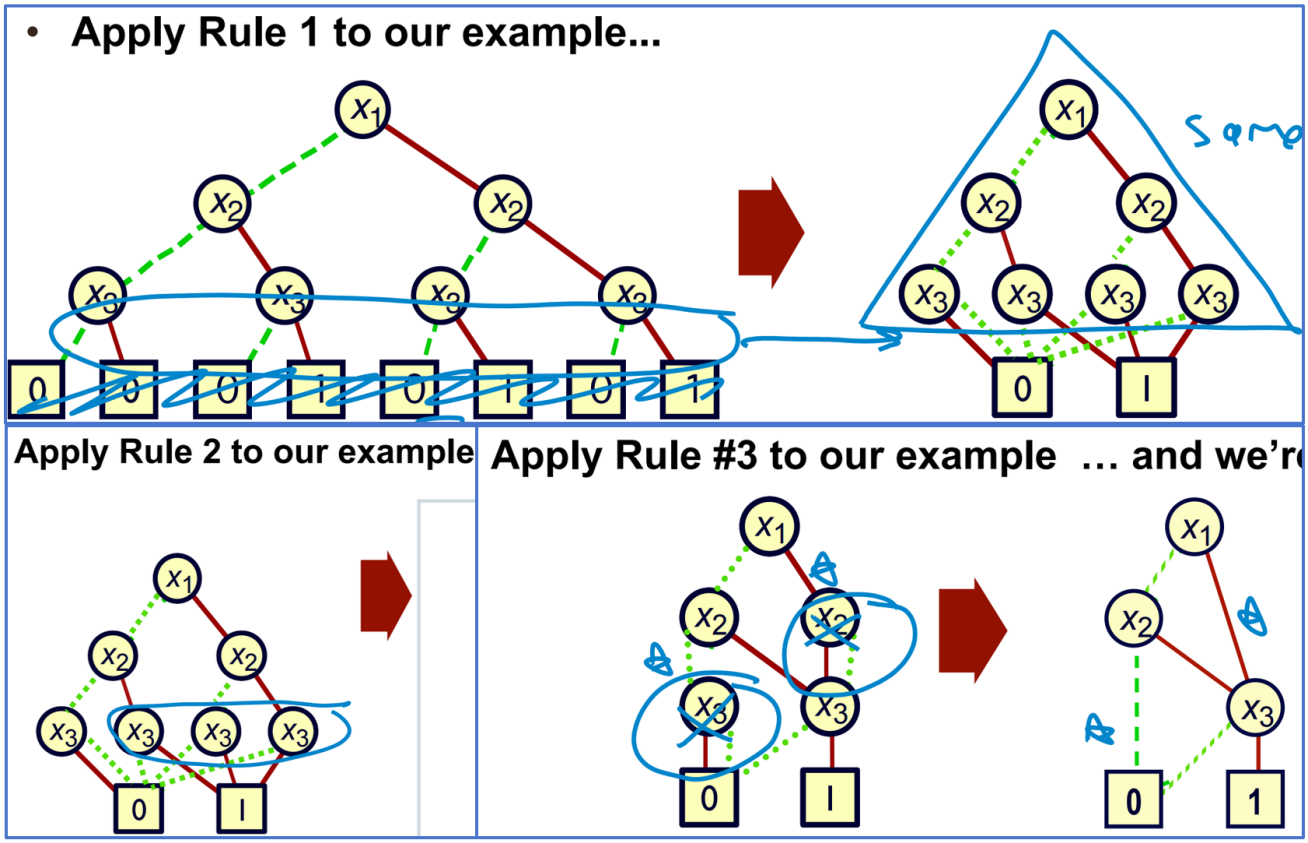
$Reduction\ rules$：

(1)合并等效的节点：主要是将最下层节点的$0/1$整合;

(2)减少同结构的节点：具有相同变量与相同子节点的当前节点可合并;

(3)删除冗余测试：一个节点无论取1还是取0, 都指向同一个节点, 则可省略.

(意味着该节点的取值并不被关心)



（3）Reduce Ordered BDD：

通过以上三条规则的化简得到的决策图即为ROBDD。通过此方法得到的决策图对于同一变量决策顺序的同一函数来说是唯一的。ROBDD是一种数据结构，对于任何函数来说都是一种范式。

(1)当且仅当$ROBDD$完全一致时，两个函数等价相同。

(2)图的最简化形式是一种$Canonical\ form$,且最节省计算资源。

2. BDD Sharing

> 概述：上一部分提到的 *ROBDD*都只有一个 *root*节点，在实际应用中，多个函数输入的数字系统总是会产生多个 *root*节点，但这多个节点产生的决策图可能会存在相互包含的逻辑部分，因此可以利用二进制决策图共享技术来简化多输入系统的决策图，使整个系统的决策图数量和节点数量大幅度减少。

（1）BDD表示的概念：

1.对于一个函数$f(x_1,x_2,\ldots,x_n)=1$：其$ROBDD$表示为

f——[1]

2.对于一个函数$f(x_1,x_2,\ldots,x_n)=0$：其$ROBDD$表示为

f——[0]

对于一个函数$f(x_1,x_2,..,x,..,x_n)=x$：其$ROBDD$表示为

f——[x]

/ \

[0] [1]

意味着：f是指向$root$节点的一个指针。

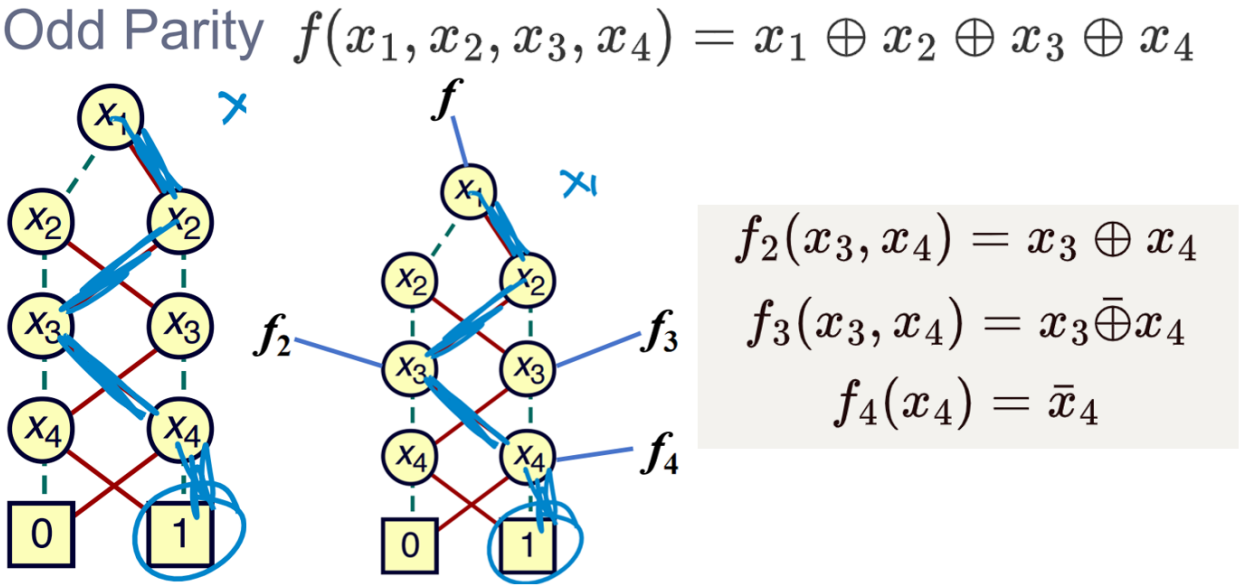
（2）Example：

下图是一个奇偶校验函数的决策图：$f(x_1,x_2,x_3,x_4)=x_1\oplus x_2\oplus x_3\oplus x_4$

假如有一个函数f_2指向左侧的x_3,则$f_2(x_3,x_4)=x_3\oplus x_4$

同理f_3指向右侧x_3,则$f_3(x_3,x_4)=x_3\bar{\oplus}x_4$

$f_4(x_4)=\bar{x}_4$



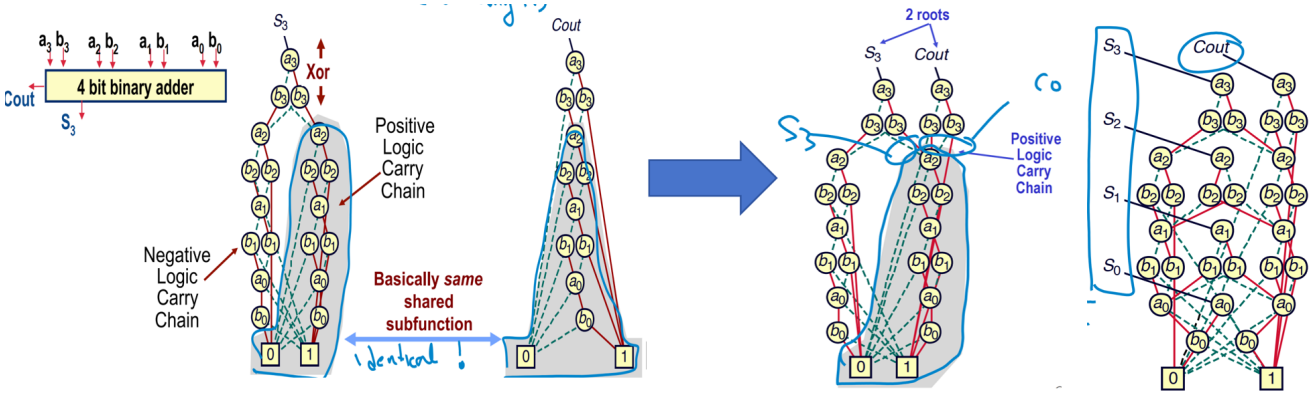
由此，该决策图就具有多个$root$节点。

这个例子所表达的道理：决策图中的每个节点都可以代表某个函数。

（3）更高级的共享：通过接入一部分节点来互相共享某个逻辑

如下图是一个$4\ bit$的全加器，其中S_3是四位输出的最高位，$Cout$是进位输出

此外还有S_2, S_1, S_0三个进位输出未标出



通过观察S_3和$Cout$的$ROBDD$决策图，其中阴影部分是完全重合的

因此可以将此部分进行共享，从而得到一个具有两个$root$节点的决策图。

这种图被称为$multi-rooted\ BDD$，通过尽可能的共享来最小化图尺寸

$Examples:$

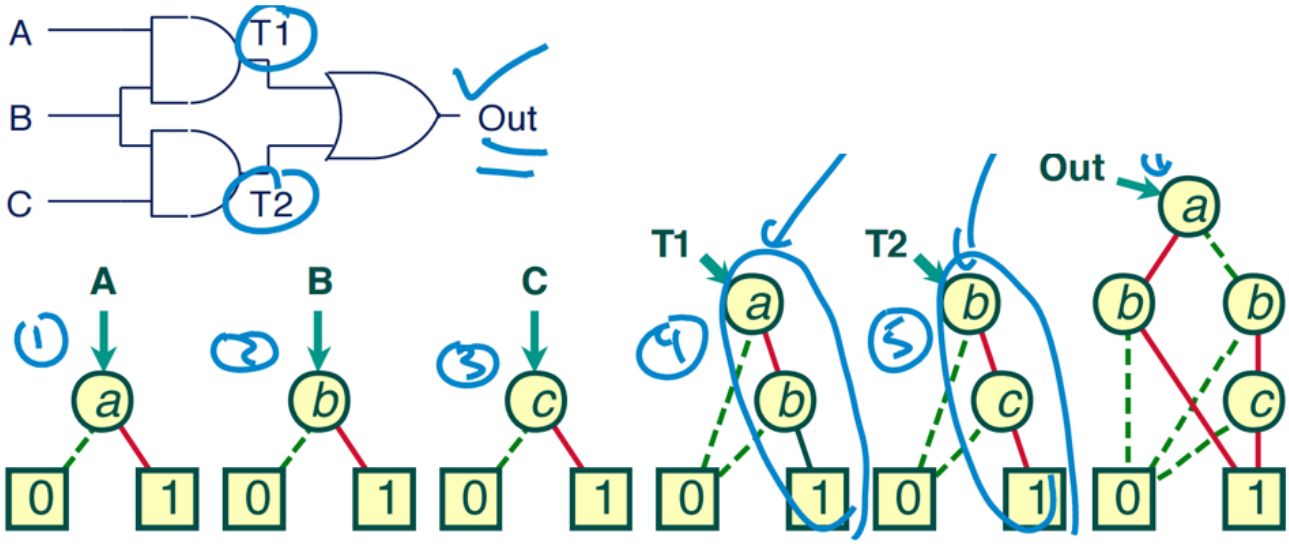
1. $4\ bit\ adder: 51\ nodes,\ after\ shared: 31\ nodes$
2. $64\ bit\ adder: 12481\ nodes,\ after\ shared: 571\ nodes$

有待补充：如何从函数上寻找相同的逻辑进行共享？

3. BDD Ordering

实际上，对于决策图来说，一些变量顺序会使决策图非常简洁，同时某些变量顺序会使决策图成指数级扩张，因此变量顺序$(Ordering)$对于决策图的选择来说是非常重要的。那么，对于一个含有100个变量的决策图，我们怎么可能把决策图画出来，然后再进行简化共享？事实上，BDD通常采用递归的方法来简化。

（1）逐步建立BDD：



如图是一个门级电路，输入是A, B, C，输出是Out.

引入思想：每个输入、输出和中间量都指向一个BDD。

由此可以建立如上图的$BDDs$，以算法逼近的思想来获得最终Out的BDD：

$$A, B, C = CreateVar("a",\ "b",\ "c");$$

$$T_1 = And(a, b);$$

$$T_2 = And(b, c);$$

$$Out = Or(T_1, T_2);$$

对于Out来说，观察到T_1或T_2为1即可，则圈出T_1、T_2为1的a, b, c的路径$(path)$，将两条路径分别指向1，其余路径均设为0即可，得到Out的BDD。

（2）Applications of BDD:

1. 找到使函数F, G输出不同的输入变量：比较BDD的节点分叉。
2. 找到哪些变量的取值使函数为1：从图上为1的$leaf$节点开始向上层节点取值

   例如上面的Out：由1向上有两条$path$，分别是$(c, b, a) = (x, 1, 1), (1, 1, 0)$，

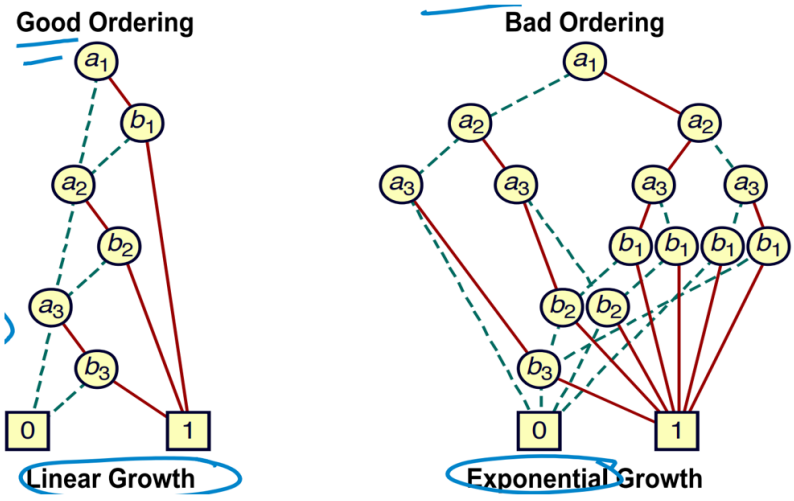
   因此，当a, b, c为$(1, 1, 1), (1, 1, 0), (0, 1, 1)$时，输出函数$Out$为1。
3. 递归检查：用URP来递归检查非常困难，但是对于BDD来说很简单。

（3）BDD Ordering：

BDD似乎好到有些难以置信，但是实际上这与变量顺序息息相关。

例如，对于$f = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$，如下图：

但对于好的变量顺序，节点数呈线性增加；不好的变量顺序，节点数呈指数型增加



对于一些指数型难度的问题，BDD可以轻松解决；问题在于如何简化BDD，只有得到小尺寸的BDD，这些问题才能被轻松解决。有些困难的问题会产生很好很小的决策图，而有些会产生超巨大的决策图。这些问题是没有通用的解决方法的。因此要特事特办，采用合适的解决方法。

1. 变量顺序探索：为合理的问题创建出$a\ nice\ BDD.$
2. 界定问题：总结哪些问题绝不可能创建出一张好的决策图(例如乘法逻辑...)。
3. 动态排序技术：通过BDD软件来选择排序。

介绍一个启发式的变量排序的经验规则：

(1)互相关联的输入应该在顺序上彼此靠近；

(2)能够决定输出的输入变量应该互相靠近，且位于BDD的较上部分。

Lec 3 Satisfiability（SAT）

1. SAT Basics

（1）满足性：Satisfiability

对于一个函数$F(x_1, x_2, \ldots, x_n)$：

SAT是寻找一或多组 $(x_1, x_2, \ldots, x_n)$ 取值，使得$F == 1.$

(满足$F = 1$的一组变量取值具有满足性)

对于求解(或是证明没有解)的问题来说，SAT比BDD更方便简单。

（2）标准SAT格式：CNF（Conjunctive Normal Form）

CNF是一种"或与"逻辑连接式，其标准格式为多个加法式相乘：

例如：$f(a,b,c)=(a+b)(b+c)(\bar{a}+\bar{b}+c)$

其中，$(a+b)$被称为一个子式$(clause)$，子式中的a,b,c等是正文本$(positive)$，$\bar{a},\bar{b}$等是负文本$(negative)$.

非常有效的地方在于：1. 如果想证明$f=0$，证明一个子式为0即可。

2. 想证明$f=1$，必须证明所有子式都为1 (子式中至少一个元素为1)。

对于一个函数：$f(a,b,c)=(a+b)(b+c)(\bar{a}+b)$

$make\ a=1,\ b=0,\ c,\ d\ are\ unassigned:$

1. $(a+b+d):a=1,it's\ SAT;$
2. $(b+c):b=0,c\ is\ unknown,\ it's\ unresolved;$
3. $(\bar{a}+b):0+0,\ not\ SAT;$

（3）SAT以解决问题：

通过决策和推演来逐步确定SAT时各个变量的取值（递归迭代）。

1. 决策：取某个变量为1或0，尽可能的简化CNF函数，或使之成为标准格式。
2. 推演：看此时的函数各子式是否确定，若还不确定，继续取一个变量为1或0。
3. 多次推演直到得出是否SAT。

2. Boolean Constraint Propagation (BCP) for SAT

BCP：布尔约束传播。用于在某些变量固定赋值下的函数值推导，一些必要的赋值被称为约束。

（1）最著名的BCP策略：Unit Clause Rule

单位子式规则：当一个子式不能确定是否 SAT 且只剩一个变量未赋值时，这个子式被称为$Unit$。也就是说，为了使函数SAT，子式中剩下的这个未赋值变量必须使子式为1。

$$Example: f = A \cdot B \cdot C \cdot D \cdot E$$
$$A = x_1 + x_2 + x_4 + x_5$$
$$B = \bar{x}_1 + x_3 + \bar{x}_4 + x_5$$
$$C = \bar{x}_2 + x_3 + \bar{x}_4 + \bar{x}_5$$
$$D = \bar{x}_1 + x_2 + \bar{x}_3 + x_5$$
$$E = \bar{x}_1 + x_2 + \bar{x}_3 + \bar{x}_5$$
$$Let\ x_1 = 1, x_2 = 0,\ try\ to\ make\ f = 1$$
$$A = 1 + \ldots = 1;$$
$$B = 0 + x_3 + \bar{x}_4\ unknown,\ so\ let\ x_3 = 1, B = 1$$
$$so\ C = 1;\ D = 0 + 0 + 0 + x_5,\ so\ let\ x_5 = 1,\ D = 1;$$
$$but\ E = 0,\ the\ equation\ is\ conflict,\ so\ this\ deduction\ error.$$
$$Conflict \Rightarrow UnSAT.$$

这个结果表明前序步骤中出现了赋值的错误，导致函数必不可能取1，因此必须回溯进行重新赋值。且由经验可得，D最后只有$x5$，此时D即为$Unit$，所以赋值错误必定出现在x_5赋值之前。

因此，每次BCP的结果有三种：$SAT, Unresolved, UnSAT$。通过不断试错达成SAT，这种思想也属于递归思想。称之为$DPLL\ Algorithm$。

这种算法的特点：1.可以系统性的完全搜索了所有的变量赋值组合;

2.使用或与逻辑子式相乘(CNF)的形式来大幅度简化计算;

3.使用回溯递归的方法，让计算机不会走重复的路。

现代$SAT\ Solver$的进展：

1. 高效的子式数据结构：搜索更快;
2. 高效的变量选择机制：更聪明的搜索,且能读懂变量背后的含义;
3. 高效的BCP机制;
4. 决策机制：学习那些永远不可能SAT的变量模型，然后避免出现。

3. Using SAT for Logic

（1）BDD与SAT对比：

$BDD\ and\ SAT$：分别对某些问题很有效，但不能完全保证一直有效。

BDD：可以完全表示出函数，但很多情况下不能以合理的计算资源表示出。

$(run\ out\ of\ memory)$

SAT：只能求解函数的满足性，但很多情况不能以合理的计算资源找到SAT.

$(run\ out\ of\ time\ to\ search\ SAT)$

（2）Gates to CNF：

这部分是本节内容的核心，讲述如何从逻辑门中抽象得到或与逻辑的范式。

技巧：一次只为一个门搭建CNF，再以此输出为下一个门的输入继续搭建。

引入逻辑门一致性函数：Gate consistency function / gate satisfiability function

例如，对于与非门$d = \bar{a}\bar{b}$，其一致性函数为：

$$\phi_d = [d == \bar{a}\bar{b}] \Rightarrow \phi_d = d \bar{\oplus} \bar{a}\bar{b}$$

展开得到：$\phi_d = (a+b)(b+d)(\bar{a}+\bar{b}+\bar{d})$

疑问：这是怎么展开的？存在一定的展开规则：

如图，对于各种逻辑门，其一致性函数如下：

z= (x)
(yes this is just a wire)

$$[\overline{x}+z][x+\overline{z}]$$

z=NOR(x1, x2, … xn)

$$\left[\prod_{i=1}^{n}(\overline{xi}+\overline{z})\right]\left[\left(\sum_{i=1}^{n}xi\right)+z\right]$$

product sum

z=OR(x1, x2, … xn)

$$\left[\prod_{i=1}^{n}(\overline{xi}+z)\right]\left[\left(\sum_{i=1}^{n}xi\right)+\overline{z}\right]$$

z=NOT(x)

$$[x+z][\overline{x}+\overline{z}]$$

z=NAND(x1, x2, … xn)

$$\left[\prod_{i=1}^{n}(xi+z)\right]\left[\left(\sum_{i=1}^{n}\overline{xi}\right)+\overline{z}\right]$$

Z=AND(x1, x2, … xn)

$$\left[\prod_{i=1}^{n}(xi+\overline{z})\right]\left[\left(\sum_{i=1}^{n}\overline{xi}\right)+z\right]$$

此外，对于$EXOR$和$EXNOR$函数：

1. 常常使得SAT搜索很困难；　2. 具有较大规模的一致性函数。

（3）Summary：

①SAT在解决满足性问题上已经很大程度地替代了BDD；

②对于大规模的问题，SAT通常更快；

③同样，SAT也无法保证解决问题所用的计算资源是否合理。

Lec 4 2-Level Logic Synthesis

> 逻辑综合：给定逻辑函数，优化逻辑后给出尽量少的硬件表示。

1. Basics

（1）寻找最小的SOP函数式：

$SOP:\ sum\ of\ products$，即多个变量先相乘再相加的函数式。

如$f = A\bar{B} + BD + \bar{C}D.$

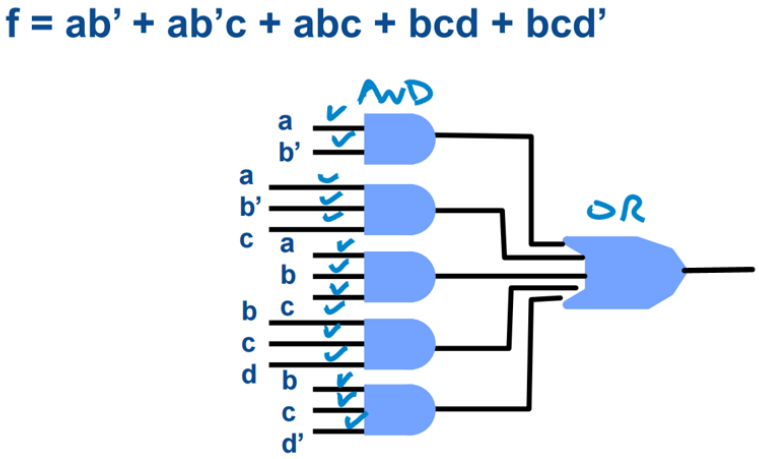
对于一个下图的这个$2\text{–}level\ logic\ gates$：

术语引入：任意一个$input$被称为$literal$，数量与逻辑式中的变量数量一样.

$this\ 2\ level\ logic\ gates\ have\ 14\ literals.$

逻辑综合优化的方向：尽可能减少输出函数式中的 $literal$ 数量。

BUT：已知的方案并不足够有效(布尔代数、$Kmap$等)，复杂性太高。



引入两个非常好的策略$(strategy)$：

1. 不用想着直接得到 $best\ solution$，尝试得到一个$good\ answer$.
2. 迭代改进。从一个解决方案开始，重塑方案以探索得到一个好一些的新方案，多次改进，直到无法继续迭代。

第一，那个一碗饭等于一碗瓜子，我没有内涵你什么，只要你学过初中生物，你就应该晓得饭和瓜子的组成都是淀粉，能量上没有什么差距。

第二，一斤等于多少千克，如果你学过初中物理，那对于这个单位制，我真的搞不懂怎么会有人搞不懂这个单位换算。f=ma，动能公式$W=1/2mv^2$，p=mv等等，他们的单位都是kg。这反映了你生活常识的缺失。

第三，我和你伯娘对你的种种关心爱护你自己应该记得，但这些真不是欠你的。包括但不限于坐车让你坐边上、你喊我就去接你，去杨希家接你等你，给你买手表手机壳，帮你贴膜等等，这只是这个暑假的一部分事情，我觉得不需要再多列举。再往前还有更多更多，你应该明白，这些事情是源于我对你的关心和爱护，而不是我应该做的。

第四，你有将自己当做这个家的一部分吗？这是我一直想问的问题。如果这是你对自己家的态度，那我觉得你在自己家里有点懒了。我可以说，暑假以来我做家务的次数都比你多一些（扫地，洗碗，擦柜子，收碗筷等），而且有时候还喊不动你。我认为你对这个家的付出真的配不上你在这个家里所享受到的，没有尽到自己应有的义务。

第五，你说你的英语有进步，我很欣慰。但是你在家里的表现，我觉得不大对劲。我跟你说过有问题要问我，但是你看看，有问过一次吗。唯一一次还是我看到你那个书上标记的单词意思，是很明显的谬误。这个地方前后衔接都是问题，但是你不以为意。这样不是我希望看到的学习状态。

第六，某些时候其实你很乖的，但是另一些时候你根本就不听话。你应该知道的，你伯娘他们不喜欢小孩一天都塞在屋头，我和罗旺到时候都会出去坐在客厅或者是外面，这是一种生活习惯，也是你伯伯伯娘希望看到的。实际上对于这种类似的问题，我觉得你应该是从来都没有好好想过。

第七，我不理解你在消息里所说的不理你，冷落你等等。你应该知道，人与人之间的关系是互相的，不是我或者谁都得一直热心的对你，并不欠你什么。你在这个家的表现很消极，那你得到的反馈也不可能会是积极的。而且，实际上，你对这些细微事物的敏感度太过了，有时候坦然一些，或许心理上的压抑会少一些。就如你所说，你也不要把我们看的很坏，你所觉得的那种冷落，究其原因是你心灵的脆弱。

假如，我说的假如哈，你前一天说不要管你，那么第二天我们应该以什么样的态度来对你？将心比心，你站到这边的角度想想，除了少说你几句还能做什么呢。有时候你站在你伯伯伯娘的角度想想，理解他们的想法，其实很多事情都会释然。这个真的非常重要，你多想想。

除此之外，你就算再怎么样，也不能给你伯娘发小作文，这是最基本的尊重。对她来说，你这个小作文会非常非常伤她的心，这是因为有代沟或者其他很多原因。沟通应该要慢慢的来，这个事情明明就是你一句话的事情，只要说明是你二嬢说要来接你你才去的就行了。而且这个事情的起因明明是你妈说，你去你家婆那边没和她说，还说你早上说的不去。所以这个真怪不得你伯娘，你妈也有责任。而且你那个小作文，你觉得真的有必要拿之前的那些事情给你伯娘那边发泄出来？你这不是翻旧账吗。说你小气你还不承认，你看你昏了头没？读书读的谨言慎行，凡事三思而后行，你觉得你有理解其中的意思没有？你把这番话说出来，你在这个家里还有容身之地么，或者说，你觉得在你说完这些过后，我们还是应该开开心心的接你回来？别把人不当人了，很多话是不能说出口的，说了后，破镜如何重圆呢。

现在这样，就算我们还是想正常下去，但是每每想起你说的话，是否会寒心呢。

好吧，也就这些了，多的不说，我还是希望你可以在这个家庭里健康茁壮的成长，成长为一个全面发展、自由健康、思想健全的人，为自己和家人而精彩的生活下去。