



功能验证

Functional Verification

定义

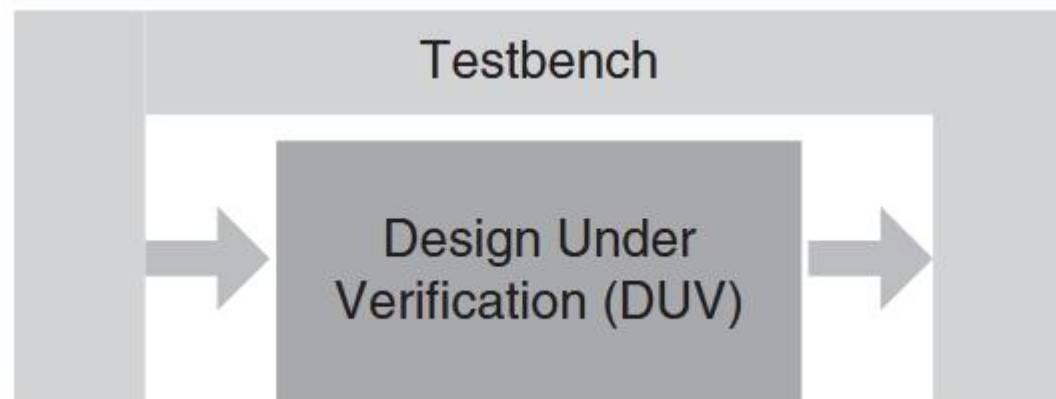
Definition

- the act of reviewing, inspecting, testing, checking, auditing, or otherwise establishing and documenting whether or not items, processes, services or documents conform to specified requirements.
- 审查、检查、测试、审计或以其他方式建立和记录项目、流程、服务或文件是否符合特定要求的行为

测试平台

testbench

测试平台是指用于将预定的输入向量序列应用于被验证设计（DUV）并观察响应的环境。该测试平台对设计意图的某些方面进行建模，并负责将输入序列传送到 DUV 并接收输出响应以供后续分析。



测试平台

testbench

- Stimuli Driver (激励器)
 - 向被测系统提供输入信号 (stimuli)
- Monitor (监视器)
 - 观测被测系统的输入、输出及内部感兴趣的信号;
- Checker (检查器)
 - 验证被测系统的响应是否符合规约




观察点布设方法

Methods of observation points

- 一个测试平台中的 monitor 和 checker 与待测系统中信号变化的观察概念紧密相连。观察方法也将决定用于产生信号的策略。
 - 黑盒方法
 - 白盒方法
 - 灰盒方法




黑盒验证

Black-Box Verification

-  特征：
 - 只观察输入 / 输出接口，不访问待测系统的内部信号（无法看状态寄存器、中间信号等）
-  优点：
 - 与设计实现无关（只关注功能行为）；
 - 验证计划只依赖规范（specification）；
-  缺点：
 - 观察粒度有限，很难精准定位内部 bug；
 - 无法指导测试方向



白盒验证

White-Box Verification

-  特征：
 - 可以访问 DUT 的内部信号、状态、控制路径，验证人员可以基于具体 RTL 实现插入断言、覆盖点
-  优点：
 - 可精确监控 待测系统 内部行为，更容易发现内部隐蔽 bug;
 - 支持复杂结构覆盖率（FSM、path、expression 等）；
-  缺点：
 - 验证平台对设计变更敏感，维护成本高；
 - 可能引入“验证污染”（例如验证逻辑干扰设计）

灰盒验证

Grey-Box Verification

-  特征：
 - 设计内部“只开放关键信号”，而非全暴露；观测点基于验证目标而定
-  优点：
 - 在可控范围内监控内部逻辑，避免过度耦合；；
 - 比黑盒更强大，比白盒更易维护；
 - 实践中最常用，是 UVM 等验证框架推荐做法

半自动化测试

Semi-automated testing

- Golden Vectors (黄金向量)
 - 验证工程师基于规范或原型手动或用工具生成一批黄金输入/输出对 (golden vectors) ;
 - testbench 把输入送入 被测系统, 记录 被测系统的实际输出; 比较实际输出 vs 黄金输出, 若不一致则报告 bug

半自动化测试

Semi-automated testing

- Transaction-Based Checking（基于事务的检查）
 - 定义每个事务
 - Scoreboard（记分板）记录每个预期事务和响应
 - Checker 检查实际事务是否能在 Scoreboard 中找到匹配项，否则报警

自动化测试

Automated testing

- Reference Model (参考模型)
 - 构建一个更高抽象层次的参考实现 (如用 C++/SystemC/ 高层 DSL 编写)
 - 同一个输入向量送入待测系统和参考模型
 - 比较两个模型输出是否一致 (通常是周期级)

自动化测试

Automated testing

- Fuzzing（模糊测试）
 - 给定预期，即DUV应该满足的规范
 - 自动生成大量随机输入，当输出违背预期时报告错误
 - 快速验证实现是否有错误

主要的问题

- 如何产生输入？
- 如何给定预期并捕获错误？
- 什么时候可以停止了？

主要的问题

- 如何产生输入？
- 如何给定预期并捕获错误？
- 什么时候可以停止了？

结构覆盖率指标

Structural coverage metrics

- Line coverage (a.k.a. statement coverage) 行覆盖率
- Branch coverage 分支覆盖
- Path coverage 路径覆盖
- Expression coverage / condition coverage 表达式/条件覆盖率
- Trigger coverage 触发覆盖
- Toggle coverage 翻转覆盖
- FSM coverage 状态机覆盖

有限状态机覆盖率

Finite state machine (FSM) coverage

Table 9.1 Typical Coverage Targets for Different Metrics

<i>Metric</i>	<i>Coverage Goal (%)</i>
Line	100
Branch	100
Condition	60~100
Path	>50
Trigger	100
Toggle	100
FSM (state and arc)	100

行覆盖率

Line coverage (a.k.a. statement coverage)

$$\text{Line Coverage} = \frac{\text{\# of exercised lines in HDL}}{\text{Total \# of lines in HDL}} \times 100\%$$

```
1. always @(in or reset) begin
2.     out = in;
3.     if (reset)
4.         out = 0;
5.     en = 1;
6. end
```

行覆盖率

Line coverage (a.k.a. statement coverage)

$$\text{Line Coverage} = \frac{\text{\# of exercised lines in HDL}}{\text{Total \# of lines in HDL}} \times 100\%$$

```
1. if (x != y)
2.     z = 0;
3. w = z;
```

$$2/3 = 66.6\%$$

行覆盖率

Line coverage (a.k.a. statement coverage)

$$\text{Line Coverage} = \frac{\text{\# of exercised lines in HDL}}{\text{Total \# of lines in HDL}} \times 100\%$$

```
1. if (x != y)
2.     z = 0;
3. w = z;
```

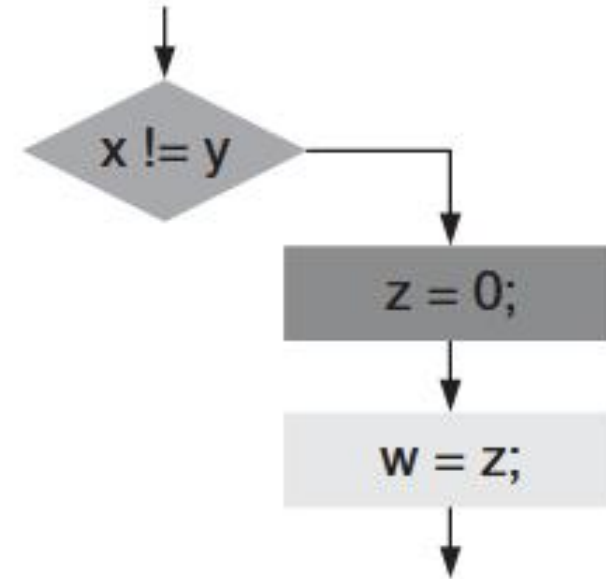
$$3/3 = 100\%$$

分支覆盖率

Branch coverage

$$\text{Branch Coverage} = \frac{\text{\# of exercised branches}}{\text{Total \# of possible branches}} \times 100\%$$

```
1. if (x != y)
2.     z = 0;
3.     w = z;
```

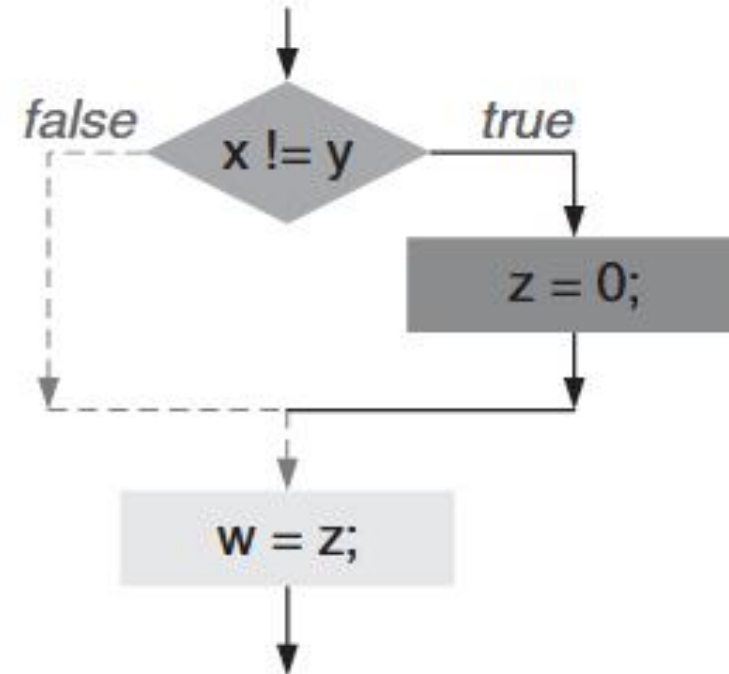


分支覆盖率

Branch coverage

$$\text{Branch Coverage} = \frac{\text{\# of exercised branches}}{\text{Total \# of possible branches}} \times 100\%$$

```
1. if (x != y)
2.     z = 0;
3. w = z;
```



1/2 = 50%

路径覆盖率

Path coverage

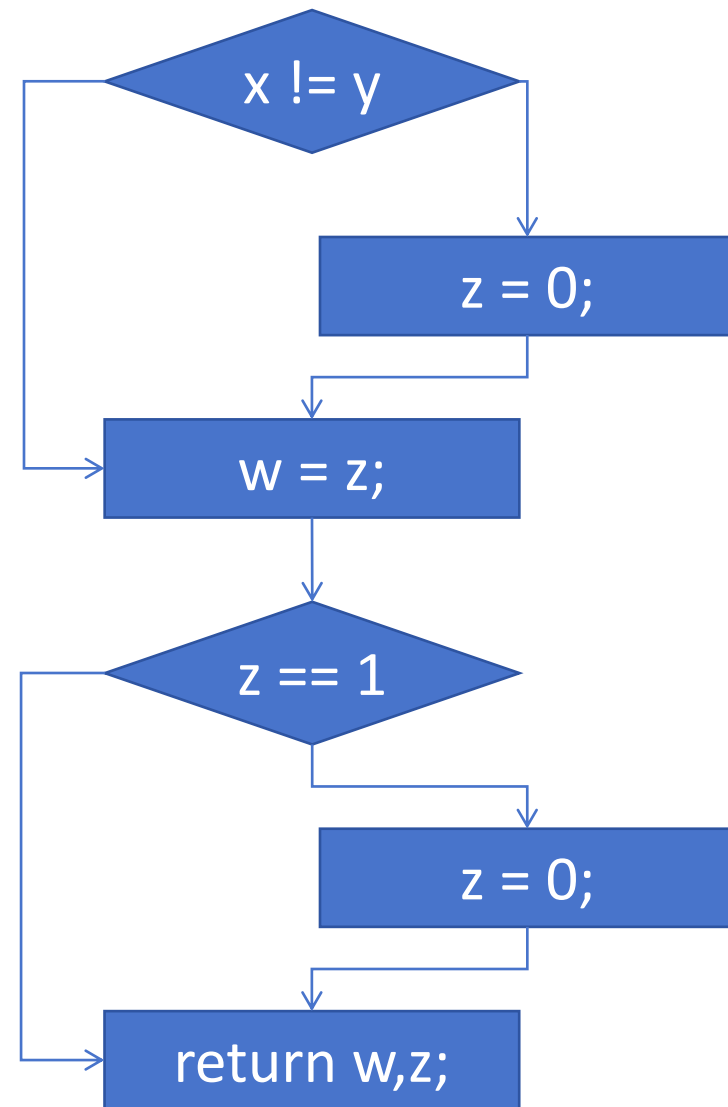
```
1. if (x != y)
2.     z = 0;
3. w = z;
3. if (z == 1)
4.     z = 0;
5. return w,z;
```

测试用例:

x=1; y=1; z=1;

x=1; y=2; z=1;

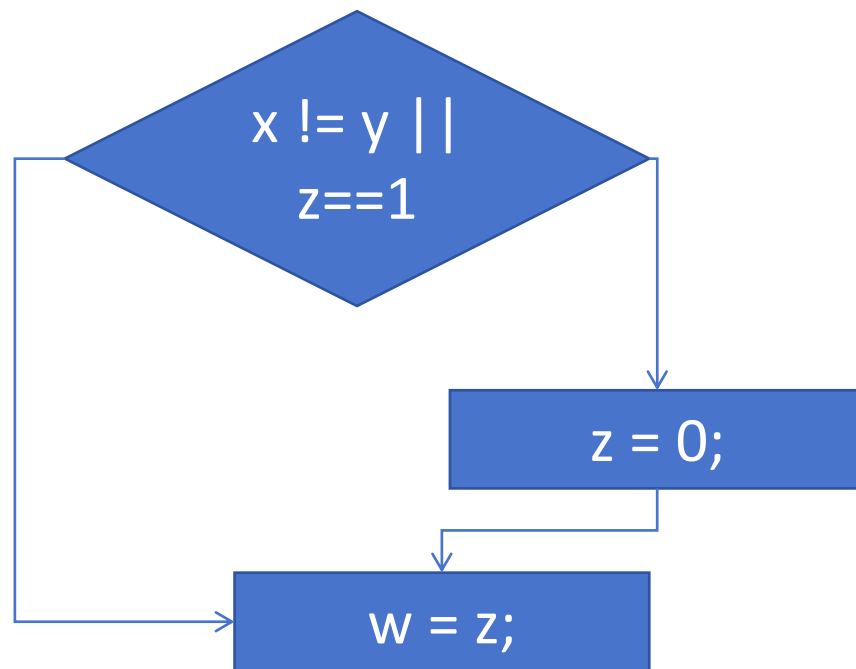
可以达到100%分支覆盖率但不能达到100%路径覆盖率



表达式/条件覆盖率

Expression/condition coverage

```
1. if (x != y || z == 1)
2.   z = 0;
3.   w = z;
```



触发覆盖率

Trigger coverage

```
always @(a or b or c)
begin
. . .
end
```

信号 a、b 和 c 在整个仿真过程中受到监控。如果在仿真过程中只有 b 和 c 更改值，则触发覆盖率将为 $2/3 = 66.67\%$ 。

翻转覆盖率

Toggle coverage

1.	// net toggle coverage			
2.	// name	Toggle	0→1	1→0
3.	clk	Yes		
4.	reset	No	Yes	No
5.	start	Yes		
6.	state[6:0]	Yes		
7.	state[9:7]	No	No	No
8.	op[2:0]	Yes		
9.	op[3]	No	No	Yes
10.	op[4]	Yes		
11.	op[5]	No	No	Yes
12.	round[1:0]	Yes		
13.	src1[63:0]	Yes		
14.	src2[63:0]	Yes		

翻转覆盖率

Toggle coverage

```
1. always @(in or reset) begin
2.     out = in;
3.     if (reset)
4.         out = 0;
5.     en = 1;
6. end
```

变量名	0→1	1→0	翻转覆盖
in	no	no	no
reset	yes	yes	yes
out	yes	yes	yes
en	no	no	no

如果有三个时刻，第一次in为1，reset标志为0；第二次in为1，reset标志为1；第三次in为1，reset标志为0。翻转覆盖率为多少？

翻转覆盖率： 50%

有限状态机覆盖率

Finite state machine (FSM) coverage

FSM 覆盖率用于衡量 RTL 设计中有限状态机 (FSM) 行为是否被充分测试。它特别适用于验证控制单元的功能完整性。

有限状态机FSM(finite state machine) 又称有限自动状态机，它拥有有限数量的状态，每个状态代表不同的意义，每个状态可以切换到 零-多 个状态。任意时刻状态机有且只能处在一个状态。

有限状态机

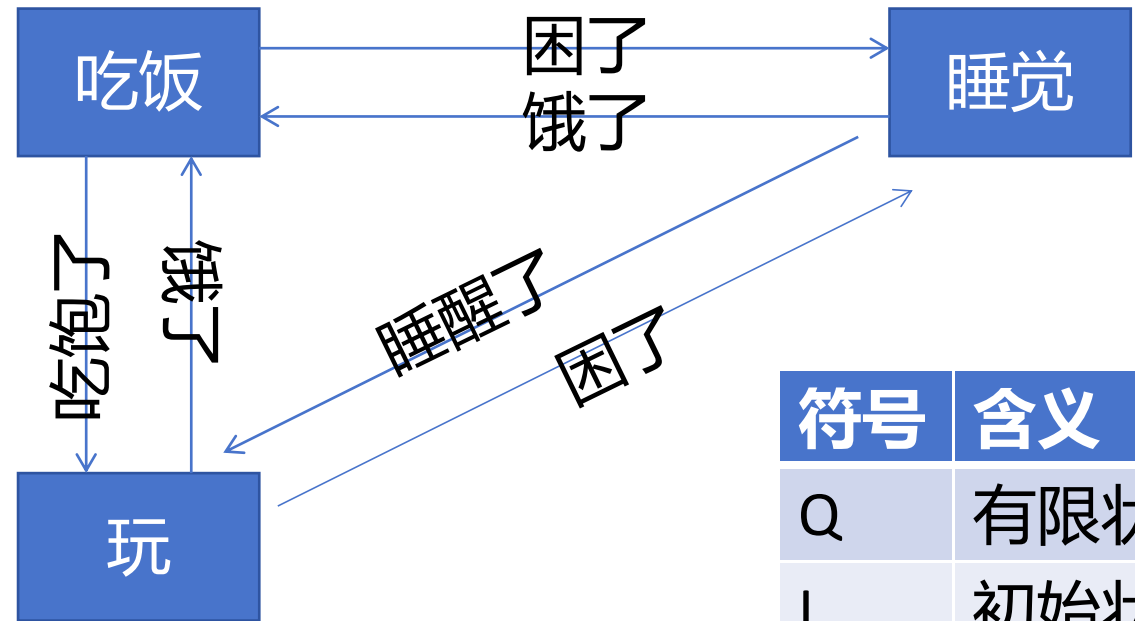
Finite state machine (FSM)

$$M = (Q, I, P, O, \delta, \lambda)$$

符号	含义
Q	有限状态集
I	初始状态集
P	输入符号集合
O	输出符号集合
δ	状态转移函数
λ	输出函数

有限状态机

Finite state machine (FSM)



$$M = (Q, I, P, O, \delta, \lambda)$$

符号	含义	例子
Q	有限状态集	{吃饭, 睡觉, 玩}
I	初始状态集	{睡觉}
P	输入符号集合	{睡醒了, 饿了, 困了, 吃饱了}
O	输出符号集合	{呼噜声, 笑声, ...}
δ	状态转移函数	$\delta(\text{状态}, \text{输入}) = \text{状态}$
λ	输出函数	$\lambda(\text{状态}) = \text{输出符号}$

有限状态机覆盖率

Finite state machine (FSM) coverage

FSM 覆盖率用于衡量 RTL 设计中有限状态机 (FSM) 行为是否被充分测试。它特别适用于验证控制单元的功能完整性。

FSM 覆盖可分为三类指标：

- 状态覆盖 (State Coverage)
- 弧覆盖 (Arc Coverage)
- 顺序弧覆盖 (Sequential Arc Coverage)

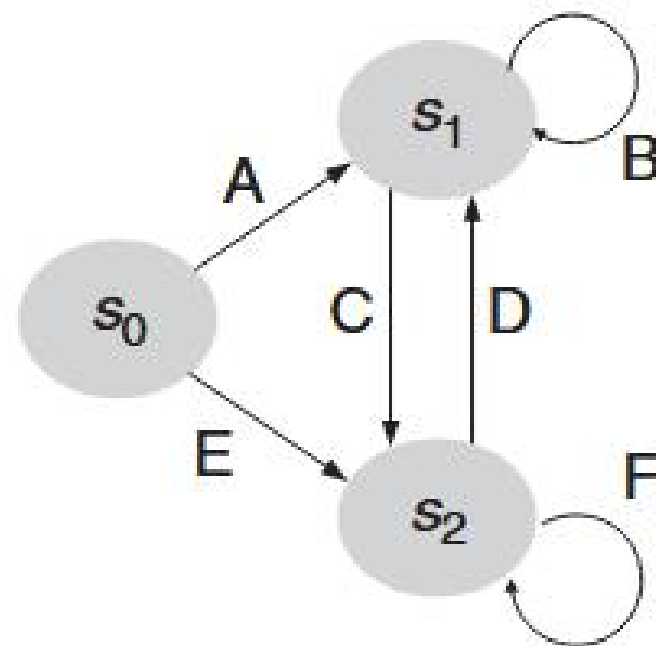
有限状态机覆盖率

Finite state machine (FSM) coverage

FSM 覆盖率用于衡量 RTL 设计中有限状态机 (FSM) 行为是否被充分测试。它特别适用于验证控制单元的功能完整性。

FSM 覆盖可分为三类指标：

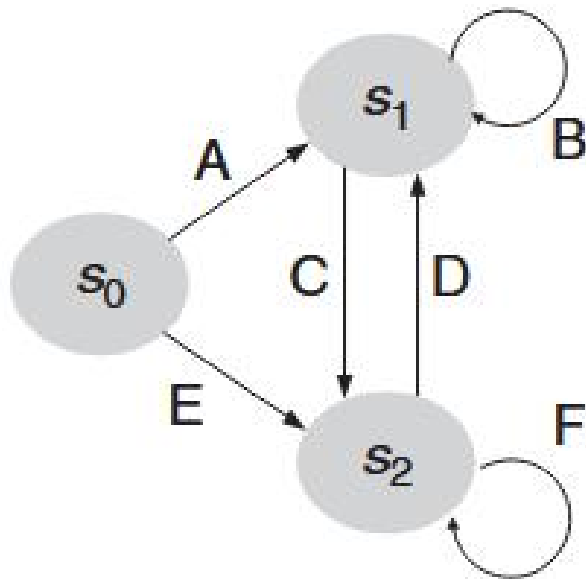
- 状态覆盖 (State Coverage)
- 弧覆盖 (Arc Coverage)
- 顺序弧覆盖 (Sequential Arc Coverage)



有限状态机覆盖率

Finite state machine (FSM) coverage

- 顺序弧覆盖 (Sequential Arc Coverage)



length of transition	1-arc	2-arc
sequence of states from s_1	$s_1 \rightarrow s_1$	$s_1 \rightarrow s_1 \rightarrow s_1$
	$s_1 \rightarrow s_2$	$s_1 \rightarrow s_1 \rightarrow s_2$
		$s_1 \rightarrow s_2 \rightarrow s_1$
		$s_1 \rightarrow s_2 \rightarrow s_2$

有限状态机覆盖率

Finite state machine (FSM) coverage

Table 9.1 Typical Coverage Targets for Different Metrics

<i>Metric</i>	<i>Coverage Goal (%)</i>
Line	100
Branch	100
Condition	60~100
Path	>50
Trigger	100
Toggle	100
FSM (state and arc)	100

主要的问题

- 如何产生输入？
- 如何给定预期并捕获错误？
- 什么时候可以停止了？

模糊测试



- 随机生成大量输入，以快速验证实现是否有错误

```
import random
def fuzzer(max_length: int = 100, char_start: int = 32, char_range: int = 32) ->
str:
    """A string of up to `max_length` characters
       in the range [`char_start`, `char_start` + `char_range`)""""
    string_length = random.randrange(0, max_length + 1)
    out = ""
    for i in range(0, string_length):
        out += chr(random.randrange(char_start, char_start + char_range))
    return out
fuzzer()
```



' !7#%"*#0=) \$;%6*; >638:*>80"= </> (/ *: - (2<4
!:5*6856&?" "11<7+%<%7, 4. 8, *+&, , \$, . "'

模糊测试

- 随机生成大量输入，以快速验证实现是否有错误
- 缺点：
 - 可能产生大量无效输入，浪费资源
 - 难以生成“目标导向”的测试场景

模糊测试

- 基于词法的测试-如变异测试：
 - 从一个合法的输入开始，每次增加、删除、修改一个或多个字符
- 基于语法的测试-如模板测试：

用两种类型的输入来约束测试生成过程：

 - (1) 用作测试用例框架的模板，其中包含一组未知的输入字段，
 - (2) 一组参数，在生成过程中可以为其设置值。用户无需直接手动创建测试，而是在其合法范围内为输入字段指定这些参数。

模糊测试

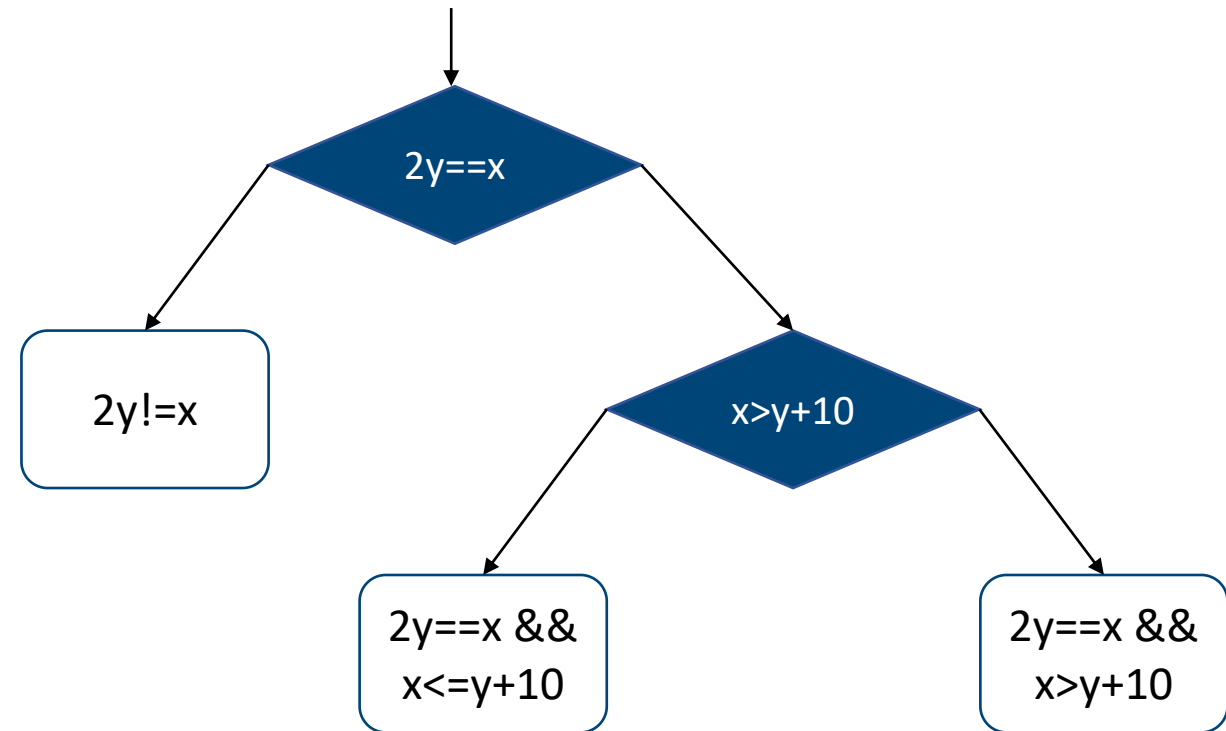
- 例子：测试一个微处理器的指令执行单元：
- 测试模板：
MUL <random R1–R4> <random R4–R8> <random R8–R20>
- 测试用例：

```
MUL R2, R5, R9
MUL R3, R7, R14
MUL R4, R4, R19
MUL R1, R6, R10
MUL R2, R8, R11
MUL R3, R5, R13
MUL R1, R7, R20
MUL R4, R6, R8
MUL R2, R4, R18
MUL R3, R8, R12
```

基于语义的测试-如符号执行

Concolic testing

```
def twice(v):  
    return 2 * v  
  
def testme(x, y):  
    z = twice(y)  
    if z == x:  
        if x > y + 10:  
            raise ValueError('ERROR')  
  
def main():  
    x = int(input("Enter value for x: "))  
    y = int(input("Enter value for y: "))  
    testme(x, y)  
  
if __name__ == "__main__":  
    main()
```



主要的问题

- 如何产生输入？
- 如何给定预期并捕获错误？
- 什么时候可以停止了？

基于断言的测试

Assertion-Based Testing

- 断言是嵌入在设计代码或验证平台中的逻辑检查语句;
- 用于在仿真过程中自动验证一个表达式是否永远为真;
- 如果表达式为假（即断言失败），则立即触发错误、警告或终止仿真。
- 断言可以分为两种类型：
 - 静态断言
 - 逻辑断言

静态断言

Static assertions

- 必须在任何时间都成立
- 可以由一阶逻辑描述
- One-hot 编码是一种特殊的二进制编码方式，特点是：只有一个位是 1，其他所有位都是 0
- 假设我们有一个 4 位宽的 one-hot 信号 bus[3:0]，我们可以写一个静态断言，意思是：“我要求 bus[3:0] 任意时刻只能有一个 1。”

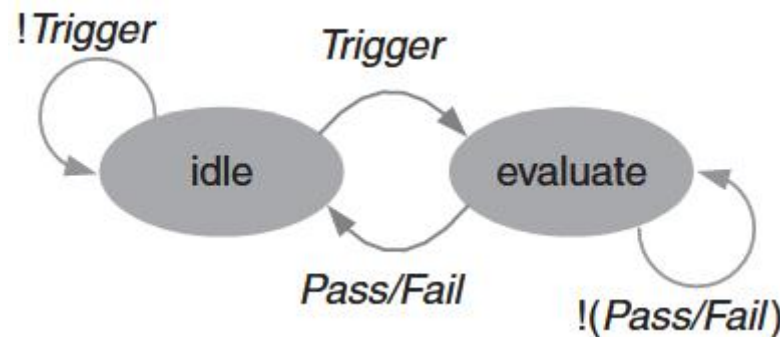
```
assert ($onehot(bus));
```

逻辑断言

Logic assertions

- 例如引入时间维度，使用时序逻辑（temporal logic）
- 在某个“前提事件”发生后，检查某个“结果事件”是否在规定时间内发生

```
@(posedge clk)  
  init_event ==> abort_event;
```



- 上例表明：一旦 `init_event` 成立，`abort_event` 必须在下一个时钟周期发生。`==>` 是 SystemVerilog 中用于非重叠时序蕴含的运算符。



逻辑综合

EXACT_COVER(A, x, b) {

如果 $\text{current_estimate} \geq |b|$ ，则返回 b ;

对矩阵 A 进行化简并更新对应的 x ;

如果 A 没有任何行，则返回 x ;

选择一个用于分支的列 c ;

$x_c = x + 1$;

\tilde{A} = 删除列 c 以及与其相关的行后的 A ;

$x^{\sim} = \text{EXACT_COVER}(\tilde{A}, x_c, b)$;

如果 $|x^{\sim}| < |b|$ ，则

$b = x^{\sim}$;

$x_c = x$;

\tilde{A} = 删除列 c 后的 A ;

$x^{\sim} = \text{EXACT_COVER}(\tilde{A}, x_c, b)$;

如果 $|x^{\sim}| < |b|$ ，则

$b = x^{\sim}$;

返回 b ;

}

	$\alpha(a'b')$	$\beta(b'c)$	$\gamma(ac)$	$\delta(ab)$
000	1	0	0	0
001	1	1	0	0
101	0	1	1	0
111	0	0	1	1
110	0	0	0	1

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

缩减矩阵的方法

- 找到基本要素：
 - 如果一行中只有一列为1
 - 选择该列
 - 从表中移除被覆盖的行

	α	β	γ	δ
000	1	0	1	0
001	0	1	1	0
101	1	1	1	0
111	1	0	0	1
110	0	0	0	1

缩减矩阵的方法

- 列（蕴含项）支配：
- 如果对所有 k ，都有 $a_{ki} \geq a_{kj}$
 - 移除列 j （被支配的）

	α	β	γ	δ
000	1	0	1	0
001	0	1	1	0
101	1	1	1	0
111	0	0	0	1
110	0	1	1	1

- 被支配的蕴含项（ j ）的最小项已经被支配蕴含项（ i ）覆盖了

- 行（最小项）支配：
- 如果对所有 k ，都有 $a_{ik} \geq a_{jk}$
 - 移除行 i （支配的）

	α	β	γ	δ
000	1	0	1	0
001	0	1	1	0
101	1	1	1	0
111	1	0	0	1
110	0	0	1	1

- 当一个蕴含项覆盖了被支配的最小项，它也覆盖了支配的最小项

EXACT_COVER(A, x, b) {

如果 $\text{current_estimate} \geq |b|$ ，则返回 b ;

对矩阵 A 进行化简并更新对应的 x ;

如果 A 没有任何行，则返回 x ;

选择一个用于分支的列 c ;

$x_c = x + 1$;

\tilde{A} = 删除列 c 以及与其相关的行后的 A ;

$x^{\sim} = \text{EXACT_COVER}(\tilde{A}, x_c, b)$;

如果 $|x^{\sim}| < |b|$ ，则

$b = x^{\sim}$;

$x_c = x$;

\tilde{A} = 删除列 c 后的 A ;

$x^{\sim} = \text{EXACT_COVER}(\tilde{A}, x_c, b)$;

如果 $|x^{\sim}| < |b|$ ，则

$b = x^{\sim}$;

返回 b ;

}

	$\alpha(a'b')$	$\beta(b'c)$	$\gamma(ac)$	$\delta(ab)$
000	1	0	0	0
001	1	1	0	0
101	0	1	1	0
111	0	0	1	1
110	0	0	0	1

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

练习

- 请写出保留 0^*01 这个蕴含项时，以下质蕴含项表可以化简为什么样？
有哪些蕴含项一定会被留下

	0^*01	010^*	*100	1^*00	100^*	*001
0001	1	0	0	0	0	1
0100	0	1	1	0	0	0
0101	1	1	0	0	0	0
1100	0	0	1	1	0	0
1000	0	0	0	1	1	0
1001	0	0	0	0	1	1

例子

保留0*01这个蕴含项时:

	0*01 [✓]	010*	*100	1*00	100*	*001
0001	1	0	0	0	0	1
0100	0	1	1	0	0	0
0101	1	1	0	0	0	0
1100	0	0	1	1	0	0
1000	0	0	0	1	1	0
1001	0	0	0	0	1	1

例子

第二列被第三列支配，删除；

	0*01 [✓]	010*	*100	1*00	100*	*001
0001	1	0	0	0	0	1
0100	0	1	1	0	0	0
0101	1	1	0	0	0	0
1100	0	0	1	1	0	0
1000	0	0	0	1	1	0
1001	0	0	0	0	1	1

例子

第六列被第五列支配，删除；

	0*01 [✓]	010*	*100	1*00	100*	*001
0001	1	0	0	0	0	1
0100	0	1	1	0	0	0
0101	1	1	0	0	0	0
1100	0	0	1	1	0	0
1000	0	0	0	1	1	0
1001	0	0	0	0	1	1

例子

第二行只被第三列覆盖，第三列是基本要素

	0*01 [✓]	010*	*100 [✓]	1*00	100*	*001
0001	1	0	0	0	0	1
0100	0	1	1	0	0	0
0101	1	1	0	0	0	0
1100	0	0	1	1	0	0
1000	0	0	0	1	1	0
1001	0	0	0	0	1	1

例子

保留 0^*01 这个蕴含项时，
 $*100$ 和 100^* 是基本要素

第六行只被第五列覆盖，第五列是基本要素

	0^*01	010^*	$*100$	1^*00	100^*	$*001$
0001	1	0	0	0	0	1
0100	0	1	1	0	0	0
0101	1	1	0	0	0	0
1100	0	0	1	1	0	0
1000	0	0	0	1	1	0
1001	0	0	0	0	1	1

两级逻辑优化

Two-level logic minimization

- 精确逻辑最小化：
 - 目标是计算出一个最小覆盖。对于大型函数是困难的
 - Quine-McCluskey方法（奎因-麦克拉斯基算法）
- 启发式逻辑最小化：
 - 致力于在短时间内计算近似的最小覆盖，这通常足够快速但可能不是最优解。
 - MINI, PRESTO, ESPRESSO

矩阵表示与运算

逻辑覆盖的矩阵表示

Matrix representation of logic covers

逻辑最小化工具使用的表示方式

不同的格式：

- 通常每个蕴含项一行

符号：

- 0, 1, *, ...
- 编码方式：

\emptyset	00
0	10
1	01
*	11

例子

\emptyset	00
0	10
1	01
*	11

- $f = a' b + ab' + ac' d$

10	01	11	11
01	10	11	11
01	11	10	01

交集

Intersection

- Intersection（交集）：两个蕴含项的交集是它们共同覆盖的那部分输入组合。

- 例子：

ab'c与b'cd'的交集：ab'cd'
101* *010 1010

例子 - Intersection: 按位与

\emptyset	00
0	10
1	01
*	11

101*

*010

01 10 01 11

11 10 01 10

01 10 01 10

1010

最小公共上界

Supercube

- Supercube（最小公共上界）：两个蕴含项的最小公共上界是能同时覆盖两个立方体所有输入组合的最小蕴含项。

- 例子：

ab'c与b'cd'的最小公共上界：b'c

101* *010

01

例子 - Supercube: 按位或

\emptyset	00
0	10
1	01
*	11

101*

*010

01 10 01 11

11 10 01 10

11 10 01 11



01

余因子

Cofactor

- 函数 f 关于 蕴含项 ϕ 的余因子为：在满足 ϕ 条件的情况下， f 的行为。

- 例子：

$$f = (a + b) \cdot c$$

$$f_a = c + bc$$

$$f_{a'} = bc$$

例子 – 取余因子

- $f = ac + bc$

- $\phi = a$

- $f_\phi = c + bc$

$$\begin{array}{r} 01 \ 11 \ 01 \\ 11 \ 01 \ 01 \\ 01 \ 11 \ 11 \\ \hline 01 \ 11 \ 01 \\ 01 \ 01 \ 01 \\ 10 \ 00 \ 00 \\ \hline 11 \ 11 \ 01 \\ 11 \ 01 \ 01 \end{array}$$

例子 – 取余因子

- $f = ac + bc$

- $\phi = a'$

- $f_{\phi'} = bc$

$$\begin{array}{r} 01 \ 11 \ 01 \\ 11 \ 01 \ 01 \\ 10 \ 11 \ 11 \\ \hline \text{void} \\ 10 \ 01 \ 01 \\ 01 \ 00 \ 00 \\ \hline 11 \ 01 \ 01 \end{array}$$

例子 – 取余因子

- $f = ab + ac + ab'c' + a'$

- $\phi = ab$

- $f_\phi = 1$

01	01	11	
01	11	01	
01	10	10	
10	11	11	
01	01	11	
<hr/>			
01	01	11	
01	01	01	
01	00	10	void
00	01	11	void
10	10	00	
<hr/>			
11	11	11	
11	11	01	

布尔运算操作

- 重言式判断
- 包含判断
- 取补

重言式判断

Tautology Judgment

- 重言式:

- 若覆盖矩阵中存在全为1的一行, 则为重言式
- 若函数仅依赖一个变量, 且在该变量域中无全为0的列, 则为重言式

- 非重言式:

- 若函数仅依赖一个变量, 且在该变量域中有全为0的列, 则函数不是重言式

重言式判断

Tautology Judgment

- 判断一个函数是否输出恒为1
- 递归范式:
 - 围绕某个变量展开
 - 若所有余因子（cofactors）都恒为真，则该函数是重言式

例子

$$f = ab + ac + ab'c' + a'$$

•选择变量a

•对a' 取余因子:

11 11 11 – 重言式.

•对a取余因子:

11 01 11

11 11 01

11 10 10

01	01	11
01	11	01
01	10	10
10	11	11
00	11	11
<hr/>		
00	01	11
00	11	01
00	10	10
00	11	11
00	00	00
<hr/>		
11	01	11
11	11	01
11	10	10

例子

$$f = ab + ac + ab'c' + a'$$

- 选择变量b

- f_a 对b' 取余因子:

11	11	01
11	11	10

- 没有一列全为0- Tautology

- f_a 对b取余因子:

有一行全为1

- 函数是重言式

11	01	11
11	11	01
11	10	10
11	00	11
11	00	11
11	00	01
11	00	10
00	00	00
11	11	01
11	11	00

单调性

Unateness

- 函数 $f(x_1, x_2, \dots, x_i, \dots, x_n)$
- 当以下式子成立时在变量 x_i 上具有正单调性:
 - $f_{x_i} \geq f_{x_i'}$
- 当以下式子成立时在变量 x_i 上具有负单调性:
 - $f_{x_i} \leq f_{x_i'}$

双相性

Binate

- 函数 $f(x_1, x_2, \dots, x_i, \dots, x_n)$
- 若在变量 x_i 上, f 既不是单调递增, 也不是单调递减, 则称函数在该变量上具有双相性 (Binate)

简单的单调性判断方法

- 如果某个变量 x 在所有蕴含项中：
 - 只以原变量 (x) 形式出现, 说明是正单调变量;
 - 只以反变量 (x') 形式出现, 说明是负单调变量;
 - 如果同时出现 x 和 x' , 说明是双相变量。

启发式策略-重言式

Heuristics

- 如果函数 f 在变量 x 上是正单调的, 则 $f_{x'} \leq f_x$

\Rightarrow 若 $f_{x'}$ 是重言式, 则 f_x 也一定是重言式

- 如果函数 f 在变量 x 上是负单调的, 则 $f_x \leq f_{x'}$

\Rightarrow 若 f_x 是重言式, 则 $f_{x'}$ 也一定是重言式

练习

请用矩阵表示法计算以下布尔函数是否是重言式：

$$f = a + a'c + a'b + ab'c'$$

•起始:

01 11 11	
10 11 01	
10 01 11	
01 10 10	

$f = a + a'c + a'b + ab'c'$

•对a取余因子:

01 11 11	
<hr/>	
01 11 11	
00 11 01	void
00 01 11	void
01 10 10	
10 00 00	
<hr/>	
11 11 11	
11 10 10	

有一行全为1, 为重言式

•对a'取余因子:

01	11	11
10	11	01
10	01	11
01	10	10
10	11	11
<hr/>		
00	11	11
10	11	01
10	01	11
00	10	10
01	00	00
<hr/>		
11	11	01
11	01	11

void
void

•对b'取余因子:

11	11	01
11	01	11
11	10	11
<hr/>		
11	10	01
11	00	11
00	01	00
<hr/>		
11	11	01

void

函数仅依赖一个变量，且在该变量域中有全为0的列，不为重言式。

=》确定一个余因子不恒为真，则该函数不是重言式

不能确定是否为重言式，继续取余因子

包含判断

Containment Judgment

- 定理:

当且仅当 F_ϕ 是一个重言式时, 函数 F 包含蕴含项 ϕ 。

- 推论:

F 是否包含 ϕ 可以通过重言式判断法来验证。

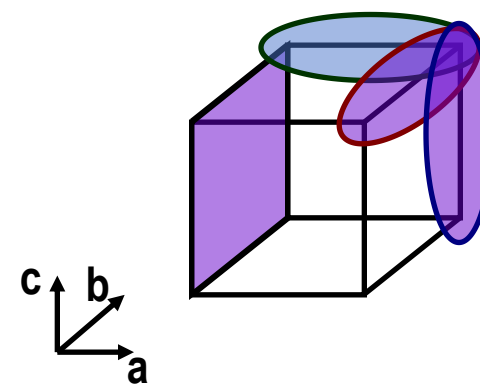
例子

$$f = ab + ac + a'$$

- 检查 $\phi = bc$ (11 01 01) 是否被 f 包含
- f 对 ϕ 取余因子:

01	11	11
01	11	11
10	11	11

- 是重言式 —— 因此 ϕ 被 f 包含.



练习

请用矩阵表示法计算 xy' 是否被以下布尔函数包含：

$$F = xz + y + xy'z'$$

•起始:

$$F = xz + y + xy'z'$$

•对 xy' 取余因子:

01 11 01

11 01 11

01 10 10

01 10 11

01 10 01

~~01 00 11~~ void

01 10 10

10 01 00

11 11 01

11 11 10

• 函数仅依赖一个变量, 且在该变量域中无全为0的列, 为重言式。

• 当且仅当 $F\alpha$ 是一个重言式时, F 包含一个蕴含项 α 。

$\Rightarrow F$ 包含 xy'

取补

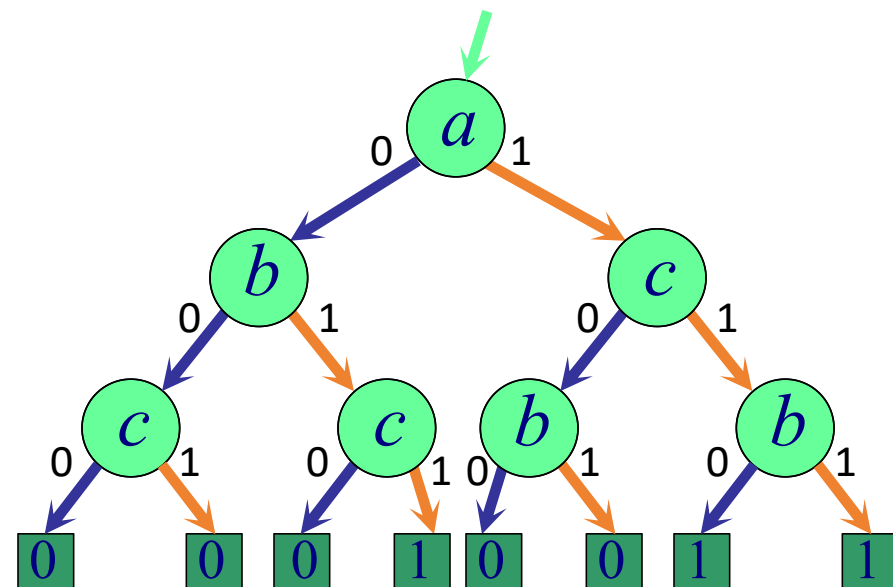
Complementation

递归范式:

$$f' = x f'_x + x' f'_{x'}$$

步骤:

1. 选择变量
2. 计算余因子
3. 对余因子取补
4. 递归, 直到余因子可以直接确定其补值为止



取补-特殊情况

- F 为 void (即0) :
 - 补集为重言式 (即1)
- F是重言式 (即1) :
 - 补集为void (即0)
- F 仅包含一个蕴含项 :
 - 补集可通过德摩根定律计算

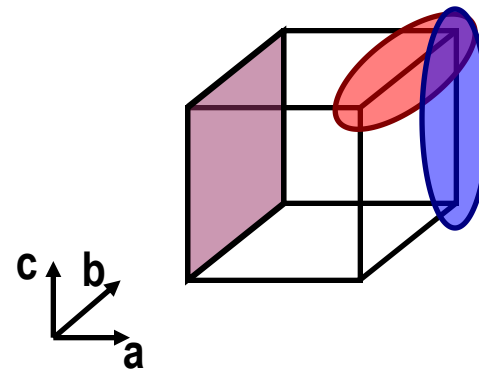
$$\neg(P \wedge Q) \equiv (\neg P) \vee (\neg Q)$$

例子

$$F = ab + ac + a'$$

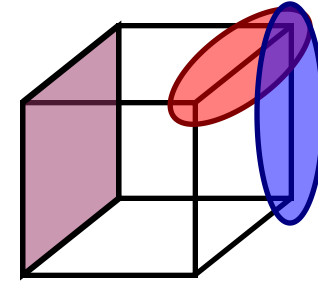
- 选择变量a
- 对a计算余因子:
 - F_a 是重言式, 因此 $F'_{a'}$ 是 void.
 - F_a 为:

11	01	11
11	11	01
 - $F' = a F'_a + a' F'_{a'} = a F'_a + 0 = a F'_a$
 - 为方便, 以下用Q代替 F_a , 即 $F' = aQ'$



例子

$$F = ab + ac + a'$$



- 选择变量 b
- Q (即 F_a) 对 b 计算余因子:
 - Q_b 是重言式, 因此 Q'_b 是 void
 - $Q_{b'} = 11 \ 11 \ 01$, 因此 $Q'_{b'}$ 是 $11 \ 11 \ 10$
- 合并结果:
 - $Q' = b Q'_b + b' Q'_{b'} = 0 + b' Q'_{b'} = 11 \ 10 \ 10$
 - $F' = a Q' = 01 \ 10 \ 10$
- 最终结果: $F' = 01 \ 10 \ 10$

启发式策略-求补

定理：

- 如果函数 f 在变量 x 上是正单调的 (positive unate) , 则
$$f' = f'_x + x' f'_{x'}$$
- 如果函数 f 在变量 x 上是负单调的 (negative unate) , 则
$$f' = x f'_x + f'_{x'}$$

推论：

- 选择有单调性的变量求补时，合并结果会更简单

练习

请用矩阵表示法求以下布尔函数的补：

$$F = ab' + b'c' + a'c$$

例子

$$F = ab' + b'c' + a'c$$

1. 求 F'

对负单调变量 b 取余因子

$$\begin{aligned} F' &= bF'_b + F'_{b'} \\ &= b(a+c') + F'_{b'} \end{aligned}$$

F_b : 01 10 11

11 10 10

10 11 01

11 01 11

01 00 11

11 00 10

10 01 01

00 10 00

10 11 01

F 仅包含一个蕴含项：补集可通过德摩根定律计算

01 11 11

11 11 10

$\Rightarrow F'b = a + c'$

$F_{b'}$: 01 10 11

11 10 10

10 11 01

11 10 11

01 10 11

11 10 10

10 10 01

00 01 00

01 11 11

11 11 10

10 11 01

还无法确认补集，
继续取余因子

例子

$$F = ab' + b'c' + a'c$$

1. 求 F'

对负单调变量 b 取余因子

$$F' = bF'_b + F'_{b'}$$

$$= b(a+c') + F'_{b'}$$

对变量 a 取余因子

$$\begin{aligned} F'_{b'} &= a F'_{ab'} + a' F'_{a'b'} \\ &= 0 + 0 \end{aligned}$$

$$\Rightarrow F' = ab + bc'$$

$$F_{ab'}: \begin{array}{ccc} 01 & 11 & 11 \\ 11 & 11 & 10 \\ 10 & 11 & 01 \end{array}$$

$$\underline{01 \ 11 \ 11}$$

$$01 \ 11 \ 11$$

$$01 \ 11 \ 10$$

$$00 \ 11 \ 01$$

$$\underline{10 \ 00 \ 00}$$

$$11 \ 11 \ 11$$

$$11 \ 11 \ 10$$

覆盖矩阵中存在全为1
的一行，为重言式

$$F_{ab'} = 1 \rightarrow F'_{ab'} = 0$$

$$F_{a'b'}: \begin{array}{ccc} 01 & 11 & 11 \\ 11 & 11 & 10 \\ 10 & 11 & 01 \end{array}$$

$$\underline{10 \ 11 \ 11}$$

$$00 \ 11 \ 11$$

$$10 \ 11 \ 10$$

$$10 \ 11 \ 01$$

$$\underline{01 \ 00 \ 00}$$

$$11 \ 11 \ 10$$

$$11 \ 11 \ 01$$

函数仅依赖一个变量，
且在该变量域中无全
为0的列，为重言式

$$F_{a'b'} = 1 \rightarrow F'_{a'b'} = 0$$

启发式最小化的基本原则

Heuristic minimization -- principles

- 从初始覆盖出发
 - 由设计者提供，或从硬件语言模型中提取
- 对当前覆盖进行修改
 - 使其成为质覆盖、无冗余覆盖
 - 重复迭代，直到获得较小的无冗余覆盖
- 通常覆盖的规模会减少
 - 对小规模覆盖的操作速度快

启发式最小化的操作

Heuristic minimization - operators

- 扩展 (Expand)
 - 使蕴含项成为质蕴含项并移除被覆盖的蕴含项
- 缩减 (Reduce)
 - 在保持覆盖范围不变的前提下，缩小每个蕴含项的大小
- 重构 (Reshape)
 - 同时修改两个蕴含项：放大一个，缩小另一个
- 去冗余 (Irredundant)
 - 移除覆盖中的冗余蕴含项

启发式最小化的操作

Heuristic minimization - operators

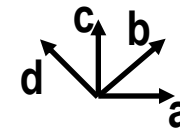
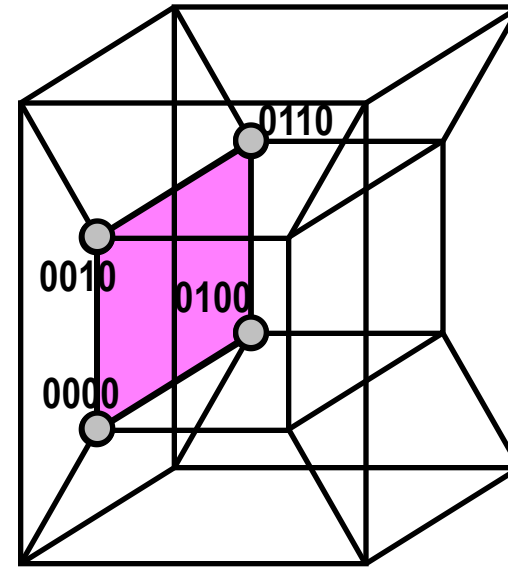
- MINI
 - 迭代执行 EXPAND、REDUCE、RESHAPE
 - MINI 只能保证最小的单个蕴含项上的非冗余覆盖
- Espresso
 - 迭代执行 EXPAND、IRREDUNDANT、REDUCE
 - Espresso 能保证无冗余覆盖，因为它包含了去冗余操作

扩展

扩展的例子

$$F = a'b'c'd' + a'b'cd' + a'b'cd + a'bcd'$$

- Expand 0000 to $\alpha = 0^{**}0$.
 - Drop 0100, 0010, 0110 from the cover.



扩展 – 简单实现

Expand - Naïve implementation

- 对于每个蕴含项
 - 对其中每个不为“无关项”（don't care）的布尔文字
 - 如果可能，将其更改为“无关项”
 - 移除所有被扩展后的蕴含项覆盖的蕴含项

有效性检查

Validity check

Espresso、MINI

- 检查扩展后蕴含项与 OFF-SET 的交集是否为空，为空代表可以
- 该过程需要做求补操作

Presto

- 检查扩展后蕴含项是否包含于 ON-SET 与 DC-SET 的并集中，包含代表可以
- 该问题可归约为递归重言式判定

例子

$$F = abc + abc' + ab'c$$

1. 求 F'
2. 尝试扩展 abc
3. 尝试扩展 abc'
4. 尝试扩展 $ab'c$

例子

$$F = abc + abc' + ab'c$$

1. 求 F'

对正单调变量 a 取余因子

$$\begin{aligned} F' &= F'_a + a'F'_{a'} \\ &= F'_a + a' \end{aligned}$$

$$F_a: \begin{array}{ccc} 01 & 01 & 01 \\ 01 & 01 & 10 \\ 01 & 10 & 01 \\ 01 & 11 & 11 \end{array} \quad F_{a'}: \begin{array}{ccc} 01 & 01 & 01 \\ 01 & 01 & 10 \\ 01 & 10 & 01 \\ 10 & 11 & 11 \end{array}$$

01 11 11

10 11 11

01 01 01

00 01 01

01 01 10

00 01 10

01 10 01

00 10 01

10 00 00

11 01 01

11 01 10

11 10 01

$$F_{a'}=0 \rightarrow F'_{a'}=1$$

例子

$$F = abc + abc' + ab'c$$

1. 求 F'

$$F' = F'_a + a'$$

$$= b F'_{ab} + b' F'_{ab'} + a'$$

$$= b'c' + a'$$

$$F_{ab}: \begin{array}{ccc} 11 & 01 & 01 \\ 11 & 01 & 10 \\ 11 & 10 & 01 \\ 11 & 01 & 11 \end{array}$$

$$11 \ 01 \ 10$$

$$11 \ 10 \ 01$$

$$11 \ 01 \ 11$$

$$11 \ 01 \ 01$$

$$11 \ 01 \ 10$$

$$11 \ 00 \ 01$$

$$00 \ 10 \ 00$$

$$11 \ 11 \ 01$$

$$11 \ 11 \ 10$$

$$F_{ab}=1 \rightarrow F'_{ab}=0$$

$$F_{ab'}: \begin{array}{ccc} 11 & 01 & 01 \\ 11 & 01 & 10 \\ 11 & 10 & 01 \\ 11 & 10 & 11 \end{array}$$

$$11 \ 01 \ 10$$

$$11 \ 10 \ 01$$

$$11 \ 10 \ 11$$

$$11 \ 00 \ 01$$

$$11 \ 00 \ 10$$

$$11 \ 10 \ 01$$

$$00 \ 01 \ 00$$

$$F_{ab'} : 11 \ 11 \ 01$$

$$F'_{ab'}: 11 \ 11 \ 10$$

例子

F' 和bc
的交集:

11 10 10
10 11 11
11 01 01

11 00 00

10 01 01

$$F = abc + abc' + ab'c$$

1. $F' = b'c' + a'$
2. 尝试扩展abc

检查abc(111)是否可以扩展为bc(*11):

->交集不为空, abc不可以扩展为bc

例子

F' 和 ac
的交集:

11 10 10
10 11 11
01 11 01

$$F = abc + abc' + ab'c$$

01 10 00

00 11 01

1. $F' = b'c' + a'$
2. 尝试扩展abc

检查abc(111)是否可以扩展为ac(1*1):

->交集为空, abc可以扩展为ac

例子

$$F = abc + abc' + ab'c$$

1. $F' = b'c' + a'$
2. 尝试扩展abc

检查ac(1*1)是否可以扩展为a(1**):

->交集不为空, ac不可以扩展为a

F' 和ac	11 10 10
的交集:	10 11 11
	01 11 11
	01 10 10
	00 11 01

最终abc扩展为ac

扩展abc为ac后移除的蕴含项:

$ab'c$

更新后的覆盖:

$$F = ac + abc'$$

例子

$$F = abc + abc' + ab'c$$

1. $F' = b'c' + a'$
2. 尝试扩展 abc
3. 尝试扩展 abc'

例子

$$F = abc + abc' + ab'c$$

1. $F' = b'c' + a'$
2. 尝试扩展 abc
3. 尝试扩展 abc'
4. 因为 $ab'c$ 已被移除, 因此不需要扩展

练习

请用矩阵表示法对以下布尔函数扩展：

$$F = ab' + b'c' + a'c$$

练习

$$F = ab' + b'c' + a'c$$

1. 求 F'

对负单调变量 b 取余因子

$$F' = bF'_b + F'_{b'}$$

$$= b(a+c') + F'_{b'}$$

对变量 a 取余因子

$$\begin{aligned} F'_{b'} &= a F'_{ab'} + a' F'_{a'b'} \\ &= 0 + 0 \end{aligned}$$

$$\Rightarrow F' = ab + bc'$$

$$F_{ab'}: \begin{array}{ccc} 01 & 11 & 11 \\ 11 & 11 & 10 \\ 10 & 11 & 01 \\ 01 & 11 & 11 \end{array}$$

$$11 \ 11 \ 10$$

$$10 \ 11 \ 01$$

$$01 \ 11 \ 11$$

$$01 \ 11 \ 11$$

$$01 \ 11 \ 10$$

$$00 \ 11 \ 01$$

$$10 \ 00 \ 00$$

$$11 \ 11 \ 11$$

$$11 \ 11 \ 10$$

覆盖矩阵中存在全为1
的一行，为重言式

$$F_{ab'} = 1 \rightarrow F'_{ab'} = 0$$

$$F_{a'b'}: \begin{array}{ccc} 01 & 11 & 11 \\ 11 & 11 & 10 \\ 10 & 11 & 01 \\ 10 & 11 & 11 \end{array}$$

$$11 \ 11 \ 10$$

$$10 \ 11 \ 01$$

$$10 \ 11 \ 11$$

$$00 \ 11 \ 11$$

$$10 \ 11 \ 10$$

$$10 \ 11 \ 01$$

$$01 \ 00 \ 00$$

$$11 \ 11 \ 10$$

$$11 \ 11 \ 01$$

函数仅依赖一个变量，
且在该变量域中无全
为0的列，为重言式

$$F_{a'b'} = 1 \rightarrow F'_{a'b'} = 0$$

例子

- $F = ab' + b'c' + a'c$

01 10 11

11 10 10

10 11 01

- $F' = ab + bc'$

01 01 11

11 01 10

- 扩展 ab' (01 10 11) :
 - 扩展为 a (01 11 11)
交集不为空, 不可以
 - 扩展为 b' (11 10 11)
交集为空, 可以

F'交a: 01 01 11

11 01 10

01 11 11

01 01 11

01 01 10

F'交b': 01 01 11

11 01 10

11 10 11

01 00 11

11 00 10

最终扩展 ab' 为 b' , 移除所有被扩展后的蕴含项覆盖的蕴含项:
 $b'c'$

更新后的覆盖:

11 10 11

10 11 01

例子

- $F = ab' + b'c' + a'c$

11 10 11

10 11 01

- $F' = ab + bc'$

01 01 11

11 01 10

- 扩展 $a'c$ (10 11 01) :

- 扩展为 a' (10 11 11)
交集不为空, 不可以
- 扩展为 c (11 11 01)
交集不为空, 不可以

F'交a': 01 01 11

11 01 10

10 11 11

00 01 11

10 01 10

F'交c: 01 01 11

11 01 10

11 11 01

01 01 01

11 01 00

最终 $a'c$ 没有被扩展, 原覆盖保持不变:

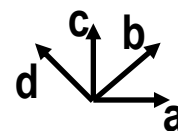
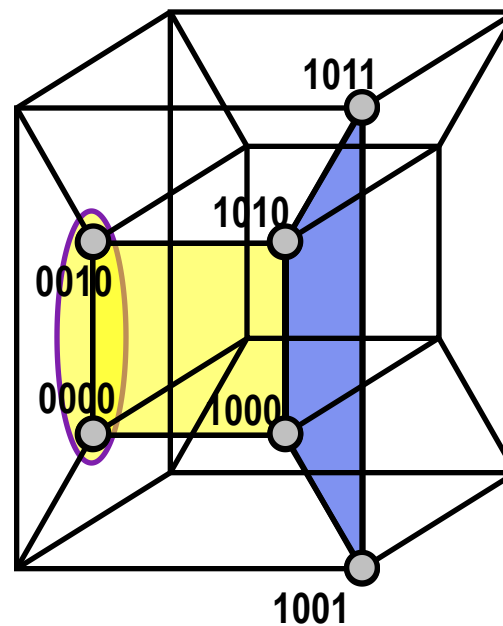
11 10 11

10 11 01

縮減

缩减的例子

- $\{*0*0, 10^{**}\}$
- Reduce $\beta = *0*0$ to $\beta' = 00*0$.



缩减

Reduce

- 最优缩减可以被精确地确定
- 定理：
 - 当要对蕴含项 α 进行缩减时
 - Q =除 α 以外的剩余的蕴含项相加
 - 则 α 缩减后的蕴含项为：

$$\acute{\alpha} = \alpha \cap \text{supercube}(Q'\alpha)$$

例子

- $F = c' + a'b'$

$$\begin{array}{ccc} 11 & 11 & 10 \\ 10 & 10 & 11 \end{array}$$

- 选择第一个蕴含项(c')作为 α :

$$\acute{\alpha} = \alpha \cap \text{supercube}(Q'\alpha)$$

- Q =除 α 以外的剩余的蕴含项相加= $a'b'$
- $Q' = a + b$
- $Q'\alpha$: $a + b$
- $\text{supercube}(Q'\alpha) = 1$
- $\acute{\alpha} = \alpha \cap \text{supercube}(Q'\alpha) = c'$ (即无法化简)

$$01 \ 11 \ 11$$

$$11 \ 01 \ 11$$

$$11 \ 11 \ 10$$

$$01 \ 11 \ 10$$

$$11 \ 01 \ 10$$

$$00 \ 00 \ 01$$

$$01 \ 11 \ 11$$

$$11 \ 01 \ 11$$

$$11 \ 11 \ 11$$

$$11 \ 11 \ 10$$

$$11 \ 11 \ 10$$

例子

- $F = c' + a'b'$

11 11 10
10 10 11

- 选择第二个蕴含项($a'b'$)作为 α :

$$\acute{\alpha} = \alpha \cap \text{supercube}(Q'\alpha)$$

- Q = 除 α 以外的剩余的蕴含项相加 = c'
- $Q' = c$
- $Q'\alpha$: c
- $\text{supercube}(Q'\alpha) = c$
- $\acute{\alpha} = \alpha \cap \text{supercube}(Q'\alpha) = a'b'c$ (最优化简)

11 11 01
10 10 11

10 10 01
01 01 00

11 11 01
10 10 11

10 10 01

练习

请用矩阵表示法对以下布尔函数缩减：

$$F = ac'd + ab'$$

- 对于每个蕴含项 α ，缩减后的蕴含项为：

$$\acute{\alpha} = \alpha \cap \text{supercube}(Q'\alpha)$$

练习

$$F = ac'd + ab'$$

01	11	10	01
01	10	11	11

- 选择第一个蕴含项($ac'd$)作为 α :

$$\acute{\alpha} = \alpha \cap \text{supercube}(Q'\alpha)$$

- Q =除 α 以外的剩余的蕴含项相加= ab'
- $Q'=a'+b$
- $Q'\alpha: b$
- $\text{supercube}(Q'\alpha)=b$
- $\acute{\alpha} = \alpha \cap \text{supercube}(Q'\alpha)=abc'd$

- 原函数更新为:
- | | | | |
|----|----|----|----|
| 01 | 01 | 10 | 01 |
| 01 | 10 | 11 | 11 |

10 11 11 11

11 01 11 11

01 11 10 01

00 11 10 01

01 01 10 01

10 00 01 10

11 01 11 11

练习

- $F = abc'd + ab'$ **01 01 10 01**
 01 10 11 11
- 选择第二个蕴含项(ab')作为 α :

$$\acute{\alpha} = \alpha \cap \text{supercube}(Q'\alpha)$$

- Q =除 α 以外的剩余的蕴含项相加= $abc'd$
- $Q'=a'+b'+c+d'$
- $Q'\alpha: 1$
- $\text{supercube}(Q'\alpha)=1$
- $\acute{\alpha} = \alpha \cap \text{supercube}(Q'\alpha)=ab'$ (即无法化简)

```

10 11 11 11
11 10 11 11
11 11 01 11
11 11 11 10
01 10 11 11

```

```

00 10 11 11
01 10 11 11
01 10 01 11
01 10 11 10
10 01 00 00

```

```

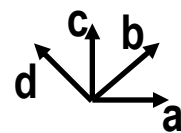
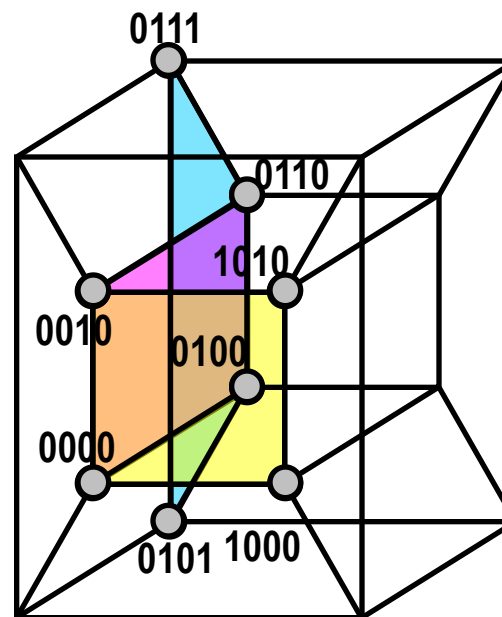
11 11 11 11
11 11 01 11
11 11 11 10

```

去冗余

去冗余的例子

- 覆盖: $\{\alpha = 0**0, \beta = *0*0, \gamma = 01**\}$.
- 其中 α 是冗余的
- 去除后得到覆盖: $\{\beta, \gamma\}$.



去冗余

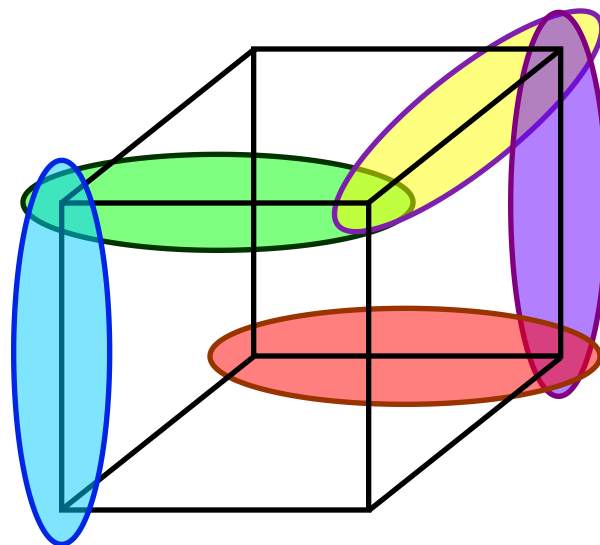
Irredundant

- 相对必要集 (E_r)
 - 覆盖了函数中某些未被其他蕴含项覆盖的最小项的蕴含项组成的集合。
 - 重要说明：做启发式逻辑优化时我们尚未知道所有的质蕴含项，只是相对必要
- 完全冗余集 (R_t)
 - 被相对必要项所覆盖的蕴含项集合。
- 部分冗余集 (R_p)
 - 剩余的蕴含项集合。

例子

α	10	10	11
β	11	10	01
γ	01	11	01
δ	01	01	11
ε	11	01	10

- $E^r = \{ \textcolor{blue}{\alpha}, \textcolor{red}{\varepsilon} \}$
- $R^t = \emptyset$
- $R^p = \{ \textcolor{green}{\beta}, \textcolor{orange}{\gamma}, \textcolor{purple}{\delta} \}$



去冗余

Irredundant

- 相对必要集 (E_r)
 - Q = 除某个蕴含项 α 以外的剩余的蕴含项相加
 - 若 Q 不包含 α , 则 α 是相对必要项 $\Leftrightarrow Q_\alpha$ 不为重言式
- 完全冗余集 (R_t)
 - 若 E_r 包含某个蕴含项 α , 则 α 是完全冗余项
- 部分冗余集 (R_p)
 - $R_p = F - (E_r \cup R_t)$

例子

$$F = a'b' + b'c + ac$$

判断 $a'b'$ 是否是相对必要项

$$Q = b'c + ac$$

$Q_{a'b'}$ 不为重言式 $\Rightarrow a'b'$ 是相对必要项

$$Er = \{a'b'\}$$

11	10	01
01	11	01
10	10	11
<hr/>		
10	10	01
00	10	01
01	01	00
<hr/>		
11	11	01

例子

$$F = a'b' + b'c + ac$$

判断 $b'c$ 是否是相对必要项

$$Q = a'b' + ac$$

$Q_{b'c}$ 为重言式 $\Rightarrow b'c$ 不是相对必要项

$$Er = \{a'b'\}$$

10 10 11

01 11 01

11 10 01

10 10 01

01 10 01

00 01 10

10 11 11

01 11 11

例子

$$F = a'b' + b'c + ac$$

判断 ac 是否是相对必要项

$$Q = a'b' + b'c$$

Q_{ac} 不为重言式 $\Rightarrow ac$ 是相对必要项

$$Er = \{a'b', ac\}$$

10	10	11
11	10	01
01	11	01
<hr/>		
00	10	01
01	10	01
10	00	10
<hr/>		
11	10	11

例子

$$F = a'b' + b'c + ac$$

$$Er = \{a'b', ac\}$$

判断 $b'c$ 是否是完全冗余项

Er 包含 $b'c \Rightarrow b'c$ 是完全冗余项

$$Er = \{a'b', ac\}$$

$$Rt = \{b'c\}$$

$$Rp = \emptyset$$

10 10 11

01 11 01

11 10 01

10 10 01

01 10 01

00 01 10

10 11 11

01 11 11

去冗余

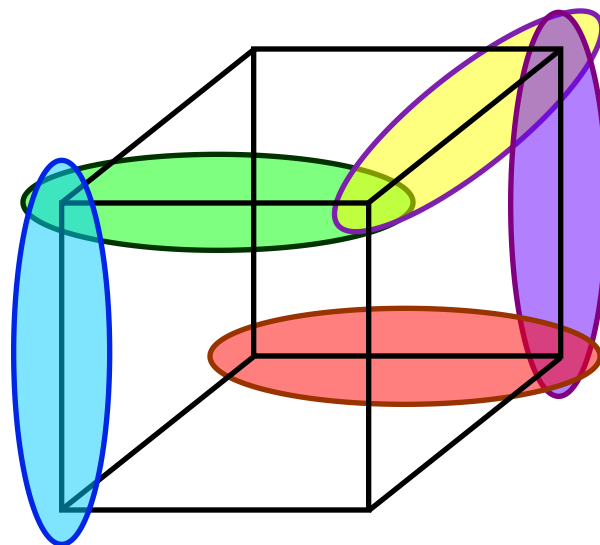
Irredundant

- 寻找一个 R_p (部分冗余集) 的子集, 使其与 E_r (相对必要集)一同覆盖整个函数
- 对 R_p 中的每个蕴含项 α , 判断其是否被 $H=R_p+E_r-\{\alpha\}$ 这个集合组成的覆盖所包含, 若包含, 则可以从 R_p 中去掉该蕴含项
- 排除所有可以去掉的蕴含项, 剩余的蕴含项与 E_r 组成非冗余覆盖

例子

α	10	10	11
β	11	10	01
γ	01	11	01
δ	01	01	11
ε	11	01	10

- $E^r = \{ \textcolor{blue}{\alpha}, \textcolor{red}{\varepsilon} \}$
- $R^t = \emptyset$
- $R^p = \{ \textcolor{green}{\beta}, \textcolor{orange}{\gamma}, \textcolor{purple}{\delta} \}$



- $E^r = \{ \alpha, \varepsilon \}$
- $R^t = \emptyset$
- $R^p = \{ \beta, \gamma, \delta \}$

例子

• 判断是否可以去掉 β :

➤ $H = R^p + E^r - \{\alpha\} = a'b' + ac + ab + bc'$

➤ H_{β} 是重言式 = 》 可以去掉 β

10 10 11	α	10 10 11
01 11 01	β	11 10 01
01 01 11	γ	01 11 01
11 01 10	δ	01 01 11
11 10 01	ε	11 01 10
<hr/>		
10 10 01		
01 10 01		
01 00 01		
11 00 00		
00 01 10		
<hr/>		
10 11 11		
01 11 11		

- $E^r = \{ \alpha, \varepsilon \}$
- $R^t = \emptyset$
- $R^p = \{ \gamma, \delta \}$

例子

10 10 11	α	10 10 11
01 01 11	β	11 10 01
11 01 10	γ	01 11 01
01 11 01	δ	01 01 11
<hr/>		
00 10 01	ε	11 01 10
01 01 01		
01 01 00		
10 00 10		
<hr/>		
11 01 11		

• 判断是否可以去掉 γ :

➤ $H = R^p + E^r - \{\alpha\} = a'b' + ab + bc'$

➤ H_γ 不是重言式 = 》不可以去掉 γ

- $E^r = \{ \alpha, \varepsilon \}$
- $R^t = \emptyset$
- $R^p = \{ \gamma, \delta \}$

例子

10 10 11	α	10 10 11
01 11 01	β	11 10 01
11 01 10	γ	01 11 01
01 01 11	δ	01 01 11
<hr/>		
00 00 11	ε	11 01 10
01 01 01		
01 01 10		
10 10 00		
<hr/>		
11 11 01		
11 11 10		

• 判断是否可以去掉 δ :

➤ $H = R^p + E^r - \{\alpha\} = a'b' + ac + bc'$

➤ H_δ 是重言式 = 》 可以去掉 δ

最终得到的非冗余覆盖: $\{ \alpha, \varepsilon, \gamma \}$

练习

请用矩阵表示法为以下布尔函数去冗余：

$$F = b'cd' + a'cd' + a'b$$

练习

$$F = b'cd' + a'cd' + a'b$$

判断 $b'cd'$ 是否是相对必要项

$$Q = a'cd' + a'b$$

$Q_{b'cd'}$ 不为重言式 $\Rightarrow b'cd'$ 是相对必要项

$$Er = \{b'cd'\}$$

10 11 01 10

10 01 11 11

11 10 01 10

10 10 01 10

10 00 01 10

00 01 10 01

10 11 11 11

练习

$$F = b'cd' + a'cd' + a'b$$

判断 $a'cd'$ 是否是相对必要项

$$Q = b'cd' + a'b$$

$Q_{a'cd'}$ 是重言式 $\Rightarrow a'cd'$ 不是相对必要项

$$Er = \{b'cd'\}$$

11 10 01 10

10 01 11 11

10 11 01 10

10 10 01 10

10 01 01 10

01 00 10 01

11 10 11 11

11 01 11 11

练习

$$F = b'cd' + a'cd' + a'b$$

判断 $a'b$ 是否是相对必要项

$$Q = b'cd' + a'cd'$$

$Q_{a'b}$ 不为重言式 $\Rightarrow a'b$ 是相对必要项

$$Er = \{b'cd', a'b\}$$

11 10 01 10

10 11 01 10

10 01 11 11

10 00 01 10

10 01 01 10

01 10 00 00

11 11 01 10

练习

$$F = b'cd' + a'cd' + a'b$$

判断 $a'cd'$ 是否是完全冗余项

之前已计算过 $a'cd'$ 包含于 $Q = b'cd' + a'b$

$\Rightarrow a'cd'$ 是完全冗余项

$$E_r = \{b'cd', a'b\}$$

$$R_t = \{a'cd'\}$$

去冗余后 $F = b'cd' + a'b$

11 10 01 10

10 01 11 11

10 11 01 10

10 10 01 10

10 01 01 10

01 00 10 01

11 10 11 11

11 01 11 11

