# 嵌入式软件测试技术

## （动态篇-需求测试）

皮永辉

2012年8月

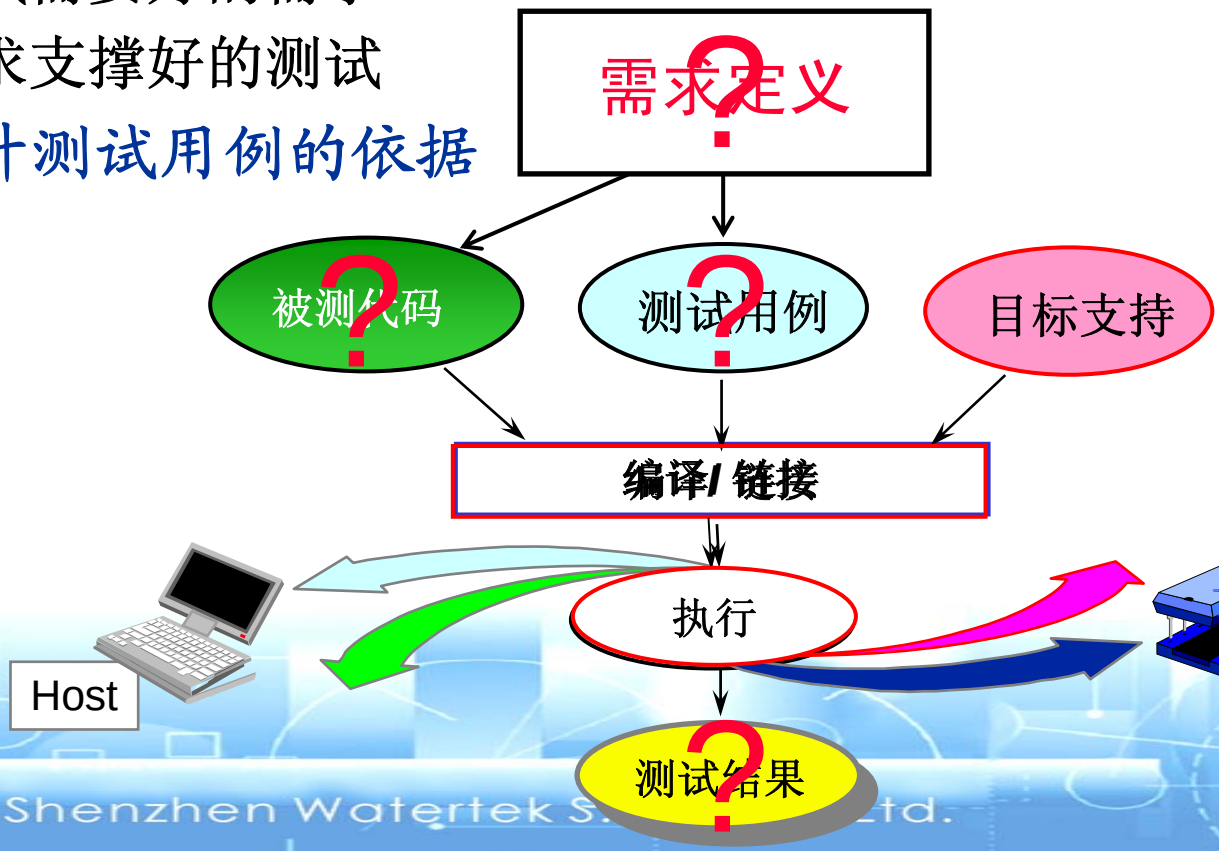
# 内容提要

- 测试从需求开始
  - 需求是动态测试的基础
  - 测试用例设计的考虑
- 关于需求测试
  - 什么是需求
  - 需求测试过程
  - 二义性审查
- 创建因果图
  - 因果图法简介
  - 图写逻辑关系
  - 添加约束
  - 从需求到因果图
- 测试用例设计
  - 敏感路径法
  - 如何检测错误
  - 所需的功能变例
  - 定义测试用例

# 嵌入式软件动态测试技术：

## 测试从需求开始

- 回顾交叉测试
- 需求是动态测试的基础
  - 没有需求，就没有真正的测试
    - 好的测试需要好的需求
    - 好的需求支撑好的测试
  - 需求是设计测试用例的依据



测试从需求开始：

# 测试用例设计的考虑

- 先有需求，然后有测试用例
- 测试用例的质量决定测试的质量
- 测试用例设计：
  - 你的需求值得信赖吗？
  - 测试用例的数量多少才合适？
    - 功能都覆盖到了吗
    - 测试的数量可否减少一些
  - 测试用例的质量如何？
    - 怎样寻找暴露缺陷的最佳路径
    - 测试结果正确吗

测试从需求开始：

# 测试用例设计的考虑

- 解决方案：基于需求的测试
  Requirements Based Testing

  - 从分析需求入手
    - 对需求进行测试
    - 纠正错误并优化
  - 以需求为基础设计测试用例
    - 最大的功能覆盖
    - 最小的测试数量
    - 确保因为“正确的激励”得到“正确的响应”

# 关于需求测试

- 什么是需求
  - 有关系统“做什么”的定义
  - A definition of WHAT the system should do.
    - 系统需求定义（Requirements）
    - 产品规格说明（Specification）

关于需求测试：

# 需求的分类

- 功能需求
  - 算法、转换、数据结构等
- 性能需求
  - 处理速度
  - 单位时间处理的事件数
- 可用性需求
  - 人体工程学的考虑
  - 外观与感觉
- 安全性需求
  - 用户权限
  - 数据安全

关于需求测试：

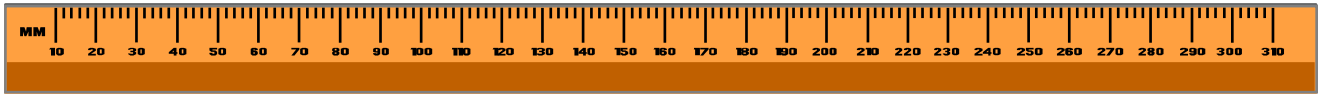
# 需求的分类

- 可靠性需求
  - 平均无故障时间等
- 可移植性需求
  - 移植到不同平台/操作系统的能力
- 本地化需求
  - 不同的国家，不同的语言
- 可维护性
  - 修复、改进
- 可扩展性
  - 增加功能、提升性能
- 可测试性
  - 诊断能力

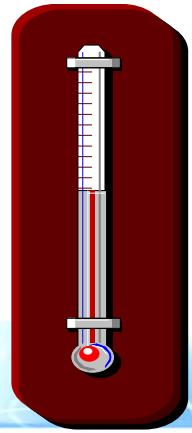
# 关于需求测试：

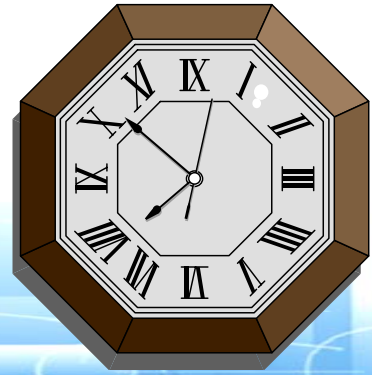
## 需求的可测性

- 需求测试要求需求具备“可测试性”




- 可测性的特点
  - 可测量的，或确定的结果
  - 给定系统的初始状态和一组输入，就可以精确地预测输出是什么





关于需求测试：

# 需求测试的内容

- RBT并不假定需求说明都是“好”的
  - 可以是任何语言的
  - 运行在任何平台
  - 概要的
  - 详细的
- 确认需求是
  - 正确
  - 完整
  - 无二义性
  - 逻辑一致
- 设计足够的测试用例
  - 验证设计和编码正确地实现了上述需求

关于需求测试：

# 可测性的需求

## 一个例子

向具备资格的客户派发俱乐部的小册子。将高级资格客户列入A组邮件列表

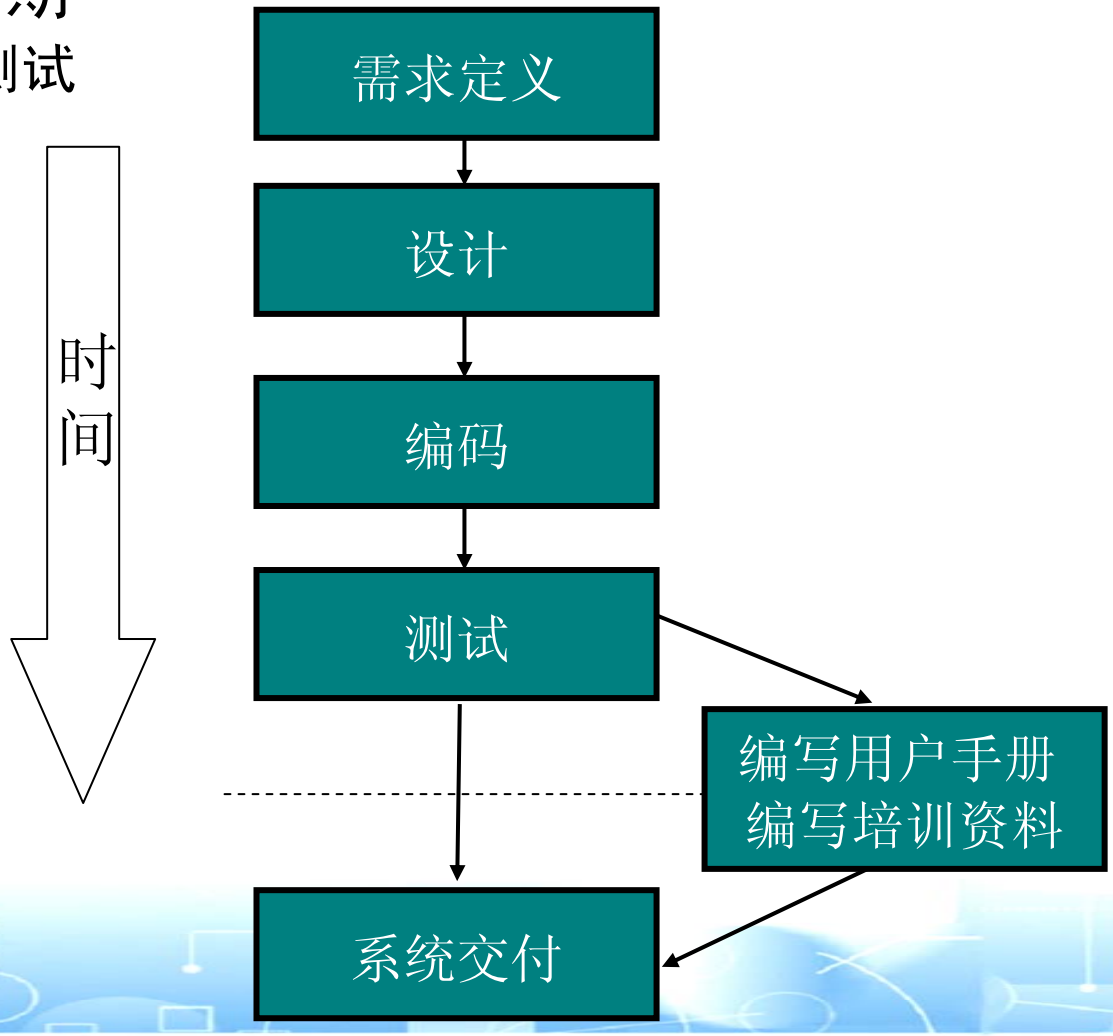
### 需求描述：

- 如果某人是18岁以下，并且打网球，那么就向他派发网球俱乐部的小册子
- 如果某人是18岁或18岁以上，或者领有摩托车驾照，就向他派发摩托车俱乐部的小册子
- 如果某客户同时被派发了这两种小册子，就将其加入到A组邮件列表

# 关于需求测试：

## 需求测试过程

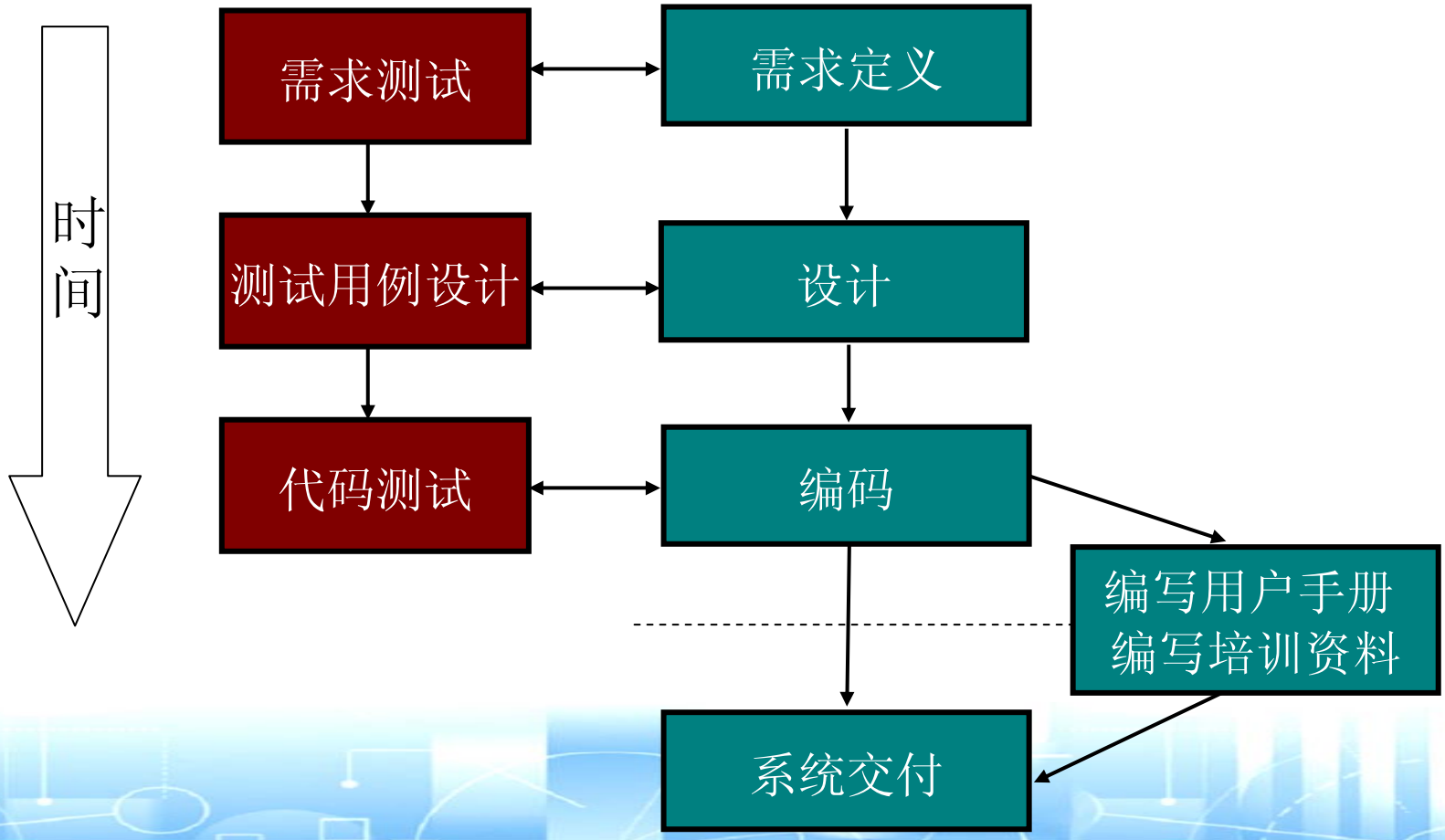
- 标准开发周期
  - 没有需求测试



关于需求测试：

# 需求测试过程

- 集成了需求测试的开发周期



关于需求测试：

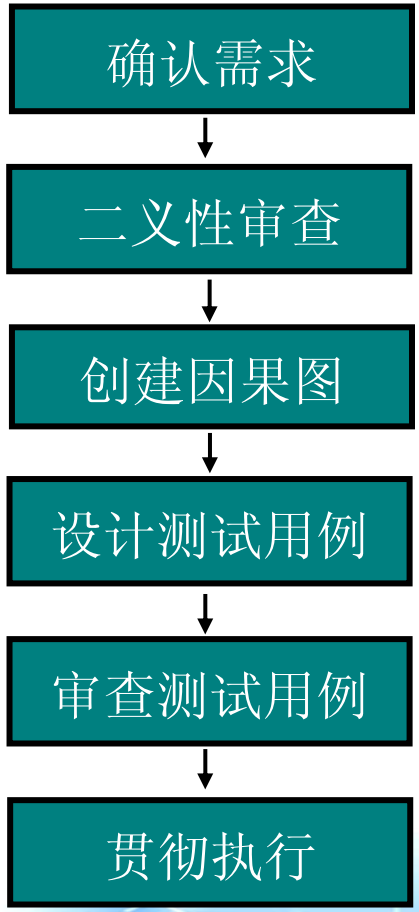
# 需求测试过程

- 收获
  - 缩短开发周期
    - 增加了并发工作过程
  - 降低资源消耗
    - 将零散工作和重复劳动减少到最小
  - 改进产品质量
    - 基于需求的测试
  - 集成式测试
    - 开发与测试紧密结合

# 关于需求测试：

## 需求测试过程

- 需求测试的具体步骤
  - 确认需求
    - 对照最初的目标
    - 对照实际场景和案例
  - 二义性审查
    - 初始级
    - 专家级
  - 创建因果图
  - 设计测试用例
    - 一致性检查
  - 审查测试用例
    - 需求定义人员、用户、开发人员
  - 贯彻执行测试用例
    - 设计、编码
    - 功能测试



# 关于需求测试：

# 二义性审查

- 需求的二义性
  - 需求最初是用自然语言写成的
  - 所有的自然语言都有其固有的模糊性
  - 但正式的需求说明语言并非可行的选择
    - 适合计算机理解
    - 作为“自然”的人难以理解
  - 需要结构化的自然语言来帮忙
  - 多数开发人员不知道如何撰写详细的、无二义性的需求

```
/* PROLOG */
/* Bidding Rules -  Opening Bid                    */

openbid(0.pass) < - totpoints (P,bidder)
        & (le(P,10) & longsuit (L,*,bidder) &
      lt(L,7)
         | ge(P,11) & le(P,12)
         | eq(P,13) & quicktricks(Q,bidder) &
      lt(Q,2)).
openbid(1.Suit) < - openpoints(bidder)
        & longsuit(L,Suite,bidder) & ge(L,5)
        & next longsuit(M,*,bidder) & lt(M,5).
openbid(1.Suit) < - openpoints(bidder)
        & longsuit(L1,S1,bidder)
        & next longsuit(L2,S2,bidder)
        & eq(L1,5) & (L2,5)
        & ( gt(S1,S2) & Suite=S1
         | gt(S2,S1) & Suite=S2).
openbid(1.Suit) < - openpoints(bidder)
        & longsuit(L,Suit,bidder) & eq(L,4)
        & hand(bidder,Suit,L.P.*) & ge(P,4)
        & (Suit=spades | Suit=hearts).
```

关于需求测试：

# 二义性审查

- 需求的二义性
  - 什么是二义性
    - 如果一个人写下来是一种意思，别人读起来却是另一种意思，这就是二义性
  - 二义性的表现
    - 背离原义
    - 不同的理解
    - 不确定
    - 不清晰
    - 遗漏
    - 忽略

# 二义性审查

## 二义性举例

- 二加二的一半等于几？

  [二加[二]的一半]等于几？ 23

- 民可使由之不可使知之 --《论语•泰伯》

  民可使由由之不可使使之知之。

# 二义性审查

- 二义性举例
  - Mokusatsu（默杀）
    - **1945**年，世界反法西斯战争已经取得决定性胜利，盟军发表了《波茨坦公告》，要求日本无条件投降。
    - 日本首相铃木召开记者招待会，说“我认为这份公告是开罗会议宣言的翻版，对政府不附加任何重要价值。我们唯一要做的就是***mokusatsu*** **…**”
    - 日本的回应传到美国，被认为是拒绝了《波茨坦公告》。杜鲁门总统下令向广岛和长崎投下了原子弹...
  - Mokusatsu一词有多种含义
    - 沉默中杀死
    - 蔑视，置之不理
    - 沉默，不予评论

# 二义性审查

## 二义性审查表

- Dangling Else（悬空else）
- Ambiguity of Reference（参考模糊）
- Scope of Action（作用范围）
- Omissions（忽略）
  - Causes without Effects(有因无果)
  - Effects without Causes(有果无因)
  - Missing Causes（忽略原因）
  - Missing Effects（忽略结果）
  - Complete Omissions(全部忽略)
- Ambiguous Logical Operators（逻辑操作符）
  - Or, And, Nor, Nand
  - Implicit Connectors
  - Compound Operators
- Negation（否定）
  - Scope of Negation
  - Unnecessary Negation
  - Double Negation

关于需求测试：

# 二义性审查

## 二义性审查表

- Ambiguous Statements（表述模糊）
  - Verbs, Adverbs, Adjectives
  - Variables, Unnecessary Aliases
- Random Organization（混乱）
  - Mixed Causes and Effects
  - Random Case Sequence
- Built-In Assumptions（内藏假设）
  - Functional/Environmental Knowledge
- Ambiguous Precedence Relationships（优先关系模糊）
- Implicit Cases（暗示）
- Etc.（等等…）
- I.E. versus E.G.（笔误）
- Temporal Ambiguity（临时性模糊）
- Boundary Ambiguity（边界歧义）

关于需求测试：

# 二义性审查

- 其他呢（悬空else）

  MUST BE, WILL BE, IS ONE OF, SHOULD BE, COULD BE, CAN BE, SHALL.

  必是，将是，是…之一，应该是，可能是，可以是

  - 一个人的婚姻状况必定是已婚、未婚或离异

    有没有其他？

- 参考、指引模糊

  IT, SUCH, THE ABOVE, THE PREVIOUS, THEM, THESE, THEY 它/它们，该…，这个/这些，上述，前者

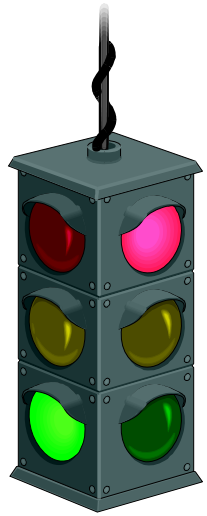
  - A加上B，该数一定为正

    哪个数？

关于需求测试：

# 二义性审查

- 忽略



  - 如果你闯红灯，就会被罚款。
    忽略——你被交警抓住

  - 如果账户已透支，就取消开票权
    忽略——通知客户

关于需求测试：

# 二义性审查

- 逻辑操作的模糊
  - 或(OR)：If A or B then C
    - “A and B each produce C.”
    - “A and B produce C.”
    - “A produces C.  B also produces C.”
    - “If A or B or both produce C.”
    - “A and/or B produce C.”
    - “Either A or B produces C.”
    - “There are 2 transactions - A and B. They each produce C.”
  - 与(AND)：If A and B then C
    - “A and B produce C.”
    - “A and B together are required  to produce C.”
    - “C is produced when A and B  are both present.”

# 二义性审查

- 逻辑操作的模糊
  - 或非(NOR)：If not [A or B] then C
    - “If it is not A or B then produce C.”
    - “If it is neither A nor B produce C.”
    - “If it is neither A or B produce C.”
    - “If A or B are both missing produce C.”
    - “The valid codes are A and B.”
    - “If it is none of the above produce C.”
    - “If it is not A and not B then produce C.”
  - 与非(NAND)：If not [A and B] then C
    - “If it is not A and B together then produce C.”
    - “As long as A and B are not together produce C.”
    - “Produce C unless A and B are both present.”
    - “If it is not A or it is not B then produce C.”

关于需求测试：

# 二义性审查

- 混乱的组合逻辑
  - “If A or B and C produce D.”

  是　　If [A or B] and C produce D?

  还是　　If A or [B and C] produce D?

- Harry “参加晚会”的规则
  - 规则1。如果Sally去，或者 Sarah去，Harry就去
  - 规则2。如果Sally 不和John同去， Harry就去
  - 规则3。如果Sarah 不和Bob同去， Harry就去

  - 请问：如果Sally去，Bob也去， Harry去吗？

    根据规则1， Harry去

    根据规则3，似乎Harry不去

关于需求测试：

# 二义性审查

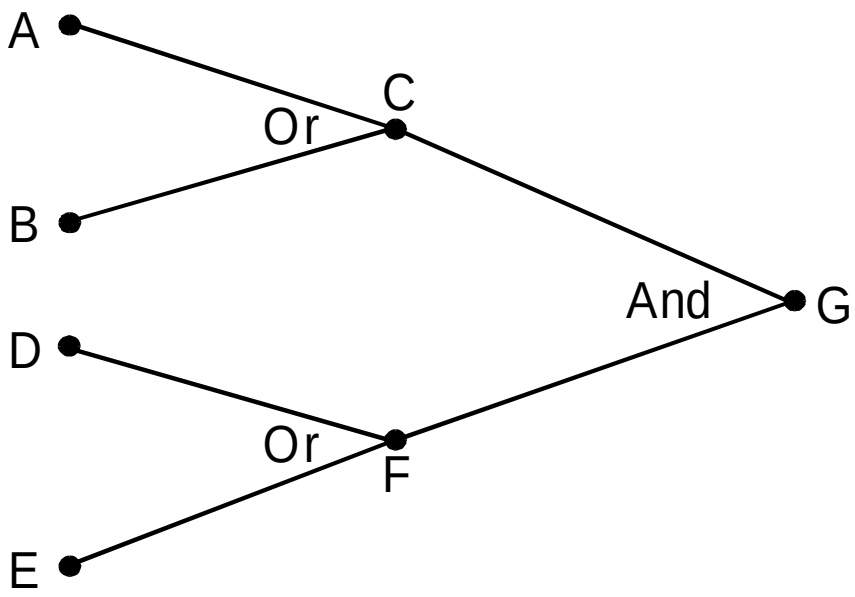
- 其他情况…
  - “在大街上骑一匹丑陋的马是违法的。”

    ——华盛顿-威尔伯法律
  - “中国代表团派出了强大阵容参加本届奥运会跳水比赛。奥运冠军吴敏霞和何姿将参加女子三米板双人和单人项目的比赛…”

    ——伦敦奥运会新闻

# 创建因果图

- 因果图法简介
- 图写逻辑关系
- 添加约束
- 从需求到因果图




If A or B, then C.
If D or E, then F.
If C and F, then G.

# 创建因果图

## ❖ 因果图法简介

### ❑ 背景：

典型的黑盒测试方法——等价类划分和边界值分析，是着重考虑输入条件，但没有考虑输入条件的各种组合、输入条件之间的相互制约关系。这样虽然各种输入条件可能出错的情况已经测试到了，但多个输入条件组合起来可能出错的情况却被忽视了。

如果在测试时必须考虑输入条件的各种组合，则可能组合数目将是天文数字，因此必须考虑采用一种适合于描述多种条件的组合、相应产生多个动作的形式来进行测试用例的设计，这就需要利用因果图（逻辑模型）。

创建因果图：

# 因果图法介绍

## ❖因果图法简介

- ❑因果图法是基于这样的一种思想：程序的功能可以用判定表（或称决策表）的形式来表示，并根据输入条件的组合情况规定相应的操作。
- ❑利用图解法分析输入的各种组合情况，从而设计测试用例，它适合于检查程序输入条件的各种组合情况。
- ❑采用因果图法设计测试用例的步骤：

1）根据程序规格说明书（即需求）描述，分析并确定“因”（输入条件）和“果”（输出结果或程序状态的改变），画出因果图。

2）将得到的因果图转换为判定表。

3）为判定表中每一列所表示的情况设计一个测试用例。

# 因果图法介绍

## 因果图法简介

### 使用因果图法的优点：

1）考虑到了输入情况的各种组合以及各个输入情况之的相互制约关系。

2）能够帮助测试人员按照一定的步骤，高效率的开发试用例。

3）因果图法是将自然语言需求转化成形式语言规格说的一种科学而严谨的方法。

创建因果图：

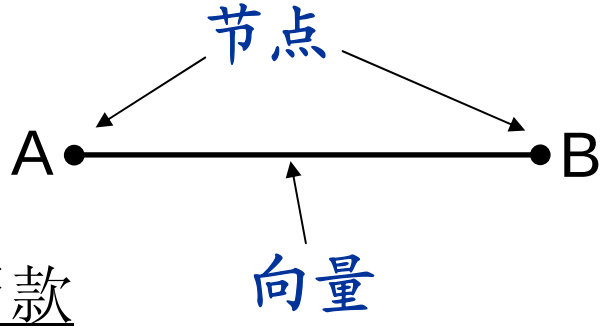
# 图写逻辑关系

- 因果图常用逻辑关系
  - 1. 简单 Simple
  - 2. 非 Negation（~）
  - 3. 或 Or （∨）
  - 4. 与 And （∧）
  - 5. 或非 Nor （↓）
  - 6. 与非 Nand （↑）
  - 7. 异或 Xor （⊻）
  - 8. 异或非 Nxor（⊼）

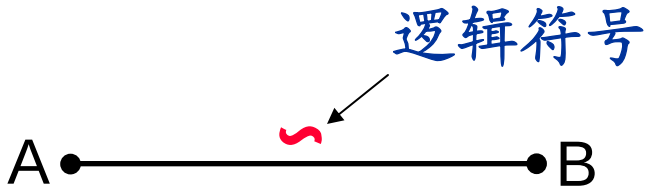
# 创建因果图：

## 图写逻辑关系

- 简单[Simple]
  - If A then B.
  - 例如：如果你闯红灯，就会被罚款




- 非 [Negation] （∽）
  - If not A then B.
  - 例如：如果明天不下雨，我就参加




节点是因果图的一个非常重要的概念

创建因果图：

# 图写逻辑关系

- 节点 [Node]
  - 变量及其状态
    - 红灯亮
    - **X>0**
  - 功能，或一个动作
    - 计算代数和
    - 点击鼠标右键
- 定义节点——变量状态
  - 将变量所有可能的输入值（输入域）分成若干个子集，用指定属性或作用在属性上的操作的方式：
    - 列表
    - 范围
    - 类别

创建因果图：

# 图写逻辑关系

- 定义节点

  举例：输入域是“某公司所有的员工”，划分子集：

  - 列表子集
    - 张三
    - 李四
    - 王五
    - …
  - 范围子集
    - 工号从**001**到**200**的所有人
    - …
  - 属性子集
    - 所有女员工
    - …

- 定义节点

  一般来说：

  - 列表
    - 短列表，每项定义一个节点
    - 长列表，一个节点
  - 范围
    - 一个节点
      - **True =** 在范围内
      - **False =** 不在范围内
  - 属性
    - 一个节点
      - **True =** 与属性相符
      - **False =** 与属性不符

创建因果图：

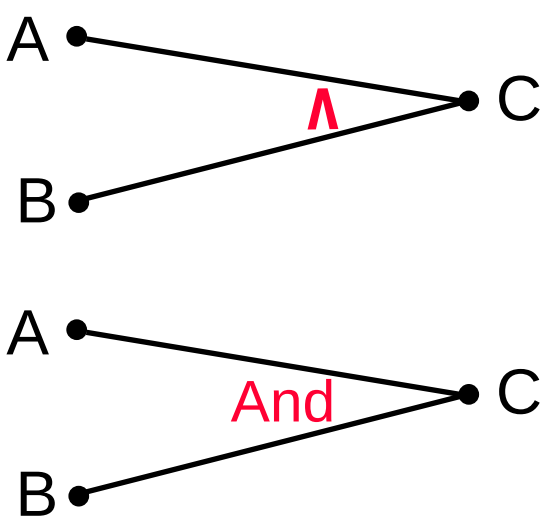
# 图写逻辑关系

- 定义节点
  - 迭代循环
    - 完成一个动作**X**次，为一个节点
    - 例如，“允许最多尝试**3**次”
      - **True =** 尝试纪录数≤**3**
      - **False =** 尝试纪录数＝**4**
  - 重复动作
    - 在一系列的步骤中的相同动作，每一个动作是一个节点
    - 例如，用户须在处理过程的**3**个不同点键入**“ENTER”**。定义为**3**个**“ENTER”**节点

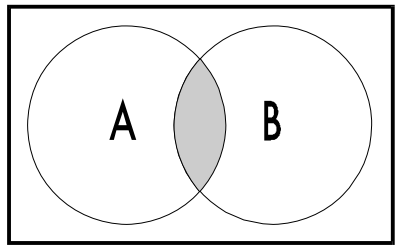
# 创建因果图：图写逻辑关系

## ❖与 [And] （∧）

- If A and B then C.




| A | B | C |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | F |
| F | F | F |



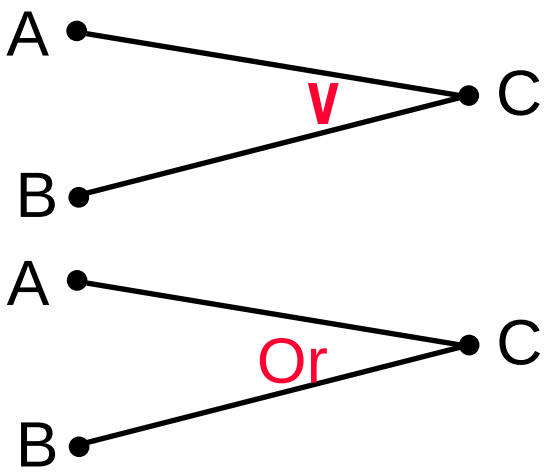

- 例如：如果你闯红灯并且被抓住，就会被罚款

创建因果图：

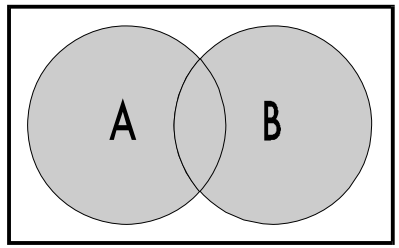
# 图写逻辑关系

## ❖或 [Or] （∨）

- If A or B then C.




| A | B | C |
|---|---|---|
| T | T | T |
| T | F | T |
| F | T | T |
| F | F | F |



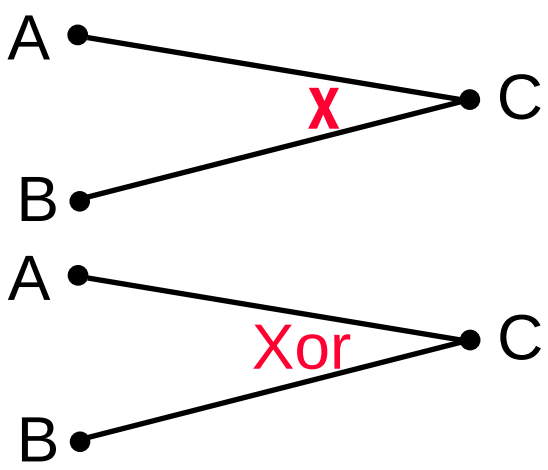

- 例如：如果你闯红灯或者逆行，就会被罚款

创建因果图：

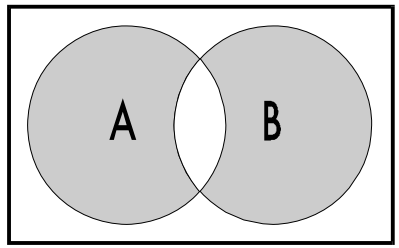
# 图写逻辑关系

## ❖异或[Xor]（x）

- If one and only one of A or B then C.




| A | B | C |
|---|---|---|
| T | T | F |
| T | F | T |
| F | T | T |
| F | F | F |



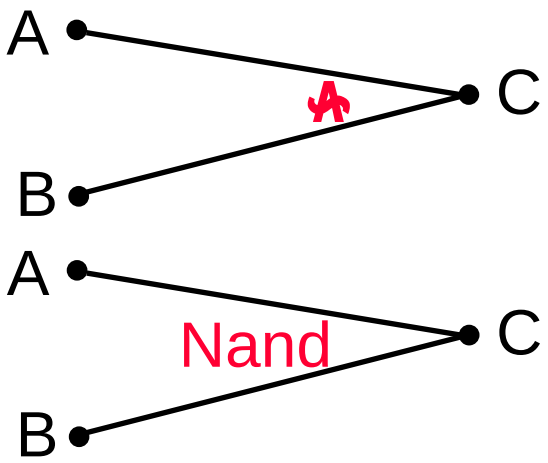

- 例如：如果客户已经购买了或者产品A或者产品B，就赠送产品C

创建因果图：

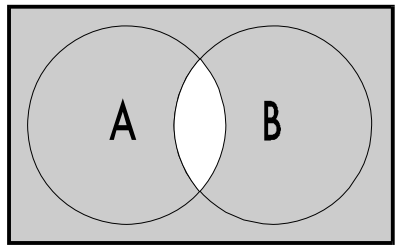
# 图写逻辑关系

## ❖与非[Nand]（↑）

- If not [A and B] then C.




| A | B | C |
|---|---|---|
| T | T | F |
| T | F | T |
| F | T | T |
| F | F | T |



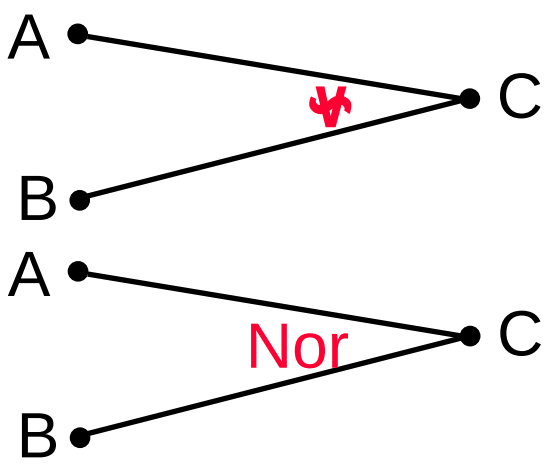

- 先“与”后“非”
- 例如：如果不是中国队对阵韩国队，我就去观战

创建因果图：

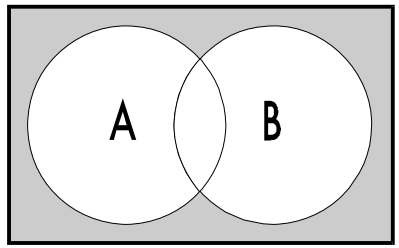
# 图写逻辑关系

## ❖或非[Nor]（↓）

- If not [A or B] then C.




| A | B | C |
|---|---|---|
| T | T | F |
| T | F | F |
| F | T | F |
| F | F | T |



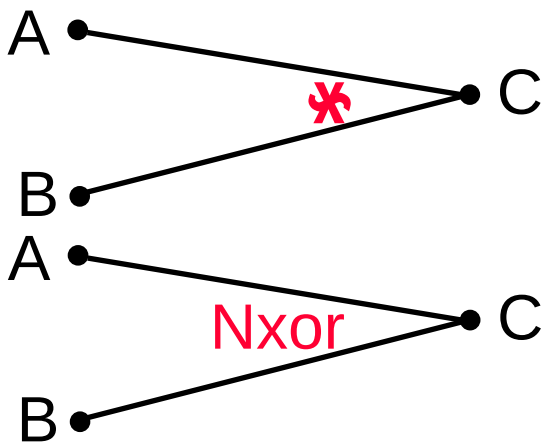

- 先“或”后“非”
- 例如：如果你全年既不缺勤也不迟到，你就会获得年度全勤奖

创建因果图：

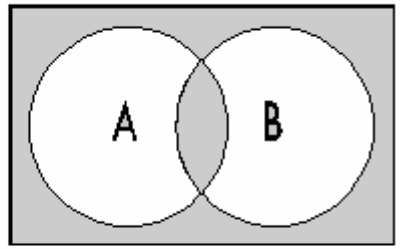
# 图写逻辑关系

## ❖异或非 [Nxor] (✲)

- If not [A xor B] then C.




| A | B | C |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | F |
| F | F | T |




- 先“异或”后“非”
- 例如：如果既没有选A也没有选B，或者既选了A又选了B，就显示错误信息

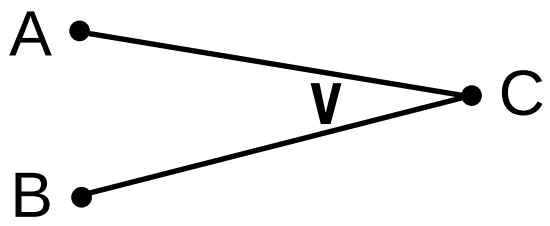
# 创建因果图：

## 图写逻辑关系

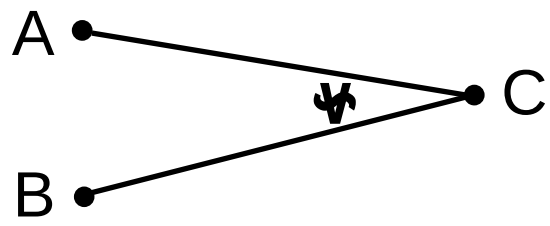
- 节点定义时不要暗含否定

  如果明天下雨或下雪，就不去登山

  - 不正确
    - A = 下雨
    - B = 下雪
    - C = 不登山




  - 正确
    - A = 下雨
    - B = 下雪
    - C = 登山



创建因果图：

## 图写逻辑关系

- 注意

$A \downarrow B = \tilde{A} \wedge \tilde{B}$

If neither A nor B = If not A and not B.

$A \uparrow B = \tilde{A} \vee \tilde{B}$

If not [A and B] = If not A or not B.

$A \uparrow B \neq \tilde{A} \wedge \tilde{B}$

If not [A and B] ≠ If not A and not B.

$A \downarrow B \neq \tilde{A} \vee \tilde{B}$

If not [A or B] ≠ If not A or not B.

创建因果图：

# 图写逻辑关系

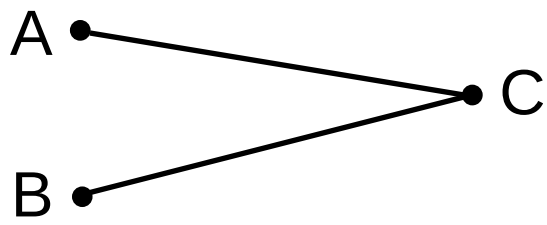
## 练习

1. 如果红灯亮，就停车.

A.
B.

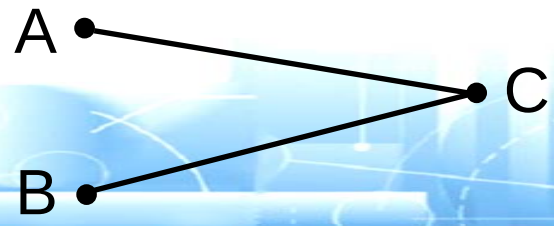
2. 如果操作码为‘2’并且金额超过 $50.00，就在报表上增设一个分录.

A.
B.
C.




3. 报告会在年终时，或在有请求的情况下打印.

A.
B.
C.



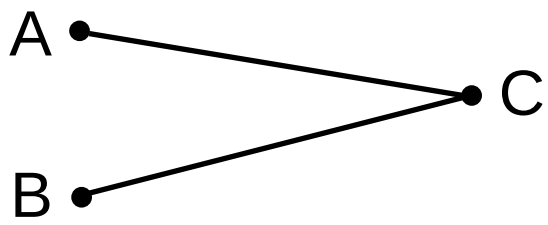
创建因果图：

# 图写逻辑关系

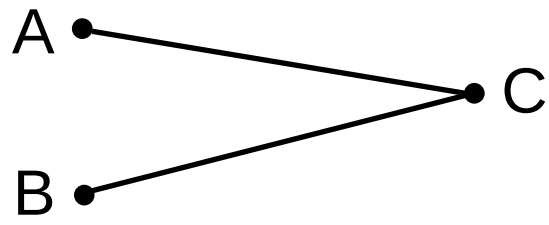
## 练习

4. If it does not rain or snow the tennis match will be played.

A.
B.
C.

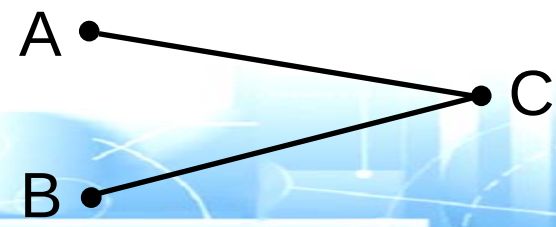



A.
B.
C.




5. If it does not rain or snow we will go skiing.

A.
B.
C.



创建因果图：

# 图写逻辑关系

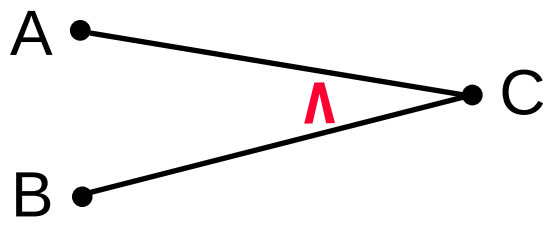
## 练习答案

1. 如果红灯亮了，就停车.

A. 红灯亮 （T=红灯亮，F=红灯不亮）
B. 停车 （T=停车，F=不停车）

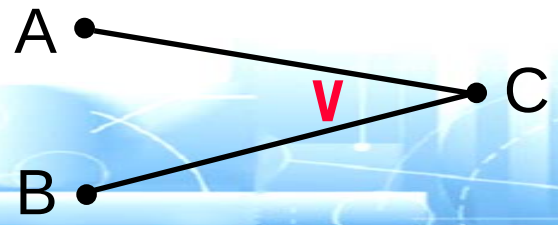

2. 如果操作码为‘2’并且金额超过 $50.00，就在报表上增一个条目.

A. 操作码=‘2’
B. 金额＞$50
C. 在报表上设置条目




3. 报表会在年终时，或在有请求的情况下打印.

A. 年终
B. 请求
C. 打印报表



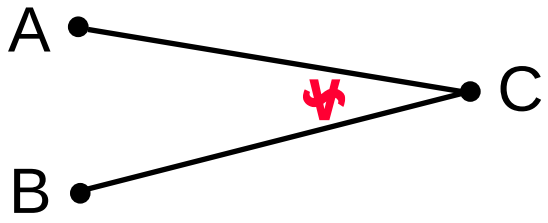
创建因果图：

# 图写逻辑关系

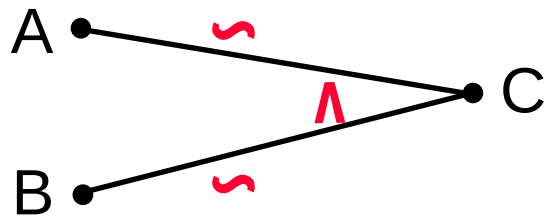
## ❖练习答案

4. If it does not [rain or snow] the tennis match will be played.

A. *Rain*
B. *Snow*
C. *Play tennis*

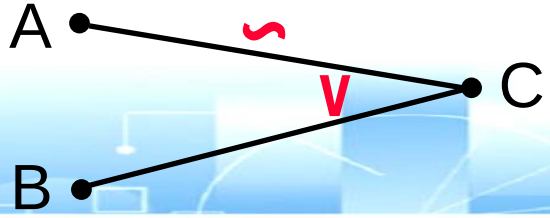



A. *Rain*
B. *Snow*
C. *Play tennis*




5. If it does [not [rain] or snow] we will go skiing.

A. *Rain*
B. *Snow*
C. *Go skiing*




创建因果图：

# 图写逻辑关系

## 基本逻辑结构

1. 简单条件
2. 组合条件
3. 同等条件 – (顺序不重要).
4. 嵌套条件 – (顺序相关).

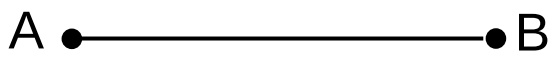
创建因果图：

# 图写逻辑关系

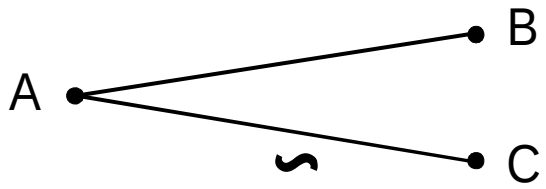
## 基本逻辑结构

### 简单条件

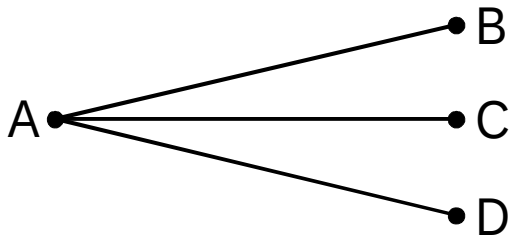
- If A, then B.



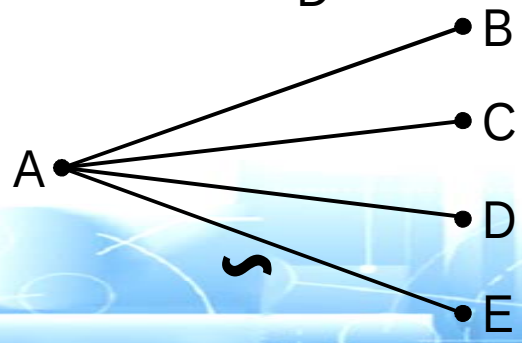

- If A, then B; else C.




- If A, then B, C, and D.




- If A, then B, C, and D; else E.



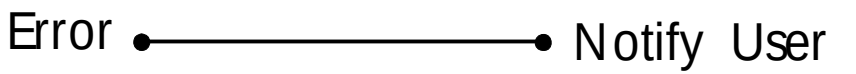
# 创建因果图：

## 图写逻辑关系

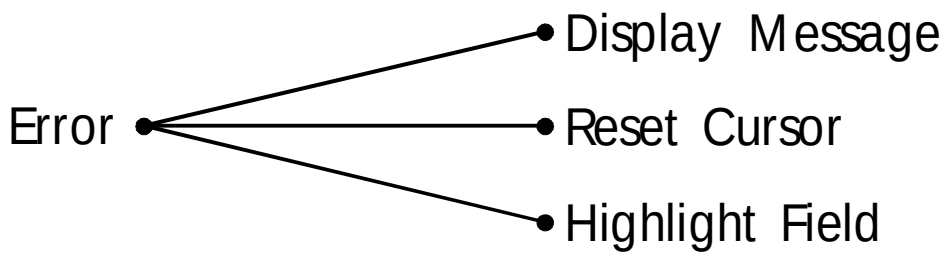
- 基本逻辑结构
  - 避免复合节点

“If error, then display message, reset cursor, and highlight the field.”

- 不正确




- 正确



# 创建因果图：

## 图写逻辑关系

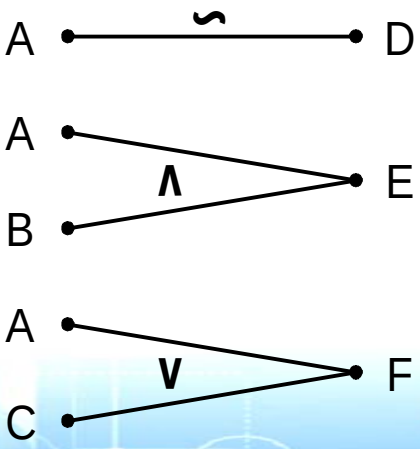
- 基本逻辑结构
  - 节点在因果图中要唯一

If not A then D.

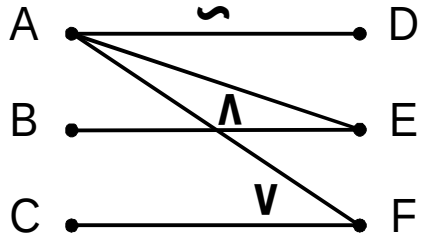
If A and B then E.

If A or C then F.

不正确




正确



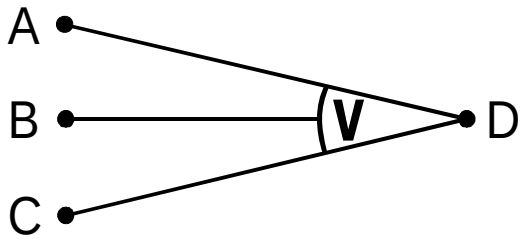
创建因果图：

# 图写逻辑关系

## 基本逻辑结构

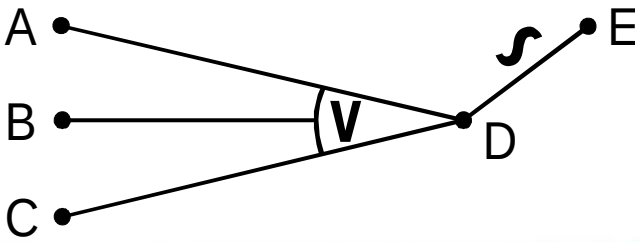
### 复合条件

- If A or B or C, then D.

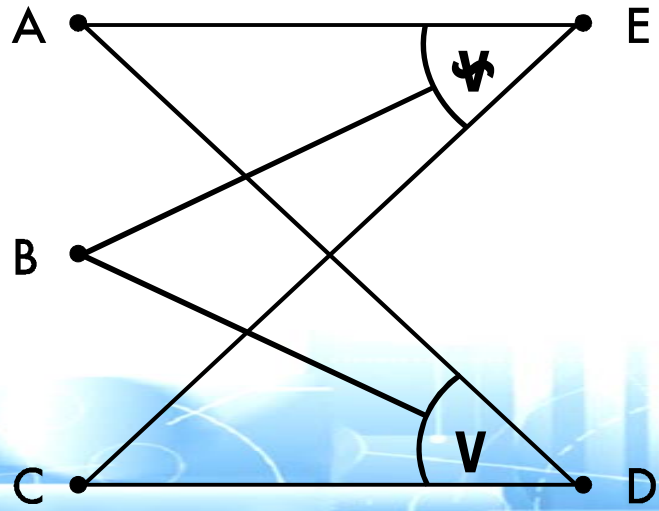



- If A or B or C, then D; else E.

简略形式.




详尽形式



# 创建因果图：图写逻辑关系

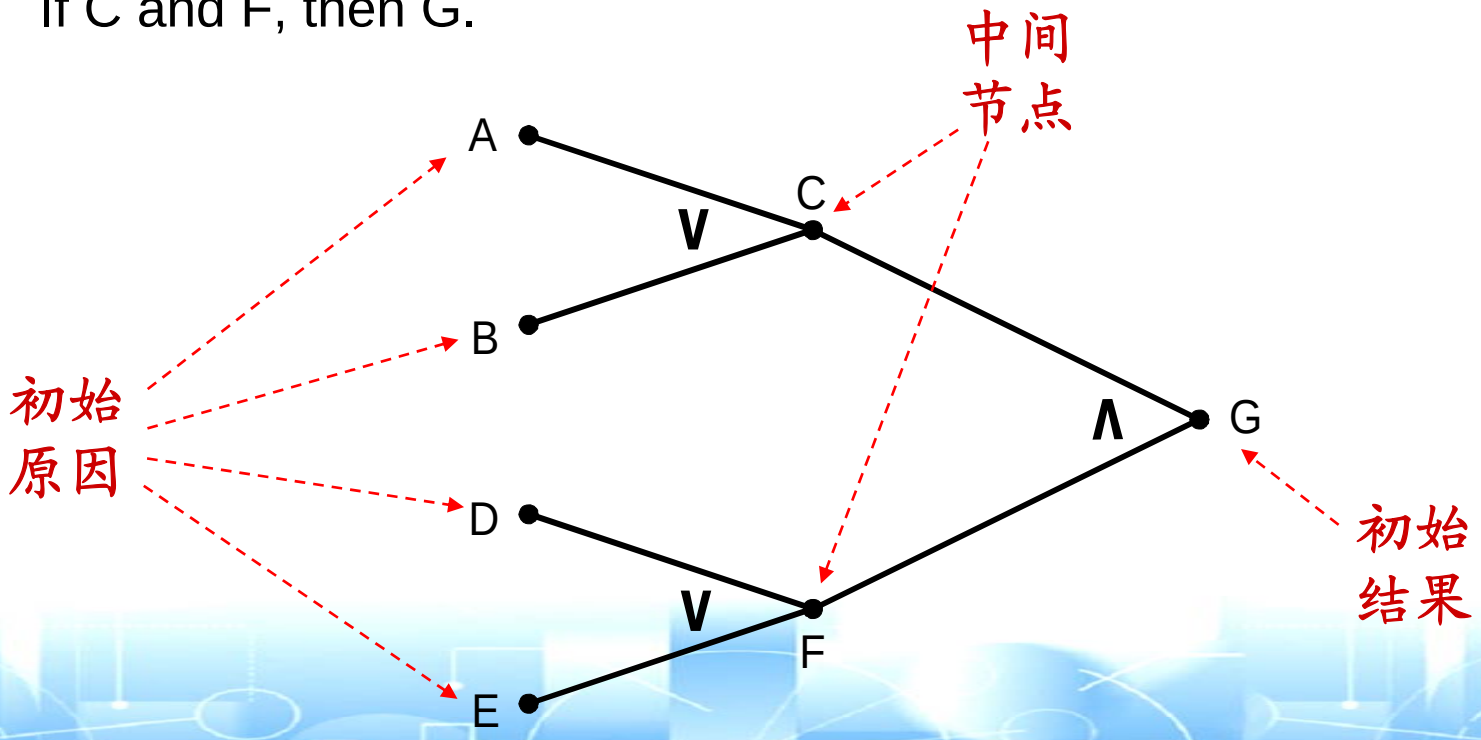
## 基本逻辑结构

### 同等条件（顺序无关）

If A or B, then C.

If D or E, then F.

If C and F, then G.



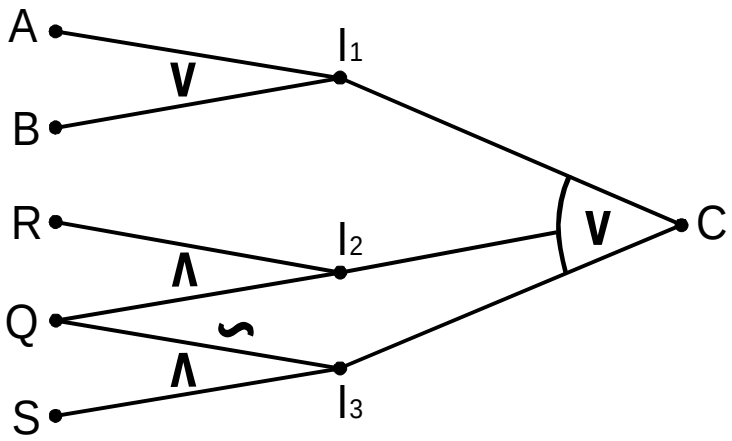
# 创建因果图：

## 图写逻辑关系

- 基本逻辑结构
  - 中间节点

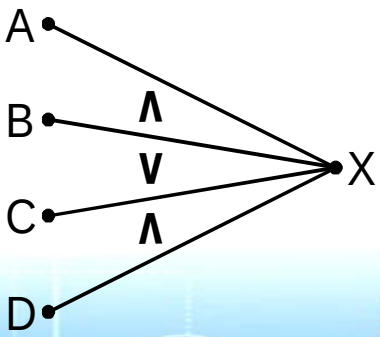
    桥梁和纽带，增加可读性

    If A or B then C.
    If Q and R then C.
    If S and not Q then C.

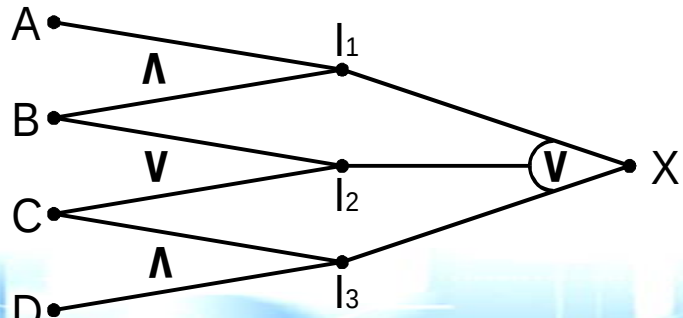



[If A or B] or [If Q and R] or [If S and not Q] then C.

  - 每个逻辑操作必须导致唯一的结果




不正确



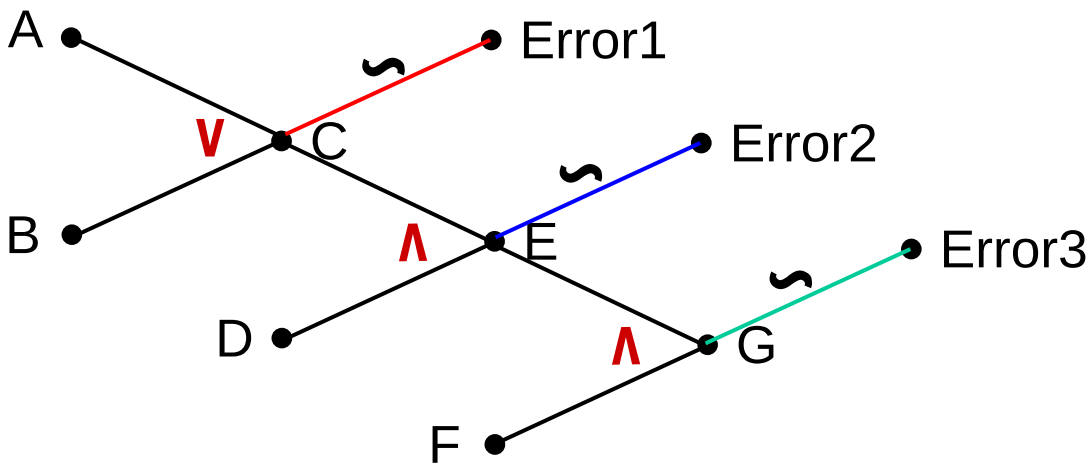

正确

# 创建因果图：

## 图写逻辑关系

- 基本逻辑结构
  - 嵌套条件（顺序相关）
    简略式

```
If A Or B
   Then C
   If C And D
      Then E
      If E And F
         Then G
         Else Error3
      Else Error2
   Else Error1.
```

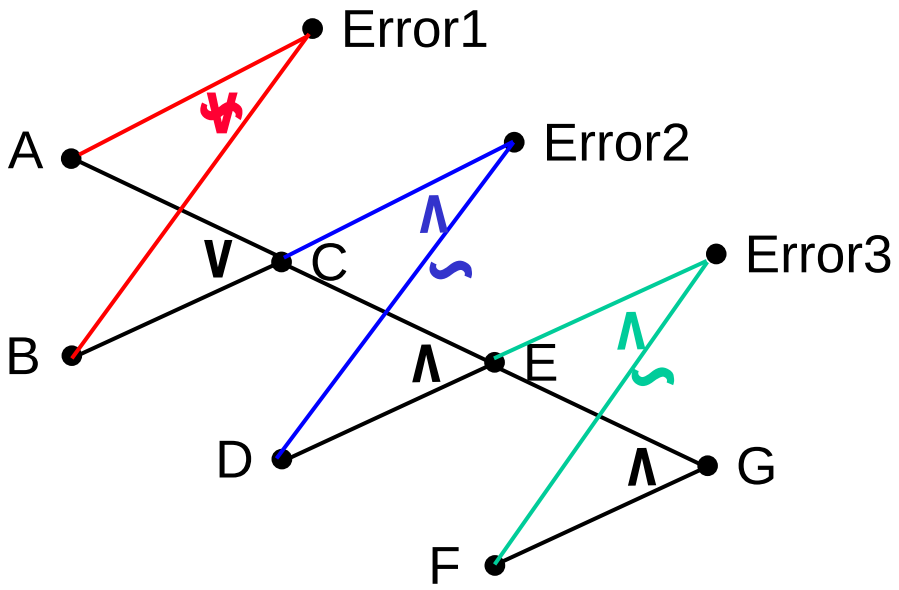


# 创建因果图：

## 图写逻辑关系

- 基本逻辑结构
  - 嵌套条件（顺序相关）
    详尽式

```
If A Or B
   Then C
   If C And D
      Then E
      If E And F
         Then G
         Else Error3
      Else Error2
   Else Error1.
```



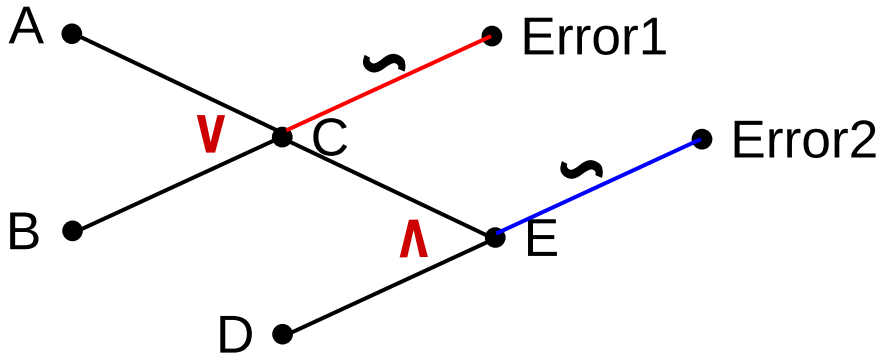
# 创建因果图：

## 图写逻辑关系

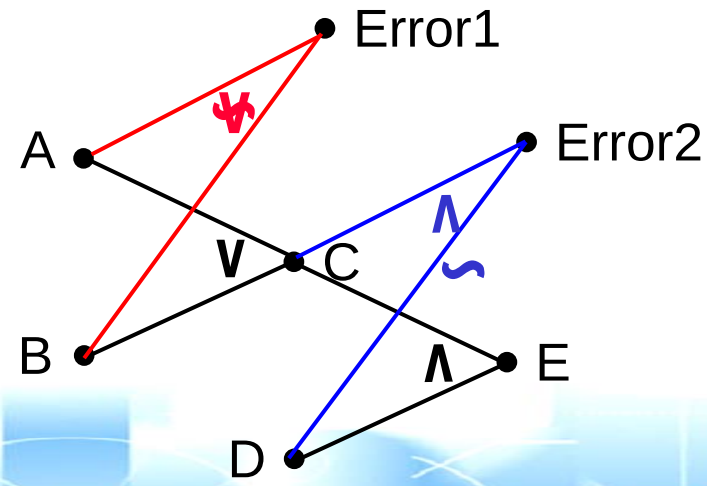
- 基本逻辑结构
  - 嵌套条件（顺序相关）

    else的不同

```
If A Or B
   Then C
   If C And D
      Then E
      If E And F
         Then G
         Else Error3
      Else Error2
   Else Error1.
```




ERROR 2: C True and D False;
C False and D True;
C False and D False.




ERROR 2: C True and D False.

创建因果图：

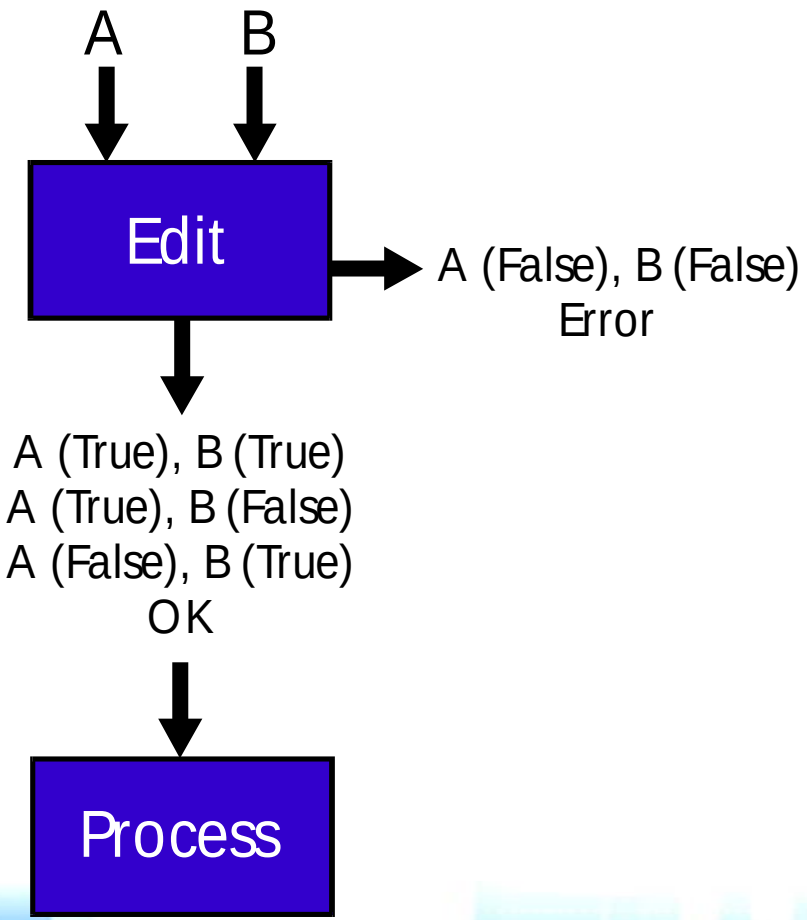
# 添加约束

- 约束
  - 环境约束
    - 预编规则
    - 数据的物理结构
  - 边界约束
    - 互斥（Exclusive）
    - 唯一（ One ）
    - 包含（ Inclusive ）
    - 需要（Exclusive）
    - 固化（ Anchor ）
  - 屏蔽
    - 属性屏蔽（Attribute Mask ）
    - 居先屏蔽（Precedence Mask）

创建因果图：

# 添加约束

- 环境约束
  - 预编规则




A (False), B (False)
的用例永远不会处理

# 创建因果图：

## 添加约束

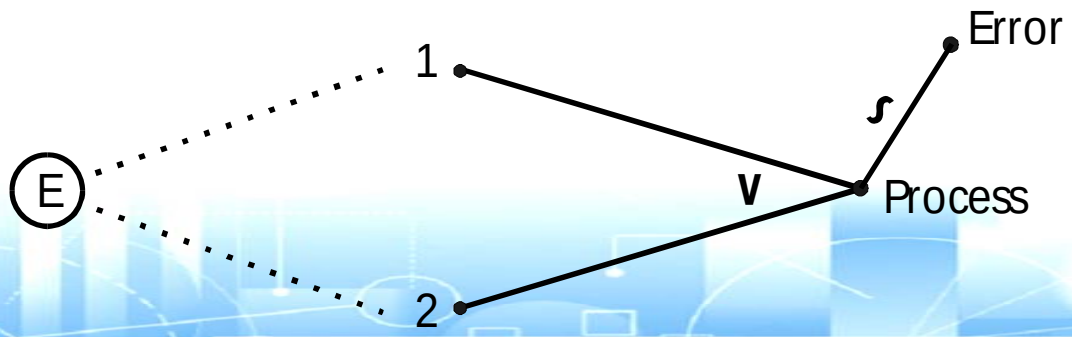
- 边界约束
  - 互斥 [Exclusive]（E）

    最多一个

  - 例如：如果交易码是1 或 2，就处理之。否则，显示错误信息
    - 1和2最多有一个成立

Exclusive

| A | B | Effect |
|---|---|---|
| T | T | Infeasible |
| T | F | OK |
| F | T | OK |
| F | F | OK |




在互斥约束下，两个或多个节点为TRUE是不可能的

创建因果图：

# 添加约束

- 边界约束
  - 唯一[One] （O）

    有且仅有一个
  - 例如：今天要么是周一，要么是周二…，必定是周一到周日中的一天
  - 包含[Inclusive] （I）

    至少一个

One And Only One

| A | B | Effect |
|---|---|---|
| T | T | Infeasible |
| T | F | OK |
| F | T | OK |
| F | F | Infeasible |

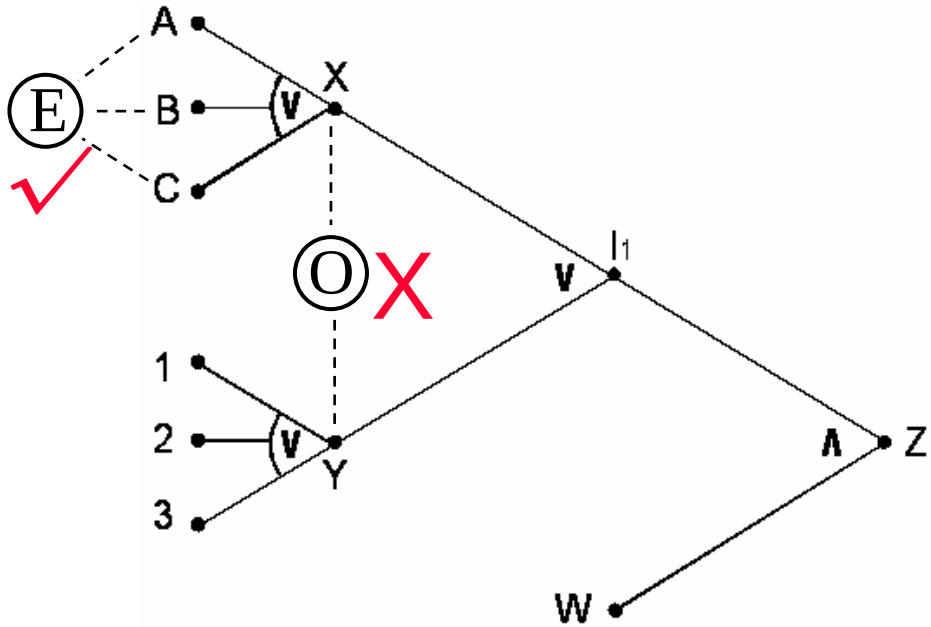
Inclusive

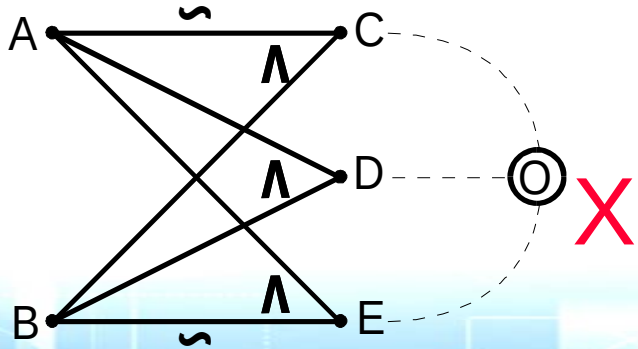
| A | B | Effect |
|---|---|---|
| T | T | OK |
| T | F | OK |
| F | T | OK |
| F | F | Infeasible |

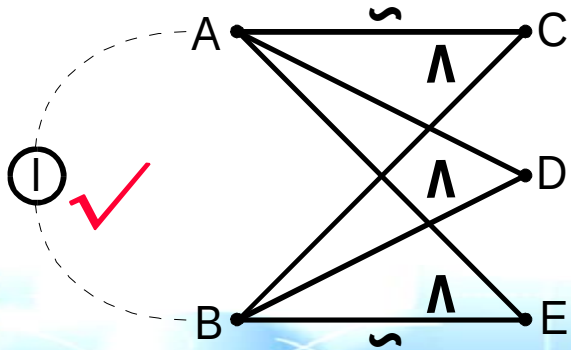
# 创建因果图：

## 添加约束

- 边界约束
  - 互斥、唯一、包含只能用在“初始原因”节点上
  - 不可用约束去强制改变逻辑关系



创建因果图：

# 添加约束

## 边界约束

### 需要[Require] （R）

需要某种前提条件：

- 欲使变量处在某一状态，需要一个或多个变量处在指定的状态
- 需要是单向的
- 例如：若B(True)以A(True)为前提，就是B需要A

Requires

| A | B | Effect |
|---|---|---|
| T | T | OK |
| T | F | OK |
| F | T | Infeasible |
| F | F | OK |


### 固化[Anchor] （A）

强制节点固定在唯一的状态

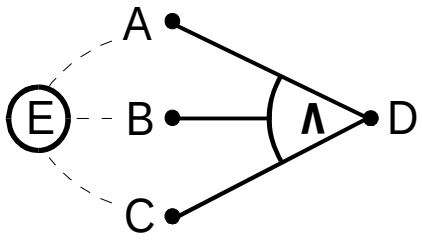
- 固化可设为TRUE或FASLE
- 例如：在操作过程中，主菜单始终有效。

创建因果图：

# 添加约束

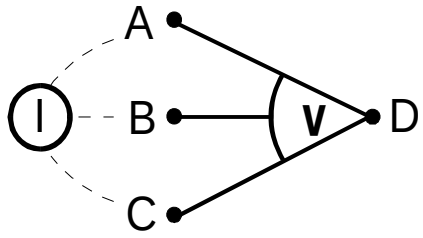
- 约束和逻辑不一致性
  - 前置条件可能与内部处理规则相冲突

1.



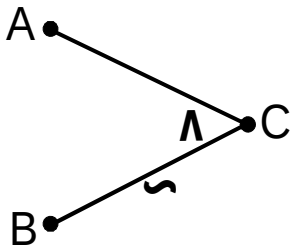

**D 始终不为 true.**

2.




**D 恒为 true.**

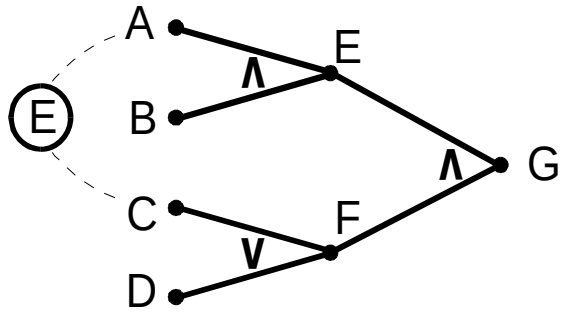
3.




A (T) 需要 B (T)

**C 始终不为 true.**

4.




D (T) 需要 C (T)
G 可能为 True吗?

**G 始终不为 true**：如果A 为true，C就为false。而C如果是false，D就是false，致使 F为false.

创建因果图：

# 添加约束

- 屏蔽[Mask] （M）
  - 什么是屏蔽
    - 一个节点的状态掩盖了其他节点的状态，使其不产生作用
  - 属性屏蔽
    - 如果B是A的一个属性，A不存在B也就不存在，是为A屏蔽B
    - 例如：账户结余是账户的一个属性
  - 居先屏蔽
    - 一个节点的状态导致其他节点被旁路
    - 例如：如果发现错误，便终止处理
  - 术语
    - Subject：对其他节点施加屏蔽的节点及其状态
    - Object：被屏蔽的节点
    - 例如：节点A屏蔽了节点B和节点C，那么A就是该屏蔽的Subject，B和C就是Object

# 从需求到因果图

## 例子：透支保护授权

### 需求：

如果一个用户是商业用户或者是优选个人用户，其支票帐户有100000元或以上的存款，目前无透支保护且在过去的12个月内透支少于5次，那么就授予该用户免费透支保护，否则就不授予。

### 步骤

- 分析需求
- 定义节点
- 绘制因果图
- 添加约束

创建因果图：

# 从需求到因果图

- 例子：透支保护授权
  - 分析需求：

If 商业用户 or 优选个人用户，
and
支票帐户存款≥100000元
and
目前无透支保护
and
12个月内透支少于5次，
then
授予免费透支保护.
else
不授予

创建因果图：

# 从需求到因果图

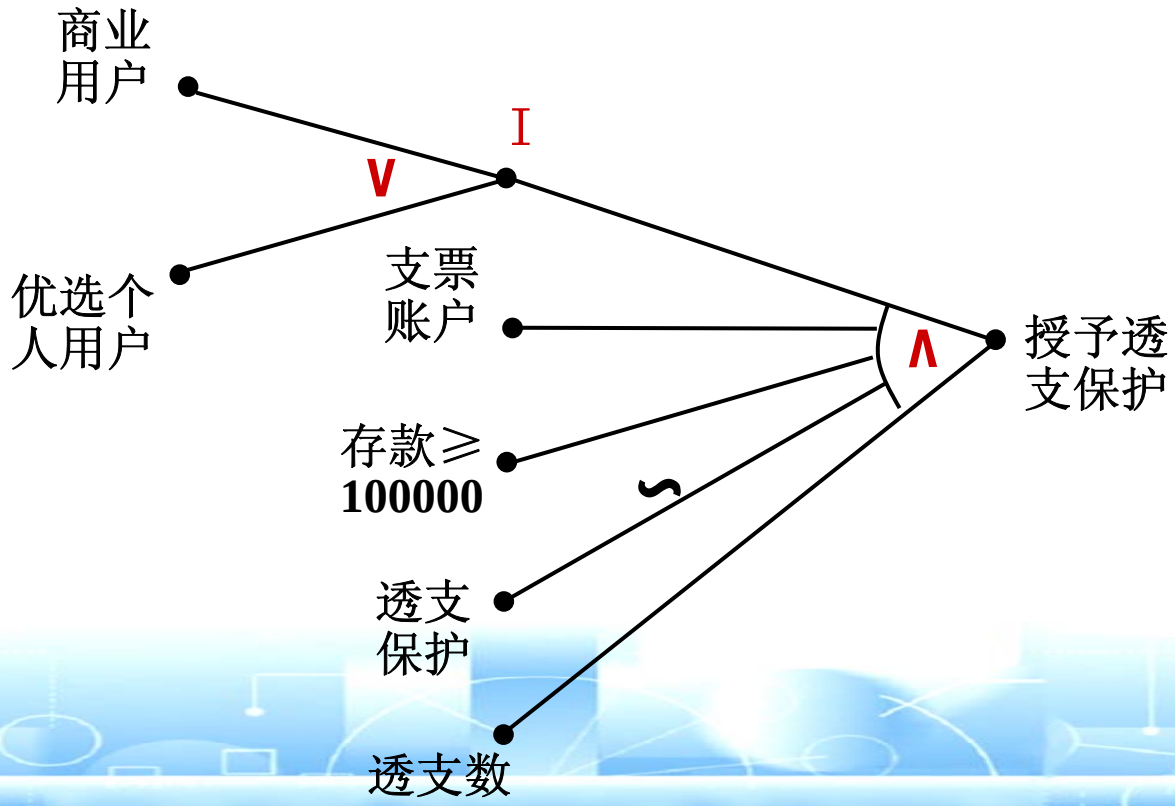
- 例子：透支保护授权
  - 定义节点

因
- 商业用户：(1=商业用户，0=非商业用户)
- 优选个人用户：(1=优选个人用户，0=非优选个人用户)
- 存款≥100000：(1=存款≥十万元，0=存款<十万)
- 无透支保护：(1=无透支保护，0=有透支保护)　内含否定
- 透支数<5：(1=12个月内透支数<5，0=透支数≥5)
- 支票账户：(1=有支票账户，0=没有支票账户)
- 透支保护：(1=有透支保护，0=无透支保护)

果
- 授予透支保护：(1=授予透支保护，0=不授予)

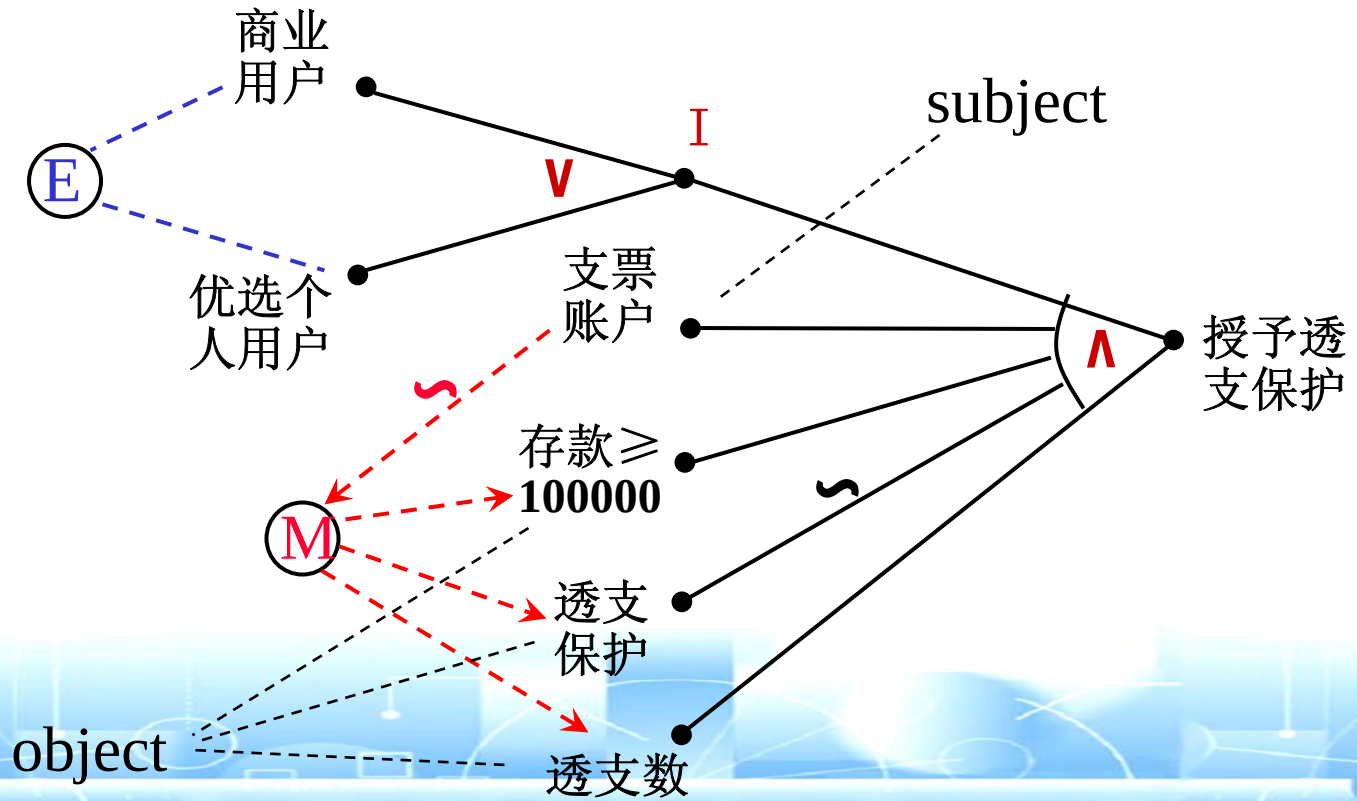
# 创建因果图：从需求到因果图

- 例子：透支保护授权
  - 绘制因果图
    - 增加一个中间节点：I = 商业用户 or 优选个人用户
    - 注意透支保护节点要取反



# 创建因果图：从需求到因果图

- 例子：透支保护授权
  - 添加约束
    - 商业用户 和 优选个人用户 是互斥的关系
    - 无支票账时户屏蔽存款额、透支保护、透支次数属性。



# 设计测试用例

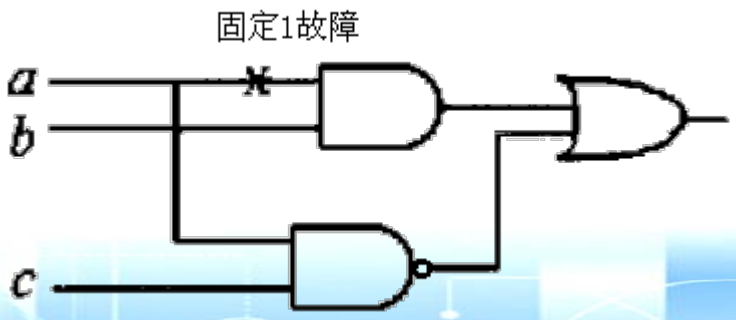
- 目标
  - 从黑盒的角度设计一套必须且充分的测试用例，使所有的功能需求全部得到测试

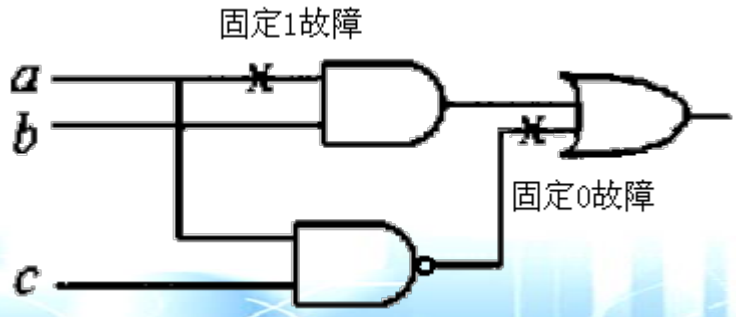
- RBT的测试用例设计技术
  - 敏感路径法
    - 采用已得到证明的、用于硬件电路逻辑测试的相同技术。
    - 能够发现那种相互直接消除的缺陷和那些被其他先前正确工作的逻辑所掩盖的缺陷
    - 确保因为正确的“激励”得到正确的“响应”

测试用例设计：

# 敏感路径法

- 来源于硬件电路逻辑测试技术
  - 逻辑电路中常见固定型故障。即逻辑门的输入或者输出值永远固定不变（恒为1或恒为0），与电路的原始输入无关。
  - 电路中的元件损坏、连线的开路和相当一部分的短路故障都可以用固定型故障模型比较准确的描述出来。
  - 如果一个电路中只存在一个固定型故障，称之为单固定型故障；如果有两个或两个以上，则称为多固定型故障。



测试用例设计：

# 敏感路径法

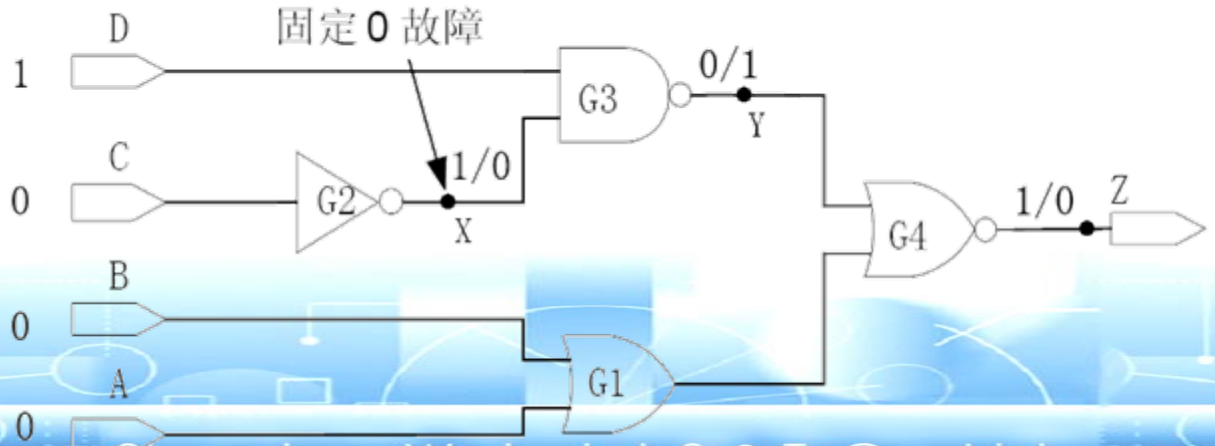
## 基本思路

- 在输入端选择合适的测试向量把故障传播到输出端，使故障时的输出不同于正常输出，进而找到故障之所在
  - 这种基于故障传播路径生成测试向量的方法，称为敏感路径法（亦称敏化路径法）
- 为了把故障传播到外部输出，要满足两个条件
  - 输入测试向量能够使得故障点在正常情况下与故障情况下的状态值不同
  - 至少有一个输出端的正常值与有故障时的值不同。为了能做到这一点，要求从故障点出发找到一条或几条路径到达输出端，使得该路径上各节点的正常值与有故障时的值不同，这条路径称为敏化路径。

# 测试用例设计：敏感路径法

## 一个例子

- 一个4门、4输入的逻辑电路。假设G2的输出端X处有固定0 故障。应如何检测？
  - 首先，必须控制G2的输入使其输出为1（与故障值相反）
    - 故 C = 0
  - 其次，要控制其它输入值，使最后的输出Z只决定于G2的输出
    - 令D = 1，则G3 的输出只决定于G2 的输出，X传播到Y
    - 令A = 0，B = 0，则G4 的输出也只决定于G2 的输出，Y传播到Z
  - 所以，G2 输出端固定0 故障的测试向量为
    - **T = {(0，0，0，1)；(1)}，其正常输出为1，故障输出为0**



# 测试用例设计：

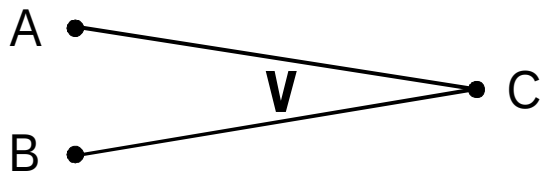
## 如何检测错误

- 软件中的“固定0”和“固定1”错误
  - 需求：“If A and B and C then produce D”
    编码：“If A and B then produce D”
    C被固定为“1”
  - 需求：“If A then do X, else if B then do Y”
    编码：“If A then do X, else do Y”
    程序默认B为“1”
  - 需求：“If A or B or C then produce D”
    编码：“If A or B then produce D”
    C被固定为“0”
  - 程序员设置一个参数作为操作码，但它不在有效范围之内
    操作码固定为“无效”

# 测试用例设计：

## 如何检测错误

- 例1：
  If A or B then C.

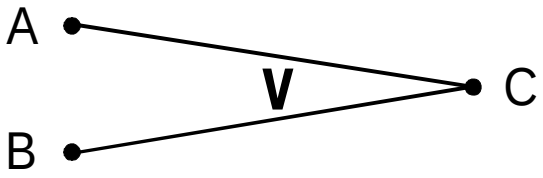



如果A固定为TRUE，则结果是：

| A | B | | C |
|---|---|---|---|
| T | T | | T |
| T | F | | T |
| F | T | | T |
| F | F | | T |

Detectable Error

- 例2
  If A or B then C.




如果A钳制为FALSE，结果是：

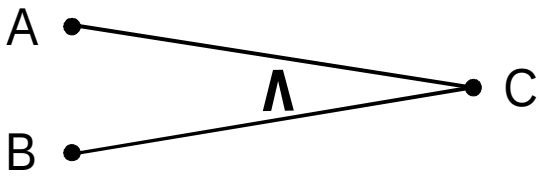
| A | B | | C |
|---|---|---|---|
| T | T | | T |
| T | F | | F |
| F | T | | T |
| F | F | | F |

Detectable Error

# 测试用例设计：

## 如何检测错误

- 例3：
  If A and B then C.

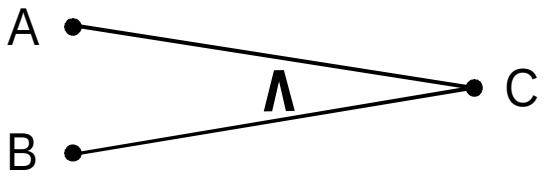



如果A固定为TRUE，则结果是：

| A | B | | C |
|---|---|---|---|
| T | T | | T |
| T | F | | F |
| F | T | | (T) |
| F | F | | F |

Detectable Error

- 例4
  If A and B then C.




如果A钳制为FALSE，结果是：

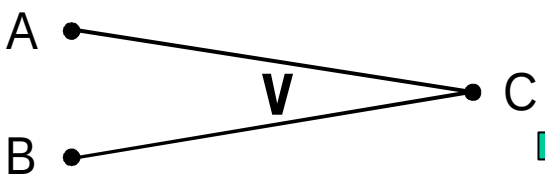
| A | B | | C |
|---|---|---|---|
| T | T | | (F) |
| T | F | | F |
| F | T | | F |
| F | F | | F |

Detectable Error

# 测试用例设计：

## 如何检测错误

- 简单的“或”有多少可能的错误？
  - 独立错误数：2 x 2 = 4
  - 组合错误数：$2^2 = 4$
  - 总错误数：8

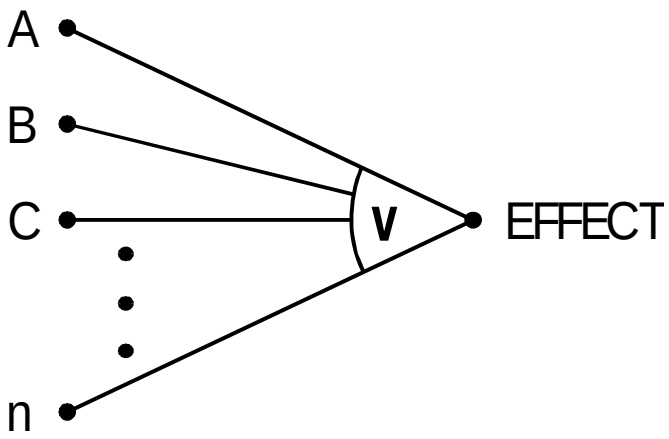



| | A | B | | C | | 0 OK | 1 OK | OK 0 | OK 1 | 1 1 | 1 0 | 0 1 | 0 0 | A B |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | T | T | | T | | T | T | T | T | T | T | T | F | |
| ⇨ | T | F | | T | | F | T | T | T | T | T | T | F | |
| ⇨ | F | T | | T | | T | T | F | T | T | T | T | F | |
| ⇨ | F | F | | F | | F | T | F | T | T | T | T | F | |
| | | | | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |

- 需要多少个测试用例？
  - FV1 - If A and not B then C ---可测试1和8
  - FV2 - If B and not A then C ---可测试3和8
  - FV3 - If not A and not B then not C ---可测试2, 4--7

# 测试用例设计：所需的功能变例

- “或” OR ( V )




| | A | B | C | | n | effect |
|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 1 | 0 | 0 | 0 | 1 |
| 3 | 0 | 0 | 1 | 0 | 0 | 1 |
| | 0 | 0 | 0 | 1 | 0 | 1 |
| n | 0 | 0 | 0 | 0 | 1 | 1 |
| n+1 | 0 | 0 | 0 | 0 | 0 | 0 |

1. A(T), B(F), C(F),..., n(F) then effect True
2. A(F), B(T), C(F),..., n(F) then effect True
3. A(F), B(F), C(T),..., n(F) then effect True

⋮ ⋮

n. A(F), B(F), C(F),..., n(T) then effect True

n+1. A(F), B(F), C(F),..., n(F) then effect False

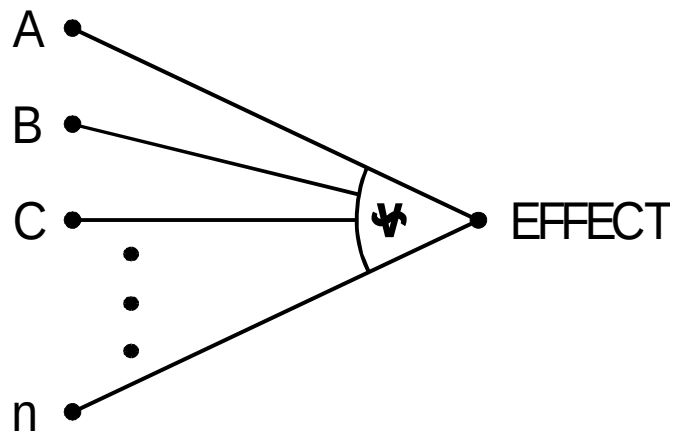
什么是功能变例

- 能够发现所有错误的最少功能组合
- 全部功能组合的一部分

# 测试用例设计：

## 所需的功能变例

- “或非” NOR (↓)




| | A | B | C | … | n | effect |
|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 | 0 | 0 |
| … | 0 | 0 | 0 | 1 | 0 | 0 |
| n | 0 | 0 | 0 | 0 | 1 | 0 |
| n+1 | 0 | 0 | 0 | 0 | 0 | 1 |

1. A(T), B(F), C(F),..., n(F) then effect False
2. A(F), B(T), C(F),..., n(F) then effect False
3. A(F), B(F), C(T),..., n(F) then effect False

⋮ ⋮

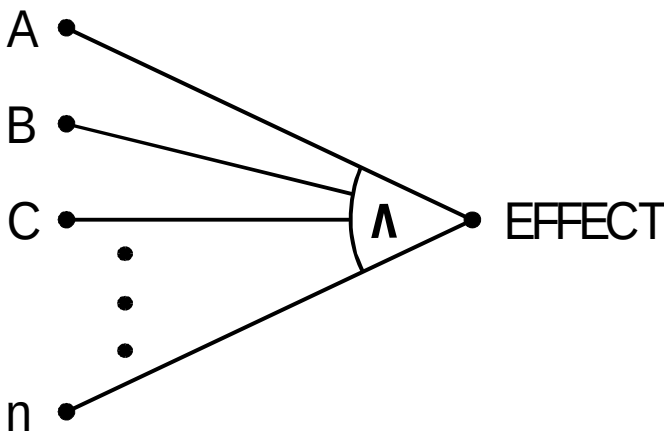
n. A(F), B(F), C(F),..., n(T) then effect False

n+1. A(F), B(F), C(F),..., n(F) then effect True

# 测试用例设计：

## 所需的功能变例

- “与” AND (∧)




| | A | B | C | … | n | effect |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 2 | 1 | 0 | 1 | 1 | 1 | 0 |
| 3 | 1 | 1 | 0 | 1 | 1 | 0 |
| … | 1 | 1 | 1 | 0 | 1 | 0 |
| n | 1 | 1 | 1 | 1 | 0 | 0 |
| n+1 | 1 | 1 | 1 | 1 | 1 | 1 |

1. A(F), B(T), C(T),..., n(T) then effect False
2. A(T), B(F), C(T),..., n(T) then effect False
3. A(T), B(T), C(F),..., n(T) then effect False

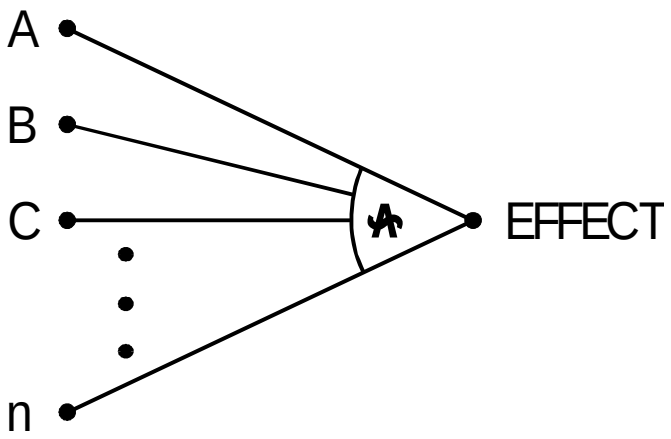
⋮ ⋮

n. A(T), B(T), C(T),..., n(F) then effect False

n+1. A(T), B(T), C(T),..., n(T) then effect True

# 测试用例设计：所需的功能变例

- “与非” NAND (↑)




| | A | B | C | … | n | effect |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 0 | 1 | 1 | 1 | 1 |
| 3 | 1 | 1 | 0 | 1 | 1 | 1 |
| … | 1 | 1 | 1 | 0 | 1 | 1 |
| n | 1 | 1 | 1 | 1 | 0 | 1 |
| n+1 | 1 | 1 | 1 | 1 | 1 | 0 |

1. A(F), B(T), C(T),..., n(T) then effect True
2. A(T), B(F), C(T),..., n(T) then effect True
3. A(T), B(T), C(F),..., n(T) then effect True

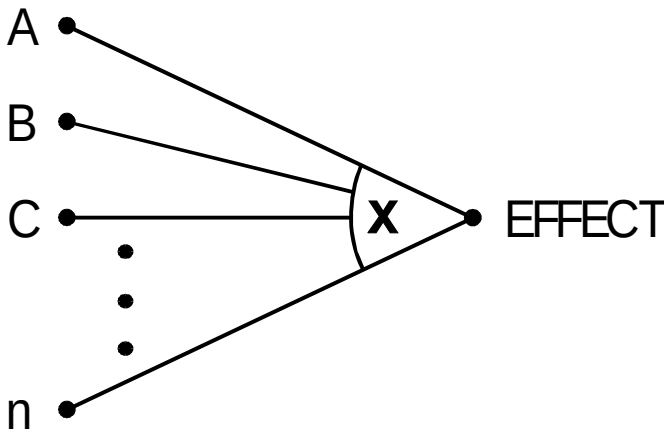
⋮ ⋮

n. A(T), B(T), C(T),..., n(F) then effect True

n+1. A(T), B(T), C(T),..., n(T) then effect False

# 测试用例设计：所需的功能变例

- “异或” XOR (X)




| | A | B | C | … | n | effect |
|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 1 | 0 | 0 | 0 | 1 |
| 3 | 0 | 0 | 1 | 0 | 0 | 1 |
| … | 0 | 0 | 0 | 1 | 0 | 1 |
| n | 0 | 0 | 0 | 0 | 1 | 1 |
| n+1 | 0 | 0 | 0 | 0 | 0 | 0 |

1. A(T), B(F), C(F),..., n(F) then effect True
2. A(F), B(T), C(F),..., n(F) then effect True
3. A(F), B(F), C(T),..., n(F) then effect True

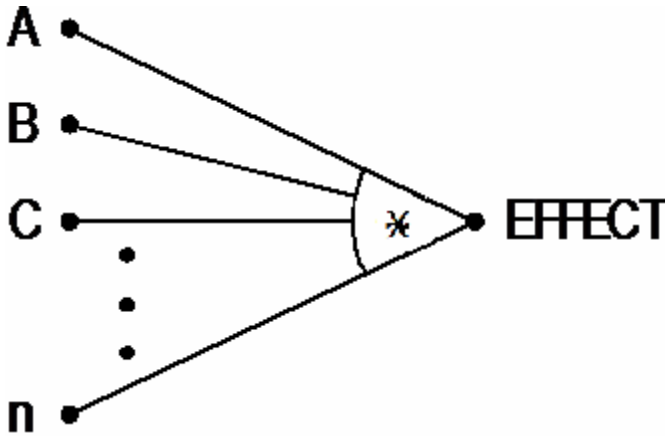
⋮ ⋮

n. A(F), B(F), C(F),..., n(T) then effect True

n+1. A(T), B(T), C(T),..., n(T) then effect False

# 测试用例设计：所需的功能变例

- “异或非” NXOR (⁂)




| | A | B | C | … | n | effect |
|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 | 0 | 0 |
| … | 0 | 0 | 0 | 1 | 0 | 0 |
| n | 0 | 0 | 0 | 0 | 1 | 0 |
| n+1 | 0 | 0 | 0 | 0 | 0 | 1 |

1. A(T), B(F), C(F),..., n(F) then effect False
2. A(F), B(T), C(F),..., n(F) then effect False
3. A(F), B(F), C(T),..., n(F) then effect False

⋮ ⋮

n. A(F), B(F), C(F),..., n(T) then effect False

n+1. A(T), B(T), C(T),..., n(T) then effect True

# 测试用例设计：所需的功能变例

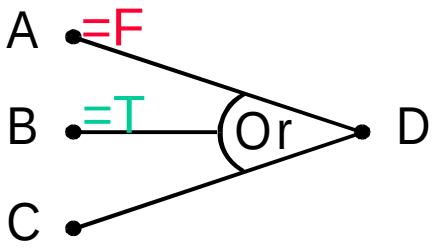
- 归纳

| 输入数 | 所需的功能变例 | 可能的组合数 | 可能的错误数 |
|---|---|---|---|
| n | n+1 | $2^n$ | $\sum_{x=1}^{n}\binom{n}{x}2^x$ |
| 2 | 3 | 4 | 8 |
| 3 | 4 | 8 | 26 |
| 4 | 5 | 16 | 80 |
| 5 | 6 | 32 | 242 |
| 6 | 7 | 64 | 728 |

Note: $\sum_{x=1}^{n}\binom{n}{x}2^x = 3^n - 1$

# 测试用例设计：所需的功能变例

- 缺陷的“屏蔽”




| | | | | |
|---|---|---|---|---|
| 1. | A | — | — | D |
| 2. | — | B | — | D |
| 3. | — | — | C | D |
| 4. | — | — | — | — |

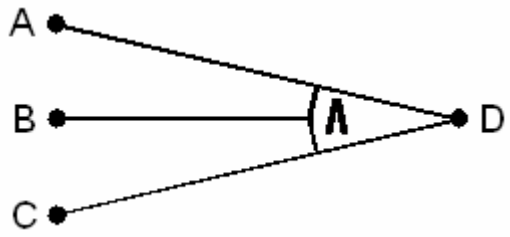
假设A固定为True，B固定为False
执行功能变例结果如下：

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1. | A | — | — | as | — | B | — | D |
| 2. | — | B | — | as | — | B | — | D |
| 3. | — | — | C | as | — | B | C | D |
| X4. | — | — | — | as | — | B | — | (D) |

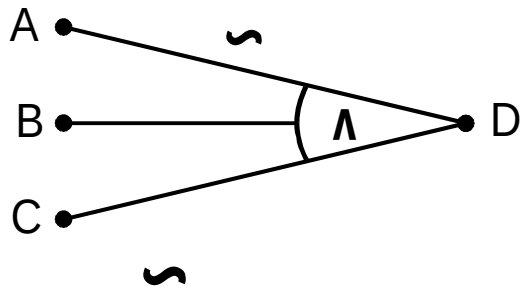
1、2、3看起来“正常”，只有4才能发现问题！
你可能由于错误的原因得到正确的结果！

# 测试用例设计：所需的功能变例

- 功能变例矩阵




| | A | B | C | D |
|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 |
| 2 | 1 | 0 | 1 | 0 |
| 3 | 1 | 1 | 0 | 0 |
| 4 | 1 | 1 | 1 | 1 |




| | A | B | C | D |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 1 | 0 | 0 |
| 4 | 0 | 1 | 1 | 1 |

其中： 1 = True, 0 = False

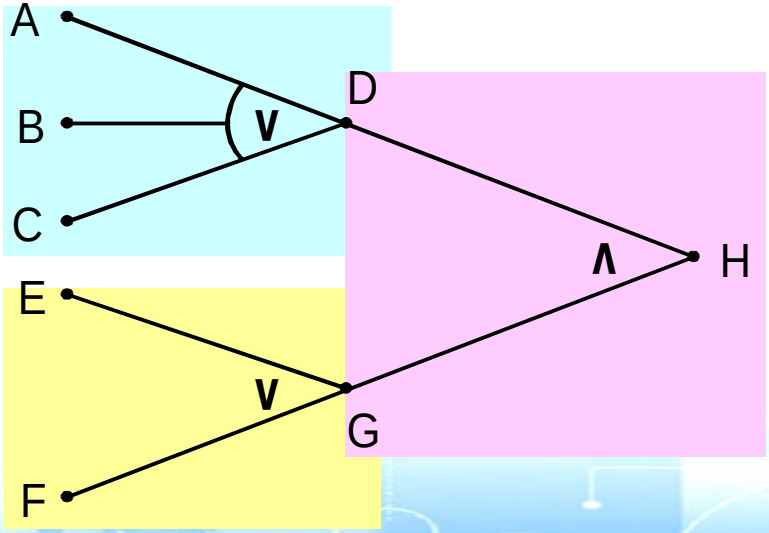
行数 = 功能变例数= 节点数

列数 = 节点数（= 输入节点数+1）

- 对于单一逻辑关系，其功能变例就是测试用例，且充分、必要
- 那么复杂的逻辑关系呢？

# 测试用例设计：所需的功能变例

- 功能变例矩阵
  - 关联矩阵
    - 将复杂的逻辑功能分解成若干个单一逻辑
    - 将每个单一逻辑的功能变例输入到同一个矩阵中
    - 得到的就是复杂逻辑的功能变例矩阵，亦称关联矩阵




关联矩阵

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| D | 1 | 0 | 0 | 1 | | | | |
| | 0 | 1 | 0 | 1 | | | | |
| | 0 | 0 | 1 | 1 | | | | |
| | 0 | 0 | 0 | 0 | | | | |
| G | | | | | 1 | 0 | 1 | |
| | | | | | 0 | 1 | 1 | |
| | | | | | 0 | 0 | 0 | |
| H | | | | 0 | | | 1 | 0 |
| | | | | 1 | | | 0 | 0 |
| | | | | 1 | | | 1 | 1 |

# 定义测试用例

- 任务1—构造关联矩阵
  - 将功能变例输入到矩阵中
  - 基于相互关系和约束判断是否有其他节点相牵连
  - 扩展补充，确保错误被传播到可观察的点上
- 任务2—组合功能变例成测试用例
  - 构造测试用例定义矩阵
  - 构造覆盖矩阵

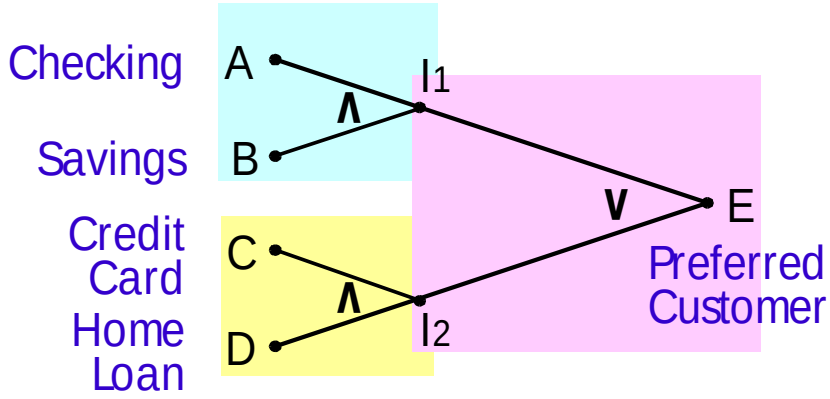
测试用例设计：

# 定义测试用例

- 构造关联矩阵
  - 一个例子

功能变例

1. If A(F), B(T) then I1(F)
2. If A(T), B(F) then I1(F)
3. If A(T), B(T) then I1(T)
4. If C(F), D(T) then I2(F)
5. If C(T), D(F) then I2(F)
6. If C(T), D(T) then I2(T)
7. If I1(T), I2(F) then E(T)
8. If I1(F), I2(T) then E(T)
9. If I1(F), I2(F) then E(F)




其中：E 可观测。 I1 和 I2 不可观测

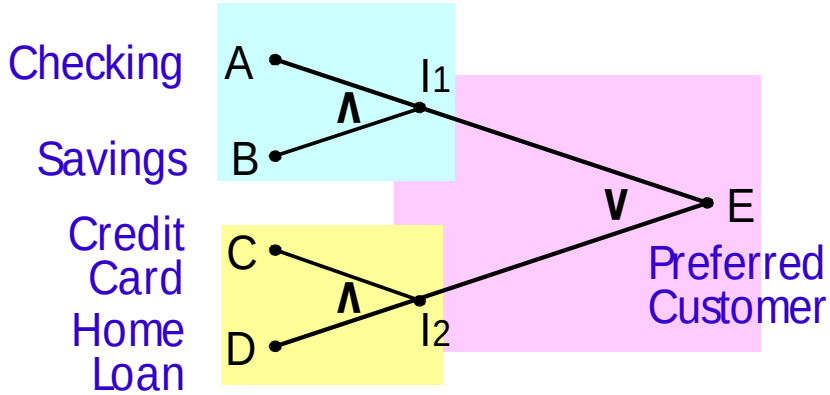
# 测试用例设计：

# 定义测试用例

## ❖ 构造关联矩阵

### ❑ 1。输入功能变例

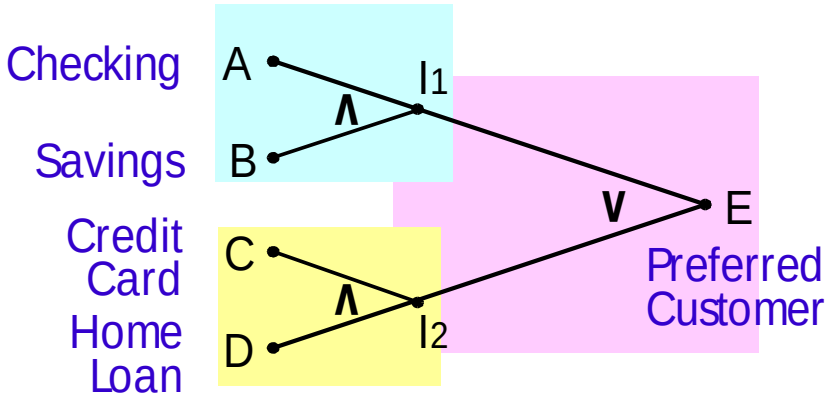
| | | | Nodes | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Variations | | | A | B | I1 | C | D | I2 | E |
| | I1 | 1 | 0 | 1 | 0 | | | | |
| | | 2 | 1 | 0 | 0 | | | | |
| | | 3 | 1 | 1 | 1 | | | | |
| | I2 | 4 | | | | 0 | 1 | 0 | |
| | | 5 | | | | 1 | 0 | 0 | |
| | | 6 | | | | 1 | 1 | 1 | |
| | E | 7 | | | 1 | | | 0 | 1 |
| | | 8 | | | 0 | | | 1 | 1 |
| | | 9 | | | 0 | | | 0 | 0 |



# 测试用例设计：

## 定义测试用例

- 构造关联矩阵
  - 2。确定隐含的关联




关联矩阵

| | | | Nodes | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | A | B | I1 | C | D | I2 | E |
| Variations | I1 | 1 | 0 | 1 | 0 | | | | |
| | | 2 | 1 | 0 | 0 | | | | |
| | | 3 | 1 | 1 | 1 | | | | 1 |
| | I2 | 4 | | | | 0 | 1 | 0 | |
| | | 5 | | | | 1 | 0 | 0 | |
| | | 6 | | | | 1 | 1 | 1 | 1 |
| | E | 7 | 1 | 1 | 1 | | | 0 | 1 |
| | | 8 | | | 0 | 1 | 1 | 1 | 1 |
| | | 9 | | | 0 | | | 0 | 0 |

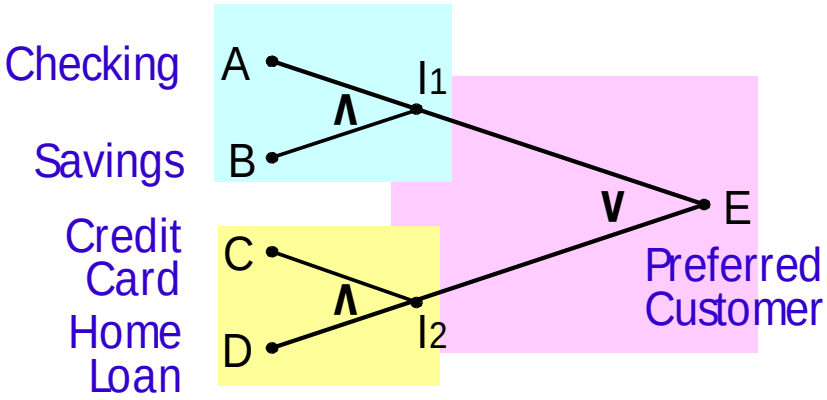
如果I1 = 1 则 E 为 1

如果I2 = 1 则 E 为 1
如果I1 = 1 则 A=1 且 B=1
如果I2 = 1 则 C=1 且 D=1

# 测试用例设计：

# 定义测试用例

- 构造关联矩阵
  - 3。补充关联




I1 不可见，所以我们要“顺流” 到下一结果，即 E。我们通过设置使流入E的其他输入为 0 (因为 E 点的逻辑操作是 “or”). 所以，为了I1 功能变例，要使 I2 为 0.

同理，为了I2 功能变例，要使 I1为 0.

| | | Nodes | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Variations | | A | B | I1 | C | D | I2 | E |
| I1 | 1 | 0 | 1 | 0 | | | 0 | 0 |
| | 2 | 1 | 0 | 0 | | | 0 | 0 |
| | 3 | 1 | 1 | 1 | | | 0 | 1 |
| I2 | 4 | | | 0 | 0 | 1 | 0 | 0 |
| | 5 | | | 0 | 1 | 0 | 0 | 0 |
| | 6 | | | 0 | 1 | 1 | 1 | 1 |
| E | 7 | 1 | 1 | 1 | | | 0 | 1 |
| | 8 | | | 0 | 1 | 1 | 1 | 1 |
| | 9 | | | 0 | | | 0 | 0 |

如果I2 = 0 则 E 为 0.
如果I2 = 0 则 E = 0.

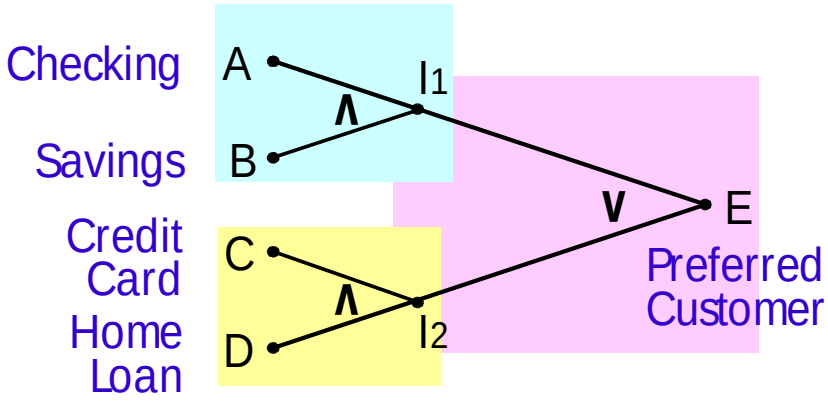
如果 I1 和 I2 都为 0 则 E 为 0.
如果 I1 和 I2 都为 0 则 E 为 0.

# 测试用例设计：

# 定义测试用例

## ❖组合功能用例

### ❑1。测试用例矩阵




关联矩阵

| | | | Nodes | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | A | B | I1 | C | D | I2 | E |
| Variations | I1 | 1 | 0 | 1 | 0 | | | 0 | 0 |
| | | 2 | 1 | 0 | 0 | | | 0 | 0 |
| | | 3 | 1 | 1 | 1 | | | 0 | 1 |
| | I2 | 4 | | | 0 | 0 | 1 | 0 | 0 |
| | | 5 | | | 0 | 1 | 0 | 0 | 0 |
| | | 6 | | | 0 | 1 | 1 | 1 | 1 |
| | E | 7 | 1 | 1 | 1 | | | 0 | 1 |
| | | 8 | | | 0 | 1 | 1 | 1 | 1 |
| | | 9 | | | 0 | | | 0 | 0 |

测试用例矩阵

| | Nodes | | | | | | |
|---|---|---|---|---|---|---|---|
| | A | B | I1 | C | D | I2 | E |
| T/C 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| T/C 2 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| T/C 3 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| T/C 4 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

# 测试用例设计：

# 定义测试用例

## ❖组合功能用例

### ❑2。测试覆盖矩阵

测试覆盖矩阵

| | Variations | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| T/C 1 | X | | | X | | | | | X |
| T/C 2 | | X | | | X | | | | X |
| T/C 3 | | | X | | | | X | | |
| T/C 4 | | | | | | X | | X | |

关联矩阵

| | Nodes | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Variations | | | A | B | I1 | C | D | I2 | E |
| | I1 | 1 | 0 | 1 | 0 | | | 0 | 0 |
| | | 2 | 1 | 0 | 0 | | | 0 | 0 |
| | | 3 | 1 | 1 | 1 | | | 0 | 1 |
| | I2 | 4 | | | 0 | 0 | 1 | 0 | 0 |
| | | 5 | | | 0 | 1 | 0 | 0 | 0 |
| | | 6 | | | 0 | 1 | 1 | 1 | 1 |
| | E | 7 | 1 | 1 | 1 | | | 0 | 1 |
| | | 8 | | | 0 | 1 | 1 | 1 | 1 |
| | | 9 | | | 0 | | | 0 | 0 |

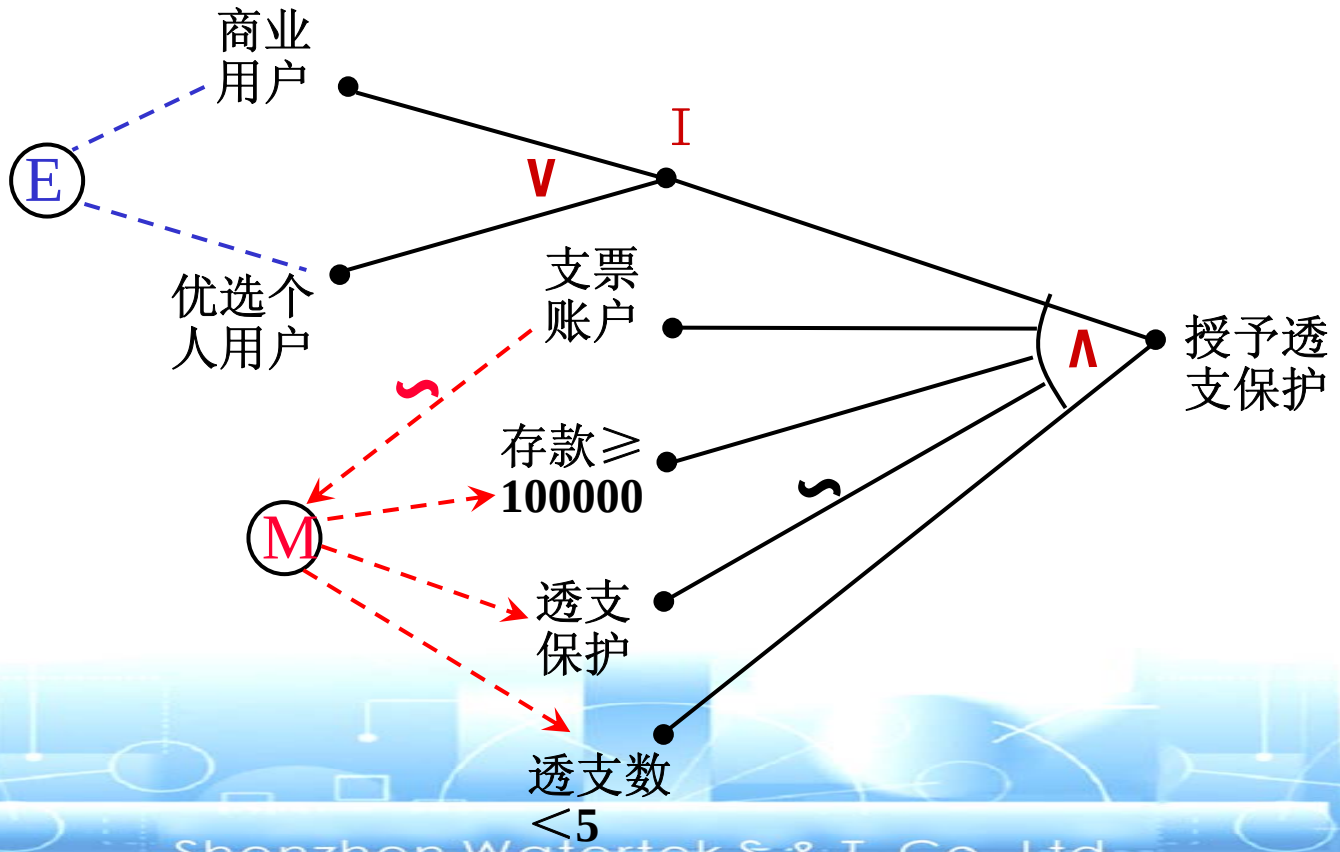
测试用例矩阵

| | Nodes | | | | | | |
|---|---|---|---|---|---|---|---|
| | A | B | I1 | C | D | I2 | E |
| T/C 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| T/C 2 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| T/C 3 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| T/C 4 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

测试用例设计：

# 定义测试用例

- 练习：透支保护授权
  - 在分析关联矩阵时，要考虑约束的影响



- 小结：从因果图到测试用例
  - 将因果图分解为若干个子逻辑图，每个子逻辑只有单一的逻辑关系，如And、Or、Xor、Nand、Nor、Nxor
  - 将每个子逻辑所需的功能变例输入到关联矩阵，得到因果图的功能变例矩阵
  - 确定隐含的关联性（输入/输出），并填入关联矩阵。注意考虑环境约束的影响
  - 在关联矩阵中补充必要的条件和结果，注意应将不可观察的节点“传播”到输出端
  - 组合功能变例，得到测试用例矩阵
  - 确认测试覆盖。一个测试用例覆盖一个或多个功能变例，全部测试用例要覆盖关联矩阵中所有的功能用例

最大的覆盖，最少的测试用例

## 本篇小结

- 测试从需求开始
  - 需求不仅是设计的基础，也是测试的前提
- 需求的二义性审查
  - 正确
  - 完整
  - 无二义性
  - 逻辑一致
- 创建因果图
  - 定义节点
  - 逻辑关系
  - 添加约束
- 设计测试用例
  - 因果图--〉分解---〉关联矩阵---〉确认补充---〉组合

# 谢谢大家