# 一．UML

## 1. Use case diagram

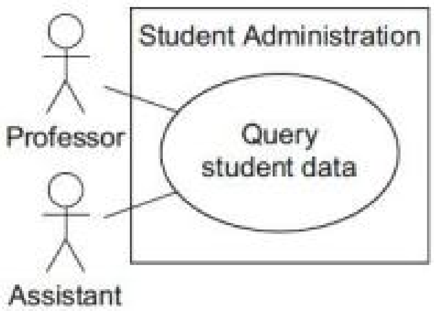
### (1)Actors（人）

- Don’t confuse with stakeholder 不要与涉众（相关者）混淆
- Not all stakeholders are actors in certain use cases 并非所有利益相关者都是特定用途中的参与者
- Not necessarily human. i.e. server 并非都是人
- Active actor: who initiates the use case 谁开始了这个 use case，所以带有方向，当一件事与多个 actor 相连时便存在这种复杂关系，参考 hw1！！！！
- Primary/secondary actor:
  - Who benefits from executing the use case　　谁从执行这件事种收益
- Represents roles, not user
  - One user can perform different role, therefore represents multiple actors 一个用户可以分饰多个角色，因此可以代表多个 actors

### (2)Use cases（事）

- 主要展示客户对于系统的基本功能需求，谁与系统有交互联系，怎么使用系统
- V + N eg. Check deposit
- Use case 需要给 actor 带来好处，i.e. Withdraw money is a use case, fill in the withdraw form 并不是

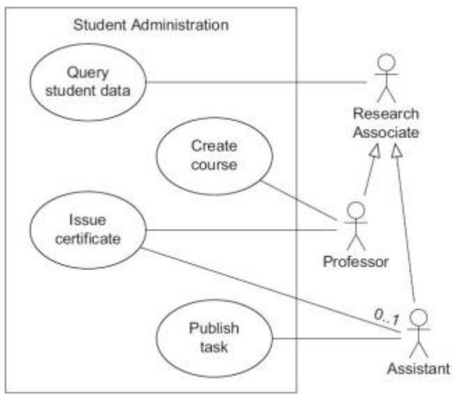
### (3)Association（人与事间的联系）




- 每个 actor 至少与一件事有关
- 每个事至少与一个 actor 相关!!!!
- actors 应该位于系统之外
- 一些事件相关者可以在系统之内
  eg. Business workers

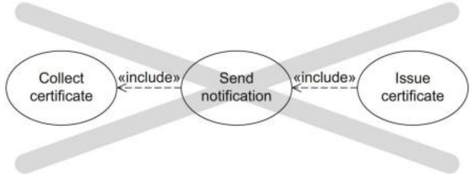
### (4)更复杂的关系

- Generalization/inheritance(泛化)
  - Abstract actor 用一个抽象的 actor 来泛化其他具有相同 association 的 actor，即将他们的相同 use case 分离出来成一个泛化 actor
    - No instance
- Extend eg. A extend B 那么 B 可独立存在，可单独干，即干 A 一定干 B，但干 B 不一定干 A
- Include eg. A include B 那么 B 不可独立存在，只有干 A 要干 B，不能光干 B

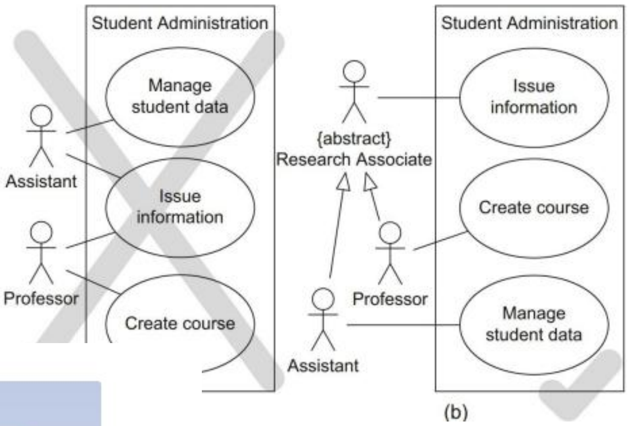


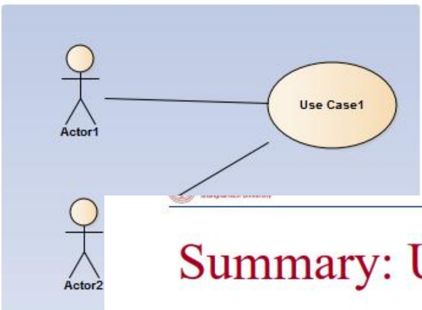
## (5)常见问题

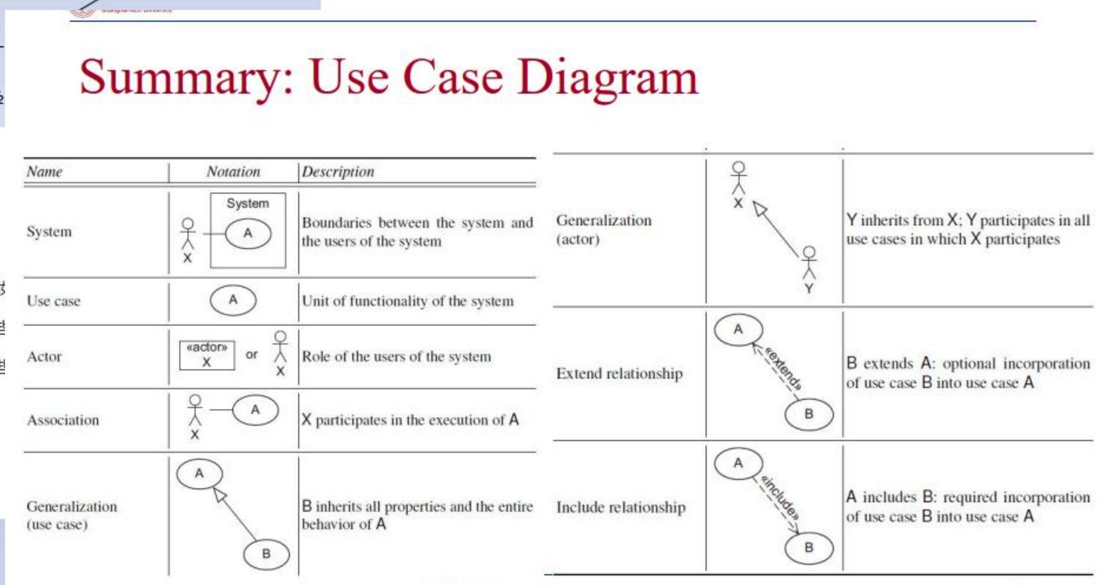
- Modeling Processe，将一些过程错误的写成一个流程




- Setting wrong system boundary，错误的将 actor 放入系统中
- Functional decomposition 将一个过程拆分成一些具体的子过程
- Incorrect association,没有用抽象类进行总结概括，然后就错误了导致了不正确的关系，即助手和教授并非通过该 use case 有关系，而是二者都能干这件事，因此需要引入抽象类



# Summary: Use Case Diagram

| Name | Notation | Description |
|---|---|---|
| System | | Boundaries between the system and the users of the system |
| Use case | | Unit of functionality of the system |
| Actor | | Role of the users of the system |
| Association | | X participates in the execution of A |
| Generalization (use case) | | B inherits all properties and the entire behavior of A |
| Generalization (actor) | | Y inherits from X; Y participates in all use cases in which X participates |
| Extend relationship | | B extends A: optional incorporation of use case B into use case A |
| Include relationship | | A includes B: required incorporation of use case B into use case A |

这是否意味着：

a) Actor1和Actor2可以使用use Case1

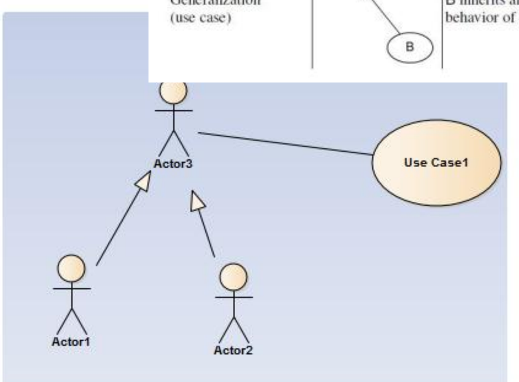
b)同时需要Actor1和Actor2才能开始使用Case1 (例

c) Actor1可以开始使用Case1，Actor2稍后再做一些

d) Actor2可以开始使用Case1，Actor1稍后再做一些

答案B是正确的，我是对的吗？

答案是：



## 2. Class case diagram

### (1) Class and object

- class 是面向对象方法的基本组成
    - 由一系列的 Attributes 和 Operations 组成
        - 可视性（Visibility）
            - + global: accessible to all
            - - private: accessible within the object
            - # protected: only accessible by its sub-classes
- object 是 class 的实例

### (2) Attributes

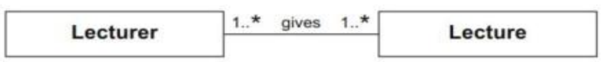
- Name
    - Noun clause, lowercase first letter, then uppercase for latter words
    - i.e. gradStudent, firstName
- Data Type
    - i.e. String
- Multiplicity: how many value it can contain
    - [min .. max]: i.e. [0 .. 1]
    - Example: address: String [1..*]= “Huanke Rd 199”

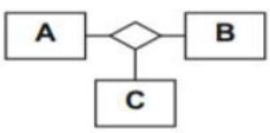
### (3) Operations

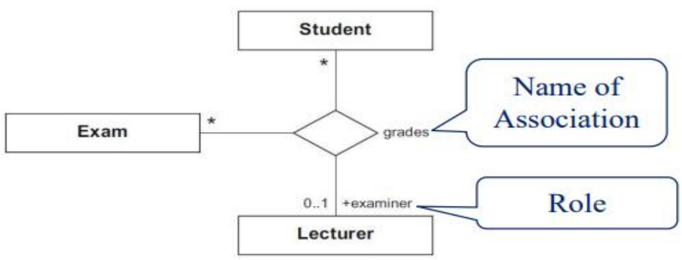
- Name
    - Verb clause: i.e. getGrade()
- Parameters
    - Direction: in, out, inout
    - Name
    - Data type
- Return value
    - Only need a data type
- Example
    - getName(out fn: String, out in: String): void
    - updateLastName(in newName: String): boolean

### (4) Association(描述类间关系)

- Multiplicity (Cardinality)
    - The number of objects that may be associated with exactly one object of the opposite side 可以与连线对面，一个 object 联系的数量
    - eg. A2--3B 意味着一个 B 有 2 个 A，一个 A 有 3 个 B
- 二元关系\多元关系

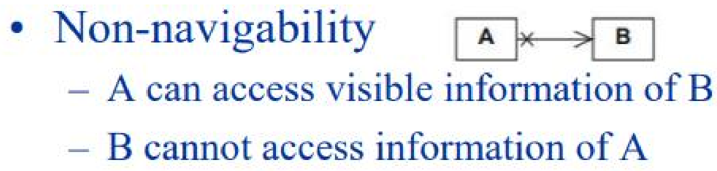



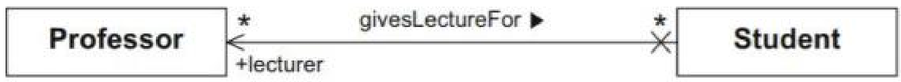

ality)

(5) 复杂类间关系

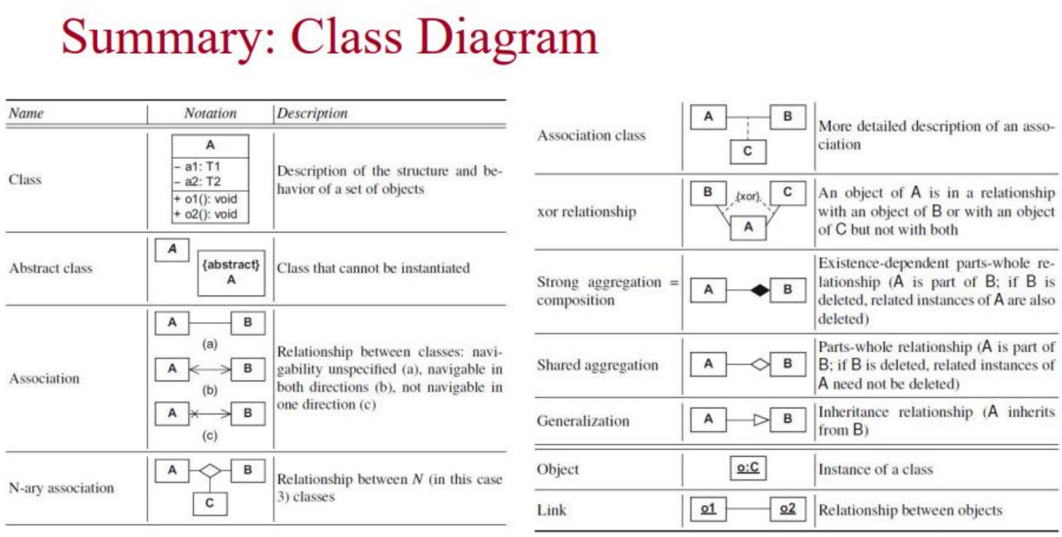
- Navigability

  - By default the information sharing is bi-directional 默认情况下双向信息共享




- Association class 同时具有连接和类的性质
- shared Aggregation A 是 B 的一部分，但没了 B，A 还可以存在 eg.学生在 lab
- （strong）Composition（约束更强） A 是 B 物理上的一部分，并且在某一特定时间，一个特定 partA 最多只能包含在一个复合对象中 eg. Lab 在 building 中，一般一个 building 可以有很多 lab，但一个 lab 只在一个 building 内，因此箭头处一般为 1！！
- Generalization/Inheritance 抽象类的概括，Highlight common attributes and methods of objects and classe，并且一个类可以从多个类进行继承

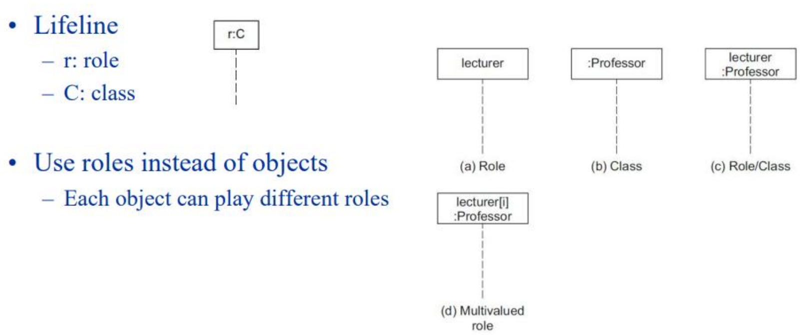


Summary: Class Diagram

| Name | Notation | Description |
|---|---|---|
| Class | A; - a1: T1; - a2: T2; + o1(): void; + o2(): void | Description of the structure and behavior of a set of objects |
| Abstract class | *A*; {abstract} A | Class that cannot be instantiated |
| Association | A — B (a); A ⟷ B (b); A ×→ B (c) | Relationship between classes: navigability unspecified (a), navigable in both directions (b), not navigable in one direction (c) |
| N-ary association | A, B, C | Relationship between *N* (in this case 3) classes |
| Association class | A, B, C | More detailed description of an association |
| xor relationship | B, {xor}, C, A | An object of A is in a relationship with an object of B or with an object of C but not with both |
| Strong aggregation = composition | A, B | Existence-dependent parts-whole relationship (A is part of B; if B is deleted, related instances of A are also deleted) |
| Shared aggregation | A, B | Parts-whole relationship (A is part of B; if B is deleted, related instances of A need not be deleted) |
| Generalization | A, B | Inheritance relationship (A inherits from B) |
| Object | o:C | Instance of a class |
| Link | o1, o2 | Relationship between objects |

## 3. Sequence diagram(Interaction partners 间的消息传递)

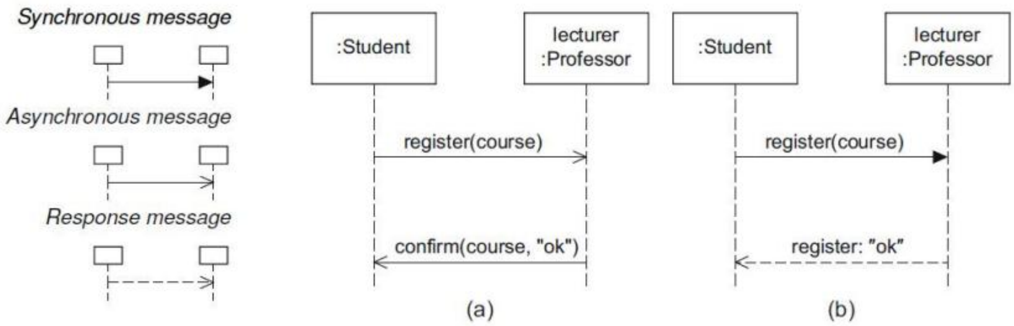
(1) Interaction Partner



(2) Message Exchange

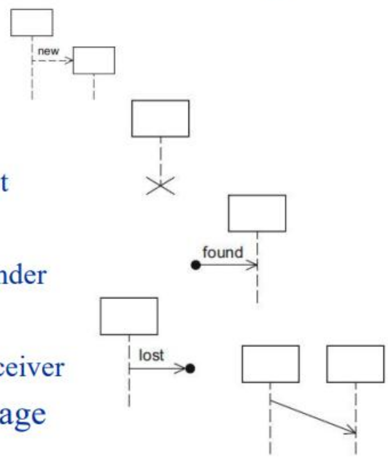
- 时间和交流对象两个维度
- Execution specification - Self message（执行占用的时间）
- 顺序问题:若信息位于相同 lifeline 上,那么我们可以便可知道其顺序(若顺序问题落在多条 lifeline 上，需引入并发来辅助判断)
- 类型问题：同步（线下面交）和异步（email 你发了但对方不一定现在就看到）。 一般同步与恢复信息同时使用，异步信息自己使用

- (a) Register a course via email
- (b) Register a course in person



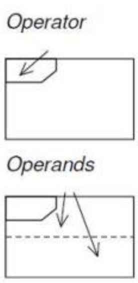

- 特殊类型：

- Create message
  - Creating new object
- Destruction event
  - Destruction of an object
- Found message
  - Unknown/irrelevant sender
- Lost message
  - Unknown/irrelevant receiver
- Time-consuming message




-Combined Fragment：

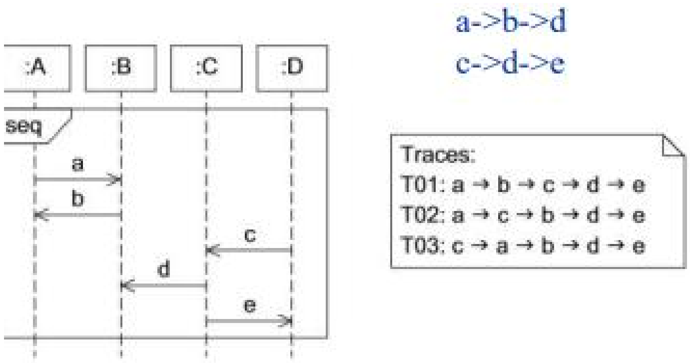
- Each operand has a guard

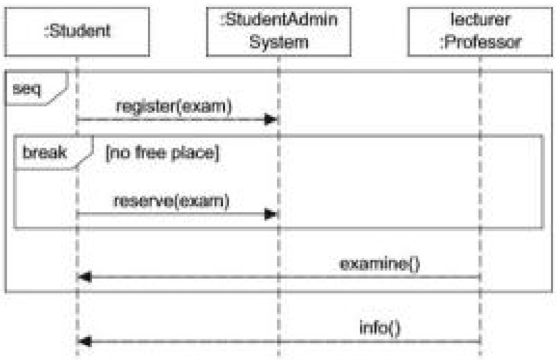



| | Operator | Purpose |
|---|---|---|
| **Branches and loops** | alt | Alternative interaction |
| | opt | Optional interaction |
| | loop | Iterative interaction |
| | break | Exception interaction |
| **Concurrency and order** | seq | Weak order |
| | strict | Strict order |
| | par | Concurrent interaction |
| | critical | Atomic interaction |

- concurrency and order

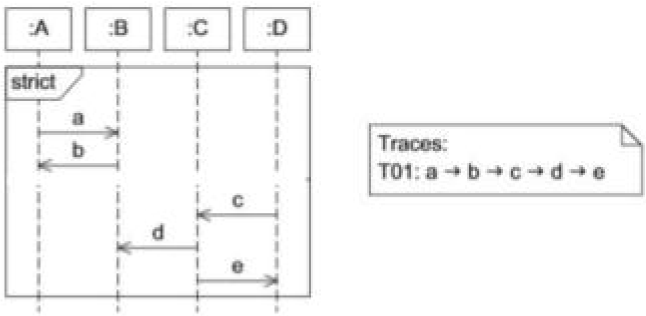
- seq 弱 order， 维护最开始的 rule 即相同 lifeline 上顺序执行，因此存在多种情况

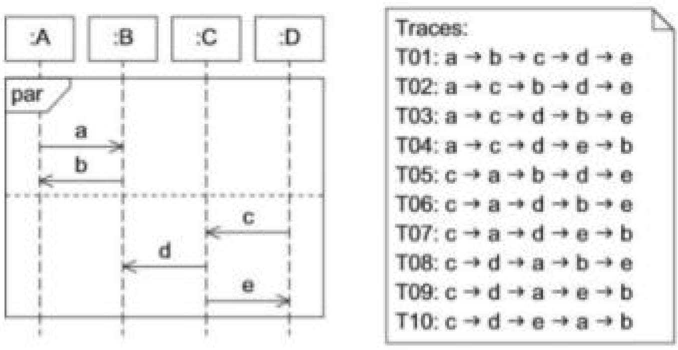


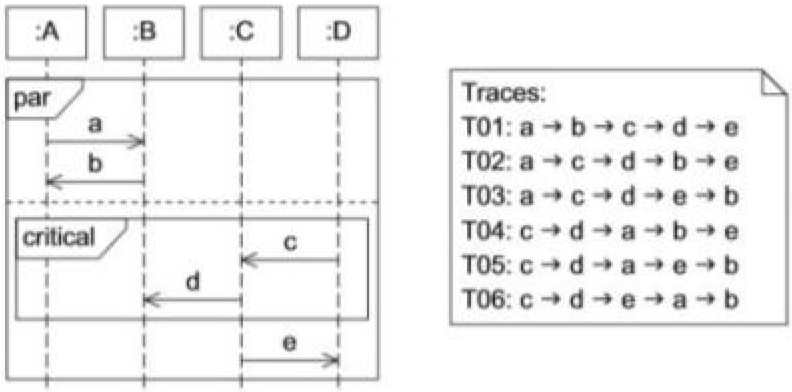

- strict 强 order， 视所有 lifeline 上顺序执行，因此只有一种情况

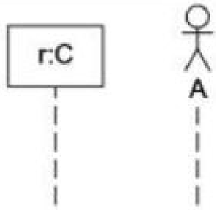
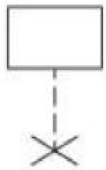
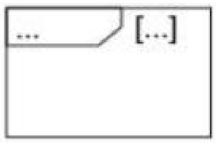
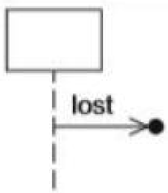
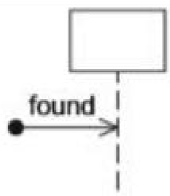
- par 分区内的顺序保留，分区间的顺序无关紧要



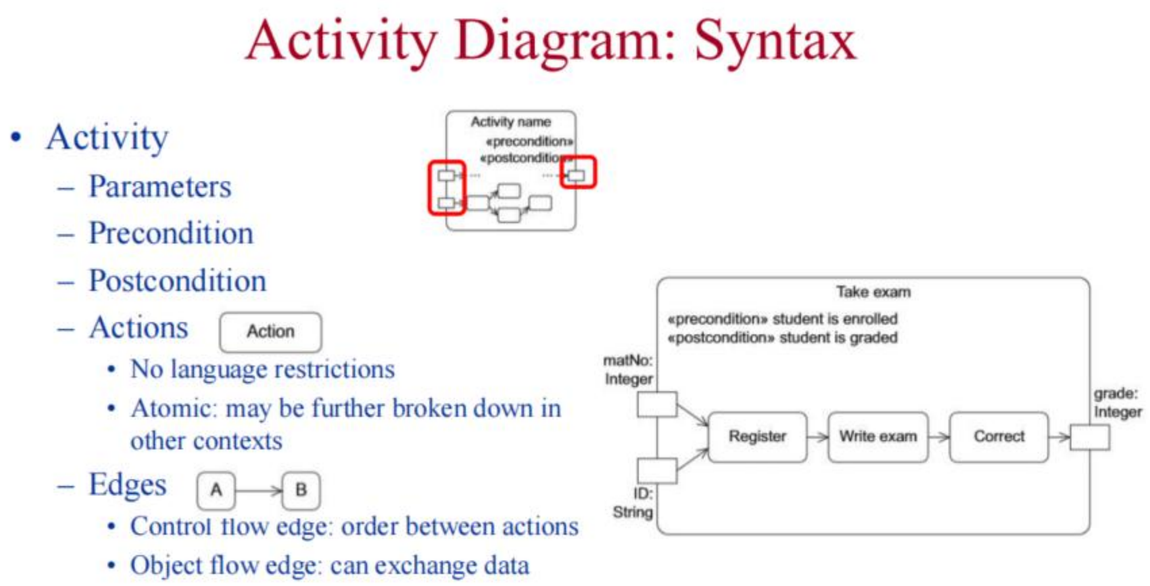




- Critical 原子操作，分区内禁止其他的操作且有顺序，分区外无所谓




| *Name* | *Notation* | *Description* |
|---|---|---|
| Lifeline |   | Interaction partners involved in the communication |
| Destruction event |  | Time at which an interaction partner ceases to exist |
| Combined fragment |   | Control constructs |
| Synchronous message |  | Sender waits for a response message |
| Response message |  | Response to a synchronous message |
| Asynchronous message |  | Sender continues its own work after sending the asynchronous message |
| Lost message |   | Message to an unknown receiver |
| Found message |   | Message from an unknown sender |

4. Activity diagram




| *Name* | *Notation* | *Description* |
|---|---|---|
| Action node | Action | Actions are atomic, i.e be broken down further |
| Activity node | Activity | Activities can be broker |
| Initial node | | Start of the execution o |
| Activity final node | | End of ALL execution tivity |
| Flow final node | | End of ONE execution tivity |
| Decision node | | Splitting of one execu two or more alternat paths |
| Merge node | | Merging of two or mo execution paths into path |
| Parallelization node | | Splitting of one execu two or more concurr paths |
| Synchronization node | | Merging of two or mo execution paths into path |
| Edge | A → B | Connection between th activity |
| Call behavior action | A | Action A refers to an same name |
| Object node | Object | Contains data and objec ated, changed, and read |
| Parameters for activities | Activity | Contain data and object output parameters |
| Parameters for actions (pins) | Action | Contain data and object output parameters |

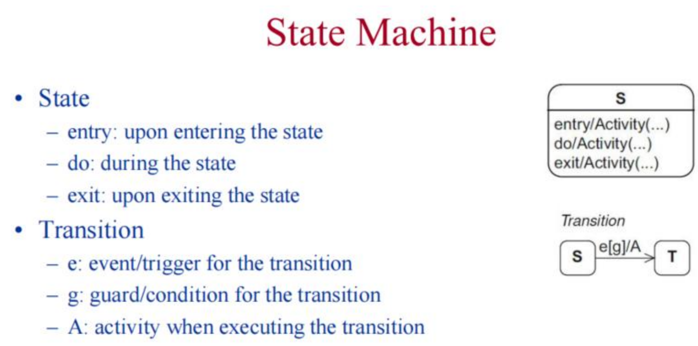
| *Name* | *Notation* | *Description* |
|---|---|---|
| Partition | A B A B | Grouping of nodes and edges within an activity |
| Send signal action | S | Transmission of a signal to a receiver |
| Asynchronous accept (time) event action | E or T | Wait for an event E or a time event T |
| Exception handler | e Exception-Handler Action | Exception handler is executed instead of the action in the event of an error e |
| Interruptible activity region | B E A | Flow continues on a different path if event E is detected |

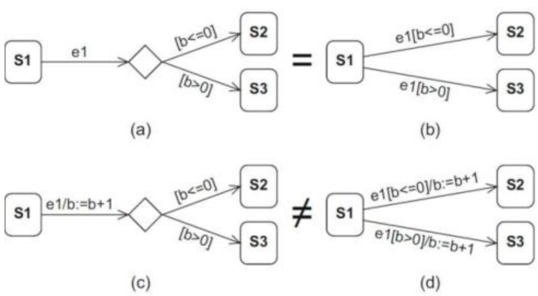
- Control flow 控制流就是对活动和对象之间的关系的描述。详细的说控制流表示动作与其参与者和后继动作之间以及动作和其输入和输出对象之间的关系。而对象流就是一种特殊的控制流。

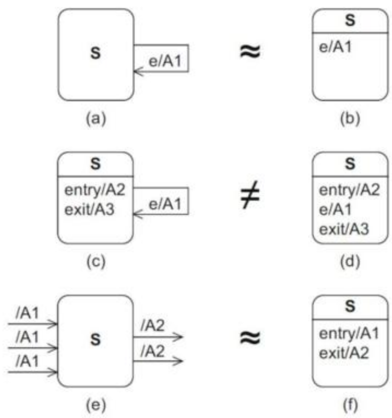
- Token
    - 多条输入边，每条都要有 token，多条输出边，token 分给所有的边
- Edge
    - Guard 条件
    - Weight 这条边消耗的 token 数量
- Activity final node 当到达时，其他并行的流也会被终止，注意和 flow final node(并发的活动结束，只终止其所在的流)的区别
- Connector 小圆圈（作用与换行时的转折线一样）

- Object flow(Object 进入 activity 经历的状态变化)是将对象流状态作为输入或输出的控制流。在活动图中，对象流描述了动作状态或者活动状态与对象之间的关系，表示了动作使用对象以及动作对对象的影响。
    - Central buffer 数据消失，便不在
    - Data store 数据消失，其中仍有备份
- Swimlane/partition 分区
- Interrupt handler

## 5. State Machine diagram




对于右二不相等的原因在于，这条线进出了 State S，因此同时会触发 entry 和 exit
而左二不相等的原因在于 transition 的 activity 在变换后的先后顺序发生变化
- Composite states 嵌套状态，可以组合成大状态并且引入 entry 和 exit points
- Orthogonal State 正交状态（并行）
- History State
    - H Restart 从同级的大状态重启 activity

- H* Continue 从当前的 activity 进行同步

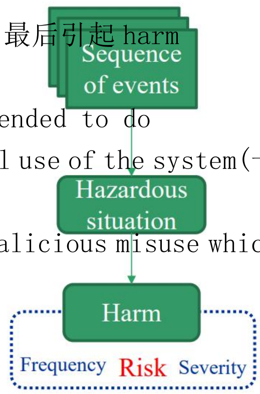
| *Name* | *Notation* | *Description* |
| --- | --- | --- |
| State | S<br>entry/Activity(...)<br>do/Activity(...)<br>exit/Activity(...) | Description of a specific "time span" in which an object finds itself during its "life cycle". Within a state, activities can be executed on the object. |
| Transition | S $\xrightarrow{e}$ T | State transition e from a source state S to a target state T |
| Initial state | ● | Start of a state machine diagram |
| Final state | ◉ | End of a state machine diagram |
| Terminate node | × | Termination of an object's state machine diagram |
| Decision node | | Node from which multiple alternative transitions can proceed |
| Parallelization node | | Splitting of a transition into multiple parallel transitions |
| Synchronization node | | Merging of multiple parallel transitions into one transition |
| Shallow and deep history state | H / H* | "Return address" to a substate or a nested substate of a composite state |

## 二. Risk Management 风险管理

### (1) 术语

- **Harm/Impact:** Loss when running the system in certain situations
  - Financial/time loss
  - Life/health loss
- Hazard: A potential source of harm
- Hazardous situation: circumstance in which people or property are exposed to one or more hazard(s)
- Risk: Combination of the probability of occurrence of harm and the severity of that harm

某一系统存在 hazard，而一系列事件可能会导致 hazard 情况，最后引起 harm




### (2) 类型

- Efficacy risk: The system fails to do what it was intended to do
- Safety risk: People & properties may be harmed under normal use of the system(一般情况下出错)
- Security risk: The system is prone to unintentional and malicious misuse which

can cause harm（非一般情况下出错）

(3) Risk Management process

- Risk analysis – Analyze the frequency and severity of harms

Eg. 比如计算一系列事件最终会引起某一 harm 的概率

- Risk evaluation – Determine what is acceptable risk

针对不同程度的 risk 我们可以采取不同的 actions

- Risk control – How to prevent hazard and/or reduce harm

降低发生的概率或降低严重程度两种

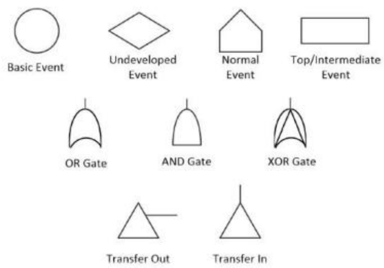
- Residual risk acceptability 剩余风险可接受性 – Are there new risks introduced by control measures 采用措施所带来的新风险

(4) FTA(Fault Tree Analysis)

- 从上到下的推理演绎结果，从不希望的系统结果出发找到可能导致其发生的一系列事件

## Fault Tree Analysis Symbols

- Basic Event:
  - Requiring no further development
- Undeveloped Event
  - An event that is not further developed due to lack of information, or when the consequences are not important
- Normal Event
  - An event that is normally expected to occur, e.g., the device gets used
- Top/Intermediate Event
  - An event that is further analyzed
- OR Gate
  - Output occurs when one or more of the inputs occur
- AND Gate
  - Output occurs when all of the inputs occur
- XOR Gate
  - Output occurs when only one of the inputs occurs
- Transfer
  - Used to manage the size of the tree on a page, and to avoid duplications

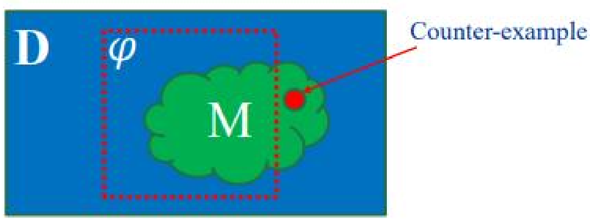



- 最小割集 当他们全发生时将会导致顶端事件的发生
- 与门相乘概率；或门相加概率

三．UPPAAL

(1) Model Checking（探索所有可到达的状态去 check 不变量,穷举遍历）

- A domain D representing the state space of a model
- The reachable state space M for the model
- Define a subset of the state space as property $\varphi$
- Explore the whole reachable state space of a model for property violations



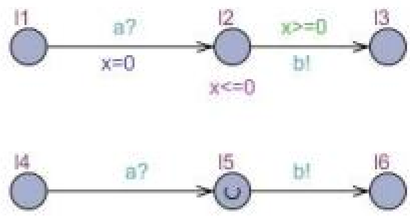
- Problem: 状态空间爆炸 eg. 实数空间
    - Simple yet expressive formalism:
        - 时间状态机(同样会有爆炸的转移系统 transition system，uppaal 的基础)
    - Symbolic states/executions:
        - 有限划分：将状态空间划分为有限多个等价类，使等价的状态表现出相似的行为，例如用坐标系中的空间表示 or 图像化
    - Model abstraction/approximation:
        - Existential Abstraction(Over-approximation)
        - with refinement，将 deadend 和 bad states 分离进不同的抽象空间
- Uppaal

## UPPAAL Syntax

- Global declarations
    - Clocks:
        - clock x1,…xn;
    - Data variables
        - int n1,…; integer with default domain
        - Int[l,u] n1,… ; integer with domain defined by [l,u]
        - Int n1[m],…; array with elements n1[0] to n1[m-1]
    - Channels
        - Chan a, … ;
        - Urgent chan b … ;
        - Broadcast chan c … ;
    - Constants
        - Const int c1=n1;

- channel 的声明要在前面加上 &
    - Urgent channel: 立即发送同步信息，并且不允许有转移条件即 guard 存在
    - Broadcast channel 同步多个进程；若接收到 channel a?, 若可以，立即进行 transition；可以在 receiver 都没准备好时就可以发送信息
- template 相当于类，可以做实例化
- System Declarations 系统声明中可以声明实例
- locaion 相当于状态，内部可设不变量，即在该 location 时需要满足的条件
    - Initial state O
    - Urgent state U: 立即！！！但允许 interleaving
    上图其实可以视为用 clock 模拟的一个立即效果，即 t = 0 就要进行转移

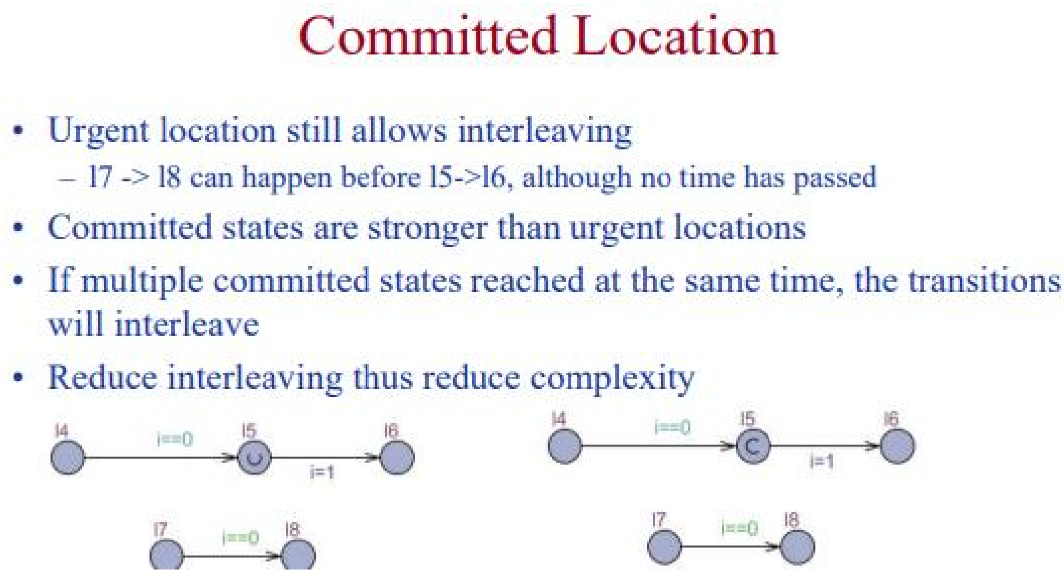
- No time pass in an urgent location
- The two automata is equivalent
- Save a clock thus reduce state space



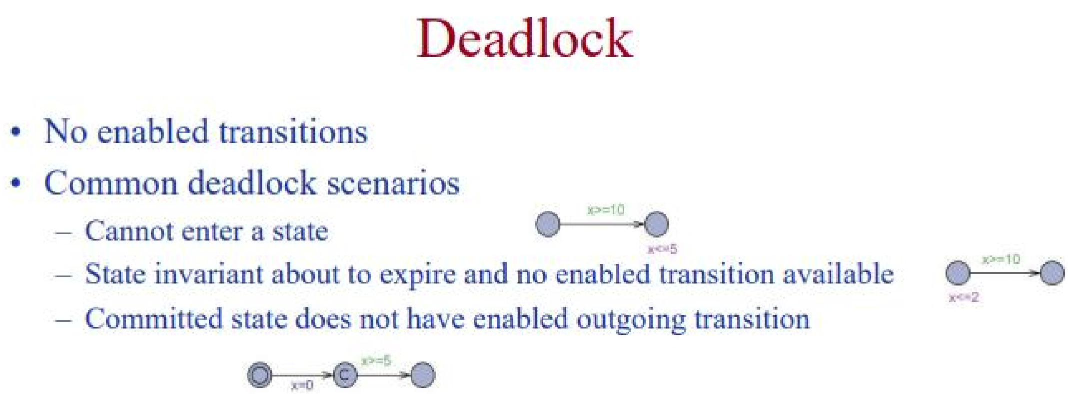
- Committed state C: 约束比 urgent 更强，但当多个 C 存在时，也会有 interleaving

Interleaving 越少， state space 的复杂度越低

比如左一图中 l5 和 l7 可能出现交错，但改为 Committed 时，则优先 l5 执行




- Edge
  - Select
  - Guard 转移发生所需要的条件
  - Sync 消息的同步 ！发消息 ？收消息
  - Update 在状态转移时所采取的 action
- Deadlock
  - 无 enabled transition
  - eg. 当 Location 没有不变量时，就处在了 Non-determinism 的状态，因为此时该 location 可以 stay forever, 因此就可以 stay 时间过长，进而不满足了 transition 出去的 guard。亦或是，可以 transition，但是不满足下一个 location 的不变量，因此无法进行 transition。这是两种可能导致 deadlock 的具体情况。




- TCTL（Temporal Computational Tree Logic）
  - A[]p “Always globally p”所有 path 上所有 state 都是 p
  - A<>p “Always eventually p”所有 path 上至少有一个 p
  - E[]p “Exist globally p” 至少一个 path 全是 p
  - E<>p “Exist eventually p”至少一个 path 有至少一个 p

- p-->p imply

四．Testing -- test design

(1) Specification-based testing（黑盒）

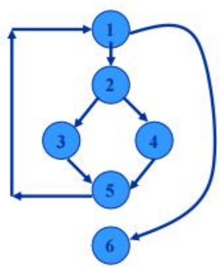
- 等价划分 equivalent partitioning
  - S1: Identify feature sets
  - S2: Derive Test Conditions
    - TCOND : Input/output valid/invalid
  - S3: Derive Test Coverage items
    -TCOVER: 对每个测试条件指定一个测试覆盖项
  - S4: Derive Test cases
    - One to one
    - Minimized
- State Transition testing 状态转移测试（FSM）
  - Single transition(0-switch coverage): 只有 valid transitions
  - All transitions: 全部的 transitions valid 和 invalid 都包括
  - Multiple transitions(N-switch coverage): valid sequences of N+1 transitions in the state model

(2) Structure-based testing(白盒)

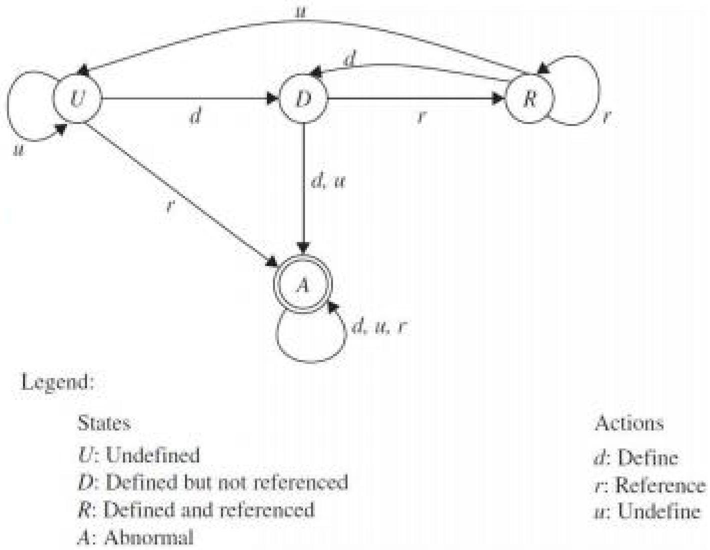
- statement coverage: 所有语句至少被执行一遍
- branch coverage: 执行过所有的语句分支，即含有表达式的每一个真值
- condition coverage: 含有复合表达式的每一个组成的真值
  - Consider the conditional expression
    - ((c1.and.c2).or.c3):
  - Branch coverage
    - ((c1.and.c2).or.c3)==true
    - ((c1.and.c2).or.c3)==false
  - Condition coverage
    - Each of c1, c2, and c3 is evaluated to true and false
- path coverage: 所有独立的线性路径（CFG, control flow graph）被执行至少一次(branch 的组合)
  - CFG
    - Sequence 顺序
    - Selection if else
    - Iteration while

## Derivation of Test Cases

- Number of independent paths: 3
  - 1,6 test case (x=1, y=1)
  - 1,2,3,5,1,6 test case(x=1, y=2)
  - 1,2,4,5,1,6 test case(x=2, y=1)



- Data flow
  - 异常类型
    - type1: 重定义
    - type2: 未定义使用
    - type3: 定义未使用

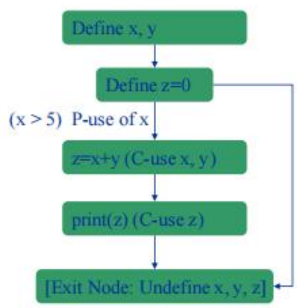



- 术语
  - Definition 当值和位置未绑定时
  - Undefinition or Kill 当值和位置未绑定时
  - Use 当从变量的内存位置获取值时
    - computation C-use 直接影响正在执行的计算
    - Predicate P-use 谓词使用 在控制执行流的谓词中使用变量

```
1: int x = 10;      // Definition of x
2: int y = 20;      // Definition of y
3: int z = 0;       // Definition of z
4: if (x > 5)       // P-use of x
5:     z = x + y;   // C-use of x and y, definition of z
6: print(z);
```

- Data flow diagram
  - 定义和c-use序列与图的每个节点相关联
  - 一组p-use与图的每条边相关联
  - 入口节点具有子程序中出现的每个参数和每个非局部变量的定义。
- 退出节点对每个局部变量都有一个未定义

```
1: int x;          // Definition of x
2: int y;          // Definition of y
3: int z = 0;      // Definition of z
4: if (x > 5)      // P-use of x
5:     z = x + y;  // C-use of x and y, definition of z
6: print(z);
```



- Data flow testing criteria

- All-Def-Criterion: Every variable definition must be reached from some test path. 到达所有变量的定义

- All-C-Use Criterion: Every computational use of a variable is executed at least once. 所有变量的 c-use 被执行至少一次

- All-P-Use Criterion: Every predicate use of a variable is executed at least once. 所有变量的 p-use 被执行至少一次

- All-C-Use/P-Use Criterion: Every possible combination of computational and predicate uses covered by paths.路径涵盖了计算和谓词使用的所有可能组合。