2019 级本科

软件工程

Software Engineering

张昕

zhangxin@cust.edu.cn

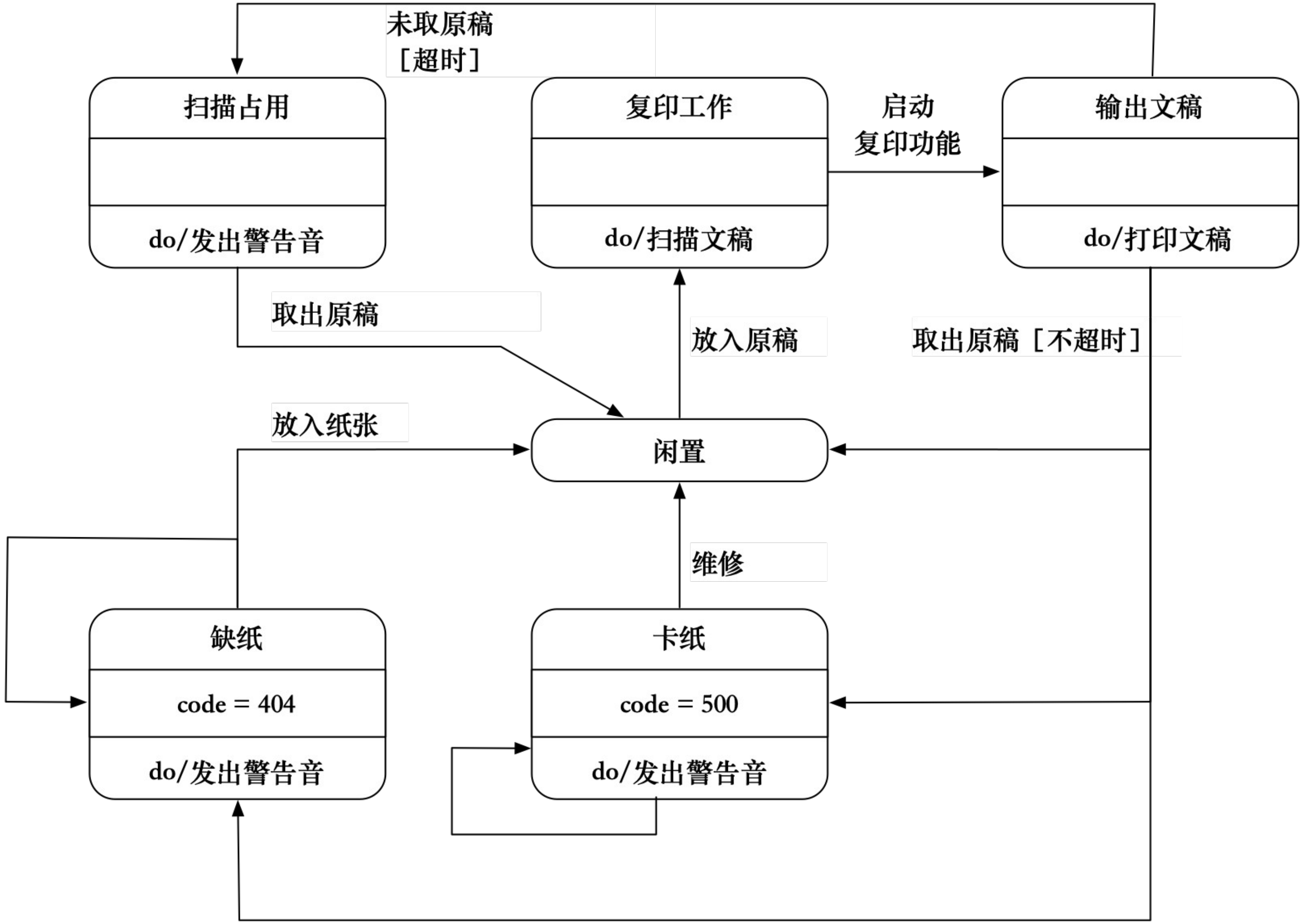
计算机科学技术学院软件工程系

Formal Specification technique

Question

- The working process of the copier is roughly as follows
  - Idle when not receiving a copy command
  - Once receiving a copy order, it will enter the copying state, and return to the idle state after completing the work specified by a copying order, waiting for the next copying order
  - If there is no paper when the copy command is executed, it will enter the out of paper state, issue a warning, wait for paper to be loaded, and enter the idle state after it is full of paper, ready to receive the copy command
  - If a paper jam occurs during copying, it will enter a paper jam state, and a warning will be issued to wait for the maintenance personnel to troubleshoot the malfunction. After the trouble is eliminated, it will return to the idle state.
- Please use the state transition diagram to describe the behavior of the copier

Question


未取原稿
［超时］
扫描占用
do/发出警告音
复印工作
do/扫描文稿
启动
复印功能
输出文稿
do/打印文稿
取出原稿
放入原稿
取出原稿［不超时］
放入纸张
闲置
维修
缺纸
code = 404
do/发出警告音
卡纸
code = 500
do/发出警告音

Problems with informal methods

- System specifications written in natural language, there may be
  - Contradiction, ambiguity, ambiguity, incompleteness and confusion of abstraction levels
    - The so-called contradiction refers to a set of conflicting statements
    - Ambiguity refers to statements that readers can understand in different ways
- System specifications are very large documents
  - Ambiguity is almost inevitable
  - Incompleteness may be one of the most frequently encountered problems in system specifications
  - Abstract level confusion means that some low-level statements about details are mixed into very abstract statements.
    - Make it difficult for readers to understand the overall functional structure of the system

Advantages of formal methods

- When people understand specifications described in natural language, they are prone to ambiguity
- In order to overcome the shortcomings of informal methods, people introduced mathematics into the software development process and created formal methods based on mathematics
- Advantage
    - Apply mathematics in the process of developing large-scale software systems, and be able to concisely and accurately describe the results of physical phenomena, objects or actions
        - So mathematics is an ideal modeling tool
        - Mathematics is particularly suitable for expressing state, which means "what to do"
    - Can smoothly transition between different software engineering activities
        - Not only functional specifications, but also system design can also be expressed in mathematics, when the program code is also a mathematical symbol
            - Although the program code is a rather tedious and lengthy mathematical symbol
    - Mathematics provides a means of high-level confirmation
        - You can use mathematical methods to prove that the design meets the specifications and the program code correctly achieves the design results

Advantages of formal methods

- The requirements specification mainly describes the state of the application system before and after operation
  - Therefore, mathematics is better than natural language to describe detailed requirements
- Under ideal circumstances, the analyst can write the mathematical specification of the system, which is accurate to almost no ambiguity, and can be verified by mathematical methods to find the existence of contradictions and incompleteness. There is no ambiguity in the specification. sex
  - But in reality, it is impossible to try to describe a complex software system with a few mathematical formulas.
  - Even if formal methods are applied, it is difficult to guarantee completeness
    - Due to insufficient communication, some customer needs may have been missed
    - The writer of the specification may intentionally omit certain features of the system, so that the designer has a certain degree of freedom when choosing an implementation method
    - It is usually impossible to imagine every possible scenario using a large and complex system

Guidelines for applying formal methods

- People's views on formal methods are not consistent
    - Formal methods are attractive to some software engineers
        - Its proponents even claim that this method can lead to a revolution in software development methods
    - Others are skeptical or even opposed to the introduction of mathematics into the software development process
- The formal method should also be "divided into two"
    - Don’t over-exaggerate its advantages or reject it altogether.

Guidelines for applying formal methods

- Appropriate representation method should be used
  - A specification technology can only describe a certain type of concept in a natural way. If this technology is used to describe a concept that is not suitable for description, not only the workload is large, but the description method is also very complicated.
- Should be formalized, but not over formalized
  - The current formal technology is not suitable for describing every aspect of the system
  - Formal specification technology requires us to describe things very accurately, so it helps prevent ambiguity and misunderstandings
  - If a formal method is used to carefully explain the error-prone or critical parts of the system, then only a moderate amount of work will be able to get a large return
- The cost should be estimated
  - In order to use formal methods, a lot of training is usually required in advance
  - It is best to estimate the required cost in advance and budget it
- There should be a formal method consultant to provide consultation at any time
  - Most software engineers are not very familiar with the mathematics and logic used in formal methods, and have not received professional training in the use of formal methods, and need expert guidance and training

Guidelines for applying formal methods [cont.]

- Shouldn't give up traditional development methods
    - It is possible to integrate formal methods with structured methods or object-oriented methods, and learning from each other can often achieve good results
- Detailed documentation should be established
    - It is recommended to use natural language to annotate the formal specifications to help users and maintainers understand the system
- Quality standards should not be abandoned
    - Formal methods do not guarantee the correctness of the software, they are just a means to help develop high-quality software
    - In addition to the use of formal specification techniques, other quality assurance activities must be implemented as always during the system development process
- Should not blindly rely on formal methods
    - Formal methods are just one of many tools
    - Formal methods do not guarantee that the software developed is absolutely correct
        - e.g., it is impossible to use formal methods to prove that the conversion from informal requirements to formal specifications is correct. Therefore, other methods (for example, review, testing) must be used to verify the correctness of the software

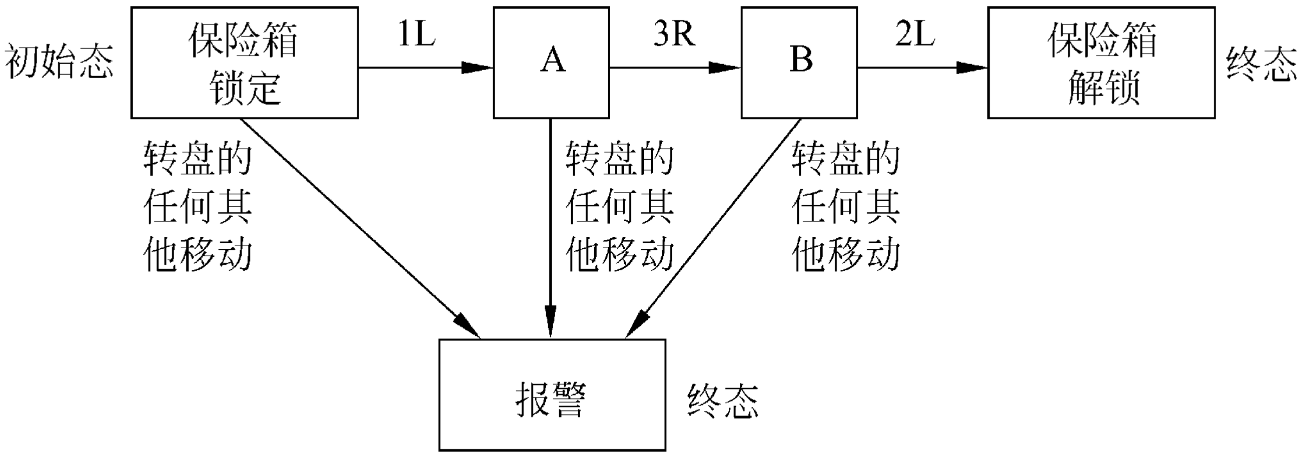
Guidelines for applying formal methods [cont.]

- Should be tested repeatedly
  - Formal methods not only cannot guarantee that the software system is absolutely correct, but also cannot prove that the system performance or other quality indicators meet the needs. Therefore, the importance of software testing has not diminished
- Should be reused
  - Even if formal methods are adopted, software reuse is still the only reasonable way to reduce software costs and improve software quality
  - The software components described by the formal method have clearly defined functions and interfaces, making them more reusable

Finite state machine

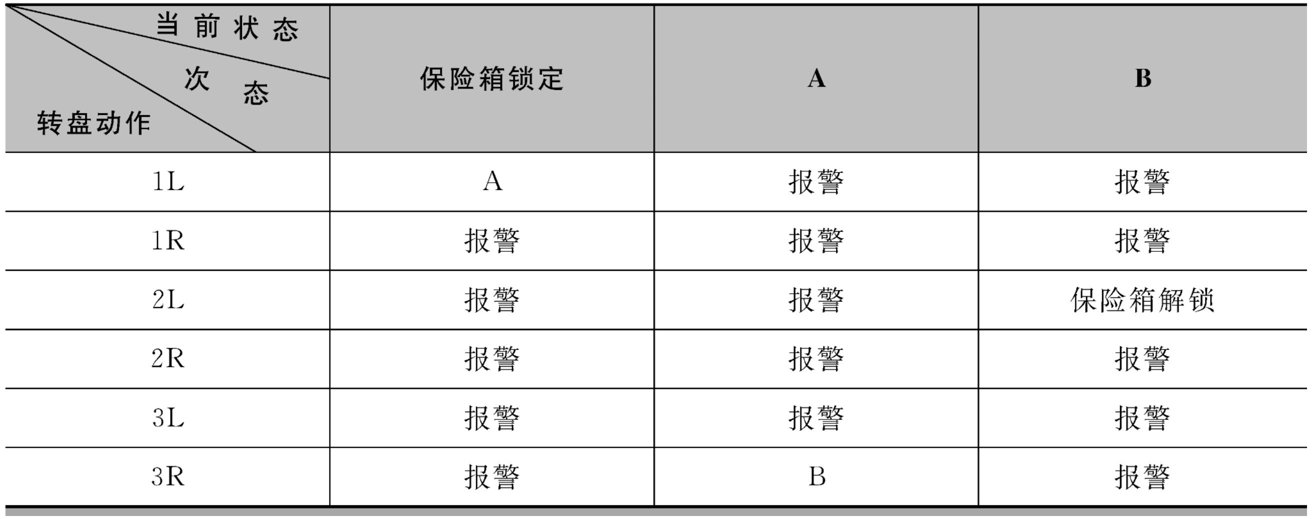
Example

- A composite lock is installed on a safe
- The lock has three positions, which are marked as 1, 2, and 3. The turntable can be rotated to the left (L) or right (R)
    - There are 6 possible movements of the turntable at any time
        - Namely 1L, 1R, 2L, 2R, 3L and 3R
    - The combination code of the safe is 1L, 3R, 2L, any other movement of the turntable will cause an alarm



Components of a finite state machine

- A finite state machine includes the following 5 parts
  - (1) State set J, (2) Input set K, (3) The transfer function T that determines the next state (secondary state) by the current state and the current input, (4) Initial state S and (5) Final state set F
- For the safe instance
  - Status set J: {safe lock, A, B, safe unlock, alarm}
  - Input set K: {1L, 1R, 2L, 2R, 3L, 3R}
  - Conversion function T: as shown in the table
  - Initial state S: Safe lock
  - Final state set F: {safe unlocked, alarm}



当前状态 次态 转盘动作	保险箱锁定	A	B
1L	A	报警	报警
1R	报警	报警	报警
2L	报警	报警	保险箱解锁
2R	报警	报警	报警
3L	报警	报警	报警
3R	报警	B	报警

Generalized representation of finite state machine

- A finite state machine can be represented as a 5-tuple (J, K, T, S, F), where
  - J is a finite set of non-empty states
  - K is a finite non-empty input set
  - T is a conversion function from (J-F)×K to J
  - S∈J, is an initial state
  - F is included in J, which is the final state set
- Another instance
  - Every menu-driven user interface is an implementation of a finite state machine
  - The display of a menu corresponds to a state, keyboard input or mouse selection of an icon is an event that causes the system to enter other states
  - Each transition of the state has the following form
    - Current state [menu] + event [selected item] ➢ next state
- In order to specify a system, it is usually necessary to make a useful extension to the finite state machine
  - Add the sixth component to the aforementioned 5-tuple: the predicate set P
  - Each of these predicates is a function of the system's global state Y
  - The conversion function T is now a function from (J-F)×K×P to J
  - The conversion rule is in the form
    - Current state 〔menu〕 +event 〔selected item〕 +predicate ➢ next state

Example

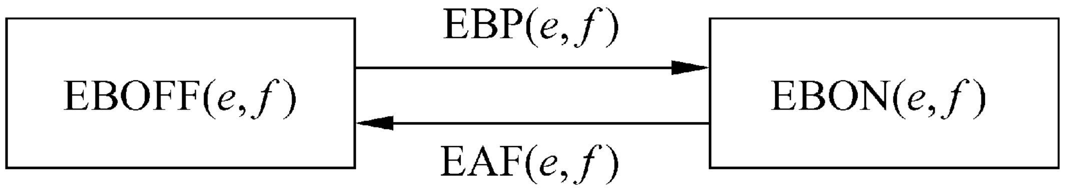
- A set of products for controlling n elevators is required in an m-storey building, and the n elevators are required to move between floors according to the constraint conditions C1, C2 and C3
  - C1
    - There are m buttons in each elevator, and each button represents a floor
    - When a button is pressed, the button indicator light is on, and the elevator moves to the corresponding floor at the same time, and the indicator light goes out when it reaches the floor specified by the button
  - C2
    - Except for the lowest and highest floors of the building, there are two buttons on each floor to request the elevator to go up and down respectively.
    - When one of these two buttons is pressed, the corresponding indicator light is on, when the elevator reaches this floor, the light is off, and the elevator moves in the required direction
  - C3
    - When there is no request for the elevator, it closes the door and stops at the current floor

Example [cont.]

- Now use an extended finite state machine to specify this product
    - There are two button sets in the question
        - Each of the n elevators has m buttons, one button corresponds to a floor
            - Because these m×n buttons are in the elevator, they are called elevator buttons
        - There are two buttons on each floor, one requesting up and the other requesting down. These buttons are called floor buttons

Example – Elevator Button [cont.]

- Let EB(e,f) mean press the button in elevator e and request to go to floor f
- EB(e,f) has two states
  - The button is illuminated (open) ➢ EBON(e,f): The elevator button (e,f) is turned on
  - The button is not illuminated (closed) ➢ EBOFF(e,f): The elevator button (e,f) is off
- Rule
  - If the elevator button (e, f) is illuminated and the elevator reaches the f floor, the button will go out
  - If the elevator button is off, the button will glow when it is pressed
- Elevator button event
  - EBP(e,f): The elevator button (e,f) is pressed ➢ Elevator Button is Pressed
  - EAF(e,f): Elevator e arrives at floor f ➢ Elevator Arrival at Floor



$\mathrm{EBOFF}(e, f)$

$\mathrm{EBP}(e, f)$

$\mathrm{EBON}(e, f)$

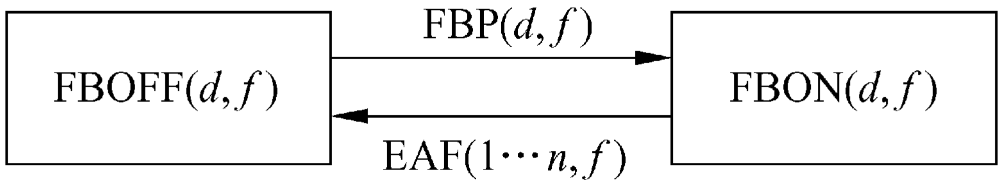
$\mathrm{EAF}(e, f)$

Example - Elevator Button [cont.]

- In order to define the state transition rules associated with these events and states, a predicate V(e,f) is needed, which has the following meaning:
  - V(e,f): elevator e stops at floor f
- If the elevator button (e, f) is in the off state [current state], and the elevator button (e, f) is pressed [event], and the elevator e is not on the f floor [predicate], the elevator button turns on and emits light [next state]
  - The formal description of the state transition rules is as follows
    - EBOFF(e,f)+EBP(e,f)+not V(e,f) ➢ EBON(e,f)
- If the elevator reaches the f floor and the elevator button is turned on, then it will go out
  - The conversion rule is expressed as:
    - EBON(e,f)+EAF(e,f) ➢ EBOFF(e,f)

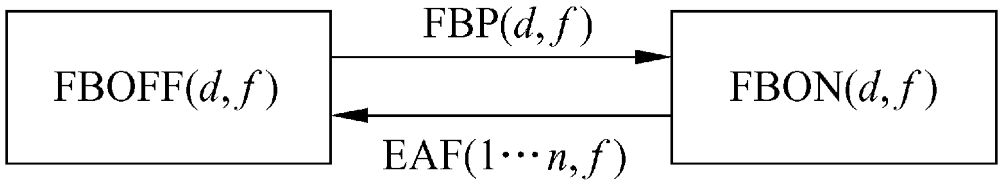
Example - Floor Button [cont.]

- Let FB(d,f) denote the button of the f floor requesting the elevator to move in the d direction
- FB(d,f) has two states
  - FBON(d,f): The floor buttons (d,f) are turned on
  - FBOFF(d,f): Floor button (d,f) is off
- Rule
  - If the floor button is turned on, and an elevator reaches the f floor, the button is turned off
  - If the floor button was turned off, the button will be turned on after being pressed
- Floor button event
  - FBP(d,f): Floor Button is Pressed ➢ Floor Button is Pressed
  - EAF(1…n,f): Elevator 1 or…or n arrives at floor f ➢ Elevator Arrival at Floor
    - Where 1...n means either 1 or 2...or n



Example - Floor Button [cont.]

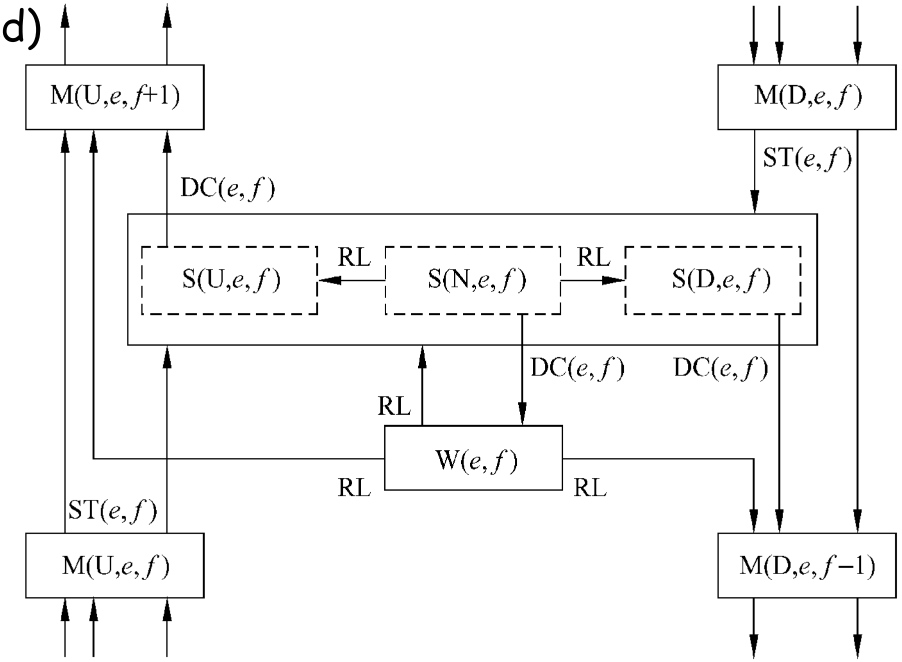
- Define a predicate S(d,e,f), its meaning is as follows:
    - S(d,e,f): The elevator e stops at floor f and the moving direction is determined by d
        - The moving direction d can be taken as upward (d=U) or downward (d=D) or to be determined (d=N)
        - The above predicate is actually a state. Formal methods allow events and states to be treated as predicates
- The formal description of the state transition rules is as follows
    - FBOFF(d,f)+FBP(d,f)+not S(d,1…n,f) ≻ FBON(d,f)
    - FBON(d,f)+EAF(1…n,f)+S(d,1…n,f) ≻ FBOFF(d,f)
        - Among them, d=U / D
    - If the floor button that requests the elevator to move in the d direction on the f floor is in the off state, and the button is now pressed, and there is no elevator stopping at the f floor and ready to move in the d direction, the floor button is turned on
    - If the floor button is turned on and at least one elevator reaches the f floor, the elevator will move in the d direction, and the button will be turned off



Example - Button [cont.]

- The predicate V(e,f) defined in the elevator button state transition rule can be redefined as follows with the predicate S(d,e,f)
  - V(e,f)=S(U,e,f)or S(D,e,f)or S(N,e,f)
    - That is, the three stop states of the elevator
- Now turn to discuss the status of the elevator and its transition rules
  - An elevator state contains many sub-states
  - M(d,e,f): The elevator e is moving in the d direction, and it will arrive at the fth floor
  - S(d,e,f): Elevator e stops at floor f and will move in direction d (the door has not been closed yet)
  - W(e,f): Elevator e is waiting at floor f (the door has been closed)

- DC(e,f): elevator e closes the door on floor f
- ST(e,f): When the elevator e is close to the f floor, the sensor is triggered, and the elevator controller determines whether the elevator stops on the current floor
- RL: The elevator button or floor button is pressed to enter the open state, log in demand



Example [cont.]

- State transition of elevator
  - For simplicity, the rules given here only happen when the door is closed
  - S(U,e,f)+DC(e,f) ➢ M(U,e,f+1)
    - If elevator e stops at floor f and is ready to move up, and the door is closed, the elevator will move up one floor
  - S(D,e,f)+DC(e,f) ➢ M(D,e,f-1)
  - S(N,e,f)+DC(e,f) ➢ W(e,f)
    - Respectively correspond to the situation that the elevator is about to descend or there is no pending request

Evaluation of finite state machine

- The finite state machine method uses a simple format to describe the specification
    - Current state + event + predicate ➢ next state
- The specification of the finite state machine expression is easy to write and verify, and it can be easily transformed into design or program code
    - A CASE tool can be developed to directly convert a finite state machine specification into source code
    - Maintenance can be achieved by re-transformation
        - I.e. if a new state or event is needed
            - First modify the specification
            - Then directly generate a new version of the product from the new specification
- The finite state machine method is more accurate than the data flow graph technique and is as easy to understand as it is
- Shortcoming
    - When developing a large system, the number of triples (i.e. states, events, predicates) will grow rapidly
    - Like the data flow graph method, the formal finite state machine method has not yet dealt with timing requirements

Petri Net

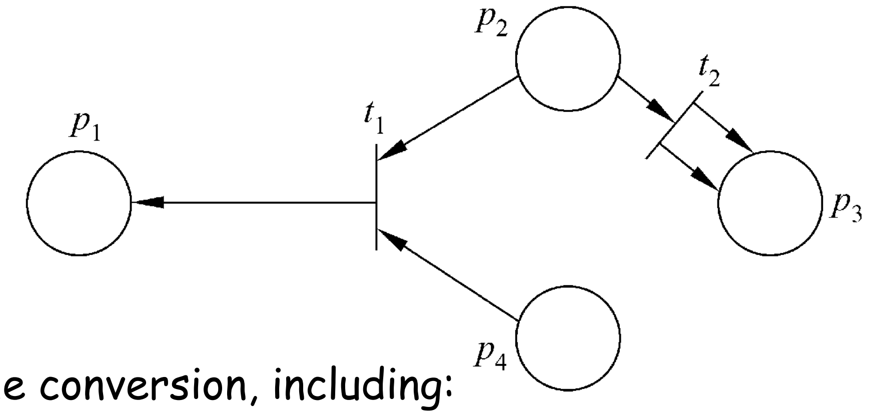
The basic role of Petri

- One of the main problems encountered in concurrent systems is the timing problem
    - The problem can take many forms
        - Such as synchronization issues, race conditions, and deadlock issues
    - Timing problems are usually caused by poor design or incorrect implementation
        - Furthermore, the design or implementation is usually caused by bad specifications
        - If the specifications are not appropriate, there is a risk of imperfect design or implementation
- An effective technique for determining the timing problems implicit in the system is Petri nets

Brief statement

- Petri net was invented by Carl Adam Petri
- At first, only automation experts were interested in Petri nets. Later, Petri nets are also widely used in computer science.
  - For example, in areas such as performance evaluation, operating systems and software engineering
- It has been proved that Petri nets can effectively describe concurrent activities
- Petri net contains four elements
  - A set of positions P
  - A set of conversion T
  - Input function I and
  - Output function O

Petri net construction



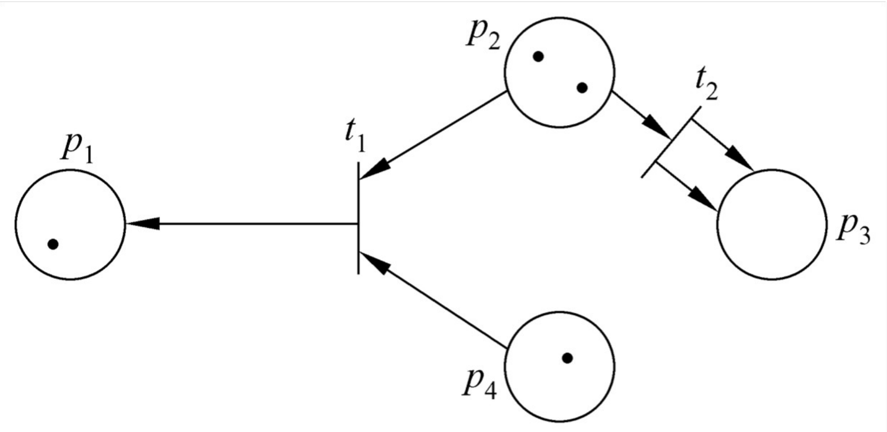
- A set of positions P is {P1, P2, P3, P4}
  - Use a circle to represent the position in the figure
- A set of conversion T is {t1, t2}
  - Use short straight lines to indicate conversion in the figure
- Two input functions for conversion
  - It is represented by an arrow pointing from the position to the conversion, including:
    - I(t1)={P2, P4}
    - I(t2)={P2}
- Two output functions for conversion
  - Represented by an arrow pointing to the position by the transition
  - O(t1)={P1}
  - O(t2)={P3, P3}
    - There are two P3 in the output function O(t2) because there are two arrows pointing from t2 to P3

Formal Petri Net

- A quadruple C = (P, T, I, O)
  - P = {P1,..., Pn} is a finite position set, n≥0
  - T ={t1,...,tm} is a finite transformation set, m≥0, and T and P do not intersect
  - I: T→P^∞ is the input function, which is the mapping from conversion to positional disordered unit groups (bags)
  - O: T→P^∞ is the output function, which is the mapping from the conversion to the positional disordered unit group
    - A disordered unit group or multiple group is a generalized set that allows multiple instances of an element

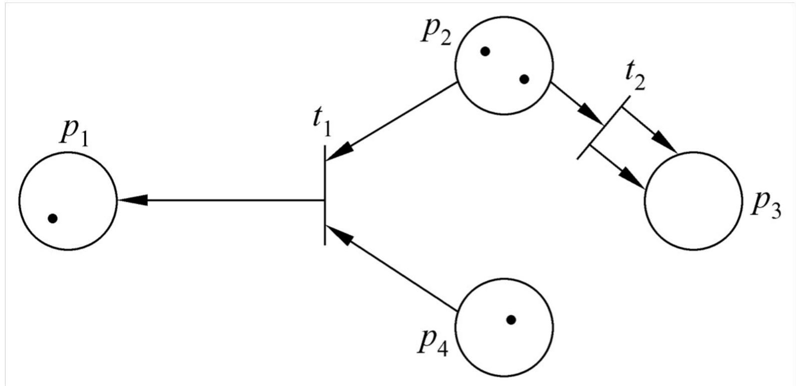
Formal Petri Net

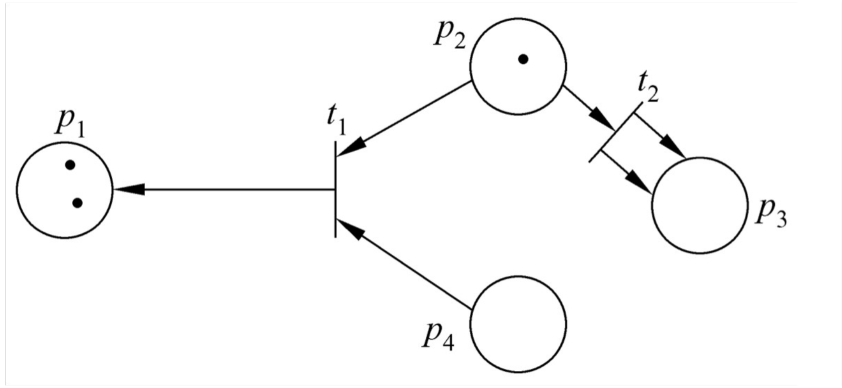
- The mark of Petri net is the distribution of tokens in Petri net
  - For example, there are 4 tokens in the example image
    - One is in P1, two are in P2, not in P3, and the other is in P4
    - The above mark can be represented by a vector (1, 2, 0, 1)
    - Since P2 and P4 have the weights, t1 is activated (that is, excited)
    - Generally, when the number of tokens owned by each input position is greater than or equal to the number of lines from that position to the conversion, conversion is allowed
      - When t1 is activated, a token on P2 and P4 is removed, and a token on P1 is added
  - The total number of tokens in Petri Net is not fixed. In the above example, two tokens are removed, and only one token can be added to P1

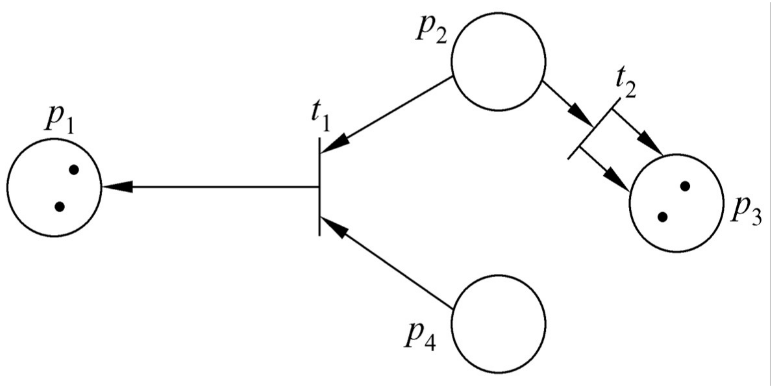


Formal Petri Net

- The mark of Petri net is the distribution of tokens in Petri net
    - P2 is superscript, so t2 can also be excited
    - When t2 is activated, one token will be removed from P2, and two tokens will be added to P3
- Petri nets are non-deterministic, that is, if several transitions meet the excitation conditions, any one of them can be excited
    - The Petri net in the figure is marked as (1, 2, 0, 1), and both t1 and t2 can be excited
        - Assuming that t1 is excited, the result is shown in the middle figure below, marked as (2, 1, 0, 0)
        - At this time, only t2 can be excited
        - If t2 is also activated, the token is removed from P2, two new tokens are placed on P3, and the result is marked as (2, 0, 2, 0)

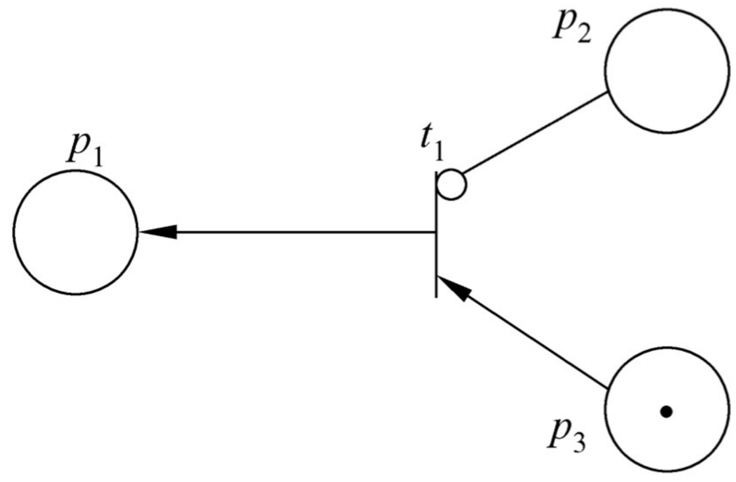






Formal Petri Net

- More formally, the mark M in Petri net C=(P, T, I, O) is a mapping from a set of positions P to a set of non-negative integers:
    - M: P→{0,1,2,...}
    - Then the labeled Petri net becomes a five-tuple (P, T, I, O, M).
- An important extension to Petri nets is to add forbidden lines
    - The forbidden line is the input line marked with a small circle instead of an arrow
    - Generally, when there is at least one token on each input line, and there is no token on the prohibition line, the corresponding conversion is allowed
        - In the example picture, there is a token on P3 and no token on P2, so the transition t1 can be stimulated



Example

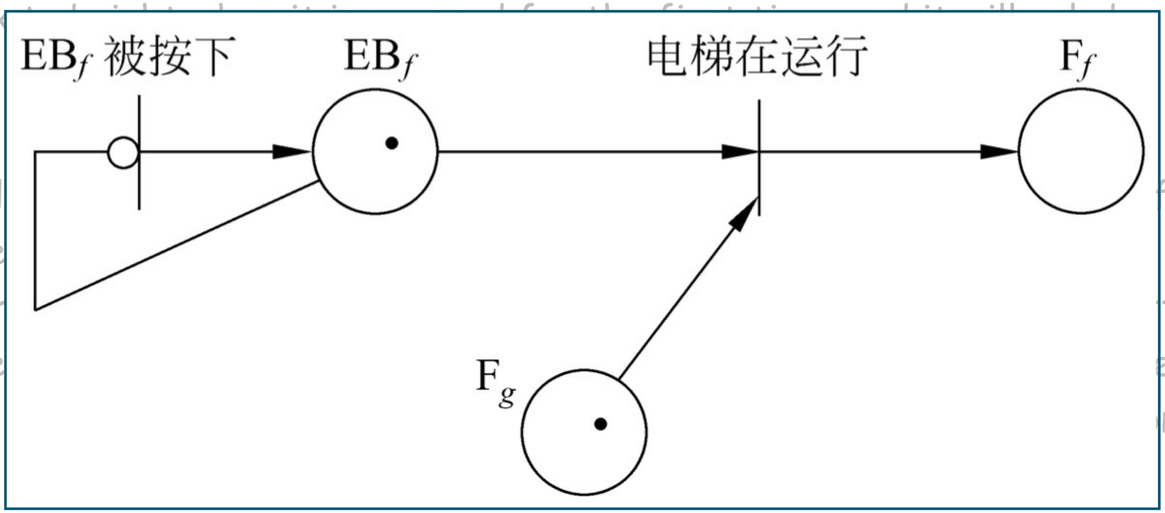
- When the Petri net is used to represent the specification of the elevator system, each floor is represented by a position Ff (1≤f≤m). In the Petri net, the elevator is represented by a token. The right mark at location Ff indicates that there is an elevator on floor f
- A set of products for controlling n elevators is required in an m-storey building, and the n elevators are required to move between floors according to the constraint conditions C1, C2 and C3
  - C1
    - There are m buttons in each elevator, and each button represents a floor
    - When a button is pressed, the button indicator light is on, and the elevator moves to the corresponding floor at the same time, and the indicator light goes out when it reaches the floor specified by the button
  - C2
    - Except for the lowest and highest floors of the building, there are two buttons on each floor to request the elevator to go up and down respectively.
    - When one of these two buttons is pressed, the corresponding indicator light is on, when the elevator reaches this floor, the light is off, and the elevator moves in the required direction
  - C3
    - When there is no request for the elevator, it closes the door and stops at the current floor

Example – Elevator Button

- The first constraint C1
  - Each elevator has m buttons, and each floor corresponds to one button
  - When a button is pressed, the button indicator light is on, indicating that the elevator moves to the corresponding floor
  - When the elevator reaches the designated floor, the button will go out
- In order to express the specifications of elevator buttons with Petri nets, other positions must be set in Petri nets
  - The button of floor f in the elevator is represented by position EBf in the Petri net (1≤f≤m)
  - There is a token on the EBf, which means that the button of floor f in the elevator has been pressed
  - The elevator button will only turn from dark to bright when it is pressed for the first time, and it will only be ignored if it is pressed later
- Describe
  - Assuming that the button is not lit, obviously there is no token on the position EBf, so that the transition "EBf is pressed" is allowed to occur in the presence of a forbidden line
  - Assuming that the button is now pressed, the conversion is activated and a token is placed on the EBf. No matter how many times the button is pressed thereafter, the combination of the prohibition line and the existing token determines that the conversion "EBf is pressed" can no longer be Excited, therefore, the number of tokens on the position EBf will not be more than 1

Example – Elevator Button

- The first constraint C1
  - Each elevator has m buttons, and each floor corresponds to one button
  - When a button is pressed, the button indicator light is on, indicating that the elevator moves to the corresponding floor
  - When the elevator reaches the designated floor, the button will go out
- In order to express the specifications of elevator buttons with Petri nets, other positions must be set in Petri nets
  - The button of floor f in the elevator is represented by position EBf in the Petri net (1≤f≤m)
  - There is a token on the EBf, which means that the button of floor f in the elevator has been pressed
  - The elevator button will only turn from dark ignored if it is pressed later



- Describe
  - Assuming that the button is not lit, obviousl f is pressed" is allowed to occur in the presence
  - Assuming that the button is now pressed, th ter how many times the button is pressed thereafte etermines that the conversion "EBf is pressed" can no n EBf will not be more than 1

Example – Elevator Button

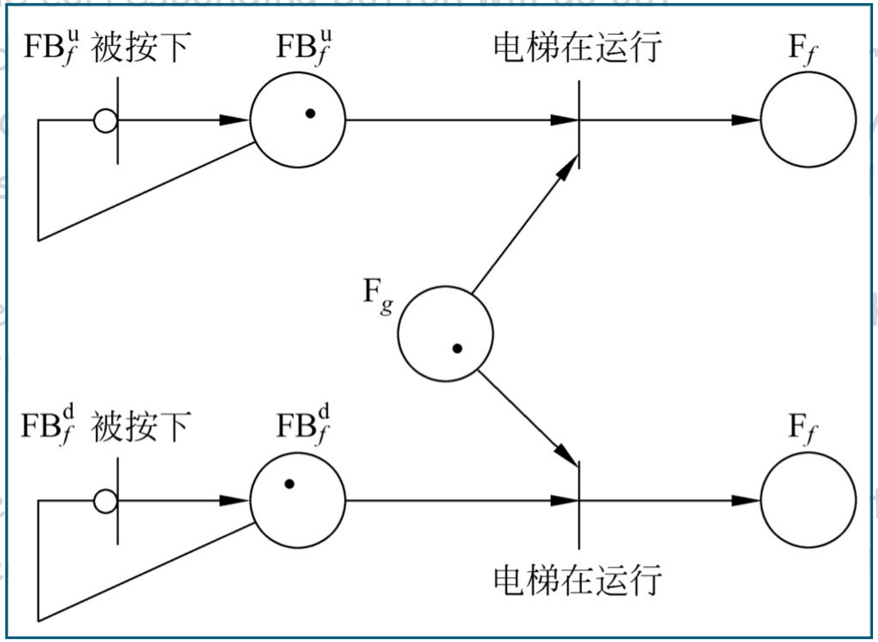
- Suppose the elevator goes from the g floor to the f floor, because the elevator is on the g floor, and there is a token at the position Fg
  - Since there is a token on each input line, the transition "elevator is running" is activated, so the tokens on EBf and Fg are removed, the button EBf is turned off, and a new token appears at the position Ff, that is, the transition The excitation of the elevator drives from the g floor to the f floor
- In fact, it takes time for the elevator to move from floor g to floor f. In order to deal with this situation and other similar problems (for example, the button cannot be illuminated immediately after being pressed due to physical reasons), the Petri net model must be added time limit
  - That is, in the standard Petri net, the conversion is completed instantaneously, but in reality, it takes time to control the Petri net so that the conversion is related to non-zero time.

Example – Floor Button

- The second constraint C2
  - Except for the first floor and the top floor, each floor has two buttons, one for the elevator to go up, and the other for the elevator to go down; these buttons light up when pressed, when the elevator reaches the floor and will move in the specified direction , The corresponding button will go out
- In the Petri net, the floor buttons are represented by the positions FBfu and FBfd, which represent the buttons on the f floor requesting the elevator to go up and down/the bottom floor respectively.
  - The button is FB1u, the button on the top layer is FBmd, and there are two buttons FBfu and FBfd on each layer in the middle (1 < f < m)
  - When the elevator moves from the g floor to the f floor, according to the requirements of the elevator passengers, a certain floor button is on or both floor buttons are on
    - If both buttons are on, only one button is off
    - The Petri net as shown in the figure can ensure that when both buttons are on, only one button goes out
      - But to ensure that the button is extinguished correctly, a more complex Petri net model is required
- Article 3 Constraint C3
  - When there is no request (there is no token on FBfu and FBfd), any transition "Elevator is running" cannot be activated

Example – Floor Button

- The second constraint C2
  - Except for the first floor and the top floor, each floor has two buttons, one for the elevator to go up, and the other for the elevator to go down; these buttons light up when pressed, when the elevator reaches the floor and will move in the specified direction , The corresponding button will go out
- In the Petri net, the floor buttons are representec esent the buttons on the f floor requesting the elevator to g .
  - The button is FB1u, the button on the top layer is FBfd on each layer in the middle (1 < f < m)
  - When the elevator moves from the g floor to the he elevator passengers, a certain floor button is on or both f
    - If both buttons are on, only one button is off
    - The Petri net as shown in the figure can ensure tton goes out
      - But to ensure that the button is extinguishe is required
- Article 3 Constraint C3
  - When there is no request (there is no token on FBfu and FBfd), any transition "Elevator is running" cannot be activated

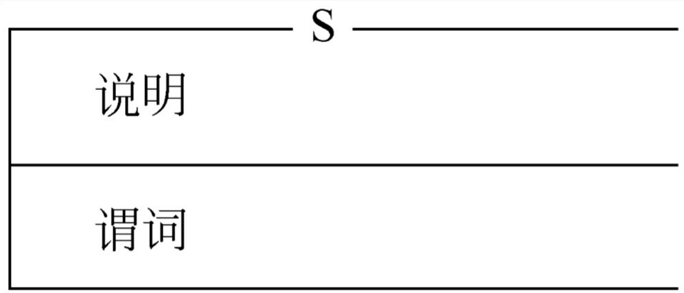


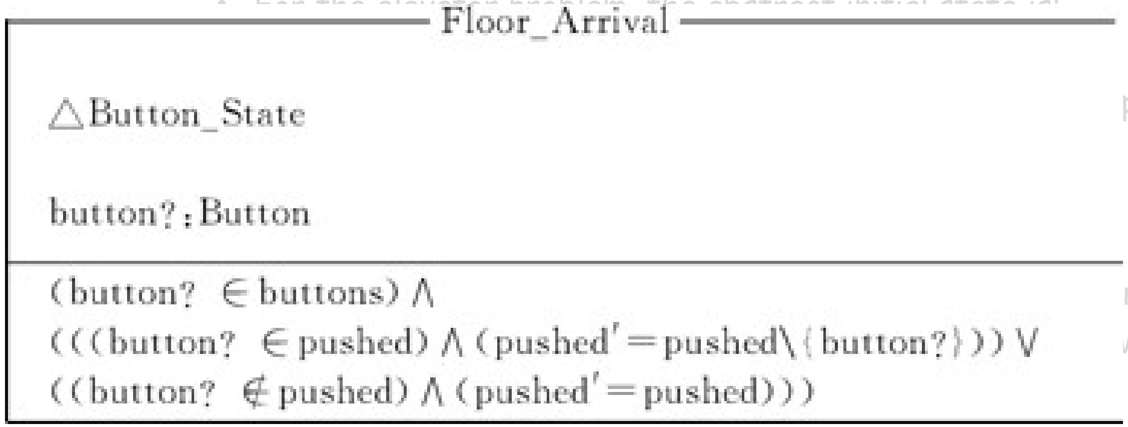
Z Language

- The simplest formal specification described in Z language contains 4 parts
  - Given set, data type and constant
    - A Z specification starts with a given set of initializations
    - The so-called initial set is a set that does not need to be defined in detail, and the set is expressed in square brackets.
      - For the elevator problem, the given initialization set is called Button, which is the set of all buttons. Therefore, the Z specification starts with: [Button]
  - State definition
    - A Z specification is composed of a number of "schemas", each of which contains a set of variable descriptions and a series of predicates that limit the value range of the variable
      - For example, the format of cell S is shown in the diagram
  - Initial state
    - The abstract initial state refers to the state when the system is first turned on
    - For the elevator problem, the abstract initial state is:
      - Button_Init≜ 〔Button_State | pushed=Φ〕
      - The above formula means that the pushed set is empty when the system is turned on for the first time, that is, all buttons are in the off state
  - Operate
    - If a button that was originally in the off state is pressed, the button is turned on, and the button is added to the pushed set
    - If the elevator reaches a certain floor, if the corresponding floor button has been turned on, it will be closed at this time; similarly, if the corresponding elevator button has been turned on, it will also be closed at this time. In other words, if "button?" belongs to the pushed set, move it out of the set

Z Language

- The simplest formal specification described in Z language contains 4 parts
  - Given set, data type and constant
    - A Z specification starts with a given set of initializations
    - The so-called initial set is a set that does not need to be defined in detail, and the set is expressed in square brackets.
      - For the elevator problem, the given initialization set is called Butt the Z specification starts with: [Button]
  - State definition
    - A Z specification is composed of a number of "schemas", each of which c ies of predicates that limit the value range of the variable
      - For example, the format of cell S is shown in the diagram
  - Initial state
    - The abstract initial state refers to the state when the system is first turned on
    - For the elevator problem, the abstract initial state is:



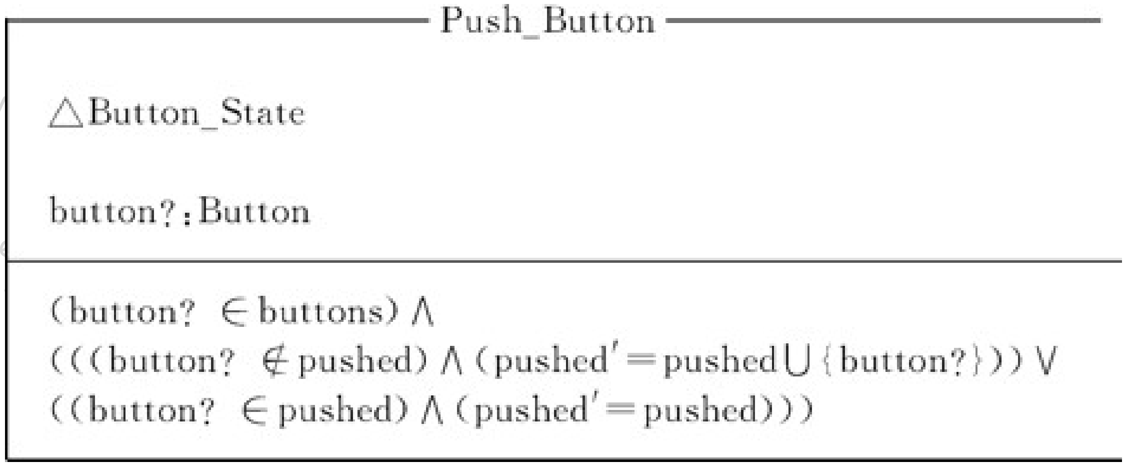


Floor_Arrival

ΔButton_State

button?: Button

$(button? \in buttons) \wedge$
$(((button? \in pushed) \wedge (pushed' = pushed \setminus \{button?\})) \vee$
$((button? \notin pushed) \wedge (pushed' = pushed)))$



Push_Button

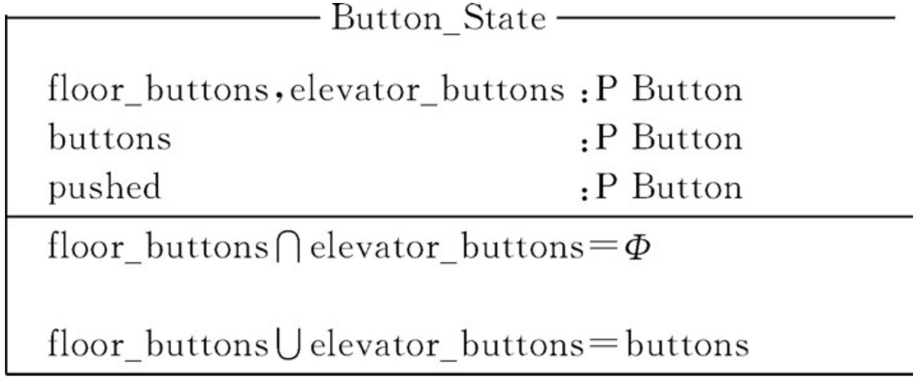
ΔButton_State

button?: Button

$(button? \in buttons) \wedge$
$(((button? \notin pushed) \wedge (pushed' = pushed \cup \{button?\})) \vee$
$((button? \in pushed) \wedge (pushed' = pushed)))$

Example

- In the elevator problem, Button has 4 subsets
  - floor_buttons (collection of floor buttons)
  - elevator_buttons (collection of elevator buttons)
  - buttons (collection of all buttons in the elevator problem)
  - pushed (the set of all pressed buttons, that is, the set of all the buttons in the open state)



Button_State

floor_buttons, elevator_buttons : P Button
buttons : P Button
pushed : P Button

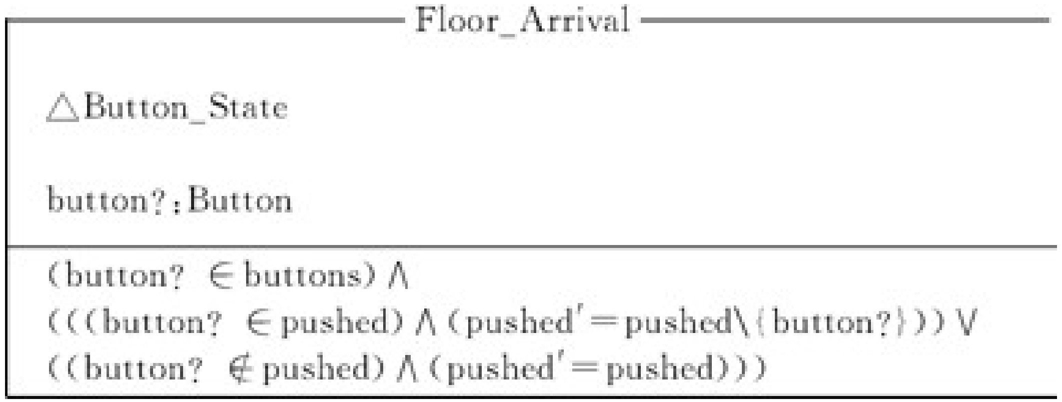
floor_buttons $\cap$ elevator_buttons $= \Phi$

floor_buttons $\cup$ elevator_buttons $=$ buttons

- Button_State schema (cell)
  - Among them, the symbol P represents the power set (that is, all subsets of a given set)
- Constraint statement
  - The floor_buttons set does not intersect with the elevator_buttons set
  - And they together form the buttons set

Example

- The predicate part of the operation contains a set of pre-conditions for calling the operation and post-conditions after the operation is completely completed
  - If the pre-condition is established, the post-condition can be obtained after the operation is completed
  - However, if the operation is called when the precondition is not established, the specified result will not be obtained (so the result is unpredictable).
    - If the elevator reaches a certain floor, if the corresponding floor button has been turned on, then it will be closed at this time
    - If the corresponding elevator button is already turned on, it will also be turned off at this time
      - That is, if "button?" belongs to the pushed set, move it out of the set



Floor_Arrival

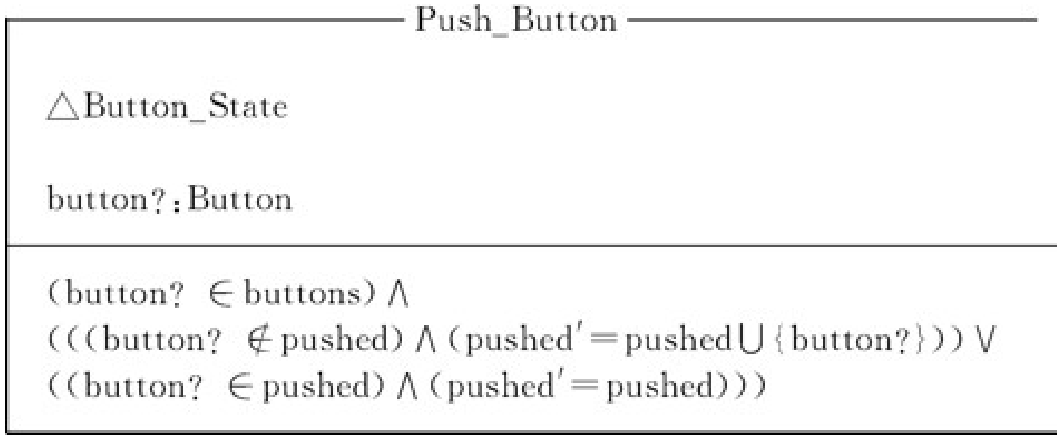
ΔButton_State

button? : Button

$(button? \in buttons) \wedge$

$(((button? \in pushed) \wedge (pushed' = pushed \setminus \{button?\})) \vee$

$((button? \notin pushed) \wedge (pushed' = pushed))$



Push_Button

ΔButton_State

button? : Button

$(button? \in buttons) \wedge$

$(((button? \notin pushed) \wedge (pushed' = pushed \cup \{button?\})) \vee$

$((button? \in pushed) \wedge (pushed' = pushed))$

Z language evaluation

- Z language has been successfully used in many software development projects,
    - At present, Z may be the most widely used formal language, especially in large-scale projects. The advantages of Z language are more obvious.
- Z language is licensed
    - It is easier to find errors in the specifications written in Z, especially when reviewing the specifications yourself and reviewing the design and code based on the formal specifications.
    - When writing specifications with Z, the author is required to use the Z specifier very accurately
        - Due to the high requirements for accuracy, compared with informal specifications, ambiguity, inconsistency and omissions are reduced
    - Z is a formal language, developers can strictly verify the correctness of specifications when needed
    - It is quite difficult to fully learn the Z language
    - Using Z language can reduce software development costs
        - Although it takes more time to write specifications in Z than to use informal techniques, the total time required for the development process is reduced
    - Although users cannot understand the specifications written in Z, they can rewrite the specifications in natural language based on the Z specifications.