Safety Analysis

Safety-critical Software

- Systems whose failure can threaten human life or cause serious environmental damage, e.g., control system for chemical plant
- Increasingly important as computers replace simpler, hard-wired control systems
- Primary safety-critical systems
  - Embedded software systems whose failure can cause the associated hardware to fail and directly threaten people.
- Secondary safety-critical systems
  - Systems whose failure results in faults in other systems which can threaten people

Other Critical Systems

- Mission-critical systems: A system whose failure may result in the failure of some goal-directed activity, e.g., navigational system for spacecraft
- Business critical system: A system whose failure may result in the failure of the business using that system,e.g., customer account bank system

Safety vs. Reliability

- Not the same thing
- Reliability is concerned with conformance to a given specification and delivery of service
- Safety is concerned with ensuring system cannot cause damage irrespective of whether or not it conforms to its specification
- A system may be reliable but not safe – but, usually, if a critical system is unreliable it is likely to be unsafe …

Reliable Unsafe Systems

- Specification errors
  - If the system specification is incorrect then the system can behave as specified but still cause an accident
- Hardware failures generating spurious inputs
  - Hard to anticipate in the specification
- Context-sensitive commands, i.e., issuing a correct command at the wrong time
  - Often the result of operator error

Definitions

- Mishap (or accident)
  - An unplanned event or event sequence which results in human death or injury. It may be more generally defined as covering damage to property or the environment
- Damage
  - A measure of the loss resulting from a mishap
- Hazard
  - A condition with the potential for causing or contributing to a mishap
  - 2 characteristics: severity, probability
- Hazard severity
  - An assessment of the worst possible damage which could result from a particular hazard

Definitions II

- Hazard probability (or likelihood)
  - The probability of the events occurring which create a hazard (qualitative or quantitative)
- Expected loss (or Hazard level): for all mishaps, probability * severity
- Risk
  - The risk is assessed by considering the hazard probability, the hazard severity, and the probability that the hazard will result in a mishap.
  - The objective of all safety systems is to minimize risk, by minimizing any or all of its components.

Severity - MIL-STD-882B

- Severity
  - Category I: Catastrophic; may cause death or system loss
  - Category II: Critical; may cause severe injury, severe occupational illness, or major system damage
  - Category III: Marginal; may cause minor injury, minor occupational illness, or minor system damage
  - Category IV: Negligible; will not result in injury, occupational illness, or system damage

Hazard Probability - Subjective Scale

- Frequent: Likely to occur frequently
- Probable: Will occur several times in unit life
- Occasional: Likely to occur sometime in unit life
- Remote: Unlikely to occur in unit life, but possible
- Improbable: Extremely unlikely to occur
- Impossible: Equal to a probability of zero

Example of Risk Evaluation

- Robot Control System:
  - Probability the computer causes a spurious or unexpected machine movement (hazard)
  - Probability a human is in the field of movement
  - Probability the human has no time to move
  - Severity of worst-case consequences
- Continuous and protective monitoring function for a plant:
  - Probability of a dangerous plant condition arising (hazard)
  - Probability of the computer not detecting it
  - Probability of the computer not initiating its safety function
  - Probability of the safety function not preventing the hazard
  - Probability of conditions occurring that will cause the hazard to lead to an accident
  - Worst-case severity of the accident

Risk Assessment

- Assesses hazard severity, hazard probability and accident probability
- Outcome of risk assessment may be defined as a statement of acceptability
  - Intolerable. Must never arise or result in an accident
  - As low as reasonably practical(ALARP). Must minimize possibility of hazard given cost and schedule constraints
  - Acceptable. Consequences of hazard are acceptable and no extra costs should be incurred to reduce hazard probability

Risk Acceptability

- The acceptability of a risk is determined by human, social and political considerations
- In most societies, the boundaries between the regions are pushed upwards with time i.e. society is less willing to accept risk
  - For example, the costs of cleaning up pollution may be less than the costs of preventing it but this may not be socially acceptable
- Risk assessment is subjective
  - Risks are identified as probable, unlikely, etc. This depends on who is making the assessment

Safety Achievement

- The number of faults which can cause significant safety-related failures is usually a small subset of the total number of faults which may exist in a system
- Safety achievement should ensure that either these faults cannot occur or, if they do occur, they cannot result in a mishap
- Should also ensure that correct functioning of the system cannot cause a mishap
- Safety-related actions: Changes in design, inclusion of safety or warning devices, operational procedures

Safety Requirements

- The safety requirements of a system should be separately specified
- These requirements should be based on an analysis of the possible hazards and risks
- Safety requirements usually apply to the system as a whole rather than to individual sub-systems

Safety Analysis Process

- *Hazard and risk analysis*: Assess the hazards and the risks of damage associated with the system
- *Safety requirements specification*: Specify a set of safety requirements which apply to the system
- *Designation of safety-critical sub-systems*: Identify the sub-systems whose incorrect operation may compromise system safety (to act on them, according to the safety specifications)
- *Safety verification*: Check controls have been implemented and are effective
- *Safety validation (certification):* Check and test the overall resulting system safety

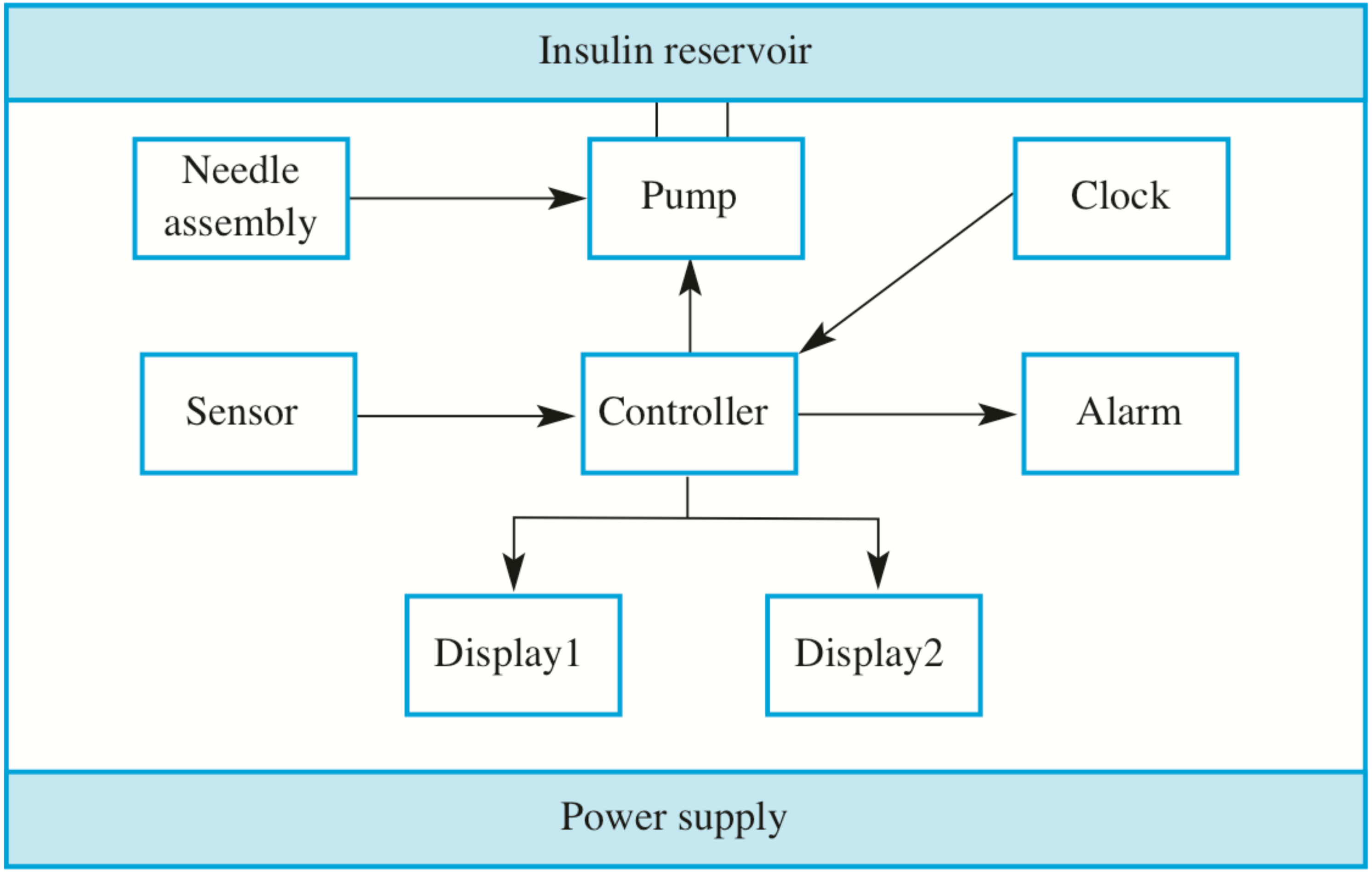
Hazard and Risk Analysis

- Hazard identification
  - Identify potential hazards which may arise
- Risk Analysis and Hazard classification
  - Assess the risk associated with each hazard
  - Rank hazards
- Hazard decomposition
  - Analyze hazards to discover their potential root causes
- Risk Reduction -> safety requirements
  - Define how each hazard must be taken into account when the system is designed, I.e., specifications of preventive or corrective measures
  - Cost benefit tradeoff

Insulin Delivery Example

- Simple example of a safety-critical system. Most medical systems are safety-critical
- People with diabetes cannot make their own insulin (used to metabolize sugar). It must be delivered externally
- Delivers a dose of insulin (required by diabetics) depending on the value of a blood sugar sensor

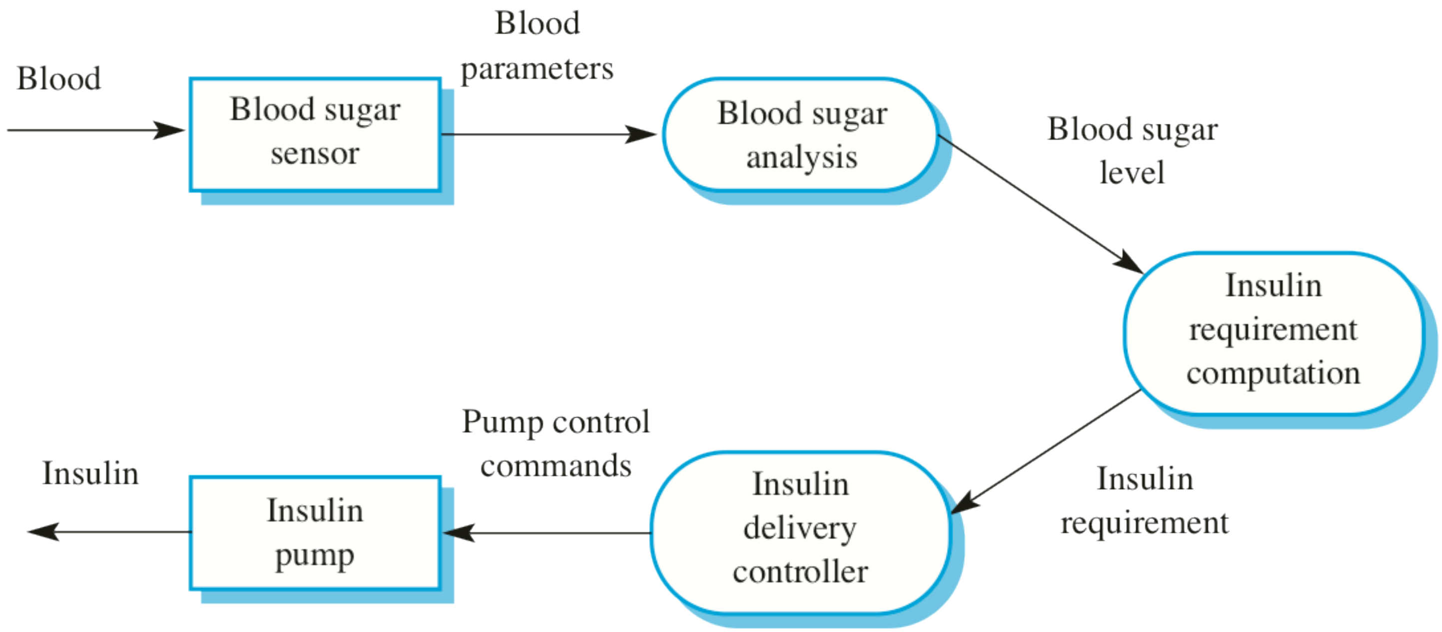
Insulin Pump



Sommerville

System Data Flow

- Data flow model of software-controlled insulin pump



Insulin System Hazard Identification

- **insulin overdose or underdose**
- power failure
- machine interferes electrically with other medical equipment such as a heart pacemaker
- parts of machine break off in patient's body
- poor sensor/actuator contact caused by incorrect fitting
- infection caused by introduction of machine
- allergic reaction to the materials or insulin used in the machine

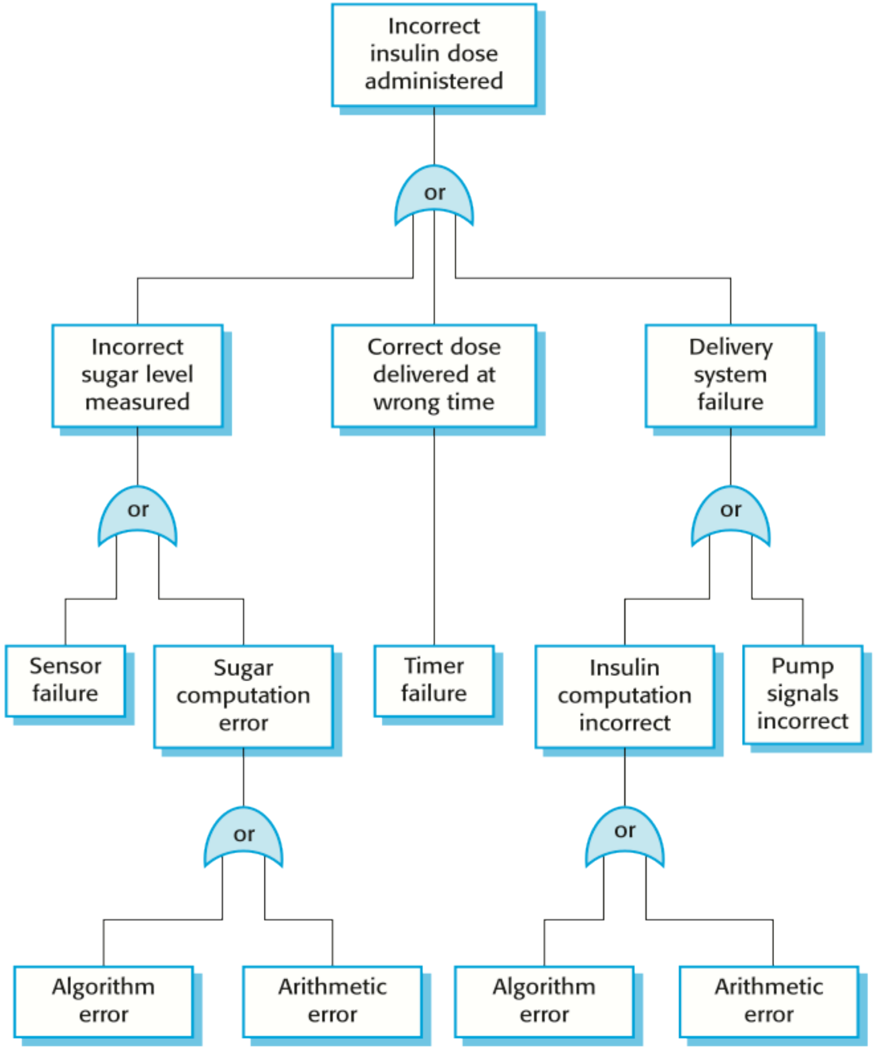
Risk Assessment Example

Identified hazard	**Hazard probability**	**Hazard severity**	**Estimated risk**	**Acceptability**
Insulin overdose	Medium	High	High	Intolerable
Insulin underdose	Medium	Low	Low	Acceptable
Power failure	High	Low	Low	Acceptable
Machine incorrectly fitted	High	High	High	Intolerable
Machine breaks in patient	Low	High	Medium	ALARP
Machine causes infection	Medium	Medium	Medium	ALARP
Electrical interference	Low	High	Medium	ALARP
Allergic reaction	Low	Low	Low	Acceptable

Fault-Tree Analysis

- Method of *hazard decomposition* which starts with an identified hazard and works backward to the causes of the hazard.
- Identify hazard from system definition
- Identify potential causes of the hazard. Usually there will be a number of alternative causes. Link these on the fault-tree with ‘or’ or ‘and’ logic gates
- Continue process until root causes are identified
- The hazard probability can then be assessed
- A design objective should be that no single cause can result in a hazard. That is, ‘or’s should be replaced by ‘and’s wherever possible

Insulin System Fault-Tree

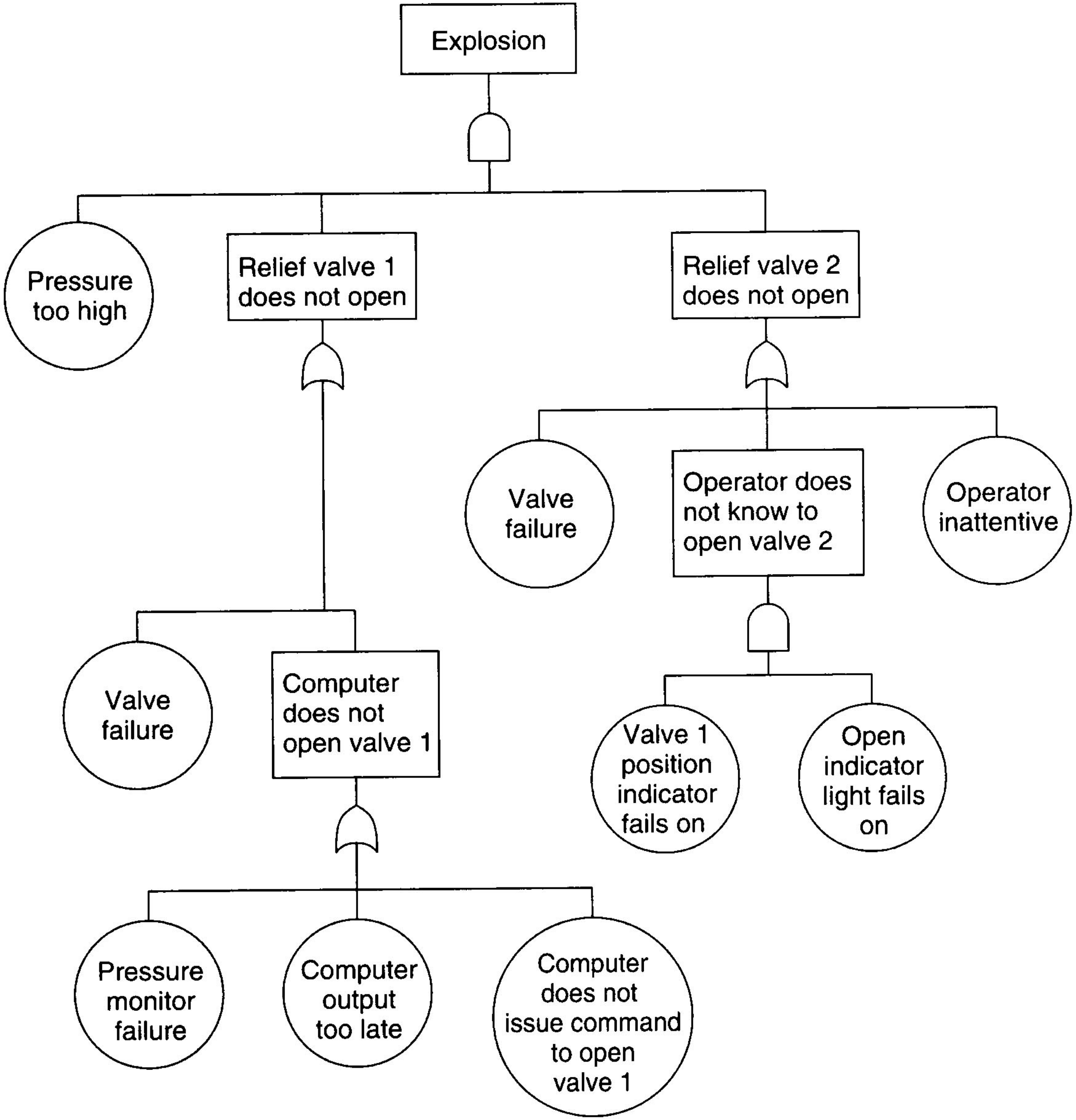


Sommerville

Fault Tree Gates

- The output of an ‘and’ gate exists only if all the inputs exists
- The output of an ‘or’ gate exists provided that at least one of the inputs exists
- The input events to an ‘or’ gate do not cause the event above the gate, but are simply re-expressions of the output event. In contrast, events attached to an ‘and’ gate are the causes of the above event.
- It is the causal relationship that differentiates an ‘and’ gate from an ‘or’ gate

Fault-Tree Example



Risk Reduction

- System should be specified so that hazards do not arise or, if they do, do not result in an accident
- Hazard avoidance
  - The system should be designed so that the hazard can never arise during correct system operation
- Hazard probability reduction
  - The system should be designed so that the probability of a hazard arising is minimized
- Accident prevention
  - If the hazard arises, there should be mechanisms built into the system to prevent an accident

Safety Assurance

Safety Validation

- Safety validation
  - Does the system always operate in such a way that accidents do not occur or that accident consequences are minimised?
- Demonstrating safety by testing is difficult because testing is intended to demonstrate what the system does in a particular situation. Testing all possible operational situations is impossible
- Normal reviews for correctness may be supplemented by specific techniques that are intended to focus on checking that unsafe situations never arise

Hazard-driven Assurance

- Effective safety assurance relies on hazard identification
- Safety can be assured by
  - Hazard avoidance
  - Accident avoidance
  - Protection systems
- Safety reviews should demonstrate that one or more of these techniques have been applied to all identified hazards

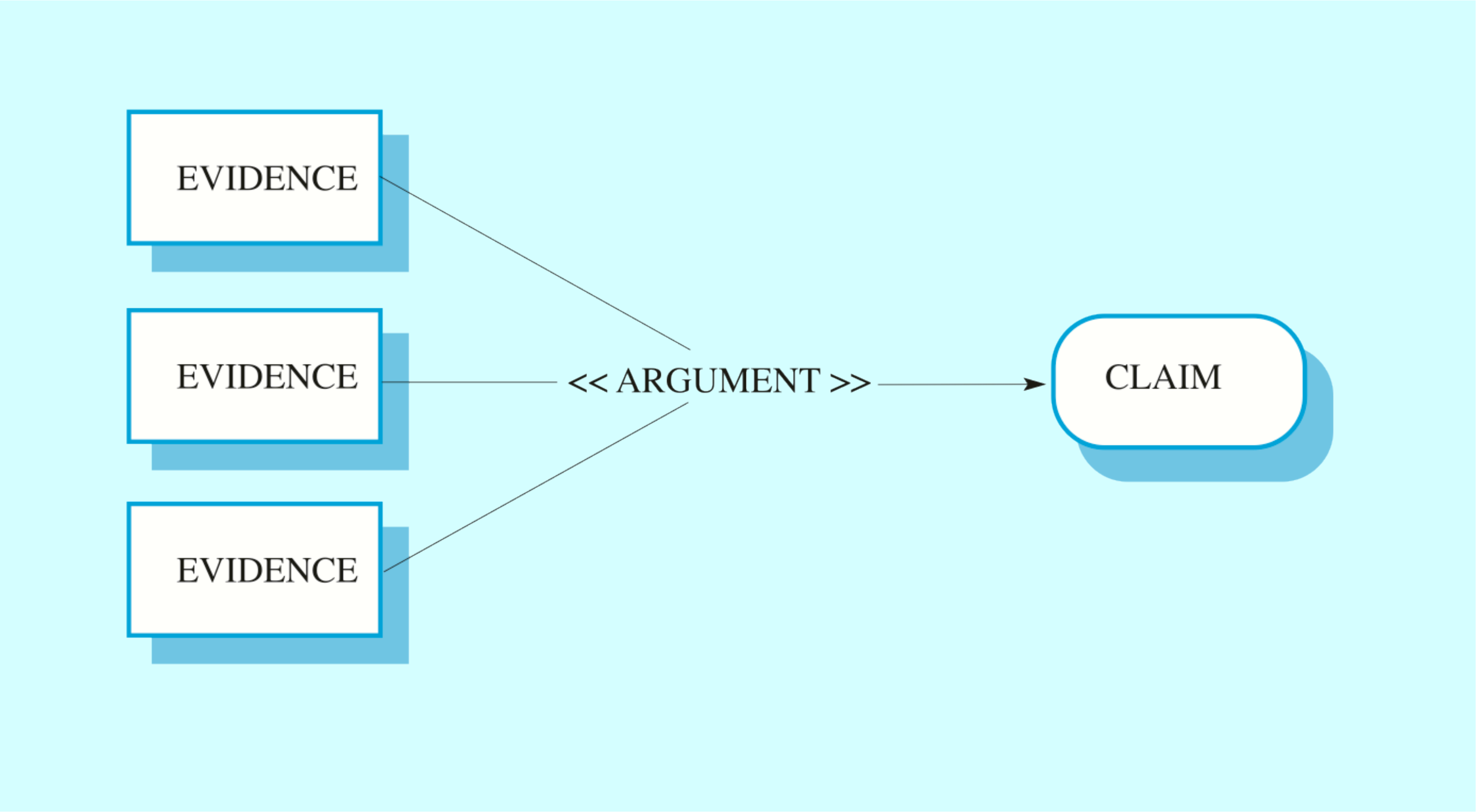
The system safety case

- It is now normal practice for a formal safety case to be required for all safety-critical computer-based systems e.g. railway signalling, air traffic control, etc.
- A safety case is:
  - A documented body of evidence that provides a convincing and valid argument that a system is adequately safe for a given application in a given environment.
- Arguments in a safety or dependability case can be based on formal proof, design rationale, safety proofs, test results, etc. Process factors may also be included.

Components of a safety case

Component	Description
System description	An overview of the system and a description of its critical components.
Safety requirements	The safety requirements abstracted from the system requirements specification.
Hazard and risk analysis	Documents describing the hazards and risks that have been identified and the measures taken to reduce risk.
Design analysis	A set of structured arguments that justify why the design is safe.
Verification and validation	A description of the V & V procedures used and, where appropriate, the test plans for the system. Results of system V &V.
Review reports	Records of all design and safety reviews.
Team competences	Evidence of the competence of all of the team involved in safety-related systems development and validation.
Process QA	Records of the quality assurance processes carried out during system development.
Change management processes	Records of all changes proposed, actions taken and, where appropriate, justification of the safety of these changes.
Associated safety cases	References to other safety cases that may impact on this safety case.

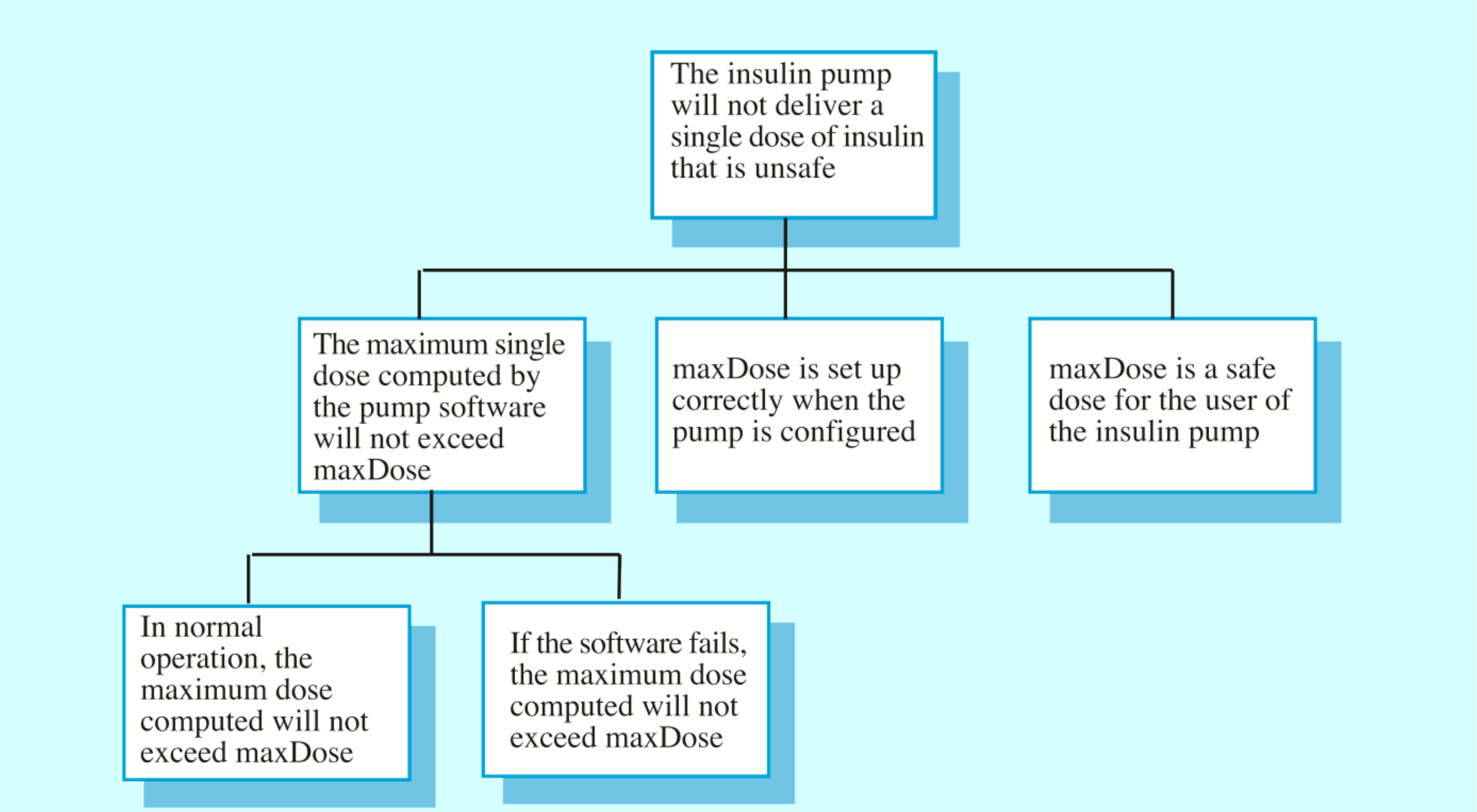
Argument structure



Insulin pump argument

Claim: The maximum single dose computed by the insulin pump will not exceed maxDose.
Evidence: Safety argument for insulin pump as shown in next slide
Evidence: Test data sets for insulin pump
Evidence: Static analysis report for insulin pump software
Argument: The safety argument presented shows that the maximum dose of insulin that can be computed is equal to maxDose.
In 400 tests, the value of Dose was correctly computed and never exceeded maxDose.
The static analysis of the control software revealed no anomalies.
Overall, it is reasonable to assume that the claim is justified.

Claim hierarchy



The insulin pump will not deliver a single dose of insulin that is unsafe

The maximum single dose computed by the pump software will not exceed maxDose

maxDose is set up correctly when the pump is configured

maxDose is a safe dose for the user of the insulin pump

In normal operation, the maximum dose computed will not exceed maxDose

If the software fails, the maximum dose computed will not exceed maxDose

Formal Methods and Safety

- Formal methods are mandated in Britain for the development of some types of safety-critical software
- Formal specification and correctness proofs increases confidence that a system meets its specification
- Formal specifications require specialized notations so domain experts cannot check for specification incompleteness (which may lead to unsafe behaviors)
- The cost-effectiveness of formal methods is unknown
- Use of formal methods for safety-critical software development is likely to increase

Safe Design Principles

- Separate critical software from the rest & make critical software as simple as possible (possibly at the expense of performance)
- Use simple techniques for software development avoiding error-prone constructs such as pointers and recursion
- Use information hiding to localize the effect of any data corruption
- Make appropriate use of fault-tolerant techniques but do not be seduced into thinking that fault-tolerant software is necessarily safe

Safety Proofs

- Safety proofs are intended to show that the system cannot reach an unsafe state
- Weaker than correctness proofs which must show that the system code conforms to its specification
- Generally based on proof by contradiction
  - Assume that an unsafe state can be reached
  - Show that this is contradicted by the program code
- May be displayed graphically

Construction of a safety proof

- Establish the safe exit conditions for a program
- Starting from the END of the code, work backwards until you have identified all paths that lead to the exit of the code
- Assume that the safe exit condition is false
- Show that, for each path leading to the exit that the assignments made in that path contradict the assumption of an unsafe exit from the program

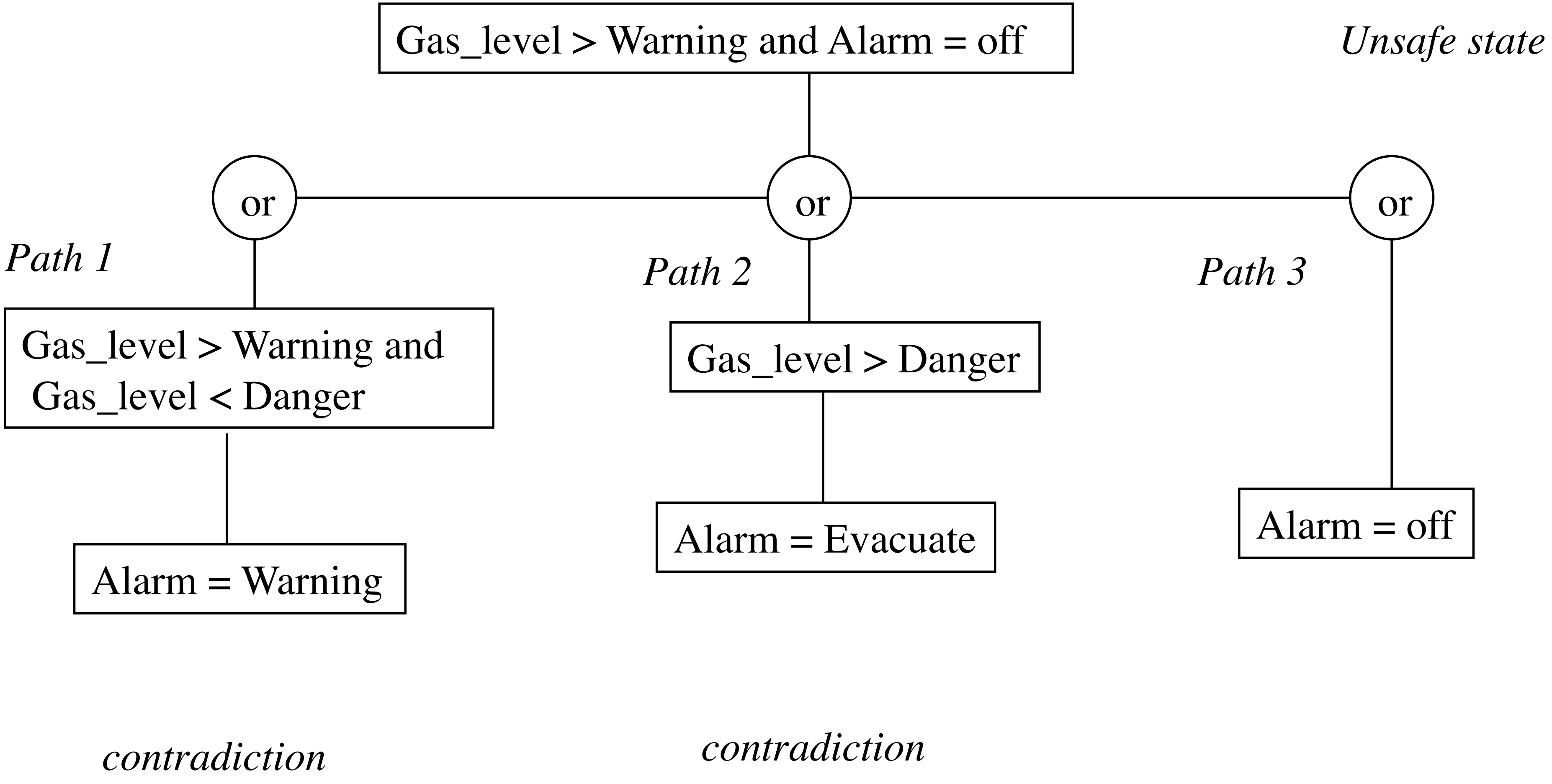
Example: Gas warning system

- System to warn of poisonous gas. Consists of a sensor, a controller and an alarm
- Two levels of gas are hazardous
  - Warning level - no immediate danger but take action to reduce level
  - Evacuate level - immediate danger. Evacuate the area
- The controller takes air samples, computes the gas level and then decides whether or not the alarm should be activated

Gas sensor control

```
Gas_level: GL_TYPE ;
loop
    -- Take 100 samples of air
    Gas_level := 0.000 ;
    for i in 1..100 loop
        Gas_level := Gas_level + Gas_sensor.Read ;
    end loop ;
    Gas_level := Gas_level / 100 ;
    if Gas_level > Warning and Gas_level < Danger then
        Alarm := Warning ;   Wait_for_reset ;
    elsif Gas_level > Danger then
        Alarm := Evacuate ;   Wait_for_reset ;
    else
        Alarm := off ;
    end if ;
end loop ;
```

Graphical argument



Condition checking

Gas_level < Warning	Path 3	Alarm = off (Contradiction)
Gas_level = Warning	Path 3	Alarm = off (Contradiction)
Gas_level > Warning and Gas_level < Danger	Path 1	Alarm = Warning (Contradiction)
Gas_level = Danger	Path 3	Alarm = off
Gas_level > Danger	Path 2	Alarm = Evacuate (Contradiction)

Code is incorrect.

Gas_level = Danger does not cause the alarm to be on

Key points

- Safety-related systems should be developed to be as simple as possible using ‘safe’ development techniques
- Safety assurance may depend on ‘trusted’ development processes and specific development techniques such as the use of formal/rigorous methods and safety proofs
- Safety proofs are easier than proofs of consistency or correctness. They must demonstrate that the system cannot reach an unsafe state. Usually proofs by contradiction

Validating the safety of the insulin pump system

Insulin delivery system

- Safe state is a shutdown state where no insulin is delivered
  - If hazard arises,shutting down the system will prevent an accident
- Software may be included to detect and prevent hazards such as power failure
- Consider only hazards arising from software failure
  - Arithmetic error The insulin dose is computed incorrectly because of some failure of the computer arithmetic
  - Algorithmic error The dose computation algorithm is incorrect

Arithmetic errors

- Use language exception handling mechanisms to trap errors as they arise
- Use explicit error checks for all errors which are identified
- Avoid error-prone arithmetic operations (multiply and divide). Replace with add and subtract
- Never use floating-point numbers
- Shut down system if exception detected (safe state)

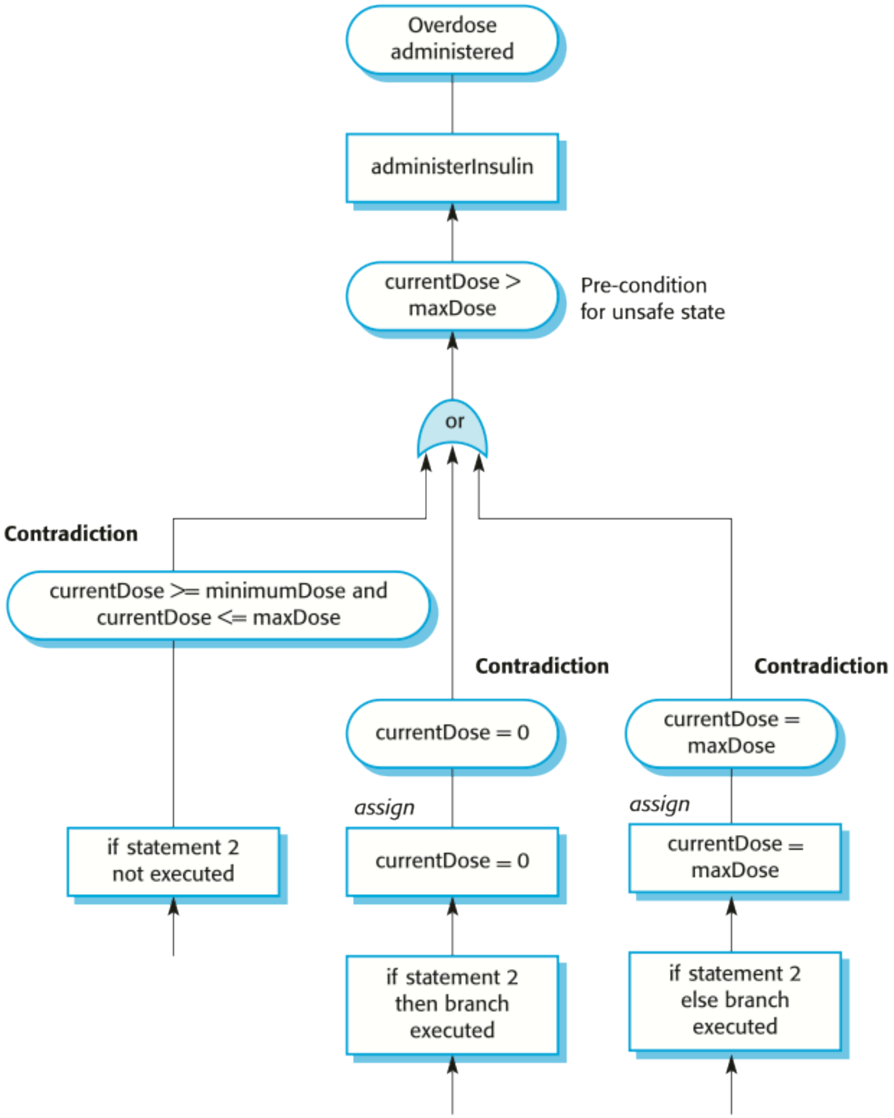
Algorithmic errors

- Harder to detect than arithmetic errors. System should always err on the side of safety
- Avoid recursion, pointers, dynamic memory allocation
- Use reasonableness checks for the dose delivered based on previous dose and rate of dose change
- Set maximum delivery level in any specified time period
- If computed dose is very high, medical intervention may be necessary anyway because the patient may be ill

Insulin delivery code

```
// The insulin dose to be delivered is a function of  blood sugar level, the previous dose
// delivered and the time of delivery of the previous dose
    currentDose = computeInsulin () ;
    // Safety check - adjust currentDose if necessary
    if (previousDose == 0)                                      // if statement 1
    {
        if (currentDose > 16)
            currentDose = 16 ;
    }
    else
        if (currentDose > (previousDose * 2) )
            currentDose = previousDose * 2 ;
    if ( currentDose < minimumDose )                            // if statement 2
            currentDose = 0 ;                                   // then branch
    else if ( currentDose > maxDose )                           // else branch
            currentDose = maxDose ;
    administerInsulin (currentDose) ;
```

Safety ‘Proofs’



Sommerville

System testing

- System testing of the software has to rely on simulators for the sensor and the insulin delivery components.
- Test for normal operation using an operational profile. Can be constructed using data gathered from existing diabetics
- Testing has to include situations where rate of change of glucose is very fast and very slow
- Test for exceptions using the simulator

Safety assertions

- Similar to defensive programming
- Predicates included in the program indicating conditions which should hold at that point
- May be based on pre-computed limits e.g. number of insulin pump increments in maximum dose
- Used to check safety constraints at run time and may throw safety–related exceptions
- Assertions should be generated from safety specifications

Safety assertions

```
static void administerInsulin ( ) throws SafetyException
    {
        int maxIncrements = InsulinPump.maxDose / 8 ;
        int increments = InsulinPump.currentDose / 8 ;
        // assert currentDose <= InsulinPump.maxDose
        if (InsulinPump.currentDose > InsulinPump.maxDose)
           throw new SafetyException (Pump.doseHigh);
        else
           for (int i=1; i<= increments; i++)
           {
               generateSignal () ;
               if (i > maxIncrements)
                   throw new SafetyException ( Pump.incorrectIncrements);
           } // for loop
    } //administerInsulin
```

Conclusions

- Safety is a system property regarding how it interacts with its environment
- Hazard analysis is a key part of the safety specification process – it can be supported by fault tree analysis
- Risk analysis involves assessing the probability of hazards, their severity and the probability that they will result in an accident
- Design strategies may be used for hazard avoidance, hazard probability reduction and accident avoidance
- Safety arguments should be used as part of product safety assurance.
- Safety arguments are a way of demonstrating that a hazardous condition can never occur.
- Safety cases collect together the evidence that a system is safe.