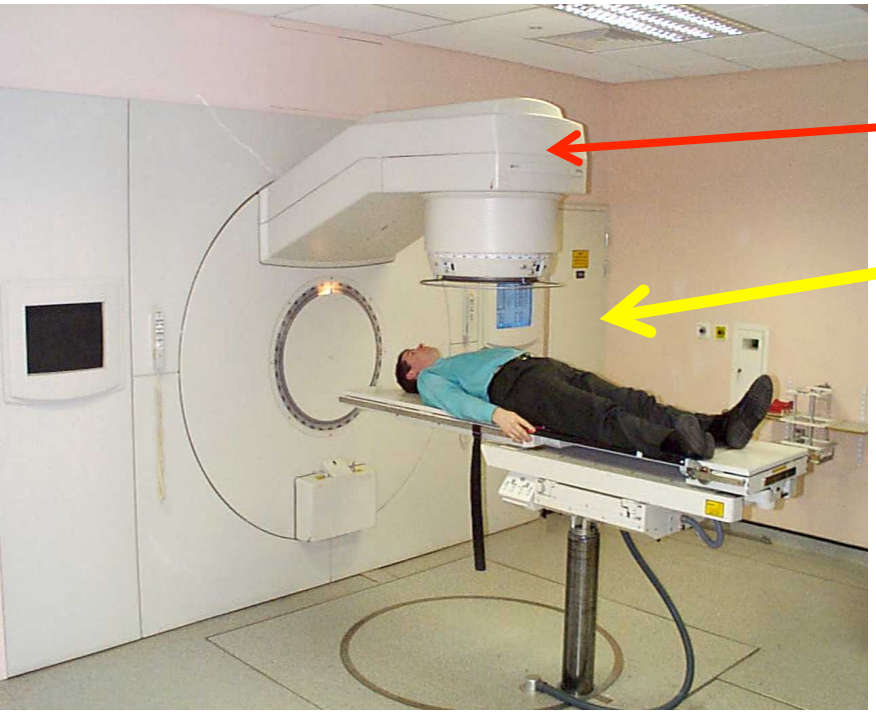
Faculty of ECMS / School of Computer Science

Software Engineering & Project
Formal Specification

Introduction to Formal Specifications

Lecture 15

Formal Methods?



The radiation beam shall not turn on unless the therapy room door is closed

Can be represented precisely and concisely with:

$$\text{Beam=on} \Rightarrow \text{Door=closed}$$

Overview

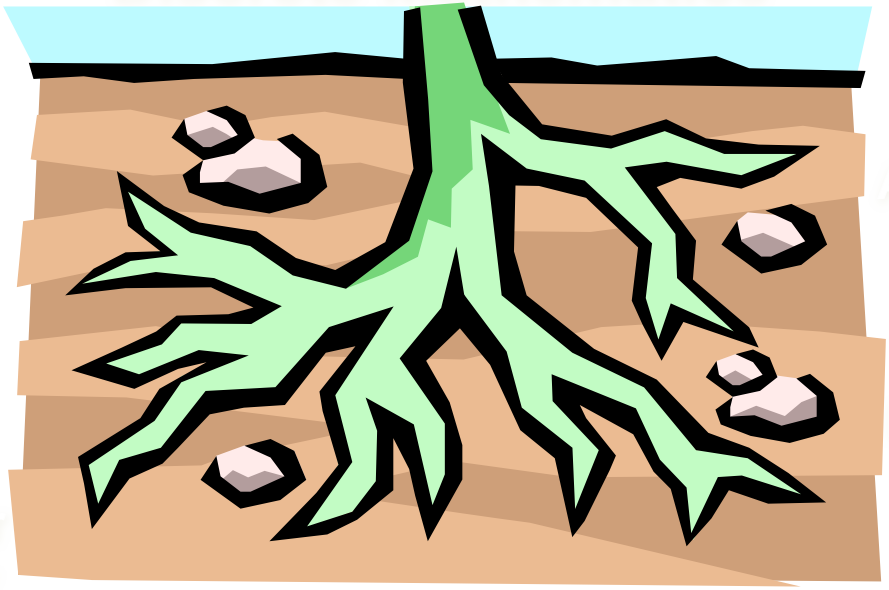
- What are formal methods?
- Formal specification
- The Z specification language
- An example: Specification of Insulin Delivery System

Concepts

- What are Formal Methods?
  - Formal methods is that area of computer science that is concerned with the application of mathematical techniques to the design and implementation of computer hardware and software.
- Diller (1988) suggests there are two main parts to formal methods:
  - Formal specification.
    - Using mathematics to specify the desired properties of a computer system unambiguously.
    - Allows automatic checks for completeness and consistency
  - Formal verification.
    - Use mathematics to prove that a computer system satisfies its specification.
- Other aspects of formal methods
  - Allow us to prove equivalence
  - Provide a step-wise refinement mechanism to incrementally transform specification to implementation

Introduction (2)

Formal methods deeply rooted in:



These roots provide a solid and defensible foundation for developing and verifying computer programs

Uptake of Formal Methods

- In the 1980's, it was predicted that a large proportion of software would be developed using FM by the 21st century.
- This hasn't happened because:
  - Improvements to Software engineering processes and methodologies.
  - Market changes:
    - Quality isn't always an overridding consideration: time-to-market.
  - Limited scope of FM
    - eg. unsuitable for specification of user interfaces and interactions
  - Limited scalability.

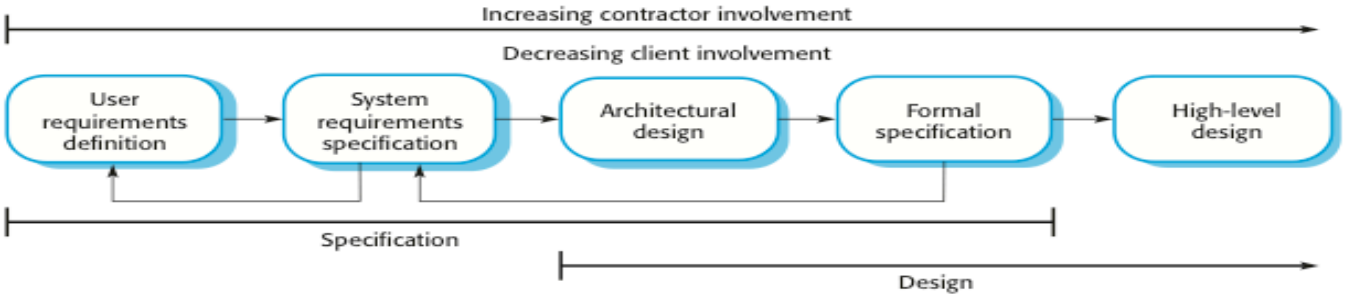
Introduction (3)

Importance of specification

- Many software errors can be traced to imprecision in the early stages of the project
  - Often due to lack of detail or imprecise wording during specification stage
  - Very costly to detect and correct such errors late in lifecycle
- Modern engineering practices and experiences reveal that for most of complicated, and safety critical software projects, it is needed to focus more attention on early stages of lifecycle, in particular on requirements engineering (including development of system specification)

Formal Specification

Figure 12.12 Formal specification in a plan-based software process



Advantages of Formal Specification

- Development of FS facilitates deeper understanding of system requirements.
- Specification is expressed in a language with formally defined semantics:
  - no ambiguity
  - can be automatically analysed to find inconsistencies and incompleteness.
- Algebraic transformation
- Possible reduced testing costs

Some disadvantages

- Most customers (and domain experts) can't understand FS.
  - Conversely, many software engineers won't have a deep understanding of the problem domain
- Hard to quantify cost savings yielded by using FS
- Most software engineers aren't trained to use FS
- Difficulty scaling current FS methodologies to large systems
- FS not compatible with agile development methods.

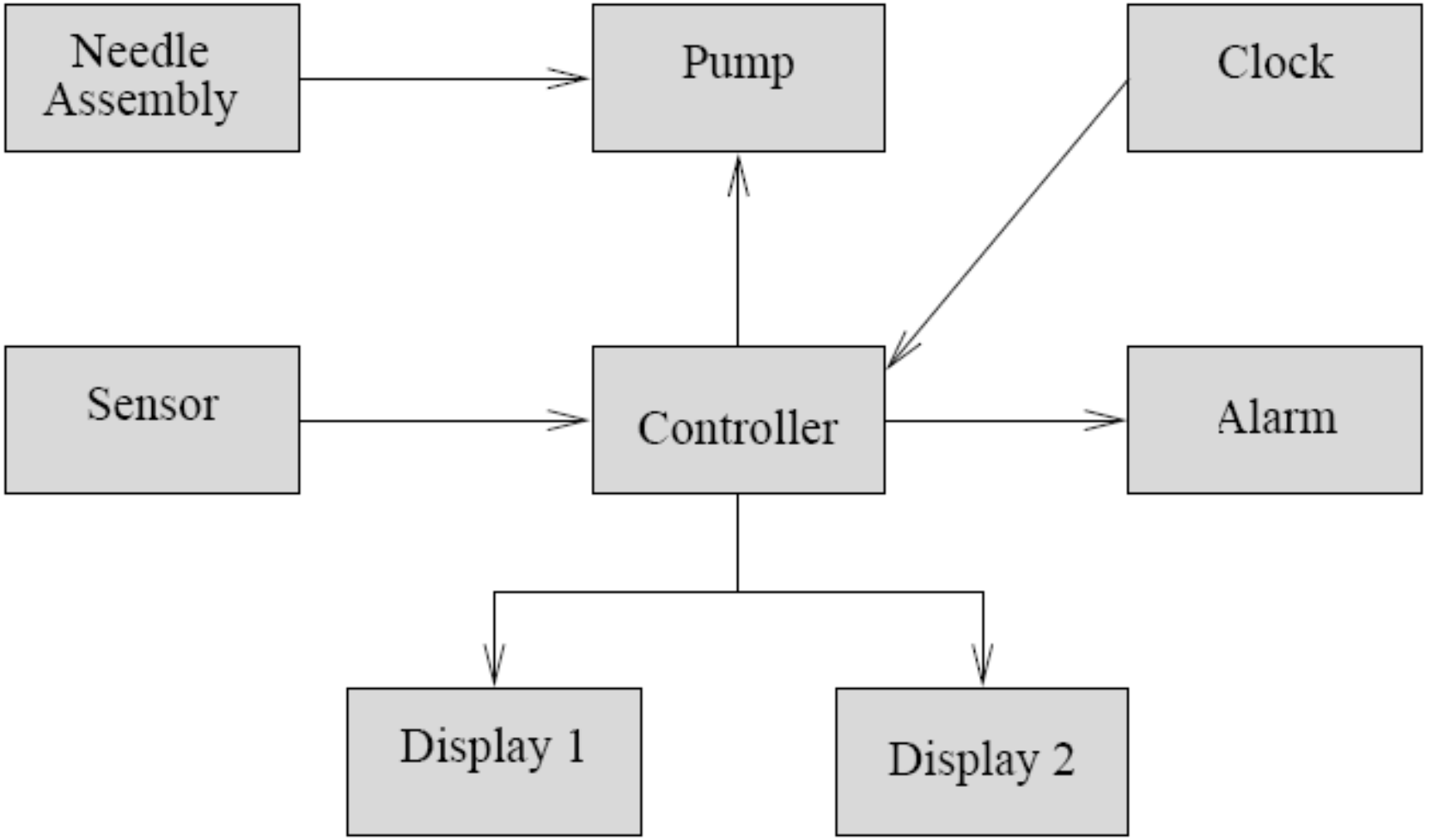
Formal Specification Languages

- Precise vocabulary, syntax and set of semantics for description of system
- Focus on what is to be solved, not how
- Based on formal mathematical logic
  - State-based
    - VDM (Vienna Development Method)
    - Z
    - Produce a model of the system based on mathematical entities
  - Algebraic: OBJ
  - Process algebras: CSP (Communicating Sequential Processes (**C.A.R. Hoare**))

Z Specification Language

- Pronounced “zed”
- Developed at the University of Oxford
- ISO/IEC standard 13568:2002
- Specifications mix formal statements and informal explanatory text
- Based on typed set theory and predicate calculus
- Uses schemas to link related objects

An Example Of Using Z Specification (1)



Insulin Delivery System (2)

Informal specification:

- Functional Requirements
  - The system shall take regular readings of blood sugar level (BSL)
  - The system shall be switched between off, manual and automatic modes
  - In automatic mode, the system shall regularly check the blood sugar levels to determine whether a dose of insulin needs to be administered
  - If insulin is required the system shall calculate a dose
  - Before administering a dose the system shall check that certain safety properties are not violated and that there is sufficient insulin available in the reservoir
- Safety requirements
  - SR1: The system shall not deliver a single dose of insulin that is greater than a specified maximum dose for a single user
  - SR2: The system shall not deliver a daily cumulative dose of insulin that is greater than a specified maximum for a system user

Insulin Delivery System (3)

Variable type definitions

Blood sugar level reading : READING == 0. .20

Insulin doses: INSULIN == 0. .100

the system shall be switched off, manually operated or run in automatically

Switch ::= off | manual | auto

All items declared are constants and globally used

Declaration

$$max\_daily\_dose, max\_single\_dose : \mathbb{N}$$
$$capacity : \mathbb{N}$$
$$minimum\_dose : \mathbb{N}$$
$$safemin, safemax : \mathbb{N}$$

- An axiomatic definition

Predicate

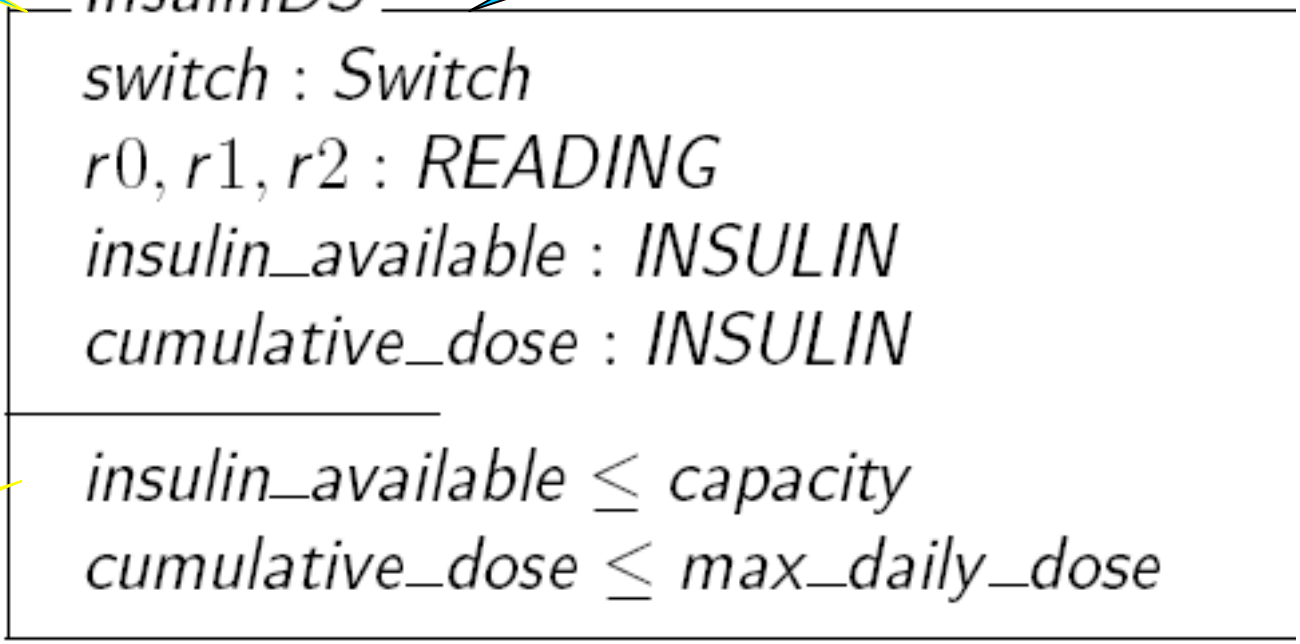
$$capacity = 100$$
$$max\_daily\_dose = 25$$
$$max\_single\_dose = 4$$
$$minimum\_dose = 1$$
$$safemin = 6 \land safemax = 14$$

Insulin Delivery System (4)

A Schema
- defines state

All items declared are variables – they determine the state

State schema



Invariant

Insulin Delivery System (5)

State variables

- switch
  - records the current position of switch
  - can be off, manual or auto
- r0, r1, r2
  - represent the last three readings of the BSL (Blood Sugar Level)
- insulin available
  - represents how much insulin is in the reservoir
- cumulative dose
  - how much insulin has been administered since midnight

Insulin Delivery System (6)

State invariant

- Defines properties of the state that should always hold
- The amount of insulin available can never exceed the capacity of the reservoir

$$insulin\_available \leq capacity$$

- The cumulative dose should never exceed the maximum daily dose

$$cumulative\_dose \leq max\_daily\_dose$$

Initialising the state

Initialise

$\Delta InsulinDS$

$r0' = 0$

$r1' = 0$

$r2' = 0$

$cumulative\_dose' = 0$

$switch' = auto$

$insulin\_available' = capacity$

Insulin Delivery System (7)

Operational schema

UpdateBSLReading

$\Delta InsulinDS$

$reading? : \mathbb{N}$

$r2' = reading?$

$r1' = r2$

$r0' = r1$

$cumulative\_dose' = cumulative\_dose$

$switch' = switch$

$insulin\_available' = insulin\_available$

Insulin Delivery System (8)

Calculating dose

To calculate the dose when the BSL is in the safe zone, we consider three different scenarios

Decreasing BSL - No insulin delivery is required

Increasing BSL, the rate of increase is decreasing - No insulin delivery is required

Increasing BSL, at a steady or increasing rate – insulin is required

Calculating dose

CalculateDose

$\Xi InsulinDS$

$dose! : INSULIN$

$r2 \geq safemin \land r2 \leq safemax$

$(r2 \leq r1 \land dose! = 0) \lor$
$(r2 > r1 \land (r2 - r1) < (r1 - r0) \land dose! = 0) \lor$
$(r2 > r1 \land (r2 - r1) \geq (r1 - r0) \land (r2 - r1) \operatorname{div} 4 = 0$
$\land dose! = minimum\_dose) \lor$
$(r2 > r1 \land (r2 - r1) \geq (r1 - r0) \land (r2 - r1) \operatorname{div} 4 > 0$
$\land dose! = (\textbf{if } (r2 - r1) \bmod 4 < 2$
$\textbf{then } (r2 - r1) \operatorname{div} 4 \textbf{ else } ((r2 - r1) \operatorname{div} 4) + 1))$

Insulin Delivery System (9)

Delivering a dose

DeliverFullDose

$\Delta InsulinDS$

$computed\_dose? : INSULIN$

$actual\_dose! : INSULIN$

$switch = auto$

$insulin\_available \geq computed\_dose?$

$computed\_dose? + cumulative\_dose \leq max\_daily\_dose$

$computed\_dose? \leq max\_single\_dose$

$actual\_dose! = computed\_dose?$

$insulin\_available' = insulin\_available - actual\_dose!$

$cumulative\_dose' = cumulative\_dose + actual\_dose!$

$switch' = switch$

$r2' = r2 \land r1' = r1 \land r0' = r0$

Other features of Z

Is that all of Z?

- Not by a long shot!
- But it is a good overview of the main features
  - Z is based on typed set theory
  - It includes types and operations for sets, relations, functions, bags, sequences, naturals, integers, ...
  - Allows users to define their own types and operations based on existing constructs

Therefore it is a rich abstract language for specifying state-based systems

Checking the specification

- Syntax and type checking
- Specification animation
- Verification
- Completeness checks

Syntax and type checking

- Check that all operators and types have been defined
- Check that all operators and types are being used correctly
  - Type checking tools are available that do this automatically
  - e.g. fuzz, HIPPO, ZEST,...

Animation

- Test the correctness of a specification by “running” it
  - Feed in test cases for various scenarios
  - Inspect the state of the system to ensure expected behavior matches actual behavior
  - A number of tools are available that support specification animation
    - e.g. Possum, ZETA, PiZA, Jaza,...

Verification

- Verifying (safety) requirements
  - e.g. check that any dose output from the Deliver Dose schema does not exceed the maximum single dosage
- Checking the state invariant
- Check that all operations maintain the state invariant
- Tools can be used to generate verification conditions
- These conditions can either be checked by hand or machine checked using a theorem prover tool

Completeness checks

- Check the specification to make sure the operations handle all valid states of the system
  - e.g. what happens when we need to calculate a dose for BSL outside of safe boundaries?
    - need to define operations that define the dose for low BSL and high BSL
      - e.g. when delivering dose, what happens if the computed dose exceeds the maximum single dose or the maximum daily dose is exceeded

Summary

- Many software errors can be traced to imprecision in specification
- Symptom of using informal specification languages
- Formal languages can overcome many of these problems
- Tools are available for checking formal specifications
  - BUT ...
    - formal specification is expensive
    - requires specialized skills
- THEREFORE ... typically only utilized for highly critical software components