

Software Engineering

FORMAL METHODS

Introduction

- All design and analysis methods discussed so far have been informal and lack mathematical rigor.
- Now, we consider specification and designs using formal syntax and semantics that specify function and behavior.
- We use predicate calculus...

Introduction

- Advocates claim it will revolutionize software engineering.
- Detractors claim it is impossibly difficult.
- We use set theory to characterize requirements unambiguously.
- We analyze the mathematical statements for correctness, look for inconsistencies, and prove theorems.

Introduction

- Where is this used?
 - Mission and safety critical systems.
 - Government.
 - Dod.
 - Medical devices.
 - Economic engines.
- Consequences?
 - Errors uncovered during specification phase before coding begins.

Introduction

- Formal languages?
 - Z
 - UML + OCL
- How do you know if what you have done is correct?
 - Use mathematical proofs to show correctness.
 - Use proof engines to aid.
 - Other engines find counter-examples to invalidate statements in your predicate calculus statements.

Introduction

- Extracting requirements using informal methods:
 - Ambiguous
 - Contradictions
 - Incompleteness
 - Multiple interpretations (language, cultural,...)
 - Etc.
- Extracting requirements using formal methods:
 - Proof engines will find any ambiguities, inconsistencies or contradictions.
 - Much harder to use.

Mathematical Preliminaries

- Sets

- Elements contained in a set are unique. No duplicates!
- Order is immaterial.
- The number of elements is called its cardinality. The operator '#' returns the cardinality of a set.

- Ex.

- {Sacajawea, Ross, Boldy, Saddle} // A set
- $\#\{\text{Sacajawea, Ross, Boldy, Saddle}\} = 4$

Mathematical Preliminaries

- We can also define a set using constructive set specification. This is much better if sets are large!
 - Ex. $\{n : \mathbb{N} \mid n < 3 \cdot n\}$
 - 3 parts : A signature, a predicate, and a term.
 - The signature represents the range of values to be considered. In this case the Natural numbers.
 - The predicate (boolean) is the constraint.
 - The term gives the general form of the elements in the set. This can be omitted if obvious.
 - $\{0, 1, 2\}$

Mathematical Preliminaries

- Sets do not have to be made up of single elements. You can have pairs, triples, etc.
 - Ex. $\{x, y : \mathbb{N} \mid x + y = 10 \cdot (x, y^2)\}$
 - Describes the sets of pairs of natural numbers that have the form (x, y^2) and where the sum of x and y is 10.
 - $\{(1, 81), (2, 64), (3, 49), \dots\}$
- Set Operators
 - The ' \in ' operator indicates set membership.
 - Ex. $x \in X$ is true if x is a member of set X .
 - Ex. $12 \in \{13, 12, 33\}$ is true.
 - The opposite of \in is \notin

Mathematical Preliminaries

- The operators \subset and \subseteq take sets as operands.
 - Ex. $A \subset B$ is true if all the members of set A are contained in set B.
 - $\{1, 2\} \subset \{4, 3, 2, 1\}$ is true.
 - $\{1, 2\} \subseteq \{1, 2\}$ is true, but $\{1, 2\} \subset \{1, 2\}$ is false.
- The empty set is denoted by \emptyset
 - $\emptyset \cup A = A$ and $\emptyset \cap A = \emptyset$; where these are the Union and Intersection operators.
- Cartesian product is denoted by 'x'.

Mathematical Preliminaries

- Cartesian product has 2 operands. The result is a set of pairs where each pair consists of an element taken from the first operand combined with an element from the second.
 - Ex. $\{1, 2\} \times \{4, 5, 6\} = \{(1, 4), (1, 5), (1, 6), (2, 4), (2, 5), (2, 6)\}$
- The powerset of a set is the collection of subsets of that set. It is denoted by $|P$.
 - Ex. $|P\{1, 2, 3\} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$

Mathematical Preliminaries

- Logic
 - \wedge and
 - \vee or
 - \neg not
 - \Rightarrow implies
 - You can look up the truth tables for each of these.
- Universal Quantification
 - This is a way of making statements about the elements of a set. It uses the symbol \forall

Mathematical Preliminaries

- Ex. $\forall i, j: \mathbf{N} \cdot i > j \Rightarrow i^2 > j^2$
 - States that for every pair of values in the set of natural numbers, if i is greater than j then i^2 is greater than j^2
- Existential Quantification uses the symbol \exists .
 - $\exists i, j: \mathbf{N} \cdot i > j \Rightarrow i^2 > j^2$
 - States that there exists a pair of values in the set of natural numbers, such that if i is greater than j then i^2 is greater than j^2 .

Mathematical Preliminaries

- Sequences

- This is a mathematical structure that models the fact that its elements are ordered. A sequence is a set of pairs whose elements range from 1 to the highest number element.
- Ex. $\{(1, \text{Jones}), (2, \text{Wilson}), (3, \text{Pete})\}$ is a sequence
- The items that form the first elements of the pairs are known as the domain of the sequence. The collection of second elements is known as the range of the sequence.

Mathematical Preliminaries

■ Sequences

- The preceding sequence is written as:
 - $\langle \text{Jones, Wilson, Pete} \rangle$
- Unlike sets, duplication is allowed and order is important.
- Ex. $\langle 1, 4, 7 \rangle$ is not equal to $\langle 1, 7, 4 \rangle$

■ Sequence Operators

- Catenation : \frown
- Ex. $\langle 23, 29, \rangle \frown \langle 23, 1, 5 \rangle = \langle 23, 29, 23, 1, 5 \rangle$

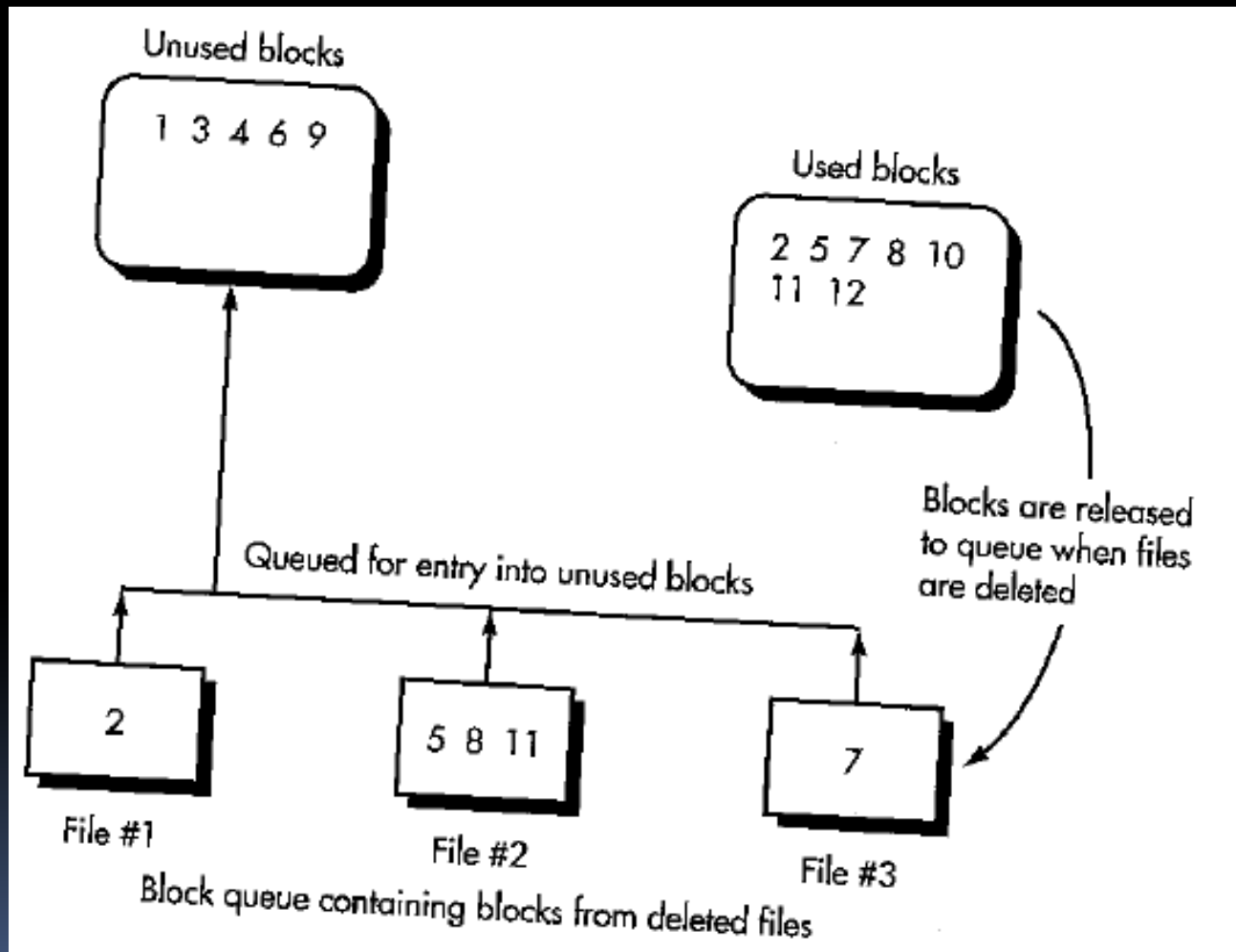
Mathematical Preliminaries

- Sequence Operators
 - $\text{head } \langle 4, 6, 8, 9 \rangle = 4$
 - $\text{tail } \langle 4, 6, 8, 9 \rangle = \langle 6, 8, 9 \rangle$
 - $\text{last } \langle 4, 6, 8, 9 \rangle = 9$
 - $\text{front } \langle 4, 6, 8, 9 \rangle = \langle 4, 6, 8 \rangle$
- Since a sequence is a set of pairs, then all set operators can be used on sequences.

- Ex. $\text{FileList} : \text{seq FILES}$
 $\text{NoUsers} : \mathbb{N}$

This state has 2 components; a sequence of files and a natural number.

Example



Example

Example 2: A Block Handler. One of the more important parts of a computer's operating system is the subsystem that maintains files created by users. Part of the filing subsystem is the *block handler*. Files in the file store are composed of blocks of storage that are held on a file storage device. During the operation of the computer, files will be created and deleted, requiring the acquisition and release of blocks of storage. In order to cope with this, the filing subsystem will maintain a reservoir of unused (free) blocks and keep track of blocks that are currently in use. When blocks are released from a deleted file they are normally added to a queue of blocks waiting to be added to the reservoir of unused blocks. This is shown in Figure 25.2. In this figure, a number of components are shown: the reservoir of unused blocks, the blocks that currently make up the files administered by the operating system, and those blocks that are waiting to be added to the reservoir. The waiting blocks are held in a queue, with each element of the queue containing a set of blocks from a deleted file.

For this subsystem the state is the collection of free blocks, the collection of used blocks, and the queue of returned blocks. The data invariant, expressed in natural language, is

- No block will be marked as both unused and used.
- All the sets of blocks held in the queue will be subsets of the collection of currently used blocks.
- No elements of the queue will contain the same block numbers.
- The collection of used blocks and blocks that are unused will be the total collection of blocks that make up files.

Example

- The collection of unused blocks will have no duplicate block numbers.
- The collection of used blocks will have no duplicate block numbers.

Some of the operations associated with these data are

- An operation that adds a collection of blocks to the end of the queue.
- An operation that removes a collection of used blocks from the front of the queue and places them in the collection of unused blocks.
- An operation that checks whether the queue of blocks is empty.

The precondition of the first operation is that the blocks to be added must be in the collection of used blocks. The postcondition is that the collection of blocks must be added to the end of the queue.

The precondition of the second operation is that the queue must have at least one item in it. The postcondition is that the blocks must be added to the collection of unused blocks.

The final operation—checking whether the queue of returned blocks is empty—has no precondition. This means that the operation is always defined, regardless of what value the state has. The postcondition delivers the value *true* if the queue is empty and *false* otherwise.

Example

- A set named *BLOCKS* will consist of every block number.
- *AllBlocks* is a set of blocks that lie between 1 and *MaxBlocks*.
- The state will be modeled by 2 sets and a sequence. The 2 sets are *used* and *free*. Both contain blocks. The *used* set contains the blocks that are currently used in files and the *free* set contains blocks that are available for new files.
- The sequence will contain sets of blocks that are ready to be released from files that have been deleted. This is our queue.

Example

- Describing the state:

used, free : |P *BLOCKS*

BlockQueue : seq |P *BLOCKS*

- *used* and *free* are sets of *BLOCKS*.
- *BlockQueue* is a sequence. Each element is a set of *BLOCKS*.

Example

- Data invariants

$$used \cap free = \emptyset \quad \wedge$$

$$used \cup free = AllBlocks \quad \wedge$$

$$\forall i: \text{dom } BlockQueue \cdot BlockQueue_i \subseteq \underline{used} \quad \wedge$$

$$\forall i, j: \text{dom } BlockQueue \cdot i \neq j \Rightarrow$$

$$BlockQueue_i \cap BlockQueue_j = \emptyset$$

Example

- Operations

- Remove an element from the head of the block queue.
 - The precondition is that there must be at least one item in the queue.

$\#BlockQueue > 0$

- The postcondition is that the head of the queue must be removed and placed in the collection of free blocks and the queue adjusted to show removal.

$used' = used \setminus head\ BlockQueue \wedge$

$free' = free \cup head\ BlockQueue \wedge$

$BlockQueue' = tail\ BlockQueue$

Example

- Operations

- Adds a collection of blocks, *Ablocks* to the block queue.
 - The precondition is that *Ablocks* is currently a set of used blocks.

$$Ablocks \subseteq \underline{used}$$

- The postcondition is that the set of blocks is added to the end of the block queue and the set of used and free blocks remains unchanged.

$$\begin{aligned} BlockQueue' &= BlockQueue \frown \langle Ablocks \rangle \wedge \\ used' &= used \wedge \\ free' &= free \end{aligned}$$