Software Engineering:
A Practitioner's Approach, 6/e
Roger Pressman

Chapter 28
Formal Methods

Problems with Conventional Specification

- contradictions
- ambiguities
- vagueness
- incompleteness
- mixed levels of abstraction

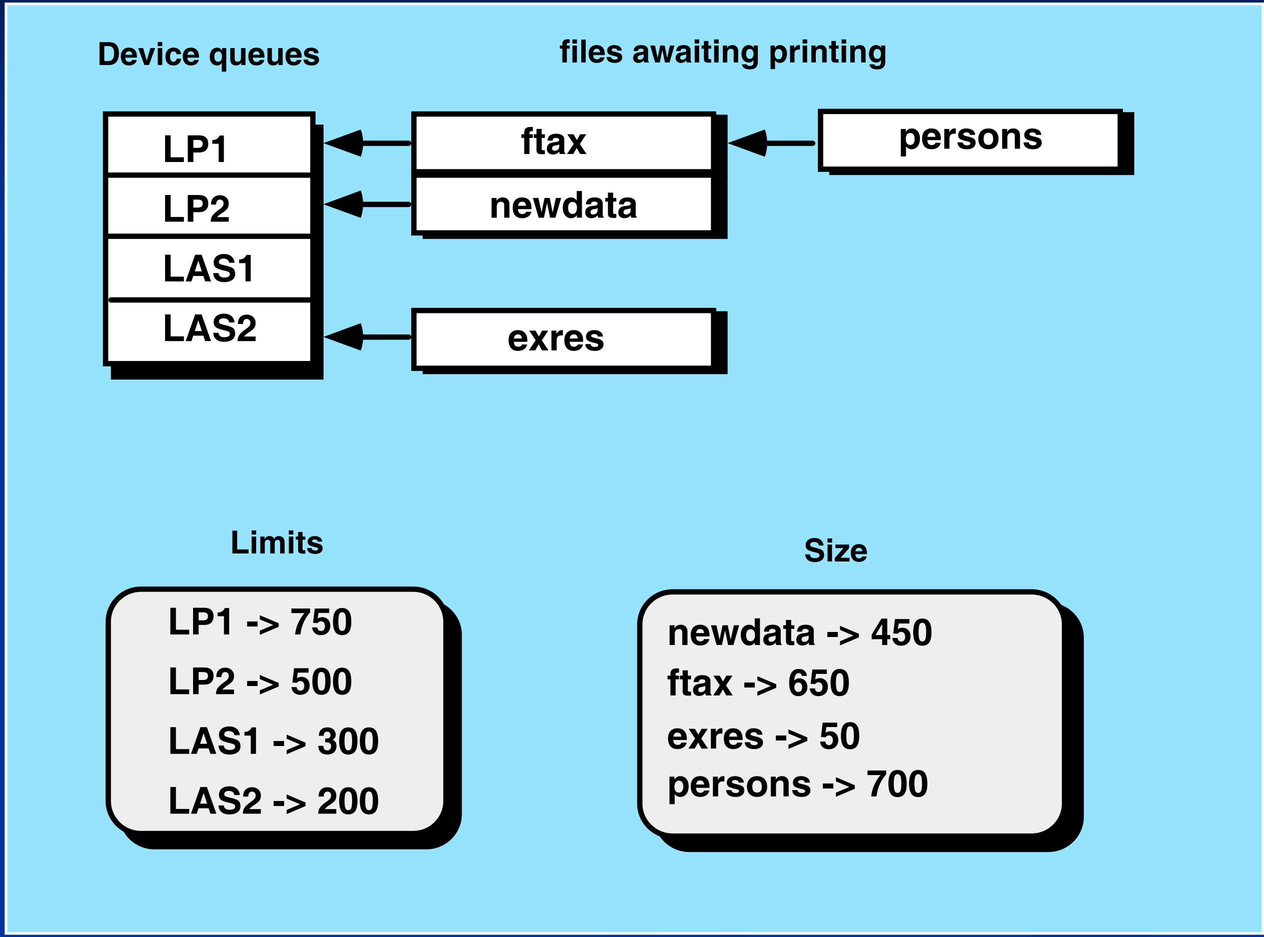
Formal Specification

- Desired properties - consistency, completeness, and lack of ambiguity - are the objectives of all specification methods
- The formal syntax of a specification language enables requirements or design to be interpreted in only one way, eliminating ambiguity that often occurs when a natural language (e.g., English) or a graphical notation must be interpreted
  - The descriptive facilities of set theory and logic notation enable clear statement of facts (requirements).
- Consistency is ensured by mathematically proving that initial facts can be formally mapped (using inference rules) into later statements within the specification.

Formal Methods Concepts

- *data invariant*—a condition that is true throughout the execution of the system that contains a collection of data
- *state*
  - Many formal languages, such as Object Constraint Language (OCL), use the notion of states, that is, a system can be in one of several states, each representing an externally observable mode of behavior.
  - The Z language defines a *state* as the stored data which a system accesses and alters
- *operation* - an action that takes place in a system and reads or writes data to a state
  - *precondition* defines the circumstances in which a particular operation is valid
  - *postcondition* defines what happens when an operation has completed its action

An Example—Print Spooler



States and Data Invariant

The state of the spooler is represented by the four components *Queues*, *OutputDevices*, *Limits*, and *Sizes*.

The data invariants are:

- Each output device is associated with an upper limit of print lines
- Each output device is associated with a possibly nonempty queue of files awaiting printing
- Each file is associated with a size
- Each queue associated with an output device contains files that each have a size less than the upper limit of the output device
- There will be no more than *MaxDevs* output devices administered by the spooler

Operations

- An operation which adds a new output device to the spooler together with its associated print limit
- An operation which removes a file from the queue associated with a particular output device
- An operation which adds a file to the queue associated with a particular output device
- An operation which alters the upper limit of print lines for a particular output device
- An operation which moves a file from a queue associated with an output device to another queue associated with a second output device

Pre- & Postconditions

For the first operation (adds a new output device to the spooler together with its associated print limit):

Precondition:

the output device name does not already exist and that there are currently less than *MaxDevs* output devices known to the spooler

Postcondition:

the name of the new device is added to the collection of existing device names, a new entry is formed for the device with no files being associated with its queue, and the device is associated with its print limit.

Mathematical Concepts

- sets and constructive set specification
- set operators
- logic operators
  - e.g., i, j: • $i > j$ i^2 => j^2
  - which 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.
- sequences

Sets and Constructive Specification

- A *set* is a collection of objects or elements and is used as a cornerstone of formal methods.
  - Enumeration
    - {C++, Pascal, Ada, COBOL, Java}
    - #{C++, Pascal, Ada, COBOL, Java} implies *cardinality* = 5
  - Constructive set specification is preferable to enumeration because it enables a succinct definition of large sets.
    - $\{x, y : \mathrm{N} \mid x + y = 10\}$

Set Operators

- A specialized set of symbols is used to represent set and logic operations.
  - Examples
    - The $\in$ operator is used to indicate membership of a set. For example, the expression
      - $x \in X$
    - The operators $\subset$, and # take sets as their operands. The predicate
      - $A \subset B$
      - has the value *true* if the members of the set A are contained in the set B and has the value *false* otherwise.
  - The union operator, $\cup$, takes two sets and forms a set that contains all the elements in the set with duplicates eliminated.
    - {File1, File2, Tax, Compiler} $\cup$ {NewTax, D2, D3, File2} is the set
    - {Filel, File2, Tax, Compiler, NewTax, D2, D3}

Logic Operators

- Another important component of a formal method is logic: the algebra of true and false expressions.
  - Examples:
    - $\vee$ or
    - $\neg$ not
    - => implies
- Universal quantification is a way of making a statement about the elements of a set that is true for every member of the set. Universal quantification uses the symbol, $\forall$. An example of its use is
  - $\forall\, i, j : \mathrm{N} \;\; i > j \Rightarrow i^2 > j^2$
  - which 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.

Sequences

- Sequences are designated using angle brackets. For example, the preceding sequence would normally be written as
  - [Jones, Wilson, Shapiro, Estavez]
- Catenation, ++, is a binary operator that forms a sequence constructed by adding its second operand to the end of its first operand. For example,
  - [2, 3, 34, 1] ++ [12, 33, 34, 200] = [2, 3, 34, 1, 12, 33, 34, 200]
- Other operators that can be applied to sequences are *head*, *tail*, *front*, and *last*.
  - *head* [2, 3, 34, 1, 99, 101] = 2
  - *tail* [2, 3, 34, 1, 99, 101] = [3, 34, 1,99, 101]
  - *last* [2, 3, 34, 1, 99, 101] = 101
  - *front* [2, 3, 34, 1, 99, 101] = [2, 3, 34, 1, 99]

Formal Specification Languages

- A formal specification language is usually composed of three primary components:
  - a syntax that defines the specific notation with which the specification is represented
  - semantics to help define a "universe of objects" that will be used to describe the system
  - a set of relations that define the rules that indicate which objects properly satisfy the specification
- The syntactic domain of a formal specification language is often based on a syntax that is derived from standard set theory notation and predicate calculus.
- The *semantic domain* of a specification language indicates how the language represents system requirements.

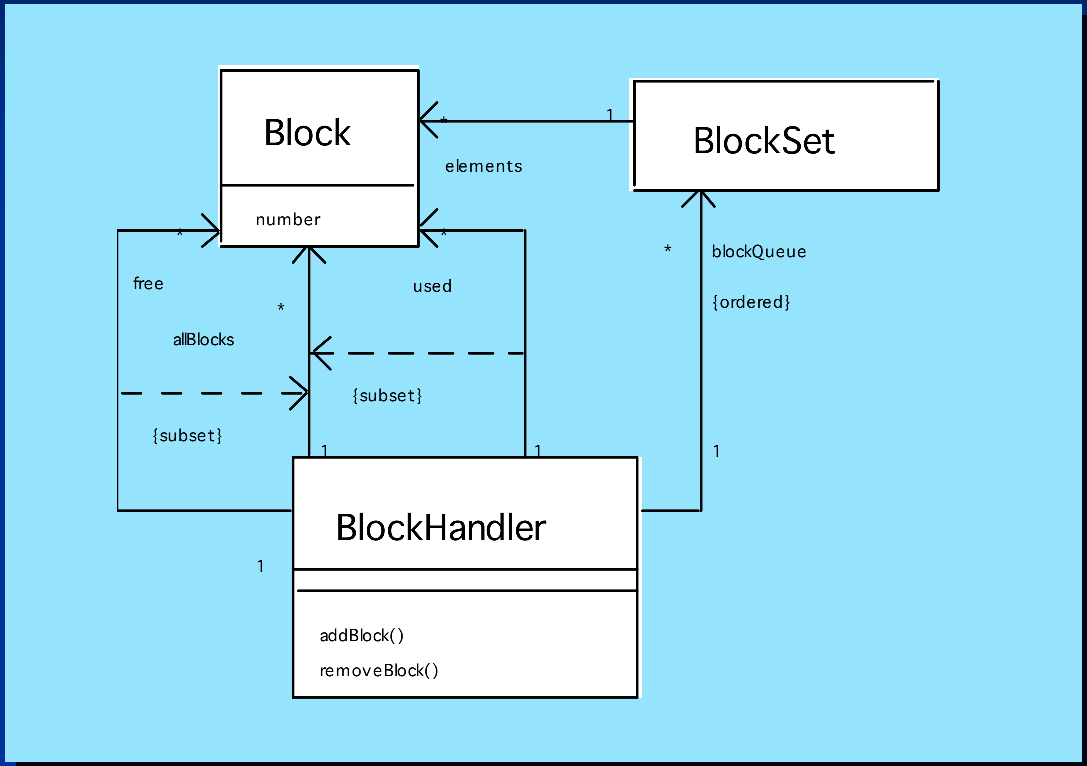
Object Constraint Language (OCL)

- a formal notation developed so that users of UML can add more precision to their specifications
- All of the power of logic and discrete mathematics is available in the language
- However the designers of OCL decided that only ASCII characters (rather than conventional mathematical notation) should be used in OCL statements.

OCL Overview

- Like an object-oriented programming language, an OCL expression involves operators operating on objects.
- However, the result of a complete expression must always be a Boolean, i.e. true or false.
- The objects can be instances of the OCL **Collection** class, of which **Set** and **Sequence** are two subclasses

BlockHandler using UML



BlockHandler in OCL

- No block will be marked as both unused and used.
  - ***context*** *BlockHandler inv:*
  - *(self.used->intersection(self.free)) ->isEmpty()*
- All the sets of blocks held in the queue will be subsets of the collection of currently used blocks.
  - ***context*** *BlockHandler inv:*
  - *blockQueue->forAll(aBlockSet | used->includesAll(aBlockSet))*
- No elements of the queue will contain the same block numbers.
  - ***context*** *BlockHandler* ***inv****:*
  - *blockQueue->forAll(blockSet1, blockSet2 |*
  - *blockSet1 <> blockSet2 implies*
  - *blockSet1.elements.number->excludesAll(blockSet2.elements.number))*
  - The expression before *implies* is needed to ensure we ignore pairs where both elements are the same Block.
- The collection of used blocks and blocks that are unused will be the total collection of blocks that make up files.
  - ***context*** *BlockHandler* ***inv****:*
  - *allBlocks = used->union(free)*
- The collection of unused blocks will have no duplicate block numbers.
  - ***context*** *BlockHandler* ***inv****:*
  - *free->isUnique(aBlock | aBlock.number)*
- The collection of used blocks will have no duplicate block numbers.
  - ***context*** *BlockHandler* ***inv****:*
  - *used->isUnique(aBlock | aBlock.number)*

The Z Language

- organized into *schemas*
  - defines variables
  - establishes relationships between variables
  - the analog for a “module” in conventional languages

The Clean Language

- Computation organized into *functional expressions*
  - Result is determined by definition and arguments
  - No side-effects
  - Semantics is determined by function evaluation substituting actual arguments in a copy of the definition
  - Specification can be executed