Data Abstractions

Lecture 4

Daqing Hou, Winter 2007

Outline

- Data Abstractions
- Representation, or *rep*
- Abstraction function & rep invariant
- Implementing abstraction functions and rep invariants
- Reasoning about data abstractions
- Design consideration

Motivating problem

Given a series of whole numbers, identify ones that repeat and ones that are unique. e.g.,

Input:

[100, 98, 5, 7, 98, 102, 5, 101, 201, 101, 33, 99, 17, 27, 100]

Output:

Unique: [7, 102, 201, 33, 99, 17, 27]

Repeating: [98, 5, 101, 100]

Hands-on: IntSet

- What operations ought an integer set support?
- Possible representations?

Data objects

- A data object can be implemented as a storage representation, e.g., an array for IntSet
- It is beneficial to limit the impact of changes to such implementations by depending on operations rather than rep. E.g., for IntSet,
  - when changing representation (from array to binary tree)
  -or its interpretation (from unsorted to sorted)
  -or optimizing memory usage for performance (shrinking array of IntSet)
- Examples other than IntSet?

Data abstractions

- data abstraction = <objects, operations>
- A way of adding new types to base languages
- Using programs use only operations, with no direct access to rep
- Adequacy for use: adding enough operations

Rep and values

- Values of a rep
- legal values
  - not all values of the rep are legitimate objects

Rep invariant

- A property all legal values satisfy.
- E.g., IntSet:
  ```
  elems!=null && for i, j: 0<=i<j<index
  =>
  elems[i]!=elems[j]
  ```

Abstraction function

- A function that maps a legal value of a rep to an abstract object
- Often *Many-to-one*
- What cause many-to-one? E.g.,
  - "unused" memory
  - Flexibility in implementations, e.g.: order for array of int's

Reasoning about data abstractions

- Preserving rep invariant
- Correct operations
- Abstract invariants

Rep exposure

- Desirable to limit the impact of changes. E.g., for IntSet,
  - when changing representation (from array to binary tree)
  -or its interpretation (from unsorted to sorted)
  -or optimizing memory usage for performance (shrinking array of IntSet)
- Thus it is an implementation error to expose rep of an abstraction.

Design considerations

- Mutability
  - mutable abstract objects must have mutable reps; immutable abstract objects may have mutable reps
- Operation categories
  - creater
  - producer
  - mutator
  - observer
- Adequacy
  - context of use
  - efficiency & easy-to-use
  - too few operations; force user to impl. operation on top of provided ones.
  - too many operations; may increase learning effort

Locality and modifiability

- Locality refers to the ability of impl'ing one abstraction on top of others' spec, without having to resort to examining their impl.
- Modifiability refers to the ability of changing impl. of abstraction without having to change its client code.
- Difference?