CSE331 midterm review

Autumn 2010

Exam structure

- 50 minutes, in class – Matt will proctor
- Open note, open book, closed neighbor, closed anything electronic (computers, web-enabled phones, etc.)
- An easier-to-read answer makes for a happier-to-give-partial-credit grader

More structure

- Three 15-minute equally weighted exam sections

A. Specifications and subtyping
B. Abstract data types, representation invariants and abstraction functions
C. Miscellaneous (mutability, testing, equality, subclassing, …)

A. Specifications and subtyping

- Role of specifications – difference from implementation
- Stronger vs. weaker specifications
- Java subtyping vs. true subtyping

A. Role of Specifications

- vs. code
- Two hats – implementer and client
  - What are the different objectives when wearing each hat?

A. Stronger and weaker

- There will be at least two questions about comparing specifications in terms of strength or weakness
  - At least one will be abstract – that is, a question of logic and mathematics without concern for software per se
  - At least one will concern this issue in the context of software (that is, may include throws clauses, etc.)

A. Key issues

Stronger and weaker specifications

- A stronger specification is
  - *harder to satisfy* (implement) because it promises more – that is, its effects clause is harder to satisfy and/or there are fewer objects in modifies clause – but
  - *easier to use* (more guarantees, more predictable) by the client – that is, the requires clause is easier to satisfy
- A weaker specification is
  - *easier to satisfy* (easier to implement and more implementations satisfy it) because it promises less – that is, the effects clause is easier to satisfy and/or there are more objects in modifies clause – but
  - *harder to use* (makes fewer guarantees) because it asks more of the client – that is, the requires clause is harder to satisfy

A. Subtyping

- At least one question focused on whether a specific Java subtype is or is not a true subtype, and why

B. Abstract data types…

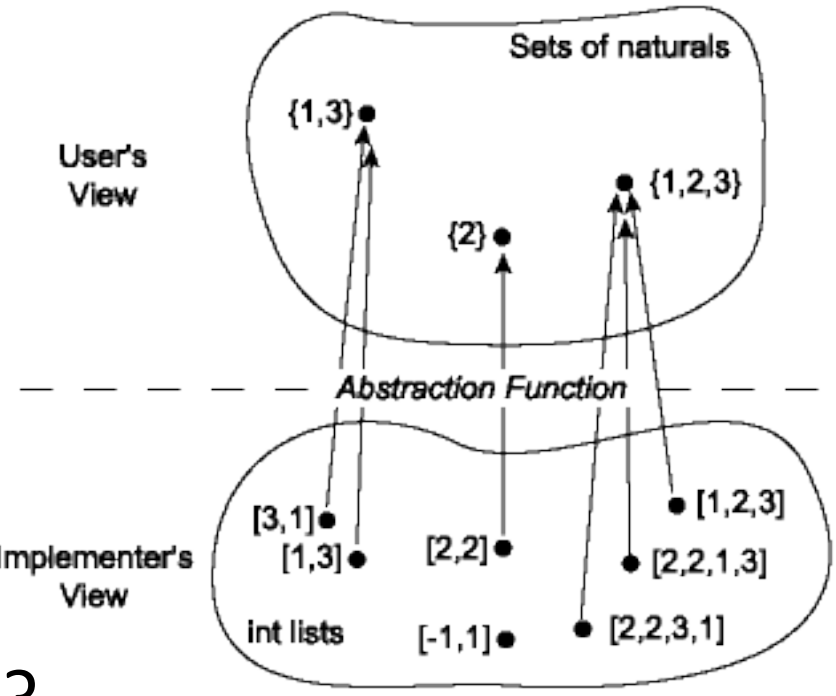
- Abstract data types, representation invariants and abstraction functions
- ADTs provide a set of operations and semantics over those operations
  - Ex: A stack ADT that provides new, push, pop and top operations – and some way of understanding "stackness" (perhaps descriptions such as if push succeeds then top returns the last pushed element)

B. Implementations

- It is common to implement ADTs in programming languages, most often OO programming languages
- What is the relationship between the ADT and the implementation?

B. Abstraction function

- The AF gives meaning to the representation of data in the implementation
- This is a figure from
- The AF maps from the representation to the abstract values and may be many-to-one
- Why not abstract to representation?
- AF formal or informal?

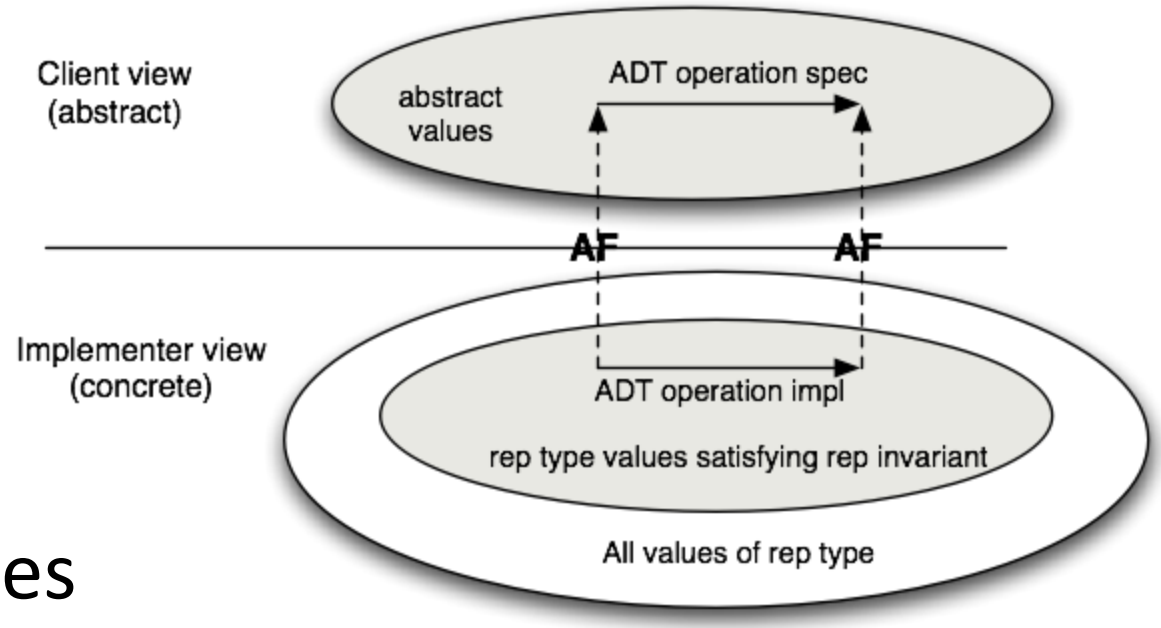


B. Representation invariant

- These are constraints on the concrete representation alone – only if this invariant is true is there a guarantee that the AF makes sense when applied to the representation
- The RI is guaranteed to hold by an implementation only at method entry and exit – why not always?

B. AF and RI relationship

- Again from
- Puts together what we discussed
- The “all values of rep type” includes all representations that satisfy and do not satisfy the RI



B. Representation exposure

- Representation exposure occurs when a client of an ADT can learn unintended properties about an implementation – this can easily preclude or complicate making later changes to the implementation
- Aliasing, mutability, etc. are common bases for representation exposure – they can be used carefully and properly, but often aren't

B. Questions

- There will be a set of (most likely) linked questions about a specific ADT and reasonable AF and RI for it
- There may be a linked rep exposure question, but if not there will be a standalone one – most likely, “Does the following have any representation exposure? If so, what?”

C. Miscellaneous

- Mutability, testing, equality, subclassing, …
- Example topics/questions (all of which would be more focused) – can't fit all these in, though!
  - Describe a situation where mutability is a good choice even with the risk of rep exposure
  - In what way can we consider testing as a way of verifying whether an implementation satisfies a specification?
  - What are the strengths of black- vs. white-box testing?

C. continued

- Example topics/questions (all of which would be more focused)
  - Some semi-tricky question about equality and the equivalence relationship
  - Subtyping vs. subclassing – sharing behavior vs. sharing code