Reasoning About ADTs

Krysta Yousoufian
CSE 331
Code Examples

• See package induction_examples for all the code for these examples.
Uses of reasoning

- Testing can only go so far
  - Can’t test every set of operations on every object
- Reasoning can prove correctness over all operations, objects
Proof by Induction

• Want to prove some property $P$ about an object

• **Base case**
  Prove that $P$ holds for newly-constructed object

• **Inductive step**
  Prove that if $P$ holds for an object $O$, it holds after any operation on $O$
Verifying Rep Invariant

- Verify that rep invariant is always satisfied
- Reason about **implementation** (instance fields)
- **Base case**: Prove that RI holds after constructor
- **Inductive step**: Prove that if RI holds going into any method, it holds going out
BankAccount

// Rep invariant:
// transactions contains no null values
// the sum of all values in transactions
// is >= 0

See BankAccount.java
BankAccount

• Base case:
  o transactions is empty => no null values
  o Transactions is empty => sum of values is 0
BankAccount

Base case:
  o transactions is empty => no null values
  o transactions is empty => sum of values is 0

Inductive case: assume RI holds on entering method
  • `getBalance()`:
    o Doesn’t modify transactions, so RI is preserved.
BankAccount

Inductive case:

- `performTxn()`:
  - `getBalance()` returns the sum of amounts in transactions
  - Case 1: current sum of transactions + amount of `txn < 0`. transactions is unchanged, so RI still holds.
  - Case 2: current sum of transactions + amount of `txn >= 0`. Therefore, adding `txn` will not make the sum negative. We also verified that `txn` is not null. The only change to transactions is that `txn` is added, so the RI still holds.
Verifying Client Code

• Verify that client code behaves correctly
• Want to prove some statement P about the object
  o e.g. abstract invariant
• Reason about specification (abstract fields)
• Assume implementation meets the specs
• **Base case:** Prove that P holds after constructor
• **Inductive step:** Prove that if P holds going into any method, it holds going out
• Can ignore observer methods
/**
 * Abstract invariant: balance >= 0
 */

See [BankAccount.java](#)
BankAccount

- \( P(X) = X.\text{balance} \geq 0 \)
- Want to prove \( P(S) \) for all \( S \)
- **Base case:** \( S \) was created by constructor
  - After constructor, balance = 0, so \( P(S) \) holds
BankAccount

• \( P(X) = X \text{.balance} \geq 0 \)
• Want to prove \( P(S) \) for all \( S \)
• **Inductive case:** \( S \) was created by a call of the form “\( T \text{.performTxn}(txn) \)”:
  o **Assume** \( P(T) \) (inductive hypothesis), prove \( P(S) \)
BankAccount

- $P(X) = X.balance \geq 0$
- Want to prove $P(S)$ for all $S$
- **Inductive case**: $S$ was created by a call of the form “$T.performTxn(txn)$”:
  - **Assume $P(T)$** (inductive hypothesis), prove $P(S)$
**BankAccount**

- $P(X) = X\.balance \geq 0$
- Want to prove $P(S)$ for all $S$
- **Inductive case:** $S$ was created by a call of the form “$T\.performTxn(txn)$”:
  - We **assume** $P(T)$ (inductive hypothesis) and will prove $P(S)$
  - Case 1: balance is left unchanged.
    - $T\.balance = S\.balance$, so $P(S)$ holds by inductive hypothesis or assumption that $P(T)$
BankAccount

- \( P(X) = X.\text{balance} \geq 0 \)
- Want to prove \( P(S) \) for all \( S \)
- **Inductive case:** \( S \) was created by a call of the form “\( T.\text{performTxn}(\text{txn}) \)”:
  - We **assume** \( P(T) \) (inductive hypothesis) and will prove \( P(S) \)
  - Case 1: balance is left unchanged.
    - \( T.\text{balance} = S.\text{balance} \), so \( P(S) \) holds by inductive hypothesis or assumption that \( P(T) \)
  - Case 2: balance = balance + \( \text{txn.\text{amount}} \).
    - Only enter this case if balance + \( \text{txn.\text{amount}} \) \( \geq 0 \). Therefore, new balance will be \( \geq 0 \) and \( P(S) \) holds
TreeSet: prove RI

1. data == null iff (left == null and right == null)
2. If data != null, all non-null values in tree rooted at left are < data and all values in tree rooted at right are > data

See TreeSet.java
TreeSet: prove RI

1. data == null iff (left == null and right == null)
2. If data != null, all non-null values in tree rooted at left are < data and all non-null values in tree rooted at right are > data

**Base case:** S was created by constructor
- data == null, left == null, right == null
- #1 holds because data == null and (left == null and right == null)
- #2 holds trivially because !(data != null)
TreeSet: prove RI

1. data == null iff (left == null and right == null)
2. If data != null, all non-null values in tree rooted at left are < data and all non-null values in tree rooted at right are > data

**Inductive case:** assume RI holds on entering method

- `contains()`: never modifies anything, so RI is preserved
TreeSet: prove RI

1. data == null iff (left == null and right == null)
2. If data != null, all non-null values in tree rooted at left are < data and all non-null values in tree rooted at right are > data

Inductive case:
• **add():** four cases:
  o Case 1: val == null. Object is unchanged, so RI is preserved
TreeSet: prove RI

1. data == null iff (left == null and right == null)
2. If data != null, all non-null values in tree rooted at left are < data and all non-null values in tree rooted at right are > data

Inductive case:

- add(): four cases:
  - Case 1: val == null. Object is unchanged, so RI is preserved
  - Case 2: data == null. data is assigned to val (which is non-null) and left and right are initialized, so #1 holds. left and right contain only null values immediately after construction, and no other values are added, so #2 holds.
TreeSet: prove RI

1. data == null iff (left == null and right == null)
2. If data != null, all non-null values in tree rooted at left are < data and all non-null values in tree rooted at right are > data

Inductive case:
- **add():** four cases:
  - Case 3: data != null and val.compareTo(data) < 0, i.e. val < data.
  - Because we assume the RI holds going in, initially data != null, left != null, and right != null. None of these values are reassigned, so #1 holds.
  - The only possible change is that val and two empty nodes are added to the left subtree. Because val < data and empty nodes contain only nulls, the first clause of #2 is preserved. Because the right subtree is unchanged, the second clause of #2 is preserved.
TreeSet: prove RI

1. data == null iff (left == null and right == null)
2. If data != null, all non-null values in tree rooted at left are < data and all non-null values in tree rooted at right are > data

Inductive case:
• add(): four cases:
  o Case 4: data != null and val.compareTo(data) > 0, i.e. val > data.
  o (Prove analogously to Case #3)
TreeSet: prove client code

• Verify that a value is contained in TreeSet iff it has been added to the TreeSet at least once.

See TreeSet.java
TreeSet: prove client code

- \( P(X) = \text{for all values } v, v \in X \iff \text{X.add(v) was called at some point} \)
- Want to prove \( P(S) \) for all \( S \)
TreeSet: prove client code

- \[ P(X) = \text{for all values } v, v \in X \text{ iff } X.\text{add}(v) \text{ was called at some point} \]
- Want to prove \( P(S) \) for all \( S \)
- **Base case:** \( S \) was created by constructor
  - After constructor, \( S \) is an empty set and there have been no calls to add, so \( P(S) \) holds
TreeSet: prove client code

- \( P(X) = \) for all values \( v, v \in X \) iff
  \( X.add(v) \) was called at some point

- **Inductive case:** \( S \) was created by a call of the form
  “\( T.add(v) \)”:
  - We **assume** \( P(T) \) (inductive hypothesis) and will
    prove \( P(S) \)
  - Case 1: \( S = T \). Only occurs if \( v \in T \) and thus \( v \in S \).
    Because \( P(T) \) holds (by the inductive hypothesis),
    \( S = T \), and \( v \in S \), \( P(S) \) must also hold.
TreeSet: prove client code

- \( P(X) = \) for all values \( v \), \( v \in X \) iff \( X.add(v) \) was called at some point

- **Inductive case:** \( S \) was created by a call of the form “\( T.add(v) \)”:  
  - We **assume** \( P(T) \) (inductive hypothesis) and will prove \( P(S) \)  
  - Case 1: \( S = T \). Only occurs if \( v \in T \) and thus \( v \in S \). Because \( P(T) \) holds (by the inductive hypothesis), \( S = T \), and \( v \in S \), \( P(S) \) must also hold.  
  - Case 2: \( S = T \cup v \). We know \( v \in S \) by the definition of union, so the newly-added value is contained in \( S \). We know \( P(T) \) by the inductive hypothesis, and the only change between \( T \) and \( S \) is the union with \( v \), so \( P(S) \) also holds.
IntQueue

• Remember IntQueue1 and IntQueue2 from HW4?
• Prove rep invariant
• Prove that values are contained in the order they were added by the user