ACCOUNTING INVARIANT

ACCOUNTING INVARIANT

Prove that bank account is always non-negative

ACCOUNTING INVARIANT

Prove that bank account is always non-negative

Prove that savings can cover every expensive operation

ACCOUNTING INVARIANT

implies

Strong: Prove that bank account is always non-negative

Sufficient: Prove that savings can cover every expensive operation

ACCOUNTING INVARIANT

Strong: Prove that bank account is always non-negative

implies

Sufficient: Prove that savings can cover every expensive operation

(typically, quantify bank credits as a function of the data structure)

ACCOUNTING INVARIANT

Strong: Prove that bank account is always non-negative

implies

Sufficient: Prove that savings can cover every expensive operation

(typically, quantify bank credits as a function of the data structure)

examples

- multipop
- binary counter
- dynamic arrays

ACCOUNTING INVARIANT

Strong: Prove that bank account is always non-negative

implies

Sufficient: Prove that savings can cover every expensive operation

(typically, quantify bank credits as a function of the data structure)

examples

- multipop: credits in bank = #elements in stack
- binary counter
- dynamic arrays

ACCOUNTING INVARIANT

Strong: Prove that bank account is always non-negative

implies

Sufficient: Prove that savings can cover every expensive operation

(typically, quantify bank credits as a function of the data structure)

examples

- multipop: credits in bank = #elements in stack
- binary counter: credits in bank = #1's in counter
- dynamic arrays

ACCOUNTING INVARIANT

Strong: Prove that bank account is always non-negative

implies

Sufficient: Prove that savings can cover every expensive operation

(typically, quantify bank credits as a function of the data structure)

examples

- multipop: credits in bank = #elements in stack
- binary counter: credits in bank = #1's in counter
- dynamic arrays: credits in bank when we double = #elements in array

ACCOUNTING INVARIANT

Strong: Prove that bank account is always non-negative

implies

Sufficient: Prove that savings can cover every expensive operation

(typically, quantify bank credits as a function of the data structure)

examples

- multipop: credits in bank = #elements in stack
- binary counter: credits in bank = #1's in counter
- dynamic arrays: credits in bank when we double = #elements in array = 2 · (# new elements since last expansion)

ACCOUNTING INVARIANT

Strong: Prove that bank account is always non-negative

implies

Sufficient: Prove that savings can cover every expensive operation

(typically, quantify bank credits as a function of the data structure)

examples

- multipop: credits in bank = #elements in stack
- binary counter: credits in bank = #1's in counter
- dynamic arrays: credits in bank when we double = #elements in array

  = 2·(# new elements since last expansion)