

Refinement

•What is refinement?

- Application of formal methods and proofs to detailed design of software
- Ensure that implementation correctly mirrors a high-level model

•Refinement components

- Abstract model
- Concrete model
- Correspondence between models

Why Use Refinement?

•Abstract model

- Reflects system requirements
 - High-level description
- But can't be implemented directly

•Concrete model

- Uses available programming features
 - Programming language capabilities
 - Data structures

•Want to ensure correct implementation

Refinement Example

•Sets

- Abstract concept
- Often used in models (especially in Z)
- Not directly supported by languages (C++)
 - Even STL set isn't really a set

•How can a set be implemented?

- Tree structure? (STL set)
- Linear array or vector? (simpler?)

Set – Abstract Model

$[X]$ ← Type of element in set

Abstract
 $s: P X$ Abstract set specification

AStore
 $\Delta Abstract$
 $x?: X$ Store an element in a set
 $s' = s \cup \{x?\}$

Set – Concrete Model

Concrete Concrete set specification
(array modeled by sequence)
 $ss: seq X$

CStore Concrete “store” operation
 $\Delta Concrete$
 $x?: X$
 $ss' = ss \hat{\ } \langle x? \rangle$

Abstract/Concrete Correspondence

Abs
Abstract
Concrete
 $s = ran ss$

“Abstraction” schema links abstract and concrete models

Predicate specifies abstraction invariant

Proof Obligations

•Initial state

- Ensure initial states are compatible

•Representation uniqueness

- Concrete unambiguously specifies abstract

•Applicability of operations

- Verify that abstract precondition implies concrete precondition

•Correctness of operations

- Verify that concrete result implies desired abstract result

Initial States

Abstract model – empty set

AInit

Abstract

$s = \emptyset$

Concrete model – empty sequence

CInit

Concrete

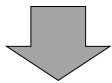
$ss = \diamond$

Initial State Proof

theorem *InitOK*

$CInit' \wedge Abs' \Rightarrow AInit'$

Initialization theorem:
concrete initialization implies
abstract initialization



InitOK

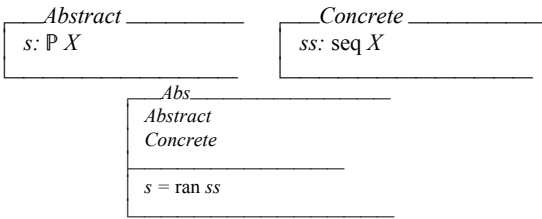
$s': P X$

$ss': seq X$

Expanded schema:
Can you prove this?

$(ss' = \diamond \wedge s' = ran ss') \Rightarrow s' = \emptyset$

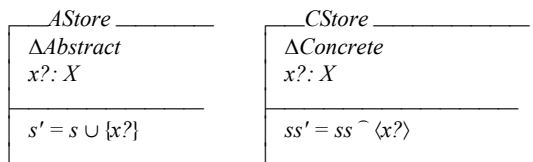
Unique Representation Proof



theorem *AUniqueForC*
 $\forall \text{Concrete} \cdot \exists_1 \text{Abstract} \cdot \text{Abs}$

Make sure concrete state uniquely determines abstract state

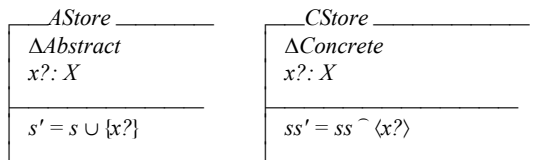
Operation Applicability Proof



theorem *CStoreApplicable*
 $\text{pre AStore} \wedge \text{Abs} \Rightarrow \text{pre CStore}$

Make sure concrete operation can be applied whenever abstract operation is valid

Operation Correctness Proof



theorem *CStoreCorrect*
 $\text{pre AStore} \wedge \Delta \text{Abs} \wedge \text{CStore} \Rightarrow \text{AStore}$

Make sure concrete operation result implies correct abstract operation result

Set Removal Operation

<i>ARemove</i>
$\Delta Abstract$ $x?: X$
$s' = s \setminus \{x?\}$

<i>CRemove</i>
$\Delta Concrete$ $x?: X$
????

How should this operation be implemented?

Refinement Proof Process

- **Z/EVES preparation**
 - Create models
 - Define theorems
- **Proofs**
 - Operation domain checks
 - As needed
 - Theorem proofs
- **A possible complication**
 - Proving implementation with available laws
