CS 3110

Abstraction Functions and Representation Invariants

Prof. Clarkson
Fall 2018

Today's music: *Never Change* by JAY-Z

Attendance question

Yesterday's lecture has been part of the course for something like 10 years but is starting to feel like review. Should I cut it next year in favor of more advanced material?

A. Yes
B. No

Review

Previously in 3110:

- Specifying functions

Today:

- Specifying data abstractions

Back to: Audience of specification

- **Clients**
  - Spec informs what they must guarantee (preconditions)
  - Spec informs what they can assume (postconditions)
- **Implementers**
  - Spec informs what they can assume (preconditions)
  - Spec informs what they must guarantee (postconditions)

But the spec isn't **enough** for implementers...

REPRESENTATION TYPES

Demo

Discussion

What sets do these lists represent?

Does it matter which implementation we use?

- `[1;2]`
- `[2;1]`
- `[1;1;2]`
- `[]`

Question

```
(** [union lst1 lst2] is the set
    union of [lst1] and [lst2]. *)
let union lst1 lst2 = lst1 @ lst2
```

Under which invariant is that correct?

A. Duplicates are allowed in lists
B. Duplicates are not allowed in lists
C. Both A and B
D. Neither A nor B

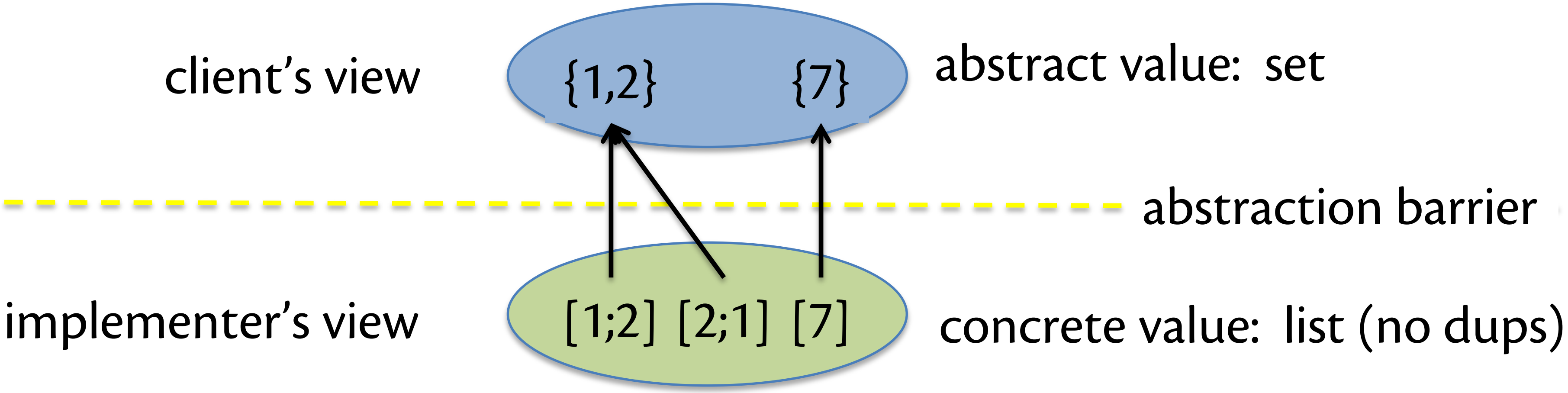
Representation types

- Q: How to interpret the representation type as the data abstraction?
- A: Abstraction function

- Q: How to determine which values of representation type are meaningful?
- A: Representation invariant

ABSTRACTION FUNCTIONS

Abstraction function



client's view

{1,2}

{7}

abstract value: set

abstraction barrier

implementer's view

[1;2] [2;1] [7]

concrete value: list (no dups)

the black arrows are the abstraction function

Abstraction function

maps

valid concrete values

to

abstract values

Documenting the AF

- Above rep type in implementation you write:

```
(* AF: comment *)
```

- Write it first before implementing operations

Demo

Discussion

When and how would you implement an AF as part of a data abstraction?

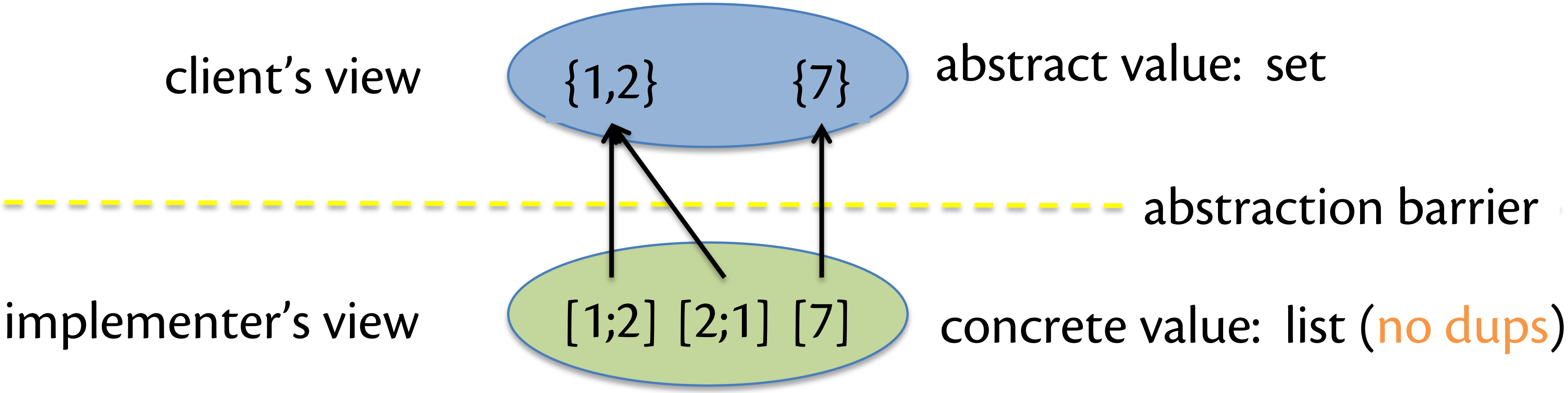
Representation types

- Q: How to interpret the representation type as the data abstraction?
- A: Abstraction function

- Q: How to determine which values of representation type are meaningful?
- A: Representation invariant

REPRESENTATION INVARIANTS

Abstraction function



client's view

{1,2}

{7}

abstract value: set

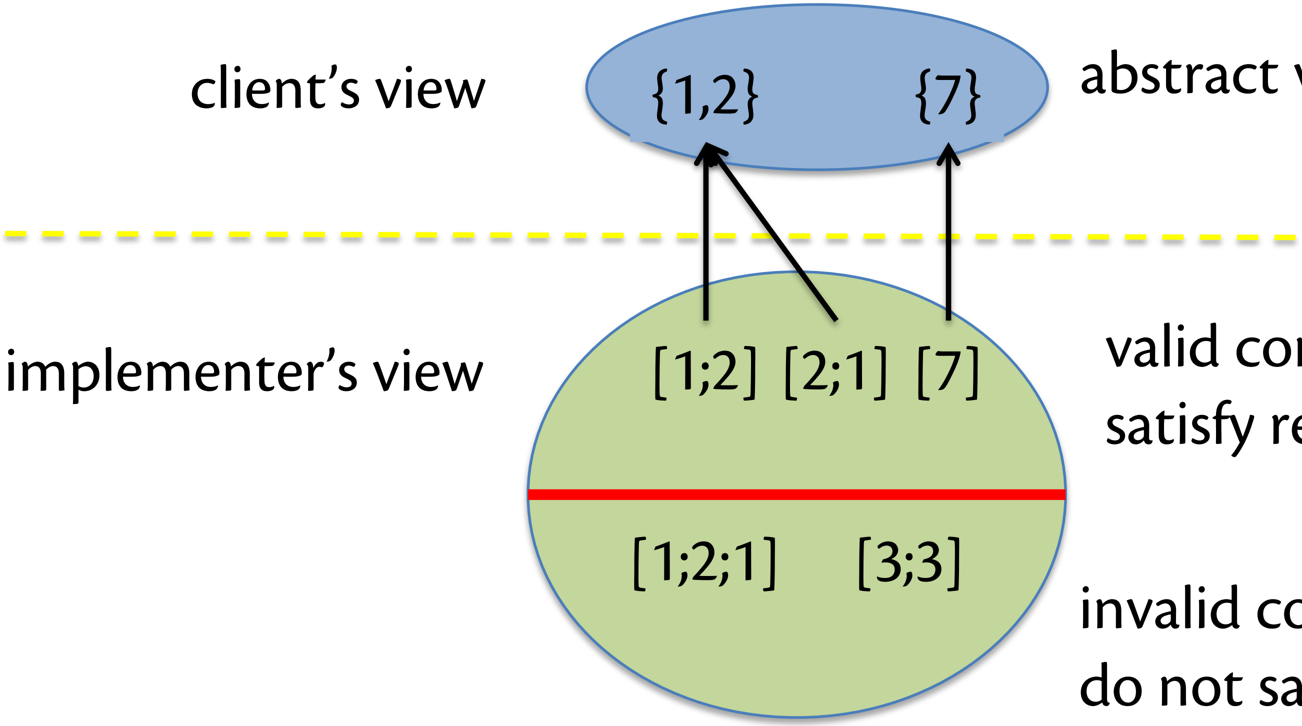
abstraction barrier

implementer's view

[1;2] [2;1] [7]

concrete value: list (no dups)

Representation invariant



client's view

{1,2} {7}

abstract value: set

abstraction barrier

implementer's view

[1;2] [2;1] [7]

valid concrete values:
satisfy rep. invariant

[1;2;1] [3;3]

invalid concrete values:
do not satisfy rep. invariant

the thick red line is the rep. invariant

Rep. invariant

distinguishes

valid concrete values

from

invalid concrete values

Documenting the RI

- Above rep type in implementation you write:

```
(* RI: comment *)
```

- Write it first before implementing operations

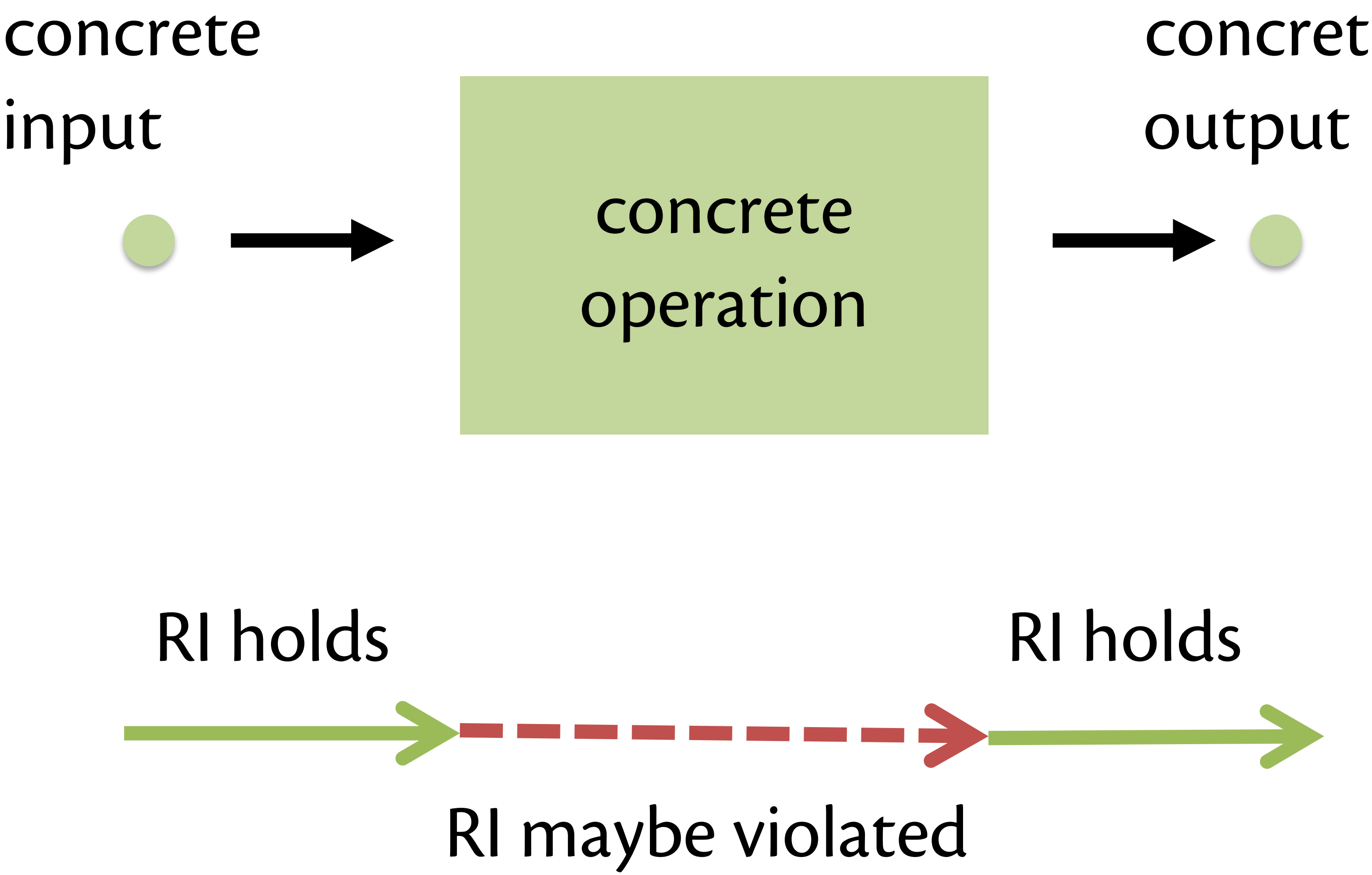
Demo

Rep. invariant

implicitly part of
every precondition and
every postcondition
in abstraction

Demo

Invariant may temporarily be violated



concrete input

concrete operation

concrete output

RI holds

RI holds

RI maybe violated

Demo

Discussion

When and how would you implement a RI as part of a data abstraction?

Implementing the RI

Idiom: if RI fails then raise exception, otherwise return concrete value

Demo

Recap

- Q: How to interpret the representation type as the data abstraction?
- A: Abstraction function

- Q: How to determine which values of representation type are meaningful?
- A: Representation invariant

Upcoming events

- N/A

This is invariant.

THIS IS 3110