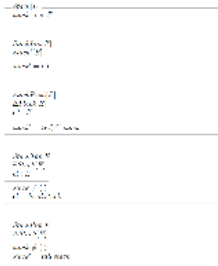




How would you represent a stack?

Could a general design of such an implementation in UML without predetermining its structure?



This is a simple function because a stack could behave as a stack even though its internal structure would be different.

How could we express a stack mathematically?

Types

$Stack : \mathbb{N}$

Functions

$push : Stack \times \mathbb{N} \rightarrow Stack$
 $pop : Stack \times \mathbb{N} \rightarrow Stack \times \mathbb{N}$
 $size : Stack \times \mathbb{N} \rightarrow \mathbb{N}$

Axioms

$\forall s : Stack, x : \mathbb{N}$

- A1: $pop(push(s, x)) = s$
- A2: $push(pop(s, x)) = s$
- A3: $pop(pop(s, x)) = s$
- A4: $push(push(s, x), y) = push(s, y)$

Preconditions

$pop : Stack \times \mathbb{N}$ requires $empty(s)$
 $size : Stack \times \mathbb{N}$ requires $empty(s)$

Algebraic specification abstract data types we do not know the internal structure

Where are the functions generalized?

What makes a stack a stack?

Does the modeling need to be applied?
 Or, how does the model work with stack?

Can we be sure that the specification is correct?

How can we ensure that the specification is correct?

Can algebraic specification be applied to application specific types?

How are the operations specified and their dependencies?

Types

$Order, Item$

Functions

$add : Order \times Item \rightarrow Order$
 $remove : Order \times Item \rightarrow Order$
 $dispatch : Order \rightarrow Order$
 $size : Order \rightarrow \mathbb{N}$
 $empty : Order \rightarrow Boolean$

Axioms

$\forall s : Order, p : Item$

- A1: $remove(remove(s, p), p) = s$
- A2: $remove(s, p) = s$
- A3: $empty(remove(s, p)) = empty(s)$

Preconditions

$remove(s, p : Order, p : Item)$ requires $empty(s)$
 $dispatch(s : Order)$ requires $empty(s)$

Types

$Order, Item$

Functions

$add : Order \times Item \rightarrow Order$
 $remove : Order \times Item \rightarrow Order$
 $dispatch : Order \rightarrow Order$
 $size : Order \rightarrow \mathbb{N}$
 $empty : Order \rightarrow Boolean$

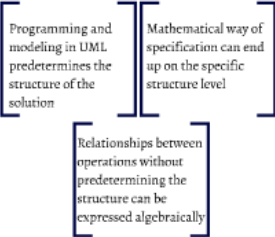
Axioms

$\forall s : Order, p : Item$

- A1: $remove(remove(s, p), p) = s$
- A2: $remove(s, p) = s$
- A3: $empty(remove(s, p)) = empty(s)$
- A4: $dispatch(dispatch(s)) = s$

Preconditions

$remove(s : Order, p : Item)$ requires $empty(s)$ and $dispatch(s)$
 $dispatch(s : Order)$ requires $empty(s)$ and $dispatch(s)$
 $size(s : Order)$ requires $empty(s)$
 $empty(s : Order, p : Item)$ requires $dispatch(s)$



Lecture 7:

Algebraic Specification

Valentino Vranić

Ústav informatiky, informačných systémov a
softvérového inžinierstva



vranic@stuba.sk

fiit.sk/~vranic

MSOFT 2019/20

5. 11. 2019

How would you
implement a stack?

Could a general design of a stack be expressed in UML without predetermining its structure?

Programming and
modeling in UML
predetermines the
structure of the
solution

How could we
express a stack
mathematically?

$$\frac{Stack[E]}{stack : seq E}$$
$$\frac{StackInit[E] \quad Stack'[E]}{stack' = \langle \rangle}$$
$$\frac{StackPush[E] \quad \Delta Stack[E] \quad e? : E}{stack' = \langle e? \rangle \frown stack}$$
$$\frac{StackTop[E] \quad \exists Stack[E] \quad e! : E}{stack \neq \langle \rangle \quad e! = head\ stack}$$
$$\frac{StackPop[E] \quad \Delta Stack[E]}{stack \neq \langle \rangle \quad stack' = tail\ stack}$$

This is **overspeification**

because a stack could behave
as a stack even though its
internal structure would be
different

Mathematical way of
specification can end
up on the specific
structure level

What makes a stack a stack?

Or: how do we know we work
with stack?

Use case modeling could be applied:

a data type + CRUD operations

Can we be more precise than
with use cases?

Operations (functions) do not
express structure

If we could express
relationships among functions,
we could avoid the structure

Types

$Stack[E]$

Functions

$new : Stack[E]$

$empty : Stack[E] \rightarrow Boolean$

$push : Stack[E] \times E \rightarrow Stack[E]$

$pop : Stack[E] \rightarrow Stack[E]$

$top : Stack[E] \rightarrow E$

Axioms

$\forall e : E, s : Stack[E]$

$$A1 : top(push(s, e)) = e$$

$$A2 : pop(push(s, e)) = s$$

$$A3 : empty(new)$$

$$A4 : \neg empty(push(s, e))$$

Preconditions

$pop(s : Stack[E])$ requires $\neg empty(s)$

$top(s : Stack[E])$ requires $\neg empty(s)$

Types

$Stack[E]$

Functions

$new : Stack[E]$

$empty : Stack[E] \rightarrow Boolean$

$push : Stack[E] \times E \rightarrow Stack[E]$

$pop : Stack[E] \rightarrow Stack[E]$

$top : Stack[E] \rightarrow E$

Axioms

$\forall e : E, s : Stack[E]$

$A1 : top(push(s, e)) = e$

$A2 : pop(push(s, e)) = s$

$A3 : empty(new)$

$A4 : \neg empty(push(s, e))$

Preconditions

$pop(s : Stack[E])$ requires $\neg empty(s)$

$top(s : Stack[E])$ requires $\neg empty(s)$

Algebraic specification

Abstract data types:

we abstract from the structure
of the data

Where are the function
postconditions?

Types

$Stack[E]$

Functions

$new : Stack[E]$

$empty : Stack[E] \rightarrow Boolean$

$push : Stack[E] \times E \rightarrow Stack[E]$

$pop : Stack[E] \rightarrow Stack[E]$

$top : Stack[E] \rightarrow E$

Axioms

$\forall e : E, s : Stack[E]$

$A1 : top(push(s, e)) = e$

$A2 : pop(push(s, e)) = s$

$A3 : empty(new)$

$A4 : \neg empty(push(s, e))$

Preconditions

$pop(s : Stack[E])$ requires $\neg empty(s)$

$top(s : Stack[E])$ requires $\neg empty(s)$

Axioms

$\forall e : E, s : Stack[E]$

$$A1 : top(push(s, e)) = e$$

$$A2 : pop(push(s, e)) = s$$

$$A3 : empty(new)$$

$$A4 : \neg empty(push(s, e))$$

Can algebraic specification
be applied to application
specific types?

Types

Order, Item

Functions

new : Order

addItem : Order × Item → Order

removeItem : Order × Item → Order

dispatch : Order → Order

delete : Order → Empty

empty : Order → Boolean

Axioms

$\forall p : \text{Item}, o : \text{Order}$

A1 : removeItem(addItem(o, p), p) = o

A2 : empty(new)

A3 : \neg empty(addItem(o, p))

Preconditions

removeItem(o : Order, p : Item) requires \neg empty(o)

dispatch(o : Order) requires \neg empty(o)

How to distinguish a
dispatched order from the
one that is not dispatched?

Types

Order, Item

Functions

new : Order

addItem : Order × Item → Order

removeItem : Order × Item → Order

dispatch : Order → Order

delete : Order → Empty

empty : Order → Boolean

Axioms

$\forall p : \text{Item}, o : \text{Order}$

A1 : removeItem(addItem(o, p), p) = o

A2 : empty(new)

A3 : \neg empty(addItem(o, p))

Preconditions

removeItem(o : Order, p : Item) requires \neg empty(o)

dispatch(o : Order) requires \neg empty(o)

Types

Order, Item

Functions

new : Order

addItem : Order × Item → Order

removeItem : Order × Item → Order

dispatch : Order → Order

delete : Order → Empty

empty : Order → Boolean

dispatched : Order → Boolean

Axioms

$\forall p : \text{Item}, o : \text{Order}$

A1 : removeItem(addItem(o, p), p) = o

A2 : empty(new)

A3 : \neg empty(addItem(o, p))

A4 : dispatched(dispatch(o))

Preconditions

removeItem(o : Order, p : Item) requires \neg empty(o) \wedge \neg dispatched(o)

dispatch(o : Order) requires \neg empty(o) \wedge \neg dispatched(o)

delete(o : Order) requires \neg dispatched(o)

addItem(o : Order, p : Item) requires \neg dispatched(o)

Relationships between operations without predetermining the structure can be expressed algebraically

Programming and modeling in UML predetermines the structure of the solution

Mathematical way of specification can end up on the specific structure level

Relationships between operations without predetermining the structure can be expressed algebraically