How would you implement a stack?

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

How could we express a stack mathematically?

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

What makes a stack a stack?

Or, how does/can we work with stack?

The state modeling could be applied

a data type + CRUD operations

Can we be more precise than with use cases?

Operations (functions) should express structure

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

Algebraic specification

Abstract data types
We abstract from the structure of the data

Where are the function preconditions?

Can algebraic specification be applied to application specific types?

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

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

Types

$$Stack[X]$$

Functions

$$\begin{aligned}
&new : Stack[X] \\
&empty : Stack[X] \to Boolean \\
&push : Stack[X] \times X \to Stack[X] \\
&pop : Stack[X] \to Stack[X] \\
&top : Stack[X] \to X
\end{aligned}$$

Axioms

$$\forall s : S, x : Stack[X]$$

$$\begin{aligned}
&A1 : empty(new) = true \\
&A2 : empty(push(s, x)) = false \\
&A3 : pop(new) = new \\
&A4 : \neg empty(s) \Rightarrow pop(push(s, x)) = s
\end{aligned}$$

Preconditions

$$\begin{aligned}
&pop(s : Stack[X]) \text{ requires } \neg empty(s) \\
&top(s : Stack[X]) \text{ requires } \neg empty(s)
\end{aligned}$$

Types

$$Order, Item$$

Functions

$$\begin{aligned}
&new : Order \\
&addItem : Order \times Item \to Order \\
&removeItem : Order \times Item \to Order \\
&dispatch : Order \to Order \\
&delete : Order \to Empty \\
&empty : Order \to Boolean
\end{aligned}$$

Axioms

$$\forall o : Item, s : Order$$

$$\begin{aligned}
&A1 : removeItem(addItem(o, p), p) = o \\
&A2 : empty(new) \\
&A3 : \neg empty(addItem(o, p)) \\
&A4 : \neg empty(addItem(s, o))
\end{aligned}$$

Preconditions

$$\begin{aligned}
&removeItem(o : Order, p : Item) \text{ requires } \neg empty(o) \\
&dispatch(o : Order) \text{ requires } \neg empty(o)
\end{aligned}$$

Types

$$Order, Item$$

Functions

$$\begin{aligned}
&new : Order \\
&addItem : Order \times Item \to Order \\
&removeItem : Order \times Item \to Order \\
&dispatch : Order \to Order \\
&delete : Order \to Empty \\
&empty : Order \to Boolean \\
&dispatched : Order \to Boolean
\end{aligned}$$

Axioms

$$\forall p : Item, s : Order$$

$$\begin{aligned}
&A1 : removeItem(addItem(o, p), p) = o \\
&A2 : empty(new) \\
&A3 : \neg empty(addItem(o, p)) \\
&A4 : dispatched(dispatch(o))
\end{aligned}$$

Preconditions

$$\begin{aligned}
&removeItem(o : Order, p : Item) \text{ requires } \neg empty(o) \wedge \neg dispatched(o) \\
&dispatch(o : Order) \text{ requires } \neg empty(o) \wedge \neg dispatched(o) \\
&delete(o : Order) \text{ requires } dispatched(o) \\
&addItem(o : Order, p : Item) \text{ requires } \neg dispatched(o)
\end{aligned}$$

Lecture 7:

Algebraic Specification

Valentino Vranić

Ústav informatiky, informačných systémov a softvérového inžinierstva

STU FIIT

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?

$Stack[E]$

$$stack : \operatorname{seq} E$$

$StackInit[E]$

$$Stack'[E]$$

$$stack' = \langle\ \rangle$$

$StackPush[E]$

$$\Delta Stack[E]$$
$$e? : E$$

$$stack' = \langle e? \rangle \frown stack$$

$StackTop[E]$

$$\Xi Stack[E]$$
$$e! : E$$

$$stack \neq \langle\ \rangle$$
$$e! = head\ stack$$

$StackPop[E]$

$$\Delta Stack[E]$$

$$stack \neq \langle\ \rangle$$
$$stack' = tail\ stack$$

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 wth use cases?

Operations (functions) do not express structure

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

Types

$$Stack[E]$$

Functions

$$new : Stack[E]$$
$$empty : Stack[E] \rightarrow Boolean$$
$$push : Stack[E] \times E \rightarrow Stack[E]$$
$$pop : Stack[E] \nrightarrow Stack[E]$$
$$top : Stack[E] \nrightarrow E$$

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

$$pop(s : Stack[E]) \text{ requires } \neg\ empty(s)$$
$$top(s : Stack[E]) \text{ requires } \neg\ empty(s)$$

Types

$$Stack[E]$$

Functions

$$new : Stack[E]$$
$$empty : Stack[E] \to Boolean$$
$$push : Stack[E] \times E \to Stack[E]$$
$$pop : Stack[E] \nrightarrow Stack[E]$$
$$top : Stack[E] \nrightarrow E$$

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

$$pop(s : Stack[E]) \text{ requires } \neg\ empty(s)$$
$$top(s : Stack[E]) \text{ requires } \neg\ empty(s)$$

Algebraic specifiction

Abstract data types:
we abstract from the structure of the data

Where are the function postconditions?

Types

$$Stack[E]$$

Functions

$$new : Stack[E]$$
$$empty : Stack[E] \to Boolean$$
$$push : Stack[E] \times E \to Stack[E]$$
$$pop : Stack[E] \nrightarrow Stack[E]$$
$$top : Stack[E] \nrightarrow E$$

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

$$pop(s : Stack[E]) \text{ requires } \neg\ empty(s)$$
$$top(s : Stack[E]) \text{ requires } \neg\ empty(s)$$

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

$$new : Order$$
$$addItem : Order \times Item \rightarrow Order$$
$$removeItem : Order \times Item \rightarrow Order$$
$$dispatch : Order \rightarrow Order$$
$$delete : Order \rightarrow Empty$$
$$empty : Order \rightarrow Boolean$$

Axioms

$$\forall\, p : Item, o : Order$$

$$A1 : removeItem(addItem(o, p), p) = o$$
$$A2 : empty(new)$$
$$A3 : \neg\ empty(addItem(o, p))$$

Preconditions

$$removeItem(o : Order, p : Item) \text{ requires } \neg\ empty(o)$$
$$dispatch(o : Order) \text{ requires } \neg\ empty(o)$$

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

Types

$$Order, Item$$

Functions

$$new : Order$$
$$addItem : Order \times Item \rightarrow Order$$
$$removeItem : Order \times Item \rightarrow Order$$
$$dispatch : Order \rightarrow Order$$
$$delete : Order \rightarrow Empty$$
$$empty : Order \rightarrow Boolean$$

Axioms

$$\forall\, p : Item, o : Order$$

$$A1 : removeItem(addItem(o, p), p) = o$$
$$A2 : empty(new)$$
$$A3 : \neg\ empty(addItem(o, p))$$

Preconditions

$$removeItem(o : Order, p : Item) \text{ requires } \neg\ empty(o)$$
$$dispatch(o : Order) \text{ requires } \neg\ empty(o)$$

Types

$$Order, Item$$

Functions

$$new : Order$$
$$addItem : Order \times Item \rightarrow Order$$
$$removeItem : Order \times Item \rightarrow Order$$
$$dispatch : Order \rightarrow Order$$
$$delete : Order \rightarrow Empty$$
$$empty : Order \rightarrow Boolean$$
$$dispatched : Order \rightarrow Boolean$$

Axioms

$$\forall\, p : Item, o : Order$$

$$A1 : removeItem(addItem(o, p), p) = o$$
$$A2 : empty(new)$$
$$A3 : \neg\, empty(addItem(o, p))$$
$$A4 : dispatched(dispatch(o))$$

Preconditions

$$removeItem(o : Order, p : Item) \text{ requires } \neg\, empty(o) \wedge \neg\, dispatched(o)$$
$$dispatch(o : Order) \text{ requires } \neg\, empty(o) \wedge \neg\, dispatched(o)$$
$$delete(o : Order) \text{ requires } \neg\, dispatched(o)$$
$$addItem(o : Order, p : Item) \text{ requires } \neg\, dispatched(o)$$

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