

# Schema Operators

Using Z

Woodcock & Davies

## State

We can use the language of schemas to describe the **state** of a system, and **operations** upon it.

Different aspects of the state—and different aspects of a given operation—can be described as separate schemas; these schemas may be combined in various ways using **schema operators**.

In this way, we may **factorise** the description, identifying common aspects for **re-use**, and providing **structure**.

## Schema operators

The logical schema operators:

$\wedge, \vee, \neg, \forall, \exists$

The relational schema operators:

$\circ, \gg$

## Merging declarations

The **merge** of two declarations is a declaration that introduces all of the names mentioned in either declaration.

If the same name appears in both declarations, then it is taken to represent the same object.

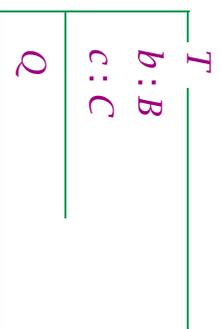
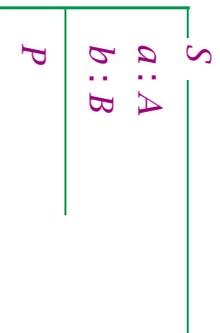
Two declarations can be merged only if common identifiers are declared with the same types.

## Schema conjunction

If  $S$  and  $T$  are two schemas, then their conjunction  $S \wedge T$  is a schema

- whose declaration is a merge of the two declarations
- whose constraint is a conjunction of the two constraints

## Example



The schema  $S \wedge T$  is equivalent to

$a : A$
$b : B$
$c : C$
$P \wedge Q$

## Example

<i>BoxOffice</i>
$sold : Seat \leftrightarrow Customer$
$seating : \mathbb{P} Seat$
$dom\ sold \subseteq seating$

*Status ::= standard | premiere*

*Friends*

*friends* :  $\mathbb{P}$  *Customer*

*status* : *Status*

*sold* : *Seat*  $\leftrightarrow$  *Customer*

*status* = *premiere*  $\Rightarrow$  *ran sold*  $\subseteq$  *friends*

*EnhancedBoxOffice*  $\hat{=} \hat{=} \text{BoxOffice} \wedge \text{Friends}$

*EnhancedBoxOffice* is equivalent to:

*status* : *Status*

*friends* :  $\mathbb{P}$  *Customer*

*sold* : *Seat*  $\leftrightarrow$  *Customer*

*seating* :  $\mathbb{P}$  *Seat*

*dom sold*  $\subseteq$  *seating*

*status* = *premiere*  $\Rightarrow$  *ran sold*  $\subseteq$  *friends*

## Schema inclusion

An alternative way of adding the declaration and constraint information of a schema is **schema inclusion**.

When a schema name appears in a declaration part of a schema, the result is a merging of declarations and a conjunction of constraints.

## Example

*EnhancedBoxOffice* could have been defined as:

```
EnhancedBoxOffice
BoxOffice
status : Status
friends : P Customer
status = premiere ⇒ ran sold ⊆ friends
```

or even as:

*EnhancedBoxOffice* \_\_\_\_\_  
*BoxOffice*  
*Friends*

## **Change of state**

To describe the effect of an operation, we will include two copies of the state schema: one describing the state before, the other describing the state afterwards.

The second of these will be **decorated** to distinguish it from the first.

## Example

The state of the box office after an operation:

$seating' : \mathbb{P} Seat$ $sold' : Seat \leftrightarrow Customer$ $dom\ sold' \subseteq seating'$
--

## Operation schemas

An operation schema will include two copies of the corresponding state schema:

$Operation$
$State$ $State'$
$\dots$

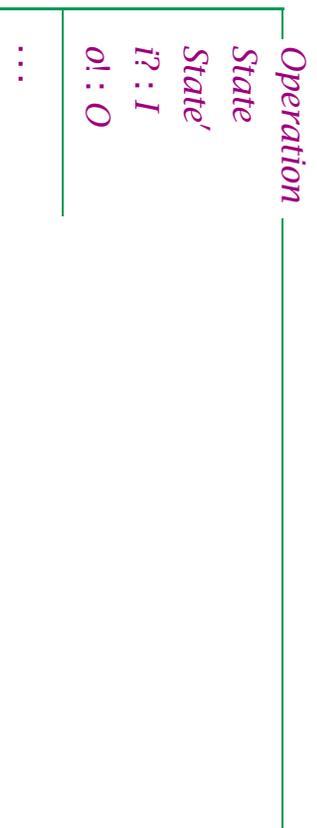
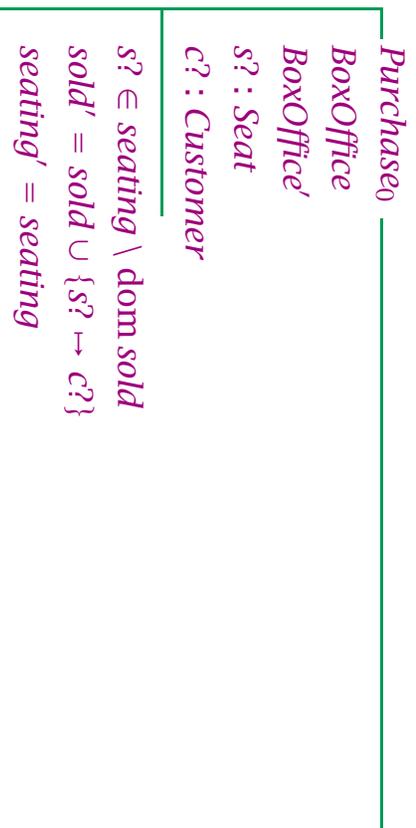
## Example

*Purchase<sub>0</sub>*  
*BoxOffice*  
*BoxOffice'*  
...  
 $s? \in \text{seating} \setminus \text{dom sold}$   
 $\text{sold}' = \text{sold} \cup \{s? \mapsto c?\}$   
 $\text{seating}' = \text{seating}$

## Input and output

An operation may involve inputs and outputs. These are declared in the normal way, although there is a convention regarding their names.

- the name of an input should end in a question mark
- the name of an output should end in an exclamation mark

**Example****Example**

## Delta and Xi

There is another convention regarding operation schemas:

- if a schema describes an operation upon a state described by *State*, we include  $\Delta State$  in its declaration (in place of *State* and *State'*)
- if, in addition, the operation leaves the state unchanged, we include  $\exists State$  (in place of  $\Delta State$ )

The schema  $\Delta Schema$  is equivalent to

*Schema*  
*Schema'*

The schema  $\exists Schema$  is equivalent to

$\Delta Schema$   
 $\theta Schema = \theta Schema'$

**Example**

We could define  $Purchase_0$  as

$Purchase_0$ $\Delta BoxOffice$ $s? : Seat$ $c? : Customer$ $s? \in seating \setminus dom\ sold$ $sold' = sold \cup \{s? \mapsto c?\}$ $seating' = seating$
---

**Example**

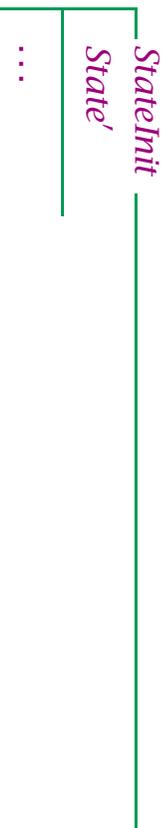
An operation that leaves the box office unchanged:

$QueryAvailability$ $\exists BoxOffice$ $available! : \mathbb{N}$ $available! = \#(seating \setminus dom\ sold)$
---

## Initialisation

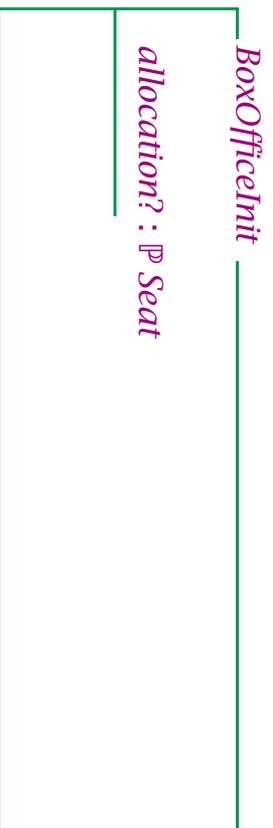
An **initialisation** is a special operation for which the before state is unimportant.

Such an operation can be modelled by an operation schema that contains only a decorated copy of the state:



## Question

How might we complete the following?

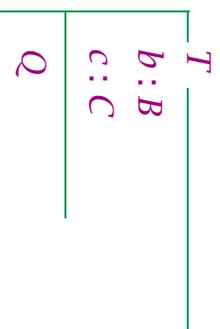
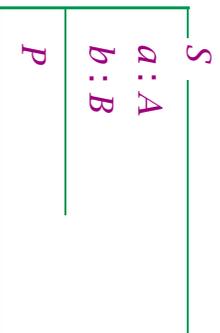


## Schema disjunction

If  $S$  and  $T$  are two schemas, then their disjunction  $S \vee T$  is a schema

- whose declaration is a merge of the two declarations
- whose constraint is a disjunction of the two constraints

## Example



The schema  $S \vee T$  is equivalent to

$a : A$
$b : B$
$c : C$
$P \vee Q$

### Example

If we define

$NotAvailable$
$BoxOffice$
$s? : Seat$
$s? \notin seating \setminus dom\ sold$

then the disjunction

$Purchase_0 \vee NotAvailable$

describes a **total** operation.

## Constructing operations

Although disjunction is the obvious operator for constructing operation schemas, conjunction can also be useful.

## Example

*Response ::= okay | sorry*

*Success*

*r! : Response*

*r! = okay*

*Failure*

*r! : Response*

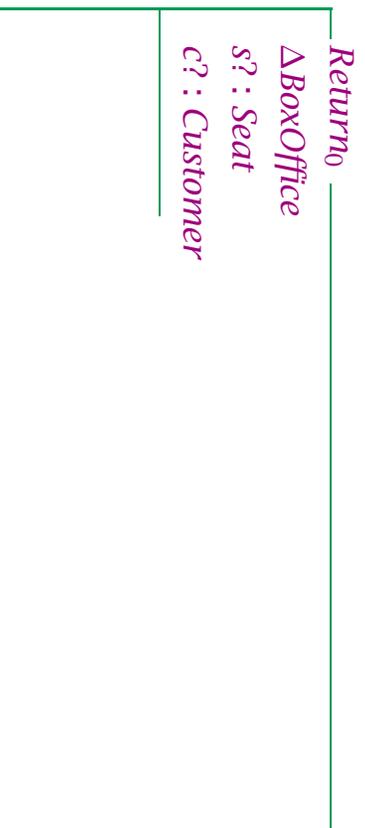
*r! = sorry*

The operation of purchasing a seat may be described by

$$\begin{aligned} \text{Purchase} &\hat{=} (\text{Purchase}_0 \wedge \text{Success}) \\ &\vee \\ &(\text{NotAvailable} \wedge \text{Failure}) \end{aligned}$$

## Question

How might we describe the operation of successfully returning a ticket to the box office?



## Question

How might we describe the operation of unsuccessfully returning a ticket?

*NotPossible*  
*EBoxOffice*  
*s? : Seat*  
*c? : Customer*

## Schema negation

¬ *Schema* is the schema that declares the same identifiers as *Schema*, but imposes the negation of the constraint.

Any constraint information in the declaration part is also negated, so it may be useful to consider the normalised form of the schema prior to negation.

**Example**

If *Schema* is defined by

$$\begin{array}{l} \text{Schema} \\ a : \mathbb{Z} \\ c : \mathbb{P}\mathbb{Z} \\ c \neq \emptyset \wedge a \in c \end{array}$$

then  $\neg$  *Schema* is equivalent to

$$\begin{array}{l} a : \mathbb{Z} \\ c : \mathbb{P}\mathbb{Z} \\ c = \emptyset \vee a \notin c \end{array}$$

**Question**

The schema *ChangedBoxOffice* describes the states of a box office that is not in the initial state. That is, a box office in which at least one ticket has been sold, or the allocation of seating has changed.

How might we define *ChangedBoxOffice*?

## Schema quantification

If  $a$  is declared in schema  $S$  with type  $A$ , and  $S$  has predicate part  $P$ , then

$$\mathcal{Q}a : A \bullet S$$

is a schema in which

- $a$  is removed from the declaration part
- the predicate part is  $\mathcal{Q}a : A \bullet P$

## Example

$\forall a : \mathbb{Z} \bullet$  Schema is equivalent to

$c : \mathbb{P} \mathbb{Z}$
$\forall a : \mathbb{Z} \bullet$
$c \neq \emptyset \wedge a \in c$

**Example**

$\exists a : \mathbb{Z} \bullet \text{Schema}$  is equivalent to

$$c : \mathbb{P} \mathbb{Z}$$
$$\exists a : \mathbb{Z} \bullet$$
$$c \neq \emptyset \wedge a \in c$$
**Example**

Recall the definition of the enhanced box office:

$$\text{EnhancedBoxOffice}$$
$$\text{status} : \text{Status}$$
$$\text{friends} : \mathbb{P} \text{Customer}$$
$$\text{sold} : \text{Seat} \leftrightarrow \text{Customer}$$
$$\text{seating} : \mathbb{P} \text{Seat}$$
$$\text{dom sold} \subseteq \text{seating}$$
$$\text{status} = \text{premiere} \Rightarrow \text{ran sold} \subseteq \text{friends}$$

$\forall$  *status* : *Status* • *EnhancedBoxOffice* is equivalent to

*friends* :  $\mathbb{P}$  *Customer*

*sold* : *Seat*  $\leftrightarrow$  *Customer*

*seating* :  $\mathbb{P}$  *Seat*

$\text{dom sold} \subseteq \text{seating}$

$\text{ran sold} \subseteq \text{friends}$

## Question

What about  $\exists$  *status* : *Status* • *EnhancedBoxOffice*?

## Hiding

Schema existential quantification is suggestive of the removal of objects from an interface.

If  $a$  has type  $A$ , then we may write the schema

$$\exists a : A \bullet S$$

as

$$S \setminus (a)$$

## Question

What kind of operation is described by  $Return_0 \setminus (c?)$  ?

## Schema composition

If two schemas describe operations upon the same state, then we can construct an operation schema that describes the effect of one followed by the other.

In a schema composition, the after state of the first operation is identified with the before state of the second.

## Example

If *OpOne* and *OpTwo* describe operations on *State*, then their composition *OpOne* ; *OpTwo* is equivalent to the schema

$$\begin{aligned} &\exists State'' \bullet \\ &OpOne[\theta State' / \theta State'] \\ &\wedge \\ &OpTwo[\theta State'' / \theta State] \end{aligned}$$

**Example**

If *OpOne* and *OpTwo* are defined by

<i>OpOne</i>	_____
	$a, a' : A$
	$b, b' : B$
	_____
	$P$
	_____

<i>OpTwo</i>	_____
	$a, a' : A$
	$b, b' : B$
	_____
	$Q$
	_____

then their composition is equivalent to the schema

$a, a' : A$	
$b, b' : B$	
_____	
$\exists a'' : A; b'' : B \bullet$	
$P[a'/a', b''/b'] \wedge Q[a''/a, b''/b]$	
_____	

## Question

Is the following a valid inference?

*Purchase<sub>0</sub> § Return*  
*EBoxOffice*

## Summary

- merging declarations
- conjunction and inclusion
- decoration and initialisation
- negation
- quantification
- composition