

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

S
$a : A$
$b : B$
P

T
$b : B$
$c : C$
Q

The schema $S \wedge T$ is equivalent to

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

Example

BoxOffice

sold : *Seat* \rightarrow *Customer*

seating : \mathbb{P} *Seat*

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

Status ::= standard | premiere

Friends

friends : \mathbb{P} *Customer*

status : *Status*

sold : *Seat* \rightarrow *Customer*

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

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

EnhancedBoxOffice is equivalent to:

status : *Status*

friends : \mathbb{P} *Customer*

sold : *Seat* \leftrightarrow *Customer*

seating : \mathbb{P} *Seat*

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

$\textit{status} = \textit{premiere} \Rightarrow \text{ran } \textit{sold} \subseteq \textit{friends}$

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 : \mathbb{P} *Customer*

status = premiere \Rightarrow ran *sold* \subseteq *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.

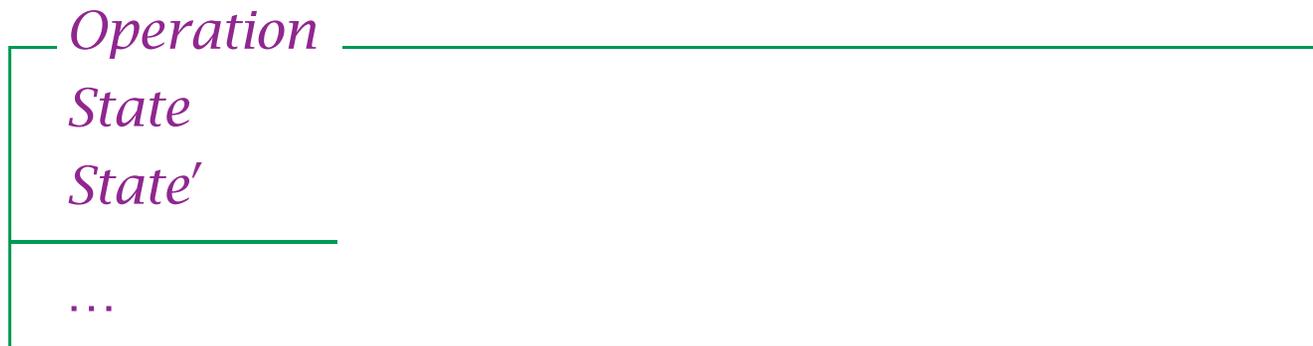
Example

The state of the box office after an operation:

$$\begin{array}{l} \textit{seating}' : \mathbb{P} \textit{Seat} \\ \textit{sold}' : \textit{Seat} \rightarrow \textit{Customer} \end{array}$$
$$\text{dom } \textit{sold}' \subseteq \textit{seating}'$$

Operation schemas

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



Example

Purchase₀

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

Operation

State

State'

i? : I

o! : O

...

Example

Purchase₀

BoxOffice

BoxOffice'

s? : Seat

c? : Customer

s? ∈ seating \ dom sold

sold' = sold ∪ {s? ↦ c?}

seating' = seating

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 $\Xi State$ (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

$Purchase_0$

$\Delta BoxOffice$

$s? : Seat$

$c? : Customer$

$s? \in seating \setminus \text{dom } sold$

$sold' = sold \cup \{s? \mapsto c?\}$

$seating' = seating$

Example

An operation that leaves the box office unchanged:

QueryAvailability

\exists *BoxOffice*

available! : \mathbb{N}

available! = $\#(\textit{seating} \setminus \textit{dom sold})$

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?

BoxOfficeInit _____

allocation? : P Seat

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

S
$a : A$
$b : B$
P

T
$b : B$
$c : C$
Q

The schema $S \vee T$ is equivalent to

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

Example

If we define

NotAvailable

$\exists \text{BoxOffice}$

$s? : \text{Seat}$

$s? \notin \text{seating} \setminus \text{dom sold}$

then the disjunction

$\text{Purchase}_0 \vee \text{NotAvailable}$

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

$$\mathit{Purchase} \hat{=} (\mathit{Purchase}_0 \wedge \mathit{Success})$$

$$\vee$$

$$(\mathit{NotAvailable} \wedge \mathit{Failure})$$

Question

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

Return₀

ΔBoxOffice

s? : Seat

c? : Customer

Question

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

NotPossible

\exists *BoxOffice*

s? : *Seat*

c? : *Customer*

Schema negation

\neg *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

Schema

$a : \mathbb{Z}$

$c : \mathcal{P}\mathbb{Z}$

$c \neq \emptyset \wedge a \in c$

then \neg *Schema* is equivalent to

$a : \mathbb{Z}$

$c : \mathcal{P}\mathbb{Z}$

$c = \emptyset \vee a \notin c$

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

$\exists a : \mathbb{Z} \bullet$ 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:

EnhancedBoxOffice

status : *Status*

friends : \mathbb{P} *Customer*

sold : *Seat* \rightarrow *Customer*

seating : \mathbb{P} *Seat*

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

$\textit{status} = \textit{premiere} \Rightarrow \text{ran } \textit{sold} \subseteq \textit{friends}$

$\forall status : Status \bullet EnhancedBoxOffice$ is equivalent to

$friends : \mathbb{P} Customer$

$sold : Seat \rightarrow Customer$

$seating : \mathbb{P} Seat$

$dom\ sold \subseteq seating$

$ran\ sold \subseteq friends$

Question

What about $\exists \textit{status} : \textit{Status} \bullet \textit{EnhancedBoxOffice}$?

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 $\text{Return}_0 \setminus (c?)$?

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 $OpOne \circ OpTwo$ is equivalent to the schema

$$\begin{array}{l} \exists State'' \bullet \\ \quad OpOne[\theta State'' / \theta State'] \\ \quad \wedge \\ \quad OpTwo[\theta State'' / \theta State] \end{array}$$

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?

$$\frac{\textit{Purchase}_0 \textit{ } \textit{Return}}{\textit{BoxOffice}}$$

Summary

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