

# Preconditions

Using Z

Woodcock & Davies

## Analysis

We may wish to show that

- the requirements are consistent: the constraint part of the state schema is satisfiable
- each operation is applied within its domain: the effect of the operation is properly defined whenever it is used

In each case, it is enough to consider **preconditions**.

## Preconditions

The precondition of an operation is that constraint which is necessary and sufficient for the operation to be defined: that is, for an after state to exist.

The nature of the after state does not concern us; neither do the outputs of the operation. The precondition will take the form of a constraint upon the combination of the before state and the inputs.

## Precondition schemas

A precondition schema is a schema that characterises the combinations of before states and inputs for which the effect of an operation is defined.

<i>State</i> <i>inputs</i>
...

## Notation

If the schema *Operation* describes an operation upon *State*, with a list of *outputs*, then we write *pre Operation* to denote its precondition.

$$\text{pre } Operation = \exists State' \bullet Operation \setminus outputs$$

## Example

pre  $Purchase_0$

=  $\exists \text{BoxOffice}' \bullet \text{Purchase}_0$  [definition of pre ]

= [ $\text{BoxOffice}; s? : \text{Seat};$  [definition of  $\text{Purchase}_0$ ]

$c? : \text{Customer} \mid$

$\exists \text{seating}' : \mathbb{P} \text{Seat} \bullet$

$\exists \text{sold}' : \text{Seat} \rightarrow \text{Customer} \bullet$

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

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

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

$\text{seating}' = \text{seating}]$

$$\begin{aligned}
&= [\textit{BoxOffice}; s? : \textit{Seat}; && \text{[one-point rule, twice]} \\
&\quad c? : \textit{Customer} \mid \\
&\quad \text{dom}(\textit{sold} \cup \{s? \mapsto c?\}) \subseteq \textit{seating} \wedge \\
&\quad s? \in \textit{seating} \setminus \text{dom } \textit{sold}] \\
&= [\textit{BoxOffice}; s? : \textit{Seat}; && \text{[property of 'dom']} \\
&\quad c? : \textit{Customer} \mid \\
&\quad s? \in \textit{seating} \setminus \text{dom } \textit{sold}]
\end{aligned}$$

## Initialisation

The operation of initialisation is a special case; there is no before state, although there may be inputs:

The statement that initialisation is possible is sometimes called the **initialisation theorem**:

$$\exists \textit{State}' \bullet \textit{StateInit} \setminus \textit{outputs}$$

## Example

*BoxOfficeInit*

*BoxOffice'*

*allocation?* :  $\mathbb{P}$  *Seat*

*seating'* = *allocation?*

*sold'* =  $\emptyset$

$\exists \text{BoxOffice}' \bullet \text{BoxOfficeInit}$

$\Leftrightarrow \exists \text{BoxOffice}' \bullet$  [definition of *BoxOfficeInit*]

$[\text{BoxOffice}'; \text{allocation?} : \mathbb{P} \text{Seat} \mid$

$\text{seating}' = \text{allocation?} \wedge$

$\text{sold}' = \emptyset]$

$\Leftrightarrow [\text{allocation?} : \mathbb{P} \text{Seat} \mid$  [schema quantification]

$\exists \text{BoxOffice}' \bullet$

$\text{seating}' = \text{allocation?} \wedge$

$\text{sold}' = \emptyset]$

$\Leftrightarrow$  [*allocation?* :  $\mathbb{P}$  *Seat* |                      [definition of *BoxOffice'*]  
     $\exists$  *seating'* :  $\mathbb{P}$  *Seat* •  
         $\exists$  *sold'* : *Seat*  $\rightarrow$  *Customer* •  
             $\text{dom } \textit{sold}' \subseteq \textit{seating}' \wedge$   
             $\textit{seating}' = \textit{allocation}' \wedge$   
             $\textit{sold}' = \emptyset$ ]



## Explicit vs implicit preconditions

There is a minor advantage to be gained by concentrating upon what an operation is supposed to do, and calculating its precondition later.

Even where an explicit precondition has been included, the calculation provides for a degree of cross-checking.

## Example

$capacity : \mathbb{N}$

$capacity > 0$

$CarPark$

$count : \mathbb{N}$

$count \leq capacity$

*Enter*<sub>0</sub>

$\Delta CarPark$

$$count' = count + 1$$

*Exit*<sub>0</sub>

$\Delta CarPark$

$$count' = count - 1$$

$$\begin{aligned} & \text{pre } \textit{Exit}_0 \\ &= \exists \textit{CarPark}' \bullet \textit{Exit}_0 && \text{[definition of } \textit{Exit}_0\text{]} \\ &= [\textit{CarPark} \mid && \text{[definition of } \textit{CarPark}'\text{]} \\ & \quad \exists \textit{count}' : \mathbb{N} \mid \\ & \quad \quad \textit{count}' \leq \textit{capacity} \bullet \\ & \quad \quad \textit{count}' = \textit{count} - 1] \\ &= [\textit{CarPark} \mid \textit{count} - 1 \in \mathbb{N}] && \text{[one-point rule]} \end{aligned}$$

Informed design:

*ExtraCar*

$\exists$ *CarPark*

*r!* : *Report*

*count* = 0

*r!* = *extra\_car*

*Exit*  $\hat{=}$  *Exit*<sub>0</sub>  $\vee$  *ExtraCar*

## A recipe for preconditions

Suppose that we wish to calculate the precondition of

*Operation*

*Declaration*

*Predicate*

## Step One

Take the various clauses of *Declaration* and assemble them to make three new declarations:

- *Before* introducing only inputs and before components (unprimed state components);
- *After* introducing only outputs and after components (primed state components);
- *Mixed* consisting of the remaining clauses.

## Step Two

If *Mixed* is not an empty declaration, expand every schema mentioned in *Mixed*; add all input and before components to *Before*; add all output and after components to *After*.

As there may be several levels of schema inclusion, repeat this step until there are no clauses left in *Mixed*.

## Step Three

The precondition of *Operation* is then

*Before*

$\exists$  *After* •

*Predicate*

## Question

Given the following schema definitions,

$S$
$a : \mathbb{N}$
$b : \mathbb{N}$
$a \neq b$

$T$
$S$
$c : \mathbb{N}$
$b \neq c$

what is the precondition of the following operation?

*Increment*

$\Delta T$

$in? : \mathbb{N}$

$out! : \mathbb{N}$

$a' = a + in?$

$b' = b$

$c' = c$

$out! = c$

## Simplification

Suppose that we wished to simplify the precondition schema

*Before*

$\exists$  *After* •

*Predicate*

## Step Four

Expand any schemas in *After* that contain equations identifying outputs or after components.

## Step Five

Expand any schemas in *After* that refer to outputs or after components for which we already have equations.

## Step Six

If *Predicate* contains an equation identifying a component declared in *After*, then use the one-point rule to eliminate that component.

Repeat this step as many times as possible.

## Step Seven

If  $After_1$  and  $Predicate_1$  are what remains of  $After$  and  $Predicate$ , then the precondition is now

*Before*

$\exists After_1 \bullet$

*Predicate*<sub>1</sub>

## Question

How may we simplify the predicate part of *pre Increment*?

$$\exists out! : \mathbb{N}; T' \bullet$$

$$a' = a + in? \wedge$$

$$b' = b \wedge$$

$$c' = c \wedge$$

$$out! = c$$

## Disjunction

If

$$Op \hat{=} Op_1 \vee Op_2$$

then

$$\text{pre } Op = \text{pre } Op_1 \vee \text{pre } Op_2$$

## Conjunction

In general, if

$$Op \hat{=} Op_1 \wedge Op_2$$

then

$$\text{pre } Op \neq \text{pre } Op_1 \wedge \text{pre } Op_2$$

However, it may be that the declarations introduce disjoint sets of variables...

## Example

*Success*

*r! : Response*

*r! = okay*

$$\text{pre } (Purchase_0 \wedge Success) = \text{pre } Purchase_0$$

## Free promotion

$$\begin{array}{ccc} \exists \textit{Local}' \bullet & \Leftrightarrow & \forall \textit{Local}' \bullet \\ \quad \exists \textit{Global}' \bullet \textit{Promote} & & \exists \textit{Global}' \bullet \textit{Promote} \end{array}$$

## A useful result

$\text{pre } GOp$

$$\Leftrightarrow \exists \text{Global}' \bullet \\ GOp$$

[definition of 'pre']

$$\Leftrightarrow \exists \text{Global}' \bullet \\ \exists \Delta \text{Local} \bullet \text{Promote} \wedge \text{LOp}$$

[definition of  $GOp$ ]

$$\Leftrightarrow \exists \Delta \text{Local} \bullet \\ \exists \text{Global}' \bullet \text{Promote} \wedge \text{LOp}$$

[property of  $\exists$ ]

$$\Leftrightarrow \exists \textit{Local} \bullet \quad \text{[definition of } \Delta \text{]} \\ \exists \textit{Local}' \bullet \exists \textit{Global}' \bullet \textit{Promote} \wedge \textit{LOp}$$

$$\Leftrightarrow \exists \textit{Local} \bullet \quad \text{[lemma]} \\ (\exists \textit{Local}' \bullet \exists \textit{Global}' \bullet \textit{Promote}) \wedge (\exists \textit{Local}' \bullet \textit{LOp})$$

$$\Leftrightarrow \exists \textit{Local} \bullet \quad \text{[definition of 'pre', twice]} \\ \textit{pre Promote} \wedge \textit{pre LOp}$$

## Lemma

The equivalence labelled ‘lemma’ is easily proved in the forward direction. A proof in the other direction ( $\Leftarrow$ ) requires the free promotion property.

We abbreviate *Local'*, *Global'*, and *Promote* to *L'*, *G'*, and *P*, respectively.



## Example

$AssignIndex \hat{=} \exists \Delta Data \bullet AssignData \wedge Promote$

$pre\ AssignIndex =$

$\exists Data \bullet pre\ AssignData \wedge pre\ Promote$

## Question

What is the precondition of *AssignData*?

*AssignData*

$\Delta Data$

*new?* : *Value*

*value'* = *new?*

## Question

What is the precondition of *Promote*?

*Promote*

$\Delta Array$

$\Delta Data$

$index? : \mathbb{N}$

$index? \in \text{dom } array$

$\{index?\} \triangleleft array = \{index?\} \triangleleft array'$

$array\ index? = \theta Data$

$array'\ index? = \theta Data'$

## Question

What is the precondition of

$$\exists \Delta Data \bullet Promote \wedge AssignData \quad ?$$

## Results

It is often useful to tabulate the results of our analysis; against each operation, we record the predicate that characterises its precondition.

We should check that the predicates are a correct reflection of our expectations: that each operation schema is exactly as prescriptive as it should be.

## Example

<i>InitBoxOffice</i>	<i>true</i>
<i>Purchase<sub>0</sub></i>	$s? \in \text{seating} \setminus \text{dom sold}$
<i>NotAvailable</i>	$s? \notin \text{seating} \setminus \text{dom sold}$
<i>Purchase</i>	<i>true</i>
<i>Return<sub>0</sub></i>	$s? \mapsto c? \in \text{sold}$
<i>NotPossible</i>	$s? \mapsto c? \notin \text{sold}$
<i>Return</i>	<i>true</i>

## Summary

- preconditions
- pre *Schema*
- initialisation
- calculation and simplification
- disjunction
- promotion