Logic Specification and Z Schema

3K04

McMaster

Basic Logic Operators

- Logical negation (¬)
- Logical conjunction (Λ or &)
- Logical disjunction (V or ||)
- Logical implication (→)
- Logical equality (= or ↔)

Logic Negation

- **NOT p** (also written as **¬p**)

p	$\neg p$
1	0
0	1

Logic Conjunction

- **p AND q** (also written as **p** $\wedge$ **q**, **p & q**, or **p·q**)

p	q	$p \cdot q$
1	1	1
1	0	0
0	1	0
0	0	0

Logic Disjunction

- **p OR q** (also written as **p** $\vee$ **q** or **p + q**)

p	q	$p + q$
1	1	1
1	0	1
0	1	1
0	0	0

Logic Implication

- **p implies q** (also written as **p → q**, **not p or q**)

p	q	$p \to q$
T	T	T
T	F	F
F	T	T
F	F	T

Logic Equality

- **p EQ q** (also written as **p = q**, **p ↔ q**, or **p ≡ q**)

p	q	$p \equiv q$
T	T	T
T	F	F
F	T	F
F	F	T

First-order logic

- While propositional logic deals with simple declarative propositions, first-order logic additionally covers predicates and quantification.

- Each interpretation of first-order logic includes a domain of discourse over which the quantifiers range.

Predicate

- A predicate resembles a function that returns either True or False.
- Consider the following sentences: "Socrates is a philosopher", "Plato is a philosopher".
- In propositional logic these are treated as two unrelated propositions, denoted for example by *p* and *q*.
- In first-order logic, however, the sentences can be expressed in a more parallel manner using the predicate Phil(*a*), which asserts that the object represented by *a* is a philosopher.

Quantifier

- $\forall$ - universal quantifier; $\exists$ - existential quantifier
- Let Phil(*a*) assert *a* is a philosopher and let Schol (*a*) assert that *a* is a scholar
- For every *a*, if *a* is a philosopher then *a* is a scholar. $\forall a(\mathrm{Phil}(a) \rightarrow \mathrm{Schol}(a))$
- If *a* is a philosopher then *a* is a scholar.

$$\exists a(\mathrm{Phil}(a) \wedge \neg \mathrm{Schol}(a)).$$

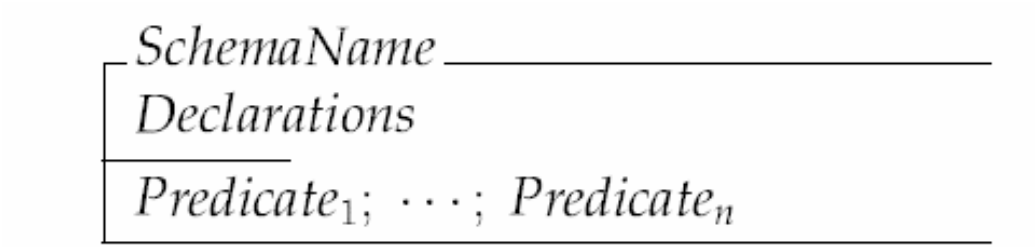
Z notation

- The **Z notation**, is a formal specification language used for describing and modeling computing systems.

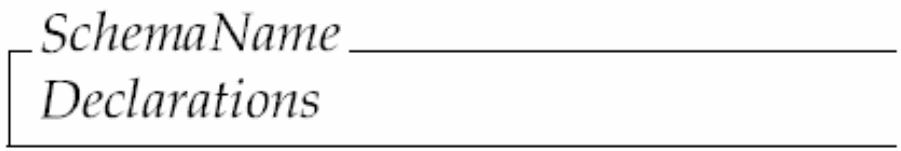
- It is targeted at the clear specification of computer programs and the formulation of proofs about the intended program behavior.

Z Schemas

- The Z schema is a graphical notation for describing
  - State spaces
  - operations



or of the form

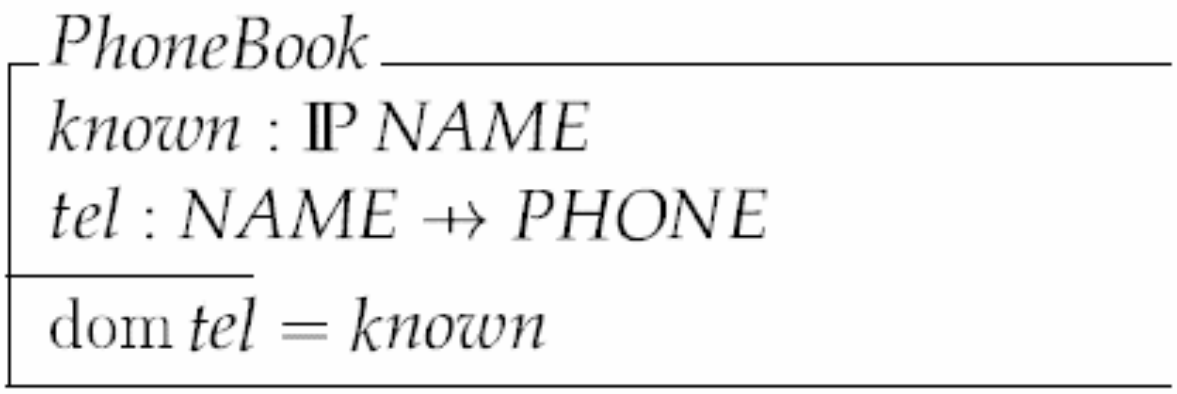


- The declarations part of the schema will contains:
  - A list of variable declarations
  - References to other schemas (schema inclusion)

- The predicate part of a schema contains a list of predicates, separated either by semi-colons or new lines.

State Space Schemas

- Here is an example state-space schema, representing part of a system that records details about the phone numbers of staff.

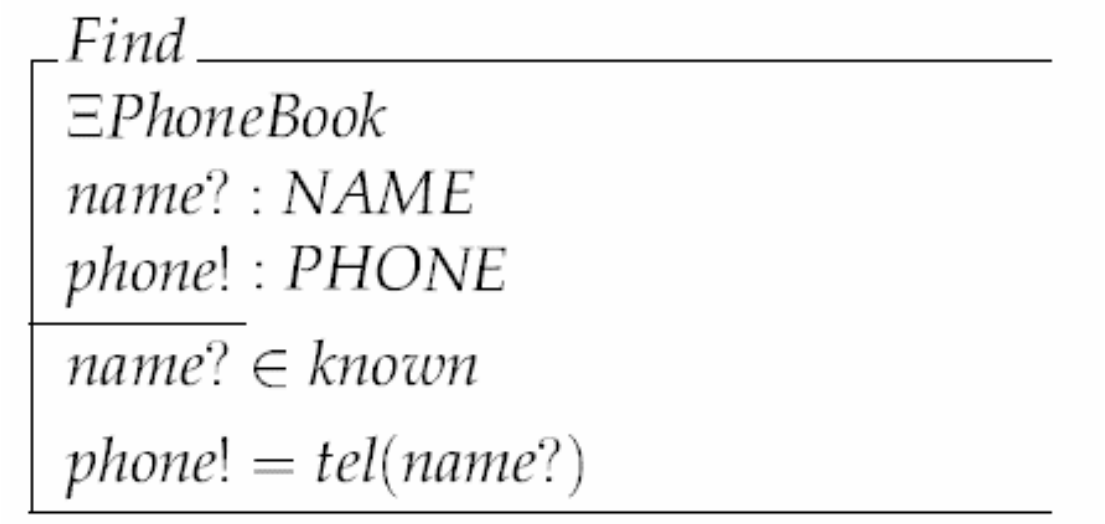


Assume that NAME is a set of names, and PHONE is a set of phone numbers.

Operation Schemas

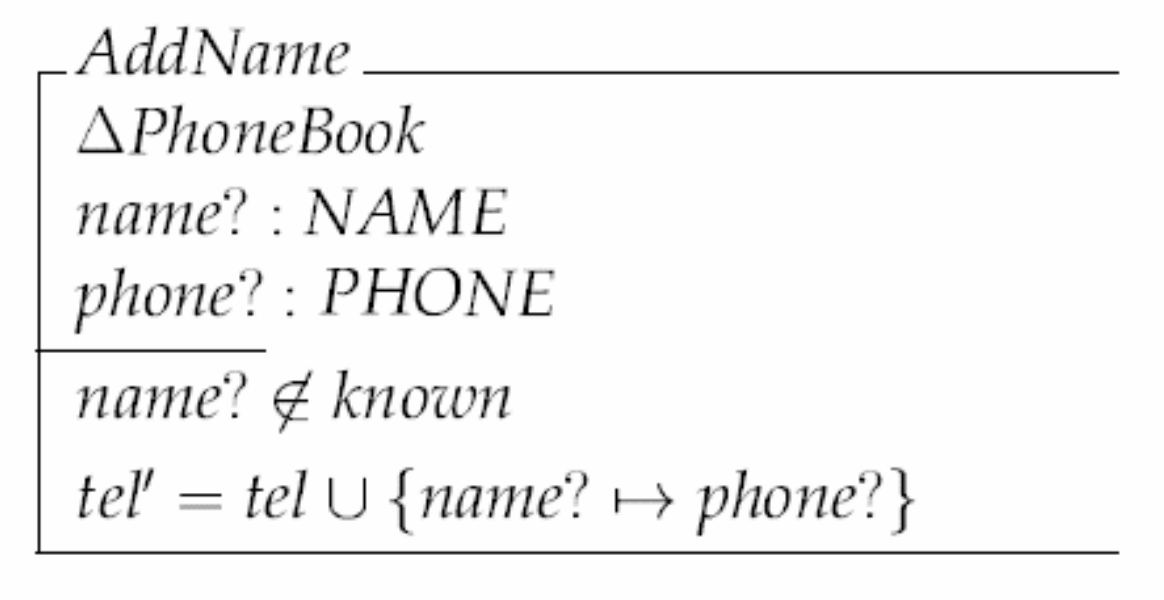
- In specifying a system operation, we must consider:
  - The objects that are accessed by the operation;
  - The pre-conditions of the operation, i.e., the things that must be true for the operation to succeed;
  - The post-conditions, i.e., the things that will be true after the operation, if the pre-condition was satisfied before the operation.

- Consider the ‘lookup’ operation: input a name, output a phone number.
  - This operation accesses the PhoneBook schema;
  - It does not change it;
  - It takes a single ‘input’, and produces a single output;
  - Pre-condition: the name is known to the database.



- This illustrates the following Z conventions:
  - Placing the name of the schema in the declaration part ‘includes’ that schema;
  - ‘input’ variable names are terminated by a question mark;
  - ‘output’ variables are terminated by an exclamation mark;
  - The Ξ (Xi) symbol means that the PhoneBook schema is not changed; if we write a Δ (delta) instead, it would mean that the PhoneBook schema did change.

- Add a name/phone pair to the phone book.



- Appending a ‘ to a variable means “the variable after the operation is performed”.

Example: Video Rental Shop

$[VIDEO]$

Video_shop

$$all\_videos, in\_stock, booked\_out : P\ \ VIDEO$$

$$in\_stock \cup booked\_out = all\_videos$$

Returning a video

$$
\begin{array}{|l}
\hline
Video\_returned \\
\Delta Video\_shop \\
video? : VIDEO \\
\hline
video? \in booked\_out \\
booked\_out' = booked\_out - \{video?\} \\
in\_stock' = in\_stock \cup \{video?\} \\
all\_videos' = all\_videos \\
\hline
\end{array}
$$

Removal of a video from the stock

$$
\begin{array}{|l}
\hline
\textit{RemoveVideo} \\
\Delta \textit{Video\_shop} \\
\textit{video?} : \textit{VIDEO} \\
\hline
\textit{video?} \in \textit{all\_videos} \\
\textit{all\_videos}' = \textit{all\_videos} - \{\textit{video?}\} \\
\textit{in\_stock}' = \textit{in\_stock} - \{\textit{video?}\} \\
\textit{booked\_out}' = \textit{booked\_out} - \{\textit{video?}\} \\
\hline
\end{array}
$$

Find a video

$$MESSAGE ::= is\_in\_stock \mid is\_booked\_out$$

FindVideo

$\Xi Video\_shop$

$video? : VIDEO$

$message! : MESSAGE$

$video? \in all\_videos$

$video? \in in\_stock \Rightarrow message! = is\_in\_stock$

$video? \notin in\_stock \Rightarrow message! = is\_booked\_out$

List all videos

Example: Sale Theater tickets

$[Seat]$
$[Person]$

*TicketsForPerformance*0

$seating : \mathbb{P}\, Seat$
$sold : Seat \pfun Person$

$\mathbf{dom}\, sold \subseteq seating$

- Informal specification
  - Theater: Tickets for the first night are only sold to friends

- Specification in Z

$$Status ::= standard \mid firstNight$$

Friends

$$friends : \mathbb{P}\, Person$$
$$status : Status$$
$$sold : Seat \pfun Person$$

$$status = firstNight \Rightarrow \mathbf{ran}\, sold \subseteq friends$$

$$TicketsForPerformance1 \mathrel{\widehat{=}} TicketsForPerformance0 \land Friends$$

and

$$\begin{array}{|l} TicketsForPerformance1 \\ \hline Friends \\ TicketsForPerformance0 \\ \hline \end{array}$$

$$TicketsForPerformance1 \mathrel{\widehat=} TicketsForPerformance0 \land Friends$$

and

$$\begin{array}{|l} \hline TicketsForPerformance1 \\ \hline Friends \\ TicketsForPerformance0 \\ \hline \end{array}$$

are the same as

$$\begin{array}{|l} \hline TicketsForPerformance1 \\ \hline friends : \mathbb{P}\, Person;\ status : Status \\ sold : Seat \pfun Person;\ seating : \mathbb{P}\, Seat \\ \hline status = firstNight \Rightarrow \mathbf{ran}\, sold \subseteq friends \\ \mathbf{dom}\, sold \subseteq seating \\ \hline \end{array}$$

Selling tickets

$$
\begin{array}{|l}
\textit{Purchase}0 \\
\hline
\textit{TicketsForPerformance}0 \\
\textit{TicketsForPerformance}0' \\
s? : \textit{Seat} \\
p? : \textit{Person} \\
\hline
s? \in \textit{seating} \setminus \mathbf{dom}\, \textit{sold} \\
\textit{sold}' = \textit{sold} \cup \{s? \mapsto p?\} \\
\textit{seating}' = \textit{seating} \\
\hline
\end{array}
$$

$$Response ::= okay \mid sorry$$

Success

$$r! : Response$$

$$r! = okay$$

Then

$$Purchase0 \land Success$$

is a schema that reports successful ticket sale.

Selling tickets, but only to friends if first night performance

*Purchase*1

$\Delta TicketsForPerformance1$

$s? : Seat$

$p? : Person$

$s? \in seating \setminus \mathbf{dom}\, sold$

$status = firstNight \Rightarrow (p? \in friends)$

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

$seating' = seating$

$status' = status$

$friends' = friends$

NotAvailable

$\Xi TicketsForPerformance1$

$s? : Seat$

$p? : Person$

$s? \in \mathbf{dom}\, sold \lor (status = firstNight \land \neg\, p? \in friends)$

Failure

$r! : Response$

$r! = sorry$

$$TicketServiceForPerformance \mathrel{\widehat=} (Purchase1 \land Success) \lor (NotAvailable \land Failure)$$

Composition of Operation Schemas

$$Purchase1 \fatsemi Purchase1[s?/s2?]$$

is equivalent to

$$\Delta TicketsForPerformance1$$
$$s? : Seat;\ s2? : Seat;\ p? : Person$$

$$s? \in seating \setminus \mathbf{dom}\, sold$$
$$s2? \in seating \setminus \mathbf{dom}(sold \cup \{s? \mapsto p?\})$$
$$status = firstNight \Rightarrow (p? \in friends)$$
$$sold' = sold \cup \{s? \mapsto p?, s2? \mapsto p?\}$$
$$seating' = seating$$
$$status' = status$$
$$friends' = friends$$

Example: Order Invoicing

$[OrderId, Product]$

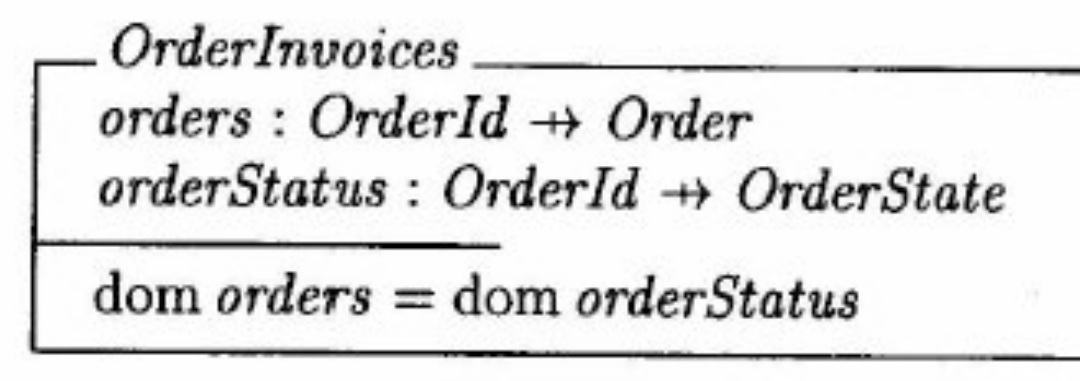
$OrderState ::= pending \mid invoiced$

Stock

$stock : \text{bag } Product$

$Order == \{order : \text{bag } Product \mid order \neq \varnothing\}$

Order Invoices



State

State

$$Stock$$
$$OrderInvoices$$
$$newids : \mathbb{P}\ OrderId$$

$$\text{dom}\ orders \cap newids = \varnothing$$

InitState

$$State'$$

$$stock' = \varnothing$$
$$orders' = \varnothing$$
$$newids' = OrderId$$

$\Delta State$

$State$

$State'$

$newids' = newids \setminus \text{dom } orders'$

$InvoiceOrder$

$\Delta State$

$id? : OrderId$

$orders(id?) \sqsubseteq stock$

$orderStatus(id?) = pending$

$stock' = stock \uplus orders(id?)$

$orders' = orders$

$orderStatus' = orderStatus \oplus \{id? \mapsto invoiced\}$

$Report ::= OK \mid order\_not\_pending \mid not\_enough\_stock \mid no\_more\_ids$

$Success$

$rep! : Report$

$rep! = OK$

InvoiceError

$\Xi State$

$id? : OrderId$

$rep! : Report$

$orderStatus(id?) \neq pending$

$rep! = order\_not\_pending$

StockError

$\Xi State$

$id? : OrderId$

$rep! : Report$

$\neg\ orders(id?) \sqsubseteq stock$

$rep! = not\_enough\_stock$

A total operation for ordering

$$InvoiceOrderOp == (InvoiceOrder \land Success) \lor InvoiceError \lor StockError$$