Logic Specification and Z Schema

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 ∧ 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 V 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** $\leftrightarrow$ **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

SchemaName

Declarations

$Predicate_1; \cdots ; Predicate_n$

or of the form

SchemaName

Declarations

- 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.

PhoneBook

$$
\begin{array}{|l}
\hline
\mathit{known} : \mathbb{P}\, \mathit{NAME} \\
\mathit{tel} : \mathit{NAME} \pfun \mathit{PHONE} \\
\hline
\operatorname{dom} \mathit{tel} = \mathit{known} \\
\hline
\end{array}
$$

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.

Find

$$\Xi PhoneBook$$
$$name? : NAME$$
$$phone! : PHONE$$

$$name? \in known$$
$$phone! = tel(name?)$$

- 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.

$$
\begin{array}{|l}
\hline
\textit{AddName} \\
\Delta \textit{PhoneBook} \\
\textit{name?} : \textit{NAME} \\
\textit{phone?} : \textit{PHONE} \\
\hline
\textit{name?} \notin \textit{known} \\
\textit{tel}' = \textit{tel} \cup \{\textit{name?} \mapsto \textit{phone?}\} \\
\hline
\end{array}
$$

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

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

OrderInvoices

$$
\begin{aligned}
&orders : OrderId \mathrel{+\!\!\!\!\!\rightarrow} Order \\
&orderStatus : OrderId \mathrel{+\!\!\!\!\!\rightarrow} OrderState
\end{aligned}
$$

$$
\operatorname{dom} orders = \operatorname{dom} orderStatus
$$

State

State

$Stock$

$OrderInvoices$

$newids : \mathbb{P}\ OrderId$

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

InitState

$State'$

$stock' = \varnothing$

$orders' = \varnothing$

$newids' = OrderId$

$$\begin{array}{|l}
\hline
\Delta State \\
State \\
State' \\
\hline
newids' = newids \setminus \mathrm{dom}\ orders' \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline
InvoiceOrder \\
\Delta State \\
id? : OrderId \\
\hline
orders(id?) \sqsubseteq stock \\
orderStatus(id?) = pending \\
stock' = stock \uplus orders(id?) \\
orders' = orders \\
orderStatus' = orderStatus \oplus \{id? \mapsto invoiced\} \\
\hline
\end{array}$$

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

$$\begin{array}{|l}
\hline
Success \\
rep! : Report \\
\hline
rep! = OK \\
\hline
\end{array}$$

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\mathit{InvoiceError}} \\
\Xi State \\
id? : OrderId \\
rep! : Report \\
\hline
orderStatus(id?) \neq pending \\
rep! = order\_not\_pending \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\mathit{StockError}} \\
\Xi State \\
id? : OrderId \\
rep! : Report \\
\hline
\neg\ orders(id?) \sqsubseteq stock \\
rep! = not\_enough\_stock \\
\hline
\end{array}
$$

A total operation for ordering

$$\begin{aligned}&InvoiceOrderOp == \\ &\quad (InvoiceOrder \land Success) \lor InvoiceError \lor StockError\end{aligned}$$

Z

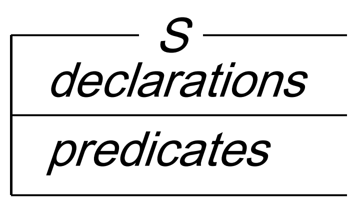
- Z is a popular formal specification language.
- Use of Z requires knowledge of set theory, functions and discrete mathematics, including first-order logic.
- Like the earlier formal systems examined, there are several variants of Z.

Z

- In its simplest form a Z specification consists of 4 sections:
  1. Given sets, data types, and constants
  2. State definitions
  3. Initial state
  4. Operations
- It uses standard set and logic operators: $\exists$, $\supset$, $\Rightarrow$
- It uses some additional symbols such as $\oplus$, $\mapsto$, $\lhd$.

Z

- Given sets
  - A Z specification begins with a list of given sets. These are sets that need not be defined in detail.
  - The names of given sets appears in brackets, e.g., [Button] to represent the set of buttons in the elevator problem.



Z

- State definition
  - A Z specification consists of a number of schemata.
  - Each schemata consists of a group of variable declarations and a list of predicates that constrain the values of the variables.
  - A schemata S is defined as:

```
┌──── S ─────
│ declarations
├────────────
│ predicates
└────────────
```

Z

- For the elevator problem, there are four subsets of Button:
  - floor buttons
  - elevator buttons
  - buttons (the set of all buttons in the elevator problem)
  - pushed (the set of buttons that have been pushed and are therefore on).
- The symbol $\mathbb{P}$ denotes powerset (the set of all subsets of a given set).

Z

Button_State

floor_button elevator_button:	$\mathbb{P}$ Button
buttons:	$\mathbb{P}$ Button
pushed:	$\mathbb{P}$ Button

floor_button ∩ elevator_button = ∅

floor_button ∪ elevator_button = buttons

Z

- Initial state
  - The abstract initial state describes the state of the system when it is first turned on.
  - For the elevator problem the initial state is:

    $Button\_init \mathrel{\widehat=} [Button\_State' \mid pushed' = \emptyset]$
  - This is a vertical schema definition (as opposed to a horizontal one).
  - The initial state above tells us that when the elevator is first turned on, the set pushed is initially empty; that is, all buttons are off.

Z

- Operations
  - If a button is pushed for the first time, then that button is turned on and added to the set pushed.
  - The Δ in the first line of the schema tells us that operation changes the state of Button_State.
  - The operation has one input variable, button?. The ? symbol denotes an input variable, whereas the ! symbol denotes an output variable.

Push_Button

Δ*Button_State*

button?: Button

$(button? \in buttons) \wedge$

$(((button? \notin pushed) \wedge (pushed' = pushed \cup \{button?\})) \vee$

$((button? \in pushed) \wedge (pushed' = pushed)))$

Z

- The predicate part of the operation consists of a group of preconditions that must hold before the operation is invoked, and a group of postconditions that must hold after the operation is complete.
- If the operation is invoked without the preconditions being satisfied, then unspecified (read unpredictable) results occur.
- pushed′ denotes an updated value of pushed.

Z

- When an elevator arrives at a floor, if the corresponding floor button is on, then it must be turned off, similarly for the corresponding elevator button.
- The symbol \ denotes set difference
- The solution below is an oversimplification on that it does not distinguish between up and down floor buttons.

Floor_Arrival

Δ*Button_State*
button?: Button

$(\text{button?} \in \text{buttons}) \land$
$(((\text{button?} \in \text{pushed}) \land (\text{pushed}' = \text{pushed} \setminus \{\text{button?}\})) \lor$
$((\text{button?} \notin \text{pushed}) \land (\text{pushed}' = \text{pushed})))$

Z

- Z is fairly widely used and has been employed on some large scale projects.

1. Easy to find faults in Z specifications, especially during inspections of the specification, and inspection of the design and/or code against the formal specification.
2. Need to be precise when using Z, resulting in fewer ambiguities, contradictions and omissions than an informal specification.
3. Can use Z to perform a formal proof of correctness.
4. Professionals with high school math can be taught to write Z specifications (although more mathematical sophistication is required to do the proof).
5. Use of Z decreases software cost by decreasing overall development time.
6. Natural language specifications (for the customer for example) derived from a Z specification has fewer problems than a natural language description written from scratch without a formal description to act as a guide.

Z, Petri nets and beyond

- Z, like Petri nets, is still undergoing development.
- Researchers are constantly extending Z and Petri nets to provide greater functionality, power and flexibility.
- There are temporal versions of Z which allow the predicates to cover time dependant semantics.
- There are many other formalisms including:
  - Anna
  - Gist
  - VDM
  - CSP
  - Hoare axiomatics
  - Operational semantics
  - Denotational semantics
  - Larch
  - OBJ
  - Lotos