The Specification Language

Z
Characterization

- Formal specification of abstract data types
- Model-oriented specification
- Data types are defined with the help of sets, relations, and functions
- Operations are specified with pre- and postconditions
- Proofs based on logic and set theory
- Specifications may be refined in an evolutionary way
Introduction into the Z Notation
Survey

- $\mathbb{Z}$ is based on set theory and predicate logic
- Sets may be defined in the following ways:
  - Extensional: Enumeration of elements
  - Intensional: Specification of a predicate
- Operations on sets: union, intersection, ...
- Base types for sets:
  - Pre-defined type $\mathbb{Z}$ (for integers)
  - User-defined types (abstract data types)
- Type constructors:
  - Power set: $\mathcal{P} X$ denotes the set of all subsets of $X$
  - Cartesian product: $X \times Y$ is the set of all pairs $(x, y)$, where $x \in X$ and $y \in Y$
- Strong typing:
  - All elements of a set must have the same type
  - Operations require operands of the same type
Running example: library

- A library lends books to readers
- For each book, there may be one or more copies
- Only registered users may borrow books from the library
- There is a maximal number of copies which may be issued to one user
- Operations:
  - Stock administration (addition and removal of copies)
  - User administration (registration and deregistration of users)
  - Issue (lending and returning of book copies)
## Examples of base types and constructed types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Book, Copy, Reader]</td>
<td>User-defined base types for books, copies, and readers</td>
</tr>
<tr>
<td>( \mathbb{Z} )</td>
<td>Set of integers</td>
</tr>
<tr>
<td>( \mathbb{P} \mathbb{Z} )</td>
<td>Set of all subsets of integers</td>
</tr>
<tr>
<td>( \mathbb{F} \mathbb{Z} )</td>
<td>Set of all finite subsets of ( \mathbb{Z} )</td>
</tr>
<tr>
<td>Book ( \times ) Copy</td>
<td>Set of all pairs of books and copies</td>
</tr>
</tbody>
</table>
### Examples for the definition of sets

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}</td>
<td>Extensional definition of the set of integers from 1 to 10</td>
</tr>
<tr>
<td>1..10</td>
<td>Interval notation</td>
</tr>
<tr>
<td>{n : \mathbb{Z} \mid 1 \leq n \land n \leq 10}</td>
<td>Intensional definition of the set of integers from 1 to 10</td>
</tr>
<tr>
<td>{1, 4, 9, 16, 25, 36, 49, 64, 81, 100}</td>
<td>Extensional definition of the square numbers (1^2 \ldots 10^2)</td>
</tr>
<tr>
<td>{n : \mathbb{Z} \mid 1 \leq n \land n \leq 10 \cdot n^2}</td>
<td>Intensional definition of the square numbers (1^2 \ldots 10^2)</td>
</tr>
</tbody>
</table>
### Elements of Z specifications

<table>
<thead>
<tr>
<th>Variable declarations</th>
<th>Predicates</th>
<th>Constant definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>readers : ( F ) Reader</td>
<td>stock : ( F (Copy \times Book) )</td>
<td>( \mathbb{N} ) == { ( n \in \mathbb{Z} \mid n \geq 0 ) }</td>
</tr>
<tr>
<td>shelved : ( F ) Copy</td>
<td>issued : ( F (Copy \times Reader) )</td>
<td>( \mathbb{N}_1 ) == { ( n \in \mathbb{Z} \mid n &gt; 0 ) }</td>
</tr>
<tr>
<td>stock : ( F (Copy \times Book) )</td>
<td>max : ( \mathbb{Z} )</td>
<td></td>
</tr>
<tr>
<td>max ( \geq 0 )</td>
<td>( #stock \leq \text{max} )</td>
<td></td>
</tr>
<tr>
<td>( \forall ) c : Copy; b(_1), b(_2) : Book ( \bullet )</td>
<td>( (c, b(_1)) \in s \land (c, b(_2)) \in s \Rightarrow b(_1) = b(_2) )</td>
<td></td>
</tr>
<tr>
<td>( \mathbb{N} ) == { ( n \in \mathbb{Z} \mid n \geq 0 ) }</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \mathbb{N}_1 ) == { ( n \in \mathbb{Z} \mid n &gt; 0 ) }</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Type concept

- Types are maximal sets (e.g., $\mathbb{Z}$)
- Predicates for restricting these sets do not modify the type (e.g., $\mathbb{N}$ does not define a new type)
- Sets constrained by predicates may be used in variable declarations
  - Example:
    
    \[
    \text{max : } \mathbb{N} \text{ stands for max : } \mathbb{Z}; \text{ max } \geq 0
    \]
- Strong typing: The operands of an operator (e.g., $\cup$) must have the same type
## Notations for quantification and sets

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall \ Decs \bullet Pred$</td>
<td>$Pred$ holds for all objects in $Decs$</td>
</tr>
<tr>
<td>$\forall \ Decs \mid Constr \bullet Pred =$</td>
<td>$Pred$ holds for all objects in $Decs$ meeting the constraint $Constr$</td>
</tr>
<tr>
<td>$\forall \ Decs \bullet Constr \Rightarrow Pred$</td>
<td>There is an object in $Decs$ which meets the predicate $Pred$</td>
</tr>
<tr>
<td>$\exists \ Decs \bullet Pred$</td>
<td>There is an object in $Decs$ which meets both the constraint $Constr$ and the predicate $Pred$</td>
</tr>
<tr>
<td>${Decs \mid Pred}$</td>
<td>Set of all objects in $Decs$ which meet the predicate $Pred$</td>
</tr>
<tr>
<td>${Decs \mid Pred \bullet Expr}$, e.g. ${n : \mathbb{Z} \mid 1 \leq n \land n \leq 10 \bullet n^2}$</td>
<td>Set of all values of all expressions $Expr$, where variables range over objects from $Decs$ satisfying the predicate $Pred$</td>
</tr>
</tbody>
</table>
Definition of enumeration types

\[ \text{BookKind} ::= \text{hardcover} \mid \text{paperback} \]

stands for

\[
\begin{align*}
\text{[BookKind]} \\
\text{hardcover, paperback} : \text{BookKind} \\
\text{hardcover} \neq \text{paperback} \\
\forall \ bk : \text{BookKind} \implies bk = \text{hardcover} \lor bk = \text{paperback}
\end{align*}
\]
There are polymorphic operators, which may be applied to operands of different types.

Such operators may be defined as generic.

Unconstrained genericity: any type may replace a generic parameter.

Example:

\[
\forall S, T : \mathbb{P}_X S \subseteq T \iff (\forall x : X \bullet x \in S \Rightarrow x \in T)
\]
### (Binary) relations

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X \leftrightarrow Y = \mathcal{P}(X \times Y)$</td>
<td>Relation between $X$ and $Y$</td>
</tr>
<tr>
<td>$x \leftrightarrow y = (x, y)$</td>
<td>Pairs</td>
</tr>
<tr>
<td>$\text{dom } R = { x : X \mid \exists y : Y \bullet x \leftrightarrow y \in R }$</td>
<td>Domain</td>
</tr>
<tr>
<td>$\text{ran } R = { y : Y \mid \exists x : X \bullet x \leftrightarrow y \in R }$</td>
<td>Range</td>
</tr>
<tr>
<td>$S \ll R = { x : X; y : Y \mid x \in S \land x \leftrightarrow y \in R \bullet x \leftrightarrow y }$</td>
<td>Domain restriction</td>
</tr>
<tr>
<td>$R \triangleright T = { x : X; y : Y \mid y \in T \land x \leftrightarrow y \in R \bullet x \leftrightarrow y }$</td>
<td>Range restriction</td>
</tr>
<tr>
<td>$S \lll R = { x : X; y : Y \mid x \notin S \land x \leftrightarrow y \in R \bullet x \leftrightarrow y }$</td>
<td>Domain subtraction</td>
</tr>
<tr>
<td>$R \gg T = { x : X; y : Y \mid y \notin T \land x \leftrightarrow y \in R \bullet x \leftrightarrow y }$</td>
<td>Range subtraction</td>
</tr>
<tr>
<td>$R^{-1} = { x : X; y : Y \mid x \leftrightarrow y \in R \bullet y \leftrightarrow x }$</td>
<td>Inverse relation</td>
</tr>
</tbody>
</table>
| $R \triangleleft S =$ \[
\begin{align*}
\{ & x : X; y : Y; z : Z \mid x \leftrightarrow y \in R \land y \leftrightarrow z \in S \bullet x \leftrightarrow z \} \\
\end{align*}
| Composition |
### Functions (1)

<table>
<thead>
<tr>
<th>Type</th>
<th>Symbol</th>
<th>(\text{dom } f)</th>
<th>injective</th>
<th>(\text{ran } f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial</td>
<td>(\mapsto)</td>
<td>(\subseteq X)</td>
<td>(\subseteq Y)</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>(\rightarrow)</td>
<td>(= X)</td>
<td>(\subseteq Y)</td>
<td></td>
</tr>
<tr>
<td>Partial and injective</td>
<td>(\mapsto)</td>
<td>(\subseteq X)</td>
<td>(+)</td>
<td>(\subseteq Y)</td>
</tr>
<tr>
<td>Total and injective</td>
<td>(\mapsto)</td>
<td>(= X)</td>
<td>(+)</td>
<td>(\subseteq Y)</td>
</tr>
<tr>
<td>Partial and surjective</td>
<td>(\mapsto)</td>
<td>(\subseteq X)</td>
<td></td>
<td>(= Y)</td>
</tr>
<tr>
<td>Total and surjective</td>
<td>(\rightarrow)</td>
<td>(= X)</td>
<td></td>
<td>(= Y)</td>
</tr>
<tr>
<td>Bijective</td>
<td>(\mapsto)</td>
<td>(= X)</td>
<td>(+)</td>
<td>(= Y)</td>
</tr>
<tr>
<td>Partial and finite</td>
<td>(\mapsto)</td>
<td>(\subseteq X)</td>
<td></td>
<td>(\subseteq Y)</td>
</tr>
<tr>
<td>Partial, finite, injective</td>
<td>(\mapsto)</td>
<td>(\subseteq X)</td>
<td>(+)</td>
<td>(\subseteq Y)</td>
</tr>
</tbody>
</table>
### Functions (2)

\[ f : X \leftrightarrow Y \text{ is a function } \iff \forall x : X; y : Y; z : Z \mid x \mapsto y \in f \land x \mapsto z \in f \implies y = z \]

- Functions are unique relations

<table>
<thead>
<tr>
<th>( fx )</th>
<th>Application of a function ( f ) to an argument ( x )</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>( \text{dom, } \llhd, \gg, \ll, \gg, f^{-1}, f \circ g )</th>
<th>Operations which are “inherited” from relations</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>( \lambda x : X \mid \text{Pred } \bullet \text{Term} = { x : X \mid \text{Pred } \bullet x \mapsto \text{Term} } )</th>
<th>Lambda notation for the definition of functions</th>
</tr>
</thead>
</table>

| \( f \oplus g = ((\text{dom } g) \llhd f) \cup g \) | Combination of functions (\( g \) wins in case of a conflict) |
# Sequences

<table>
<thead>
<tr>
<th>Notation for sequences</th>
<th>Formal definition of sequences</th>
<th>Example</th>
<th>Non-empty sequences</th>
<th>Head and tail of a non-empty sequence</th>
<th>Concatenation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle\text{Reagan, Bush, Clinton, Bush}\rangle$</td>
<td>$\text{seq } X == { f : \mathbb{N} \rightarrow X \mid \text{dom } f = 1 .. #f }$</td>
<td>$\langle\text{Reagan, Bush, Clinton, Bush}\rangle = {1 \mapsto \text{Reagan}, 2 \mapsto \text{Bush}, 3 \mapsto \text{Clinton}, 4 \mapsto \text{Bush}}$</td>
<td>$\text{seq}_1 X == \text{seq } X \setminus { \langle \rangle }$</td>
<td>$\forall s : \text{seq}_1 X \bullet$ $\text{head } s = s 1 \land \text{tail } s = \lambda n : 1 .. #s - 1 \bullet s (n + 1)$</td>
<td>$\forall s, t : \text{seq } X \bullet$ $s \sim t = s \cup { n : 1 .. #t \bullet (n + #s) \mapsto t n }$</td>
<td>$\langle\text{Reagan, Bush}\rangle \sim \langle\text{Clinton, Bush}\rangle = \langle\text{Reagan, Bush, Clinton, Bush}\rangle$</td>
</tr>
</tbody>
</table>
Schemata
On schemata

- Schemata are specification units
- A schema consists of a set of declarations and a set of (conjunctive) predicates
- Schemata may be combined with the help of several operations, including e.g. schema inclusion, schema conjunction and schema disjunction
- Data types are specified in a model-oriented way as follows:
  - There is one schema for defining the representation of the data type (state) and the respective state invariants
  - For each operation, there is one corresponding schema which defines its input and output behavior as well as the state changes affected by the operation
Schema for the state and its invariants

Library

stock : Copy \rightarrow Book
issued : Copy \rightarrow Reader
shelved : F Copy
readers : F Reader

\begin{align*}
\text{shelved } \cup \text{ dom issued} &= \text{ dom stock} \\
\text{shelved } \cap \text{ dom issued} &= \emptyset \\
\text{ran issued } &\subseteq \text{ readers} \\
\forall r : \text{ readers } \bullet \#(\text{issued } \triangleright \{ r \}) &\leq \text{maxloans}
\end{align*}
Schema for an operation

- An operation is defined by a schema which has to obey certain conventions (i.e., Z does not introduce special-purpose “operation schemata”)
- The operation is not declared explicitly!
- Operation name = Schema name
- Parameter:
  - \( x \): Input parameter
  - \( y \): Output parameter
- States:
  - \( s \): “Before” state of an operation
  - \( s' \): “After” state of an operation
- All declarations and predicates for \( s \) and \( s' \) must be repeated in the operation schema
- To be introduced: Short-hand notation
Example: Lending a book

\textbf{Issue}

\begin{align*}
\text{stock}, \text{stock}^\prime & : \text{Copy} \leftrightarrow \text{Book} \\
\text{issued}, \text{issued}^\prime & : \text{Copy} \leftrightarrow \text{Reader} \\
\text{shelved}, \text{shelved}^\prime & : \mathbb{F} \text{Copy} \\
\text{readers}, \text{readers}^\prime & : \mathbb{F} \text{Reader} \\
\text{c}^? & : \text{Copy}; \text{r}^? & : \text{Reader}
\end{align*}

\begin{align*}
\text{shelved} \cup \text{dom issued} & = \text{dom stock} \\
\text{shelved}^\prime \cup \text{dom issued}^\prime & = \text{dom stock}^\prime \\
\text{shelved} \cap \text{dom issued} & = \emptyset; \text{shelved}^\prime \cap \text{dom issued}^\prime = \emptyset \\
\text{ran} & \text{issued} \subseteq \text{readers}; \text{ran} \text{issued}^\prime \subseteq \text{readers}^\prime \\
\forall r : \text{readers} & \bullet \#(\text{issued} \triangleright \{r\}) \leq \text{maxloans} \\
\forall r : \text{readers}^\prime & \bullet \#(\text{issued}^\prime \triangleright \{r\}) \leq \text{maxloans} \\
\text{c}^? & \in \text{shelved}; \text{r}^? \in \text{readers} ; \#(\text{issued} \triangleright \{r\}) < \text{maxloans} \\
\text{issued}^\prime & = \text{issued} \oplus \{\text{c}^? \mapsto \text{r}^?\}; \text{stock}^\prime = \text{stock}; \text{readers}^\prime = \text{readers} \\
\text{shelved}^\prime & = \text{shelved} \setminus \{\text{c}^?\}
\end{align*}
Schema operators (1)

Schema inclusion

Schema decoration and $\Delta$ schema
Schema operators (2)

\[ S_1 \quad S_2 \]
\[ \text{Decl}_1 \quad \text{Decl}_2 \]
\[ \text{Pred}_1 \quad \text{Pred}_2 \]

Schema disjunction
\[ S_1 \lor S_2 \]
\[ \text{Decl}_1 \cup \text{Decl}_2 \]
\[ \text{Pred}_1 \lor \text{Pred}_2 \]

Schema conjunction
\[ S_1 \land S_2 \]
\[ \text{Decl}_1 \cup \text{Decl}_2 \]
\[ \text{Pred}_1 \land \text{Pred}_2 \]
Schema operators (3)

Schema composition

» Given: Schemata for two operations $\text{Op}_1$ and $\text{Op}_2$ on the same state $\text{State}$

» Schema composition describes the sequential application of $\text{Op}_1$ and $\text{Op}_2$:

$$\Rightarrow \text{Op}_1[/''] \text{ denotes the schema which is derived from } \text{Op}_1 \text{ by replacing variables } v' \text{ with } v''$$

$$\Rightarrow \text{Op}_2[/''] \text{ denotes the schema which is derived from } \text{Op}_2 \text{ by replacing variables } v \text{ with } v'$$

$$\Rightarrow \text{Op}_1 \circ \text{Op}_2 = \exists \text{State''} \bullet \text{Op}_1[/''] \land \text{Op}_2[/'']$$

Precondition

» Let $\text{Op}$ be a schema for an operation on state $\text{State}$ with output variables $\text{Outs}!$.

» $\text{pre } \text{Op}$ returns the precondition under which $\text{Op}$ is applicable:

$$\Rightarrow \text{pre } \text{Op} = \exists \text{State'}; \text{Outs}! \bullet \text{Op}$$
Example of a schema inclusion

\[\text{LibDB}\]
\[
\begin{align*}
\text{stock} & : \text{Copy} \leftrightarrow \text{Book} \\
\text{readers} & : \mathbb{F} \text{Reader}
\end{align*}
\]

\[\text{LibLoans}\]
\[
\begin{align*}
\text{issued} & : \text{Copy} \leftrightarrow \text{Reader} \\
\text{shelved} & : \mathbb{F} \text{Copy} \\
\text{shelved} \cap \text{dom issued} & = \emptyset \\
\forall r : \text{readers} & \cdot \#(\text{issued} \upharpoonright \{ r \}) \leq \text{maxloans}
\end{align*}
\]

\[\text{Library}\]
\[
\begin{align*}
\text{LibDB} \\
\text{LibLoans}
\end{align*}
\]
\[
\begin{align*}
\text{shelved} \cup \text{dom issued} & = \text{dom stock} \\
\text{ran issued} & \subseteq \text{readers}
\end{align*}
\]
Specification of a change operation with a $\Delta$ schema

**Issue**

$\Delta$Library

$c? : Copy; r? : Reader$

\[
\begin{align*}
  c? \in shelved; r? \in readers; \ &(\#(issued \triangleright \{ r \}) < maxloans) \\
  issued' = issued \oplus \{ c? \mapsto r? \}; \ &stock' = stock; \ &readers' = readers \\
  shelved' = shelved \setminus \{ c? \}
\end{align*}
\]}
Specification of a read operation with a $\exists$ schema

$\exists Library$
$
\Delta Library$

$NoChange \equiv$
$issued' = issued; stock' = stock;$
$shelved' = shelved; readers' = readers$

$WhoHasCopy$
$\exists Library$
$c? : Copy; r! : Reader$

$c? \in \text{dom issued}; r! = \text{issued c?}$
Example of a schema disjunction (1)

AddKnownTitle

$\Delta Library$
$b? : Book$
$rep! : Report$

$b? \in \text{ran stock}$
$\exists c : Copy \mid c \not\in \text{dom stock}$ •

$stock' = stock \oplus \{ c \mapsto b? \} \land$
$shelved' = shelved \cup \{ c \}$

$issued' = issued; \ readers' = reader$
$rep! = \text{FurtherCopyAdded}$

AddNewTitle

$\Delta Library$
$b? : Book$
$rep! : Report$

$b? \not\in \text{ran stock}$
$\exists c : Copy \mid c \not\in \text{dom stock}$ •

$stock' = stock \oplus \{ c \mapsto b? \} \land$
$shelved' = shelved \cup \{ c \}$

$issued' = issued; \ readers' = reader$
$rep! = \text{NewTitleAdded}$

AddCopy $\equiv$ AddKnownTitle $\lor$ AddNewTitle
Example of a schema disjunction (2)

AddCopy

ΔLibrary

\( b? : Book \)

\( rep! : Report \)

\[ \exists c : Copy \mid c \notin \text{dom } stock \bullet \]

\[ \text{stock}' = \text{stock} \oplus \{ c \mapsto b? \} \land \]

\[ \text{shelved}' = \text{shelved} \cup \{ c \} \]

\[ \text{issued}' = \text{issued}; \text{readers}' = \text{reader} \]

\( b? \in \text{ran } stock \Rightarrow rep! = \text{FurtherCopyAdded} \)

\( b? \notin \text{ran } stock \Rightarrow rep! = \text{NewTitleAdded} \)
Example of a schema conjunction

EnterNewCopy

$\Delta Library$

$b? : Book$

$\exists c : Copy \mid c \notin \text{dom } stock \bullet$

$stock' = stock \oplus \{ c \mapsto b? \} \land$

$shelved' = shelved \cup \{ c \}$

$issued' = issued; readers' = readers$

AddCopyReport

$\text{Stock} : Copy \rightarrow Book$

$b? : Book$

$\text{rep!} : \text{Report}$

$b? \notin \text{ran } stock$

$\Rightarrow \text{rep!} = \text{NewTitleAdded}$

$b? \in \text{ran } stock$

$\Rightarrow \text{rep!} = \text{FurtherCopyAdded}$

$AddCopy \equiv \text{EnterNewCopy} \land \text{AddCopyReport}$
Sample Specification
Example: Electronic dictionary

- Translation between two languages, called *Native* and *Foreign*
- Only orthographically correct words may be stored in the dictionary (*OrthoNative* and *OrthoForeign*, respectively)
- Each word of the native language is mapped onto a set of words of the foreign language (and vice versa)
- Operations to be provided:
  - Insertion of a valid pair
  - Output of all translations of a native word
  - Output of all translations of a foreign word
  - Testing the knowledge of a user:
    - System selects a word randomly
    - User supplies his translations
    - System calculates the percentage of correct answers
Structure of the specification

- Base types and global definitions
- Abstract states
- Initialization
- Partial operations under normal conditions
- Calculation of preconditions
- Total operations (including error conditions)
- Summary and index
Syntax of Z

<specification> ::= (<paragraph>)* (* Specification consists of paragraphs *)

<paragraph> ::= "[" <ident> ("," <ident>)* "]" (* Base types *)
| <axiomatic-box> (* Declarations plus optional predicates *)
| <generic-box> (* ... plus generic parameters *)
| <schema-box> (* "graphical" schema definition *)
| <schema-name> [<gen-formals>] ≡ <schema-expr>
  (* linear schema definition *)
| <def-lhs> "==" <expr> (* Constant declaration *)
| <ident> "::=" <branch> ("|" <branch>)+ (* Enumeration type *)
| <predicate> (* Predicate for global variables *)
Base types and global definitions

\[ \text{[Native, Foreign]} \]

(* All character strings in the respective alphabets *)

\[ \text{OrthoNative} : \mathbb{P} \text{ Native} \]
\[ \text{OrthoForeign} : \mathbb{P} \text{ Foreign} \]

(* Orthographically correct words *)

\[ \text{Message ::= Ok} \mid \text{AlreadyKnownPair} \mid \text{NewPairEntered} \]
\[ \mid \text{ErrorInForeignWord} \mid \text{ErrorInNativeWord} \mid \text{ErrorInBothWords} \]
\[ \mid \text{UnknownNativeWord} \mid \text{UnknownForeignWord} \]
\[ \mid \text{VocabIsEmpty} \mid \text{NoCorrectResponses} \]

(* Return codes for operations *)
Abstract states

\[\text{WellFormedVocab}\]
\[
\begin{align*}
\text{Vocab} & : \text{OrthoNative} \leftrightarrow \text{OrthoForeign} \\
\text{NativeWordsKnown} : & \mathbb{F} \text{OrthoNative} \\
\text{ForeignWordsKnown} : & \mathbb{F} \text{OrthoForeign}
\end{align*}
\]
\[\text{RecordOfProgress}\]
\[
\begin{align*}
\text{CumuMaxMarks}, \text{CumuMarksScored}, \text{AveragePercent} : & \mathbb{N} \\
0 \leq & \text{AveragePercent} \leq 100 \\
\text{CumuMarksScored} \leq & \text{CumuMaxMarks} \\
\text{AveragePercent} = & \text{percent}(\text{CumuMarksScored}, \text{CumuMaxMarks})
\end{align*}
\]
\[\text{WordForWord}\]
\[
\begin{align*}
\text{WellFormedVocab} \\
\text{RecordOfProgress}
\end{align*}
\]

(* Dictionary *)

(* Testing of user *)

(* Overall state *)
Initialization

\[
\text{InitWord-For-Word}
\]
\[
\text{WordForWord}
\]
\[
\text{Vocab} = \emptyset
\]
\[
\text{CumuMaxMarks} = \text{CumuMaxMarksScored} = 0
\]
Definition of partial operations (1)

AddPair \equiv EnterPair \land ReportIfAlreadyKnown

(* Insertion of a pair into the dictionary with return code *)

EnterPair

\[ \Delta \text{WellFormedVocab} \]

\[ \Xi \text{RecordOfProgress} \]

\[ n? : \text{OrthoNative}; f? : \text{OrthoForeign} \]

\[ Vocab' = Vocab \cup \{ n? \mapsto f? \} \]

(* Insertion of a pair *)

ReportIfAlreadyKnown

\[ Vocab : \text{OrthoNative} \leftrightarrow \text{OrthoForeign} \]

\[ n? : \text{OrthoNative}; f? : \text{OrthoForeign}; rep! : \text{Message} \]

\[ n? \mapsto f? \in Vocab \Rightarrow rep! = \text{AlreadyKnownPair} \]

\[ n? \mapsto f? \notin Vocab \Rightarrow rep! = \text{NewPairEntered} \]

(* Information if pair was already known *)
Definition of partial operations (2)

\[ \text{ToForeign} \equiv \text{ForeignTranslations} \land \text{ReportIfKnownNative} \]

(* Translation of a word with return code *)

\[ \text{ForeignTranslations} \]
\[ \exists \text{WordForWord} \]
\[ n? \colon \text{OrthoNative}; ftrans! : \mathbb{F} \text{OrthoForeign} \]

\[ ftrans! = \text{ran} ( \{ n? \} \triangleleft \text{Vocab} ) \]

(* Retrieval of translations *)

\[ \text{ReportIfKnownNative} \]
\[ \exists \text{WellFormedVocab} \]
\[ n? \colon \text{OrthoNative}; rep! : \text{Message} \]

\[ n? \in \text{NativeWordsKnown} \Rightarrow rep! = \text{Ok} \]
\[ n? \notin \text{NativeWordsKnown} \Rightarrow rep! = \text{UnknownNativeWord} \]

(* Information whether there is a translation for the given word *)
Definition of partial operations (3)

\[ VocabTestNtoF \equiv SelectTestWordN \land CheckResponsesF \land UpdateScoreNtoF \]

(* Test: Select word, check responses, update score *)

SelectTestWordN

\( WellFormedVocab \)
\( TestWord! : OrthoNative \)
\( Translations : F OrthoForeign \)
\( TransCount! : \mathbb{N} \)

\( TestWord! \in NativeWordsKnown \)
\( Translations = \text{ran}(\{TestWord!\} \triangle Vocab) \)
\( TransCount! = \#Translations \)

(* Selection of a word *)
Definition of partial operations (4)

\[
\begin{align*}
\text{CheckResponsesF} & \quad \text{Translations, CorrectResponses! : } F \text{ OrthoForeign} \\
& \quad \text{Responses? : seq Foreign} \\
& \quad \text{rep! : Message} \\
\end{align*}
\]

\[
\begin{align*}
\text{CorrectResponses!} & = \text{Translations} \cap \text{ran Responses?} \\
\text{CorrectResponses!} & = \emptyset \Rightarrow \text{rep!} = \text{NoCorrectResponses} \\
\text{CorrectResponses!} & \neq \emptyset \Rightarrow \text{rep!} = \text{Ok} \\
\end{align*}
\]

(* Correct responses included? *)
Definition of partial operations (5)

\[\text{UpdateScoreNtoF} \]
\[\Xi \text{WellFormedVocab} \]
\[\Delta \text{RecordOfProgress} \]
\[\text{Translations, CorrectResponses!} : \mathbb{F} \text{ OrthoNative} \]
\[\text{TransCount!}, \text{NewAverage!} : \mathbb{N} \]

\[\text{CumuMaxMarks'} = \text{CumuMaxMarks} + \text{TransCount!} \]
\[\text{CumuMarksScored'} = \text{CumuMarksScored} + \#\text{CorrectResponses!} \]
\[\text{NewAverage!} = \text{AveragePercent'} \]

(* Output of the number of correct responses and new average percentage *)
## Calculation of preconditions

<table>
<thead>
<tr>
<th>Operation</th>
<th>Inputs and outputs</th>
<th>Preconditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddPair</td>
<td>$n^? : \text{Native}; f^? : \text{Foreign}$</td>
<td>$n^? \in \text{OrthoNative}$</td>
</tr>
<tr>
<td></td>
<td>$\text{rep}! : \text{Message}$</td>
<td>$f^? \in \text{OrthoForeign}$</td>
</tr>
<tr>
<td>ToForeign</td>
<td>$n^? : \text{Native}$</td>
<td>$n^? \in \text{OrthoNative}$</td>
</tr>
<tr>
<td></td>
<td>$f\text{trans}! : \text{OrthoForeign}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{rep}! : \text{Message}$</td>
<td></td>
</tr>
<tr>
<td>VocabTestNtoF</td>
<td>Responses? : seq $\text{Foreign}$</td>
<td>$\text{Vocab} \neq \emptyset$</td>
</tr>
<tr>
<td></td>
<td>$\text{TestWord}! : \text{OrthoNative}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{CorrectResponses}! : \text{OrthoForeign}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{TransCount}! : \mathbb{N}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{NewAverage}! : \mathbb{N}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{rep}! : \text{Message}$</td>
<td></td>
</tr>
</tbody>
</table>
Total operations (error handling)

\[ \text{TotalAidPair} \equiv \text{AddPair} \lor \text{AddPairError} \]

\[ (* \text{Total operation} = \text{normal operation} + \text{error handling} *) \]

\[ \text{AddPairError} \]

\[ \exists \text{WordForWord} \]

\[ n? : \text{Native}; f? : \text{Foreign}; rep! : \text{Message} \]

\[ n? \in \text{OrthoNative} \land f? \notin \text{OrthoForeign} \]

\[ \Rightarrow \text{rep!} = \text{ErrorInForeignWord} \]

\[ n? \notin \text{OrthoNative} \land f? \in \text{OrthoForeign} \]

\[ \Rightarrow \text{rep!} = \text{ErrorInNativeWord} \]

\[ n? \notin \text{OrthoNative} \land f? \notin \text{OrthoForeign} \]

\[ \Rightarrow \text{rep!} = \text{ErrorInBothWords} \]
AddPair ≡ EnterPair \land ReportIfAlreadyKnown

ToForeign ≡ ForeignTranslations \land ReportIfKnownNative

VocabTestNtoF ≡ SelectTestWordN \land CheckResponsesF \land UpdateScoreNtoF

TotalAddPair ≡ AddPair \lor AddPairError

...
Proving of Specification Properties
Survey

- Foundations for proving specification properties:
  - Proposition logic
  - Predicate logic
  - Set theory

- Example-based demonstration of
  - Correctness of the initial state of a data type
  - Simplification of a precondition of an operation
  - Proving a property of an operation composition

- Not all used axioms will be introduced explicitly

- Example: Administration of soccer fans
  - Each fan is registered under a unique identification number
  - A subset of fans may be banned (hooligans)
  - Operations for inserting, deleting, banning fans, etc.
Z specification of soccer fan administration (1)

\[
\begin{align*}
[\text{PERSON}, \text{ID}] \\
\quad (* \text{Given sets} *)
\end{align*}
\]

\[
\begin{align*}
\text{Fid} & \\
\text{members} : \text{ID} \leftrightarrow \text{PERSON} & \\
\text{banned} : \mathbb{P} \text{ID} & \\
\quad (* \text{State} *)
\end{align*}
\]

\[
\begin{align*}
\text{InitFid} & \\
\text{Fid'} & \\
\quad (* \text{Initial state (without members)} *)
\end{align*}
\]

\[
\begin{align*}
\text{members'} = \emptyset & \\
\text{banned'} = \emptyset & \\
\end{align*}
\]
Z specification of soccer fan administration (2)

AddMember

\[ \Delta Fid \]
applicant? : PERSON
id! : ID

applicant? \( \notin \text{ran members} \)
id! \( \notin \text{dom members} \)
members' = members \( \cup \{ id! \rightarrow \text{applicant?} \} \)
banned' = banned

DeleteMember

\[ \Delta Fid \]
id? : ID

id? \( \in \text{dom members} \)
members' = \{ id? \} \( \triangleleft \) members
banned' = banned \( \setminus \{ id? \} \)

BanMember

\[ \Delta Fid \]
ban? : ID

ban? \( \in \text{dom members} \)
members' = members
banned' = banned \( \cup \{ \text{ban?} \} \)
Correctness of the initial state

\[ \vdash \exists Fid' \bullet InitFid \]

\[ \iff (\text{Substitution of } Fid' \text{ and } InitFid) \]

\[ \vdash \exists \text{members'} : ID \leftrightarrow \text{PERSON}; \ bann\text{ed'} : \mathcal{P} \text{ ID} \mid \]

\[ \quad \text{bann\text{ed'} } \subseteq \text{dom members'} \bullet \]

\[ \quad \text{members'} = \emptyset \land \text{bann\text{ed'} } = \emptyset \]

\[ \iff (\exists \text{Decs} \mid \text{Constr} \bullet \text{Pred} \equiv \exists \text{Decs} \bullet \text{Constr} \land \text{Pred}) \]

\[ \vdash \exists \text{members'} : ID \leftrightarrow \text{PERSON}; \ bann\text{ed'} : \mathcal{P} \text{ ID} \bullet \]

\[ \quad \text{bann\text{ed'} } \subseteq \text{dom members'} \land \text{members'} = \emptyset \land \text{bann\text{ed'} } = \emptyset \]

This proposition holds because:

- \( \emptyset : ID \leftrightarrow \text{PERSON} \)
- \( \emptyset : \mathcal{P} \text{ ID} \)
- \( \emptyset \subseteq \emptyset \)
Simplification of a precondition (1)

\[\begin{align*}
&\text{PreAddMember} \\
&\quad Fid \\
&\quad \text{applicant?} : \text{PERSON} \\
&\quad \exists Fid': id! : ID \bullet \\
&\quad \quad \text{applicant?} \notin \text{ran members} \land \\
&\quad \quad id! \notin \text{dom members} \land \\
&\quad \quad \text{members'} = \text{members} \cup \{id! \mapsto \text{applicant?}\} \land \\
&\quad \quad \text{banned'} = \text{banned}
\end{align*}\]

\[\Leftrightarrow (\text{Expansion of } Fid')\]
Simplification of a precondition (2)

\( PreAddMember \)

\begin{align*}
\text{Fid} \\
\text{applicant? : PERSON} \\
\exists \text{members’ : ID } & \mapsto \text{PERSON}; \text{banned’ : } \mathcal{P} \text{ ID}; \text{id! : ID} \\
\text{banned’ } & \subseteq \text{dom members’ } \land \\
\text{applicant? } & \notin \text{ran members } \land \\
\text{id! } & \notin \text{dom members } \land \\
\text{members’ } & = \text{members } \cup \{ \text{id! } \mapsto \text{applicant? } \} \land \\
\text{banned’ } & = \text{banned}
\end{align*}

\( \iff (\text{Elimination of existential quantifiers for members’ and banned’}) \)
PreAddMember

\[ \exists id! : ID \bullet \]

\[ \text{members} \cup \{ id! \mapsto \text{applicant}\} \in ID \mapsto \text{PERSON} \land \]
\[ \text{banned} \in \mathbb{P} ID \land \]
\[ \text{banned} \subseteq \text{dom members} \cup \{ id! \mapsto \text{applicant}\} \land \]
\[ \text{applicant} \notin \text{ran members} \land \]
\[ id! \notin \text{dom members} \]

\[ \iff (Fid \Rightarrow \text{banned} \in \mathbb{P} ID) \]
Simplification of a precondition (4)

\[ \text{PreAddMember} \]

\[ \text{Fid} \]

\[ \text{applicant} : \text{PERSON} \]

\[ \exists \text{id!} : \text{ID} \quad \bullet \]

\[ \text{members} \cup \{ \text{id!} \mapsto \text{applicant} \} \subseteq \text{ID} \leftrightarrow \text{PERSON} \land \]

\[ \text{banned} \subseteq \text{dom members} \cup \{ \text{id!} \mapsto \text{applicant} \} \land \]

\[ \text{applicant} \notin \text{ran members} \land \]

\[ \text{id!} \notin \text{dom members} \]

\[ \Leftrightarrow (\text{dom (R} \cup \text{S}) = \text{dom R} \cup \text{dom S}) \]
Simplification of a precondition (5)

\[ \text{PreAddMember} \]

\[ \text{Fid} \]

\[ \text{applicant? : PERSON} \]

\[ \exists id! : ID \bullet \]

\[ \text{members} \cup \{ id! \mapsto \text{applicant?} \} \in ID \leftrightarrow \text{PERSON} \land \]

\[ \text{banned} \subseteq \text{dom members} \cup \text{dom} \{ id! \mapsto \text{applicant?} \} \land \]

\[ \text{applicant?} \notin \text{ran members} \land \]

\[ id! \notin \text{dom members} \]

\[ \iff (\text{Fid} \Rightarrow \text{banned} \subseteq \text{dom members}) \]
Simplification of a precondition (6)

\[\text{PreAddMember} \]
\[
\begin{align*}
\text{Fid} & \\
\text{applicant? : PERSON} & \\
\exists \, id! : ID & \\
\quad \text{members} \cup \{ id! \mapsto \text{applicant} \} & \in ID \mapsto \text{PERSON} \land \\
\quad \text{applicant?} & \notin \text{ran members} \land \\
\quad id! & \notin \text{dom members}
\end{align*}
\]

\[\iff (\text{members} \in ID \mapsto \text{PERSON} \land \\
\quad \text{applicant?} \notin \text{ran members} \land \\
\quad id! \notin \text{dom members} \Rightarrow \\
\quad \text{members} \cup \{ id! \mapsto \text{applicant?} \} \in ID \mapsto \text{PERSON})\]
Simplification of a precondition (7)

\[ \text{PreAddMember} \]

\[ \text{Fid} \]

\[ \text{applicant? : PERSON} \]

\[ \exists \text{id! : ID} \quad \bullet \]

\[ \text{applicant?} \notin \text{ran members} \land \]

\[ \text{id!} \notin \text{dom members} \]

\[ \Leftrightarrow \quad \text{(Remove first subexpression from the existential quantifier)} \]
Simplification of a precondition (8)

\[ PreAddMember \]
\[ Fid \]
\[ applicant? : PERSON \]
\[ applicant? \notin ran members \land \exists id! : ID \bullet id! \notin \text{dom members} \]

\[ \Leftrightarrow (\text{Elimination of the existential quantifier}) \]

\[ PreAddMember \]
\[ Fid \]
\[ applicant? : PERSON \]
\[ applicant? \notin ran members \land \text{dom members} \neq ID \]
Proving a property of the specification (1)

Sequential execution of AddMember (with output \textit{id}!) and DeleteMember (with input \textit{id}?) does not change the state:

\[ \text{AddandDelete} \equiv \text{AddMember} \sqsubset \text{DeleteMember} \mid \text{id!} = \text{id}? \uparrow \exists \text{Fid} \]

\begin{align*}
\text{AddandDelete} \\
\Delta \text{Fid} \\
\text{applicant?} : \text{PERSON} \\
\text{id?} : \text{ID}; \text{id!} : \text{ID} \\
\end{align*}

\[ \exists \text{Fid}'' \bullet \\
\text{applicant?} \notin \text{ran members} \land \\
\text{id!} \notin \text{dom members} \land \\
\text{members}'' = \text{members} \cup \{ \text{id!} \mapsto \text{applicant?} \} \land \\
\text{banned}'' = \text{banned} \land \\
\text{id?} \in \text{dom members}'' \land \\
\text{members}' = \{ \text{id?} \} \subseteq \text{members}'' \land \\
\text{banned}' = \text{banned}'' \setminus \{ \text{id?} \} \land \\
\text{id!} = \text{id}? 
\]
Proving a property of the specification (2)

\[ \Leftrightarrow \text{(Expansion of } Fid'' \text{)} \]

\[ \Delta Fid \]
\[ \text{applicant}?: \text{PERSON} \]
\[ i\text{d}? : \text{ID}; i\text{d}! : \text{ID} \]

\[ \exists \text{members}'' : i\text{d} \mapsto \text{PERSON}; \text{banned}'' : P \text{ ID} \bullet \]
\[ \text{banned}'' \subseteq \text{dom members}'' \land \]
\[ \text{applicant} \notin \text{ran members} \land \]
\[ i\text{d}! \notin \text{dom members} \land \]
\[ \text{members}'' = \text{members} \cup \{ i\text{d}! \mapsto \text{applicant} \} \land \]
\[ \text{banned}'' = \text{banned} \land \]
\[ i\text{d}? \in \text{dom members}'' \land \]
\[ \text{members}' = \{ i\text{d}? \} \triangleq \text{members}'' \land \]
\[ \text{banned}' = \text{banned}'' \setminus \{ i\text{d}? \} \land \]
\[ i\text{d}! = i\text{d}? \]
Proving a property of the specification (3)

\( \Leftrightarrow (\text{Elimination of existential quantifiers for } \text{banned}'' \text{ and } \text{members}'') \)

\[
\begin{align*}
\text{AddandDelete} & \\
\Delta Fid & \\
\text{applicant}? : \text{PERSON} & \\
\text{id}? : \text{ID}; \text{id}! : \text{ID} & \\
\text{members} \cup \{ \text{id}! \mapsto \text{applicant}? \} \in \text{ID} & \Leftrightarrow \text{PERSON} \land \\
\text{banned} \in \mathcal{P} \text{ ID} & \land \\
\text{banned} \subseteq \text{dom members} \cup \{ \text{id}! \mapsto \text{applicant}? \} & \land \\
\text{applicant}? \notin \text{ran members} & \land \\
\text{id}! \notin \text{dom members} & \land \\
\text{id}? \in \text{dom members} \cup \{ \text{id}! \mapsto \text{applicant}? \} & \land \\
\text{members}' = \{ \text{id}? \} \triangleq (\text{members} \cup \{ \text{id}! \mapsto \text{applicant}? \}) & \land \\
\text{banned}' = \text{banned} \setminus \{ \text{id}? \} & \land \\
\text{id}! = \text{id}? & \\
\end{align*}
\]
Proving a property of the specification (4)

Calculation of \( members' \):

\[
\begin{align*}
\{ id? \} \triangleleft (members \cup \{ id! \rightarrow applicant? \}) &= (id! = id?) \\
\{ id! \} \triangleleft (members \cup \{ id! \rightarrow applicant? \}) &= (R \triangleleft (S \cup T) = (R \triangleleft S) \cup (R \triangleleft T)) \\
(\{ id! \} \triangleleft members) \cup \{ id! \} \triangleleft \{ id! \rightarrow applicant? \} &= \text{(Definition von \( \triangleleft \))} \\
(\{ id! \} \triangleleft members) \cup \emptyset &= \\
\{ id! \} \triangleleft members &= \text{(id! \( \notin \) dom members)} \\
members &=
\end{align*}
\]
Proving a property of the specification (5)

Calculation of \( \text{banned}' \):

\[
\begin{align*}
\text{banned}' &= \\
\text{banned} \setminus \{ \text{id}? \} &= (\text{id}! = \text{id}?)
\end{align*}
\]

\[
\begin{align*}
\text{banned} \setminus \{ \text{id}! \} &= (\text{id}! \not\in \text{dom members} \land \\
&\phantom{=}(\text{banned} \subseteq \text{dom members})
\end{align*}
\]

\[
\text{banned}
\]

\[
\begin{align*}
\text{members}' &= \text{members} \land \text{banned}' = \text{banned} \Rightarrow
\end{align*}
\]

\[
\begin{align*}
\text{Fid}' &= \text{Fid} \Rightarrow
\end{align*}
\]

\[
\exists \text{Fid}
\]
Refinement of Specifications
Goal and approach

- Starting point: abstract specification with an abstract state and abstract operations
- Goal: transformation into a concrete specification which is nearer to the final implementation
- Refinement may be performed multiple times (i.e., multiple levels)
- Definition of a refinement:
  - Relation between abstract and concrete states, where each concrete state is mapped onto at most one abstract state
  - Each concrete initial state must be mapped onto a correct abstract initial state
  - Each concrete operation is mapped onto a corresponding abstract operation
  - The behavior of the concrete operation must be consistent with the behavior of the abstract operation
Abstract state

$AS_0 \xleftarrow{\text{Retr}} \xrightarrow{\text{State refinement}} AS_1' \xrightarrow{\text{Retr}} AS_3$

Concrete state

$CS_0 \xleftarrow{\text{Retr}} \xrightarrow{\text{Concrete operation}} CS_1 \xrightarrow{\text{Retr}} CS_3$

Input parameter

$AOp_{i1} \xrightarrow{?} CS_{0} \xrightarrow{!} CS_{1}$

Output parameter

$AOp_{i2} \xrightarrow{?} CS_{1} \xrightarrow{!} CS_{3}$

Concrete operation

$AOp_{i1} \xrightarrow{?} CS_{0} \xrightarrow{!} CS_{1}$

Alternative successor state

$AS_{0} \xleftarrow{\text{Retr}} \xrightarrow{\text{Issue}} AS_{1}^\prime \xrightarrow{\text{Retr}} AS_{3}$
Formal definition of a refinement

- **AS, CS**: Schemata for abstract and concrete states
  - **InitAS, InitCS**: Schemata for initial states
- **Retr(ieve)**: Schema for the correlation of abstract and concrete states

\[
\text{Retr} \quad \begin{array}{c} \text{AS} \\ \text{CS} \end{array} \quad \text{RelASCS}
\]

- Each schema **AO** for an abstract operation is mapped onto a schema **CO** for the corresponding concrete operation
Theorems to be proved

- **Initialization theorem**
  Each concrete initial state represents an abstract initial state:
  \[ \text{InitCS} \land \text{Retr'} \vdash \text{InitAS} \]

- **Applicability theorems**
  If an abstract operation is applicable in an abstract state, the corresponding concrete operation is applicable in the corresponding concrete state:
  \[ \text{pre AOp} \land \text{Retr} \vdash \text{pre COp} \]

- **Correctness theorems**
  If an abstract operation is applicable and the corresponding concrete operation is applied, the behavior is latter is consistent with the behavior of the former:
  \[ \text{pre AOp} \land \text{Retr} \land \text{COp} \land \text{Retr'} \vdash \text{AOp} \]
Abstract state of the soccer fan administration

\[ Fid\text{Scheme} \]
\[ \text{members} : ID \leftrightarrow \text{PERSON} \]
\[ \text{banned} : P ID \]

\[ \text{banned} \subseteq \text{dom members} \]
\[ \#\text{members} \leq \text{maxmems} \]

(* Abstract state, now with maximal number of members maxmems *)

\[ \text{InitFid\text{Scheme}} \]
\[ Fid' \]
\[ \text{members}' = \emptyset \]
\[ \text{banned}' = \emptyset \]

(* Initial state (without members) *)
Concrete state: array-based realization

<table>
<thead>
<tr>
<th>banarr</th>
<th>membarr</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>index</td>
</tr>
<tr>
<td></td>
<td>ID</td>
</tr>
<tr>
<td></td>
<td>PERSON</td>
</tr>
<tr>
<td>1</td>
<td>WW8901</td>
</tr>
<tr>
<td></td>
<td>Tom Cobbley</td>
</tr>
<tr>
<td>2</td>
<td>WW8903</td>
</tr>
<tr>
<td></td>
<td>Bill Vandal</td>
</tr>
<tr>
<td>3</td>
<td>WW9001</td>
</tr>
<tr>
<td></td>
<td>Daisy Widden</td>
</tr>
<tr>
<td>4</td>
<td>WW9002</td>
</tr>
<tr>
<td></td>
<td>Joe Hooly</td>
</tr>
<tr>
<td>5</td>
<td>WW9004</td>
</tr>
<tr>
<td></td>
<td>Sandra Skintight</td>
</tr>
<tr>
<td>6</td>
<td>WW9007</td>
</tr>
<tr>
<td></td>
<td>Joan Brewer</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Index array `banarr` and member array `membarr`.
**Z specification for the concrete state**

iseq[X] == seq X ∩ (N → X)

(* Representation of arrays by injective sequences *)

\[\begin{align*}
& CFidScheme \\
& \text{membarr} : \text{iseq}[ID \times \text{PERSON}] \\
& \text{banarr} : \text{iseq}[\mathbb{N}]
\end{align*} \]

ran membarr ∈ ID ↦ PERSON
ran banarr ⊆ 1..#membarr
#membarr ≤ maxmems

(* Concrete state, with maximal number of members maxmems *)

\[\begin{align*}
& \text{InitCFidScheme} \\
& CFidScheme' \\
& \text{membarr'} = \langle \rangle \\
& \text{banarr'} = \langle \rangle 
\end{align*} \]

(* Initial state (without members *)
Relation between abstract and concrete states

\[ \text{Retr} \]
\[ \text{FidScheme} \]
\[ \text{CFidScheme} \]

\[ \text{members} = \text{ran membarr} \]
\[ \text{banned} = \text{dom ( ran ( ran banarr < membarr ) )} \]

(* Members are pairs occurring as elements of \text{membarr}. The identifiers of banned persons are obtained as the first components of pairs which are marked by indices in \text{banarr}. *)
Initialization theorem

To demonstrate:

\[ InitCFidScheme \land Retr' \vdash InitFidScheme \]

\[ members' = \text{ran} \ \text{membarr'} = \text{ran} \langle \rangle = \emptyset \]

\[ banned' = \text{dom} ( \text{ran} ( \text{ran} \ \text{banarr'} \triangleleft \text{membarr'} ) ) \]
\[ = \text{dom} ( \text{ran} ( \text{ran} \langle \rangle \triangleleft \langle \rangle ) ) \]
\[ = \emptyset \]
Abstract and concrete operation

AddMember

ΔFidScheme
applicant? : PERSON
id! : ID

applicant? ∉ ran members
id! ∉ dom members
members' = members ∪ { id! → applicant? }
banned' = banned

CAddMember

ΔCFidScheme
applicant? : PERSON
id! : ID

applicant? ∉ ran (ran membarr)
id! ∉ dom (ran membarr)
membarr' = membarr ∩ (id!, applicant?)
banarr' = banarr
Preconditions

**PreAddMember**

<table>
<thead>
<tr>
<th>FidScheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicant? : PERSON</td>
</tr>
</tbody>
</table>

applicant? \notin ran members  
don members \neq ID  
#members < maxmems

**PreCAddMember**

<table>
<thead>
<tr>
<th>CFidScheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicant? : PERSON</td>
</tr>
</tbody>
</table>

applicant? \notin ran (ran membarr)  
don (ran membarr) \neq ID  
#membarr < maxmems
Applicability theorem

To demonstrate:

\[ \text{PreAddMember} \land \text{Retr} \vdash \text{PreCAddMember} \iff \]

\[ \text{FidScheme; applicant? : PERSON; CFidScheme |} \]

\[ \begin{align*}
\text{applicant?} & \notin \text{ran members} & \text{(H1)} \\
\text{dom members} & \neq \text{ID} & \text{(H2)} \\
\#\text{members} & < \text{maxmems} & \text{(H3)} \\
\text{members} & = \text{ran membarr} & \text{(H4)} \\
banned & = \text{dom ( ran ( ran banarr \triangleleft membarr ) )} & \text{(H5)}
\end{align*} \]

\[ \vdash \]

\[ \begin{align*}
\text{applicant?} & \notin \text{ran (ran membarr)} & \text{(G1)} \\
\text{dom (ran membarr)} & \neq \text{ID} & \text{(G2)} \\
\#\text{membarr} & < \text{maxmems} & \text{(G3)}
\end{align*} \]
Proof of the applicability theorem

Proof of (G1):

\[ applicant? \not\in \text{ran members} \quad \text{(H1)} \]
\[ \Rightarrow applicant? \not\in \text{ran (ran membarr)} \quad \text{(H4)} \]

Proof of (G2):

\[ \text{dom (ran membarr)} = \text{dom members} \quad \text{(H4)} \]
\[ \neq ID \quad \text{(H2)} \]

Proof of (G3):

\[ \#membarr = \#(\text{ran membarr}) \quad \text{(membarr is injective)} \]
\[ = \#\text{members} \quad \text{(H4)} \]
\[ < \text{maxmems} \quad \text{(H3)} \]
Correctness theorem

To demonstrate:

\[ \text{PreAddMember} \land \text{Retr} \land \text{CAddMember} \land \text{Retr'} \vdash \text{AddMember} \iff \]

\[ \text{PreAddMember} \land \text{Retr} \land \text{CAddMember} \land \text{Retr'} \]

\[ \vdash \]

\[ \begin{align*}
\text{applicant?} & \notin \text{ran members} \\
\text{id!} & \notin \text{dom members} \\
\text{members'} & = \text{members} \cup \{ \text{id!} \mapsto \text{applicant?} \} \\
\text{banned'} & = \text{banned}
\end{align*} \]

(G1)  
(G2)  
(G3)  
(G4)
Proof of the correctness theorem

Proof of (G1):
applicant? \notin \text{ran members} \quad (\text{PreAddMember})

Proof of (G2):
id! \notin \text{dom (ran membarr)} \quad (\text{CAddMember})
\iff id! \notin \text{dom members} \quad (\text{Retr})

Proof of (G3):
\text{members'}
= \text{ran membarr'} \quad (\text{Retr'})
= \text{ran (membarr} \setminus \langle \langle id!, \text{applicant?} \rangle \rangle \rangle) \quad (\text{CAddMember})
= \text{ran membarr} \cup \{ \langle id!, \text{applicant?} \rangle \} \quad (\text{Properties of ran and } \setminus)
= \text{members} \cup \{ id! \mapsto \text{applicant?} \} \quad (\text{Retr})

Proof of (G4):
\text{banned'}
= \text{dom(ran (ran banarr' \triangleleft \text{membarr'})}) \quad (\text{Retr'})
= \text{dom(ran (ran banarr \triangleleft (membarr \setminus \langle \langle id!, \text{applicant?} \rangle \rangle \rangle))}) \quad (\text{CAddMember})
= \text{dom(ran (ran banarr \triangleleft \text{membarr})}) \quad (\text{CFidScheme})
= \text{banned} \quad (\text{Retr})
Advantages of Z

- Based on theoretical foundations (logic and set theory) which should be known at least to mathematically trained users
- Very general approach
- Compact specifications
- Model-oriented specification is easier to understand/construct than behavioral specification
- Proofs with the help of predicate logic and set theory
- Step-wise refinement of specifications is supported
Disadvantages of Z

- Complex notation with many, many operators
- Abstract data types are modeled only implicitly, relying on certain conventions
- Modeling of operations with Δ schemata is hard to understand at first glance
- Notations and methods for structuring large specifications are missing (schemata are too fine-grained for this purpose)
- Transition from the specification to the implementation is difficult
- Often, Z is used only as a documentation aid
Literature

  *Introductory textbook, on which this chapter is based.*

  *Another textbook.*

  *Reference Manual including the language definition. Not appropriate as a textbook.*

  *Textbook with a brief introduction into Z, followed by many case studies.*