Specification of state-based Systems

Prof. Martin Wirsing

25.11.2002

MiS
MIS

Model-oriented Specifications with Z

Model-oriented Specification

- based on concrete mathematical structures (such as sequences, sets,....)
- operations are described by properties of pre- and post states

Z

1978 designed by B. Suffrin, J.R. Abrial (Oxford)

VDM

1970 "Vienna Development Method (VDM)" designed by Cliff Jones, D. Bjorner (IBM Vienna)

RAISE

1985 D. Bjorner combines VDM with alg. spec.

Object-Z, Z++

1990 Z with object-oriented constructs

The Specification Language Z

- set-oriented specification language
- based on Zermelo-Fraenkel set theory
- specification consists of schemata

A specification consists of

- basic data types
- schemata

A schema describes

- static aspects
  - possible states of a system
  - state invariants
- dynamic aspects
  - operations
  - relations between input and output
  - change of state

Basic Computation Structures of Z

Z is based on first order predicate logic and typed set theory.

Logic

$\neg P$	not P
$P \wedge Q$	P and Q
$P \vee Q$	P or Q
$P \Rightarrow Q$	P implies Q
$P \Leftrightarrow Q$	P holds if, and only if Q holds
$\forall x : T \mid P \bullet Q$	for all x of type T, which satisfy P, Q holds
$\forall x : T \bullet Q$	for all x of type T, Q holds (special case)
$\exists x : T \mid P \bullet Q$	there exists one x of type T, which satisfies P and Q
$\exists_1 x : T \mid P \bullet Q$	there exists exactly one x of type T, which satisfies P and Q

Set Theory

$x \in S$	x is element of S
$S \subseteq T$	S is subset of T, i.e. $\forall x : S \bullet x \in T$
$\emptyset$	empty set
$\{x_1, \ldots, x_n\}$	the set consisting of $x_1, \ldots, x_n$
$\{x : T \mid P\}$	the set of all x of type T, which satisfy P
$\{x : T \mid P \bullet t\}$	the set of all values of $t(x)$ s.t. x satisfies P, i.e. $\{t(x) \mid x \in T \wedge P(x)\}$
$\mu\, x : T \mid P$	the only x of type T, which satisfies P
$\mu\, x : T \mid P \bullet t$	the value of t for the only x of type T, which satisfies P
$(x_1, \ldots, x_n)$	ordered n-tuple
$S_1 \times \ldots \times S_n$	cartesian product, i.e. $\{x_1 : S_1;\ \ldots;\ x_n : S_n \bullet (x_1, \ldots, x_n)\}$

$\mathbb{P} S$	the set of all subsets of S
$\mathbb{F}\, S$	the set of all *finite* subsets of S
$S \cap T$	the intersection of S and T, i.e. $\{x : S \mid x \in T\}$
$S \cup T$	the union of S and T, i.e. $\{x : X \mid x \in S \vee x \in T\}$ (X type of elements of S and T)
$S \setminus T$	the set difference, i.e. $\{x : S \mid x \notin T\}$
$\bigcup SS$	generalised union, i.e. $\{x : X \mid (\exists\, S : SS \bullet x \in S)\}$
$\#S$	number of elements of finite set S
$\mathbb{N}$, $\mathbb{Z}$	natural numbers, the integers
$m \ldots n$	interval from m to n, i.e. $\{k : \mathbb{N} \mid m \leq k \wedge k \leq n\}$

Relations

$X \leftrightarrow Y$	binary relations between X and Y, i.e. $\mathbb{P}(X \times Y)$
$x \underline{R} y$	x and y are in relation R, i.e. $(x, y) \in R$
$x \mapsto y$	maplet of x and y, also written (x, y)
$\mathrm{dom}\, R$	domain of R, i.e. $\{x : X \mid (\exists\, y : Y \bullet xRy)\}$
$\mathrm{ran}\, R$	codomain of R, i.e. $\{y : Y \mid (\exists\, x : X \bullet xRy)\}$
$R_1 \mathbin{;} R_2$	relation composition, $\{x : X;\ z : Z \mid (\exists\, y : Y \bullet x\ \underline{R_1}\ y \land y\ \underline{R_2}\ z) \bullet (x \mapsto z)\}$
R^{-1} (also $R^{\sim}$)	inverse of R, i.e. $\{y : Y;\ x : X \mid x\ \underline{R}\ y \bullet (y \mapsto x)\}$
$\mathrm{id}\, S$	identity relation of S, i.e. $\{x : S \bullet x \mapsto x\}$
$R(\!\mid S \mid\!)$	relation image, i.e. $\{y : Y \mid (\exists\, x : S \bullet x\ \underline{R}\ y)\}$
$S \lhd R$	restriction of the domain, $\{x : X;\ y : Y \mid x \in S \land x\ \underline{R}\ y \bullet (x \mapsto y)\}$
$S \ntriangleleft R$	anti restriction of the domain, $\{x : X;\ y : Y \mid x \notin S \land x\ \underline{R}\ y \bullet (x \mapsto y)\}$
$R \rhd T$	restriction of the codomain, $\{x : X;\ y : Y \mid x\ \underline{R}\ y \land y \in T \bullet (x \mapsto y)\}$
$R \ntriangleright T$	anti restriction of the codomain, $\{x : X;\ y : Y \mid x\ \underline{R}\ y \land y \notin T \bullet (x \mapsto y)\}$
$R_1 \oplus R_2$	overwriting of R_1, where R_2 is defined, $(\mathrm{dom}\, R_2 \ntriangleleft R_1) \cup R_2$

Functions

$X \pfun Y$	partial functions from X to Y, $\{ f : X \leftrightarrow Y \mid \forall x : X;\ y_1, y_2 : Y \bullet x \underline{f}\ y_1 \wedge x \underline{f}\ y_2 \Rightarrow y_1 = y_2 \}$
$X \rightarrow Y$	total functions from X to Y, $\{ f : X \pfun Y \mid \mathrm{dom}\, f = X \}$
$X \ffun Y$	finite partial functions from X to Y, $\{ f : X \pfun Y \mid \mathrm{dom}\, f \in \mathbb{F}\, X \}$
$X \pinj Y$	partial injective functions from X to Y, $\{ f : X \pfun Y \mid f^{-1} \in Y \pfun X \}$
$X \rightarrowtail Y$	total injective functions from X to Y, $(X \rightarrow Y) \cap (X \pinj Y)$
$X \psurj Y$	partial surjective functions from X to Y, $\{ f : X \pfun Y \mid \mathrm{ran}\, f = Y \}$
$X \twoheadrightarrow Y$	total surjective functions from X to Y, $(X \rightarrow Y) \cap (X \psurj Y)$
$X \bij Y$	bijections from X to Y, $(X \rightarrowtail Y) \cap (X \twoheadrightarrow Y)$
$f\ x$, $f(x)$	application of the function f to the argument x, $\mu\, y : Y \mid x\ \underline{f}\ y$
$\lambda\, x : T \mid P \bullet t$	lambda-notation, $\{ x : T \mid P \bullet x \mapsto t \}$

Sequences

$\mathrm{seq}\, X$	sequences over X, $\{ s : \mathbb{N} \pfun X \mid \mathrm{dom}\, s = 1 \mathinner{..} \#s \}$
$\#s$	length of s (see $\#$ sets)
$\langle\rangle$	empty sequence ϵ
$\langle x_1, \ldots, x_n \rangle$	enumeration of a finite sequence, $\{(1 \mapsto x_1), \ldots, (n \mapsto x_n)\}$
$s \frown t$	concatenation of s and t, $s \cup \{i : 1 \mathinner{..} \#t \bullet (i + \#s \mapsto t(i))\}$

Basic Schemata

The (name for) data type D

$$[D]$$

A schema S has the form

$$\begin{array}{|l}
\hline S \\
x_1 : T_1;\ \ldots;\ x_n : T_n \\
\hline
P \\
\hline
\end{array}$$

whereby

- $x_1 : T_1;\ \ldots;\ x_n : T_n$ is a set of declarations and
- P is a predicate, that can include a set G of global variables beside $x_1, \ldots, x_n$.

The semantics of S is given by a state signature and a class of models.

$$\mathsf{StateSig}(S) \quad =_{def} \quad \{x_1 : T_1, \ldots, x_n : T_n\},$$

Let G be the signature of the basic structures of Z:

$$\begin{aligned}
\mathsf{Sig}(S) \quad &=_{def} \quad G \cup \mathsf{StateSig}(S) \\
\mathsf{SStruct}(\mathsf{StateSig}(S)) \quad &=_{def} \quad \{A \in \mathsf{Struct}(\mathsf{Sig}(S)) \;\mid \\
&\qquad A \text{ is standard over the interpretation} \\
&\qquad \text{of the given datatypes } A_D\} \\
\mathsf{Mod}(S) \quad &=_{def} \quad \{A \in \mathsf{Struct}(\mathsf{Sig}(S)) \;\mid\; A \models P\}.
\end{aligned}$$

Every structure A denotes a possible state of the variables of the schema.

Example (Basic Schemata):

1. The semantic of the schema

$$\begin{array}{|l} \hline S_0 \\ x : \mathbb{Z} \\ y : \mathsf{seq}\mathbb{Z} \\ \hline x < \#y \\ \hline \end{array}$$

$$\mathsf{StateSig}(S_0) = \{x : \mathbb{Z}, y : \mathsf{seq}\mathbb{Z}\} \quad \text{with types } \mathbb{Z}, \mathsf{seq}\mathbb{Z}$$
$$\mathsf{Mod}(S_0) = \{A \in \mathsf{SStruct}(\mathsf{StateSig}(S_0)) \mid A \models x < \#y\}$$

2. The schema

T

$z : 1 \mathinner{..} 10$
$x : \mathbb{N}$

$x = z * z$

is abreviation for

T

$z : \mathbb{Z}$
$x : \mathbb{Z}$

$z \in 1 \mathinner{..} 10$
$x \in \mathbb{N}$
$x = z * z$

The signature only considers the types of variables, not the state information.

$$\mathsf{StateSig}(T) = G \cup \{x : \mathbb{Z}, z : \mathbb{Z}\}$$
$$\mathsf{Mod}(T) = \{A \in \mathsf{SStruct}(\mathsf{StateSig}(T)) \mid A \models z \in 1..10 \land x \in \mathbb{N} \land x = z^2\}$$

3. A birthday book:

$$[NAME, DATE]$$

are the basic data types $NAME$ and $DATE$.

The state space is described by following schema:

$$
\begin{array}{|l}
\hline
\textit{BirthdayBook} \\
known : \mathbb{P} NAME \\
birthday : NAME \rightarrow\!\!\!\!\!+ DATE \\
\hline
known = \operatorname{dom} birthday \\
\hline
\end{array}
$$

$\mathbb{P}$ = power set,
dom = domain,
$birthday$ is a partial function,
$known = \operatorname{dom} birthday$ is an invariant,
$BirthdayBook$ has two new variables.

A possible state is

$$\langle\!\langle\, known = \{\text{"Martin"}, \text{"Thomas"}, \text{"Sabine"}\},$$
$$birthday = \{\text{"Martin"} \mapsto \text{"24. 12."}, \text{"Thomas"} \mapsto \text{"8. 02."}, \text{"Sabine"} \mapsto \text{"8. 02."}\} \,\rangle\!\rangle$$

$$\mathsf{StateSig}(BirthdayBook) = \{known : \mathbb{P} NAME, birthday : NAME \nrightarrow DATE\}$$
$$\mathsf{Mod}(BirthdayBook) = \{A \in \mathsf{SStruct}(\mathsf{StateSig}(BirthdayBook)) \mid A \models known = \mathsf{dom}\, birthday\}$$

A schema S can be considered as a record with the selectors $x_1 : T_1, \ldots, x_n : T_n$. The renaming of a schema results in a new schema.
e.g.

$$S_1 = [\, a : \mathbb{N};\; b : \mathsf{seq}\mathbb{N} \mid a < \#b \,]$$

is different from S_0 .

The name is preserved after a combination of schemata. Adding schemata extends the neighbourhood by new schema names.

Schema Combination

Operators for combining schemata:

- schema inclusion
- logical composition
- export and hiding
- decoration

Schema Inclusion

A schema R can be a extension of a schema S:

$$\begin{array}{|l} \hline R \\ S \\ y_1 : R_1;\ \ldots;\ y_m : R_m \\ \hline P_1 \\ \hline \end{array}$$

defines the schema

$$\begin{array}{|l} \hline R \\ x_1 : T_1;\ \ldots;\ x_n : T_n \\ y_1 : R_1;\ \ldots;\ y_m : R_m \\ \hline P \wedge P_1 \\ \hline \end{array}$$

Example:

$$\begin{array}{|l} \hline R \\ S_0 \\ z : \mathbb{N} \\ \hline z < x \\ \hline \end{array}$$

stands for

$$\begin{array}{|l} \hline R \\ x, z : \mathbb{N} \\ y : \mathsf{seq}\mathbb{N} \\ \hline x < \#y \wedge z < x \\ \hline \end{array}$$

Logical Combination of schemata

1. Conjunction

$$S \wedge T =_{def} S \textbf{ and } T$$

i.e.

$$\begin{aligned} \mathsf{Sig}(S \wedge T) &= \mathsf{Sig}(S) \cup \mathsf{Sig}(T) \\ \mathsf{Mod}(S \wedge T) &= \{A \in \mathsf{Struct}(\mathsf{Sig}(S \wedge T)) \mid \\ & \quad A\,|_{\mathsf{Sig}(S)} \in \mathsf{Mod}(S) \text{ and } A\,|_{\mathsf{Sig}(T)} \in \mathsf{Mod}(T)\} \end{aligned}$$

$S \wedge T$ is the intersection of the models.

Example:

$$S_0 \wedge T \quad = \quad \begin{array}{|l} x : \mathbb{N} \\ y : \mathsf{seq}\,\mathbb{Z} \\ z : 1 \,..\, 10 \\ \hline x < \#y \wedge x = z * z \\ \hline \end{array}$$

2. Disjunction
   The disjunction $S \vee T$ denotes the union of the models

$$
\begin{aligned}
\mathsf{Sig}(S \vee T) &= \mathsf{Sig}(S) \cup \mathsf{Sig}(T) \\
\mathsf{Mod}(S \vee T) &= \{A \in \mathsf{Struct}(\mathsf{Sig}(S \vee T)) \mid \\
&\quad A\,|_{\mathsf{Sig}(S)} \in \mathsf{Mod}(S) \text{ or } A\,|_{\mathsf{Sig}(T)} \in \mathsf{Mod}(T)\}
\end{aligned}
$$

Example:

$$
S_0 \vee T \quad = \quad
\begin{array}{|l}
\hline
x : \mathbb{Z} \\
y : \mathsf{seq}\mathbb{Z} \\
z : 1 \mathinner{..} 10 \\
\hline
x < \#y \vee (x \in \mathbb{N} \wedge x = z * z) \\
\hline
\end{array}
$$

3. Negation The schema $\neg\, S$ represents the complement of the models under preservation of the types:

$$\begin{aligned} \mathsf{Sig}(\neg\, S) &= \mathsf{Sig}(S) \\ \mathsf{Mod}(\neg\, S) &= \{\, A \in \mathsf{Struct}(\mathsf{Sig}(S)) \mid A \notin \mathsf{Mod}(S)\,\} \end{aligned}$$

Example:

$$\neg\, T \quad = \quad \begin{array}{|l} x : \mathbb{Z} \\ z : \mathbb{Z} \\ \hline x \notin \mathbb{N} \vee z \notin \mathbb{N} \vee z \notin 1\,..\,10 \vee x \neq z * z \\ \hline \end{array}$$

4. Quantification hides (free) variables.

$$Qx_1 : T_1;\ \ldots;\ x_k : T_k \mid P \bullet S$$

(where $k < n$, i.e. the variables $x_1, \ldots, x_k$ are in S and have the same type as in S)

$x_{k+1} : T_{k+1};\ \ldots;\ x_n : T_n$

$Qx_1 : T_1;\ \ldots;\ x_k : T_k \mid P \bullet S$

Example:

$\exists\, z : \mathbb{N} \mid z > 5 \bullet T \quad =$

$x : \mathbb{N}$

$\exists\, z : \mathbb{N} \mid z > 5 \bullet z \in 1 \mathinner{..} 10 \land x = z * z$

reduces to

$x : \mathbb{N}$

$\exists\, z : 6 \mathinner{..} 10 \bullet x = z * z$

If we quantify over all declared variables of a schema S, we write:

$$QS \bullet T$$

as abreviation for

$$Qx_1 : T_1;\ \ldots;\ x_n : T_n \mid P \bullet T$$

where

$$\begin{array}{|l} \hline S \\ \hline x_1 : T_1;\ \ldots;\ x_n : T_n \\ \hline P \\ \hline \end{array}$$

5. Export and hiding of symbols
   (a) $S \upharpoonright (x_1, \ldots, x_k)$
   (b) $S \setminus (x_1, \ldots, x_k)$

   are schemata, that restrict the signature of S by restriction (a) and hiding (b).
   These operations can be defined by quantification.

$$S \upharpoonright (x_1, \ldots, x_k) \quad =_{def} \quad \exists\, x_{k+1} : T_{k+1};\ \ldots;\ x_n : T_n \bullet S$$

$$S \setminus (x_1, \ldots, x_k) \quad =_{def} \quad \exists\, x_1 : T_1;\ \ldots;\ x_k : T_k \bullet S$$

Example:

$$T \upharpoonright x \quad = \quad \exists\, z : \mathbb{N} \bullet T \quad = \quad \begin{array}{|l} x : \mathbb{N} \\ \hline \exists\, z : \mathbb{N} \mid z \in 1 \mathinner{..} 10 \bullet x = z * z \end{array}$$

Decoration

The identificators in schemata can be decorated:

$$
\begin{array}{|l}
\hline
S' \\
x_1' : T_1;\ \ldots;\ x_n' : T_n \\
\hline
P[x_1'/x_1, \ldots, x_n'/x_n] \\
\hline
\end{array}
$$

Example:

$$
\begin{array}{|l}
\hline
S_0' \\
x' : \mathbb{Z} \\
y' : \mathsf{seq}\mathbb{Z} \\
\hline
x' < \# y' \\
\hline
\end{array}
$$

Semantics of S':

$$\begin{array}{rcl} \mathsf{StateSig}(S') & = & \{x' : T \mid x : T \in \mathsf{StateSig}(S)\} \\ \mathsf{Mod}(S') & = & \{A \in \mathsf{SStruct}(\mathsf{StateSig}(S')) \mid \exists\, B \in \mathsf{Mod}(S) : A\mid_{copy} = B\}, \end{array}$$

where $copy(x) = x'$ for all $x : T \in \mathsf{Sig}(S)$ is the signature morphism, that decorates all declared symbols of S with $'$.

An equivalent form is:

$$S' = S \textbf{ with } copy$$

State Transitions

$\Delta S =_{def} S \wedge S'$

Generally the schema S denotes a state space of a abstract data type.

Every model of ΔS has the signature $\mathsf{Sig}(S) \cup \mathsf{Sig}(S')$, i.e. let $\mathsf{StateSig}(S) = \{x_1 : T_1, \ldots, x_n : T_n\}$. So

$$\begin{aligned} \mathsf{StateSig}(\Delta S) &= \{x_1 : T_1, \ldots, x_n : T_n, x_1' : T_1, \ldots, x_n' : T_n\} \\ \mathsf{Mod}(\Delta S) &= \{A \in \mathsf{SStruct}(\mathsf{StateSig}(\Delta S)) \mid A|_{\mathsf{Sig}(S)} \in \mathsf{Mod}(S) \text{ and} \\ &\qquad A|_{\mathsf{Sig}(S')} \in \mathsf{Mod}(S')\} \end{aligned}$$

ΔS can be considered as a state transition, i.e. every element of $\mathsf{Mod}(\Delta S)$ consists of a pair $\langle B, B' \rangle$ of algebras, we write:

$$B \rightarrow_S B'$$

where $B = A|_{\mathsf{Sig}(S)}$ and $B' = A|_{\mathsf{Sig}(S')}$ holds for a $A \in \mathsf{Mod}(\Delta S)$.

ΔS defines the relation between B and B' through the axioms of S, i.e. B and B' can be any models (modulo renaming).

Preservation of Values

$\Xi S = \Delta S \wedge \bigwedge_{i=1,\ldots,n} x_i = x'_i$

ΞS is an abreviation for not changing the values in the post state, that have been declared by the variables of S.

In imperative programing languages, we express local changes of a single variable by $x := e$.

Let StateSig$(S) = \{x_1 : T_1, \ldots, x_n : T_n\}$. Then for the post-constraint of $x_i := e$ holds :

$$x'_1 = x_1 \wedge \ldots \wedge x'_i = e \wedge \ldots \wedge x'_n = x_n$$

Example (Counter):

Counter

$$
\begin{array}{|l}
\hline
\textit{Counter} \\
\textit{value}, \textit{limit} : \mathbb{N} \\
\hline
\textit{value} < \textit{limit} \\
\hline
\end{array}
$$

Define the operation Inc (increment):

$$
\begin{array}{|l}
\hline
\textit{Inc} \\
\Delta \textit{Counter} \\
\hline
\textit{value}' = \textit{value} + 1 \\
\textit{limit}' = \textit{limit} \\
\hline
\end{array}
$$

Initial state:

$$
\begin{array}{|l}
\hline
\textit{InitCounter} \\
\textit{Counter} \\
\hline
\textit{value} = 0 \wedge \textit{limit} = 100 \\
\hline
\end{array}
$$

Satisfiability conclusion for $InitCounter$:

$$\exists\, Counter \bullet InitCounter \quad \equiv$$
$$\exists\, value, limit : \mathbb{N} \bullet value < limit \wedge value = 0 \wedge limit = 100$$

Addition (with Input/ Output):

$$\begin{array}{|l}
\textbf{Add} \\
\hline
\Delta Counter \\
jump? : \mathbb{N}, new_value! : \mathbb{N} \\
\hline
value' = value + jump? \\
limit' = limit \\
new_value! = value' \\
\hline
\end{array}$$

Example (Operations of birthday book):

$$
\begin{array}{l}
\textit{AddBirthday} \\
\hline
\Delta \textit{BirthdayBook} \\
\textit{name?} : \textit{NAME} \\
\textit{date?} : \textit{DATE} \\
\hline
\textit{name?} \notin \textit{known} \\
\textit{birthday}' = \textit{birthday} \cup \{\textit{name?} \mapsto \textit{date?}\}
\end{array}
$$

The following property can be proven:

$$\textit{known}' = \textit{known} \cup \{\textit{name?}\}$$

Proof 1 (Property of $AddBirthday$):

$$
\begin{array}{lll}
 & known' & \\
= & \mathrm{dom}\, birthday' & [\text{ invariant of } Birthday' \text{ }] \\
= & \mathrm{dom}(birthday \cup \{name? \mapsto date?\}) & [\text{ specification } AddBirthday \text{ }] \\
= & \mathrm{dom}\, birthday \cup \mathrm{dom}\{name? \mapsto date?\} & [\text{ set theory }] \\
= & \mathrm{dom}\, birthday \cup \{name?\} & [\text{ property of } \mathrm{dom} \text{ }] \\
= & known \cup \{name?\} & [\text{ invariant of } Birthday \text{ }]
\end{array}
$$

We used the mathematical properties:

$$
\begin{array}{rcl}
\mathrm{dom}(f \cup g) & = & (\mathrm{dom} f) \cup (\mathrm{dom} g) \\
\mathrm{dom}\{a \mapsto b\} & = & \{a\}
\end{array}
$$

Semantically $AddBirthday$ describes a state transition of StateSig$(BirthdayBook)$-algebras into post-algebras with input variables $name?$ and $date?$:

$$A \in \mathsf{Mod}(BirthdayBook) \rightarrow_{AddBirthday} A'$$

where in A' holds:

$$\begin{array}{rcl} known^{A'} & = & \mathsf{dom}\, birthday^{A'} \\ birthday^{A'} & = & birthday^{A} \cup \{name?^{A'} \mapsto date?^{A'}\} \\ date?^{A'} & & \text{any element of the carrier set } Date^{A'} = Date^{A} \\ name?^{A'} & & \text{any element } \notin known^{A} \text{ of the carrier set } Name^{A'} = Name^{A} \end{array}$$

The exclamation mark describes an output variable. The following operations do not change the state of $BirthdayBook$

$$\Xi BirthdayBook \quad \equiv \quad \Delta BirthdayBook \land known' = known \land birthday' = birthday$$

FindBirthday

$$\Xi BirthdayBook$$
$$name? : NAME$$
$$date! : DATE$$

$$name? \in known$$
$$date! = birthday(name?)$$

Remind

$$\Xi BirthdayBook$$
$$today? : DATE$$
$$cards! : \mathbb{P} NAME$$

$$cards! = \{\, n : NAME \mid n \in known \land birthday(n) = today? \,\}$$

$$\textit{InitBirthdayBook}$$

$$\textit{BirthdayBook}$$

$$known = \emptyset$$

Sequential Composition

Two schemata S_1 and S_2 can be combined sequentially by $S_1 \fatsemi S_2$.

$$A \longrightarrow_{S_1} A'', A'' \longrightarrow_{S_2} A'' \quad \Rightarrow \quad A \longrightarrow_{S_1 \fatsemi S_2} A'$$

Formally: Let S_1 and S_2 be defined over the same signature Σ , then:

$$S_1 \fatsemi S_2 =_{def} \exists\, S'' \bullet S_1[S''/S'] \wedge S_2[S''/S]$$

holds.

Example (Composition of counter operations):

1. $Inc \mathbin{\text{\textfrak{;}}} Inc \quad = \quad \exists\, value'', limit'' : \mathbb{N} \bullet$

$$\begin{array}{|l} \hline \Delta Counter \\ \hline value'' = value + 1 \\ limit'' = limit \\ value' = value'' + 1 \\ limit' = limit'' \\ \hline \end{array} \quad = \quad \begin{array}{|l} \hline Inc \fatsemi Inc \\ \Delta Counter \\ \hline value' = value + 2 \\ limit' = limit \\ \hline \end{array}$$

2. $Inc \fatsemi Add$

$$
\begin{array}{|l}
\multicolumn{1}{l}{Inc \fatsemi Add} \\
\hline
\Delta Counter \\
jump? : \mathbb{N}, new\_value! : \mathbb{N} \\
\hline
value' = value + jump? + 1 \\
limit' = limit \\
new\_value! = value' \\
\hline
\end{array}
$$

3. $Add \fatsemi Inc$

$$
\begin{array}{|l}
\multicolumn{1}{l}{Add \fatsemi Inc} \\
\hline
\Delta Counter \\
jump? : \mathbb{N}, new\_value! : \mathbb{N} \\
\hline
value' = value + jump? + 1 \\
limit' = limit \\
new\_value! = value + jump? \quad (= value' - 1) \\
\hline
\end{array}
$$

Summary

- Z is a model-oriented specification language for state-based systems.
- System states are described by Z-schemata

$$\begin{array}{|l} \hline S \\ x_1 : T_1;\ \ldots;\ x_n : T_n \\ \hline P \\ \hline \end{array}$$

equivalent to $S \mathrel{\widehat{=}} [x_1 : T_1;\ \ldots;\ x_n : T_n \mid P]$

Basic sorts:
$[\,SORT\,]$, e.g. $[\,ADDRESS\,]$.

- Combination of schemata
  - propositional logic $\wedge, \vee, \neg$
  - quantification $\exists\, \vec{x} : \vec{T} \bullet S$, $\forall\, \vec{x} : \vec{T} \bullet S$
  - export/hiding $S \upharpoonright (\vec{x})$, $S \setminus (\vec{x})$

- Decoration of names by $'$ (post-state), $?$ (input) or $!$ (output)
- Specification of state changes

$$\begin{aligned} \Delta S &\equiv S \wedge S' \\ \Xi S &\equiv \Delta S \wedge \theta S = \theta S' \end{aligned}$$

- Sequential composition $S_1 \fatsemi S_2$

$$S_1 \fatsemi S_2 \quad \equiv \quad \exists\, \Sigma'' \bullet S_1[\Sigma''/\Sigma'] \wedge S_2[\Sigma''/\Sigma]$$

where Σ is the signature of S_1 and S_2.