Formal Specification of Software

The Z Specification Language

Bernhard Beckert



UNIVERSITÄT KOBLENZ-LANDAU

B. Beckert: Formal Specification of Software - p.1

Based on

- Typed first-order predicate logic
- Sermelo-Fraenkel set theory
- Rich notation

Based on

- Typed first-order predicate logic
- Sermelo-Fraenkel set theory
- Rich notation

Invented/developed by

J.-R. Abrial, Oxford University Computing Laboratory

International standard

ISO/IEC JTC1/SC22

Tools

- **JAT_EX style**
- Type checker
- Z/Eves deduction system

But

No tools for simulation/execution/testing

Logical operators

- \neg negation
- $\wedge\,$ conjunction
- \vee disjunction
- \Rightarrow implication (note: not \rightarrow)
- \Leftrightarrow equivalence (note: not \leftrightarrow)

Equality

= equality

On all types (but not predicates)

Quantification

$$Q x_1: S_1; \ldots; x_n: S_n \mid p \bullet q$$

where Q is one of $\forall \exists \exists_1$

Meaning

$$\forall x_1 : S_1; \dots; x_n : S_n(p \Rightarrow q)$$
 resp.
$$\exists x_1 : S_1; \dots; x_n : S_n(p \land q)$$

Abbreviation

$$\forall x: T \bullet q$$
 for $\forall x: T \mid true \bullet q$

Enumeration

$$\{e_1,\ldots,e_n\}$$

The set of type-compatible elements e_1, \ldots, e_n

Example

 $\{3, 5, 8, 4\}$

Set comprehension

 $\{x: T \mid pred(x) \bullet expr(x)\}$

The set of all elements that result from evaluating expr(x) for all x of type T for which pred(x) holds

Example

 $\{x: \mathbb{Z} \mid prime(x) \bullet x * x\}$

The set of all squares of prime numbers

Abbreviation

$$\{x: T \mid pred(x)\}$$
 for $\{x: T \mid pred(x) \bullet x\}$

Example

$$\mathbb{N} = \{ x : \mathbb{Z} \mid x \ge 0 \}$$

The empty set

$$\emptyset = \{ x : T \mid \mathbf{false} \}$$

Note:

 $\varnothing = \varnothing[T]$ is typed

Set Operations

- \in element-of relation
- \subseteq subset relation

 S_1 and S_2 must have the same type

 $S_1 \subseteq S_2 \Leftrightarrow (\forall x : S_1 \mid x \in S_2)$

- \mathbb{P} power set operator
 - $S' \in \mathbb{P}S \Leftrightarrow S' \subseteq S$
- \times cartesian product

 $(x_1,\ldots,x_n)\in S_1\times\ldots\times S_n\Leftrightarrow (x_1\in S_1\wedge\ldots\wedge x_n\in S_n)$

Set Operations

\cup, \bigcup union

Involved sets must have the same type T

$$x \in S_1 \cup S_2 \Leftrightarrow (x \in S_1 \lor x \in S_2)$$
$$x \in \bigcup S \Leftrightarrow (\exists S' : T \bullet x \in S')$$

\cap, \bigcap intersection

set difference

Types

Pre-defined types

 \mathbb{Z} with constants: $0, 1, 2, 3, 4, \dots$ functions: +, -, *, /predicates: $<, \le, >, \ge$

Sets

Every set can be used as a type

Basic types (given sets)

Example

[Person]

Example

weekDay ::= mon | tue | wed | thu | fri | sat | sun

Example

 $Tree ::= leaf \langle\!\langle \mathbb{Z} \rangle\!\rangle \mid node \langle\!\langle Tree \times Tree \rangle\!\rangle$

Meaning

[*Tree*] **generated by** *leaf*, *node*

 $\forall x_1, y_1, x_2, y_2 : Tree \mid node(x_1, y_1) = node(x_2, y_2) \bullet (x_1 = x_2 \land y_1 = y_2) \\ \forall x_1, x_2 : \mathbb{Z} \mid leaf(x_1) = leaf(x_2) \bullet x_1 = x_2 \\ \forall x : \mathbb{Z}; y, z : Tree \bullet leaf(x) \neq node(y, z)$

Note: Generatedness is not expressible in first-order logic

Set type: $\mathbb{P}T$

The type of sets of elements of type ${\cal T}$

Cartesian product type: $T_1 \times \cdots \times T_n$

The type of tuples (t_1, \ldots, t_n) with $t_i \in T_i$

Possible type definitions

- $T = \mathbb{Z}$
- T = [Type]
- $T := \dots$ (free type)
- $T = \mathbb{P} T'$
- $T = T_1 \times \cdots \times T_n$

Possible type definitions

- $T = \mathbb{Z}$
- T = [Type]
- $T := \dots$ (free type)
- $T = \mathbb{P} T'$
- $T = T_1 \times \cdots \times T_n$

Note

All types are disjoint(not for sets that are used as types)All terms have a unique type

Variable declarations

Example

 $x : \mathbb{Z}$ sold : \mathbb{P} Seat

Variables can range over types and over sets

Abbreviations

- must not be recursive
- can be generic ٩

Examples

numberPairs == $\mathbb{Z} \times \mathbb{Z}$

pairWithNumber[
$$S$$
] == $\mathbb{Z} \times S$

Note

Type variables are "meta-variables" (cannot be quantified)

$$weekDay1 == \{mon, tue, wed, thu, fri, sat, sun\}$$

VS.

WeekDay2 ::= mon | tue | wed | thu | fri | sat | sun

$$weekDay1 == \{mon, tue, wed, thu, fri, sat, sun\}$$

VS.

$$WeekDay2 := mon \mid tue \mid wed \mid thu \mid fri \mid sat \mid sun$$

Not the same

Type definition implies elements to be different

Form of an axiomatic definition

Symbol Declarations Constraining Predicates

Example

$$\mathbb{N}_1: \mathbb{P}\mathbb{Z}$$
$$\forall z: \mathbb{Z} \bullet (z \in \mathbb{N}_1 \leftrightarrow z \ge 1)$$

Relations

Relation types/sets

$S \leftrightarrow T$ is the type/set of relations between types/sets S and T

 $S \leftrightarrow T = \mathbb{P}(S \times T)$

Notation

 $a \mapsto b$ for (a, b) if $(a, b) \in S \leftrightarrow T$

Domain dom R

$$\operatorname{dom} R = \{a: S, b: T \mid a \mapsto b \in R \bullet a\}$$

Range $\operatorname{ran} R$

 $\operatorname{ran} R = \{a: S; b: T \mid a \mapsto b \in R \bullet b\}$

Domain dom R

$$\operatorname{dom} R = \{a: S, b: T \mid a \mapsto b \in R \bullet a\}$$

Range $\operatorname{ran} R$

$$\operatorname{ran} R = \{a: S; b: T \mid a \mapsto b \in R \bullet b\}$$

Restrictions of relations

$$S' \lhd R = \{a : S; b : T \mid a \mapsto b \in R \land a \in S' \bullet a \mapsto b\}$$
$$R \triangleright T' = \{a : S; b : T \mid a \mapsto b \in R \land b \in T' \bullet a \mapsto b\}$$
$$S' \lhd R = \{a : S; b : T \mid a \mapsto b \in R \land a \notin S' \bullet a \mapsto b\}$$
$$R \triangleright T' = \{a : S; b : T \mid a \mapsto b \in R \land b \notin T' \bullet a \mapsto b\}$$

Operations on Relations

Inverse relation R^{-1}

$$R^{-1} = \{a: S; b: T \mid a \mapsto b \in R \bullet b \mapsto a\}$$

Operations on Relations

Inverse relation R^{-1}

$$R^{-1} = \{a: S; b: T \mid a \mapsto b \in R \bullet b \mapsto a\}$$

Composition $R \circ R' = \{a : S; b : T; c : U \mid a \mapsto b \in R \land b \mapsto c \in R' \bullet a \mapsto c\}$

Operations on Relations

Inverse relation R^{-1}

$$R^{-1} = \{a: S; b: T \mid a \mapsto b \in R \bullet b \mapsto a\}$$

Composition $R \underset{\circ}{\circ} R'$ $R : S \leftrightarrow T$ and $R' : T \leftrightarrow U$ $R \underset{\circ}{\circ} R' = \{a : S; b : T; c : U \mid a \mapsto b \in R \land b \mapsto c \in R' \bullet a \mapsto c\}$

Closures $R: S \leftrightarrow S$

iteration identity refl./trans. transitive symetric reflexive

$$R^{n} = R \circ R^{n-1}$$

$$R^{0} = \{a : S \mid \mathbf{true} \bullet a \mapsto a\}$$

$$R^{*} = \bigcup \{n : \mathbb{N} \mid \mathbf{true} \bullet R^{n}\}$$

$$R^{+} = \bigcup \{n : \mathbb{N} \mid n \ge 1 \bullet R^{n}\}$$

$$R^{s} = R \cup R^{-1}$$

$$R^{r} = R \cup R^{0}$$

Functions

Special relations

Functions are special relations

Notation

Instead of \leftrightarrow

- \rightarrow total function
- \rightarrow partial function

Functions

Partial functions

$$\begin{array}{l} f \in S \leftrightarrow T \quad \Leftrightarrow \\ f \in S \leftrightarrow T \quad \land \\ \forall a : S, b : T, b' : T \mid (a \mapsto b \in f \land a \mapsto b' \in f) \bullet b = b' \end{array}$$

Functions

Partial functions

$$\begin{array}{l} f \in S \nleftrightarrow T \quad \Leftrightarrow \\ f \in S \leftrightarrow T \quad \land \\ \forall a : S, b : T, b' : T \mid (a \mapsto b \in f \land a \mapsto b' \in f) \bullet b = b' \end{array}$$

Total functions

$$\begin{array}{ccc} f \in S \to T & \Leftrightarrow \\ f \in S \to T & \land \\ \forall a : S \bullet \exists b : T \bullet a \mapsto b \in f \end{array}$$

General form

$$\lambda a: S \mid p \bullet e$$

Example

$$double : \mathbb{Z} \to \mathbb{Z}$$
$$double = \lambda n : \mathbb{Z} \mid n \ge 0 \bullet n + n$$

Equivalent to

$$double : \mathbb{Z} \to \mathbb{Z}$$
$$double = \{n : \mathbb{N} \mid \mathbf{true} \bullet n \mapsto n + n\}$$

Notation

Relations and functions can be declared prefix and infix

Parameter positions are indicated with "_"

Example

$$even_{-}: \mathbb{P}\mathbb{Z}$$
$$\forall x : \mathbb{Z} \bullet (even \ x \Leftrightarrow (\exists y : \mathbb{Z} \bullet x = y + y))$$

Equivalent to

$$even_{-}: \mathbb{P}\mathbb{Z}$$
$$even_{-}: \mathbb{P}\mathbb{Z} \mid (\exists y : \mathbb{Z} \bullet x = y + y)\}$$

Notation

- >+> partial injective function
- \rightarrowtail total injective function
- -+>> partial surjective function
- >>>> total bijective function

Relation (in infix notation)

$$_abs_: \mathbb{Z} \leftrightarrow \mathbb{N}$$
$$\forall m: \mathbb{Z}, n: \mathbb{N} \bullet (m \ abs \ n) \leftrightarrow (m = n \lor -m = n)$$

Relation (in infix notation)

$$_abs _: \mathbb{Z} \leftrightarrow \mathbb{N}$$
$$\forall m : \mathbb{Z}, n : \mathbb{N} \bullet (m \ abs \ n) \leftrightarrow (m = n \lor -m = n)$$

Function

$$abs: \mathbb{Z} \to \mathbb{Z}$$
$$abs = (\lambda m : \mathbb{Z} \mid m \le 0 \bullet -m) \cup (\lambda m : \mathbb{Z} \mid m \ge 0 \bullet m)$$

Relation (in infix notation)

$$_abs _: \mathbb{Z} \leftrightarrow \mathbb{N}$$
$$\forall m : \mathbb{Z}, n : \mathbb{N} \bullet (m \ abs \ n) \leftrightarrow (m = n \lor -m = n)$$

Function

$$abs: \mathbb{Z} \to \mathbb{Z}$$
$$abs = (\lambda m : \mathbb{Z} \mid m \le 0 \bullet -m) \cup (\lambda m : \mathbb{Z} \mid m \ge 0 \bullet m)$$

Function (in prefix notation)

$$abs _: \mathbb{Z} \to \mathbb{Z}$$
$$\forall x : \mathbb{Z} \mid x \le 0 \bullet x = -(abs x)$$
$$\forall x : \mathbb{Z} \mid x \ge 0 \bullet x = abs x$$

Finite subsets of $\ensuremath{\mathbb{Z}}$

$$m..n = \{n' : \mathbb{N} \mid m \le n' \land n' \le n\}$$

Finite subsets of $\ensuremath{\mathbb{Z}}$

$$m..n = \{n' : \mathbb{N} \mid m \le n' \land n' \le n\}$$

Finite sets

$\mathbb{F} T$ consists of the finite sets in $\mathbb{P} T$

$$\begin{bmatrix} S \end{bmatrix} \\ \mathbb{F} : \mathbb{P}(\mathbb{P}S) \\ \mathbb{F} = \{ s : \mathbb{P}S \mid (\exists n : \mathbb{N} \bullet (\exists f : 1..n \rightarrowtail s \bullet true)) \}$$

Cardinality operator

$$= [S] =$$

$$\# : \mathbb{F}S \to \mathbb{N}$$

$$\forall s : \mathbb{F}S; n : \mathbb{N} \bullet (n = \#s \leftrightarrow (\exists f : 1..n \rightarrowtail s \bullet true))$$

Notation

- > finite (partial) injective functions(e.g. duplicate-free arrays) $S \rightarrow T = \{f: S \rightarrow T \mid dom f \in \mathbb{F}S\}$

Definition

$$\operatorname{seq} T == \{s : \mathbb{Z} \twoheadrightarrow T \mid \operatorname{dom} s = 1..\#s\}$$

Note

- sequences are functions, which are relations, which are sets
- **•** the length of *s* is #*s*

Definition

$$\operatorname{seq} T == \{s : \mathbb{Z} \twoheadrightarrow T \mid \operatorname{dom} s = 1..\#s\}$$

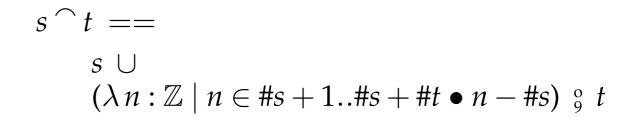
Note

- sequences are functions, which are relations, which are sets
- the length of s is #s

Notation

The sequence
$$\{1 \mapsto x_1, 2 \mapsto x_2, \ldots, n \mapsto x_n\}$$
is written as $\langle x_1, x_2, \ldots, x_n \rangle$

Example: Concatenation of Sequences



General form

_Name_____ SymbolDeclarations

ConstrainingPredicates

Linear notation

Name \cong [*SymbolDeclarations* | *ConstrainingPredicates*]

With empty predicate part

Linear notation

Name \cong [*SymbolDeclarations*]

Theater tickets

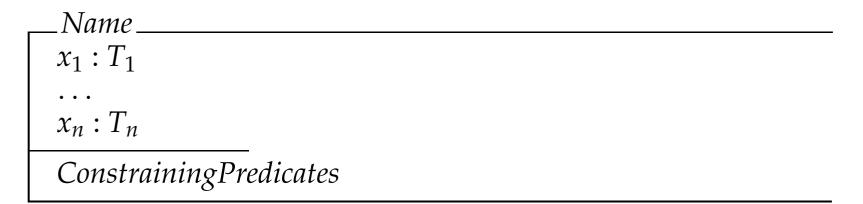
[Seat] [Person]

__TicketsForPerformance0_ seating : ℙ Seat sold : Seat → Person

dom sold \subseteq seating

Schemata as Sets/Types

Schema



can be seen as the following set (type) of tuples:

Name = $\{x_1: T_1; \ldots; x_n: T_n \mid Constraining Predicates \bullet (x_1, \ldots, x_n)\}$

Schema Inclusion

Inclusion

Schemata can be used (included) in

- schema
- set comprehension
- quantification

by adding the schema name to the declaration part

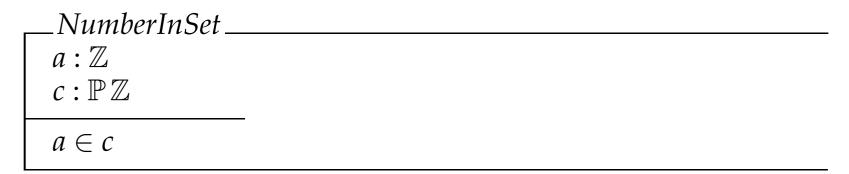
Meaning

- declarations
- constraining predicates

are added to the corresponding parts of the including schema / set comprehension / quantification

Note: Matching names merge and must be type compatible

Example



{*NumberInSet* $| a = 0 \bullet c$ }

is the same as

 $\{a: \mathbb{Z}, c: \mathbb{P}\mathbb{Z} \mid a \in c \land a = 0 \bullet c\}$

(the set of all integer sets containing 0)

Schemata can be used as predicates in

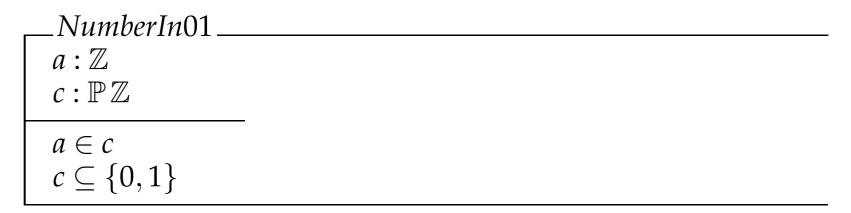
- schema
- set comprehension
- quantification

by adding the schema name to the predicate part (occurring variables must already be declared)

Meaning

The constraining predicates (not: the declaration part) are added to the corresponding part of the schema / set comprehension / quantification

Example



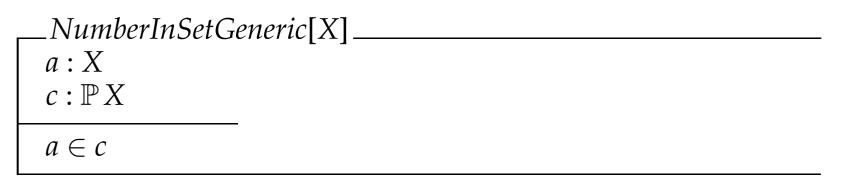
 $\forall a : \mathbb{Z}; c : \mathbb{P}\mathbb{Z} \mid NumberIn01 \bullet NumberInSet$

is the same as

 $\forall a : \mathbb{Z}; c : \mathbb{P}\mathbb{Z} \mid a \in c \land c \subseteq \{0, 1\} \bullet a \in c$

Type/set variables can be used in schema definitions

Example



Then

 $NumberInSetGeneric[\mathbb{Z}] = NumberInSet$

Variables in schemata can be renamed

Example

NumberInSet[a/q, c/s]

is equal to

$q:\mathbb{Z}\ s:\mathbb{P}\mathbb{Z}$		
$q \in s$		

Conjunctions of Schemata

Schemata can be composed conjunctively

Example

Given

Then the following are equivalent

 $\textit{ConDis1} \land \textit{ConDis2}$

Disjunctions of Schemata

Schemata can be composed disjunctively

Example

Given

Then the following are equivalent

 $\textit{ConDis1} \lor \textit{ConDis2}$

$$a:A; b:B; c:C$$
$$P \lor Q$$

Informal specification

Theater: Tickets for first night are only sold to friends

Specification in Z

Status ::= *standard* | *firstNight*

 $\begin{array}{l} _Friends _\\ friends : \mathbb{P} \ Person \\ status : Status \\ sold : Seat \rightarrow Person \\ \hline status = firstNight \Rightarrow ran sold \subseteq friends \end{array}$

 $TicketsForPerformance1 \cong TicketsForPerformance0 \land Friends$

and

__TicketsForPerformance1 _ Friends TicketsForPerformance0 $TicketsForPerformance1 \cong TicketsForPerformance0 \land Friends$

and

__TicketsForPerformance1 _ Friends TicketsForPerformance0

are the same as

__TicketsForPerformance1 _____ friends : ℙ Person; status : Status sold : Seat → Person; seating : ℙ Seat

 $status = firstNight \Rightarrow ran sold \subseteq friends$ **dom** sold \subseteq seating

Normalisation

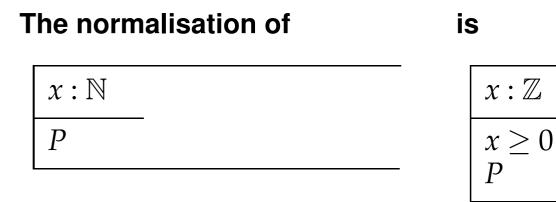
- A schema is normalised if in the declaration part
- Variables are typed
- but not restricted to subsets of types

Normalisation

A schema is normalised if in the declaration part

- Variables are typed
- but not restricted to subsets of types

Example



A schema is negated by negating the predicate part in its normalised form

Example

The negation of

 $x:\mathbb{N}$

Р

which is

$$x:\mathbb{Z}$$
$$\neg (x \in \mathbb{N} \land P)$$

is the negation of

$$\begin{array}{c} x:\mathbb{Z}\\ \hline x\in\mathbb{N}\\ P \end{array}$$

States

- A state is a variable assignment
- A schema describes a set of states

Operations

To describe an operation, a schema must describe pairs of states (pre/post)

States

- A state is a variable assignment
- A schema describes a set of states

Operations

To describe an operation, a schema must describe pairs of states (pre/post)

Notation

Variables are decorated with ' to refer to their value in the post state

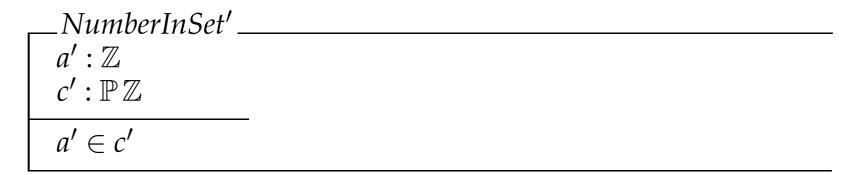
Whole schemata can be decorated

Schemata as Operations

Example

NumberInSet'

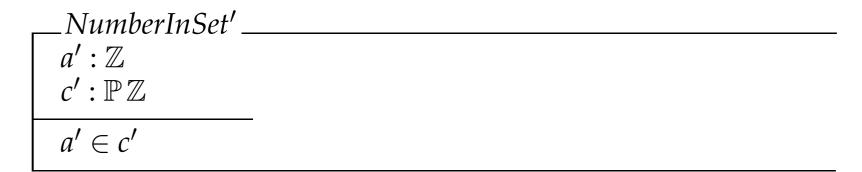
is the same as



Example

NumberInSet'

is the same as



Further decorations

- input variables are decorated with "?"
- output variables are decorated with "!"

Example

Theater: Selling tickets

 $_Purchase0_\\TicketsForPerformance0\\TicketsForPerformance0'\\s?:Seat\\p?:Seat\\p?:Person\\s? \in seating \setminus dom sold\\sold' = sold \cup \{s? \mapsto p?\}\\seating' = seating$

(no output variables in this schema)

Response ::= *okay* | *sorry*

$$Success ______ r! : Response$$
$$r! = okay$$

Then

 $Purchase0 \land Success$

is a schema that reports successful ticket sale

Schemata as Operations: General Form

$$\underbrace{StateSpace}_{x_1: T_1; \dots; x_n: T_n}$$

$$\underbrace{inv(x_1, \dots, x_n)}$$

$$\begin{array}{c} _ Operation _ \\ StateSpace \\ StateSpcae' \\ i_1?: U_1; \ldots; i_m?: U_m \\ o_1!: V_1; \ldots; o_p!: V_p \\ \hline pre(i_1?, \ldots, i_m?, x_1, \ldots, x_n) \\ op(i_1?, \ldots, i_m?, x_1, \ldots, x_n, x'_1, \ldots, x'_n, o_1!, \ldots, o_p!) \end{array}$$

Definition

 $\Delta Schema$ **abbreviates** *Schema* \wedge *Schema*'

General form of operation schema using Δ

Definition

 $\exists Schema \quad abbreviates \quad \Delta Schema \land (x_1 = x'_1 \land \ldots \land x_n = x'_n)$

where $x_1, \ldots x_n$ are the variables declared in *Schema*

General form of operation schema using $\boldsymbol{\Xi}$

$$_ Operation _ \\ \exists StateSpace \\ i_1?: U_1; \dots; i_m?: U_m \\ o_1!: V_1; \dots; o_p!: V_p \\ pre(i_1?, \dots, i_m?, x_1, \dots, x_n) \\ op(i_1?, \dots, i_m?, x_1, \dots, x_n, o_1!, \dots, o_p!) \\ \end{cases}$$

Using $\boldsymbol{\Xi}$ indicates that the operation does not change the state

The Operators Δ and Ξ : Example

The following schemata are equivalent

 $\Xi NumberInSet$

 $\Delta NumberInSet$ a = a' c = c'

NumberInSet NumberInSet' a = a'c = c'

Theater: Selling tickets, but only to friends if first night performance

```
\begin{array}{l} \_Purchase1 \\ \Delta TicketsForPerformance1 \\ s?: Seat \\ p?: Person \\ \hline s? \in seating \backslash dom sold \\ status = firstNight \Rightarrow (p? \in friends) \\ sold' = sold \cup \{s? \mapsto p?\} \\ seating' = seating \\ status' = status \\ friends' = friends \\ \end{array}
```

Example

_NotAvailable ______ ETicketsForPerformance1 s? : Seat p? : Person

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

 $\begin{array}{l} TicketServiceForPerformance \cong \\ (Purchase1 \land Success) \lor \\ (NotAvailable \land Failure) \end{array}$

Quantifying (Hiding) Variables in Schemata

Schema quantification

 $\forall x : S \bullet Schema \qquad \text{resp.} \\ \exists x : S \bullet Schema \qquad \end{cases}$

(existential quantification is also called "variable hiding")

Quantifying (Hiding) Variables in Schemata

Schema quantification

 $\forall x : S \bullet Schema \qquad \text{resp.} \\ \exists x : S \bullet Schema \qquad \end{cases}$

(existential quantification is also called "variable hiding")

Example

 $\exists a : \mathbb{Z} \bullet NumberInSet$

is the same as

$$c: \mathbb{P}\mathbb{Z}$$
$$\exists a: \mathbb{Z} \bullet a \in c$$

Definition

Operation schemata can be composed using *^o*, where

- every variable with ' in the first schema must occur without ' in the second schema
- these variables are identified and
- hidden from the outside

Composition: General form

$$\begin{array}{c} Op1 \\ \hline x_1:T_1; \dots; x_p:T_p \\ z_1:V_1; \dots; z_n:V_n \\ z'_1:V_1; \dots; z'_n:V_n \\ \hline op1(x_1,\dots,x_p, \\ z_1,\dots,z_n, z'_1,\dots,z'_n) \end{array} \qquad \begin{array}{c} Op2 \\ y_1:U_1;\dots;y_q:U_q \\ z_1:V_1;\dots;z_n:V_n \\ \hline op2(y_1,\dots,y_q, \\ z_1,\dots,z_n,z'_1,\dots,z'_n) \\ \end{array}$$

$$\begin{array}{c}
Op1 \circ Op2 \\
x_1 : T_1; \dots; x_p : T_p \\
y_1 : U_1; \dots; y_q : U_q \\
z_1 : V_1; \dots; z_n : V_n \\
z'_1 : V_1; \dots; z'_n : V_n \\
\exists z''_1 : V_1; \dots; z''_n : V_n \bullet \\
op1(x_1, \dots, x_p, z_1, \dots, z_n, z''_1, \dots, z''_n) \\
op2(y_1, \dots, y_q, z''_1, \dots, z_n, z'_1, \dots, z'_n)
\end{array}$$

Purchase1 % Purchase1[s?/s2?]

is equivalent to

 $\Delta TicketsForPerformance1$ s? : Seat; s2? : Seat; p? : Person $s? \in seating \setminus dom sold$ $s2? \in seating \setminus dom(sold \cup \{s? \mapsto p?\})$ $status = firstNight \Rightarrow (p? \in friends)$ $sold' = sold \cup \{s? \mapsto p?, s2? \mapsto p?\}$ seating' = seating status' = statusfriends' = friends