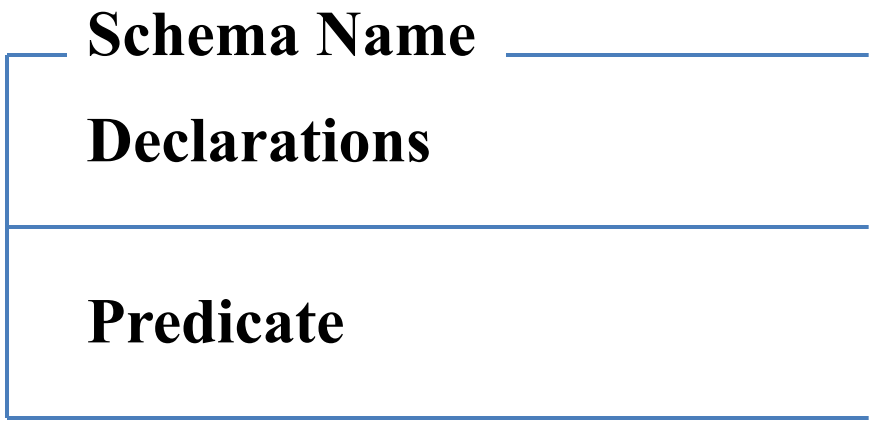
1. Schemas

- **Schema**
  - Structuring specifications
- **General diagrammatic form of schema**



- **Text form of schema**
  - SchemaName == [Declarations | Predicate]
  - The operator '==' (abbreviation definition) means 'stands for'.

1. Schemas

- **Declarations and predicates**
  - Each line of declarations is regarded as being terminated by a semicolon, i.e., several lines consist of a sequence.
  - Several lines of predicates are regarded as being joined by and operators.
- **Anonymous schema**
  - The schema name is omitted
- **Schema without predicate part**
  - A schema without predicate part simply declares a new variable or variables without applying a constraining predicate.
- **Local variables**
  - A variable introduced by a schema is local to that schema and may only be referenced in another schema by explicitly including the variable's defining schemas.

1. Schemas

- **Global variables**
  - Global variables, which are available throughout the specification, are introduced by an axiomatic definition.
  - In general, the values of global variables cannot be changed by operations of the specification.
- **Axiomatic definition of global variables**

Declarations

Declarations

Predicate

2. Schema Calculus

- **Schema calculus**
  - Schemas can be regarded as units and manipulated by various operators that are analogous to logical operators.
- **Schema operators**
  - Decoration
  - Inclusion
  - Conjunction
  - Disjunction
  - Delta convention
  - Xi convention
  - Renaming
  - Hiding
  - Projection
  - Composition

2. Schema Calculus

- **Decoration**
  - The schema name S decorated with a prime, S', is defined to be the same as the schema S with its all variables decorated with a prime.
  - It is used to signify the value of a schema after some operation has been carried out.

S

$a, b : \mathbb{N}$

$a < b$

S'

$a', b' : \mathbb{N}$

$a' < b'$

2. Schema Calculus

- **Inclusion**
  - The name of schema S can be included in the declarations of another schema. The effect is for the included schema to be textually imported: its declarations are merged with those of the including schema and its predicated part is conjoined with that of the including schema.

IncludeS

$c : \mathbb{N}$

S

$c < 10$

IncludeS

$c : \mathbb{N}$

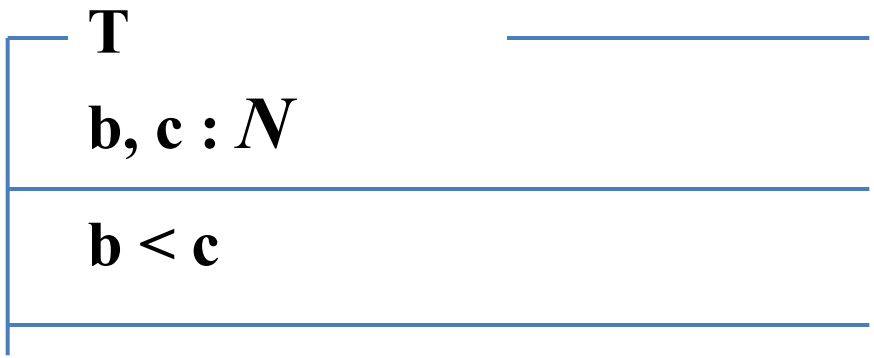
$a, b : \mathbb{N}$

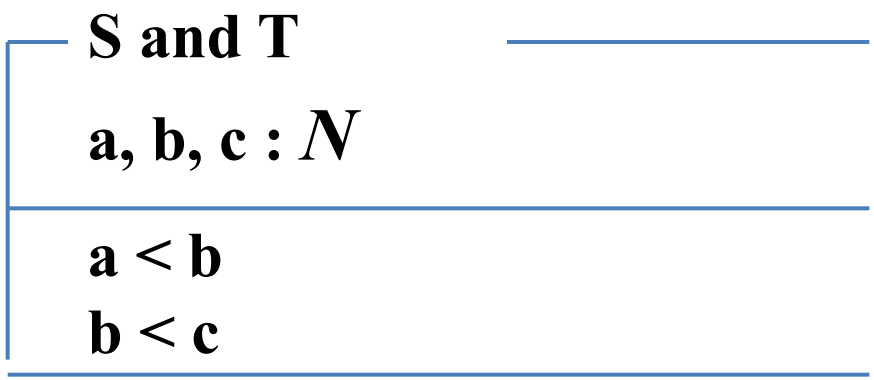
$c < 10$

$a < b$

2. Schema Calculus

- **Conjunction**
  - **Two schemas S and T can be joined by a schema conjunction operator, written like the logic conjunction operator. The effect is to make a new schema with the declarations of the two component schemas merged and their predicated conjoined.**





2. Schema Calculus

Disjunction

- Two schemas S and T can be joined by a schema disjunction operator, written like the logical disjunction operator. The effect is to make a new schema with the declarations of the two component schemas merged and their predicates disjoined.

T

$b, c : \mathbb{N}$

$b < c$

S or T

$a, b, c : \mathbb{N}$

$a < b \lor b < c$

2. Schema Calculus

- **Delta convention**
  - **A schema with the Greek character capital delta (‘Δ ’) as the first character of its name, such as Δ S, is defined to be:**

$$\Delta S$$
$$a, b : \mathbb{N}$$
$$a', b' : \mathbb{N}$$
$$a < b$$
$$a' < b'$$

  - **The Greek delta character, in Δ S, is used, as in other areas of mathematics, to signify a change in S.**

2. Schema Calculus

- **Xi convention**
  - A schema with the Greek character capital xi (‘Ξ’) as the first character of its name, such as Ξ S, is defined to be the same as ΔS but with the constraint that the new value of every variable is the same as the old:

$$
\begin{array}{|l}
\Xi S \\
a, b : \mathbb{N} \\
a', b' : \mathbb{N} \\
\hline
a < b \\
a' < b' \\
a = a' \\
b = b'
\end{array}
$$

  - The Greek symbol is chosen for its visual similarity to the equivalence symbol, ≡, showing that the new state is equivalent to the old.

2. Schema Calculus

Renaming

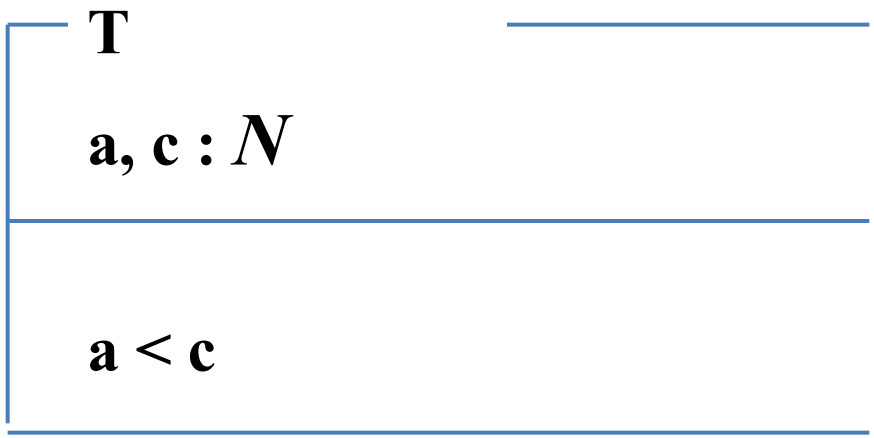
- The observation in a schema may be renamed by the following form:

  newSchemaName ==

  oldSchemaName [newName1/oldName1,

  newName2/oldName2, …]

  T == S [c / b]

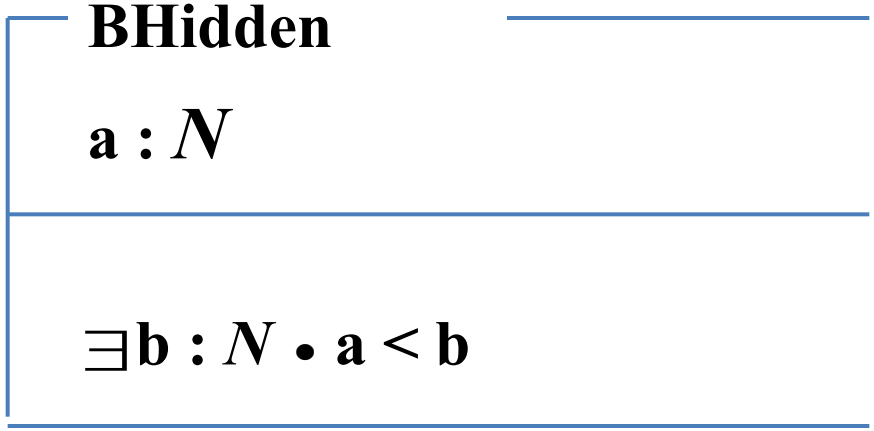


2. Schema Calculus

- **Hiding**
  - The schema hiding operator hides specified variables so that they are no longer variables of the schema and simply become local variables of existential operators in the predicate part of the schema.
  - newSchemaName ==

    oldSchemaName \ (varName1, varName2, …)

    Bhidden == S \ (b)



2. Schema Calculus

- **Projection**
  - **Schema projection is similar to hiding except all but the named variables are hidden.**
  - **newSchemaName ==**

    oldSchemaName ↑ (varName1, varName2, …)

    Aprojected == S ↑ (a)

AProjected

$a : \mathbb{N}$

$\exists b : \mathbb{N} \bullet a < b$

2. Schema Calculus

- **Composition**
  - **The composition of schemas S with schema T is written as S ; T and signifies the effect of doing S, then doing T.**
  - **It is equivalent to renaming the variables describing the after state of S to some temporary names and the equivalent variables describing the before state of T with the same temporary names and then hiding the temporary names.**
- **Example**
  - **PressRight == CursorControlKey ∧ [key?: KEY | key? = right]**
  - **PressLeft == CursorControlKey ∧ [key?: KEY | key? = left]**
  - **PressRight ; PressLeft ==**

    (PressRight [tempCol/column', tempLine/line'] ∧

    PressLeft [tempCol/colimn, tempLine/line]) \ (tempCol, tempLine)

3. Input and Output Variables

- **Input variables**
  - Finishing the variable's name with a question mark (?) indicates that the variable is an input to the system.
- **Output variables**
  - Finishing the variable's name with an exclamation mark (!) indicates that the variable is an output from the schema.
- **Example**

Add

$a?, b? : \mathbb{N}$

$sum! : \mathbb{N}$

$sum! = a? + b?$

3. Example of Schemas with Input

KEY ::= home | return | left | right | up | down

$$
\begin{array}{|l}
\textbf{numLines} : \mathbb{N} \\
\textbf{numColumns} : \mathbb{N} \\
\hline
1 \leq \textbf{numLines} \\
1 \leq \textbf{numColumns}
\end{array}
$$

3. Example of Schemas with Input

Cursor

line : $\mathbb{N}$

column : $\mathbb{N}$

---

line $\in$ 1.. numLines

column $\in$ 1..numColumns

HomeKey

Δcursor

key? : KEY

---

key? = home

line' = 1

column' = 1

3. Example of Schemas with Input

DownKeyNormal

Δ Cursor
key? : KEY

---

key? = down
line < numLines
line' = line + 1
column' = column

DownKeyAtBottom

Δ Cursor
key? : KEY

---

key? = down
line = numLines
line' = 1
column' = column

3. Example of Schemas with Input

ReturnKey

Δ Cursor
key? : KEY

---

key? = return
column' = 1
((line < numLines $\wedge$ line' = line + 1)
$\vee$
(line = numLines $\wedge$ line' = 1))

3. Example of Schemas with Input

RightKey

Δ Cursor

key? : KEY

---

key? = right

((column < numColumns $\wedge$ column' = column + 1 $\wedge$ line' = line)

$\vee$

(column = numColumns $\wedge$ column' = 1 $\wedge$

((line < numLines $\wedge$ line' = line + 1)

$\vee$

(line = numLines $\wedge$ line' = 1))))

CursorControlKey == HomeKey $\vee$ ReturnKey $\vee$ LeftKey $\vee$ RightKey $\vee$ Upkey $\vee$ Downkey

4. Overall Structure of a Z Specification Document

- **Overall structure**
  - A Z specification document consists of mathematical text in the Z notation, interleaved with explanatory text in a natural language. The explanatory text should be expressed in terms of the problem and should not refer directly to the mathematical formulation.
- **Sections**
  - Introduction.
  - The types used in the specification.
  - The state and its invariant properties.
  - An operation to set the variables to some initial state.
  - Operations and enquiries.
  - Error handling.
  - Final versions of operations and enquiries.