

# Relational Calculus

<http://profs.sci.univr.it/~spoto/compilatori/compilatori08.html>

Watcharee Jumpamule, PhD.  
Department of Computer Science  
Chiangmai University

1

## Relation

- *Relations* are sets of tuples. They can resemble *tables* or *databases*.

ID	NAME	DEPT
0018	Frank	Admin
0308	Philip	Research
080	Iris	Research
...	...	...

2

# BINARY RELATION

- Definition

A binary relation is a subset of a Cartesian product. The relation may restrict the association between the elements of the two sets, whereas the Cartesian product allows the association between every pair of elements in the two sets.

3

## Defining Relation

There are 3 ways

- Relation as a type

- $accounts : AccountHolderID \leftrightarrow Account$

- $accounts : \mathbb{P}(AccountHolderID \times Account)$

- Relation enumeration

- $accounts = \{(121, Account\_A), (171, Account\_B)\}$

The type of elements inside *accounts* and the type of the relation *accounts* both must have been declared earlier before enumeration.

- Relation comprehension

- $accounts = \{n : \mathbb{N}; a : \{Account\_A, Account\_B\} \mid 120 \leq n \leq 172 \cdot (n, a)\}$

Type of *accounts* is derived from the declaration.

Notice that  $\leftrightarrow$  can be used only to declare binary relations whereas the other syntax can be used to declare any relation.

4

# Relational Calculus I

- The tool-kit provides many operators for binary relations.
- *Domain* and *range* are the sets of first and second components, respectively.

$\text{dom phone} = \{ \dots, \text{aki}, \text{philip}, \text{doug}, \text{frank}, \dots \}$

$\text{ran phone} = \{ \dots, 4019, 4107, 4136, 0113, \dots \}$

- *Relational image* can model table lookup.

The expression  $R(S)$  defines the relational image of  $R$  that contains the set of range elements of the relation mapped by the domain elements in the set  $S$ . You can also say  $R(S) = \text{ran}(S \triangleright R)$ .

$\text{phone}(\{ \text{doug}, \text{philip} \}) = \{ 4107, 4136, 0113 \}$

5

# Relational Calculus II

- *Restriction operators* can model database queries.
- *Domain restriction* selects pairs based on their first component.

$\{ \text{doug}, \text{philip} \} \triangleleft \text{phone} = \{ (\text{philip}, 4107), (\text{doug}, 4107),$   
 $(\text{doug}, 4136), (\text{philip}, 0113) \}$

- *Range restriction* selects according to the second element.

$\text{phone} \triangleright (4000..4999) = \{ (\text{aki}, 4019), (\text{philip}, 4107),$   
 $(\text{doug}, 4107), (\text{doug}, 4136), \dots \}$

- *Overriding* can model database updates.

The expression  $R1 \oplus R2$  defines the overriding of relation  $R1$  by the elements of  $R2$ . The type of domain elements must be the same in both relations and so is the type of range elements. Typically, it follows the steps listed below:

– Identify the domain elements that are common to both  $R1$  and  $R2$ . Let  $S$  denote the set of these common domain elements.

– Remove those ordered pairs in  $R1$  whose domain elements are in  $S$ ; i.e., perform  $S \triangleleft R1$ .

- Include all elements from  $R2$  into  $R1$ .

$\text{phone} \oplus \{ (\text{heather}, 4026), (\text{aki}, 4026) \} = \{ \dots, (\text{aki}, 4026), (\text{philip}, 4107),$   
 $(\text{doug}, 4107), (\text{doug}, 4136), (\text{philip}, 0113), (\text{frank}, 0110), (\text{frank},$   
 $6190), (\text{heather}, 4026), \dots \}$

6

# Relational Calculus III

- The expression  $R \triangleright S$  denotes the complement of  $R \triangleright S$ ; that is, it returns those ordered pairs of the relation whose range elements are not in the set  $S$ .
- The expression  $S \triangleleft R$  denotes the complement of  $S \triangleleft R$ ; that is, it returns those ordered pairs of the relation whose domain elements are not in the set  $S$ .
- The inverse of a relation is a relation in which the domain and range elements of the original relation are switched.
- An identity relation is one in which every element is related to itself. Consequently, the relation is defined from a set to itself.
- If  $R_1 : A \leftrightarrow B$  and  $R_2 : B \leftrightarrow C$  are two relations, then the relational composition  $R_1 \circ R_2$  is a relation  $R$  defined as  $R : A \leftrightarrow C$  such that for every ordered pair  $(a, b) \in R_1$  and  $(b, c) \in R_2$ , a new ordered pair  $(a, c) \in R$  is created.
  - *Composition* merges two relations by combining pairs that share a matching component.

7

## Some examples using relational operators

- Let  $X = \{1, 2, 3, 4, 5\}$ ,  $Y = \{a, b, c, d, e\}$ , and  $Z = \{\theta, \phi, \psi\}$   
Let  $R : X \leftrightarrow Y$ ,  $S : Y \leftrightarrow Z$  and  $T : X \leftrightarrow X$   
Let  
 $R = \{(1 \mapsto a), (1 \mapsto b), (2 \mapsto b)\}$   
 $S = \{(a \mapsto \theta), (c \mapsto \phi), (c \mapsto \psi)\}$   
 $T = \{(1 \mapsto 1), (2 \mapsto 2), (3 \mapsto 3), (4 \mapsto 4), (5 \mapsto 5)\}$   
 $\text{dom } R = \{1, 2\}$ ,  $\text{dom } S = \{a, c\}$ ,  $\text{dom } T = \{1, 2, 3, 4, 5\}$   
 $\text{ran } R = \{a, b\}$ ,  $\text{ran } S = \{\theta, \phi, \psi\}$ ,  $\text{ran } T = \{1, 2, 3, 4, 5\}$
- $T = \text{id } X$   
 $R \circ S = S \circ R = \{(1 \mapsto \theta)\}$

8

## Some examples using relational operators

$$\{1\} \triangleleft R = \{(1 \mapsto a), (1 \mapsto b)\}$$

$$\{1, 2\} \triangleleft R = \{(1 \mapsto a), (1 \mapsto b), (2 \mapsto b)\}$$

$$\{1\} \triangleleft R = \{(2 \mapsto b)\}$$

$$\{2\} \triangleleft R = \{(1 \mapsto a), (1 \mapsto b)\}$$

$$R \triangleright \{a\} = \{(1 \mapsto a)\}$$

$$R \triangleright \{a, b\} = \{(1 \mapsto a), (1 \mapsto b), (2 \mapsto b)\}$$

$$R \triangleright \{a\} = \{(1 \mapsto b), (2 \mapsto b)\}$$

$$R \triangleright \{b\} = \{(1 \mapsto a)\}$$

9

## Some examples using relational operators

$$R \sim = \{(a \mapsto 1), (b \mapsto 1), (b \mapsto 2)\}$$

$$R(\{1\}) = \{a, b\}$$

$$R(\{1, 2\}) = \{a, b\}$$

$$R(\{2\}) = \{b\}$$

$$R \oplus \{(2 \mapsto c)\} = \{(1 \mapsto a), (1 \mapsto b), (2 \mapsto c)\}$$

$$R \oplus \{(1 \mapsto c)\} = \{(1 \mapsto c), (2 \mapsto b)\}$$

$$R \oplus \{(2 \mapsto c), (3 \mapsto d), (5 \mapsto a)\} =$$

$$\{(1 \mapsto a), (1 \mapsto b), (2 \mapsto c), \\ (3 \mapsto d), (5 \mapsto a)\}$$

10

# Relation's Conclusions

- To remember: A relation is a set.
- Therefore, all operators applied to sets can equally be applied to relations.
- If  $R_1$ ,  $R_2$  and  $R_3$  are relations, you can say

$$\begin{aligned}R_1 \cap R_2 &= \emptyset \\R_3 &= R_1 \cup R_2 \\R_2 &= R_3 \setminus R_1 \\(a \mapsto b) &\in R_1 \\&\dots\end{aligned}$$

11

# FUNCTIONS

- *Functions* are binary relations where each element in the domain appears just once. Each domain element is a *unique key*.

phonef : NAME  $\rightarrow$  PHONE

phonef = {..(aki,4019), (philip, 4107), (doug, 4107),  
(frank, 6180), ...}

- *Function application* is a special case of relational image. It associates a domain element with its unique range element.

phonef (doug) = 4107

phonef(doug) = 4107

phonef doug = 4107

12

# BINARY RELATIONS AND FUNCTIONS

$X \leftrightarrow Y$	Binary relations: many-to-many
$X \rightarrow Y$	Partial functions: many-to-one    Some function applications undefined
$X \rightarrow Y$	Total functions:    All function applications defined
$X \rightsquigarrow Y$	Partial injections: one-to-one    Inverse is also a function
$X \rightsquigarrow Y$	Total injections
$X \rightsquigarrow Y$	Bijections: cover entire range (onto)

All have type  $\mathbb{P}(X \times Y)$

13

## LINKED DATA STRUCTURES

- Binary relations can model linked data structures such as lists, graphs and trees.
- Each *arc* can be modelled by a pair of *nodes*.
- For example a *tree* can be modelled by its *child* function.

*child*  $\text{NODE} \rightarrow \text{NODE}$

*child* = { (a, b), (a, c), (b, d), (b, e), (c, f), (c, g), (c, h) }

- The *transitive closure* operator builds the *descendant* relation from the *child* relation.

*descendant* : *child*<sup>+</sup>

*descendant* = { (a, b), (a, c), (a, d), (a, e), ... }

14

# SEQUENCE

- Sequences can model *arrays* and *lists*.
- A sequence represents an ordered collection of elements, all belonging to the same type. Each element has a distinct position in the sequence.
- A sequence may have duplicate elements. A sequence with no duplicates is called an injective sequence.
- In  $\mathbb{Z}$ , the starting index of a sequence is 1.
- Individual elements of a sequence are accessed using subscripts. A valid subscript ranges from 1 to  $N$  where  $N$  is the length of (or the number of elements in) the sequence.
- A sequence is synonymous to an array in programming languages but a sequence can be infinite.
- Two sequences are said to be equal if and only if they contain the same elements at the same positions.

15

## Examples

- $\langle \text{Smith, Anna, Brucei} \rangle$
- $\langle 10, 2, 7 \rangle$
- $\langle 1, \text{asi}, 2, \text{John}, \text{Ai} \rangle$  is an invalid sequence because it has elements of different types.
- The sequence  $\langle 1, 15, 9 \rangle$  and the sequence  $\langle 1, 9, 15 \rangle$  are different even though they contain the same set of elements; the elements are in different positions.

16

# Defining a Sequence

- A sequence can be defined in three different ways: Sequence as a type, Sequence Enumeration, and Sequence Comprehension

- **Sequence as a type**; used to declare a variable of sequence type

*users* : seq*User*

– *users* is a sequence whose elements belong to the type *User* .

- Examples

*users* = ⟨ Al , Jeff ,Kristie ⟩

*users* = ⟨ Michael ,Michael ,Michael ⟩

*users* = ⟨ ⟩

*staff* : seq1 *Faculty*

– *staff* is a non-empty sequence whose elements belong to the type *Faculty*.

- Examples

*staff* = ⟨ Al , Jeff, Kristie ⟩

*staff* = ⟨ Michael, Michael ⟩

17

## example

*animals* : iseq*Animal*

– *animals* is an injective sequence whose elements belong to the type *Animal* ; this sequence does not contain duplicates.

- Examples

• *animals* = ⟨ Sherley, Lucky, Vicky ⟩

• *animals* = ⟨ Sherley ⟩

18

# Sequence Enumeration

- used to enumerate all elements of the sequence including their respective positions.
- useful for finite sequences only

$$S = \langle \text{ounce, pint, gallon} \rangle$$

- S is a finite sequence whose elements are ounce, pint and gallon in that order. The type of these elements must have been defined earlier.

$$N = \langle 10, 100, 1000, 10000 \rangle$$

- N is a finite sequence whose elements are integers.

19

# Sequence Comprehension

- used to define (potentially) infinite sequences

$$\text{Odd} = \langle n : \mathbb{Z} \mid n \bmod 2 = 1 \rangle$$

- Odd defines a (potentially infinite) sequence of odd numbers, derived from the infinite set of integers. The elements of Odd are arranged in the same order in which they appear in  $\mathbb{Z}$ .

- What if I define the following?

$$\text{TopUniversities} = \langle \text{univ} : \text{University} \mid \text{univ.rank} \leq 1 \rangle$$

20

# Exercises

Write sequence expressions for the following:

1. Declare two variables to denote two sequences representing old and new computers. You can choose the type name you want.
2. Declare a non-empty sequence of new computers.
3. Declare an injective sequence of old computers.
4. Define an enumerated sequence that consists of all numbers between 1 and 50 that are divisible by 6.
5. Define using sequence comprehension the sequence containing all numbers between 1 and 1000 that are multiples of 17.

21

## Operators on Sequences

Operator	Synopsis	Meaning
#	# S	length of sequence S
$\diamond$	$\diamond$	empty sequence
$\wedge$	$S_1 \wedge S_2$	sequence concatenation
<i>rev</i>	<i>rev</i> S	reverse of S
<i>head</i>	<i>head</i> S	first element of S
<i>last</i>	<i>last</i> S	last element of S
<i>tail</i>	<i>tail</i> S	S with its first element removed
<i>front</i>	<i>front</i> S	S with its last element removed
$\wedge/$	$\wedge/$ SS	distributed concatenation of sequences; SS is a sequence of sequences

22

# Operators on Sequences(cont.)

Operator	Synopsis	Meaning
$\subseteq$	$S \subseteq T$	S is a prefix of T
suffix	S suffix T	S is a suffix of T
in	S in T	S is a segment inside T
$\upharpoonright$	$U \upharpoonright S$	extract the elements from S; U is an index set. returns a subsequence of S
$\upharpoonright$	$S \upharpoonright V$	extract the elements of set V from S; returns a subsequence of S

23

## Examples for sequence operators

- Let  $S1 = \langle a, b, c, d \rangle$ ,  $S2 = \langle a, b \rangle$ ,  $S3 = \langle f, g, h \rangle$ ,  $S4 = \langle b, c \rangle$ ,  $S5 = \langle c, d \rangle$
- $SS = \langle S1, S2, S3 \rangle$
- $\#S1 = 4$  and  $\#S3 = 3$
- $S2 \neq \langle \rangle$ . . . not an empty sequence
- S1, S2 and S3 are all injective sequences.
- $S1 \frown S2 = \langle a, b, c, d, a, b \rangle$
- $rev(S1) = \langle d, c, b, a \rangle$
- $head(S1) = a$
- $tail(S2) = \langle b \rangle$
- $front(S3) = \langle f, g \rangle$
- $last(S3) = h$
- $\frown / SS = \langle a, b, c, d, a, b, f, g, h \rangle$
- $S2 \subseteq S1$  is true
- $S4 \subseteq S1$  is false
- $S5$  suffix S1 is true

24

- S4 suffix S1 is false
- S4 in S1 is true
- S5 in S1 is true
- $\langle \rangle \subseteq S2$  is true
- $S3 \subseteq S3$  is true
- S1 suffix S1 is true
- $\{2, 3\} \upharpoonright S1 = \langle b, c \rangle$
- $S1 \upharpoonright \{b, d, f, g\} = \langle b, d \rangle$

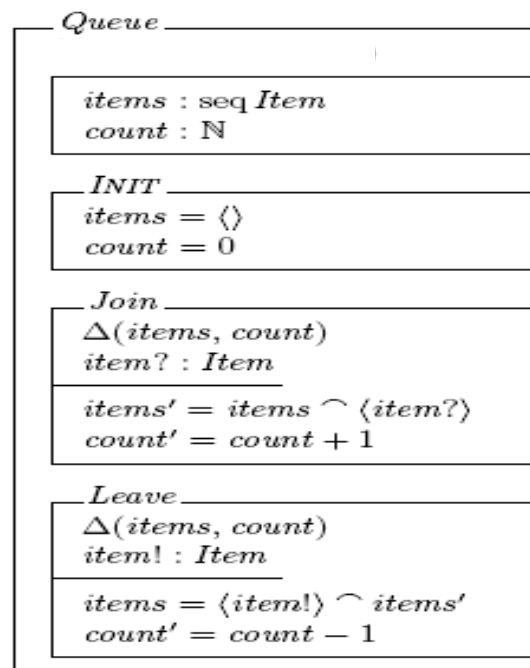
25

## Specification of Queue

- Queue is a sequence of elements. The operation rule of queue is FIFO(First In First Out). There are 3 operations operate on this queue; *count*, *join* and *leave*. Initially, the queue is empty.
- *count* --- to count the number of elements in queue.
- *join* --- add a new element into the end of queue.
- *leave* --- extract the head element from queue.

26

- [Item]



27

## An Example

- Employment Exchange

28

# SCHEMAS and SCHEMA CALCULUS

- Schema: math in a box, with a name attached.
- Unique to Z.
- Schema calculus builds large schemas from smaller ones.
- Model states and operations.
- Can be used as declarations, predicates, expressions, types, sets ...

29

## STATE SCHEMA

- Simple text editor with limited memory.

[CHAR]

TEXT == seq CHAR

maxsize: $\mathbb{N}$
maxsize $\leq 65535$

- The editor state is modelled by two state variables, the texts to the left and right of the cursor.

Editor
left, right: TEXT
# (left $\wedge$ right) $\leq$ maxsize

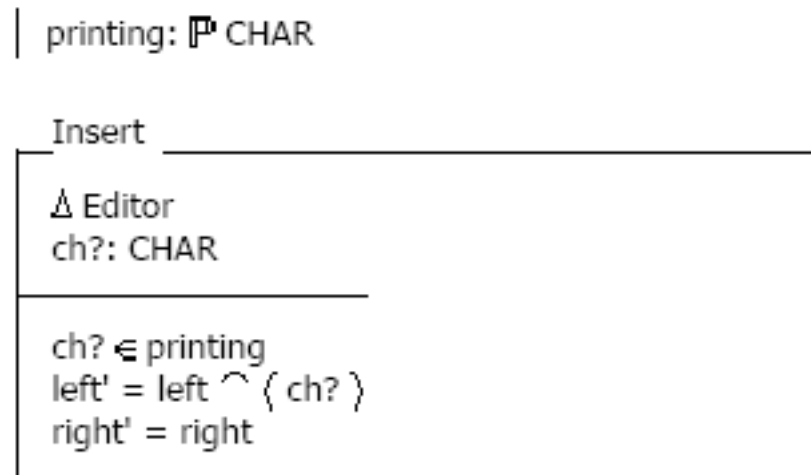
- The editor starts up empty.

Init  $\hat{=}$  [ Editor | left = right = ( ) ]

30

# OPERATION SCHEMA

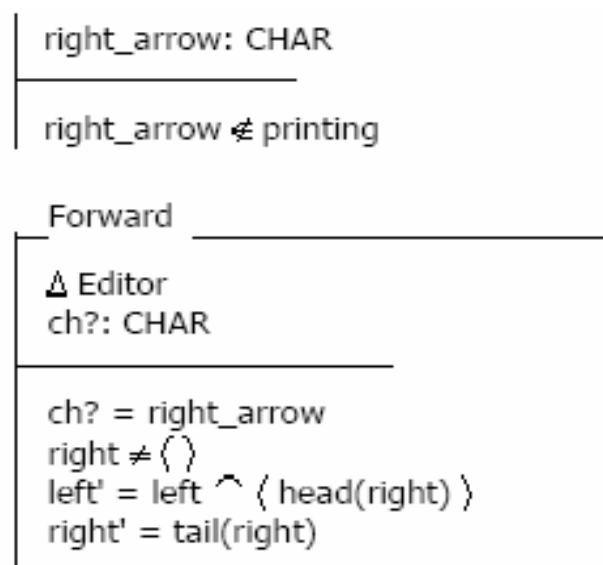
- The *Insert* operation inserts a printing character to the left of the cursor.



31

# PRECONDITION AND PARTIAL OPERATION

- The *Forward* operation moves the cursor forward.
- 
- The *precondition* is the predicate that must be true for the operation to be defined, involving only inputs and unprimed “before” variables. *Forward* is a *partial* operation. It does not cover the situation where the cursor is already at the end of the buffer,  $right = \langle \rangle$



32

# SCHEMA CALCULUS

- In *EOF* states the cursor is at end-of-file.

EOF
Editor
right = {}

- Box up right\_arrow for schema expressions.

RightArrow
ch?: CHAR
ch? = right_arrow

- 

- Xi in  $\exists Editor$  indicates no change in state.

$$T\_Forward \cong Forward (EOF \wedge RightArrow \wedge \exists Editor)$$

- *T\_Forward* is a *total* operation that is defined in all *Editor* states.

33

# SCHEMA CALCULUS

- SCHEMA INCLUSION, EXPANSION

- Schemas can be used like macros.

- Any *included* schema name (or *schema reference*) can be *expanded* by replacing the name by the text.

- Merge declarations, combine predicates.

- 

Quotient
$n, d, q, r: \mathbb{N}$
$d \neq 0$ $n = q*d + r$

- Division  $\cong$  Quotient  $\wedge$  Remainder

Remainder
$r, d: \mathbb{N}$
$r < d$

34

Any schema expression can be expressed as a single schema box.

$T\_Forward \hat{=} Forward$   
 $\vee (EOF \wedge RightArrow \wedge \text{Editor})$

means the same as

<p>T_Forward</p> <hr/> <p>left, right, left', right': TEXT  ch?: CHAR</p> <hr/> <p># (left ^ right) ≤ maxsize  # (left' ^ right') ≤ maxsize  ch? = right_arrow  ((right ≠ { }  ^ left' = left ^ { head right }  ^ right' = tail right) ∨  (right = { } ^ left' = left ^ right' = right))</p>
--

35

## SCHEMA CALCULUS

- Division  $\hat{=} Quotient \wedge Remainder$

<p>Quotient</p> <hr/> <p>n, d, q, r: <math>\mathbb{N}</math></p> <hr/> <p>d ≠ 0  n = q*d + r</p>
<p>Remainder</p> <hr/> <p>r, d: <math>\mathbb{N}</math></p> <hr/> <p>r &lt; d</p>

<p>Division</p> <hr/> <p>n,d,q,r: <math>\mathbb{N}</math></p> <hr/> <p>d ≠ 0  r &lt; d  n = q*d + r</p>
---

36

# OTHER SCHEMA CALCULUS

- There is a schema operator for every logical connective and quantifier.
- Conjunction and disjunction are most useful.
- Schema negation is un-intuitive, not very useful.
- Two operators are not based on logical connectives:
  - $S \circ T$  Schema composition
  - $S \gg T$  Schema piping
- Composition is like relational composition and piping is like conjunction, *not* sequencing.
- Z is not a programming language!

37

# SCHEMAS AS DECLARATIONS, PREDICATES

Schemas can be used like macros to write compact formulas.

``When an editor is in its initial state, the cursor is at the end of the file''

$\forall$  Editor | Init • EOF

This expands to an ordinary predicate.

$\forall$  left, right: TEXT | # (left  $\wedge$  right)  $\leq$  maxsize •  
left = right =  $\langle \rangle \Rightarrow$  right =  $\langle \rangle$

38

# GENERIC DEFINITIONS

- A definition of sequence concatenation ( $\_ \hat{\_}$ ) that applied to only one type would not be very useful.

$\_ \hat{\_} : \text{TEXT} \times \text{TEXT} \rightarrow \text{TEXT}$
...

- The tool-kit provides this *generic definition*.

[X]
$\_ \hat{\_} : \text{seq } X \times \text{seq } X \rightarrow \text{seq } X$
...

- $X$  is a *formal generic parameter* that can be *instantiated* with an *actual generic parameter*.
- Generic definitions can use patterns.
- $X \leftrightarrow Y \equiv \mathbb{P}(X \times Y)$
- Generic definitions can extend the Z notation.

39

# GENERIC SCHEMA

- *Generic schemas* have type parameters.

Pool[RESOURCE]
owner: RESOURCE $\rightarrow$ USER free: $\mathbb{P}$ RESOURCE
(dom owner) $\cup$ free = RESOURCE (dom owner) $\cap$ free = $\emptyset$

- RESOURCE could be instantiated with different types for memory pages, disk blocks, etc.

40

# FREE TYPE

- Free types can model *recursive data types* used to build linked structures.
- Example: define syntax of simple arithmetic expressions on natural numbers, such as  $2+3$  and  $(12 \text{ div } (2+3)) - 7$ :
- $OP ::= \text{plus} \mid \text{minus} \mid \text{times} \mid \text{divide}$
- $EXP ::= \text{const } \langle\langle \mathbb{N} \rangle\rangle \mid \text{binop } \langle\langle OP \times EXP \times EXP \rangle\rangle$