Lecture # 06

Relations & Functions

BINARY RELATION

- Although we may define relations that express links between any finite number of objects, it is enough to employ binary relations: relations that express links between pairs of objects.

- In our mathematical language, a relation is a set of ordered pairs, a subset of a Cartesian product. If X and Y are sets, then $X \leftrightarrow Y$ denotes the set of all relations between X and Y . The relation symbol may be defined by generic abbreviation:
  - $X \leftrightarrow Y == P(X \times Y)$

- Any element of $X \leftrightarrow Y$ is a set of ordered pairs in which the first element is drawn from X , and the second from Y : that is, a subset of the Cartesian product set $X \times Y$

BINARY RELATION

For example, the set

$$\{a, b : \mathbf{N} \mid a + b = 4 \cdot (a, b)\}$$

defines the relation

$$\{(0, 4), (1, 3), (2, 2), (3, 1), (4, 0)\}.$$

Note that since ordered pairs are only equal if their corresponding elements are equal the relation contains both $(0, 4)$, $(1, 3)$, and $(4, 0)$, $(3, 1)$.

A relation is used to express the fact that there is a connection between the elements that make up an ordered pair.

BINARY RELATION

A relation is used to express the fact that there is a connection between the elements that make up an ordered pair. In the constructive specification of a relation it is the predicate which defines this connection. For example, the predicate $a + b = 4$ used in the constructive specification above expresses the fact that the elements of each ordered pair that make up the relation always add up to 4.

Formally, *dom* is defined as

$$\operatorname{dom} A = \{t_1 \colon T_1 \mid \exists t_2 \colon T_2 \cdot t_1 A t_2\}$$

Formally define the *rng* operator.

$$\operatorname{rng} A = \{t_2 \colon T_2 \mid \exists t_1 \colon T_1 \cdot t_1 A t_2\}$$

BINARY RELATION

If the domain of *phone* includes every employee who can be reached by telephone: *aki, doug,* and the others and the range of *phone* includes all the numbers that have been assigned to telephones: 4117,4017, and so forth, we can write :

$$\mathit{phone} : \mathit{NAME} \leftrightarrow \mathit{PHONE},$$

Binary relations are sets of pairs.

$$\mathbb{P}\,(\mathrm{NAME} \times \mathrm{PHONE})$$

or

$$\mathrm{NAME} \leftrightarrow \mathrm{PHONE}$$

Relational image **can model table lookup**

$$\mathrm{phone}\ (\!|\{\ \mathrm{doug},\ \mathrm{philip}\ \}|\!)\ = \{\ 4107,\ 4136,\ 0113\ \}$$

Binary relations can model lookup tables.

NAME	PHONE
Aki	4019
Philip	4107
Doug	4107
Doug	4136
Philip	0113
Frank	0110
Frank	6190
...	...

ran phone = { ..., 4019, 4107, 4136, 0113, ... }

dom phone = { ..., aki, philip, doug, frank, ... }

Domain & Range Restriction Operators

In system specifications there is a need for relations to be restricted over their domain or range.

Domain Restriction :

- The domain restriction and range restriction operators can model database queries.
- The domain restriction $\lhd$ operator selects tuples based on the values of their first elements
- Its first argument is a set of elements from the domain of a relation, its second argument is a relation, and its value is the matching tuples from the relation.
- To retrieve all the tuples for Doug and Philip from the phone relation, we apply domain restriction as:

$$\{doug, philip\} \lhd phone =$$

$$\{philip \mapsto 4107, \\ doug \mapsto 4107, \\ doug \mapsto 4136, \\ philip \mapsto 0113\}$$

Domain & Range Restriction Operators

Range Restriction :

- The range restriction ▻ operator selects tuples based on the values of their second elements.
- Its first argument is a relation, its second argument is a set of elements from the range, and its value is the matching tuples.
- To retrieve all the tuples that have numbers in the 4000s from the phone relation, we apply range restriction:

$$
\begin{array}{l}
\mathit{phone} \rhd (4000 \,..\, 4999) = \{ \\
\quad \vdots \\
\quad \mathit{aki} \mapsto 4117, \\
\quad \mathit{philip} \mapsto 4107, \\
\quad \mathit{doug} \mapsto 4107, \\
\quad \mathit{doug} \mapsto 4136, \\
\quad \vdots \\
\quad \}
\end{array}
$$

Domain & Range Restriction Operators

- We can combine domain and range restriction . This expression finds the numbers for Doug and Philip in the 4000s:

$$\{doug, philip\} \lhd phone \rhd (4000 \,..\, 4999) =$$

$$\begin{aligned}&\{philip \mapsto 4107,\\ &doug \mapsto 4107,\\ &doug \mapsto 4136\}\end{aligned}$$

- There are also domain and range anti-restriction operators ⩤ and ⩥ respectively. S ⩤ R is the binary relation R, except without the pairs whose first element is in S, and R ⩥ T is R without the pairs whose second element is in T.

Override Operator

- The *override* operator $\oplus$ can model database updates. Both of its arguments are relations.

- Its value is a relation that contains the tuples from both relations, except that tuples in the second argument replace any tuples from the first argument that have the same first component.

- This has the effect of adding new tuples and replacing old ones.

- For example:

$$
\begin{aligned}
&\mathit{phone} \oplus \{\mathit{heather} \mapsto 4026, \mathit{aki} \mapsto 4026\} = \{\\
&\quad \vdots\\
&\quad \mathit{aki} \mapsto 4026,\\
&\quad \mathit{philip} \mapsto 4107,\\
&\quad \mathit{doug} \mapsto 4107,\\
&\quad \mathit{doug} \mapsto 4136,\\
&\quad \mathit{philip} \mapsto 0113,\\
&\quad \mathit{frank} \mapsto 0110,\\
&\quad \mathit{frank} \mapsto 6190,\\
&\quad \mathit{heather} \mapsto 4026,\\
&\quad \vdots\\
&\quad \}
\end{aligned}
$$

Inverse Operator

- The inverse operator reverses the direction of a binary relation by exchanging the first and second components of each pair. It is a postfix unary operator that is notated as a tilde ~.

- The inverse of the phone relation is a reverse directory from telephone numbers to names:

$$
\begin{aligned}
&\mathit{phone}^{\sim} = \{\\
&\quad \vdots\\
&\quad 4117 \mapsto \mathit{aki},\\
&\quad 4107 \mapsto \mathit{philip},\\
&\quad 4107 \mapsto \mathit{doug},\\
&\quad 4136 \mapsto \mathit{doug},\\
&\quad 0013 \mapsto \mathit{philip},\\
&\quad 0110 \mapsto \mathit{frank},\\
&\quad 6190 \mapsto \mathit{frank},\\
&\quad \vdots\\
&\quad \}
\end{aligned}
$$

Composing relations

When we have several relations that describe the same collection of objects, we can make inferences by forming chains of associations from different relations.

Relational composition formalizes this kind of reasoning: It merges two relations into one by combining pairs that share a matching component.

For example, we can infer employees' departments from their telephone numbers.

This is possible because each pool of telephone numbers is assigned to a different department, as described by the *dept* relation:

$$
\begin{aligned}
&dept : PHONE \leftrightarrow DEPARTMENT \\
&dept = \{ \\
&\qquad 0000 \mapsto administration, \\
&\qquad \vdots \\
&\qquad 0999 \mapsto administration, \\
&\qquad 4000 \mapsto research, \\
&\qquad \vdots \\
&\qquad 4999 \mapsto research, \\
&\qquad 6000 \mapsto manufacturing, \\
&\qquad \vdots \\
&\qquad 6999 \mapsto manufacturing\}
\end{aligned}
$$

Composing relations

- The range of *phone* matches the domain *of dept,* so we can *compose* the two relations. Match up pairs from *phone* and *dept* that contain the same phone number, then form new pairs from these, with just the name and department.

- For example, we match *philip* $\mapsto$ 0113 from *phone* with 0113 $\mapsto$ *administration* from *dept,* obtaining *philip* $\mapsto$ *administration.*

- When we perform all such matches, we obtain a new relation with domain *NAME* and range *DEPARTMENT.* The *relational composition* symbol $⨾$ notates this operation:

$$
\begin{array}{l}
phone ⨾ dept = \{ \\
\quad \vdots \\
\quad aki \mapsto research, \\
\quad philip \mapsto research, \\
\quad doug \mapsto research, \\
\quad philip \mapsto administration, \\
\quad frank \mapsto administration, \\
\quad frank \mapsto manufacturing, \\
\quad \vdots \\
\quad \}
\end{array}
$$

FUNCTION

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

A **function** is an important type of relation. It has the property that each element of its domain is associated with just *one* element of its range. Thus,

$$\{(1, \text{file2}), (3, \text{file4}), (7, \text{file2}), (6, \text{filetax}), (9, \text{fileupd})\}$$

is an example of a function while

$$\{(1, \text{file2}), (3, \text{file5}), (7, \text{file3}), (1, \text{file5}), (2, \text{file4})\}$$

is not an example since 1 is associated with both the files *file2* and *file5*. When a pair of elements occurs in a function it is said that the function **maps** the first element to the second element. Thus, in the function above 1 is mapped to *file2* and 6 is mapped to *filetax*.

PARTIAL FUNCTION

A **partial function** is a function whose domain is a proper subset of the set from which the first elements of its pairs is taken. Thus, the function

$$\{(1,3), (4,9), (8,3)\}$$

over $\mathbf{N} \times \mathbf{N}$ is a partial function because its domain: $\{1,4,8\}$ is a proper subset of the natural numbers.

A function R over $T_1 \times T_2$ is a partial function if and only if

$$\forall t_1: T_1; t_2, t_3: T_2 \cdot (t_1 R t_2 \wedge t_1 R t_3) \Rightarrow t_2 = t_3.$$

TOTAL FUNCTIONS

An important type of function is a **total function**. This is a function whose domain is equal to the set from which the first elements of its pairs is taken. For example, if the set *sysprogs* is

$$\{\text{archiver, editor, compilerA, compilerB, filer}\}$$

and *location* is a function over $sysprogs \times \mathbf{N}$

$$\{(\text{archiver}, 12), (\text{editor}, 480), (\text{compilerA}, 903),$$
$$(\text{compilerB}, 202), (\text{filer}, 17)\},$$

then *location* is a total function because its domain

$$\{\text{archiver, editor, compilerA, compilerB, filer}\}$$

is equal to *sysprogs*.

TOTAL FUNCTIONS

Formally, a function R over $T_1 \times T_2$ is total if

$$\forall t_1 : T_1 ; t_2, t_3 : T_2 \cdot (t_1 R t_2 \wedge t_1 R t_3 \Rightarrow t_2 = t_3) \wedge \operatorname{dom} R = T_1$$

In general, a **total function is** usually just another **name** for a regular **function**. The use of the term **is** to make it clear that the **function is** defined for all elements in its domain, compared to partial **functions** which are only defined for part of the domain.

Functions as Lambda Expressions

An alternative way of writing functions which is often used in mathematics is known as a **lambda expression**. The general form of a lambda expression is $\lambda\,\boxed{\text{Signature}} \mid \boxed{\text{Predicate}} \cdot \boxed{\text{Term}}$

The signature establishes the types of the variables used.

The predicate gives a condition which each first element of every pair in the function must satisfy;

the term gives the form of the second element of each pair in the function.

Functions as Lambda Expressions

An example of a lambda expression is

$$\lambda m : \mathbf{N} \mid m > 4 \cdot m + 5$$

It denotes the infinite function

$$\{(5, 10), (6, 11), \ldots ,\}$$

Another example is

$$\lambda x : 0..10 \mid (x, x^2)$$

which is a finite function which maps natural numbers between 0 and 10 to a pair whose first element is the natural number and the second element its square, i.e.

$$\{(0, (0, 0)), (1, (1, 1)), (2, (2, 4)), \ldots , (10, (10, 100))\}$$

SEQUENCES

- Sets are *unordered* collections; it is not meaningful to speak of the first or last element in a set, or whether one element follows another.

- When we write a set, we have to write down the elements in some order, but the ordering we choose is not significant.

- In many situations the ordering of elements is significant. These are modelled by the *sequence.*

- Sequences can model arrays, lists, queues, and other sequential structures.

- A sequence of items from set *S* is declared seq S; sequences are notated inside angle brackets.

SEQUENCES

- The days of the week form a sequence. First we need to declare the names of all the days.

  DAYS ::= friday | monday | Saturday | Sunday | thursday | tuesday | Wednesday

- There is no ordering implied by this definition. To express the ordering, we need to define sequences:

  weekday : seq *DAYS*

  weekday = ⟨*monday, tuesday, Wednesday, thursday, friday*⟩

SEQUENCES

Sequence operators include *head* and *concatentation*, $\frown$.

head weekday = monday

week == ⟨ sunday ⟩ $\frown$ weekday $\frown$ ⟨ saturday ⟩

Here we use the concatenation operator twice to make the entire week

Sequences are functions, and functions are sets.

weekday = { 1 ↦ monday, 2 ↦ tuesday, ... }

weekday 3 = wednesday

week = 7

FURTHER READING

APPENDIX D “The Z mathematical tool-kit”
- Jonathan Jacky-The Way of Z_ Practical Programming with Formal Methods -Cambridge University Press