Relations

Jan van Eijck

May 11, 2003

Abstract

We introduce relations in an abstract manner, and explain some fundamental relational notions.

Preview: Relations

Consider the type of `actP` in the database program in Chapter 4.

```
STAL> :t actP
actP :: ([Char],[Char]) -> Bool
```

This characterizes sets of pairs.

And the type of queries:

```
STAL> :t q2
q2 :: [([Char],[Char])]
```

This gives lists of pairs.

A relation is simply a set of pairs.

More precisely:

If A is a set, then a relation on A is a subset of A^2.

Relations as Boolean Functions and as Sets of Pairs

A relation R on a set A can be viewed as a function of type $A \to A \to$ Bool.

Example: $<$ on $\mathbb{N}$. For every two numbers $n, m \in \mathbb{N}$, the statement $n < m$ is either true or false. E.g., $3 < 5$ is true, whereas $5 < 2$ is false. In general: to a relation you can "input" a pair of objects, after which it "outputs" either *true* or *false*. depending on whether these objects are in the relationship given.

Alternative view: relation R on A as a set of pairs:

$$\{(a, b) \in A^2 \mid Rab \text{ is true } \}.$$

$< \text{ on } \mathbb{N} = \{(0,1), (0,2), (1,2), (0,3), (1,3), (2,3), \ldots\}$

Domain and Range

The set $\mathrm{dom}\,(R) = \{x \mid \exists y\,(\,Rxy\,)\}$, the set of all first coordinates of pairs in R, is called the *domain* of R.

The set $\mathrm{ran}(R) = \{y \mid \exists x\,(\,Rxy\,)\}$, the set of second coordinates of pairs in R, is called the *range* of R.

The relation R is a relation *from* A *to* B or *between* A and B, if $\mathrm{dom}\,(R) \subseteq A$ and $\mathrm{ran}(R) \subseteq B$.

A relation from A to A is called *on* A.

$R = \{(1,4),(1,5),(2,5)\}$ is a relation from $\{1,2,3\}$ to $\{4,5,6\}$, and it also is a relation on $\{1,2,4,5,6\}$. Furthermore, $\mathrm{dom}\,(R) = \{1,2\}$, $\mathrm{ran}(R) = \{4,5\}$.

Identity and Inverse

$\Delta_A = \{ (a, b) \in A^2 \mid a = b \} = \{ (a, a) \mid a \in A \}$ is a relation on A, the *identity on* A.

If R is a relation between A and B, then $R^{-1} = \{ (b, a) \mid aRb \}$, the *inverse* of R, is a relation between B and A.

The inverse of the relation 'parent of' is the relation 'child of'.

$A \times B$ is the biggest relation from A to B.

$\emptyset$ is the smallest relation from A to B.

For the usual ordering $<$ of $\mathbb{R}$, $<^{-1} = >$.

$(R^{-1})^{-1} = R$; $\Delta_A^{-1} = \Delta_A$; $\emptyset^{-1} = \emptyset$ and $(A \times B)^{-1} = B \times A$.

Properties of Relations

A relation R is **reflexive** on A if for every $x \in A$: xRx.

A relation R on A is **irreflexive** if for no $x \in A$: xRx.

A relation R on A is **symmetric** if for all $x, y \in A$: if xRy then yRx.

Fact: A relation R is symmetric iff $R \subseteq R^{-1}$, iff $R = R^{-1}$.

A relation R on A is **asymmetric** if for all $x, y \in A$: if xRy then not yRx.

A relation R on A is **antisymmetric** if for all $x, y \in A$: if xRy and yRx then $x = y$.

A relation R on A is **linear** (or: has the comparison property) if for all $x, y \in A$: xRy or yRx or $x = y$.

A relation R on A is **transitive** if for all $x, y, z \in A$: if xRy and yRz then xRz.

A relation R on A is **intransitive** if for all $x, y, z \in A$: if xRy and yRz then not xRz.

Classifying Relations with Relational Properties

A relation R on A is a **pre-order** if R is transitive and reflexive.

A relation R on A is a **strict partial order** if R is transitive and irreflexive.

Fact: every strict partial order is asymmetric.

Fact: every transitive and asymmetric relation is a strict partial order.

A relation R on A is a **partial order** if R is transitive, reflexive and antisymmetric.

Fact: Every strict partial order R on A is contained in a partial order (given by $R \cup \Delta_A$).

A relation R on A is a **total order** if R is transitive, reflexive, antisymmetric and linear.

A relation R on A is an **equivalence relation** if R is reflexive, symmetric and transitive.

Closures of Relations

If $\mathcal{O}$ is a set of properties of relations on a set A, then the $\mathcal{O}$-closure of a relation R is the smallest relation S that includes R and that has all the properties in $\mathcal{O}$.

The most important closures are the *reflexive closure*, the *symmetric closure*, the *transitive closure* and the *reflexive transitive closure* of a relation.

To show that R is the smallest relation S that has all the properties in $\mathcal{O}$, show the following:

1. R has all the properties in $\mathcal{O}$,
2. If S has all the properties in $\mathcal{O}$, then $R \subseteq S$.

Fact: $R \cup \Delta_A$ is the reflexive closure of R.

Fact: $R \cup R^{-1}$ is the symmetric closure of R.

Composing Relations

Suppose that R and S are relations on A. The *composition* $R \circ S$ of R and S is the relation on A that is defined by

$$x(R \circ S)z \;:\equiv\; \exists y \in A(xRy \wedge ySz).$$

For $n \in \mathbb{N}, n \geq 1$ we define R^n by means of $R^1 := R$, $R^{n+1} := R^n \circ R$.

Fact: a relation R is transitive iff $R^2 \subseteq R$.

Fact: for any relation R on A, the relation $R^+ = \bigcup_{n \geq 1} R^n$ is the transitive closure of R.

Fact: for any relation R on A, the relation $R^+ \cup \Delta_A$ is the reflexive transitive closure of R. Abbreviation for the reflexive transitive closure of R: R^*.

Sets as Ordered Lists without Duplicates

```
insertList x [] = [x]
insertList x ys@(y:ys') = case compare x y of
                              GT -> y : insertList x ys'
                              EQ -> ys
                              _  -> x : ys

deleteList x [] = []
deleteList x ys@(y:ys') = case compare x y of
                              GT -> y : deleteList x ys'
                              EQ -> ys'
                              _  -> ys

list2set :: Ord a => [a] -> Set a
list2set [] = Set []
list2set (x:xs) = insertSet x (list2set xs)
```

```
powerSet :: Ord a => Set a -> Set (Set a)
powerSet (Set xs) =
   Set (sort (map (\xs -> (list2set xs)) (powerList xs)))

powerList  :: [a] -> [[a]]
powerList  [] = [[]]
powerList  (x:xs) = (powerList xs)
                    ++ (map (x:) (powerList xs))

takeSet :: Eq a => Int -> Set a -> Set a
takeSet n (Set xs) = Set (take n xs)

infixl 9 !!!

(!!!) :: Eq a => Set a -> Int -> a
(Set xs) !!! n = xs !! n
```

Implementing Relations as Sets of Pairs

```
type Rel a = Set (a,a)
```

domR gives the domain of a relation.

```
domR :: Ord a => Rel a -> Set a
domR (Set r) = list2set [ x | (x,_) <- r ]
```

ranR gives the range of a relation.

```
ranR :: Ord a => Rel a -> Set a
ranR (Set r) = list2set [ y | (_,y) <- r ]
```

idR creates the identity relation Δ_A over a set A:

```
idR :: Ord a => Set a -> Rel a
idR (Set xs) = Set [(x,x) | x <- xs]
```

The total relation over a set is given by:

```
totalR :: Set a -> Rel a
totalR (Set xs) = Set [(x,y) | x <- xs, y <- xs ]
```

`invR` inverts a relation (i.e., the function maps R to R^{-1}).

```
invR :: Ord a => Rel a -> Rel a
invR (Set []) = (Set [])
invR (Set ((x,y):r)) = insertSet (y,x) (invR (Set r))
```

`inR` checks whether a pair is in a relation.

```
inR :: Ord a => Rel a -> (a,a) -> Bool
inR r (x,y) = inSet (x,y) r
```

The complement of a relation $R \subseteq A \times A$ is the relation $A \times A - R$. The operation of relational complementation, relative to a set A, can be implemented as follows:

```
complR :: Ord a => Set a -> Rel a -> Rel a
complR (Set xs) r = Set [(x,y) | x <- xs, y <- xs,
                                 not (inR r (x,y))]
```

A check for reflexivity of R on a set A can be implemented by testing whether $\Delta_A \subseteq R$:

```
reflR :: Ord a => Set a -> Rel a -> Bool
reflR set r = subSet (idR set) r
```

A check for irreflexivity of R on A proceeds by testing whether $\Delta_A \cap R = \emptyset$:

```
irreflR :: Ord a => Set a -> Rel a -> Bool
irreflR (Set xs) r =
   all (\ pair  -> not (inR r pair)) [(x,x) | x <- xs]
```

A check for symmetry of R proceeds by testing for each pair $(x, y) \in R$ whether $(y, x) \in R$:

```
symR :: Ord a => Rel a -> Bool
symR (Set []) = True
symR (Set ((x,y):pairs))
   | x == y = symR (Set pairs)
   | otherwise = inSet (y,x) (Set pairs)
                 && symR (deleteSet (y,x) (Set pairs))
```

A check for transitivity of R tests for each couple of pairs $(x, y) \in R, (u, v) \in R$ whether $(x, v) \in R$ if $y = u$:

```
transR :: Ord a => Rel a -> Bool
transR (Set []) = True
transR (Set s) = and [ trans pair (Set s) | pair <- s ]
  where
      trans (x,y) (Set r) =
        and [ inSet (x,v) (Set r) | (u,v) <- r, u == y ]
```

Now what about relation composition? This is a more difficult matter, for how do we implement $\exists z(Rxz \wedge Szy)$? The key to the implementation is the following procedure for composing a single pair of objects (x, y) with a relation S, simply by forming the relation $\{(x, z) \mid (y, z) \in S\}$. This is done by:

```
composePair :: Ord a => (a,a) -> Rel a -> Rel a
composePair (x,y) (Set []) = Set []
composePair (x,y) (Set ((u,v):s))
   | y == u    = insertSet (x,v) (composePair (x,y) (Set s))
   | otherwise = composePair (x,y) (Set s)
```

For relation composition we need set union:

```
unionSet :: (Ord a) => Set a -> Set a -> Set a
unionSet (Set [])     set2  =  set2
unionSet (Set (x:xs)) set2  =
   insertSet x (unionSet (Set xs) (deleteSet x set2))
```

Relation composition is defined in terms of `composePair` and `unionSet`:

```
compR :: Ord a => Rel a -> Rel a -> Rel a
compR (Set []) _ = (Set [])
compR (Set ((x,y):s)) r =
   unionSet (composePair (x,y) r) (compR (Set s) r)
```

Composition of a relation with itself (R^n):

```
repeatR :: Ord a => Rel a -> Int -> Rel a
repeatR r n | n < 1     = error "argument < 1"
            | n == 1    = r
            | otherwise = compR r (repeatR r (n-1))
```