Relations

Klaus Draeger

- Data types in most programs are not independent but connected by **relations.**
- Example: **favouriteColours**
  relates the set of people to the set of colours
- Example: **parent**
  relates people to other people
  (i.e. this is a relation **within** a set, not **between** sets)
- Ubiquitous in computing, e.g.
  - In **databases** (relating products to orders to customers. . .)
  - In OOP (e.g. the parent relation implemented as **father** and **mother** attributes in the **person** class)

Relations in maths

- In maths, a **(binary) relation** between sets X and Y is
  - A set of pairs (x,y) with $x : X$ and $y : Y$
  - A subset of X*Y (which is the set of all such pairs)
  - An element of POW(X*Y) (which contains all such subsets)

– These are all equivalent

- In B notation:
  - favouriteColours : People <-> Colours
  - Here **People <-> Colours** is the type of relations (between the People and Colours sets)
  - Pairs usually written in **maplet** notation e.g. **(Alice |-> green)** or **(3 |-> 4)**
  - ProB also accepts the **(Alice, green)** notation (but Atelier B does not)

Beyond Binary Relations

- The relations we use will be **binary**
  i.e. they involve **two** (possibly equal) sets
- In maths, higher **arities** are possible
  e.g. **ternary** relations: subsets of X*Y*Z
- Often they are essentially combinations of binary relations
- But there are occasional exceptions, especially in maths
- Example: in geometry, the relation
  B(x,y,z) meaning that point y is **between** points x and z

Colours Example

```
SETS
    Colours = {Black, White, Red, Green, Blue};
    People = {Alice, Bob, Carl, Diane}
CONSTANTS
    Favourites
PROPERTIES
    Favourites : People <-> Colours &
    Favourites ={Alice |-> Black, Alice |-> Green,
                 Bob |-> Green, Diane |-> Blue}
```

- There are four sets associated with this relation:
  - The **source** and **target** sets (People and Colours) i.e. the underlying types of the values in the pairs
  - The **domain** and **range**
    dom(Favourites) = {Alice, Bob, Ellen}
    ran(Favourites) = {Black, Green, Blue}
    i.e. the source and target values which **actually occur**

Languages Example

COUNTRY = { France, Canada, England, Wales, . . . }
LANGUAGE = { French, English, Welsh, . . . }
We can define *speaks : COUNTRY <-> LANGUAGE* by

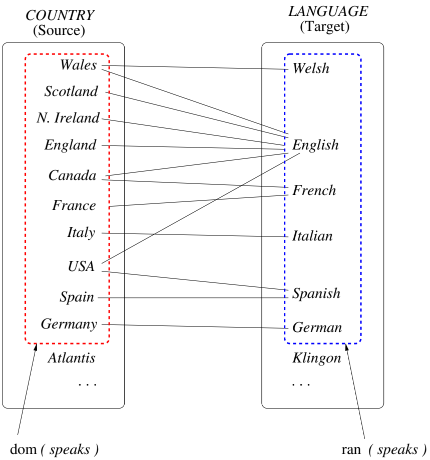
speaks = { *France |-> French*, *Canada |-> French*,
Canada |-> English, *England |-> English*,
Wales |-> Welsh, *Wales |-> English*, . . . }

and its **inverse** *spoken_in : LANGUAGE <-> COUNTRY* by

spoken$_i$*n* = { *French |-> France*, *French |-> Canada*,
English |-> Canada, *English |-> England*,
Welsh |-> Wales, *English |-> Wales*, . . . }

Languages Example

We can represent the *speaks* relation diagrammatically as follows:



Projections of a Relation

- Given an ordered pair **p : X*Y** we can access its parts using the **projection** functions
  - $prj1(X, Y)(p)$ returns the first part (from X)
  - $prj2(X, Y)(p)$ returns the second part (from Y)
  - In ProB you can omit the types (i.e. just **prj1(p)** and **prj1(p)**)
- Examples:
  - prj1(France |-> French) = France
  - prj2(Wales |-> English) = English

Selecting Subsets

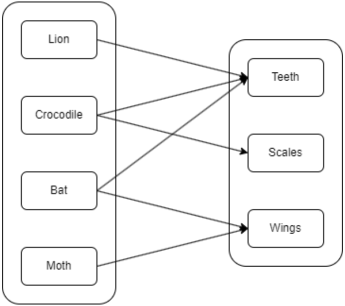
- One use of projections is to pick specific subsets:
- Given **Grid = (1..8)*(1..8)**, then
- We can get **rows and columns** like this:
  - Row1 = {p | p:Grid & prj1(p) = 1}
  - Column3 = {p | p:Grid & prj2(p) = 3}
- Or using **restriction** operators
  - Row1 = {1} <| Grid
  - Column3 = Grid |> {3}
- More Generally,
  - The **domain restriction** A <| R contains those pairs from R whose **first** part is in A
  - The **range restriction** R |>B contains those pairs from R whose **second** part is in B
  - So A <| R = {p | p:R & prj1(p):A} and
    R |> B = {p | p:R & prj2(p):B}

Selecting Subsets

- One use of projections is to pick specific subsets:
- Given **Grid = (1..8)*(1..8)**, then
- We can get the **main diagonal** like this:
  - MainDiag = {p | p:Grid & prj1(p) = prj2(p)}
- Or using the **identity** relation
  - MainDiag = identity(1..8)
  - Generally, identity(A) contains all pairs x|->x where x:A
- We can get the **off-diagonal** like this:
  - MainDiag = {p | p:Grid & prj1(p) + prj2(p) = 9}
    (i.e. it contains the pairs 1|->8, 2|->7, 3|->6, . . .)

Lookups in a Relation

- Related to the restriction operators is the **relational image**
- The domain restriction A<|R gives you all **pairs** x|->y where x is in A
- The **image** R[A] instead gives you just the y values
- For this **features** relation:



features[{Crodocile}] = {Teeth, Scales}
features[{Lion, Moth}] = {Teeth, Wings}

Modifying Relations

- Suppose **favourites : People <-> Colours** is a variable
- The usual way we want to update it is by
  - Changing the favourites of **some** people
  - Leaving the rest unchanged
- This is done by the **override** operator:
  In the relation R <+ S,
  - For those values occurring in dom(S), their values in R are replaced by those in S
  - All others keep the values in R

Override example

- Suppose that favourites is currently
  { Alice |-> black, Alice |-> green,
  Bob |-> red,
  Carl |-> blue,
  Ellen |-> green}
- Then **favourites <+ {Bob |-> green, Dina |-> red}** is
  {Alice |-> black, Alice |-> green,
  Bob |-> green,
  Carl |-> blue,
  Dina |-> red
  Ellen |-> green}
- Where only Bob’s and Dina’s favourites are affected.
- To actually change the variable, use an assignment
  favourites := favourites <+ {. . .}

Relational Algebra

- Interactions between several relations can be used to express important properties.
- This is similar to how **functions** can be composed
  - Which is not a coincidence:
    Functions are just a **special case** of relations
    We will explore this further next week
- This **relational algebra** is also important in e.g. data bases

Relational Algebra: Identity

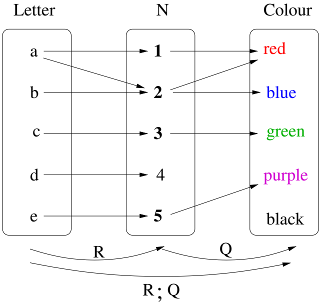
- We previously introduced the **identity** relations
- **identity(A)** is the set of all pairs x|->x where x:A
- For example:
  - X <: NAT1 & X = {1, 2, 3, 4, 5}
  - identity(X) = {1|->1, 2|->2, 3|->3, 4|->4, 5|->5}
- It is called identity because applying it (using the relational image) gives the original set back:
  For any B<:A,
  identity(A)[B] = B

Relational Algebra: Composition

- Given two relations
  R : A<->B
  S : B<->C
- We can **compose them** into
  R;S : A<->C
- This relation contains exactly those pairs x|->z such that:
  There is (at least) one y:B with x|->y in R and y|->z in S

Relational Algebra: Composition

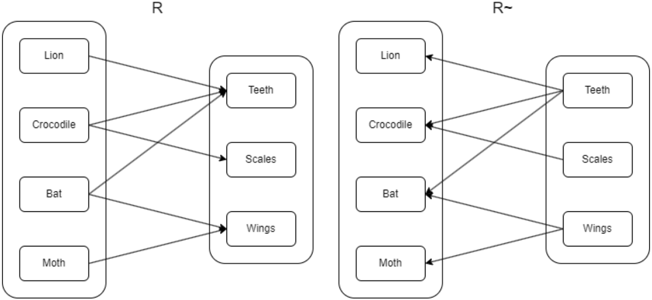
- For relations like these:



  the composition contains x|->z if you can get from x to z following the arrows
- For example, it includes c|->purple
  because there is a path c|->5|->purple

Relational Algebra: Inverse

- The **inverse** relation R$\sim$ is another important tool for relational algebra.
- It contains those pairs x|->y for which y|->x is in R.
- In a diagram, this means that the arrows are reversed:



Relational Algebra

- Relational algebra lets us express many things. Some examples:
  - Given **parent : Person<->Person**, we can define **grandparent = parent ; parent**
  - To find (half-)siblings we can use
    sibling = parent ; parent$\sim$
    since this contains a|->b exactly if there is some c with a|->c in parent and c|->b in parent$\sim$
    i.e. a|->c and b|->c both in parent
    i.e. c is a common parent of a and b
  - The second example is like an **inner join** in SQL (combining rows from two tables with a common key value)

Relational Algebra

- Since relations are sets, we can also use set operations:
- Suppose we are modelling people involved in lawsuits
  - Relations
    plaintiffs : Lawsuit <-> People
    defendants : Lawsuit <-> People
    attorney : People <-> People
  - Then the same attorney must not represent a plaintiff and a defendant in the same lawsuit:
    (plaintiffs ; attorney) /\(defendants ; attorney) = {}