Relations

Klaus Draeger

Lecture 4: More Relations

- There are several special kinds of relations.
  - **Functions** of various kinds (partial/total, injective, surjective)
  - **Equivalence relations** represent ways in which elements of a set are **similar**
  - **Partial orders** according to which a set can be sorted
- These let us make relational specifications more expressive.

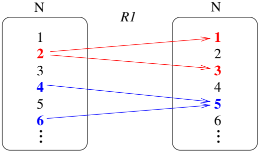
Functions

- A **function** f:A--> B maps each element x:A to the associated **function value** f(x):B
- Mathematically, **f** is a set of pairs (x,y) where each x:A occurs **exactly once**
  i.e. a restricted kind of **relation**
- Example:
  square : NAT1 --> NAT1,
  square = {(1,1), (2,4), (3,9), ...}
- In this case the pairs are often also written like $2 \mapsto 4$ (this is where the **maplet** notation x |-> y comes from)

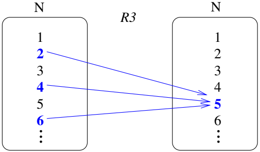
Partial Functions

- The functions we have described are **total**,
  i.e. they assign a value to **every** x:A
- In many cases we have to deal with **partial** functions
  - Example: Some functions are **undefined** in special cases (division by 0, minimum of an empty set, . . .)
  - Example: We can model sequences/arrays like
    2,3,5,7,11
    as finite maps from NAT1 to (in this case) NAT1:
    {1|->2, 2|->3, 3|->5, 4|->7, 5|->11}
    where numbers above 5 are not mapped to anything
- The set of partial functions is written A+->B
- Note that this **includes** the total functions:
  Unmapped elements are **allowed**, not mandatory

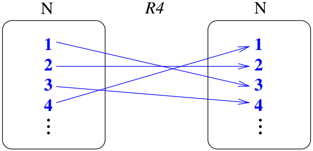
- This relation is **not** a function:



- This is a **partial** function:



- And this is a **total** function:



Function Classes

- Additional properties that functions f : A->B can have are:
  - **Injectivity**: no function value occurs more than once
    i.e. for every y:B there is **at most** one x:A with f(x)=y
  - **Surjectivity**: all possible function values occur
    i.e. for every y:B there is **at least** one x:A with f(x)=y
  - **Bijectivity**: combines injectivity and surjectivity,
    i.e. for every y:B there is **exactly** one x:A with f(x)=y
- The corresponding function classes are represented by
  - placing an additional ">" **before** the arrow for injectivity
  - placing an additional ">" **after** the arrow for surjectivity
  - both for bijectivity
- Example:
  - A >+-> B is the set of **injective partial** functions from A to B
  - A -->> B is the set of **surjective total** functions from A to B

Notation Overview

Relation

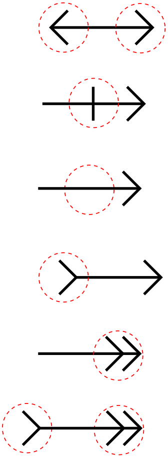
Partial

Total

Injective

Surjective

Bijective



Some examples

- The function f(x)=2*x is
  - **total**: every natural number can be doubled
  - **injective**: doubling different numbers gives different results (remember that these are mathematical integers, not e.g. 32-bit ones which can wrap around)
  - **not surjective**: e.g. we can never get f(x)=1
- The function f(x)=x/2 is
  - **total**: we can always halve (and round down if needed)
  - **not injective**: rounding means that e.g. f(2)=f(3)=1
  - **surjective**: for every y, f(2*y)=y
- The function min(S) is
  - **not total**: gives no result for empty S
  - **not injective**: min({1}) = min({1,2,3}) = 1
  - **surjective**: for every y, min({y}) = y

Function Domains and Ranges

- Like for any relation, a function has
  - a **domain**:
    dom(f) is the set of all x for which f contains a pair (x,y)
  - a **range**:
    ran(f) is the set of all y for which f contains a pair (x,y)
- They are generally **subsets** of the function's **source** and **target** sets
- Then a function f is
  - **total** if dom(f) is the entire source set
  - **surjective** if ran(f) is the entire target set

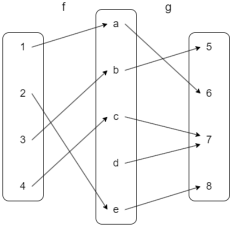
  And we can always interpret f as a total surjective function dom(f)-->>ran(f)

Relational algebra and functions

- Since functions are relations, we can do things like
  - Take their **inverse** $f\sim$
    - This will usually **not** be a function
    - e.g. if f(x) = x/2, then $f\sim$ has both (1,2) and (1,3)
    - But it is a function if f is **injective**
  - **Compose** them, i.e. compute f;g
    - This will always be a function
    - If both f and g are total/injective/surjective then so is f;g
    - **Not** "exactly if": e.g. f;g may be total even if g is not
  - Use the **identity** which is always a total bijective function

Composition example

- Consider this example:

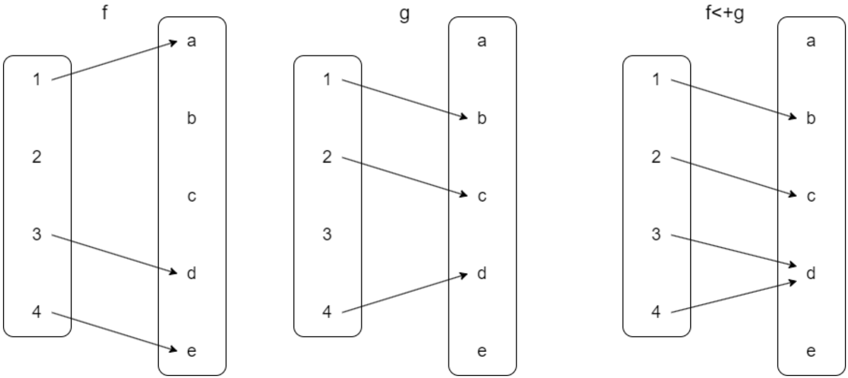


- Here f;g is
  - **total** since both f and g are
  - **injective** even though g is not
  - **surjective** even though f is not
- If we remove the pair (d,7) from g, the composite is **still total** even though g is not

Modifying Functions

- The **override** operator can also be applied to functions
- It works the same way as for general relations:
- In f<+g,
  - For all x in dom(g) the function values are those in g
  - For all other x the values in f (if any) are used
- Note that g<+f will differ
  (Can you tell why?)

Override example



Birthday example

```
SETS
  FAMILY = { Alice, Bob, Cindy }
VARIABLES
  age
INVARIANT
  age : FAMILY --> NAT1
INITIALISATION
  age := {Alice|->21, Bob|->22, Cindy|->23}
OPERATIONS
  birthday(fm) =
  PRE
    fm : FAMILY
  THEN
    age := age <+ { fm|-> (age(fm) + 1)}
  END
```

Birthday example

- The main update in the family example is
  age := age <+ { fm|-> (age(fm) + 1)}
- and the starting state i given by
  age := {Alice|->21, Bob|->22, Cindy|->23}
- Executing **birthday(Cindy)** leads to the new state:

age	:=	age <+ { Cindy |-> (age(Cindy) + 1)}
	=	age <+ { Cindy |-> (23 + 1)}
	=	{Alice|->21, Bob|->22, Cindy|->23} <+ {Cindy|->24}
	=	{Alice|->21, Bob|->22, Cindy|->24}

Exercise: Relations and Functions

Which of the following **relations** are **functions**?
Of those that are functions, which are **total**, which are **injective**, and which are **surjective**?

```
owner       : CAR <-> PERSON

child       : PERSON <-> PERSON

child~      : PERSON <-> PERSON

mother      : PERSON <-> PERSON

birthday    : PERSON <-> DATE

PassportNo  : PERSON <-> NAT1
```

Equivalence Relations

- An **equivalence relation** splits a set into subsets of elements that are **alike** in some sense.
- The usual definition is that R : X<->X is an equivalence relation if
  - It is **reflexive**:
    Each element is equivalent to itself, i.e. (x,x):R
  - It is **symmetric**:
    If (x,y):R then (y,x):R
  - It is **transitive**:
    If (x,y) and (y,z) are in R then so is (x,z)
- Equivalence relations are usually written $\sim$, and the above axioms written as: for all x,y,z:
  - $x \sim x$
  - $x \sim y \Rightarrow y \sim x$
  - $x \sim y, y \sim z \Rightarrow x \sim z$

Uses of Equivalence Relations

- Equivalences are used to **group** data
- Some containers (like Sets) use equivalences to judge which values are "essentially equal"
- One common way of obtaining an equivalence is:
  - Start with the set of values **A**
  - Single out the relevant characteristic – given by a (total) function **f : A-->B**
  - Define x∼y **exactly if** f(x)=f(y) i.e. they agree with respect to the chosen characteristic
- For example,
  - For some event, people may be grouped by age bracket
  - Tasks could be "equally urgent" if they have the same time stamp

Partial Orders

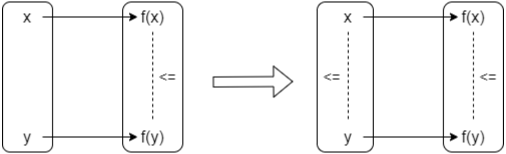
- A **partial order** allows a set to be (partially) sorted.
- The usual definition is that R : X<->X is a partial order if
  - It is **reflexive**:
    Each element is less-or-equal to itself, i.e. (x,x):R
  - It is **anti-symmetric**:
    If (x,y):R and (y,x):R then x=y
  - It is **transitive**:
    If (x,y) and (y,z) are in R then so is (x,z)
- Partial orders are usually written $\leq$, and the above axioms written as: for all x,y,z:
  - $x \leq x$
  - $x \leq y \;\&\; y \leq x \Rightarrow x = y$
  - $x \leq y, y \leq z \Rightarrow x \leq z$

Uses of Equivalence Relations

- Any data that may have to be sorted needs a suitable order
- This often has to satisfy additional properties
- Numbers already come with the standard order pre-defined
- Can define others, for example
  - For numbers: $x \leq y$ if x **divides** y
    (**coarser** than standard order, e.g. 2 and 3 incomparable)
  - For pairs, triples, sequences, . . . :
    lexicographic order, e.g. (1,3,**2**,5,7) < (1,3,**3**,1,8)
  - Using a function **f:A-->B** and an existing order on **B**:
    Define $x \leq y$ **exactly if** $f(x) \leq f(y)$

Pulbacks

- Both for equivalences and orders we have seen examples of this construction:
  - Given
    - Some **structure** on a set **B**
    - A **(total) function f:A-->B**
  - Define the same kind of structure on A by "pulling it back" along f



- This can be done for many kinds of structures
- Can be expressed in relational algebra:
  - If **R** is (e.g.) a partial order in B, and **f:A-->B**
  - Then **f; R; f**$\sim$ is the pullback (of R along f) on A.

Expressing Relationship Types

- Sometimes you may need to specify that (e.g.) R is a partial order on A
- Remember the definition:
  - Reflexivity:
    For all x, (x,x):R
  - Anti-symmetry:
    For all x and y, if (x,y) and (y,x) are in R then x=y
  - Transitivity:
    For all x, y, z if (x,y) and (y,x) are in R then so is (x,z)

These can be expressed in relational algebra as well:

- Reflexivity: id(A) <: R
- Anti-symmetry: (R / R∼) = id(A)
- (Transitivity:) (R ; R) <: R

Expressing Relationship Types

- Sometimes you may need to specify that (e.g.) R is a partial order on A
- Remember the definition:
  - Reflexivity:
    For all x, (x,x):R
  - Anti-symmetry:
    For all x and y, if (x,y) and (y,x) are in R then x=y
  - Transitivity:
    For all x, y, z if (x,y) and (y,x) are in R then so is (x,z)

  These can be expressed in relational algebra as well:
  - Reflexivity: id(A) <: R
  - Anti-symmetry: (R / R∼) = id(A)
  - (Transitivity:) (R ; R) <: R

The DEFINITIONS Clause

- **Shorthands** for longer definitions are often helpful
- This is where the DEFINITIONS clause comes in
- Example:

```
DEFINITIONS
    // Defines when nn is even
    even(nn) == ( (nn mod 2) = 0);
    // Defines when xx,yy,zz are an ascending triple
    ascending(xx,yy,zz) == ( (xx<=yy) & (yy<=zz))
    // Defines when RR is a partial order on AA
    partialOrder(RR,AA) == ( (RR : AA <-> AA)
                           & (id(AA) <: RR)
                           & (RR /\ (RR~) = id(AA) )
                           & ((R ; R) <: R))
```

- Then pre-conditions or invariants can contain e.g. **partialOrder(myOrder, NAT)**
- This also comes in handy when using **logic** – next week's topic