

Computer Science I

Professor Tom Ellman
Lecture 22

Relations

- Some facts involve one object:
“Bill is president”.
- Some facts involve two objects:
“Bill and Hillary are married”.
- Some facts involve three objects:
“Chelsea is the offspring of Bill and Hillary.”

A Binary Relation Involves Exactly Two Objects

- “Bill and Hillary are Married”.
- “Carter was the successor of Ford”.
- “Ferraro was Mondale’s running mate”.
- “California shares a border with Nevada”.
- “Austin is the capital of Texas”.
- “Pataki is the governor of New York”.
- “The Knicks are better than the Celtics”.

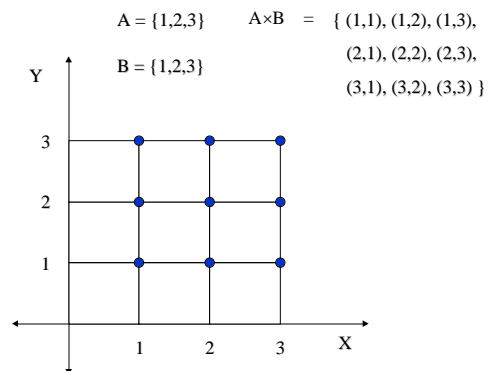
The Legacy of Rene Descartes

“Cogito Ergo Sum” → 300 Years of Philosophical Mumbo Jumbo

Cartesian Product → Analytic Geometry, Calculus, Physics, Industrial Revolution, (Etc.)

Cartesian Product

- Given two sets A and B, form the set $A \times B$ of all ordered pairs (x,y) such that:
 - Object x is a member of set A.
 - Object y is a member of set B.
- The set $A \times B$ is called the “Cartesian Product” of A and B.
- Rene Descartes of “Cogito Ergo Sum” fame.



$A = \{\text{Larry, Curly, Moe}\}$

$B = \{\text{Tom, Jerry}\}$

$A \times B = \{ (\text{Larry, Tom}), (\text{Larry, Jerry}),$
 $(\text{Curly, Tom}), (\text{Curly, Jerry}),$
 $(\text{Moe, Tom}), (\text{Moe, Jerry}) \}$

The two sets A and B may or may not be the same set.
They may contain numbers, symbols or anything else.

Defining a Relation

- Choose two sets A and B.
- Form the set $A \times B$ of all ordered pairs (x,y) such that:
 - Object x is a member of set A.
 - Object y is a member of set B.
- Let R be any subset of $A \times B$.
- We say that “R is a relation on $A \times B$ ”.
- If $A=B$ we may abbreviate this by saying that “R is a relation on A” or “R is a relation on B”.

Example of Defining a Relation

- Let both A and B be the set of all fifty U.S. states.
- Then $A \times B$ is a set of 2500 ordered pairs of states.
- Let R be the set of all pairs (s_1, s_2) in $A \times B$ such that s_1 and s_2 share a border:
 - R includes (Nevada, California).
 - R does not include (Florida, Wyoming).
- We could call R the “Neighboring-States” relation.

Example of Defining a Relation

- Let A be the set of all men.
- Let B be the set of all women.
- Then $A \times B$ is a set of $9 \cdot 10^{18}$ ordered pairs (M,W) such that M is a man and W is a woman.
- Let R be the set of all pairs (M,W) in $A \times B$ such that M is married to W.
 - R includes (Bill, Hillary).
 - R does not include (Monica, Newt).
- We could call R the “Married-Couples” relation.

Implementing a Relation as a Scheme Data Structure

- Define an abstract data type for representing an ordered pair (x,y) .
- Define a relation to be a set of ordered pairs.

Implementation of Ordered Pairs

```
(define make-op (lambda (x y) (list x y)))  
(define op-1st (lambda (op) (car op)))  
(define op-2nd (lambda (op) (cadr op)))
```

The *SUCCESSORS* Relation and SUCCESSOR? Predicate

```
(define *successors*
  (make-set (make-op 'washington 'adams)
            ...
            (make-op 'bush 'clinton)))

(define successor?
  (lambda (x y)
    (member? (make-op x y) *successors*)))
```

Manipulating Relations

- Domain: Given a relation, return a set.
- Range: Given a relation, return a set.
- Inverse: Given a relation, return a relation.
- Cartesian product: Given two sets, return a relation.
- Composition: Given two relations, return a third relation.

Domain and Range

```
(define domain (lambda (relation) ...?...))
(define range (lambda (relation) ...?...))
```

Welcome to Dr. Scheme.

```
> (define one-more
  (make-set (make-op 1 2) (make-op 2 3) (make-op 3 4)))
> (domain one-more)
("set" 1 2 3)
> (range one-more)
("set" 2 3 4)
```

Domain and Range

```
(define domain (lambda (relation) (set-map op-1st relation)))
(define range (lambda (relation) (set-map op-2nd relation)))
```

```
(define *married-couples*
  (make-set (make-op 'adam 'eve) ...
            (make-op 'threapleton 'winslet)))
```

```
(define *married-men* (domain *married-couples*))
(define *married-women* (range *married-couples*))
```

Inverse

```
(define inverse (lambda (relation) ...?...))
```

Welcome to Dr. Scheme.

```
> (define one-more
  (make-set (make-op 1 2) (make-op 2 3) (make-op 3 4)))
> one-more
("set" (1 2) (2 3) (3 4))
> (define one-less (inverse one-more))
> one-less
("set" (2 1) (3 2) (4 3))
```

Inverse

```
(define inverse
  (lambda (relation)
    (set-map (lambda (op) (make-op (op-2nd op) (op-1st op)))
             relation)))
```

```
(define *successors*
  (make-set (make-op 'washington 'jefferson)
            ... (make-op 'bush 'clinton)))
```

```
(define *predecessors* (inverse *successors*))
```

Looking up a Value in a Relation

```
(define image (lambda (item relation) ...?...))
```

Welcome to Dr. Scheme.

```
> (define *neighboring-states*  
  (make-set ...  
    (make-op 'NY 'NJ) (make-op 'NJ 'NY)  
    (make-op 'NY 'CT) (make-op 'CT 'NY)  
    (make-op 'NY 'MA) (make-op 'MA 'NY)  
    (make-op 'NY 'VT) (make-op 'VT 'NY)  
    (make-op 'NY 'PA) (make-op 'PA 'NY) ... ))  
> (image 'NY *neighboring-states*)  
("set" NJ CT MA VT PA)      IMAGE returns the set of  
                             states neighboring NY.
```

Looking up a Value in a Relation

```
(define value (lambda (item relation) ...?...))
```

Welcome to Dr. Scheme.

```
> (define *neighboring-states*  
  (make-set ...  
    (make-op 'NY 'NJ) (make-op 'NJ 'NY)  
    (make-op 'NY 'CT) (make-op 'CT 'NY)  
    (make-op 'NY 'MA) (make-op 'MA 'NY)  
    (make-op 'NY 'VT) (make-op 'VT 'NY)  
    (make-op 'NY 'PA) (make-op 'PA 'NY) ... ))  
> (value 'NY *neighboring-states*)  
CT      VALUE randomly selects one  
> (value 'NY *neighboring-states*) among the states neighboring NY.  
PA
```

Looking up a Value in a Relation

```
(define value  
  (lambda (x)  
    (lambda (relation)  
      (pick (image x relation))))))  
  
(define image  
  (lambda (x relation)  
    (set-map op-2nd  
      (set-comprehension relation  
        (lambda (op)  
          (equal? (op-1st op) x)))))))
```

Looking up a Value in a Relation

```
(define inverse-value  
  (lambda (x)  
    (lambda (relation)  
      (pick (inverse-image x relation))))))  
  
(define inverse-image  
  (lambda (x relation)  
    (set-map op-1st  
      (set-comprehension relation  
        (lambda (op)  
          (equal? (op-2nd op) x)))))))
```

Cartesian Product

```
(define cartesian-product (lambda (s1 s2) ...?...))
```

Welcome to Dr. Scheme.

```
> (define set1 (make-set 1 2 3))  
> (define set2 (make-set 4 5 6))  
> (cartesian-product set1 set2)  
("set" (1 4) (1 5) (1 6) (2 4) (2 5) (2 6) (3 4) (3 5) (3 6))
```

CARTESIAN-PRODUCT

```
(define cartesian-product  
  (lambda (s1 s2)  
    (if (empty-set? s1)  
        the-empty-set  
        (let ((e (pick s1)))  
          (let ((rest ((residue e) s1)))  
            (union (set-map (lambda (x) (make-op e x)) s2)  
                  (cartesian-product rest s2))))))))
```

Pairs Whose Product is N

```
(define pairs-with-product (lambda (n) ...?...))
```

Welcome to Dr. Scheme.

```
> (pairs-with-product 7)
```

```
("set" (1 7) (7 1))
```

```
> (pairs-with-product 6)
```

```
("set" (1 6) (2 3) (3 2) (6 1))
```

```
> (pairs-with-product 12)
```

```
("set" (1 12) (2 6) (3 4) (4 3) (6 2) (12 1))
```

Pairs with Product

```
(define pairs-with-product
```

```
(lambda (n)
```

```
(let ((numbers (apply make-set (from-to 1 n))))
```

```
(set-comprehension (cartesian-product numbers numbers)
```

```
(lambda (op)
```

```
(= n (* (op-1st op) (op-2nd op))))))
```

Compose Relations

```
(define compose-relations (lambda (r1 r2) ...?...))
```

Welcome to Dr. Scheme.

```
> (define one-more
```

```
(make-set (make-op 1 2) (make-op 2 3) (make-op 3 4)))
```

```
> (define one-less
```

```
(make-set (make-op 2 1) (make-op 3 2) (make-op 4 3)))
```

```
> (compose-relations one-more one-less)
```

```
("set" (1 3) (2 4))
```

```
> (compose-relations one-more one-less)
```

```
("set" (1 1) (2 2) (3 3))
```

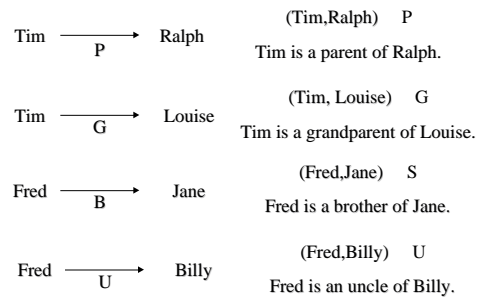
Composition of Relations

- Let Q and R be relations on set S.
- The composition of Q and R is the set of all ordered pairs (x,z) such that for some y ∈ S, (x,y) ∈ Q and (y,z) ∈ R.
- The composition of Q and R is sometimes written in the following way: Q•R.
- Notice that Q•R is a relation on the set S.

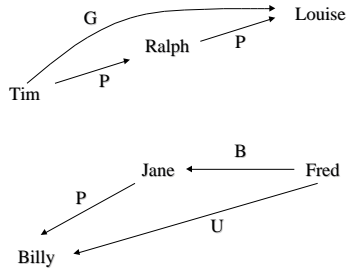
Examples of Relation Composition

- Let P be the *parent* relation, i.e., be the set of all pairs (x,y) of people such that x is a parent of y.
- The composition of P and P is the *grandparent* relation.
- Let B be the *brother* relation, i.e., set of all pairs (y,z) such that y is a brother of z.
- The composition of B and P is the *uncle* relation.

Representing Relations with Diagrams



Examples of Relation Composition



COMPOSE-RELATIONS

```
(define compose-relations
  (lambda (q r)
    (set-map (lambda (op)
              (make-op (op-1st (op-1st op))
                       (op-2nd (op-2nd op))))
             (set-comprehension (cartesian-product q r)
                                (lambda (op)
                                  (equal? (op-2nd (op-1st op))
                                           (op-1st (op-2nd op))))))))
```