CS109A Notes for Lecture 3/6/95

Cartesian Product

$A \times B$ = set of pairs of elements (a, b) such that $a \in A$ and $b \in B$.

Example: S = set of my shirts = {white, blue, green}; P = set of my pants = {blue, brown}.

- $S \times P$ = set of ensembles = {(white, blue), (white, brown), (blue, blue), (blue, brown), (green, blue), (green, brown)}.

Multiway Products

Two approaches:

1. Nest binary products, e.g., $A \times (B \times C)$.
   - □ Produces nested pairs, e.g., $(a, (b, c))$.
2. Products of more than two, e.g. $A \times B \times C$.
   - □ Produces k-tuples, e.g., (a, b, c).

- Compare with tuple types in ML, e.g., `int*int*int` vs. `int*(int*int)`.
- Natural equivalence between values like (a, b, c) and $(a, (b, c))$.

Relations

A (k-ary) *relation* is a set of k-tuples for some k.

- *Binary* relations, the important case $k = 2$.
- Common notation (infix) for binary relations: aRb means $(a, b) \in R$.

Why Relations?

- Model of sets of records — vital for holding information of all types.
  - □ e.g., course grades as sets of triples (StudentID, Course, Grade).
- Model of many operators, e.g., $<$, $\subseteq$.

Domain and Range

Binary relation A is a subset of $D \times R$ for some subsets D (the *domain*) and R (the *range*).

- Must distinguish between:
    1. *Declared domain* = set of values such that at all times the first components of A are members of this set (essentially the "type" of the first component), and
    2. *Current domain* = set of values that currently appear in the first components of pairs in A.
- Similarly: declared/current range.

Example: Let A be a relation consisting of pairs of strings and integers. Let the current value of A be {(`"foo"`, 1), (`"bar"`, 2)}.

- Declared domain = `string`, the set of all character strings.
- Declared range = `int`, the set of all integers.
- Current domain = {`"foo"`, `"bar"`}.
- Current range = $\{1, 2\}$.

Functions

If for every a in the domain of binary relation R there is at most one b such that aRb, then we say R is a (*partial*) *function*.

- Common notation: $R(a) = b$.
- Compare with "functions" in C or ML.
    - ☐ Those functions pair arguments with results, and this set of pairs is a function in the set-theoretic sense.
    - ☐ But a set-theoretic function can be a set of arbitrary pairs, with the range value not computable from the domain value.

Example: Domain, range = integers. aRb if and only if $b = a^2$.

- Can say: $3R9$, $R(-6) = 36$, $(2, 4) \in R$.

Why Functions?

Important difference in representation when a relation is a function.

Example: Store relation (StudentID, Phone).

- If we store only one phone/student, a 10-byte array suffices for the `phone` field.
- If we wish to store any number of phones per student, `phone` must be a linked list or similar, requiring extra space and extra work to store/retrieve a single phone.

Special Kinds of Functions

- If for every a in the domain of function F there is a pair (a, b) in F for some b, then F is a *total* function.
- Let the *inverse* of a relation R be $R^{-1} = \{(b, a) \mid (a, b) \in R\}$.
- If both F and F^{-1} are total functions, then F is *one-to-one* (a *bijection*).

Implementing Functions and Binary Relations

Linked-list, BST, Characteristic-vector, and Hash-table methods exist.

- Dictionary-like operations for functions F:
  - □ $lookup(a)$ returns $F(a)$.
  - □ $insert(a, b)$ makes $F(a) = b$.
  - □ $delete(a)$ makes $F(a)$ undefined.
- Dictionary-like operations for relations R:
  - □ $lookup(a)$ returns $\{b \mid aRb\}$.
  - □ $insert(a, b)$ adds (a, b) to R.
  - □ $delete(a, b)$ removes (a, b) from R.

Linked List Implementations

- For a function, use cells with fields for domain and range elements.

- ☐ i.e., type of list is *(dtype * rtype) list.*

- For a relation, use cells with a field for the domain and a field that is the header for a list of associated range elements.
  - ☐ i.e., type is *(dtype * (rtype list)) list.*

BST Implementation

- For a function F, use domain element as a *key*. $(a, b) < (c, d)$ iff $a < c$.
  - ☐ Store both a and $F(a)$ at the node for a.
- For a relation R, also use domain element as a key. However, stored at a node for key (domain element) a is a list of all the b's such that aRb.

Characteristic Vector Implementation

Suitable only if the domain is a "small" set that can serve as index of arrays.

- For a function F, store in $F[a]$ the value $F(a)$.
  - ☐ If F is not total, we need an "undefined" value outside the range that may appear in $F[a]$.
- For a relation R, store in $R[a]$ the header of a list of b's such that aRb.

Hash Table Implementation

We use only the domain element as a key (value to be hashed).

- Buckets are lists of related pairs (a, b).
- For both functions and relations, store (a, b) in the bucket $h(a)$.
- Perform *lookup*(a) by searching the bucket $h(a)$.
- Only difference between functions and relations: a relation of size n may not distribute nicely among n buckets, because the number of domain elements may be much less than n.