Massachusetts Institute of Technology — Handout 22
6.042J/18.062J: Mathematics for Computer Science — September 29, 1998
Professors Nancy Lynch and Srinivas Devadas

Supplement to Lecture 4

Readings: Rosen, Chapter 6, esp. 6.1, 6.6; Chapter 7, esp. 7.1-7.4, 7.8; Section 8.1; Last fall's lecture notes on state machines.

Outline of today's lecture:

1. Relations
   (a) Definitions
   (b) Properties
   (c) Representation
   (d) Paths
   (e) Operations
2. Types of Relations
   (a) DAGs
   (b) Posets
   (c) Equivalence relations (not covered in lecture)
   (d) Graphs
   (e) (next time) Trees

1 Relations

Relations are important structures that arise in computer science in many contexts. I'll give a quick tour today and the first half of Thursday, presenting definitions and examples and a few typical results.

1.1 Definitions

A relation is a basic mathematical notion expressing a relationship between elements of sets. It can be used to model many different sorts of relationships, e.g., ordering or equivalence relationships. Relations form the basis of the relational database model, a standard data model used in commercial data processing systems.

Defintion. A *binary relation from A to B* is any subset $R \subseteq A \times B$. A *binary relation on A* is any subset $R \subseteq A \times A$. A relation R between elements a and b holds if $(a, b) \in R$. Ternary relations and n-ary relations are defined similarly, with 3 or n components respectively.

Notation. The fact that a relation R holds between a and b may be indicated by $(a, b) \in R$, by aRb, or by $a \sim_R b$.

Examples.

1. $A =$ MIT students, $B =$ MIT courses, $R =$ "is taking"
2. $A =$ MIT students, $R =$ "lives in the same dorm as"
3. $A =$ cities in the world, $R =$ "can drive between"
4. $A =$ computers, $R =$ "connected directly by a wire"
5. $A =$ people, $R =$ "likes"
6. $A =$ N, aRb means $a \leq b$
7. $A = \mathcal{P}(n)$, aRb means $a \subseteq b$
8. $A = \mathcal{P}(n)$, aRb means $a \cap b$ is a finite set

1.2 Properties of Binary Relations

A binary relation R on A is:

- *Reflexive* if for every $a \in A$, aRa.
- *Irreflexive* if for every $a \in A$, $a \not{R} a$.
- *Symmetric* if for every $a, b, \in A$, $aRb \rightarrow bRa$.
- *Antisymmetric* if for every $a, b \in A$, aRb and bRa together imply $a = b$.
- *Transitive* if for every $a, b, c \in A$, aRb and bRc together imply aRc.
- *Total* if for every $a, b \in A$, either aRb or bRa.

Examples. The properties that hold for the example relations are listed below.

1. ("is taking" is not a relation on a single set A.)
2. "lives in the same dorm as" is reflexive, symmetric, transitive.

3. "can drive between" is reflexive, symmetric, and transitive, provided that all roads are 2-way.

4. "connected directly by a wire" is symmetric, but not transitive; whether it's reflexive is open to interpretation.

5. "likes" is unfortunately not symmetric: it might be that Cal likes Rose but Rose does not like Cal. Not antisymmetric either: it might be that Rose likes Jack and Jack likes Rose. Not transitive: we can have that Cal likes Rose and Rose likes Jack but Cal doesn't like Jack. Not even reflexive. Not irreflexive.

6. $\leq$ on the natural numbers is reflexive, antisymmetric, transitive, total.

7. $\subseteq$ on subsets of the natural numbers is reflexive, antisymmetric, and transitive.

8. "have finite instersection" on subsets of the natural numbers is not reflexive, not irreflexive, but is symmetric. Not transitive: $\text{evens} \cap \text{odds} = \emptyset$, which is finite. $\text{odds} \cap \text{evens} = \emptyset$, which is finite. But $\text{evens} \cap \text{evens} = \text{evens}$, which is an infinite set.

1.3 Representing relations

There are different ways of representing relations, e.g., for handling in a computer program. We can describe them using properties, as above. That's about all we can do, for infinite sets. But for finite sets, we can use various methods that explicitly represent all the elements:

1.3.1 Lists

We could just list all the pairs in the relation. For example, the divisibility relation on the numbers 1 to 10 could be written:

$$(1,1),(1,2),(1,3),\ldots,(1,10),(2,2),(2,4),(2,6),\ldots,(2,10),(3,3),\ldots$$

We could also write *adjacency lists*:

$$\begin{array}{rcl} 1 & : & 1,2,3,\ldots,10 \\ 2 & : & 2,4,6,8,10 \\ 3 & : & 3,6,9 \\ 4 & : & 4,8 \\ 5 & : & 5 \\ & \cdots & \\ 10 & : & 10 \end{array}$$

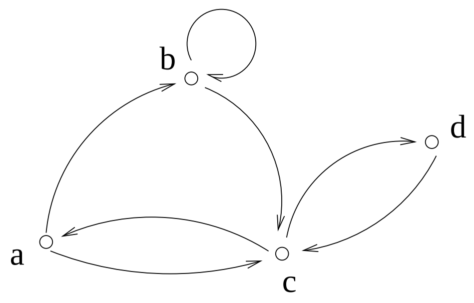
This kind of representation can be used even for relations from A to B, where $A \neq B$.

Example. In the relational database model, data is organized into *relations*, each of which is is (at least conceptually) a list of pairs (more generally, of n-tuples).

1.3.2 Boolean adjacency matrices

For A, B, finite, let the rows be indexed by A and columns by B. Put a 1 in position (a, b) if $(a, b) \in R$, and a 0 otherwise.

Example. Cities directly connected by flights. Includes a tourist sightseeing flight from b to b.



Matrix representation:

$$\begin{array}{ccccc} & a & b & c & d \\ a & 0 & 1 & 1 & 0 \\ b & 0 & 1 & 1 & 0 \\ c & 1 & 0 & 0 & 1 \\ d & 0 & 0 & 1 & 0 \end{array}$$

Again, this representation can be used even for relations from A to B, where $A \neq B$. Often in practice, such matrices are sparse (have lots of 0s). Compression methods are used to get more compact representations.

1.3.3 Network diagrams

Vertices (dots) and edges (arrows), as in the previous diagram. This is good for blackboard and paper, but not for internal computer representation. (It is good for user visualization, though.) This is used in the case where $A = B$.

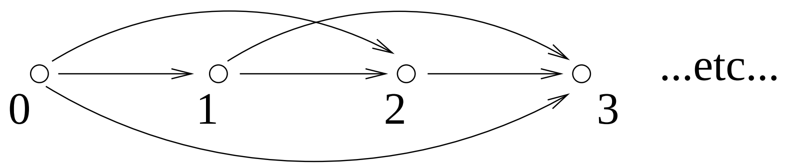
Can have self-loops like vertex b in the preceding figure. Sometimes the arrowhead is omitted.

Sometimes two arrows running in opposite directions between the same vertices (like c and d above) are represented with a single, double headed arrow or with just a plain line with no arrowheads.

Definition. A *digraph*, or *directed graph*, is the same as a general binary relation.

However, when we call it a digraph, we're generally thinking about the network diagram representation. In this case, we usually use notation $G = (V, E)$ for a digraph G with vertex set V and edge set E (instead of relation notation A, R).

A digraph can be infinite. For example, we can depict $(N, <)$ as follows:



In most work on digraphs, they are assumed to be finite (have finitely many vertices and edges). I'll try to say "finite" explicitly when I need to assume it.

1.4 Paths in Binary Relations (or Digraphs)

Suppose that we have a relation $R \subseteq A^2$ (or V^2, in digraph notation). A *path* in R is a sequence of one or more pairs in R, $(a_0, a_1), (a_1, a, 2), (a_2, a_3), \ldots$, where successive pairs "match up" – the second element of each pair is the same as the first element of the next pair. A path can be finite or infinite. It can revisit elements, or repeat the same pair. We often write such a path in terms of its vertices, as in: $a_0, a_1, a_2, a_3, \ldots$.

Example. Some paths in the flights digraph shown previously are: a, b, c, d, c (finite) and $a, b, b, b, b, \ldots$ (infinite).

Example. In the divisibility digraph shown previously, $1, 2, 6, 30, 120$ is a path.

A *circuit* is a finite path that ends up back where it started. For example, in the flights dgraph, a, b, b, c, a is a circuit.

A *cycle* is a circuit that doesn't revisit any vertices, except that the last vertex is the same as the first. For example, a, b, c, a is a cycle in the flights digraph. As another example, we discussed the presence of cycles in tournaments (which are digraphs) last week.

A *simple path* is either a cycle or else a path with no repeated vertices. For example, a, b, c, a and a, c, d are simple paths in the flights digraph.

1.5 Operations on Relations

There are several useful ways of combining and transforming relations. For example, relational databases have a collection of such operations, including restriction, join, projection, etc. Here I'll just describe one fundamental operation: composition of binary relations:

If $R_1 \subseteq A \times B$ and $R_2 \subseteq B \times C$ then $R_1 \cdot R_2$, the relational composition of R_1 and R_2, is a relation from A to C defined as follows:

$$(a, c) \in R_1 \cdot R_2 \quad \text{if and only if} \quad \exists b \in B : (a, b) \in R_1 \text{ and } (b, c) \in R_2$$

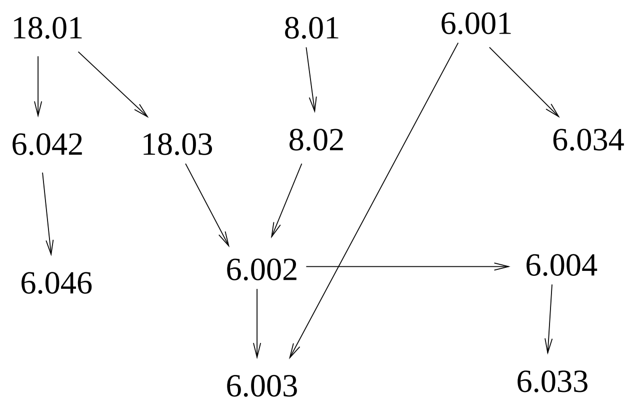
Examples. Composition of parent-of and parent-of gives grandparent-of. Composition of child-of with parent-of gives "sibling or self". Composition of parent-of with child-of gives "parents of the same child". (As a special case, a parent of any child is related to himself or herself.)

Nice fact: If relations are represented by Boolean matrices, then the relational composition corresponds exactly to Boolean matrix multiplcation, where $+$ corresponds to $\vee$ and $\times$ corresponds to $\wedge$.

2 Special Types of Relations (Digraphs)

2.1 Directed Acyclic Graphs (DAGs)

These are a special case of digraphs having *no cycles.* This means no self-loops (a, a), and no longer cycles $a_0, a_1, \ldots a_k = a_0$, where $k \geq 2$. Directed acyclic graphs are often callsed DAGs. As an example, course requirement can be described with a DAG.



Theorem 2.1 *Suppose that the longest path in a finite DAG R has k vertices. Then the vertices can be partitioned into k sets $V_1, V_2, \ldots, V_k$ such that if $(a, b) \in R$, then the index of the set containing a is smaller than the index of the set containing b.*

This implies, for example, that there exists an assignment of courses to terms such that all courses can be completed in k terms. The set of courses taken in the k-th term is the set V_k. A proof of this theorem— which is slightly more complicated than the one shown in lecture— is included in the Fall 97 lecture notes.

2.2 Partially Ordered Sets (Posets)

A poset is a kind of ordering relationship, a $\leq$ order. These are a special case of digraphs (relations) with the reflexive, antisymmetric, transitive properties.

Example. $\leq$ for N, $\subseteq$ for $\mathcal{P}(N)$, divisibility for N^+.

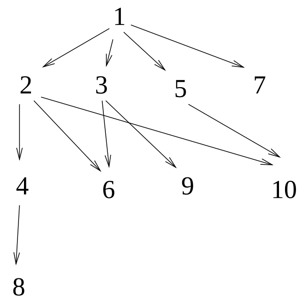
A poset is not the same as a DAG, e.g., because a poset has (a, a) cycles.

Theorem 2.2 *A poset has no cycles of length (number of edges) greater than 1.*

A proof is given in the Fall 97 lecture notes.

A finite poset is often represented by a *Haase diagram*, which looks like a DAG:

Example. The following Haase diagram illustrates the divisibility relation on the numbers 1 to 10.



Note that the Haase diagram representation isn't exactly the same as the usual network diagram representation of division. Some edges are missing: the self-edges $(1,1),(2,2),\ldots$, and the "transitive edges" $(1,4),(1,6),(1,9),(1,10),(1,8),(2,8)$.

The reason they aren't there is that Haase diagrams are used only for posets, which always satisfy reflexivity and transitivity. So we can leave these edges implicit, and deduce that they must be there, even though they aren't represented.

Special case of a poset: Totally ordered set. That's just a poset that's *total*, i.e., it satisfies reflexivity, antisymmetry, transitivity and totality (which says that every two elements are related one way or the other: $\forall a,b, aRb \vee bRa$.

Example. The $\leq$ relation on the natural numbers defines a totally ordered set. However, the divisibility relation on the natural numbers does not define a totally ordered set.

Theorem 2.3 *Any finite poset P has a totally ordered set T that can be obtained from P just by adding more edges (pairs) to P.*

Example. For divisibility, can use 1,2,3,4,5,6,7,8,9,10, or 1,7,5,3,9,2,10,6,4,8, or other possibilities. The point here is that in each sequence, no number divides a number to the left.

2.3 Equivalence Relations

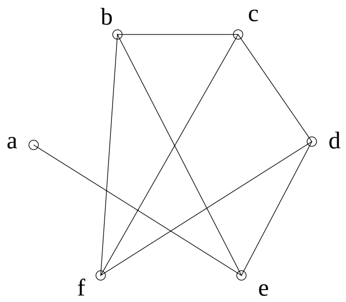
An equivalence relation is a relation that is reflexive, symmetric, and transitive.

Examples. "lives in the same dorm", "can drive between" .

This was not covered in lecture, but see the definitions in Rosen, Section 6.5

2.4 Graphs

These are relations (digraphs) that are irreflexive and symmetric. All the finite ones can be drawn with diagrams containing vertices and undirected (bidirectional) edges. There are no self-loops. An example is shown below.



Again, this kind of drawing is so standard that people identify graphs with these pictures. In fact the notation $G = (V, E)$ is usually used.

Graph theory is a big subfield of mathematics, which is devoted to just studying properties of (finite) graphs. Many different kinds of properties are studied.

The rest of the material in this lecture was on vertex degrees and graph coloring and was covered as in fall 97.