Notes 6

1 Relations

Last time we talked about

- relations
- properties: they can be reflexive, symmetric, antisymmetric, transitive
- 3 ways of representing them: lists of tuples, matrices, and graphs

1.1 Operations on Relations

Notice the difference between properties of relations and operations on relations. A property, such as symmetry or transitivity, is something a relation may or may not satisfy. An operation on relations, such as relation composition we are going to define, is a way to combine one or more relations to build some other relation.

1.1.1 Set Operations

Relations are sets of tuples, so we can apply set operations. The intersection of two relations gives pairs satisfying both relations. The union gives pairs satisfying one—the union of "brother-of" and "sister-of" is "sibling-of". Complement gives pairs not satisfying.

1.1.2 Inverse

Definition 1.1 If R is a relation on $A \times B$, then R^{-1} is a relation on $B \times A$ given by $R^{-1} = \{(b, a) \mid (a, b) \in R\}$.

It's just the relation "turned backwards."

Example 1.2 The inverse of "parent-of" is "child-of."

1.1.3 Composition

Definition 1.3 The *composition* of relations $R_1 \subseteq A \times B$ and $R_2 \subseteq B \times C$ is the relation $R_2 \circ R_1 = \{(a,c) \mid (\exists b)((a,b) \in R_1) \wedge ((b,c) \in R_2)\}$.

Notice that $R_1 \circ R_2$ and $R_2 \circ R_1$ are different. We are using the notation used in the text book. Some people prefer to write $R_1 \circ R_2$ for the composition of R_1 and R_2, which may seem more natural, but from now on we will stick with the book.

Example 1.4 The composition of parent-of relation with itself gives grandparent-of. Composition of child-of relation with parent-of relation gives sibling-of relation.

Does composition of parent-of with child-of give married-to/domestic partners? No, because it misses childless couples.

Composition of relations is equivalent to matrix multiplication, with $+$ replaced by $\vee$ (Boolean or) and $*$ replaced by $\wedge$.

Example 1.5 Let R_1 be relation from the first example above:

	a	b	c
0	1	0	1
1	1	0	1
2	0	1	0
3	0	0	0

Let R_2 be relation from $\{a,b,c\}$ to $\{d,e,f\}$ given by:

	d	e	f
a	1	1	1
b	0	1	0
c	0	0	1

Then $R_2 \circ R_1 = \{(0,d), ...\}$, that is,

	d	e	f
0	1	1	1
1	1	1	1
2	0	1	0
3	0	0	0

This is the same as matrix multiplication, using the Boolean operations "and" and "or".

Relational composition is associative, just like matrix multiplication is:

Lemma 1.6 $R_1 \circ (R_2 \circ R_3) = (R_1 \circ R_2) \circ R_3$

The composition $R \circ R$ of R with itself is written R^2. Similarly R^n denotes R composed with itself n times. Formally (by recursive definition) $R_1 = R$, $R^n = R \circ R^{n-1}$.

1.1.4 Closure

A closure "extends" a relation to satisfy some property. But extends it as little as possible.

Definition 1.7 The *closure* of relation R with respect to property P is the relation S that (i) contains R, (ii) has property P, and (iii) is contained in *any* relation satisfying (i) and (ii). That is, S is the "smallest" relation satisfying (i) and (ii).

There might be no one relation satisfying the definition; in that case the closure is not defined.

Lemma 1.8 The reflexive closure of R is $S = R \cup \{(a, a) \mid a \in A\}$

Proof. It contains R and is reflexive by design. Furthermore (by definition) any relation satisfying (i) must contain R, and any satisfying (ii) must contain the pairs (a, a), so any relation satisfying both (i) and (ii) must contain S. ∎

Lemma 1.9 The symmetric closure of R is $S = R \cup R^{-1}$.

Proof. This relation is symmetric and contains R. It is also the smallest such. For suppose we have some symmetric relation T with $R \subseteq T$. Consider $(a, b) \in R$. Then $(a, b) \in T$ so by symmetry $(b, a) \in T$. It follows that $R^{-1} \subseteq T$. So $S = R \cup R^{-1} \subseteq T$. ∎

For the transitive closure, we need to introduce some new terminology.

Definition 1.10 A *walk* in a relation R is a sequence $a_0, \ldots, a_k$ with $k \geq 1$ such that $(a_i, a_{i+1}) \in R$ for every $i < k$. We call k the *length* of the walk.

In the graph model, a walk is something you can trace out by following arrows from vertex to vertex, without lifting your pen. Note that a singleton vertex is *not* a length 0 walk (this is just for convenience).

Some books call the above a *path*. But this is ambiguous because of the following definition:

Definition 1.11 A *simple path* is a walk with no repeated vertices.

Many people say path when they mean simple path, so we will use walk to allow for possibly non-simple paths.

Walks give us the terminology for transitive closure.

Lemma 1.12 The transitive closure of a relation R is that set

$$S = \{(a, b) \mid \text{there is a walk from } a \text{ to } b \text{ in } R\}.$$

Proof. First let's show S is transitive. Suppose $(a, b) \in S$ and $(c, d) \in S$. This means that there is an (a, b) walk and a (b, c) walk in R. If we "concatenate" them (attach the end of the (a, b) walk to the start of the (b, c) walk), we get an (a, c) walk. So $(a, c) \in S$. So S is transitive.

Now consider any transitive relation T containing R. We have to show it contains S. That is, we must show $(a, b) \in S$ implies $(a, b) \in T$. But $(a, b) \in S$ means there is a walk from a to b in R, so we need only show that for any a-b walk in R, we have aTb.

We do so by induction, proving that for any a-b walk of length n, aTb. The base case is clear: a length one walk is an edge of R, and $R \subseteq T$. So assume it for length n walks. Consider a length-$n+1$ walk $a_0, \ldots, a_{n+1}$ in R. Then $a_0, \ldots, a_n$ is a length-n walk in R, so $a_0 T a_n$. Also $a_n T a_{n+1}$. So by transitivity of T (assumed) we have $a_0 T a_n$. This proves the inductive step. ∎

1.2 Computing the Transitive Closure

With reflexive and symmetric closure, it is pretty clear how to actually build them. But for transitive closure, how do we actually find all the walks we need?

Let's start by finding walks of a given length. Recall the definition of R^n of R composed with itself n times.

Lemma 1.13 $R^n = \{(a, b) \mid \text{ there is a length } n \text{ walk from } a \text{ to } b \text{ in } R\}$

Proof. By induction. The base case is clear—walks of length one are edges of R. Suppose it is true for R^n; let's prove it for R^{n+1}. Suppose there is a walk $a_0, \ldots, a_{n+1}$ in R. This is a walk $a_0, \ldots, a_n$ in R followed by a tuple (a_n, a_{n+1}) of R. Thus (by induction) we have $(a_0, a_n) \in R^n$ and $(a_n, a_{n+1}) \in R$. Thus (by definition of composition) $(a_0, a_{n+1}) \in R^{n+1}$ as claimed.

Am I done? No, I only proved set containment in one direction. I still need to show that if $(a, b) \in R^n$ there is a walk of length n from a to b. But I'll let you do it: use the induction above, but turn the proof backwards. ∎

This means we can write the transitive closure of R as $R \cup R^2 \cup R^3 \cdots$. Better (since we know how to do composition), but still a problem: there are infinitely many terms!

Lemma 1.14 Suppose A has n elements. If there is a walk from A to B, there is a walk of length at most n from A to B.

Proof. We'll use well ordering (for a change). Consider the shortest walk $a = a_0, a_1, \ldots, a_k = b$ from a to b (we know one exists, so by well ordering there is a shortest one). Suppose $k > n$. Then some element of A appears twice in the list (with more than n list entries, one must be a repeat). This means the walk is at some point circling back to where it was before. That is, $a_i = a_j$ for some $i < j$. We can cut out this cycle from the walk and get a shorter walk. That is, $a_0, \ldots, a_i, a_{j+1}, \ldots, a_k$ is a shorter walk. This contradicts that we had a shortest walk. So we cannot have $k > n$. ∎

So we don't need infinitely many terms. It is enough to take walks of length up to n, namely $R^1 \cup R^2 \cdots \cup R^n$.

Wait a minute. Well ordering is applied to sets of *numbers*; we applied it to a set of walks. How? Well, look at the set of "lengths of walks. It is nonempty, so has a smallest element. There is walk that has this length—so it is a shortest walk.

2 Equivalence relations and partitions

Now we consider an important special type of relation, called an *equivalence relation.* We give an important special case – modular arithmetic. We show an application to RSA.

2.1 Definitions

Definition 2.1 An *equivalence relation* is a relation that is reflexive, symmetric and transitive.

For example, the "roommates" relation is an equivalence relation. So is "married to", "same size as", and "on same Ethernet hub". A trivial example is "=" relation on natural numbers. The hallmark of equivalence relations is the word "same". It provides a way to "hide" unimportant differences.

Equivalence relations partition the universe into subsets in a natural way:

Definition 2.2 A *partition* of a set A is a collection of subsets $\{A_1, \cdots, A_k\}$ such that any two of them are disjoint (for any $i \neq j$, $A_i \cap A_j = \emptyset$) and such that their union is A.

That is, every element of A is in exactly one subset.

Fix an equivalence relation R on A. For an element $a \in A$, let $[a]$ denote the set $\{b \in A \mid a \sim_R b\}$. We call this set the *equivalence class of a under R*. We call a a *representative* of $[a]$

Lemma 2.3 The sets $[a]$ for $a \in A$ constitute a partition of A. That is,

- $\cup_{a \in A}[a] = A$ (their union is A)
- for every $a, b \in A$, either $[a] = [b]$ or $[a] \cap [b] = \emptyset$ (distinct sets are disjoint).

Proof. To prove the first part, note that by reflexivity $a \in [a]$. Therefore $A = \cup_{a \in A} \{a\} \subseteq \cup_{a \in A}[a]$.

Note let's prove the second part. Fix $a, b \in A$. If $[a] = [b]$ then we are done, so suppose not. Let c be any element that is in one of the sets but not the other. Without loss of generality we can assume that $c \in [b] - [a]$ (we know that either $c \in [b] - [a]$ or $c \in [a] - [b]$. In the latter case we can simply swap a and b and reduce to the first case.). That is, $c \in [b]$ but $c \notin [a]$.

At the same time if $[a] \cap [b] = \emptyset$ then we are done, so suppose not. So let d be any element in $d \in [a] \cap [b]$.

We will get a contradiction by showing that $a \sim_R c$ and therefore that $c \in [a]$.

First, $a \sim_R d$ because $d \in [a]$ by choice of d. Second, $d \sim_R b$ and $b \sim_R c$ because both $c, d \in [b]$ and R is symmetric. This implies, by transitivity, that $d \sim_R c$. Finally, by transitivity, $a \sim_R c$ because $a \sim_R d$ and $d \sim_R c$. ∎

Conversely, any partition $\{A_1, \cdots, A_k\}$ of A defines an equivalence relation by letting $a \sim_R b$ iff a and b are in the same A_i. (Note the appearance of "same".) I claim this is an equivalence relation:

Reflexivity: for all a we know $a \in A_i$ for some i, by definition of partition. Clearly a and a are in the same A_i, and $a \sim_R b$.

Symmetry: Assume $a \sim_R b$, that is a and b are in the same A_i. Also b and a are in the same A_i and therefore $b \sim_R a$.

Transitivity: Assume $a \sim_R b$ and $b \sim_R c$. By definition of $\sim_R$, a and b are in the same A_i for some i, and b and c are in the same A_j for some j. But by definition of partition b cannot be in two different A_i's. So, it must be $A_i = A_j$ and a and c are in the same A_i, proving $a \sim_R c$.

Therefore, we can look at partitions and at equivalence relations as the same thing.

2.2 Integers modulo m

We now discuss a particular equivalence relation on integers (positive, negative and 0):

Definition 2.4 If a and b are integers, then we say $a = b \pmod m$ if $m \mid (a - b)$.

An equivalent formulation says that $a = b + km$ for some integer k.

Theorem 2.5 Equality modulo m is an equivalence relation.

Proof. We need to show the relation is reflexive, symmetric, and transitive.

Reflexive: Clearly $m \mid a - a = 0$, so $a = a \pmod m$.

Symmetric: If $a = b + km$ then $b = a + (-k)m$.

Transitive: Suppose $a = b \pmod m$ and $b = c \pmod m$. Then $m \mid (a - b)$ and $m \mid (b - c)$. So $(a - b) = k_1 m$ and $(b - c) = k_2 m$. So $(a - c) = (a - b) + (b - c) = (k_1 + k_2)m$. Therefore $m \mid= a - c$.

∎

The equivalence class of a is simply the set $[a] = \{b \in \mathbb{Z} \mid a = b \pmod m\}$, or $\{km + a \mid k \in \mathbb{Z}\}$.

It turns out that we can extend a lot of standard arithmetic to work modulo m. In fact, we can define notions of sum and product for the equivalence classes mod m. For example, we define $[a] + [b]$, given two equivalence classes mod m, to be the equivalence class $[a + b]$. This is not as obvious as it seems: notice that the result is given in terms of a and b, two selected representative from the equivalence classes, but that it is supposed to apply to the equivalence classes themselves. To prove that this works, we have to show that it doesn't matter which representatives of the equivalence classes we choose:

Lemma 2.6 If $a = x \pmod m$ and $b = y \pmod m$ then $(a + b) = (x + y) \pmod m$.

Proof. $m \mid (a - x)$ and $m \mid (b - y)$ so $m \mid ((a - x) + (b - y)) = (a + b) - (x + y)$. ∎

It follows that if we are interested only the result of the addition modulo m, which is the case, for example, in the RSA cryptosystem, that we can at any time replace a given number with a different number equivalent to it, without changing the value (equivalence class) of the final answer.

The same fact can be proven for multiplication. This is the basis of the fast exponentiation algorithm presented in the next subsection.

2.3 Fast Exponentiation

Recall that for RSA we wanted to compute the remainder of x^a. on division by b—that is, the equivalence class $\bmod b$ of x^a.

In the section on recursive algorithms, we already saw how to reduce the number of multiplications substantially. But there is still a problem: x^a is a really big number. It is bigger than 2^a, so has $2^{2^{128}}$ bits if we use a 128-bit key a. It will take a long time to write this number down (if we could find a place to fit it).

But notice that we only care about the remainder modulo b. So using modular arithmetic we can solve the large-number problem. Whenever the number we are computing gets too big, replace it with a smaller representative of the same equivalence class. What is the smallest representative? The remainder on division by b. We've argued that switching representatives doesn't change the answer. But now all our math is done on number smaller than b—i.e. 128-bit numbers.