9.5 Directed Acyclic Graphs & Scheduling

Some of the prerequisites of MIT computer science subjects are shown in Figure 9.6. An edge going from subject s to subject t indicates that s is listed in the catalogue as a direct prerequisite of t. Of course, before you can take subject t, you have to take not only subject s, but also all the prerequisites of s, and any prerequisites of those prerequisites, and so on. We can state this precisely in terms of the positive walk relation: if D is the direct prerequisite relation on subjects, then subject u has to be completed before taking subject v iff $u \ D^+ \ v$.

Of course it would take forever to graduate if this direct prerequisite graph had a positive length closed walk. We need to forbid such closed walks, which by Lemma 9.2.6 is the same as forbidding cycles. So, the direct prerequisite graph among subjects had better be *acyclic*:

Definition 9.5.1. A *directed acyclic graph (DAG)* is a directed graph with no cycles.

DAGs have particular importance in computer science. They capture key concepts used in analyzing task scheduling and concurrency control. When distributing a program across multiple processors, we're in trouble if one part of the program needs an output that another part hasn't generated yet! So let's examine DAGs and their connection to scheduling in more depth.

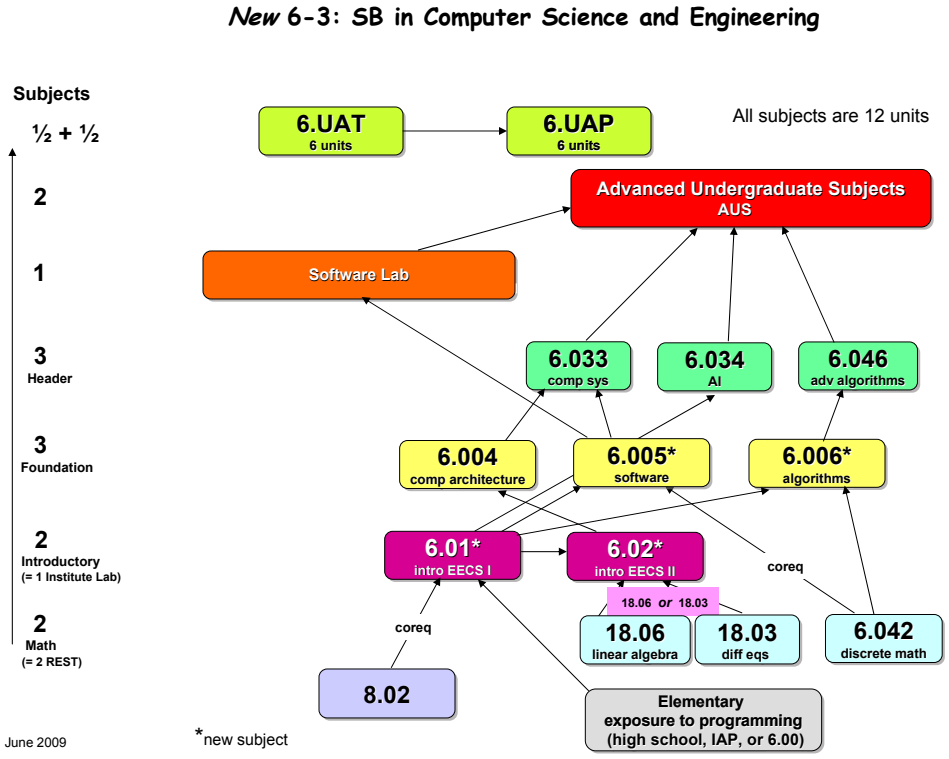

Figure 9.6 Subject prerequisites for MIT Computer Science (6-3) Majors.

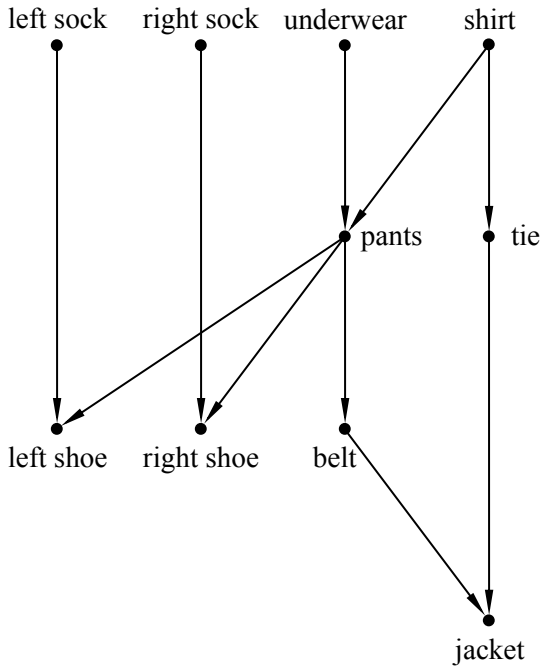

Figure 9.7 DAG describing which clothing items have to be put on before others.

9.5.1 Scheduling

In a scheduling problem, there is a set of tasks, along with a set of constraints specifying that starting certain tasks depends on other tasks being completed beforehand. We can map these sets to a digraph, with the tasks as the nodes and the direct prerequisite constraints as the edges.

For example, the DAG in Figure 9.7 describes how a man might get dressed for a formal occasion. As we describe above, vertices correspond to garments and the edges specify which garments have to be put on before which others.

When faced with a set of prerequisites like this one, the most basic task is finding an order in which to perform all the tasks, one at a time, while respecting the dependency constraints. Ordering tasks in this way is known as *topological sorting*.

Definition 9.5.2. A *topological sort* of a finite DAG is a list of all the vertices such that each vertex v appears earlier in the list than every other vertex reachable from v.

There are many ways to get dressed one item at a time while obeying the constraints of Figure 9.7. We have listed two such topological sorts in Figure 9.8. In

underwear	left sock
shirt	shirt
pants	tie
belt	underwear
tie	right sock
jacket	pants
left sock	right shoe
right sock	belt
left shoe	jacket
right shoe	left shoe
(a)	(b)

Figure 9.8 Two possible topological sorts of the prerequisites described in Figure 9.7

.

fact, we can prove that *every* finite DAG has a topological sort. You can think of this as a mathematical proof that you can indeed get dressed in the morning.

Topological sorts for finite DAGs are easy to construct by starting from *minimal* elements:

Definition 9.5.3. An vertex v of a DAG, D, is *minimum* iff every other vertex is reachable from v.

A vertex v is *minimal* iff v is not reachable from any other vertex.

It can seem peculiar to use the words "minimum" and "minimal" to talk about vertices that start paths. These words come from the perspective that a vertex is "smaller" than any other vertex it connects to. We'll explore this way of thinking about DAGs in the next section, but for now we'll use these terms because they are conventional.

One peculiarity of this terminology is that a DAG may have no minimum element but lots of minimal elements. In particular, the clothing example has four minimal elements: leftsock, rightsock, underwear, and shirt.

To build an order for getting dressed, we pick one of these minimal elements—say, shirt. Now there is a new set of minimal elements; the three elements we didn't chose as step 1 are still minimal, and once we have removed shirt, tie becomes minimal as well. We pick another minimal element, continuing in this way until all elements have been picked. The sequence of elements in the order they were picked will be a topological sort. This is how the topological sorts above were constructed.

So our construction shows:

Theorem 9.5.4. *Every finite DAG has a topological sort.*

There are many other ways of constructing topological sorts. For example, instead of starting from the minimal elements at the beginning of paths, we could build a topological sort starting from *maximal* elements at the end of paths. In fact, we could build a topological sort by picking vertices arbitrarily from a finite DAG and simply inserting them into the list wherever they will fit.[5]

9.5.2 Parallel Task Scheduling

For task dependencies, topological sorting provides a way to execute tasks one after another while respecting those dependencies. But what if we have the ability to execute more than one task at the same time? For example, say tasks are programs, the DAG indicates data dependence, and we have a parallel machine with lots of processors instead of a sequential machine with only one. How should we schedule the tasks? Our goal should be to minimize the total *time* to complete all the tasks. For simplicity, let's say all the tasks take the same amount of time and all the processors are identical.

So given a finite set of tasks, how long does it take to do them all in an optimal parallel schedule? We can use walk relations on acyclic graphs to analyze this problem.

In the first unit of time, we should do all minimal items, so we would put on our left sock, our right sock, our underwear, and our shirt.[6] In the second unit of time, we should put on our pants and our tie. Note that we cannot put on our left or right shoe yet, since we have not yet put on our pants. In the third unit of time, we should put on our left shoe, our right shoe, and our belt. Finally, in the last unit of time, we can put on our jacket. This schedule is illustrated in Figure 9.9.

The total time to do these tasks is 4 units. We cannot do better than 4 units of time because there is a sequence of 4 tasks that must each be done before the next. We have to put on a shirt before pants, pants before a belt, and a belt before a jacket. Such a sequence of items is known as a *chain*.

Definition 9.5.5. Two vertices in a DAG are *comparable* when one of them is reachable from the other. A *chain* in a DAG is a set of vertices such that any two of them are comparable. A vertex in a chain that is reachable from all other vertices in the chain is called a *maximum element* of the chain. A finite chain is said to *end at* its maximum element.

[5] In fact, the DAG doesn't even need to be finite, but you'll be relieved to know that we have no need to go into this.

[6] Yes, we know that you can't actually put on both socks at once, but imagine you are being dressed by a bunch of robot processors and you are in a big hurry. Still not working for you? Ok, forget about the clothes and imagine they are programs with the precedence constraints shown in Figure 9.7.

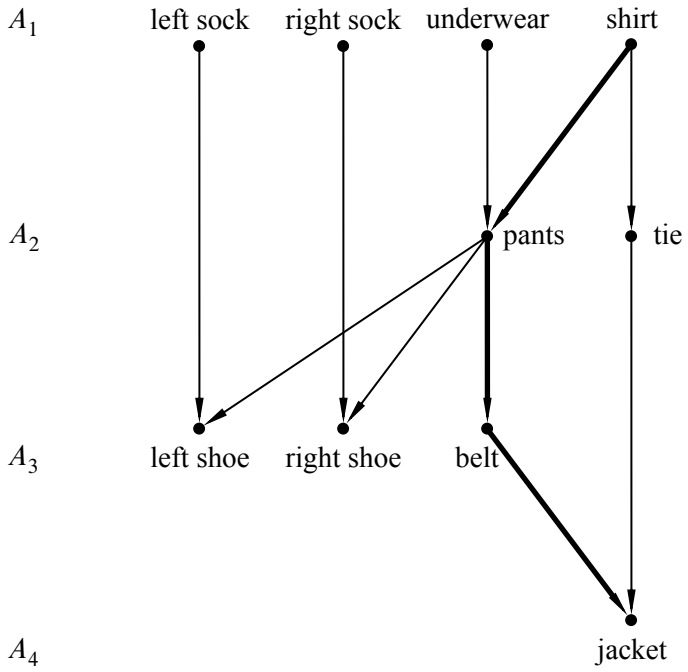

Figure 9.9 A parallel schedule for the tasks-getting-dressed digraph in Figure 9.7. The tasks in A_i can be performed in step i for $1 \leq i \leq 4$. A chain of 4 tasks (the critical path in this example) is shown with bold edges.

The time it takes to schedule tasks, even with an unlimited number of processors, is at least as large as the number of vertices in any chain. That's because if we used less time than the size of some chain, then two items from the chain would have to be done at the same step, contradicting the precedence constraints. For this reason, a *largest* chain is also known as a *critical path*. For example, Figure 9.9 shows the critical path for the getting-dressed digraph.

In this example, we were able to schedule all the tasks with t steps, where t is the size of the largest chain. A nice feature of DAGs is that this is always possible! In other words, for any DAG, there is a legal parallel schedule that runs in t total steps.

In general, a *schedule* for performing tasks specifies which tasks to do at successive steps. Every task, a, has to be scheduled at some step, and all the tasks that have to be completed before task a must be scheduled for an earlier step. Here's a rigorous definition of schedule.

Definition 9.5.6. A *partition* of a set A is a set of nonempty subsets of A called the *blocks*[7] of the partition, such that every element of A is in exactly one block.

For example, one possible partition of the set $\{a, b, c, d, e\}$ into three blocks is

$$\{a, c\} \qquad \{b, e\} \qquad \{d\}.$$

Definition 9.5.7. A *parallel schedule* for a DAG, D, is a partition of $V(D)$ into blocks $A_0, A_1, \ldots$, such that when $j < k$, no vertex in A_j is reachable from any vertex in A_k. The block A_k is called the set of elements *scheduled at step* k, and the *time* of the schedule is the number of blocks. The maximum number of elements scheduled at any step is called the *number of processors* required by the schedule.

A *largest* chain ending at an element a is called a *critical path* to a, and the number of elements less than a in the chain is called the *depth* of a. So in any possible parallel schedule, there must be at least depth (a) steps before task a can be started. In particular, the minimal elements are precisely the elements with depth 0.

There is a very simple schedule that completes every task in its minimum number of steps: just use a "greedy" strategy of performing tasks as soon as possible. Schedule all the elements of depth k at step k. That's how we found the above schedule for getting dressed.

[7] We think it would be nicer to call them the *parts* of the partition, but "blocks" is the standard terminology.

Theorem 9.5.8. *A minimum time schedule for a finite DAG D consists of the sets $A_0, A_1, \ldots$, where*

$$A_k ::= \{a \in V(D) \mid \text{depth}(a) = k\}.$$

We'll leave to Problem 9.19 the proof that the sets A_k are a parallel schedule according to Definition 9.5.7. We can summarize the story above in this way: with an unlimited number of processors, the parallel time to complete all tasks is simply the size of a critical path:

Corollary 9.5.9. *Parallel time = size of critical path.*

Things get more complex when the number of processors is bounded; see Problem 9.20 for an example.

9.5.3 Dilworth's Lemma

Definition 9.5.10. An *antichain* in a DAG is a set of vertices such that *no* two elements in the set are comparable—no walk exists between any two different vertices in the set.

Our conclusions about scheduling also tell us something about antichains.

Corollary 9.5.11. *In a DAG, D, if the size of the largest chain is t, then $V(D)$ can be partitioned into t antichains.*

Proof. Let the antichains be the sets $A_k ::= \{a \in V(D) \mid \text{depth}(a) = k\}$. It is an easy exercise to verify that each A_k is an antichain (Problem 9.19). ■

Corollary 9.5.11 implies[8] a famous result about acyclic digraphs:

Lemma 9.5.12 (Dilworth)**.** *For all $t > 0$, every DAG with n vertices must have either a chain of size greater than t or an antichain of size at least n/t.*

Proof. Assume that there is no chain of size greater than t. Let ℓ be the size of the largest antichain. If we make a parallel schedule according to the proof of Corollary 9.5.11, we create a number of antichains equal to the size of the largest chain, which is less than or equal t. Each element belongs to exactly one antichain, none of which are larger than ℓ. So the total number of elements at most ℓ times t—that is, $\ell t \geq n$. Simple division implies that $\ell \geq n/t$. ■

[8] Lemma 9.5.12 also follows from a more general result known as Dilworth's Theorem, which we will not discuss.

Corollary 9.5.13. *Every DAG with n vertices has a chain of size greater than $\sqrt{n}$ or an antichain of size at least $\sqrt{n}$.*

Proof. Set $t = \sqrt{n}$ in Lemma 9.5.12. ■

Example 9.5.14. When the man in our example is getting dressed, $n = 10$.
Try $t = 3$. There is a chain of size 4.
Try $t = 4$. There is no chain of size 5, but there is an antichain of size $4 \geq 10/4$.

9.6 Partial Orders

After mapping the "direct prerequisite" relation onto a digraph, we were then able to use the tools for understanding computer scientists' graphs to make deductions about something as mundane as getting dressed. This may or may not have impressed you, but we can do better. In the introduction to this chapter, we mentioned a useful fact that bears repeating: any digraph is formally the same as a binary relation whose domain and codomain are its vertices. This means that *any* binary relation whose domain is the same as its codomain can be translated into a digraph! Talking about the edges of a binary relation or the image of a set under a digraph may seem odd at first, but doing so will allow us to draw important connections between different types of relations. For instance, we can apply Dilworth's lemma to the "direct prerequisite" relation for getting dressed, because the graph of that relation was a DAG.

But how can we tell if a binary relation is a DAG? And once we know that a relation is a DAG, what exactly can we conclude? In this section, we will abstract some of the properties that a binary relation might have, and use those properties to define classes of relations. In particular, we'll explain this section's title, *partial orders*.

9.6.1 The Properties of the Walk Relation in DAGs

To begin, let's talk about some features common to all digraphs. Since merging a walk from u to v with a walk from v to w gives a walk from u to w, both the walk and positive walk relations have a relational property called *transitivity*:

Definition 9.6.1. A binary relation, R, on a set, A, is *transitive* iff

$$(a \; R \; b \text{ AND } b \; R \; c) \text{ IMPLIES } a \; R \; c$$

for every $a, b, c \in A$.

So we have

Lemma 9.6.2. *For any digraph, G, the walk relations G^+ and G^* are transitive.*

Since there is a length zero walk from any vertex to itself, the walk relation has another relational property called *reflexivity*:

Definition 9.6.3. A binary relation, R, on a set, A, is *reflexive* iff $a \ R \ a$ for all $a \in A$.

Now we have

Lemma 9.6.4. *For any digraph, G, the walk relation G^* is reflexive.*

We know that a digraph is a DAG iff it has no positive length closed walks. Since any vertex on a closed walk can serve as the beginning and end of the walk, saying a graph is a DAG is the same as saying that there is no positive length path from any vertex back to itself. This means that the positive walk relation of D^+ of a DAG has a relational property called *irreflexivity*.

Definition 9.6.5. A binary relation, R, on a set, A, is *irreflexive* iff

$$\text{NOT}(a \ R \ a)$$

for all $a \in A$.

So we have

Lemma 9.6.6. *R is a DAG iff R^+ is irreflexive.*

9.6.2 Strict Partial Orders

Here is where we begin to define interesting classes of relations:

Definition 9.6.7. A relation that is transitive and irreflexive is called a *strict partial order*.

A simple connection between strict partial orders and DAGs now follows from Lemma 9.6.6:

Theorem 9.6.8. *A relation R is a strict partial order iff R is the positive walk relation of a DAG.*

Strict partial orders come up in many situations which on the face of it have nothing to do with digraphs. For example, the less-than order, $<$, on numbers is a strict partial order:

- if $x < y$ and $y < z$ then $x < z$, so less-than is transitive, and
- NOT$(x < x)$, so less-than is irreflexive.

The proper containment relation $\subset$ is also a partial order:

- if $A \subset B$ and $B \subset C$ then $A \subset C$, so containment is transitive, and
- NOT$(A \subset A)$, so proper containment is irreflexive.

If there are two vertices that are reachable from each other, then there is a positive length closed walk that starts at one vertex, goes to the other, and then comes back. So DAGs are digraphs in which no two vertices are mutually reachable. This corresponds to a relational property called *asymmetry*.

Definition 9.6.9. A binary relation, R, on a set, A, is *asymmetric* iff

$$a \; R \; b \text{ IMPLIES NOT}(b \; R \; a)$$

for all $a, b \in A$.

So we can also characterize DAGs in terms of asymmetry:

Corollary 9.6.10. *A digraph D is a DAG iff D^+ is asymmetric.*

Corollary 9.6.10 and Theorem 9.6.8 combine to give

Corollary 9.6.11. *A binary relation R on a set A is a strict partial order iff it is transitive and asymmetric.*[9]

A strict partial order may be the positive walk relation of different DAGs. This raises the question of finding a DAG with the *smallest* number of edges that determines a given strict partial order. For *finite* strict partial orders, the smallest such DAG turns out to be unique and easy to find (see Problem 9.25).

9.6.3 Weak Partial Orders

The less-than-or-equal relation, $\leq$, is at least as familiar as the less-than strict partial order, and the ordinary containment relation, $\subseteq$, is even more common than the proper containment relation. These are examples of *weak partial orders*, which are just strict partial orders with the additional condition that every element is related to itself. To state this precisely, we have to relax the asymmetry property so it does not apply when a vertex is compared to itself; this relaxed property is called *antisymmetry*:

[9] Some texts use this Corollary to define strict partial orders.

Definition 9.6.12. A binary relation, R, on a set A, is *antisymmetric* iff, for all $a \neq b \in A$,

$$a \; R \; b \quad \text{IMPLIES} \quad \text{NOT}(b \; R \; a)$$

Now we can give an axiomatic definition of weak partial orders that parallels the definition of strict partial orders.[10]

Definition 9.6.13. A binary relation on a set is a *weak partial order* iff it is transitive, reflexive, and antisymmetric.

The following lemma gives another characterization of weak partial orders that follows directly from this definition.

Lemma 9.6.14. *A relation R on a set, A, is a weak partial order iff there is a strict partial order, S, on A such that*

$$a \; R \; b \quad \textit{iff} \quad (a \; S \; b \;\; \text{OR} \;\; a = b),$$

for all $a, b \in A$.

Since a length zero walk goes from a vertex to itself, this lemma combined with Theorem 9.6.8 yields:

Corollary 9.6.15. *A relation is a weak partial order iff it is the walk relation of a DAG.*

For weak partial orders in general, we often write an ordering-style symbol like $\preceq$ or $\sqsubseteq$ instead of a letter symbol like R.[11] Likewise, we generally use $\prec$ or $\sqsubset$ to indicate a strict partial order.

Two more examples of partial orders are worth mentioning:

Example 9.6.16. Let A be some family of sets and define $a \; R \; b$ iff $a \supset b$. Then R is a strict partial order.

Example 9.6.17. The divisibility relation is a weak partial order on the nonnegative integers.

For practice with the definitions, you can check that two more examples are vacuously partial orders on a set D: the identity relation Id_D is a weak partial order, and the *empty relation*—the relation with no arrows—is a strict partial order.

[10] Some authors define partial orders to be what we call weak partial orders, but we'll use the phrase "partial order" to mean either a weak or strict one.

[11] General relations are usually denoted by a letter like R instead of a cryptic squiggly symbol, so $\preceq$ is kind of like the musical performer/composer Prince, who redefined the spelling of his name to be his own squiggly symbol. A few years ago he gave up and went back to the spelling "Prince."

9.7 Representing Partial Orders by Set Containment

Axioms can be a great way to abstract and reason about important properties of objects, but it helps to have a clear picture of the things that satisfy the axioms. DAGs provide one way to picture partial orders, but it also can help to picture them in terms of other familiar mathematical objects. In this section, we'll show that every partial order can be pictured as a collection of sets related by containment. That is, every partial order has the "same shape" as such a collection. The technical word for "same shape" is "isomorphic."

Definition 9.7.1. A binary relation, R, on a set, A, is *isomorphic* to a relation, S, on a set B iff there is a relation-preserving bijection from A to B; that is, there is a bijection $f : A \to B$ such that for all $a, a' \in A$,

$$a \; R \; a' \quad \text{iff} \quad f(a) \; S \; f(a').$$

To picture a partial order, $\preceq$, on a set, A, as a collection of sets, we simply represent each element A by the set of elements that are $\preceq$ to that element, that is,

$$a \longleftrightarrow \{b \in A \mid b \preceq a\}.$$

For example, if $\preceq$ is the divisibility relation on the set of integers, $\{1, 3, 4, 6, 8, 12\}$, then we represent each of these integers by the set of integers in A that divides it. So

$$\begin{aligned} 1 &\longleftrightarrow \{1\} \\ 3 &\longleftrightarrow \{1, 3\} \\ 4 &\longleftrightarrow \{1, 4\} \\ 6 &\longleftrightarrow \{1, 3, 6\} \\ 8 &\longleftrightarrow \{1, 4, 8\} \\ 12 &\longleftrightarrow \{1, 3, 4, 6, 12\} \end{aligned}$$

So, the fact that $3 \mid 12$ corresponds to the fact that $\{1, 3\} \subseteq \{1, 3, 4, 6, 12\}$.

In this way we have completely captured the weak partial order $\preceq$ by the subset relation on the corresponding sets. Formally, we have

Lemma 9.7.2. *Let $\preceq$ be a weak partial order on a set, A. Then $\preceq$ is isomorphic to the subset relation, $\subseteq$, on the collection of inverse images under the $\preceq$ relation of elements $a \in A$.*

We leave the proof to Problem 9.29. Essentially the same construction shows that strict partial orders can be represented by sets under the proper subset relation, $\subset$ (Problem 9.30). To summarize:

Theorem 9.7.3. *Every weak partial order, $\preceq$, is isomorphic to the subset relation, $\subseteq$, on a collection of sets.*

Every strict partial order, $\prec$, is isomorphic to the proper subset relation, $\subset$, on a collection of sets.

9.8 Linear Orders

The familiar order relations on numbers have an important additional property: given two different numbers, one will be bigger than the other. Partial orders with this property are said to be *linear orders*. You can think of a linear order as one where all the elements are lined up so that everyone knows exactly who is ahead and who is behind them in the line. [12]

Definition 9.8.1. Let R be a binary relation on a set, A, and let a, b be elements of A. Then a and b are *comparable* with respect to R iff [$a \ R \ b$ OR $b \ R \ a$]. A partial order for which every two different elements are comparable is called a *linear order*.

So $<$ and $\leq$ are linear orders on $\mathbb{R}$. On the other hand, the subset relation is *not* linear, since, for example, any two different finite sets of the same size will be incomparable under $\subseteq$. The prerequisite relation on Course 6 required subjects is also not linear because, for example, neither 8.01 nor 6.042 is a prerequisite of the other.

9.9 Product Orders

Taking the product of two relations is a useful way to construct new relations from old ones.

[12] Linear orders are often called "total" orders, but this terminology conflicts with the definition of "total relation," and it regularly confuses students.

Being a linear order is a much stronger condition than being a partial order that is a total relation. For example, any weak partial order is a total relation but generally won't be linear.

Definition 9.9.1. The product, $R_1 \times R_2$, of relations R_1 and R_2 is defined to be the relation with

$$\begin{aligned} \text{domain}(R_1 \times R_2) &::= \text{domain}(R_1) \times \text{domain}(R_2), \\ \text{codomain}(R_1 \times R_2) &::= \text{codomain}(R_1) \times \text{codomain}(R_2), \\ (a_1, a_2)\,(R_1 \times R_2)\,(b_1, b_2) &\quad \text{iff} \quad [a_1\ R_1\ b_1 \text{ and } a_2\ R_2\ b_2]. \end{aligned}$$

It follows directly from the definitions that products preserve the properties of transitivity, reflexivity, irreflexivity, and antisymmetry (see Problem 9.41). If R_1 and R_2 both have one of these properties, then so does $R_1 \times R_2$. This implies that if R_1 and R_2 are both partial orders, then so is $R_1 \times R_2$.

Example 9.9.2. Define a relation, Y, on age-height pairs of being younger *and* shorter. This is the relation on the set of pairs (y, h) where y is a nonnegative integer ≤ 2400 that we interpret as an age in months, and h is a nonnegative integer ≤ 120 describing height in inches. We define Y by the rule

$$(y_1, h_1)\ Y\ (y_2, h_2) \quad \text{iff} \quad y_1 \leq y_2 \ \text{ AND } \ h_1 \leq h_2.$$

That is, Y is the product of the $\leq$-relation on ages and the $\leq$-relation on heights.

Since both ages and heights are ordered numerically, the age-height relation Y is a partial order. Now suppose we have a class of 101 students. Then we can apply Dilworth's lemma 9.5.12 to conclude that there is a chain of 11 students—that is, 11 students who get taller as they get older–or an antichain of 11 students—that is, 11 students who get taller as they get younger, which makes for an amusing in-class demo.

On the other hand, the property of being a linear order is not preserved. For example, the age-height relation Y is the product of two linear orders, but it is not linear: the age 240 months, height 68 inches pair, (240,68), and the pair (228,72) are incomparable under Y.

9.10 Equivalence Relations

Definition 9.10.1. A relation is an *equivalence relation* if it is reflexive, symmetric, and transitive.

Congruence modulo n is an important example of an equivalence relation:

- It is reflexive because $x \equiv x \pmod{n}$.

- It is symmetric because $x \equiv y \pmod{n}$ implies $y \equiv x \pmod{n}$.
- It is transitive because $x \equiv y \pmod{n}$ and $y \equiv z \pmod{n}$ imply that $x \equiv z \pmod{n}$.

There is an even more well-known example of an equivalence relation: equality itself.

Any total function defines an equivalence relation on its domain:

Definition 9.10.2. If $f : A \to B$ is a total function, define a relation $\equiv_f$ by the rule:

$$a \equiv_f a' \text{ IFF } f(a) = f(a').$$

From its definition, $\equiv_f$ is reflexive, symmetric and transitive because these are properties of equality. That is, $\equiv_f$ is an equivalence relation. This observation gives another way to see that congruence modulo n is an equivalence relation: the Remainder Lemma 8.6.1 implies that congruence modulo n is the same as $\equiv_r$ where $r(a)$ is the remainder of a divided by n.

In fact, a relation is an equivalence relation iff it equals $\equiv_f$ for some total function f (see Problem 9.47). So equivalence relations could have been defined using Definition 9.10.2.

9.10.1 Equivalence Classes

Equivalence relations are closely related to partitions because the images of elements under an equivalence relation are the blocks of a partition.

Definition 9.10.3. Given an equivalence relation $R : A \to A$, the *equivalence class*, $[a]_R$, of an element $a \in A$ is the set of all elements of A related to a by R. Namely,

$$[a]_R ::= \{x \in A \mid a \; R \; x\}.$$

In other words, $[a]_R$ is the image $R(a)$.

For example, suppose that $A = \mathbb{Z}$ and $a \; R \; b$ means that $a \equiv b \pmod{5}$. Then

$$[7]_R = \{\ldots, -3, 2, 7, 12, 22, \ldots\}.$$

Notice that 7, 12, 17, etc., all have the same equivalence class; that is, $[7]_R = [12]_R = [17]_R = \cdots$.

There is an exact correspondence between equivalence relations on A and partitions of A. Namely, given any partition of a set, being in the same block is obviously an equivalence relation. On the other hand we have:

Theorem 9.10.4. *The equivalence classes of an equivalence relation on a set A are the blocks of a partition of A.*

We'll leave the proof of Theorem 9.10.4 as a basic exercise in axiomatic reasoning (see Problem 9.46), but let's look at an example. The congruent-mod-5 relation partitions the integers into five equivalence classes:

$$\{\ldots, -5, 0, 5, 10, 15, 20, \ldots\}$$
$$\{\ldots, -4, 1, 6, 11, 16, 21, \ldots\}$$
$$\{\ldots, -3, 2, 7, 12, 17, 22, \ldots\}$$
$$\{\ldots, -2, 3, 8, 13, 18, 23, \ldots\}$$
$$\{\ldots, -1, 4, 9, 14, 19, 24, \ldots\}$$

In these terms, $x \equiv y \pmod 5$ is equivalent to the assertion that x and y are both in the same block of this partition. For example, $6 \equiv 16 \pmod 5$, because they're both in the second block, but $2 \not\equiv 9 \pmod 5$ because 2 is in the third block while 9 is in the last block.

In social terms, if "likes" were an equivalence relation, then everyone would be partitioned into cliques of friends who all like each other and no one else.

9.11 Summary of Relational Properties

A relation $R : A \to A$ is the same as a digraph with vertices A.

Reflexivity R is *reflexive* when

$$\forall x \in A.\ x\ R\ x.$$

Every vertex in R has a self-loop.

Irreflexivity R is *irreflexive* when

$$\text{NOT}[\exists x \in A.\ x\ R\ x].$$

There are no self-loops in R.

Symmetry R is *symmetric* when

$$\forall x, y \in A.\ x\ R\ y \text{ IMPLIES } y\ R\ x.$$

If there is an edge from x to y in R, then there is an edge back from y to x as well.

Asymmetry R is *asymmetric* when

$$\forall x, y \in A.\ x\ R\ y \text{ IMPLIES NOT}(y\ R\ x).$$

There is at most one directed edge between any two vertices in R, and there are no self-loops.

Antisymmetry R is *antisymmetric* when

$$\forall x \neq y \in A.\ x\ R\ y \text{ IMPLIES NOT}(y\ R\ x).$$

Equivalently,

$$\forall x, y \in A.\ (x\ R\ y \text{ AND } y\ R\ x) \text{ IMPLIES } x = y.$$

There is at most one directed edge between any two distinct vertices, but there may be self-loops.

Transitivity R is *transitive* when

$$\forall x, y, z \in A.\ (x\ R\ y \text{ AND } y\ R\ z) \text{ IMPLIES } x\ R\ z.$$

If there is a positive length path from u to v, then there is an edge from u to v.

Linear R is *linear* when

$$\forall x \neq y \in A.\ (x\ R\ y \text{ OR } y\ R\ x)$$

Given any two vertices in R, there is an edge in one direction or the other between them.

For any finite, nonempty set of vertices of R, there is a directed path going through exactly these vertices.

Strict Partial Order R is a *strict partial order* iff R is transitive and irreflexive iff R is transitive and asymmetric iff it is the positive length walk relation of a DAG.

Weak Partial Order R is a *weak partial order* iff R is transitive and anti-symmetric and reflexive iff R is the walk relation of a DAG.

Equivalence Relation R is an *equivalence relation* iff R is reflexive, symmetric and transitive iff R equals the *in-the-same-block*-relation for some partition of domain(R).

6.042J / 18.062J Mathematics for Computer Science
Spring 2015