Massachusetts Institute of Technology
6.042J/18.062J: Mathematics for Computer Science
Professors David Karger and Nancy Lynch

Lecture 6
February 15, 2000

Notes 6

1 Partial Orders

Partial orders are a particular type of binary relation.
They are very important in computer science.
Applications to task scheduling,
database concurrency control,
logical time in distributed computing.

1.1 Definitions for Partial Orders

Definition 1.1 A binary relation $R \subseteq A \times A$ is a **partial order** if it is reflexive, transitive, and antisymmetric.

Quick review:

reflexive: aRa

transitive: $aRb \wedge bRc \Rightarrow aRc$

symmetric: $aRb \Rightarrow bRa$

antisymmetric: $aRb \wedge bRa \Rightarrow a = b$. i.e., $\forall a \neq b, aRb \Rightarrow \neg bRa$. This means *never* symmetric!

For comparison, recall that a relation that is reflexive, transitive, and symmetric is an equivalence relation.

The reflexivity, antisymmetry and transitivity properties are abstract properties that generally describe "ordering" relationships such as "less than or equal to."
So we often write an ordering-style symbol like $\preceq$ instead of just a letter like R, for a p.o. relation.
This lets us use notation similar to for $\leq$.
For example, we write $a \prec b$ if $a \preceq b$ and $a \neq b$.
Similarly, we write $b \succeq a$ as equivalent to $a \preceq b$.

Note this could be misleading – note that $\geq$ is a p.o. on natural numbers, as well as $\leq$.
If we use the $\preceq$ symbol for $\geq$, things look really funny.
In cases like this, better to use R.

A partial order is always defined on some set A.
The set together with the partial order is called a "poset":

Definition 1.2 A set A together with a partial order $\preceq$ is called a **poset** $(A, \preceq)$.

Examples of posets:

- $A = \mathbb{N}, R = \leq$, easy to check reflexive, transitive, antisymmetric
- $A = \mathbb{N}, R = \geq$, same.
- $A = \mathbb{N}, R = <$, NOT because not reflexive
- $A = \mathbb{N}, R = \mid$ (divides), easy to check reflexive, transitive, antisymmetric
- $A = P(\mathbb{N}), R = \subseteq$, check reflexive: $S \subseteq S$, transitive: $S \subseteq S' \wedge S' \subseteq S'' \Rightarrow S \subseteq S''$, antisymmetric: $S \subseteq S' \wedge S' \subseteq S \Rightarrow S = S'$.

The following are *not* posets:

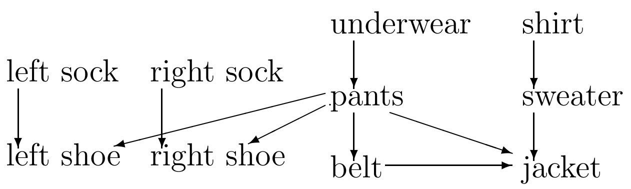
- $A =$ "set of all computers in the world", $R =$ "is (directly or indirectly) connected to". This is not a p.o. because it is not true that $aRb \wedge bRa \Rightarrow a = b$.
  In fact, it is symmetric, transitive. Equivalence relation.
- $A =$ "the set of all propositions", $R = \Rightarrow$, is not antisymmetric. Not symmetric either, so not equivalence relation.

1.2 Directed Acyclic Graph Definition

A common source of partial orders in computer science is in "task graphs".
You have a set of tasks A, and a relation R in which aRb means "b cannot be done until a is finished".
Implicitly, "if all the things that point at b are done, I can do b."

This can be nicely drawn as a graph.
We draw an arrow from a to b if aRb.

For example, below is a graphs that describes the order in which one would put on clothes. The set is of clothes, and the edges say what should be put on before what.



The "depends on" graph imposes an ordering on tasks. But is it a partial order?
No, because it isn't reflexive or transitive.
But there is a natural extension: the reflexive transitive closure *is* a partial order.
It gives the relation "must be done before."

Wait, we know the reflexive transitive closure is reflexive and transitive, but is it antisymmetric?

What if I add a relation edge from belt to underwear?
In that case my dependency graph stops making sense: there is no way to get dressed!

What goes wrong? A cyclic dependency.

Definition 1.3 A *cycle* is a walk that ends where it started (i.e., the last vertex equals the first).

Definition 1.4 A *directed acyclic graph (DAG)* is a directed graph with no cycles.

Theorem 1.5 Any partial order $\preceq$, with the reflexive "loops" $a \preceq a$ removed, is a DAG.

Proof. By contradiction.
Suppose the resulting relation had a cycle $a_1, \ldots, a_k, a_1$ where $a_k \neq a_1$.
By transitivity (and induction on the length of the path) we have $a_1 \preceq a_k$ and $a_k \preceq a_1$.
Thus (since $\preceq$ is a partial order) $a_k = a_1$.
This means the final step in our path is a loop on a_1.
But we removed the loops so this step does not exist, a contradiction.

■

Theorem 1.6 The transitive reflexive closure of any DAG is a partial order.

That is, given a DAG, the relation "there is a path of length 0 or greater from a to b in the DAG" is a partial order.

Proof. The given relation is reflexive by definition, and transitive because, as we saw last time, we can "concatenate" paths from a to b and from b to c to get an a-c path.
So we just have to prove antisymmetry.
So suppose there is a path from a to b and a path from b to a but $a \neq b$.
Then concatenating these paths gives a path from a to a that contains a second vertex $b \neq a$—that is, a cycle.
This contradicts the claim that we started with a DAG.

■

1.3 Hasse Diagrams

If we draw the graph of a poset, we may get *lots* of edges (consider the $\leq$ order on some natural numbers).
But a lot of those edges are "implicit" from the transitivity of the order.
We can get a much less messy picture by leaving out these arrows.

Consider the clothing example.
Under the transitive closure, underwear precedes belt.
But we don't need to see that edge to know it is there.
Similarly loops (which express reflexive relationships) aren't shown.

Definition 1.7 A **Hasse diagram** for a poset $(A, \preceq)$ is a graph:

- whose vertices are the elements of A,
- whose edges (arrows) represent pairs (a, b), where $a \leq b$ but $a \neq b$,
- that contains no "transitive edges",
- for which all pairs in $\preceq$ are obtainable by transitivity from edges in the diagram.

It's a "transitive disclosure" of the poset.

To convert the clothing diagram above to an "official form" Hasse diagram, we must remove the pants $\rightarrow$ jacket arrow.

1.4 Partial vs. Total Orders

A partial order is called *partial* because it is not necessary that an ordering exist between every pair of elements in the set.
E.g., Lshoe and Rshoe have no prescribed ordering between them above.
In the "is a subset of" partial order, for two sets, it's not necessary that either be a subset of the other.

When there is no prescribed order between two elements we say that they are "incomparable".

Definition 1.8 a and b are **incomparable** if neither $a \preceq b$ nor $b \preceq a$.

Definition 1.9 a and b are **comparable** if $a \preceq b$ or $b \preceq a$.

E.g., for subsets of $\mathbb{N}$, $\{1, 2, 3\}$ and $\{2, 3, 4\}$ are incomparable.
However, a partial order need not have incomparable elements.
As a special case, we can have a p.o. in which there is a specified order between every pair of elements.
Such partial orders are called "total orders."

Definition 1.10 A poset $(S, \preceq)$ is **totally ordered** if $(\forall a, b \in S)[a \preceq b \vee b \preceq a]$.

Examples:

- $S = \mathbb{N}, \preceq = \leq$
- $S = \mathbb{N}, \preceq = |$, NOT because neither $3|5$ nor $5|3$
- $S = P(\mathbb{N}), \preceq = \subseteq$, NOT because neither $\{3\} \subseteq \{5\}$ nor $\{5\} \subseteq \{3\}$
- Lexicographic order on pairs of natural numbers, defined by: $(a_1, a_2) \preceq (b_1, b_2)$ if and only if either $a_1 < b_1$, or else $a_1 = b_1$ and $a_2 \leq b_2$.
  This is called "lexicographic" because it's like the dictionary order.

The Hasse diagram of a total order looks like a line.

This has been the basic material. The Book has a bit more: minimal, maximal elements (nothing smaller, nothing larger)
lower bounds, upper bounds (for a subset – $\leq$ everything in the subset, or $\geq$)
lub, glb,
lattices.
Read these parts.

1.5 Topological Sorting

Sometimes when we have a partial order, e.g., of tasks to be performed, we want to obtain a consistent total order.
That is, an order in which to perform all the tasks, one at a time, so as not to conflict with the precedence requirements.

The task of finding an ordering that is consistent with a partial order is known as "topological sorting". I think because the sort is based only on the shape (topology) of the poset, and not on the actual values.

Definition 1.11 A **topological sort** of a finite poset $(A, \preceq)$ is a total ordering of all the elements of A, $a_1, a_2, \cdots, a_n$ in such a way that $\forall i < j$, $a_i \not\succ a_j$.
That is, either $a_i \preceq a_j$ or a_i and a_j are incomparable.

For example, underwear, shirt, Lsock, Rsock, pants, sweater, Lshoe, Rshoe, belt, jacket is a topological sort of how to dress.

One of the nice facts about posets is that such an ordering always exists and is even easy to find:

Theorem 1.12 Every finite poset has a topological sort.

The basic idea to prove this theorem is to pick off a "first" element and then proceed inductively.

Definition 1.13 A **minimal** element a in a poset $(A, \preceq)$ is one for which $(\forall b \in A)[a \preceq b \vee a$ and b are incomparable]. Equivalently, it is an element a for which no $b \prec a$.

Notice that there can be more than one minimal element. There are 4 in the clothes example: Lsock, Rsock, underwear, shirt.

Lemma 1.14 Every finite poset $(A, \preceq)$ has a minimal element.

We'll do two proofs of this theorem.

Proof. Suppose no element of A is minimal.
So every element has one preceding it.
We derive a contradiction.
Take some element $a \in A$ and define the following sequence recursively. $a_0 = a$, and for $i > 0$, a_i is some element that precedes a_{i-1}.

This definition is a bit odd because it is nondeterministic.
But it doesn't matter which preceding element you pick at each step.
The key is that it defines some a_i for every i.

Now note that the sequence $a_{n+1}, \ldots, a_0$ is a walk of length $n + 1$.
As we saw before, this means some element is repeated: $a_i = a_j$ for some $i < j$.
But this means $a_i, \ldots, a_j$ is a cycle, which is impossible in a poset.
So we have a contradiction. ■

If you didn't like that "repeated elements" argument, here is a clean proof by induction.
The basic idea is to pick a "smallest" element to start and then proceed recursively.

If you didn't like that "repeated elements" argument, here is a clean proof by induction.

Proof. By induction on the number of elements in the poset.
The base case is clear. Suppose it is true for all size-n posets.
Consider any size $n + 1$ poset. Delete some element a from the poset.
This leaves a poset on n elements (you can verify this).
So the smaller poset has a minimal element m.

Now consider two cases. First suppose $a \not\preceq m$.
The m is a minimal element of the whole poset.
Now suppose $a \preceq m$. Then I claim a is a minimal element of the whole poset.

To prove this claim we argue by contradiction.
If a is not minimal then some $b \preceq a$.
But then, but transitivity, since $a \preceq m$, we have $b \preceq m$.
But this would prevent m from being minimal in the poset with a removed.
This contradicts our finding of m. ■

Note that an infinite poset might have no minimal element. Consider $\mathbb{Z}$.

Now we can prove the existence of the topological sort.

Proof. By (ordinary) induction.
Let $P(n)$ be "any poset with n elements has a topological sort".
Base case ($P(1)$): for a poset with one element the one element makes a topological sort.

Inductive step: Assume ($P(n)$). Consider a poset of $n+1$ elements.
By the previous lemma it has a minimal element u.
Consider the same relation on $A - \{u\}$.
It is easy to check that it is still reflexive, transitive, and anti-symmetric.
So by the inductive hypothesis, $A - \{u\}$ has a topological sort.
We put u first and now have a topological sort on the whole thing.
(According to the definition of topological sort – u can't be in the wrong order w.r.t. any of the later elements.)

Thus every finite poset has a topological sort, by induction. ■

Example 1.15 Consider the dressing example.
Construct an ordering by picking one item at a time.
At each step, look at the remaining p.o. for remaining elements.
Lsock, shirt, sweater, Rsock, underwear, pants, Lshoe, belt, jacket, Rshoe

E.g., subsets of $\{1,2,3,4\}$
$\emptyset$ is the unique minimal element, then have choices, e.g., do:
$\{1\}$, $\{2\}$, $\{1,2\}$
$\{3\}$, $\{1,3\}$, $\{2,3\}$, $\{1,2,3\}$,
$\{4\}$, $\{1,4\}$, $\{2,4\}$, $\{3,4\}$,
$\{1,2,4\}$, $\{1,3,4\}$, $\{2,3,4\}$, $\{1,2,3,4\}$

I'm not sure if infinite posets always have topological sorts—I think this might be equivalent to the well ordering property, which is equivalent to the axiom of choice, a hotly debated mathematical axiom.

1.6 Parallel Task Scheduling

When elements of a poset are tasks that need to be done and the partial order is precedence constraints, topological sorting provides us with a legal way to execute tasks sequentially, i.e., without violating any precedence constraints.
But what if we have the ability to execute more than one task at the same time?
For example, say tasks are programs, partial order 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 poset of tasks, how long does it take to do them all, in an optimal parallel schedule?
We can use partial order concepts to analyze this problem.

On the clothes example, we could do all the minimal elements first (Lsock, Rsock, underwear, shirt), remove them and repeat.

(We'd need lots of hands, or maybe dressing servants).
We can do pants and sweater next, and then Lshoe, Rshoe, and belt, and finally jacket.

We can't do any better, because the sequence shirt, pants, belt, jacket must be done in that order.
A sequence like this is called a chain.

Definition 1.16 A **chain** in a poset is a sequence of elements of the domain, each of which is smaller than the next in the partial order ($\preceq$ and $\neq$).

That is, a sequence is a simple path in the partial order.

Clearly, the parallel time $\geq$ length of any chain.
For if we used less time, then two tasks in the chain would have to be done at the same time.
But by definition of chains this violates precedence constraints.

A longest chain is also known as a *critical path*.

So we need at least t steps, where t is the length of the longest chain.
Fortunately, it is always possible to use only t:

Theorem 1.17 Given any finite poset $(A, \preceq)$ for which the longest chain has length t, it is possible to partition A into t subsets, $A_1, A_2, \cdots, A_t$ such that $(\forall i \in [1, t])(\forall a \in A_i)(\forall b \preceq a, b \neq a)[b \in A_1 \cup A_2 \cup \cdots \cup A_{i-1}]$.

That is, we can divide up the tasks into t "groups" so that for each group i, all tasks that have to precede tasks in i are in smaller-numbered groups.

Corollary 1.18 For $(A, \preceq)$ and t as above, it is possible to schedule all tasks in t steps.

Proof. $\forall i$, schedule all elements of A_i at time i.
This satisfies the precedence requirements, because all tasks that must precede a task are scheduled at preceding times. ■

Corollary 1.19 parallel time = length of longest chain

So it remains to prove the partition theorem:

Proof. Use the following rule: Put each a in set A_i, where i is the length of the longest chain ending at a.

Let's see why this works.
It gives just t sets, because the longest path is length t.
We need to show $(\forall i)(\forall a \in A_i)$
$(\forall b \preceq a, b \neq a)[b \in A_1 \cup A_2 \cup \cdots \cup A_{i-1}]$.

The proof is by contradiction.
If $b \notin A_1 \cup A_2 \cup \cdots \cup A_{i-1}$ then there is a path of length $> i-1$ ending in b.
This means there is a path of length $> i$ ending in a, which means $a \notin A_i$.
This is a contradiction. ■

So with an unlimited number of processors, the time to complete all the tasks is the length of the longest chain.

It turns out that this theorem is good for more than parallel scheduling. It is usually stated as follows.

Corollary 1.20 If t is the length of the longest chain in a poset $(A, \preceq)$ then A can be partitioned into t antichains.

Definition 1.21 An **antichain** is a set of incomparable elements.

This is a corollary because we showed before how to partition the tasks into sets that could be done at the same time, which means that these were incomparable. (If any two comparable then one would have to precede other and so couldn't be done at same time.)

Review:

- **chain**: all elements comparable (total order)
- **antichain**: all elements incomparable