RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

CS 516 Compilers and Programming Languages II

Approximation 1

Road Ahead

Next week, we will start selecting papers for presentation.

Presenters of the paper are the discussion leaders. A few slides should show the basic "takeaways" of the paper

You can use slides that are available online, and augment them with your views/comments

You will have a meeting with me before presenting your paper

How to Read A Research Paper?

Nice paper by S. Keshav "How to Read a Paper"
Available on our seminar website (on the bottom of page)

Idea: Read paper in three passes

1. Quick read (bird's eye view)
   - Carefully read title, abstract, introduction
   - Read subheadings
   - Read conclusion
2. Read with greater care, but ignore details
   (no proofs, detailed algorithms, etc.)
   - Look at graphs
   - Mark skipped sections
3. Read in detail
   - Try to "re-implement" the paper
   - Put work into bigger context

There are videos and other presentation on the bottom of our course website. Will help you to survive grad school!

How to Structure Your Research Presentation?

Research talk: Chicken Chicken Chicken
Doug Zongker, University of Washington, 2007

https://www.youtube.com/watch?v=yL_-1d9OSdk

Why is this so funny ?
It reflect the basic structure of a **good research talk**,
but is totally content free.

1. Title and motivation of work
   - what are you trying to do?
   - why is this important and we should care?
2. Outline of talk, i.e., what will you talk about?
3. Related work
   - what have others done and what are you doing?
4. Overall system description
   - what was actually done?
5. Experimental validation
   - how did you show how well it worked?
6. Summary and conclusion

There is no "free" lunch!

You cannot have everything, so
something has to give, i.e., has to be approximated!

Why should we care about approximations?

- New dimension of opportunity to improve overall application outcomes

- Basic insight: Nothing is "free"; you get what you "paid" for

- **Possible approximation / "lower quality" tradeoffs**:
  - + performance (execution time)
  - + energy consumption and peak power
  - + privacy
  - + cooling / thermal issues
  - + data / memory size
  - + reliability and robustness of behavior
  - + cost in $$$ (e.g.: money for cloud services)

- Modern architectures are heterogeneous; approximation may help to better take advantage of hardware resources (cores, shadow-cores, GPUs, memory hierarchies, …). Algorithm has to fit into resource envelope.

- There are cases where the ground truth cannot be computed, but an approximation can;
  ME: "Our current algorithms are only approximations of future algorithms"

Key challenges

Challenges

- How to build systems/applications with approximations?
- How to proof properties about an approximation (error bounds)?
- How to reason about approximations?
- How to assess the benefits of different tradeoffs?

Observation

- Approximations and their tradeoff are semantic issues since they modify an application's answer. Therefore, the programmer has to be involved!

- Absolute figures are important (e.g.: 125KJ, 12.35secs, error within 13%);

- Quality is subjective! This means that attaching a "meaning" to absolute figures in terms of their impact on user's quality experience is essential.

Redundancy vs. Approximation

Redundancy has been mainly used to make systems more reliable and robust:

Provide the same functionality

→ input/output behavior at the same quality level

under adverse conditions

→ system component failures (Space Shuttle, Tandem computers, RAID)

→ bit errors (error correction codes)

→ program bugs (multi-version codes implementing different algorithms)

→ communication failures ("robust" protocols: resend message, rerouting)

Approximation is a form of redundancy where you drop the requirement of the same input/output behavior at the same quality level.

Traditional examples:

→ lossy compression

→ dropping communication packets (packets don't reach)

→ asynchronous communication (packets reach, but are overwritten)

My research: How to use redundancy and approximations to build an overall more effective system with a desired and acceptable cost/performance/energy tradeoff.

Classification of Approximations

Underlying assumption: There is **a ground truth** consisting of the **perfect/correct/optimal answer**. This answer may not be effectively computable. An **approximate answer** comes "close" to the ground truth, where "closeness" is an application-specific metric.

Model approximation (solving a real-world problem through an algorithm):

- use a simpler view of the world (ground truth) that still works for a particular application
  (example: assume the world is flat for a car-navigation application)
- even optimal solution of the model is approximation to ground truth (e.g.: simplified physical interactions)

Data approximations :

- less precise data (e.g.: single vs. double)
- use a subset of data (e.g.: statistical representation, subset probing)
- probabilistic data values

Computation approximation:

- solve a model non-optimally (e.g.: use a heuristic)
  - + relaxed convergence criteria
  - + imprecise solution strategies (e.g.: simpler stencil for PDE solvers)
  - + probabilistic solution strategies

Example: Data flow analysis

How to represent information?

Answer: Represent information as "elements" in a semi-lattice

A meet semi-lattice (Info, $\wedge$)
- domain of values Info
- **meet** operator $\wedge$: Info x Info -> Info

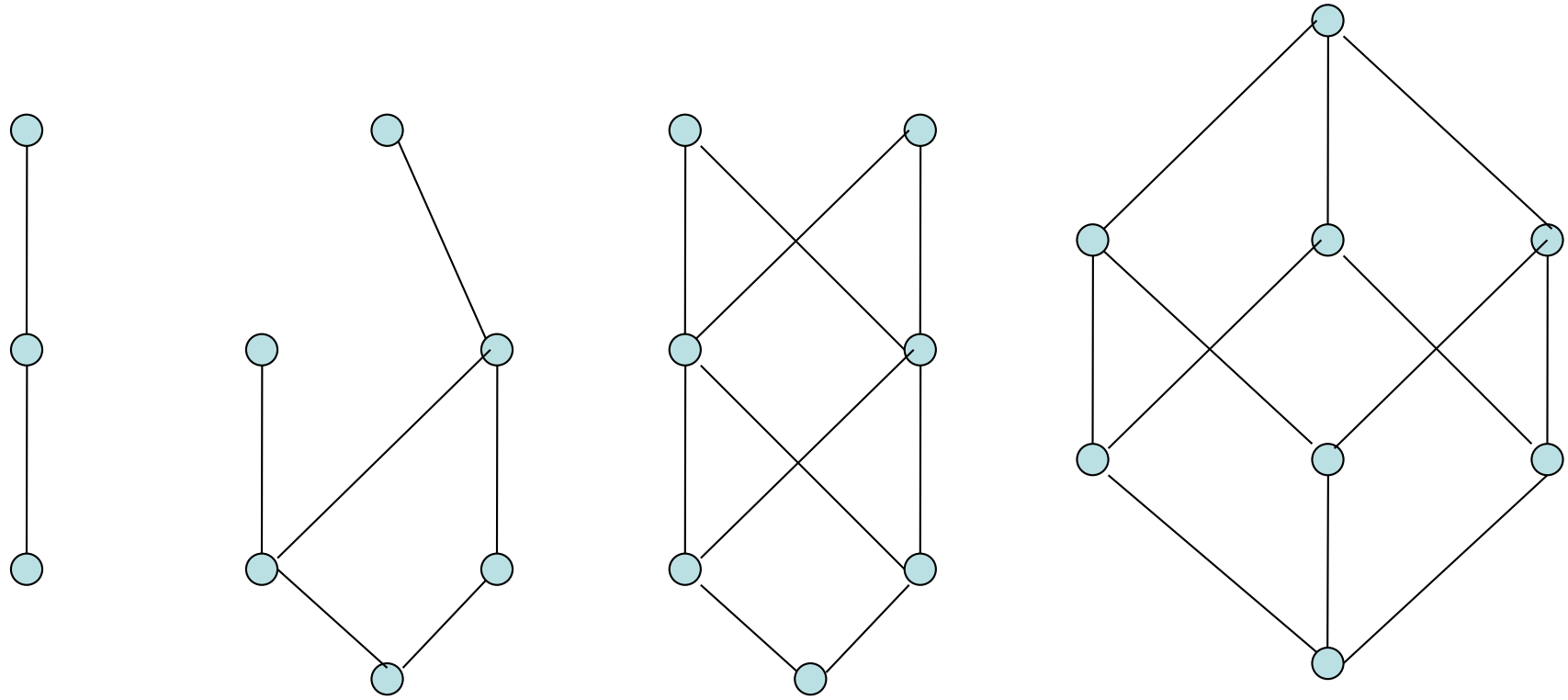
With binary operator $\wedge$ is
- commutative: $a \wedge b = b \wedge a$
- associative: $a \wedge (b \wedge c) = (a \wedge b) \wedge c$
- idempotent: $a \wedge a = a$

The meet operator $\wedge$ can be used to define a partial order over pieces of information:

$$a \wedge b = a \quad \text{IFF} \quad a \leq b$$

Semi-lattice

Examples of partial orders over domain Info:



Greatest lower bound: $x \wedge y$ = First common descendant of x & y;
Has to be uniquely defined!

Are they all meet semi-lattices?

Bounded semi-lattice

A meet semi-lattice is bounded if there exists a top element T,
Such that $x \wedge T = x$ for all x.

A bottom element $\perp$ exists , if $x \wedge \perp = \perp$ for all x.

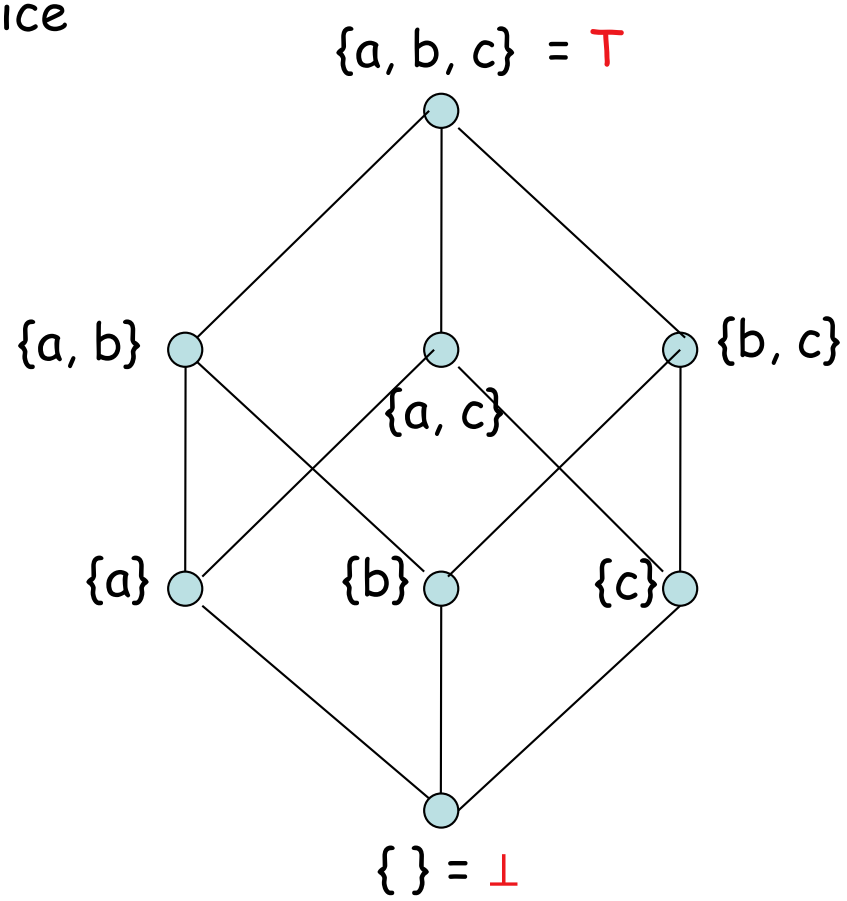
Which of the meet semi-lattice on the previous slides are bounded?

Which have also a bottom element?

What would be an interpretation of the top and bottom element in terms of their information content?

"Canonical" semi-lattice

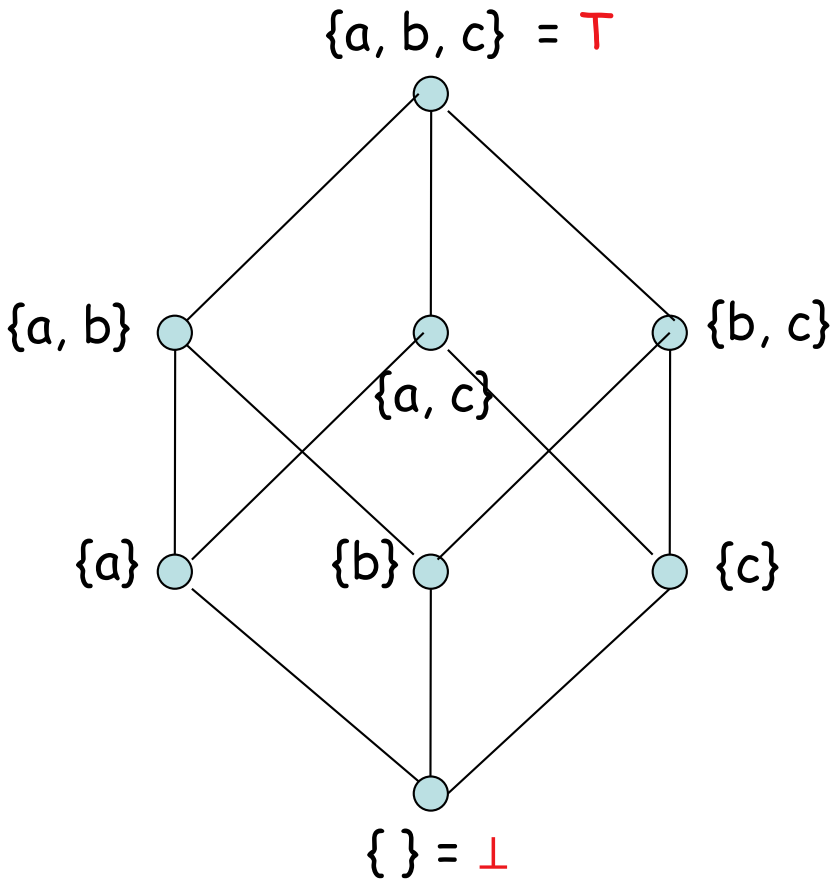
Example semi-lattice



Approximation: We could think of ⊤ as the best quality answer that we can produce, and ⊥ as the least quality answer.

Lattice

Example lattice



A lattice (Info, ∧, ∨) has two binary operators
- domain of values Info
- **meet** operator ∧: Info x Info -> Info
- **join** operator : ∨: Info x Info -> Info

with binary operators are basically "duals" of each other that model information "intersection" (meet) and "union" (join). Both (Info, ∧) and (Info, ∨) are semi-lattices.

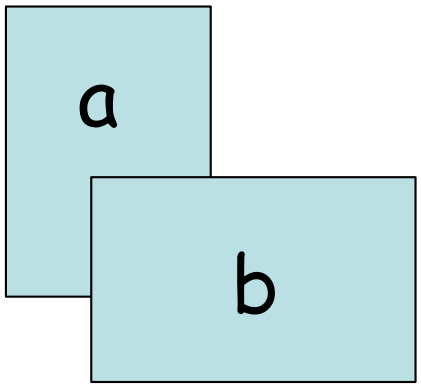
The operators ∧ **and** ∨ define a partial order over Info (pieces of information):

$$a \vee b = b \quad \text{IFF} \quad a \wedge b = a \quad \text{IFF} \quad a \leq b$$

Exercise

You are given a rectangular, 2-dimensional array / data regions.

Build a lattice that models the "intersection" and the "union" of two rectangular regions "a" and "b".
Note: the regions may or may not overlap.



a ∧ b =

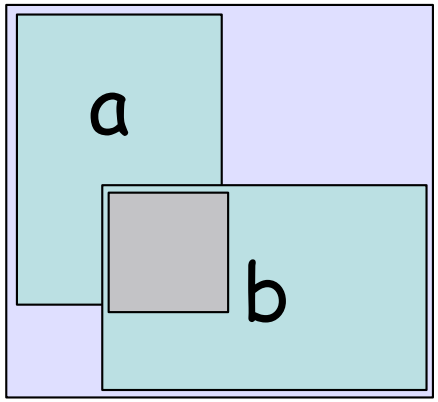
a ∨ b =

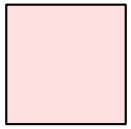
Do we have to use approximation here?

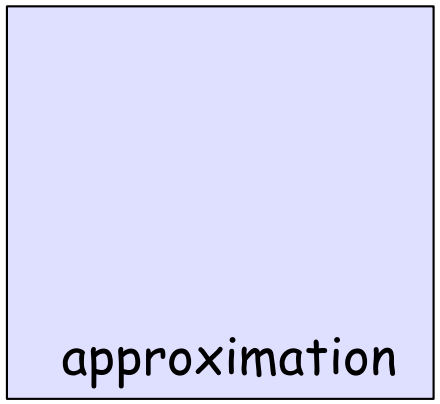
Exercise

You are given a rectangular, 2-dimensional array / data regions.

Build a lattice that models the "intersection" and the "union" of two rectangular regions "a" and "b".
Note: the regions may or may not overlap.



$a \wedge b =$  precise

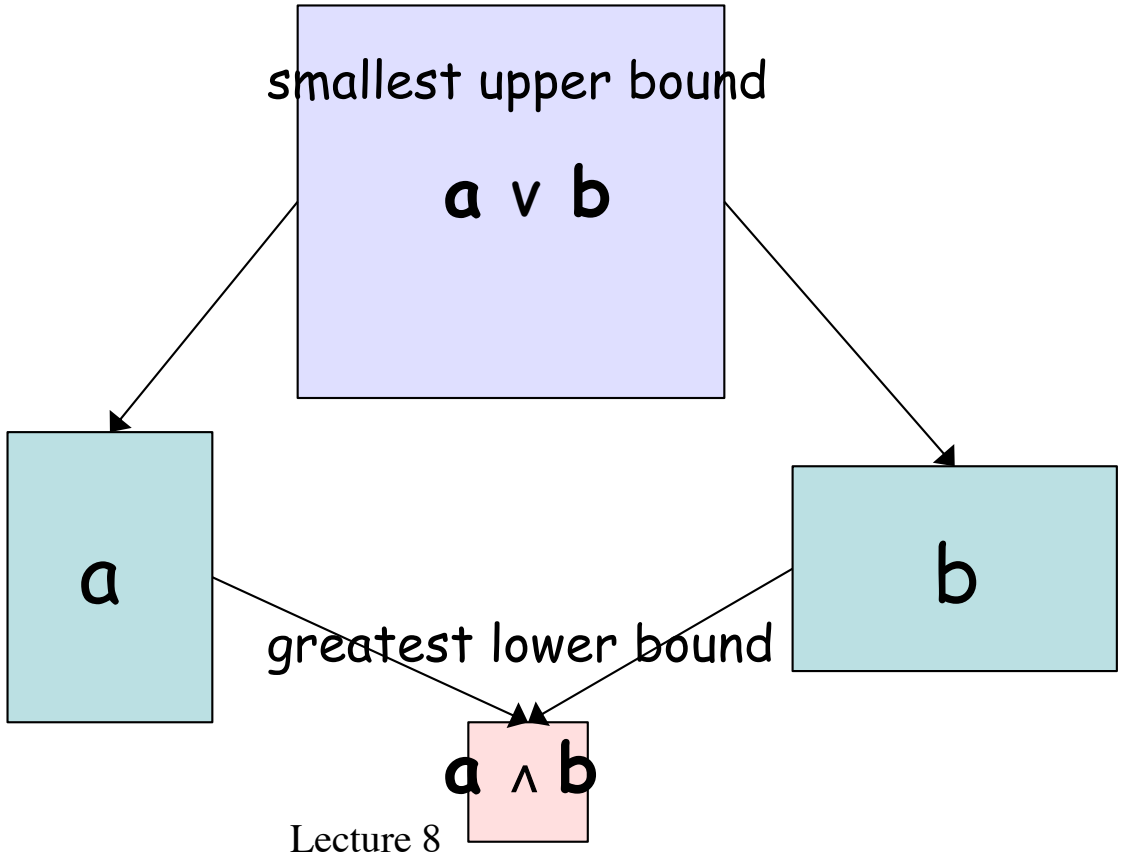
$a \vee b =$ 

Do we have to use approximation here?

Exercise

You are given a rectangular, 2-dimensional array / data regions.

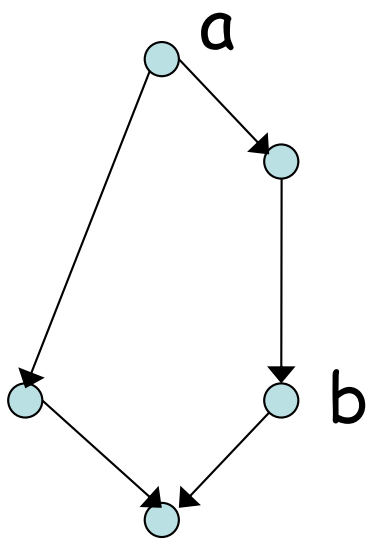
Build a lattice that models the "intersection" and the "union" of two rectangular regions "a" and "b".
Note: the regions may or may not overlap.



Formal: Drawing semi-lattice diagram

- $(x < y) \equiv (x \leq y) \wedge (x \neq y)$

- **A semi-lattice diagram:**
- Set of nodes: set of values/information units
- Set of edges $\{(y, x)$: $x < y$ and $\neg\exists z$ s.t. $(x<z) \wedge (z<y) \}$



a < b, but no edge from a to b

Avoid transitive edges

Descending chain property

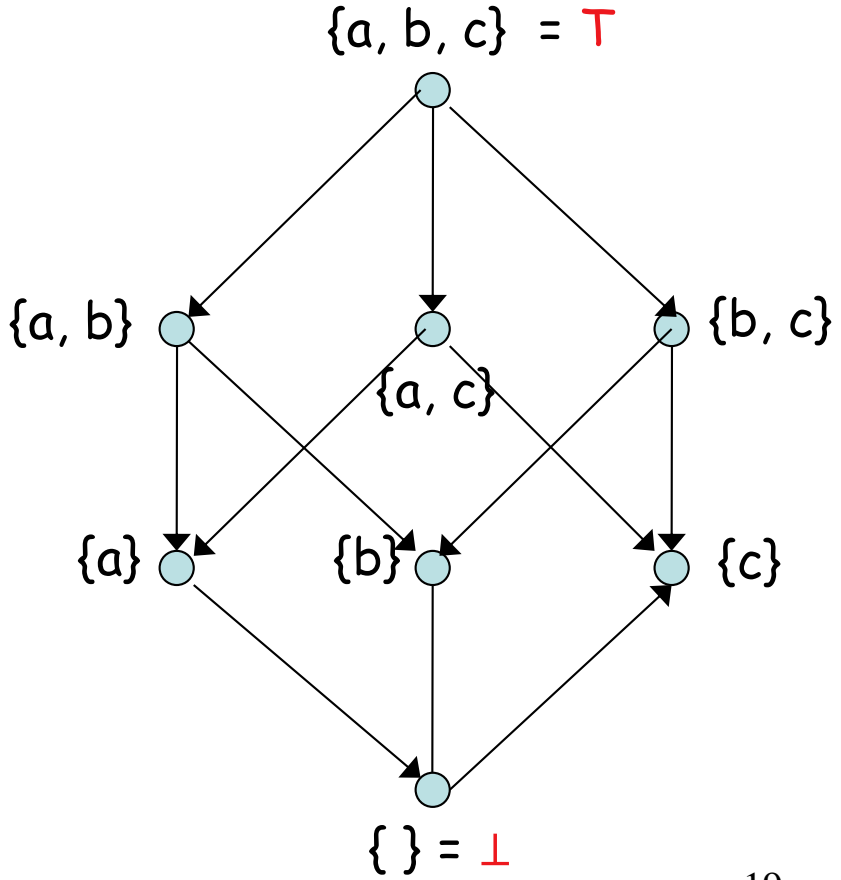
Definition

The **height** of a semi-lattice is the largest number of > relations that will fit in a descending chain:

x0 > x1 >...

Descending chain property:

All chains are of finite height



Example:

What is the maximal height of a chain for the ({a, b, c}, ∩} semi-lattice?

What about the ({a, b, c}, ∪ } semi-lattice?

Next time

More on data flow problems and their relation to approximation