NTNU – Trondheim
Norwegian University of
Science and Technology

Data flow analysis framework

Partial orders, lattices, and operators

From last time

- We defined control flow graphs in terms of
  - Operations
  - Basic blocks of operations (that end in jumps)
  - Program points
- As an example, we looked at live variables...

  (variables that may still be used before their next assignment)

  ...how they can be found by traversing a control flow graph...
  - Collect them in sets attached to program points
  - Find out how instructions affect the sets attached to the neighboring program points
  - Find out how to handle the sets at points where several control flows meet

  ...and how the control flow graph captures every possible execution of the program

  (as well as a few impossible ones, to stay on the safe side)

There's a general procedure here

- Associate program points with sets that represent the information we're after
- Figure out how the sets change
  - As a function of instructions
  - As a function of meeting points between control paths
- Make a safe assumption at an initial point
- Work out the function throughout the graph
- Repeat until the sets stop changing

There are two issues with it

- Will the sets ever stop changing?
- Does the analysis get better by repeated applications?
- We'll talk about the first one today, and the second one later

Convergence

- Will the scheme always work?
  - It will under certain conditions:
    - If the sets have a maximum and minimum possible size, and
    - If the changes we make either only add or remove elements,

    they will necessarily reach a point where they stop changing, so the analysis ends.
- It's good to guarantee that it does reach an end, so that the compiler won't get stuck on analyzing some programs forever

Precision

- How good is the outcome of the analysis?
  - We can call it *precise* if it reflects all the control flows the program can/will take, and none of those it will not take
- A perfectly precise analysis can not be derived by a computer
- It's still good to see if we can say anything about how much precision is lost, and why

Sets and orders

- Some sets have a sequence we're taught in grade school
  - Take the natural numbers, 1 < 2 < 3 < 4 < …
  - The *ordering relation* here is '<'
  - It is a *total* order, because it puts any pair of natural numbers in relation to each other
- Other sets don't have any
  - Take the complex numbers, you can neither say that 1 is bigger, smaller, nor equal to the imaginary unit
- Some sets let you consistently pick how to order them
  - And you can write the ordering relation with some mildly deformed comparison operator like "$\sqsubseteq$", to distinguish it from $\leq$, $\subseteq$, and others

NTNU – Trondheim
Norwegian University of Science and Technology

Partial order relations

- A partial order $(P, \sqsubseteq)$ contains
  - A set of things (P)
  - A partial order relation ($\sqsubseteq$)
- The partial order relation is
  - Reflexive: $x \sqsubseteq x$
  - Anti-symmetric: if $x \sqsubseteq y$ and $y \sqsubseteq x$, then $x = y$
  - Transitive: if $x \sqsubseteq y$ and $y \sqsubseteq z$, then $x \sqsubseteq z$
- For a *total* order then for every y,x either $x \sqsubseteq y$ or $y \sqsubseteq x$
- In partial orders, not every pair needs to be comparable

An example

- We can partially order some food ingredients, for illustration
- Let x ⊑ y denote that x is an ingredient in y

    flour ⊑ bread

    flour ⊑ pasta

    eggs ⊑ pasta

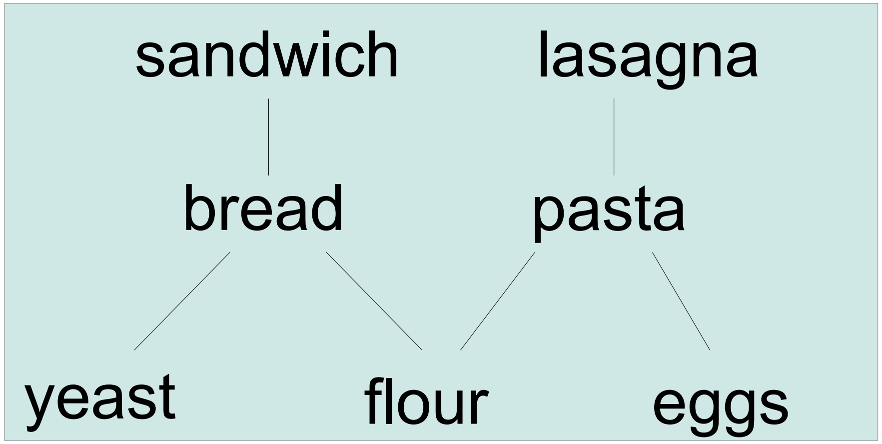
    yeast ⊑ bread

    pasta ⊑ lasagna

    bread ⊑ sandwich

Hasse diagrams

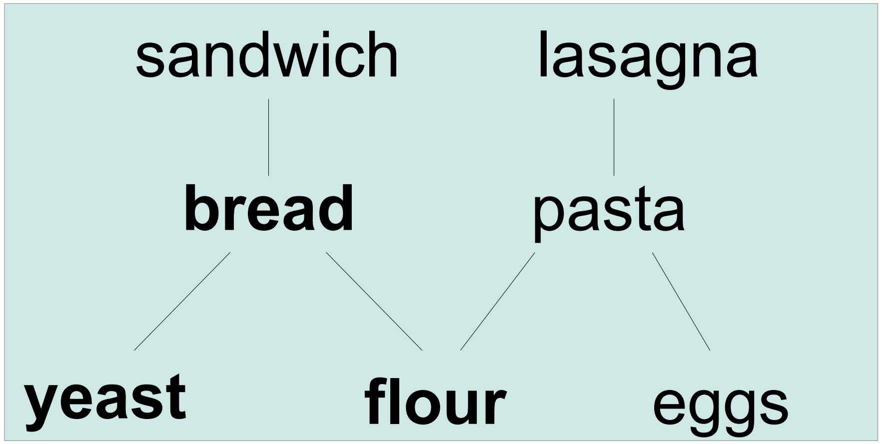
- Keeping transitivity in mind, we can draw a picture of this order



- It's implied that yeast goes into making a sandwich via the bread connection
- There are pairs here which are not comparable by our ingredient relation

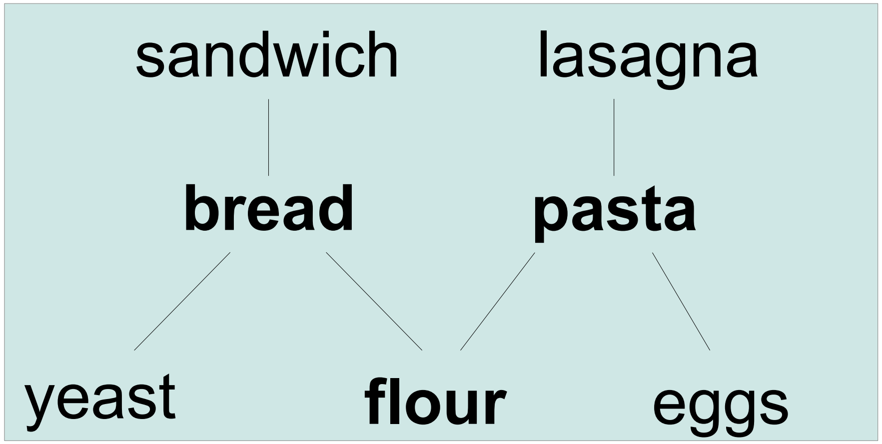
Least Upper Bound (LUB)

- The least upper bound of an element pair is the first thing they have in common, going *up* the order
- LUB(yeast,flour) = bread



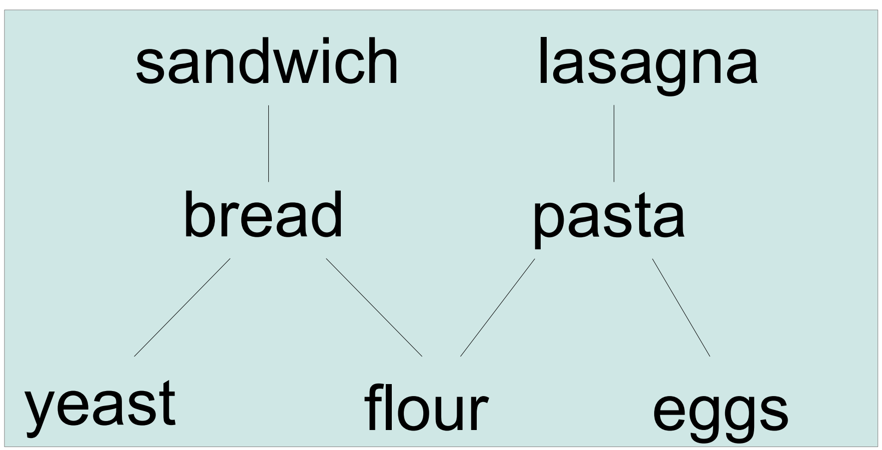
Greatest Lower Bound (GLB)

- The greatest lower bound of an element pair is the first thing they have in common, going *down* the order
- GLB(bread,pasta) = flour



Maximum and minimum

- Partial orders don't necessarily have a unique top or bottom
- GLB(yeast,eggs) doesn't exist
- LUB(sandwich, pasta) doesn't exist either



NTNU – Trondheim
Norwegian University of Science and Technology

Lattices

- A partial order is a *lattice* if any finite (non-empty) subset has a LUB and a GLB
- The natural numbers ordered by < is a lattice
  - If you pick a finite subset, LUB is the biggest number you picked, and GLB is the smallest one
- The natural numbers do have a unique bottom element ($\bot$)
  - It's zero
- They don't have a unique top element ($\top$)
  - They are a countably infinite sequence
- You can pick infinite subsets
  - The even numbers, the odd numbers, the primes...

Complete lattices

- A lattice is complete if **any** (non-empty) subset has a LUB and GLB
- These have top ("biggest") and bottom ("smallest") elements

  For a complete lattice $(L,\sqsubseteq)$

  $\top$ = LUB(L)

  $\bot$ = GLB(L)
- Every finite lattice is complete

NTNU – Trondheim
Norwegian University of Science and Technology

Meet and join

- Just to have some symbols that are independent of how we choose the order, define two operators
- "Meet"

  $x \sqcap y = GLB(x,y)$
- "Join"

  $x \sqcup y = LUB(x,y)$

(…with their natural extension to sets of more elements...)

Power sets

- Enough with the food ingredients, consider the set {a,b,c}
- Its Cartesian* product with itself is the set of all pairs

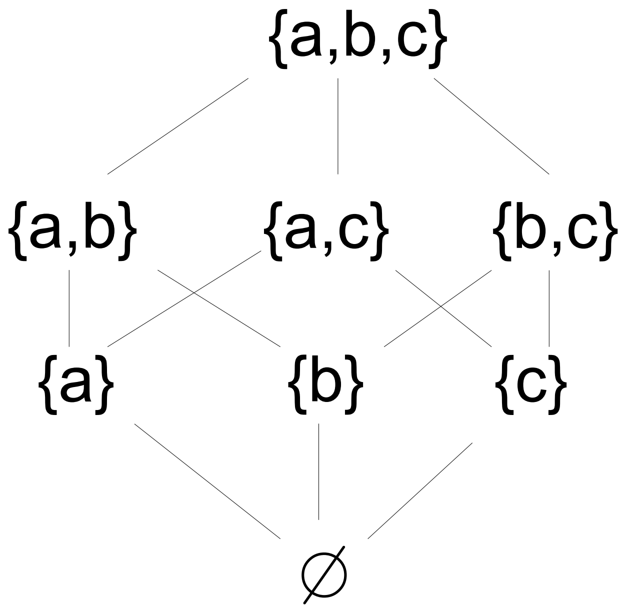
  { {a,b}, {a,c}, {b,c} }
- Its power set is

  {∅, {a}, {b}, {c}, {a,b}, {a,c}, {b,c}, {a,b,c} }
- The power set gives a partial order by the subset relation $\subseteq$

** Technically, the product of all unordered pair combinations is not called "Cartesian", but "n-th symmetric product" is cumbersome to say, and we won't need the distinction for anything.*

The power set lattice

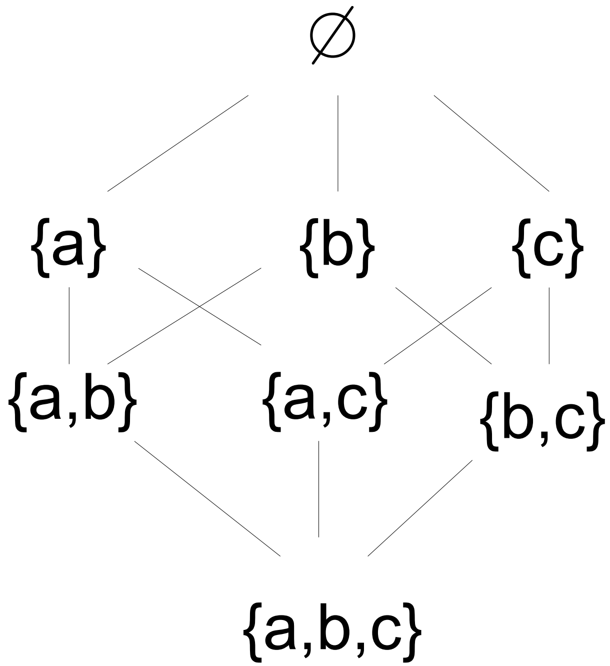
- Ordering relation: $\subseteq$
- Meet operator: $\cap$
- Join operator: $\cup$
- Top: {a,b,c}
- Bottom: $\varnothing$



We can turn it upside/down

Just switch the operators around:

- Ordering relation: $\supseteq$
- Meet operator: $\cup$
- Join operator: $\cap$
- Top: $\emptyset$
- Bottom: {a,b,c}



Connection to live variables

- If we take {a,b,c} to be the three variables in a short program, every possible choice of live variables corresponds to a point in the power set lattice
- If we can express the effect of statements as a *transfer function* from one place to another in the lattice, we can argue that the set attached to a program point only moves in one direction *wrt.* the order when it is applied repeatedly
- That means it will either end up at the top, or stop somewhere before it

Transfer functions

- This is just a formalization of the idea that the instruction between two program points is a function from one place in the lattice to another
- For an instruction I
  - Forward analysis: out[I] = F(in[I])
  - Backward analysis: in[I] = F(out[I])

Extension to basic blocks

- The function of a block B is just a nesting of the functions of its component instructions
- Forward:

  out[B] = F_n (F_{n-1} (… (F_2 (F_1 (in[B])))))

- Backward:

  in[B] = F_1 (F_2 (… (F_{n-1} (F_n (out[B])))))

Where paths meet up

- For the points where multiple control flows intersect:
- Forward:

  in[B] = ⊓ { out[B'] | B' is a predecessor of B }
- Backward:

  out[B] = ⊓ { in[B'] | B' is a successor of B }

If we really wanted to, we could use ⊔ instead and reverse the orders

With ⊓, transfers in the lattice move toward its bottom

With ⊔, transfers in the lattice move toward its top

NTNU – Trondheim
Norwegian University of Science and Technology