Lesson 2

Data flow-based analysis

Static Program Analysis and Constraint Solving

Master's Degree in Formal Methods in Computer Science

Manuel Montenegro (montenegro@fdi.ucm.es)

Departamento de Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid

Outline

1. Introduction to static analysis

2. Data-flow analysis

3. Live variable analysis

4. Available expressions analysis

5. Monotone frameworks

    Introduction to lattices

    Lattice construction

    Instances of monotone frameworks

Introduction to static analysis

- It addresses the properties a program may satisfy.
  - Does a function return an integer value?
  - Does a program terminate for any input?
  - Would a program try to access dangling pointers?
  - Would variables x and y refer to the same memory location?
- Static analysis: it is carried out without executing the program.

In which ways do analysis behave?

Does P hold for the input program?

Does the analysis report P?

	YES	NO
YES	✓	False positive
NO	False negative	✓

Does P hold for the input program?

Does the analysis report P?

	YES	NO
YES	✓	
NO		✓

Theorem
Every non-trivial and extensional property is ***undecidable****.*

- **Non-trivial**: Some programs satisfy the property, some others do not.
- **Extensional:** Replacing a program with an equivalent one preserves the property.

What we would like

	Does P hold for the input program? YES	NO
Does the analysis report P? YES	✓	
NO		✓

- If we want to achieve this, we have to relax our expectations:
  - Analysis with manual intervention, or
  - Analysis constrained to a decidable subset of the target language, or
  - Analysis of programs that finish after a finite amount of computation steps.

What we could achieve: upper approximation

Does P hold for the input program?

Does the analysis report P?

	YES	NO
YES	✓	False positive
NO		✓

- The analysis returns either *No* or *Don't know*.
- Used to detect **unpleasant** properties in a program: dangling pointers, type-related inconsistencies, unexpected sharing, etc.

What we could achieve: lower approximation

Does P hold for the input program?

Does the analysis report P?

	YES	NO
YES	✓	
NO	False negative	✓

- The analysis returns either *Yes* or *Don't know*.
- Used to detect **desirable** properties in programs, e.g., termination.

Static analysis techniques

- Data-flow analysis
  - Gather values computed in every execution point.
- Abstract interpretation
  - Approximate execution of programs by using properties as computable values.
- Type systems
  - Classify computable values into types, and check for discrepancies.
- Symbolic execution
  - Compute those inputs that make the program take a given path.
- **Constraint-based analysis**
- **Model checking**
- **Hoare's logic**, etc.

Data-flow analysis

- A data-flow analysis obtains information on the values computed by a program in each execution point.
- It receives the *Control-Flow Graph* (CFG) of the program being analysed.
- More info on CFGs and how to build them:

S. Muchnick
Advanced Compiler Design and Implementation
Morgan-Kaufmann (1998)

- Variables: $x, y, z \in \textbf{Var}$.
- Numbers: $m \in \textbf{Num}$.
- Arithmetic expressions (**Exp**):

$$e ::= x \mid m \mid e_1 + e_2 \mid e_1 * e_2 \mid \cdots$$

- Boolean expressions:

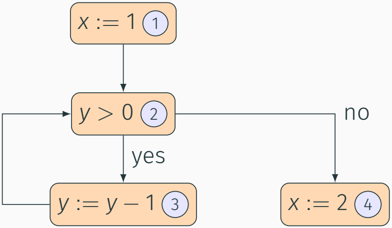
$$b ::= \texttt{true} \mid \texttt{false} \mid \texttt{not}\ b \mid b_1\ \texttt{and}\ b_2 \mid \cdots$$

- Statements (**Stm**):

$$\begin{array}{rcl} S & ::= & \texttt{skip} \\ & \mid & x := e \\ & \mid & S_1; S_2 \\ & \mid & \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2 \\ & \mid & \texttt{while}\ b\ \texttt{do}\ S \end{array}$$

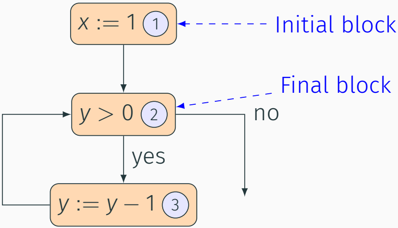
- We assume that programs have been transformed into a CFG, where the basic blocks are `skip` statements, assignments, and boolean expressions.
- Each basic block will contain a unique label taken from the set $\mathbf{Lab} = \{1, 2, 3, \ldots\}$.

```
x := 1;

while y > 0 do
    y := y - 1;

x := 2;
```



- Each program has an **initial block** and a **final block**.
- We assume that the initial block has no predecessors.
    - If this does not hold, add a `skip` block at the beginning.

```
x := 1;

while y > 0 do
    y := y - 1;
```



- Context sensitive/insensitive.
  - Depending on whether it takes into account the call stack leading to a given program point.
- Path sensitive/insensitive.
  - Depending on whether identifies the conditions that lead the execution to take a given path.
- Control-flow sensitive/insensitive.
  - Depending if they consider the order of the basic statements in the program.

- Must analysis
    - It determines whether a property holds in **every** execution path of the program.
- May analysis
    - It determines whether a property holds in **some** execution path of the program.

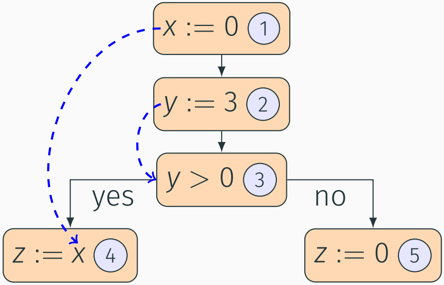
Forward and backward analysis

- Forward analysis
  - The information on a program point is computed from the program points that have been executed before.
- Backward analysis
  - The information on a program point is computed from the program points that might be reached later.

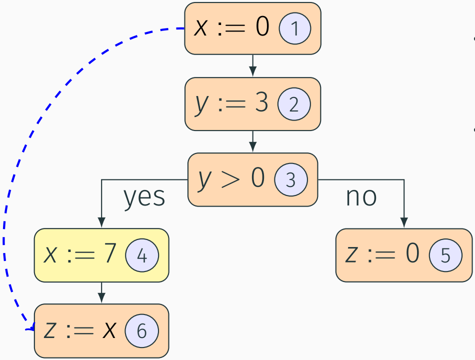
Live variable analysis

Live variable analysis

- A variable is live at some point n if there is an execution path from n to the end of the program in which x is read before being assigned another value.
  - In other words, x is live at some point if its value at that point may be needed in the future.



- x is live at the end of block 1.
- y is live at the end of block 2.
- z is not live in any block.



- x is **not** live at the end of block 1.
- …because its occurrence in block 5 is preceded by an assignment to x in block 4.

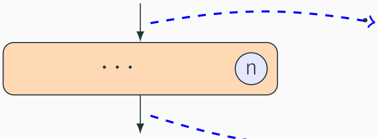
Why is it useful?

- **Register allocation**
  If two variables are never live at the same point, they can be stored in the same machine register.

- **Dead code elimination**
  Any assignment to a variable that is not live at the end of the block can be removed.

Live variable analysis: backward traversal

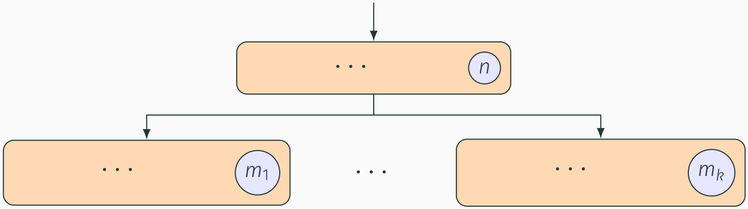
- **Aim:** Determine which variables **may** be live at each program point.
- It will be a **backward analysis**.
    - The information in a given block will be obtained from the blocks executed after it.
- We compute, for each block:



LVIn$_n$: Live variables at the entry of block n.

LVOut$_n$: Live variables at the exit of block n.

- A variable is live at the **exit** of a block if it is live at the **entrance** of any of the blocks following it (successors).



$$LVOut_n = \bigcup_{m \in succ(n)} LVIn_m$$

- If n is a final block, then $LVOut_n = \emptyset$.

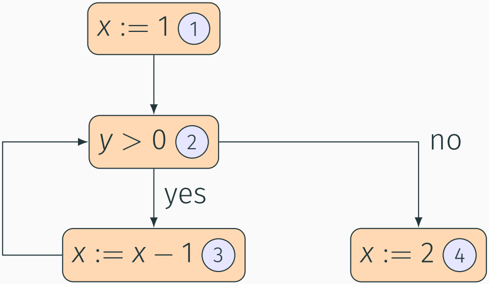
Live variable analysis: data-flow equations

- A variable is live at the **entry** of a block if:
  - This variable is read in that block, or
  - It was live at the end of the block, and it has been assigned to during its execution.

- Let us define:
  - Gen_n: Variables that are read in block n.
  - $Kill_n$: Variables that are written to in block n.
    - If block n contains an assignment $x := e$, then $Kill_n = \{x\}$. Otherwise $Kill_n = \emptyset$.

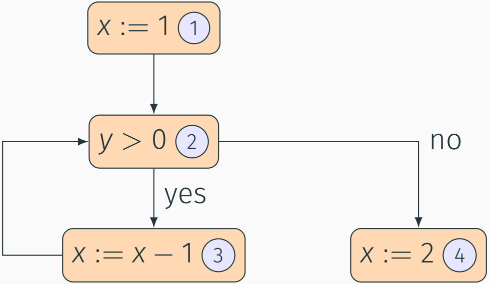
Dataflow equation:

$$LVIn_n = (LVOut_n - Kill_n) \cup Gen_n$$



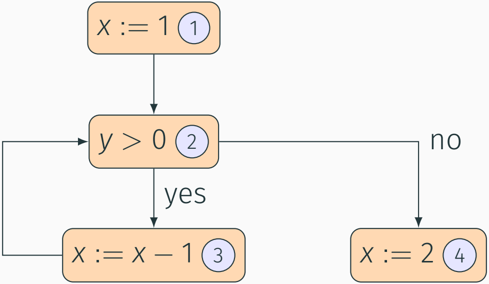
$$
\begin{aligned}
Gen_1 &= \emptyset & \quad Kill_1 &= \{x\} \\
Gen_2 &= \{y\} & \quad Kill_2 &= \emptyset \\
Gen_3 &= \{x\} & \quad Kill_3 &= \{x\} \\
Gen_4 &= \emptyset & \quad Kill_4 &= \{x\}
\end{aligned}
$$

Example



$$
\begin{aligned}
LVOut_1 &= LVIn_2 & LVIn_1 &= (LVOut_1 - \{x\}) \cup \emptyset \\
LVOut_2 &= LVIn_3 \cup LVIn_4 & LVIn_2 &= (LVOut_2 - \emptyset) \cup \{y\} \\
LVOut_3 &= LVIn_2 & LVIn_3 &= (LVOut_3 - \{x\}) \cup \{x\} \\
LVOut_4 &= \emptyset & LVIn_4 &= (LVOut_4 - \{x\}) \cup \emptyset
\end{aligned}
$$

Example



$$
\begin{aligned}
LVOut_1 &= LVIn_2 & LVIn_1 &= LVOut_1 - \{x\} \\
LVOut_2 &= LVIn_3 \cup LVIn_4 & LVIn_2 &= LVOut_2 \cup \{y\} \\
LVOut_3 &= LVIn_2 & LVIn_3 &= LVOut_3 \cup \{x\} \\
LVOut_4 &= \emptyset & LVIn_4 &= LVOut_4 - \{x\}
\end{aligned}
$$

Equation systems

- The result is a system of equations whose unknown variables are $(LVOut_1, \ldots, LVOut_4, LVIn_1, \ldots, LVIn_4)$.

$$
\begin{array}{lcl@{\qquad\qquad}lcl}
LVOut_1 & = & LVIn_2 & LVIn_1 & = & LVOut_1 - \{x\} \\
LVOut_2 & = & LVIn_3 \cup LVIn_4 & LVIn_2 & = & LVOut_2 \cup \{y\} \\
LVOut_3 & = & LVIn_2 & LVIn_3 & = & LVOut_3 \cup \{x\} \\
LVOut_4 & = & \emptyset & LVIn_4 & = & LVOut_4 - \{x\}
\end{array}
$$

- Solutions of this system are subsets of the set of variables in the program. In our case: $\mathsf{Var} = \{x, y\}$.

- Here we have a solution:

$$
\begin{array}{llll}
LVOut_1 = \{x, y\} & LVOut_2 = \{x, y\} & LVOut_3 = \{x, y\} & LVOut_4 = \emptyset \\
LVIn_1 = \{y\} & LVIn_2 = \{x, y\} & LVIn_3 = \{x, y\} & LVIn_4 = \emptyset
\end{array}
$$

- Live variable analysis requires solving a system of equations which involves operations on set.
- Pending questions:

1. Does the system have a solution?
2. Could it have several solutions?
3. In case it has several solutions, is there a solution that is better than the others?
4. In that case, how can we compute the best solution?

- Let us define $F : \mathcal{P}(\mathsf{Var})^8 \to \mathcal{P}(\mathsf{Var})^8$ as follows:

$$F \begin{pmatrix} LVOut_1 \\ LVOut_2 \\ LVOut_3 \\ LVOut_4 \\ LVIn_1 \\ LVIn_2 \\ LVIn_3 \\ LVIn_4 \end{pmatrix} = \begin{pmatrix} LVIn_2 \\ LVIn_3 \cup LVIn_4 \\ LVIn_2 \\ \emptyset \\ LVOut_1 - \{x\} \\ LVOut_2 \cup \{y\} \\ LVOut_3 \cup \{x\} \\ LVOut_4 - \{x\} \end{pmatrix}$$

- If $\mathbf{LV}$ denotes the following vector

$$\mathbf{LV} = (LVOut_1, \ldots, LVOut_4, LVIn_1, \ldots, LVIn_4)^T$$

we can formalize the equation system as follows:

$$\mathbf{LV} = F(\mathbf{LV})$$

- Given a function $F : L \to L$, we say that $x \in L$ is a **fixed point** of F if $F(x) = x$.

$$\mathsf{LV} = F(\mathsf{LV})$$

The solutions of this system are the fixed points of F

1. Does the system have a solution?
2. Could it have several solutions?
3. In case it has several solutions, is there a solution that is better than the others?
4. In that case, how can we compute the best solution?

- Given a function $F : L \to L$, we say that $x \in L$ is a **fixed point** of F if $F(x) = x$.

$$\mathsf{LV} = F(\mathsf{LV})$$

The solutions of this system are the fixed points of F

1. Does F have a fixed point?
2. Could it have several fixed points?
3. In case it has several fixed points, is there a fixed point that is better than the others?
4. In that case, how can we compute the best fixed point?

- In order to know whether a solution is better than the others, we have to set up an **order** between solutions.
- A partial order on a set L is a relation $\sqsubseteq$ satisfying the following properties:
  - *Reflexive:* For all $x \in L$: $x \sqsubseteq x$.
  - *Antisymmetric:* For all $x, y \in L$: if $x \sqsubseteq y$, $y \sqsubseteq x$, then $x = y$.
  - *Transitive:* For all $x, y, z \in L$: if $x \sqsubseteq y$, $y \sqsubseteq z$, then $x \sqsubseteq z$.
- A partially ordered set (*poset*) is a set L together with an order relation $\sqsubseteq$ on this set. It is denoted by $(L, \sqsubseteq)$.

Example

$(\mathcal{P}(\mathbf{Var}), \subseteq)$ is an ordered set, where $\mathbf{Var} = \{x, y\}$.

- Graphical representation of $(\mathcal{P}(\mathsf{Var}), \subseteq)$:

$$
\begin{tikzcd}
 & \{x, y\} \arrow[rr, dashed, bend left] & & \top \\
\{x\} \arrow[ur, dash] & & \{y\} \arrow[ul, dash] & \\
 & \emptyset \arrow[ul, dash] \arrow[ur, dash] \arrow[rr, dashed, bend right] & & \bot
\end{tikzcd}
$$

- A poset is **bounded** if it has a pair of elements $\top, \bot$ such that:
  - $\bot \sqsubseteq x$ for all $x \in L$.
  - $x \sqsubseteq \top$ for all $x \in L$.

- The solutions of our system are not elements of $\mathcal{P}(\mathsf{Var})$, but **tuples** of the set $\mathcal{P}(\mathsf{Var})^8$.
- If $(A_1, \sqsubseteq_1)$, $(A_2, \sqsubseteq_2)$,. . .,$(A_n, \sqsubseteq_n)$ are *posets*, we can define an order $\sqsubseteq$ on $A_1 \times A_2 \times \cdots \times A_n$ as follows:

$$(x_1, \ldots, x_n) \sqsubseteq (y_1, \ldots, y_n) \text{ iff } x_1 \sqsubseteq_1 y_1, x_2 \sqsubseteq_2 y_2, \ldots, x_n \sqsubseteq_n y_n$$

Example

- In $\mathcal{P}(\{x, y\})^2$:

$$(\emptyset, \{x\}) \sqsubseteq (\{x\}, \{x\}) \qquad (\emptyset, \emptyset) \sqsubseteq (\{x\}, \{y\})$$

but

$$(\emptyset, \{x\}) \not\sqsubseteq (\{y\}, \emptyset) \qquad (\{y\}, \emptyset) \not\sqsubseteq (\emptyset, \{x\})$$

- Therefore, $(\mathcal{P}(\mathsf{Var})^8, \sqsubseteq)$ is a poset.
- Moreover, it is bounded. Its lower bound is:

$$\bot = (\underbrace{\emptyset, \emptyset, \cdots, \emptyset}_{\text{8 times}})$$

and its upper bound is:

$$\top = (\underbrace{\mathsf{Var}, \mathsf{Var}, \cdots, \mathsf{Var}}_{\text{8 times}})$$

Monotonic functions

- Given a *poset* $(L, \sqsubseteq)$, a function $F : L \rightarrow L$ is monotonically increasing of only if for all $x, y \in L$:

$$x \sqsubseteq y \implies F(x) \sqsubseteq F(y)$$

- In our example, if F is defined as follows:

$$F\begin{pmatrix} LVOut_1 \\ LVOut_2 \\ LVOut_3 \\ LVOut_4 \\ LVIn_1 \\ LVIn_2 \\ LVIn_3 \\ LVIn_4 \end{pmatrix} = \begin{pmatrix} LVIn_2 \\ LVIn_3 \cup LVIn_4 \\ LVIn_2 \\ \emptyset \\ LVOut_1 - \{x\} \\ LVOut_2 \cup \{y\} \\ LVOut_3 \cup \{x\} \\ LVOut_4 - \{x\} \end{pmatrix}$$

Then it is monotonically increasing.

Ascending Kleene chain

- Let us start with an ordered set $(L, \sqsubseteq)$ with a lower bound $\perp$ and a monotonically increasing function $F : L \rightarrow L$.
- Obviously, $\perp \sqsubseteq F(\perp)$, by definition of $\perp$.
- Moreover, since F is monotonic, $F(\perp) \sqsubseteq F(F(\perp))$.
- Moreover, since F is monotonic, $F(F(\perp)) \sqsubseteq F(F(F(\perp)))$.
- By successively applying F to the $\perp$ element, we obtain an ascending chain.

$$\perp \sqsubseteq F(\perp) \sqsubseteq F(F(\perp)) \sqsubseteq F(F(F(\perp))) \sqsubseteq \dots$$

- or, equivalently

$$\perp \sqsubseteq F(\perp) \sqsubseteq F^2(\perp) \sqsubseteq F^3(\perp) \sqsubseteq \dots \sqsubseteq F^n(\perp) \sqsubseteq \dots$$

Ascending Kleene chain

$$F\begin{pmatrix} LVOut_1 \\ LVOut_2 \\ LVOut_3 \\ LVOut_4 \\ LVIn_1 \\ LVIn_2 \\ LVIn_3 \\ LVIn_4 \end{pmatrix} = \begin{pmatrix} LVIn_2 \\ LVIn_3 \cup LVIn_4 \\ LVIn_2 \\ \emptyset \\ LVOut_1 - \{x\} \\ LVOut_2 \cup \{y\} \\ LVOut_3 \cup \{x\} \\ LVOut_4 - \{x\} \end{pmatrix}$$

We obtain the following chain:

$$\underbrace{\begin{pmatrix} \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \end{pmatrix}}_{\perp} \sqsubseteq \underbrace{\begin{pmatrix} \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \{y\} \\ \{x\} \\ \emptyset \end{pmatrix}}_{F(\perp)} \sqsubseteq \underbrace{\begin{pmatrix} \{y\} \\ \{x\} \\ \{y\} \\ \emptyset \\ \emptyset \\ \{y\} \\ \{x\} \\ \emptyset \end{pmatrix}}_{F^2(\perp)} \sqsubseteq \underbrace{\begin{pmatrix} \{y\} \\ \{x\} \\ \{y\} \\ \emptyset \\ \{y\} \\ \{x,y\} \\ \{x,y\} \\ \emptyset \end{pmatrix}}_{F^3(\perp)} \sqsubseteq \underbrace{\begin{pmatrix} \{x,y\} \\ \{x,y\} \\ \{x,y\} \\ \emptyset \\ \{y\} \\ \{x,y\} \\ \{x,y\} \\ \emptyset \end{pmatrix}}_{F^4(\perp)} \sqsubseteq \dots$$

Can this chain increase indefinitely?

- Therefore, our chain has the following behaviour:

$$\perp \sqsubset F(\perp) \sqsubset F^2(\perp) \sqsubset \ldots \sqsubset F^k(\perp) = F^{k+1}(\perp) = F^{k+2}(\perp) = \ldots$$

- We say that the chain **stabilizes** after the k-th iteration.
- Since $F^k(\perp) = F^{k+1}(\perp) = F(F^k(\perp))$, we get that $F^k(\perp)$ **is a fixed point** of F.

$$F\begin{pmatrix} LVOut_1 \\ LVOut_2 \\ LVOut_3 \\ LVOut_4 \\ LVIn_1 \\ LVIn_2 \\ LVIn_3 \\ LVIn_4 \end{pmatrix} = \begin{pmatrix} LVIn_2 \\ LVIn_3 \cup LVIn_4 \\ LVIn_2 \\ \emptyset \\ LVOut_1 - \{x\} \\ LVOut_2 \cup \{y\} \\ LVOut_3 \cup \{x\} \\ LVOut_4 - \{x\} \end{pmatrix}$$

We get the following chain:

$$\underbrace{\begin{pmatrix} \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \end{pmatrix}}_{\perp} \sqsubseteq \underbrace{\begin{pmatrix} \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \emptyset \\ \{y\} \\ \{x\} \\ \emptyset \end{pmatrix}}_{F(\perp)} \sqsubseteq \underbrace{\begin{pmatrix} \{y\} \\ \{x\} \\ \{y\} \\ \emptyset \\ \emptyset \\ \{y\} \\ \{x\} \\ \emptyset \end{pmatrix}}_{F^2(\perp)} \sqsubseteq \underbrace{\begin{pmatrix} \{y\} \\ \{x\} \\ \{y\} \\ \emptyset \\ \{y\} \\ \{x,y\} \\ \{x,y\} \\ \emptyset \end{pmatrix}}_{F^3(\perp)} \sqsubseteq \underbrace{\begin{pmatrix} \{x,y\} \\ \{x,y\} \\ \{x,y\} \\ \emptyset \\ \{y\} \\ \{x,y\} \\ \{x,y\} \\ \emptyset \end{pmatrix}}_{F^4(\perp)} = \underbrace{\begin{pmatrix} \{x,y\} \\ \{x,y\} \\ \{x,y\} \\ \emptyset \\ \{y\} \\ \{x,y\} \\ \{x,y\} \\ \emptyset \end{pmatrix}}_{F^5(\perp)}$$

Theorem
Assume a monotonically increasing function $F : L \to L$ and the following ascending Kleene chain:

$$\bot \sqsubseteq F(\bot) \sqsubseteq F^2(\bot) \sqsubseteq F^3(\bot) \sqsubseteq F^4(\bot) \sqsubseteq \dots$$

that stabilizes after the k-th iteration. Then:

- $F^k(\bot)$ *is a fixed point of* F.
- $F^k(\bot)$ ***is the least fixed point of*** F.

- Actually this result can be generalized to those cases in which L is infinite, and the chain does not stabilize.
- It requires stronger conditions on F (Scott-continuity).

- Assume that the variables in a system can take values from a **finite** poset.

1. Does F have a fixed point?
    - Yes, because the chain eventually stabilizes.
    - The chain will stabilize in a fixed point.
2. Could it have several fixed points?
    - Yes.
3. In case it has several fixed points, is there a fixed point that is better than the others?
    - The lower the fixed point, the more accurate it is, from the live variables analysis point-of-view.
4. In that case, how can we compute the best fixed point?
    - By applying F successively on $\perp$ until the chain stabilizes.

1. Does the system have a solution?
    - Yes, because the chain eventually stabilizes.
    - The chain will stabilize in a solution.
2. Could it have several solutions?
    - Yes.
3. In case it has several solutions, is there a solution that is better than the others?
    - The lower, the more accurate, from the live variables analysis point-of-view.
4. In that case, how can we compute the best solution?
    - By applying F successively on $\perp$ until the chain stabilizes.

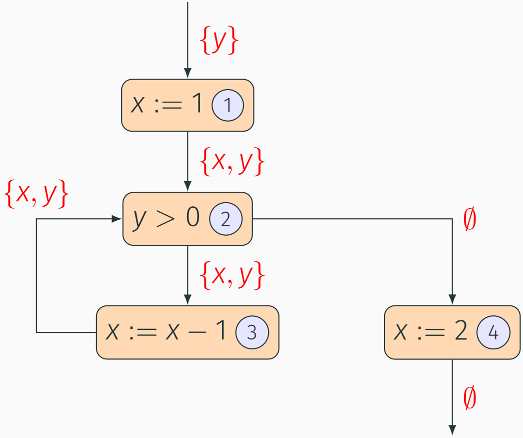
- In our example, the chain has stabilized at the following fixed point:

$$\underbrace{\begin{pmatrix} \{x,y\} \\ \{x,y\} \\ \{x,y\} \\ \emptyset \\ \{y\} \\ \{x,y\} \\ \{x,y\} \\ \emptyset \end{pmatrix}}_{F^5(\perp)}$$

which corresponds to the solution shown previously:

$$\begin{array}{llll} LVOut_1 = \{x,y\} & LVOut_2 = \{x,y\} & LVOut_3 = \{x,y\} & LVOut_4 = \emptyset \\ LVIn_1 = \{y\} & LVIn_2 = \{x,y\} & LVIn_3 = \{x,y\} & LVIn_4 = \emptyset \end{array}$$

Example: our result



Available expressions analysis

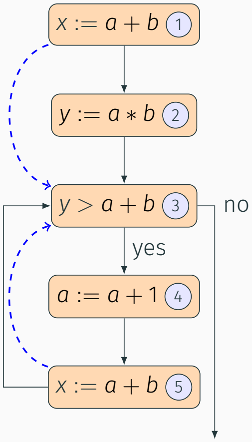
Available expressions analysis

- We say that an expression e is available at a given program point p if all paths from the beginning of the program to p compute this expression, and the computation is not followed by an assignment to one of the operands in the expression.
- In other words, it determines those expressions that have been computed previously and need not be recomputed at p.
- An available expressions analysis determines, for each program point, which expressions must be available at that point.

Why is it useful?

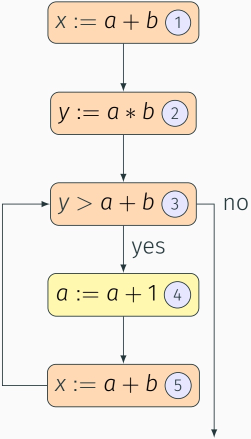
- Common subexpression elimination (CSE)
  - Af an expression is computed in a given block, but it is already available at the entry of that block, it does not have to be recomputed.
  - Redundant expressions are stored in a temporary variable, and they are replaced by that variable in when they are available.

Example



- Expression $a + b$ is available at block 3

Example

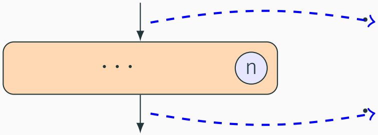


- Expression $a + b$ is available at block 3
- Expression $a + b$ is not available at block 5, since the value of a is changed before reaching block 5.

Available expressions analysis

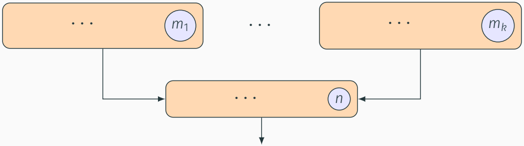
- We denote by **AExp** the set of nontrivial arithmetic expressions in the program.
  - Nontrivial = excluding variables and constants.
  - In our example: **AExp** $= \{a + b, a * b, a + 1\}$.

- The analysis computes, for each block, a pair of subsets of **AExp** which contain the available expressions at the entry and exit of the block, respectively.



- $AEIn_n$: Available expressions at block entry.
- $AEOut_n$: Available expressions at block exit.

- This time we will be doing a **forward analysis**.
  - An expression is available at the **entry** of a block if it is at the exit of **all** its predecessors.



$$AEIn_n = \bigcap_{m \in pred(n)} AEOut_m$$

- If n is the initial block, then $AEIn_n = \emptyset$.

Available expression analysis: data-flow equations

- An expression is available at the **exit** of a block if:
  - It is computed in that block, and none of its operands is modified in it, or
  - It was available at the beginning of the block, and none of its operators is modified in it.

Available expression analysis: data-flow equations

- For each block n, let us define $Kill_n$ as follows:
    - If the block contains an expression of the form $x := e$:

      $$Kill_n = \{e' \in \mathbf{AExp} \mid x \text{ appears in } e'\}$$

    - Otherwise, $Kill_n = \emptyset$.
- For each block n, let us define Gen_n as follows:
    - If n contains an expression of the form $x := e$:

      $$Gen_n = \{e' \in \mathbf{AExp} \mid e' \text{ is a subexpression of } e \text{ not containing } x\}$$

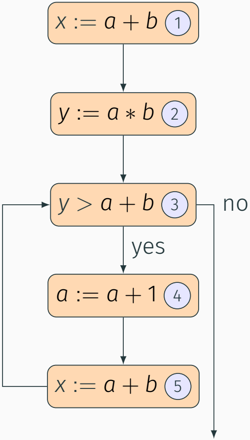
    - If n contains a boolean expression:

      $$Gen_n = \{e' \in \mathbf{AExp} \mid e' \text{ is a subexpression of } e\}$$

    - Otherwise, $Gen_n = \emptyset$.

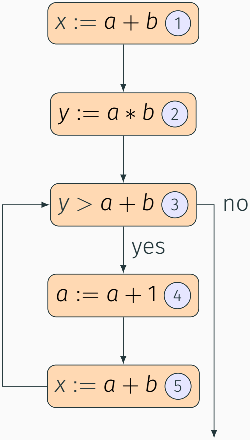
Then: $AEOut_n = (AEIn_n - Kill_n) \cup Gen_n$

Available expression analysis: data-flow equations



$$Kill_1 = \emptyset$$
$$Kill_2 = \emptyset$$
$$Kill_3 = \emptyset$$
$$Kill_4 = \{a + b, a * b, a + 1\}$$
$$Kill_5 = \emptyset$$

$$Gen_1 = \{a + b\}$$
$$Gen_2 = \{a * b\}$$
$$Gen_3 = \{a + b\}$$
$$Gen_4 = \emptyset$$
$$Gen_5 = \{a + b\}$$



$$AEIn_1 = \emptyset$$
$$AEIn_2 = AEOut_1$$
$$AEIn_3 = AEOut_2 \cap AEOut_5$$
$$AEIn_4 = AEOut_3$$
$$AEIn_5 = AEOut_4$$

$$AEOut_1 = AEIn_1 \cup \{a + b\}$$
$$AEOut_2 = AEIn_2 \cup \{a * b\}$$
$$AEOut_3 = AEIn_3 \cup \{a + b\}$$
$$AEOut_4 = AEIn_4 - \{a + b, a * b, a + 1\}$$
$$AEOut_5 = AEIn_5 \cup \{a + b\}$$

- The unknowns of the equation system take values in $\mathcal{P}(\mathsf{AExp})$.
- A solution to the system is a vector AE from $\mathcal{P}(\mathsf{AExp})^{10}$.
- Let us define an order relation $\sqsubseteq$ in the same way as in the previous analysis.

Assume we have two solutions AE and AE' such that $\mathsf{AE} \sqsubseteq \mathsf{AE}'$. Which one gives us more information on available expressions?

Solving the system of equations

- Let us define $F : \mathcal{P}(\mathsf{AExp})^{10} \to \mathcal{P}(\mathsf{AExp})^{10}$ as follows:

$$F\begin{pmatrix} AEIn_1 \\ AEIn_2 \\ AEIn_3 \\ AEIn_4 \\ AEIn_5 \\ AEOut_1 \\ AEOut_2 \\ AEOut_3 \\ AEOut_4 \\ AEOut_5 \end{pmatrix} = \begin{pmatrix} \emptyset \\ AEOut_1 \\ AEOut_2 \cap AEOut_5 \\ AEOut_3 \\ AEOut_4 \\ AEIn_1 \cup \{a+b\} \\ AEIn_2 \cup \{a*b\} \\ AEIn_3 \cup \{a+b\} \\ AEIn_4 - \{a+b, a*b, a+1\} \\ AEIn_5 \cup \{a+b\} \end{pmatrix}$$

- This is an monotonically increasing function.
- The difference with the previous analysis is that now we are looking for a greatest fixed point.

Theorem
Assume a monotonically increasing $F : L \to L$ and a Kleene descending chain

$$\top \sqsupseteq F(\top) \sqsupseteq F^2(\top) \sqsupseteq F^3(\top) \sqsupseteq F^4(\top) \sqsupseteq \dots$$

that stabilizes at k-th iteration. Then:

- $F^k(\top)$ *is a fixed point of F.*
- $F^k(\top)$ ***is greater than any other fixed point of*** F.

- Therefore, in our available expression analysis, we have to apply successively F starting from $\top$. In this case:

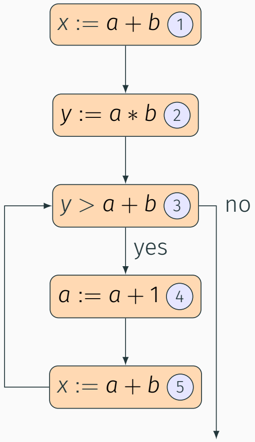
$$\top = (\underbrace{\mathsf{AExp}, \mathsf{AExp}, \dots, \mathsf{AExp}}_{\text{10 times}})$$

$$\begin{pmatrix} \{a+b, a*b, a+1\} \\ \{a+b, a*b, a+1\} \\ \{a+b, a*b, a+1\} \\ \{a+b, a*b, a+1\} \\ \{a+b, a*b, a+1\} \\ \{a+b, a*b, a+1\} \\ \{a+b, a*b, a+1\} \\ \{a+b, a*b, a+1\} \\ \{a+b, a*b, a+1\} \\ \{a+b, a*b, a+1\} \end{pmatrix} \to \begin{pmatrix} \emptyset \\ \{a+b, a*b, a+1\} \\ \{a+b, a*b, a+1\} \\ \{a+b, a*b, a+1\} \\ \{a+b, a*b, a+1\} \\ \{a+b, a*b, a+1\} \\ \{a+b, a*b, a+1\} \\ \{a+b, a*b, a+1\} \\ \emptyset \\ \{a+b, a*b, a+1\} \end{pmatrix} \to \begin{pmatrix} \emptyset \\ \{a+b, a*b, a+1\} \\ \{a+b, a*b, a+1\} \\ \{a+b, a*b, a+1\} \\ \emptyset \\ \{a+b\} \\ \{a+b, a*b, a+1\} \\ \{a+b, a*b, a+1\} \\ \emptyset \\ \{a+b, a*b, a+1\} \end{pmatrix} \to \cdots$$

$$\cdots \to \begin{pmatrix} \emptyset \\ \{a+b\} \\ \{a+b, a*b, a+1\} \\ \{a+b, a*b, a+1\} \\ \emptyset \\ \{a+b\} \\ \{a+b, a*b, a+1\} \\ \{a+b, a*b, a+1\} \\ \emptyset \\ \{a+b\} \end{pmatrix} \to \begin{pmatrix} \emptyset \\ \{a+b\} \\ \{a+b\} \\ \{a+b, a*b, a+1\} \\ \emptyset \\ \{a+b\} \\ \{a+b, a*b\} \\ \{a+b, a*b, a+1\} \\ \emptyset \\ \{a+b\} \end{pmatrix} \to \begin{pmatrix} \emptyset \\ \{a+b\} \\ \{a+b\} \\ \{a+b, a*b, a+1\} \\ \emptyset \\ \{a+b\} \\ \{a+b, a*b\} \\ \{a+b\} \\ \emptyset \\ \{a+b\} \end{pmatrix} \to \cdots$$

$$\cdots \to \begin{pmatrix} \emptyset \\ \{a+b\} \\ \{a+b\} \\ \{a+b\} \\ \emptyset \\ \{a+b\} \\ \{a+b, a*b\} \\ \{a+b\} \\ \emptyset \\ \{a+b\} \end{pmatrix} = \begin{pmatrix} \color{blue}\emptyset \\ \color{blue}\{a+b\} \\ \color{blue}\{a+b\} \\ \color{blue}\{a+b\} \\ \color{blue}\emptyset \\ \color{blue}\{a+b\} \\ \color{blue}\{a+b, a*b\} \\ \color{blue}\{a+b\} \\ \color{blue}\emptyset \\ \color{blue}\{a+b\} \end{pmatrix}$$

Analysis results



$$
\begin{aligned}
AEIn_1 &= \emptyset \\
AEOut_1 &= \{a + b\} \\
AEIn_2 &= \{a + b\} \\
AEOut_2 &= \{a + b, a * b\} \\
AEIn_3 &= \{a + b\} \\
AEOut_3 &= \{a + b\} \\
AEIn_4 &= \{a + b\} \\
AEOut_4 &= \emptyset \\
AEIn_5 &= \emptyset \\
AEOut_5 &= \{a + b\}
\end{aligned}
$$

Monotone frameworks

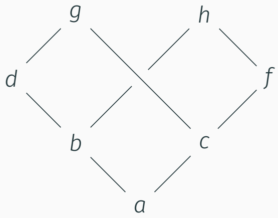
Our aim

- To find a general framework able to express a vast amount of data-flow analyses.
- The data-flow analyses seen so far compute some values In_n and Out_n for each program block.
- But the kind of values computed depends on the particular analysis:
  - Live variables: $\mathcal{P}(\mathbf{Var})$.
  - Available expressions: $\mathcal{P}(\mathbf{AExp})$
- Although both compute values in $\mathcal{P}(X)$ for some X, this is not always the case.
  - For example, a constant propagation analysis returns a function $\mathbf{Var} \to \mathbb{Z} \cup \{\bot, \top\}$ in each program point.

- In general, let us assume that In_n and Out_n take values in a set L.
- L is the **property space** of our analysis.
- In order to build an ascending or descending chain, L is required to have an order relation $\sqsubseteq$.
- L has to be a **lattice** so that the data-flow equations make sense.

Lattices: upper bounds and LUBs

- Assume an ordered set $(L, \sqsubseteq)$:

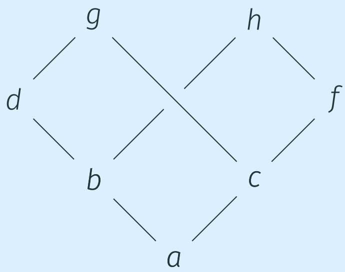


- Given a subset $L_0 \subseteq L$, an element $x \in L$ is an upper bound of L_0 if it is greater or equal than all the elements of L_0:

$$\forall y \in L_0 : \; y \sqsubseteq x$$

- The least upper bound (lub) of L_0, denoted by $\bigsqcup L_0$ is the lowest upper bound of all (if there is such lowest).

Assume the following ordered set L



- Upper bounds of $\{a, b, c\}$: $\bigsqcup\{a, b, c\}$
- Upper bounds of $\{a, c\}$: $\bigsqcup\{a, c\}$
- Upper bounds of $\{d, f\}$: $\bigsqcup\{d, f\}$

Lattices: lower bounds and GLBs

- Similarly, an element $x \in L$ is a **lower bound** of L_0 if it is lower or equal than all the elements of L_0.

$$\forall y \in L_0 : \; x \sqsubseteq y$$

- The **greatest lower bound** (glb) of $L_0 \subseteq L$, denoted by $\bigsqcap L_0$ is the greatest lower bound of all (if there is such).

- When L_0 has two elements, we use $x \sqcup y$ instead of $\bigsqcup\{x, y\}$ and $x \sqcap y$ instead of $\bigsqcap\{x, y\}$.
- $\sqcup$ and $\sqcap$ are commutative, associative and idempotent.

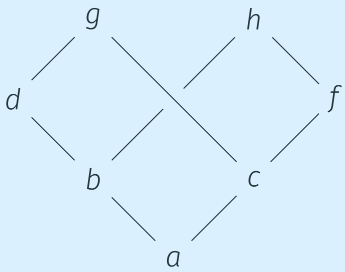
- A lattice is an ordered set $(L, \subseteq)$ such that every pair of elements has a glb and lub in L.

$$\forall x, y \in L : \; x \sqcup y \in L \quad \text{y} \quad x \sqcap y \in L$$

- A complete lattice is an ordered set $(L, \subseteq)$ in which every subset $L_0 \subseteq L$ has a glb and lub in L.

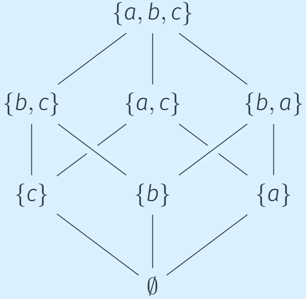
$$\forall L_0 \subseteq L. \; \bigsqcup L_0 \in L \quad \text{y} \quad \bigsqcap L_0 \in L$$

Is the following ordered set a lattice?



-

Is $(\mathcal{P}(\{a, b, c\}), \subseteq)$ a lattice? And a complete lattice?



-

- In general, for every set X, the ordered set $(\mathcal{P}(X), \subseteq)$ is a complete lattice.
- In this case, $\bigsqcup$ is set union ($\bigcup$) and $\sqcap$ is set intersection ($\bigcap$).

Is $(\mathbb{N}, \leq)$ a lattice? Is it a complete lattice?

What if we add a new element $+\infty$?

- Assume that $x \leq +\infty$ para todo $x \in \mathbb{N}^{\infty}$.

$$
\begin{tikzcd}
+\infty \arrow[d, dash, dashed] \\
2 \arrow[d, dash] \\
1 \arrow[d, dash] \\
0
\end{tikzcd}
$$

- Complete lattices are bounded sets. That is, they have $\bot$ and $\top$ elements.
- In fact:

$$\bot = \bigsqcap L \qquad \top = \bigsqcup L$$

- Given $(L_1, \sqsubseteq_1)$, $(L_2, \sqsubseteq_2)$, and an order relation $\sqsubseteq$ on $L_1 \times L_2$.

Theorem
If L_1 and L_2 are (complete) lattices, so is $L_1 \times L_2$.

- Let $\sqcup_1$ and $\sqcup_2$ be the lub operators in $(L_1, \sqsubseteq_1)$ and $(L_2, \sqsubseteq_2)$ respectively. Then, for every $(x_1, x_2), (y_1, y_2) \in L_1 \times L_2$:

$$(x_1, x_2) \sqcup (y_1, y_2) = (x_1 \sqcup_1 y_1, x_2 \sqcup_2 y_2)$$

- Similarly with greatest lower bounds:

$$(x_1, x_2) \sqcap (y_1, y_2) = (x_1 \sqcap_1 y_1, x_2 \sqcap_2 y_2)$$

- This can be extended to n-ary products: $L_1 \times L_2 \times \cdots \times L_n$:

$$(x_1, x_2, \ldots, x_n) \sqcup (y_1, y_2, \ldots, y_n) = (x_1 \sqcup_1 y_1, x_2 \sqcup_2 y_2, \ldots, x_n \sqcup_n y_n)$$

$$(x_1, x_2, \ldots, x_n) \sqcap (y_1, y_2, \ldots, y_n) = (x_1 \sqcap_1 y_1, x_2 \sqcap_2 y_2, \ldots, x_n \sqcap_n y_n)$$

  for every $(x_1, \ldots, x_n), (y_1, \ldots, y_n) \in L_1 \times \cdots \times L_n$.
- The $\bot$ element in $L_1 \times \cdots \times L_n$ is $(\bot_1, \ldots, \bot_n)$.
- The $\top$ element in $L_1 \times \cdots \times L_n$ is $(\top_1, \ldots, \top_n)$.

- Given a set S and an ordered set $(L_1, \sqsubseteq_1)$, we denote by $S \to L_1$ the set of total functions from S to L_1.
- Let us define a relation $\sqsubseteq$ in $S \to L_1$ as follows:
  - Given $f_1, f_2 : S \to L_1$, we say that $f_1 \sqsubseteq f_2$ if and only if:

$$\forall s \in S : \ f_1(s) \sqsubseteq_1 f_2(s)$$

Theorem

- *$\sqsubseteq$ is an order relation.*
- *If L_1 is a (complete) lattice, so is $(S \to L_1, \sqsubseteq)$.*

- If L_1 is a lattice we have, for any $s \in S$:

$$(f_1 \sqcup f_2)(s) = f_1(s) \sqcup_1 f_2(s)$$
$$(f_1 \sqcap f_2)(s) = f_1(s) \sqcap_1 f_2(s)$$

Which are the $\top$ and $\perp$ elements of $S \to L_1$?

-
-

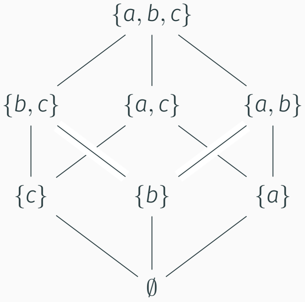
- Given an ordered set $(L, \sqsubseteq)$, let us consider the inverse relation $\sqsupseteq$.

Theorem

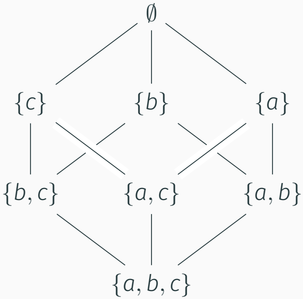
- *$\sqsupseteq$ is an order relation.*
- *If $(L, \sqsubseteq)$ is a complete lattice, so is $(L, \sqsupseteq)$.*

- If $(L, \sqsubseteq)$ is a lattice:
  - The $\sqcup$ operator in $(L, \sqsupseteq)$ is the $\sqcap$ operator in $(L, \sqsubseteq)$.
  - The $\sqcap$ operator in $(L, \sqsupseteq)$ is the $\sqcup$ operator in $(L, \sqsubseteq)$.
  - The $\bot$ element in $(L, \sqsupseteq)$ is the $\top$ element in $(L, \sqsubseteq)$.
  - The $\top$ element in $(L, \sqsupseteq)$ is the $\bot$ element in $(L, \sqsubseteq)$.

$(\mathcal{P}(\{a,b,c\}), \subseteq)$



$(\mathcal{P}(\{a,b,c\}), \supseteq)$



- Given a *poset* $(L, \sqsubseteq)$, an **ascending chain** (l_n) is a sequence $l_0, l_1, \ldots, l_n, \ldots$ of elements in L such that:

$$l_0 \sqsubseteq l_1 \sqsubseteq \cdots \sqsubseteq l_n \sqsubseteq \cdots$$

- We say that an ascending chain (l_n) **stabilizes** if all its elements are equal from a given one.

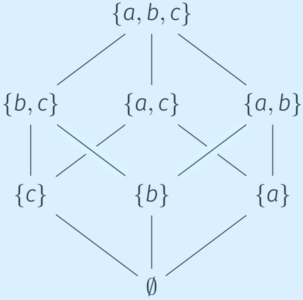
$$l_0 \sqsubset l_1 \sqsubset \cdots \sqsubset l_k = l_{k+1} = \cdots$$

- We say that a *poset* $(L, \sqsubseteq)$ satisfies the **ascending chain condition** if every possible chain in L stabilizes.

Does $(\mathbb{N}^\infty, \leq)$ satisfy the ascending chain condition?

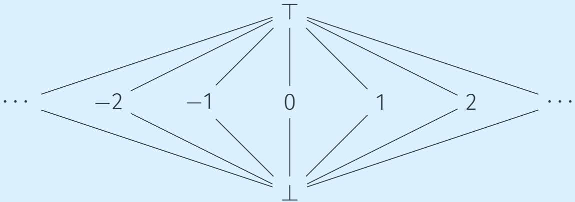
-

Does $(\mathcal{P}(\{a,b,c\}),\subseteq)$ satisfy the ascending chain condition?



-

Does the following lattice $(\mathbb{Z} \cup \{\bot, \top\}, \sqsubseteq)$ satisfy the ascending chain condition?



-

Theorem
If $L_1, L_2, \ldots, L_n$ satisfy the ascending chain condition, so does $L_1 \times \cdots \times L_n$.

Theorem
If L satisfies the ascending chain condition and S is finite, $S \to L$ also satisfies the ascending chain condition.

- A monotone framework is made up of:
  - A complete lattice L satisfying the ascending chain condition.
    - We denote by $\bigsqcup$ the lub operator associated with L.
  - A set $\mathcal{F}$ of monotonically increasing functions $L \to L$ (transfer functions) such that:
    - $\mathcal{F}$ contains the identity function.
    - $\mathcal{F}$ is closed under function composition:

$$\forall f, g \in \mathcal{F} :\ f \circ g \in \mathcal{F}$$

- Assume we want to define a control-flow analysis in a given program.
- An instance of a monotone framework is made up of:
  - A monotone framework $(L, \mathcal{F})$.
  - An extremal value, applied to the initial block of the program.
  - For every block n in the program, a transfer function $f_n \in \mathcal{F}$.

- From a instance in a monotone framework, we can define the equations of a forward analysis as follows:

$$In_n = \begin{cases} \iota & \text{if } n \text{ is the initial block} \\ \bigsqcup \{Out_m \mid m \in pred(n)\} & \text{otherwise} \end{cases}$$

$$Out_n = f_n\left(In_n\right)$$

- The equations of a backward analysis are defined as follows:

$$Out_n = \begin{cases} \iota & \text{if } n \text{ is a final block} \\ \bigsqcup \{In_m \mid m \in succ(n)\} & \text{otherwise} \end{cases}$$

$$In_n = f_n\,(Out_n)$$

- Some formalizations unify the equations of forward and backward analyses.
  - In order to perform a backward analysis, it is enough to reverse the direction of the arrows of the flow graph, and perform a forward analysis in the resulting program.

- Lattices used so far:

	Live. variables	*Reaching defs.*	*Available exps.*
L	$\mathcal{P}(\mathbf{Var})$	$\mathcal{P}(\mathbf{Var} \times \mathbf{Lab}^?)$	$\mathcal{P}(\mathbf{AExp})$
$\sqsubseteq$	$\subseteq$	$\subseteq$	$\supseteq$
$\bigsqcup$	$\bigcup$	$\bigcup$	$\bigcap$
$\bot$	$\emptyset$	$\emptyset$	$\mathbf{AExp}$
ι	$\emptyset$	$\{(x, ?) \mid x \in \mathbf{Var}\}$	$\emptyset$

- In these cases, the set $\mathcal{F}$ of transfer functions is made up of those functions $f : L \to L$ such that for every $l \in L$ there exist l_k and l_g such that:

$$f(l) = (l - l_k) \cup l_g$$

Bibliography

- F. Nielson, H.R. Nielson, C. Hankin
  Principles of Program Analysis
  Springer (1999)
  Chapter 2

- U.P. Khedker, A. Sanyal, B. Karkare
  Data Flow Analysis: Theory and Practice
  CRC Press (2009)