Abstract Interpretation

Welf Löwe
Welf.Lowe@lnu.se

Outline

- Summary of Data Flow Analysis (yesterday’s lecture)
- Problems left open
- Abstract interpretation idea

Complete Partial Order (CPO)

- Partially ordered sets \((U, \sqsubseteq)\) over a universe \(U\)
- Smallest element \(\bot\)
- Smallest element \(c_i\)
- Constructor for next element \(c_i = \text{next}(c_{i-1}, \ldots, c_0)\)
- Any (may be countable) ascending chain \(C \subseteq U\) has a unique largest element \(s\) larger than all chain elements \(c_i\)
- \(s\) called supremum \(s = \bigcup(C)\)
- Ascending chain property: any (may be countable) ascending chain \(C \subseteq U\) has an element \(c_i\) with
  * \(i\) is finite and
  * for all elements \(c_j < c_i\) and
  * for all elements \(c_k \geq c_i\),
  then \(c_i = \bigcup(C)\)
- Example: \((P^S, \subseteq)\) and \(C = \{\emptyset, \{1\}, \{1,2\}, \ldots\}\) then \(s = \bigcup(C) = N\) but ascending chain property does not hold

CPOs and Lattices

- Lattice \(L = (U, \sqsubseteq, \sqcup, \sqcap)\)
  * any two elements \(a, b\) of \(U\) have
    * an infimum \(a \sqcap b\) - unique largest smaller of \(a, b\)
    * a supremum \(a \sqcup b\) - unique smallest bigger of \(a, b\)
  * unique smallest element \(\bot\) (bottom)
  * unique largest element \(\top\) (top)
- A lattice \(L = (U, \sqsubseteq, \sqcup, \sqcap)\) defines two of CPOs \((U, \sqsubseteq)\)
  * upwards:
    * \(a \sqsubseteq b\) if \(a \sqcup b = b\), smallest \(\bot\)
    * ascending chain property holds \(c_i = \top\)
  * downwards:
    * \(b \sqsubseteq a\) if \(a \sqcap b = b\), smallest \(\top\)
    * ascending chain property holds \(c_i = \bot\)

Special lattices of importance

- Boolean Lattice over \(U = \{\text{true, false}\}\)
  * \(\bot = \text{false}, \top = \text{true, false} \sqsubseteq \text{true}, \text{true}(a,b) = a \lor b, \text{true}(a,b) = a \land b\)
- Bit Vector Lattice over \(U = \{\text{true, false}\}\)
- Power Set Lattice over \(2^S\) (set of all subsets of a set \(S\))
  * \(\bot = \emptyset, \top = S, C \sqsubseteq D \Rightarrow \text{true}(a,b) = a \lor b, \text{true}(a,b) = a \land b\) or the dual lattice
  * \(\bot = S, \top = \emptyset, C \sqsubseteq D \Rightarrow \text{true}(a,b) = a \land b, \text{true}(a,b) = a \lor b\)
Functions on CPOs

- Functions \( f: U \to U' \) (in the following we assume \( U = U' \))
- \( f \) monotone: \( x, y \in U \Rightarrow f(x) \subseteq f(y) \) with \( x, y \in U \)
- \( f \) continuous: \( f_K[u] = \bigcup \{ f(C) : C \subseteq f_K' \} \) with \( \bigcup \{ f(C) : C \subseteq f_K' \} = \emptyset \)
- \( f \) continuous \( \Rightarrow f \) monotone
- \( f \) monotone \( \Rightarrow U \) is finite \( \Rightarrow f \) continuous
- \( f \) monotone \( \times (U, \sqsubseteq) \) a CPO with ascending chain property \( \Rightarrow f \) continuous
- \( f \) monotone \( \times \{U, \sqsubseteq, \sqcap\} \) a lattice \( \Rightarrow f \) continuous

Fix Point Theorem (Knaster-Tarski)

For CPO \((U, \sqsubseteq)\) and monotone functions \( f: U \to U \)
- Minimum (or least or smallest) fix point \( x \) exists \( \Rightarrow X \) is unique
- \( X \) is iteratively computable

For CPO \((U, \sqsubseteq)\) with smallest element \( \bot \) and continuous functions \( f: U \to U \)
- Minimum fix point \( X = f^\omega(\bot) \)
- \( X \) is iteratively computable

If CPO \((U, \sqsubseteq)\) fulfills ascending chain property then \( X \) is computable effectively

Special cases:
- \((U, \sqsubseteq)\) with \( U \) finite,
- \((U, \sqsubseteq)\) defined by a lattice

Example

- \( U = \mathcal{P}(\mathbb{N}) \) (set of all subsets of Natural numbers \( \mathbb{N} \))
- Define:
  - \( f(\emptyset) = \emptyset \in U \) finite
  - \( f(\mathbb{N}) = \mathbb{N} \in U \) infinite
- \( f \) is monotone \( \Rightarrow f(u) \subseteq f(u') \), e.g.,
  - \( \emptyset \subseteq \{0\} \subseteq \{0,1\} \subseteq \ldots \Rightarrow f(\emptyset) \subseteq f(\{0\}) \subseteq f(\{0,1\}) \subseteq \ldots \)
- \( f \) is not continuous \( \Rightarrow f(\mathcal{C}) = \cup f(C) \), e.g.,
  - \( \mathcal{C} = \{\emptyset, \{0\}, \{0,1\}, \ldots\} \)
  - \( f(\mathcal{C}) = \cup f(C) = \cup \{f(\emptyset), f(\{0\}), f(\{0,1\}), \ldots\} = \emptyset \cup \{\emptyset, \{0\}, \{0,1\}, \ldots\} = \emptyset \)
  - \( f(\mathcal{C}) = \cup f(C) = \emptyset \)

4 DFA Equations Schemata

- forward and must: \( P_n(A) = \bigcap_{x \in X_n(A)} P_n(x) \)
- backward and must: \( P_n(A) = \bigcap_{x \in X_n(A)} P_n(x) \)
- forward and may: \( P_n(A) = \bigcup_{x \in X_n(A)} P_n(x) \)
- backward and may: \( P_n(A) = \bigcup_{x \in X_n(A)} P_n(x) \)

Monotone DFA Framework

- Solution of a set of DFA equations is a fix point computation
- Contribution of computational node \( x \) of kind \( k \) (Add, Load, Store, Call etc.) is modeled by monotone transfer function
- \( f^x: U \to U \), \( f^x \)
- Set \( F \) of transfer functions is closed under composition
- Contribution of predecessors \( P_n \) of \( x \) is modeled by supremum \( \cup \) of predecessor properties (successor \( \text{Succ} \), resp., for backward problems)
- Monotone DFA Framework: \((U, \sqsubseteq, F, I)\)
- \((U, \sqsubseteq)\) a CPO of analysis values fulfilling the ascending chain property
- \( F = \{ f^x : U \to U : f^x \subseteq U \} \) set of transfer functions (closed under composition, problem- and node-kind specific)
- \( I \subseteq U \) initial value for the start node's \( x \) property
- Analysis instance of a Monotone DFA Framework is given by a graph \( G \)
  - \( G = (X, E, s) \) control flow graph of a specific program, with
    - the start node \( s \in X \)

Solutions of Monotone DFA Framework

- Existence of the minimum fix point \( x \) is guaranteed, if domain \( U \) of properties \( P_i(A) \) is completely partially ordered \((U, \sqsubseteq)\) is a CPO
- Minimum fix point \( x \) is effectively computable if \((U, \sqsubseteq)\) additionally fulfills the ascending chain property


Proof (Sketch)

- Let \((U, \sqsubseteq, F, x)\) be a Monotone DFA Framework
  - \((U, \sqsubseteq)\) is a CPO of analysis values fulfilling the ascending chain property
  - \(F = \{f_i: U \to U, f_j: U \to U, \ldots\}\) set of transfer functions (closed under composition, problem- and node-kind specific)
  - \(x \in U\) initial value for the start node's \(x^i \in X\) property
- Let \(G = (N, E, a^i)\) be a control flow graph of an arbitrary program
  - Then the extension \(\{(N \times U \times U)^N, \sqsubseteq\}\) defines a CPO to:
    \[\text{let } a(i, j, k, a, b, u) \in (N \times U \times U)\]
    \[\text{let } m = [\{a^1, a^2, \ldots, a^m\}, \{b^1, b^2, \ldots, b^m\}, m, e] \in (N \times U \times U)^N\]
  - Smallest element is vector \([\{a^i, b^i, \ldots, \}, \{\}, \ldots\]\)
  - Data flow equations define monotone functions on \((N \times U \times U, \sqsubseteq)\)
    \[P_u(A) = \{x \in U| A(x) = \text{true} \}\]
    \[P_d(A) = \{x \in U| A(x) = \text{false} \}\]
- Fix Point Theorem (Knaster-Tarski) proofs the claim
- Forward – must problem
  \[\text{Supremum on lattice and inverse lattice, respectively}\]
  \[\text{Transfer function dependent on problem and statement kind}\]

Initialization

- Assume a power set lattice \(\mathcal{P}(\mathbb{S})\)
  - Initialization with the smallest element
  - Generalizations with \(\sqsubseteq\) for all but start node \(s^i\):
    \[\text{must}: \text{Initialization with } [\{s^1, s, S, S, \ldots S\}, \{\}, \ldots]\text{as universe of values } \mathbb{S}\text{ is the smallest element for each position}\]
    \[\text{may}: \text{Initialization with } [\{s^1, i, S, S, \ldots S\}, \{\}, \ldots]\text{as empty set } \mathbb{S}\text{ is the smallest element for each position}\]
    \[\text{Special (problem specific) initializations }\]
      \[\text{forward: } [\{s^1, i, \}, \ldots] \text{as general initialization not defined before the start node}\]
      \[\text{backward: } [\], \ldots, \{s^i, \}, \ldots] \text{as general initialization not defined after the end node}\]

Example I

- Property \(P, x = 1\) guaranteed?
  - Universe Boolean, CPO Boolean Lattice
  - Transfer functions: \true, \false, id
    \[\text{Statement } e; f_i = \true\]
    \[\text{Statement } R; f_i = \false\]
    \[\text{Statement } C; f_i = id\]
    \[\text{i.e. does not change}\]
  - Let \(P, P_0, P_1, P_2, P_3\) be values of \(P\)
    \[\text{after statements } A, B, C, (P_3)\]
    \[\text{and let } P, P_0, P_1, P_2 \text{ be values of } P\]
    \[\text{before statements } A, B, C, (P_3)\]
- Forward – must problem
  \[\text{it holds } P_0 = P, P_1 = P, P_2 \text{ before statements (assumption } s = 1)\]
  \[\text{Begin with } P_{\text{def}} = \false \text{ before statement } \text{if}\]
  \[\text{initialization } P_{\text{def}} = \false \text{ before statement } \text{if}\]
  \[\text{iteration provides fix point } P_0 = \false\]
  \[s = 1.1\text{ more difficult:}\]
    \[\text{Obvious transfer function not monotone}\]
    \[\text{Conservatively } s = 1\text{ is not guaranteed any more}\]

Example II

- Property \(P, x = 1\) possible?
  - Universe Boolean, CPO Boolean Lattice
  - Transfer functions identical
  \[\text{Forward – may problem}\]
    \[\text{Begin with } P_{\text{def}} = \false \text{ before statement } \text{if}\]
    \[\text{Initialization } P_{\text{def}} = \false \text{ before statement } \text{if}\]
    \[\text{Iteration provides fix point } P_0 = \false\]
  \[\text{Generalization:}\]
    \[\text{compute properties of several (all) variables in each step}\]
    \[\text{Property: are variables equal to a specific constant or are variables actually compile time constants at a certain program point}\]
    \[\text{ Universe: Bit vector with a vector element for each variable}\]
    \[\text{CPO induced by bit vector lattice}\]

What does Data Flow Analysis?
Path Graph

- For nodes $n \in N$ of $G=(N, E)$ define path graph $G'(n)=(N', E')$

  - For every path $\Pi$ ending in $n$:
    $$n' \in \Pi \iff n'' \in N$$
  $$n', n'' \in E' \iff (n', n'') \in \Pi$$

- The path graph acyclic by definition

- Since the set of paths to a node $n$ in $G$ is possibly countable (iff $G$ contains loops) the path graph $G'(n)$ is possibly countable

MFP vs. MOP

- Let a data flow problem $p_{\text{original}}$ be defined by a monotone DFA framework $(U, \subseteq, F, \alpha)$ and instance $G=(N, E, \beta)$

  - Minimum fix point $\text{MFP}$ computed by:
    - Initializing nodes in $G$ with the smallest $P'_n$ and $P_n$, respectively
    - Iteratively updating $P_n$ and $P'_n$ for the nodes of the instance graph $G$ using the transfer functions $f, f^+$
    - Until stabilized

- Let a data flow problem $p_{\text{original}}$ be defined by
  $$(U, \subseteq, F, \alpha)$$ and instance $G'(n)$ with $G'(n)$ the path graph of

  - Meet over all path $\text{MOP}$ of $p_{\text{original}}$ in node $n$ is defined
    - \& $P'_n$ for the nodes of the paths ending in node $n$; hence, $\text{MFP}$ of $p_{\text{original}}$ in node $n$.

- $\text{MFP}$ is the answer to the actual analysis questions: what holds for any execution path ending in $n$ (for any $u$)?

Example: Path Graph

- $\text{MFP}$ is $\text{MOP}$, if transfer functions $f$ distribute over $\cup$ in $U$, i.e., $f(u \cup b) = f(u) \cup f(b)$ which rarely holds.

- Otherwise, the $\text{MFP}$ is a conservative approximation of the $\text{MOP}$ ($\text{MOP} \subseteq \text{MFP}$)

  - Caution: It is not decidable if a path is actually executable; hence, $\text{MOP}$ may already be an approximation of the actual result

Example: $\text{MOP}(G) \subseteq \text{MFP}(G)$

- Constant propagation,
  - Universe is vectors, one entry for each variable $\in \{0, 1, \text{unknown}\}$

  $$G' = \begin{pmatrix}
  (1, 0, 1) & \text{u, u, u} \\
  (0, 1, 0) & \text{u, u, 0} \\
  (0, 1, 0) & \text{u, u, 0} \\
  (0, 0, 1) & \text{u, u, 0} \\
  \end{pmatrix}$$

  $$G = \begin{pmatrix}
  (0, 1, 0) & \text{u, u, u} \\
  (0, 1, 0) & \text{u, u, 0} \\
  (0, 1, 0) & \text{u, u, 0} \\
  (0, 0, 1) & \text{u, u, 0} \\
  \end{pmatrix}$$

- $(1, 0, 1) \cup (0, 1, 1) = (u, u, 1) \subset (u, u, u)$

Outline

- Summary of Data Flow Analysis (yesterday's lecture)
- Problems left open
- Abstract interpretation idea
Problems left open

- How to derive the transfer functions for a DFA
- How to make sure they compute the intended result, i.e.,
  - $MOP \subseteq MFP$?

Example: Reaching Definitions (RD)

- Which „Definitions“ (assignments) are guaranteed to be valid in node $N$?
- Universe: for all variables $v_i \in \{v_1, \ldots, v_n\}$ all definitions $\{d_{i,1}, \ldots, d_{i,k}\}$ are possible
- Forward, Must
  - Schema: $RD_{in}(A) = \bigcup RD_{in}(A) \cdot \text{kill}(A) \cdot \text{gen}(A)$
  - Initialization:
    - Universe, i.e. all definitions $(\{d_{i,1}, \ldots, d_{i,k}\})$ match each variable
    - Start node: no definition reaches, i.e. $RD_{in}(A) = \emptyset$ for all variables
  - A definition is generated $\text{gen}(A)$ by assignment $x_i := \text{expr}$
  - A definition $x_i := \text{expr}$ is removed $\text{kill}(A)$ by a new definition $x_i := \text{expr'}$

Example: MFP

$RD_{in}(A, x) = RD_{in}(A) \cdot \{M, A, B\} \cup \{M\} = \{A\}$
$RD_{out}(A, x) = RD_{out}(A) \cdot \{M, A, B\} \cup \{A\} = \{A\}$

Example: Run (for $x$)

$RD_{in}(M) = RD_{in}(M)$
$RD_{out}(M) = RD_{out}(M)$
$RD_{in}(A) = RD_{in}(A)$
$RD_{out}(A) = RD_{out}(A)$
$RD_{in}(N) = RD_{in}(N)$
$RD_{out}(N) = RD_{out}(N)$

Problems left open

- How to make sure RD computes the correct result?
- Exact result or a conservative approximation
- Actually, in the Reaching Definition example and the specific run it behave correctly:
  - Static analysis: $RD_{in}(N) = \emptyset$
  - Example run: $RD_{in}(M) = \{M\}$
    - $[M] \subseteq \emptyset$
      - Mind that RD was a must problem, ascending on the downwards CPO induced by the lattice power set lattice
      - Hence $\subseteq$ relation is set inclusion $\supseteq$ on the label sets

- How does this generalize?
  - For all runs and for all dataflow problems
  - We cannot test all (countable many) paths and all possible problem
Abstract Interpretation

- Relates semantics of a programming language to abstract analysis semantics
- Allows to compute or prove correct data flow equations (transfer functions)
  - Define abstraction of the execution semantics wrt. analysis problem
  - Define abstraction of execution traces to program points (in general, finite many contexts for each program point)
  - Prove that they are abstractions indeed.
- Idea even generalizes even to other than dataflow analyses as well (e.g., control flow analysis)

RD Execution Semantics

- Each program run is defined by a trace \( tr \in \text{Labels}^* \)
- Traces are defined by the programming language semantics, e.g.,
  - \( n[\text{assign}]@n[\text{assign}] \)
  - \( n[\text{if } expr \text{ then stats } \text{ else stats }] \)
  - \( n[\text{while } expr \text{ do stats } \text{ od }] \)
- Actual reaching definitions \( RD_{act} \) is a mapping \( RD_{act} : \text{Tr} \to \mathcal{P}_{\text{Labels}} \)
  - Basis for recursive definitions
  - \( RD_{act}(tr) = \emptyset \)
  - Analysis semantics of a trace \( tr \) expanded by the next dynamic step \( label \) is recursively defined on analysis semantics of trace \( tr \) and execution semantics of the static programming language construct of step \( label \)
  - \( RD_{act}(tr \oplus label : S) := \) \( \{ l | l : x:=expr' \in N \} \cup \{ label \} \)
- Solution
  - Define an abstraction \( \alpha \) of traces to make universe finite
  - Perform an abstract analysis on the abstraction of traces
  - Define a inverse concretization function \( \gamma \) to map results back to the semantic domain of the programming language
  - \( \alpha \) and \( \gamma \) should form a so called adjunction, or Galois connection, i.e.,
    - \( \alpha(x) \leq Y \Leftrightarrow x \subseteq \gamma(Y) \)
    - Mind the different domains of \( \alpha \) and \( \gamma \)
    - Consequently, there are different partial order relations \( \subseteq \)
      - \( \subseteq \) on semantics domain, countable and
      - \( \subseteq \) on analysis domain, finite
- This is one of our proof obligations to prove the analysis correct
Galois Connections

\[ \alpha(X) \leq Y \Leftrightarrow X \subseteq \gamma(Y) \]

Countable Semantic CPO \((U, \subseteq)\)

Finite Analysis CPO \((U', \leq)\)

Reaching Definitions \((\alpha)\)

- Let \(Tr_{\text{label}}\) be the set of all traces ending with program point label \(\text{label} \in \text{Label} \land \text{tr} \in \text{Tr}_{\text{label}}\) is an admissible trace of \(G\)
- We abstract a set \(RD_{\text{act}} \subseteq P^n\) with program point label \(\text{label} \in \text{Label}\)
- Let \(RD_{\text{act}}(\text{label})\) be the set of pairs of traces in \(Tr_{\text{label}}\) and their respective analysis results:
  \(RD_{\text{act}}(\text{label}) = \{ \text{tr} \to RD_{\text{act}}(\text{tr}) \mid \text{tr} \in Tr_{\text{label}} \}\)
- We abstract the actual analysis results \(RD_{\text{act}}\) of \(Tr_{\text{label}}\) with the union of all definitions reaching the end of any of the trace \(\text{tr} \in Tr_{\text{label}}\)
  \(RD_{\text{act}}(\text{tr}) = \text{label} \to \omega_{\text{tr} \in \text{Tr}_{\text{lab}}} RD_{\text{act}}(\text{tr})\)

Reaching Definitions \((\gamma)\)

- Conversely we concretize each program point label with the set of all traces ending in label
- The concretization function on labels is
  \(\gamma(\text{label}) = Tr_{\text{label}}\)
- Consequently we concretize the abstract analysis results \(RD(\text{label})\) of a program point label by assuming for any of the trace \(\text{tr} \in Tr_{\text{label}}\)
  \(RD(\text{tr}) = \{ \text{tr} \to RD(\text{tr}) \mid \text{tr} \in Tr_{\text{label}} \}\)

Reaching Definitions (revisited)

- To show (i): \((\alpha, \gamma)\) is a Galois connection (obvious)
- To show (ii): \(\alpha \ast RD_{\text{act}} \ast \gamma\) is approximated with \(RD\)
  i.e., \(\alpha \ast RD_{\text{act}} \ast \gamma \leq RD\)
- Proof (sketch): for each node \(n\) of \(G\)
  - By our definition of \(\gamma(\text{label}) = Tr_{\text{label}}\) of corresponds to path graph of \(G\) in \(n = (\text{label})\)
  - By our definition of \(RD_{\text{act}}\) \(\alpha \ast RD_{\text{act}} \ast \gamma\) in a node \(n\) is \(MFP\)
    of \(RD\) of path graph of \(G\) in \(n\)
  - \(MFP\) of \(RD\) of path graph of \(G\) in \(n\) is \(MOP\) of \(G\) in \(n\)
  - \(MOP \leq MFP\) of \(RD\)

How to perform the analysis?

- Take (any) abstract analysis \(F\) approximating \(\alpha \ast Act \ast \gamma : U' \to U'\) where \(Act\) is the actual semantics analysis function
- Analysis terminates if \((U', \leq)\) a CPO and \(F\) monotone
- Analysis is conservative if \(Act\) is monotone and \((\alpha, \gamma)\) a Galois connection
- Then conservative approximation computable by fix point iteration
- It holds for the minimum fix points \(MFP\):
  \(\alpha(MFP(Act)) \leq MFP(\alpha \ast Act \ast \gamma) \leq MFP(F)\)

General Proof Obligations

- To show (i): \((\alpha, \gamma)\) is a Galois connection
- To show (ii): \(\alpha \ast Act \ast \gamma\) is approximated with \(F\) i.e.,
  \(\alpha \ast Act \ast \gamma \leq F\)
- Proof (sketch): for each node \(n\) of \(G\)
  - By our definition of \(\gamma(\text{label}) = Tr_{\text{label}}\) of corresponds to path graph of \(G\) in \(n = (\text{label})\)
  - By our definition of \(Act\) and \(F\), \(\alpha \ast Act \ast \gamma(n) \subseteq P(n)\) in every node \(n\) (sufficient to show this for every \(f_k(n)\))
  - Then \(\alpha \ast Act \ast \gamma\) in a node \(n\) is \(MFP\) of \(RD\) of path graph of \(G\) in \(n\)
  - \(MFP\) of \(F\) of path graph of \(G\) in \(n\) is \(MOP\) of \(G\) in \(n\)
  - \(MOP \leq MFP\) of \(F\)