Data Flow Analysis and Abstract Interpretation

Welf Löwe
Welf.Lowe@lnu.se

1

Welf Löwe

- Professor in Computer Science at Linnaeus University (Sweden) + Postdoc Berkeley (USA) + PhD Karlsruhe (Germany) + MSc Dresden (Germany)
- Co-founder Softwerk AB, DueDive AB, Aimo GmbH
- Director of Research excellence center "Data Intensive Sciences & Applications" with an industry grad school "Data Intensive Applications"
- Current research interests: AI based software solutions
- 20+ years in compiler construction

- Grew up in Berlin + moved from Germany to Sweden in 2002 with three (meanwhile grown-up) children + married to a professor in German language and literature + TKD blackbelt + love to be out in the forests with my dog Maja

http://welf.se

2

Outline

Part 1: Data Flow Analysis and Abstract Interpretation
Part 2: Inter-procedural and Points-to analysis
Part 3: Static Single Assignment (SSA) form
Part 4: SSA based optimizations

3

3

Outline Part 1

- Summary of Data Flow Analysis
- Problems left open
- Abstract Interpretation idea

4

4

Complete Partial Order (CPO)

- Partially ordered sets $(U, \sqsubseteq)$ over a universe U
  - Smallest element $\perp \in U$
  - Partial order relation $\sqsubseteq$
- Ascending chain $C=[c_1,c_2,\ldots] \subseteq U$
  - Smallest element c_1
  - $c_{i-1} \sqsubseteq c_i$
  - Maybe finite or countable: constructor for next element $c_i = next([c_1,c_2,\ldots, c_{i-1}])$
- Unique largest element s of the chain $C=[c_1,c_2,\ldots]$
  - $c_i \sqsubseteq s$ (larger than all chain elements c_i)
  - s called supremum $s = \sqcup(C)$
- Ascending chain property of a universe U: any (may be countable) ascending chain $C \subseteq U$ has an element c_i with
  - i is finite and
  - for all elements $c_{<i} \sqsubseteq c_i$ and
  - for all elements $c_{>i} = c_i$, and hence $c_i = \sqcup(C)$
- Example: $(\mathcal{P}^{\mathcal{N}}, \subseteq)$ and $C=[\varnothing,\{1\},\{1,2\},\{1,2,3\},\ldots]$, $c_i = \{\max(c_{i-1})+1\} \cup c_{i-1}$
  $s = \cup(C) = \mathcal{N}$ but, the ascending chain property does not hold!

5

5

CPOs and Lattices

- Lattice $L = (U, \sqcup, \sqcap)$
  - any two elements a, b of U have
    - an infimum $\sqcap(a, b)$ - unique largest smaller of a, b
    - a supremum $\sqcup(a, b)$ - unique smallest bigger of a, b
  - unique smallest element $\perp$ (bottom)
  - unique largest element $\top$ (top)
- A lattice $L = (U, \sqcup, \sqcap)$ defines two CPOs $(U, \sqsubseteq)$
  - "upwards"
    - $a \sqsubseteq b \Leftrightarrow a \sqcup b = b$, smallest $\perp$,
    - If L finite heights $\Rightarrow$ ascending chain property holds ($c_i = \top$)
  - "downwards"
    - $b \sqsubseteq a \Leftrightarrow a \sqcup b = b$ ($\Leftrightarrow a \sqcap b = a$), smallest $\top$,
    - If L finite heights $\Rightarrow$ ascending chain property holds ($c_i = \perp$)

6

6

Special lattices of importance

- Boolean Lattice over $U=\{true, false\}$
  - $\bot = true$, $\top = false$, $true \sqsubseteq false$, $\bigsqcup(a,b)=a \vee b$, $\bigsqcap(a,b)=a \wedge b$
  - Finite heights
- Generalization: Bit Vector Lattice over $U = \{true, false\}^n$
  - Finite heights if n is finite
- Power Set Lattice $\mathcal{P}^S$ over S (set of all subsets of a set S)
  - $\bot = \emptyset$, $\top = S$, $\sqsubseteq\, = \,\subseteq$, $\bigsqcup(a,b)=a \bigcup b$, $\bigsqcap(a,b)=a \bigcap b$ or the dual lattice
  - $\bot = S$, $\top = \emptyset$, $\sqsubseteq\, = \,\supseteq$, $\bigsqcup(a,b)=a \bigcap b$, $\bigsqcap(a,b)=a \bigcup b$
  - Finite heights if S is finite

Functions on CPOs

- Functions f: $U \to U'$ (if not indicated otherwise, we assume $U = U'$)
- f monotone: $x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$ with $x, y \in U$
- f continuous: $f(\bigsqcup C) = \bigsqcup f(C)$ with $f(C) = f([x_1, x_2, \ldots]) = [f(x_1), f(x_2), \ldots]$
- f continuous $\Rightarrow f$ monotone,
- f monotone $\wedge$ $(U, \sqsubseteq)$ a CPO with ascending chain property $\Rightarrow f$ continuous
- f monotone $\wedge$ U is finite $\Rightarrow f$ continuous
- f monotone $\wedge$ $(U, \bigsqcup, \bigsqcap)$ a lattice with finite heights $\Rightarrow f$ continuous

Example

- Power Set Lattice $(\mathcal{P}^{\mathcal{N}}, \cup, \cap)$, U=set of all subsets of Natural numbers $\mathcal{N}$
- Define a (meaningless) function:
  - $f(u) = \emptyset \Leftrightarrow u \in U$ finite
  - $f(u) = \mathcal{N} \Leftrightarrow u \in U$ infinite
- f is monotone $u \subseteq u' \Rightarrow f(u) \subseteq f(u')$, e.g.,
  - $\emptyset \subseteq \{0\} \subseteq \{0,1\} \subseteq \ldots \Rightarrow f(\emptyset) \subseteq f(\{0\}) \subseteq f(\{0,1\}) \subseteq \ldots = \emptyset \subseteq \emptyset \subseteq \emptyset \subseteq \ldots$
- f is not continuous $f(\cup\, C) \neq \cup f(C)$, e.g.:
  - $C = [\emptyset, \{0\}, \{0,1\}, \ldots]$
  - $f(C) = [f(\emptyset), f(\{0\}), f(\{0,1\}), \ldots] = [\emptyset, \emptyset, \emptyset, \ldots]$
  - $\cup f(C) = \cup\, [f(\emptyset), f(\{0\}), f(\{0,1\}), \ldots] = \cup\, [\emptyset, \emptyset, \emptyset, \ldots] = \emptyset$
  - $\cup\, C = \cup\, [\emptyset, \{0\}, \{0,1\}, \ldots] = \mathcal{N}$
  - $f(\cup\, C) = f(\cup\, [\emptyset, \{0\}, \{0,1\}, \ldots]) = f(\mathcal{N}) = \mathcal{N}$
- Note: Power Set Lattice $(\mathcal{P}^{\mathcal{N}}, \cup, \cap)$ is not of finite heights and ascending chain property does not hold

Fixed Point Theorem (Knaster-Tarski)

Fixed point of a function: X with $f(X) = X$

For CPO $(U, \sqsubseteq)$ and monotone functions f: $U \to U$
- Minimum (or least or smallest) fixed point X exists
- X is unique

For CPO $(U, \sqsubseteq)$ with smallest element $\bot$ and continuous functions f: $U \to U$
- Minimum fixed point $X = \bigsqcup f^n(\bot)$
- X iteratively computable

CPO $(U, \sqsubseteq)$ fulfills ascending chain property $\Rightarrow X$ is computable effectively
Special cases:
- $(U, \sqsubseteq)$ with U finite,
- $(U, \sqsubseteq)$ defined by a finite heights lattice.

Monotone DFA Framework

- Solution of a set of DFA equations is a fix point computation
- Contribution of a computation A of kind K (*Alloc*, *Add*, *Load*, *Store*, *Call* …) is modeled by monotone transfer function
  - f_K: $U \to U$,
  - Define a set F of transfer functions closed under composition
  - Any composed transfer function is monotone as well
- Contribution of predecessor computations Pre of A is modeled by supremum $\bigsqcup$ of predecessor analysis values $P(Pre(A))$ (successor $Succ$, resp., for backward problems)
- Existence of the smallest fix point X is guaranteed, if domain U of analysis values $P(A)$ completely partially ordered $(U, \sqsubseteq)$
- It is efficiently computable if $(U, \sqsubseteq)$ additionally fulfills the ascending chain property

Monotone DFA Framework (cont'd)

- Monotone DFA Framework: $(U, \sqsubseteq, F, \iota)$
  - $(U, \sqsubseteq)$ a CPO of analysis values fulfilling the ascending chain property
  - $F = \{f_K: U \to U, f_{K'}: U \to U, \ldots\}$ set of monotone transfer functions (closed under composition, analysis problem specific)
  - $\iota \in U$ initial value (analysis problem-specific)
- Analysis instance of a Monotone DFA Framework is given by a graph G
  - $G = (N, E, n^1)$ data flow graph of a specific program, with
  - the start node $n^1 \in N$
- $((N \times U \times U)^{|N|}, \sqsubseteq_{\text{Vector}})$ defines a CPO:
  - Let $a=(i, x_{in}, x_{out})$, $b=(j, y_{in}, y_{out})$, $a, b \in (N \times U \times U)$
    $a \sqsubseteq_{\text{Triple}} b \Leftrightarrow i = j \wedge x_{in} \sqsubseteq y_{in} \wedge x_{out} \sqsubseteq y_{out}$
  - Let $m = [a^1, a^2, \ldots, a^{|N|}]$, $n = [b^1, b^2, \ldots, b^{|N|}]$, $m, n \in (N \times U \times U)^{|N|}$
    $m \sqsubseteq_{\text{Vector}} n \Leftrightarrow a^1 \sqsubseteq_{\text{Triple}} b^1 \wedge a^2 \sqsubseteq_{\text{Triple}} b^2 \wedge \ldots \wedge a^{|N|} \sqsubseteq_{\text{Triple}} b^{|N|}$
  - Smallest element is vector $[(n^1, \iota, \bot), (n^2, \bot, \bot), \ldots, (n^{|N|}, \bot, \bot)]$

Monotone DFA Framework (cont'd)

- Data flow equations define monotone functions in $(N \times U \times U, \sqsubseteq_{\text{Triple}})$:
  $P_{in}(A) = \bigsqcup_{X \in Pre(A)} (P_{out}(X))$
  $P_{out}(A) = f_{Kind(A)}(P_{in}(A))$ with $f_{Kind(A)} \in F$ transfer function of A
- Smallest fix point of this system of equations is efficiently computable since
  - $(N \times U \times U)$ and hence $(N \times U \times U)^{|N|}$ completely partially ordered and fulfill the ascending chain property
  - System of equations defines monotone function in $(N \times U \times U)^{|N|}$
- Data flow analysis algorithm:
  - Start with the smallest element: $[(n^1, \iota, \bot), (n^2, \bot, \bot), \ldots, (n^{|N|}, \bot, \bot)]$
  - Apply equations in any (fair) order
  - Until no $P_{in}(A)$ nor $P_{out}(A)$ changes

13

13

4 DFA Equations Schemata

sup

$f_K(P_{in}(A))$

- forward and must:
  $P_{in}(A) = \bigsqcap_{X \in Pre(A)} P_{out}(X)$
  $P_{out}(A) = P_{in}(A) - kill(A) \cup gen(A)$
- backward and must:
  $P_{out}(A) = \bigsqcap_{X \in Succ(A)} P_{in}(X)$
  $P_{in}(A) = P_{out}(A) - kill(A) \cup gen(A)$
- forward and may:
  $P_{in}(A) = \bigsqcup_{X \in Pre(A)} P_{out}(X)$
  $P_{out}(A) = P_{in}(A) - kill(A) \cup gen(A)$
- backward and may:
  $P_{out}(A) = \bigsqcup_{X \in Succ(A)} P_{in}(X)$
  $P_{in}(A) = P_{out}(A) - kill(A) \cup gen(A)$

14

14

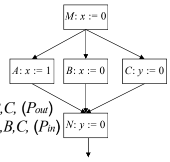
Initialization

- Assume a Power Set Lattice $\mathcal{P}^S$
- General initialization with the smallest element $\bot$ for all but start node n^1:
  - may: Initialization with $[(n^1, \iota, \emptyset), (n^2, \emptyset, \emptyset), \ldots, (n^{|N|}, \emptyset, \emptyset)]$ as empty set $\emptyset$ is the smallest element for each position
  - must: Initialization with $[(n^1, \iota, S), (n^2, S, S), \ldots, (n^{|N|}, S, S)]$ as universe of values S is the smallest element for each position in the inverse lattice
- Special (problem specific) initializations ι
  - forward: $[(n^1, \iota, \bot), \ldots]$, the general initialization ($\emptyset$ or S) is not defined before the start node
  - backward: $[\ldots, (n^e, \bot, \iota)]$, the general initialization ($\emptyset$ or S) is not defined after the end node

15

15

Example I

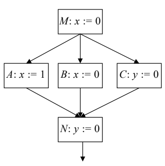


- Property P: $x = 1$ guaranteed?
- Universe Boolean, CPO Boolean Lattice
- Transfer functions: *true, false, id*
  - Statement A: $f_A = true$
  - Statement B: $f_B = false$
  - Statement C: $f_C = id$ i.e., does not change
- Let P_A, P_B, P_C, P_N be values of P after statements A,B,C, (P_{out})
- Let $\underline{P}_A$, $\underline{P}_B$, $\underline{P}_C$, $\underline{P}_N$ be values of P before statements A,B,C, (P_{in})
- Assume a forward – must problem
  - It holds $\underline{P}_N = P_A \wedge P_B \wedge P_C$
  - Begin with $\underline{P}_{A,B,C,N} = true$ before statements (assumption $x = 1$)
  - Initialization $\underline{P}_M = false$ before statement M is $x \neq 1$
  - Iteration leads to fixed point $P_N = false$
- $x := neg\ x$ more difficult:
  - Obviously, a naive transfer function for *neg* is not monotone
  - Conservative transfer function: $f = false$
  - Conservatively, $x = 1$ is not guaranteed any more by analysis in some cases where we (as humans) could see it holds

16

16

Example II

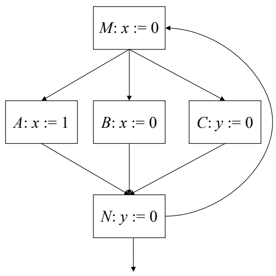


- Property P: $x = 1$ possible?
- Universe Boolean, CPO Boolean Lattice
- Transfer functions identical
- Forward – may problem
  - $\underline{P}_N = P_A \vee P_B \vee P_C$
  - Begin with $\underline{P}_{A,B,C,N} = false$ (assumption $x \neq 1$)
  - Initialization $\underline{P}_M = false$
  - Iteration leads to fixed point $P_N = true$
- Generalization:
  - Compute properties of several (all) variables in each step
  - Property: are variables equal to a specific constant or are variables actually compile time constants at a certain program point
  - Universe: Bit vector with a vector element for each variable
  - CPO induced by bit vector lattice

17

17

What does Data Flow Analysis?



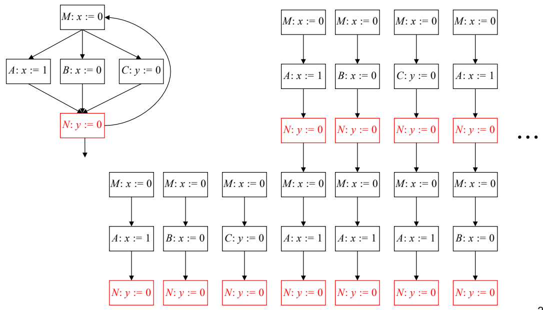
18

18

Path Graph

- For nodes $n \in N$ of $G{=}(N, E)$ define path graph $G'(n){=}(N', E')$ contains all paths Π ending in n:
  - $n' \in \Pi \Leftrightarrow n' \in N'$
  - $(n', n'') \in \Pi \Leftrightarrow (n', n'') \in E'$
- 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 graph $G'(n)$ is in general not finite

Example: Path Graph



MFP and *MOP*

For a monotone DFA problem (set of equations) $DFE = (U, \sqsubseteq, F, \iota)$ and G

- Define: Minimum Fixed Point *MFP* is computed by iteratively applying F beginning with the smallest element in U

Let $DFE'(n) = (U, \sqsubseteq F, \iota)$ and $G'(n)$ (same equations as DFE, applied to path graphs)

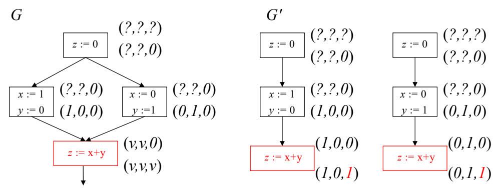
- Define: Meet Over all Paths *MOP* of DFE in (any arbitrary) node n is the supremum $\bigsqcup$ of minimum fix point *MFP* of $DFE'(n)$ in node n.
- *MFP* is equivalent with *MOP*, if f are distributive over $\bigsqcup$ in U (rarely).
- *MFP* is a conservative approximation of the *MOP* (otherwise).

Attention:

- It is not decidable if a path is actually executable
- Hence, *MOP* is already conservative approximation of the envisaged analysis result since, some paths may be not executable in any program run
- $MOP \neq MOEP$ (meet over all executable paths)

Example for $MFP(G) \neq MOP(G)$

Constant propagation: $(x,y,z) \in \{?, 0, 1, variable\}^3$



MFP: (v,v,v) MOP: $\bigsqcup((1,0,1), (0,1,1)) = (v,v,1)$

Errors due to our DFA Method

- Call Graphs:
  - Nodes – Procedures, Edges – calls
  - Only a conservative approximation of actually possible calls, some calls represented in the call graph might never occur in any program run
  - Allows impossible paths like call → procedure → another call
- Data flow graph of a procedure:
  - Nodes – Statements (Expressions), Edges – (syntactic or essential) dependencies between them
  - Application of a monotone DFA framework computes *MFP* not *MOP*

Outline

- Summary of Data Flow Analysis
- 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 approximates the intended question, and
  - $MOP \sqsubseteq MFP$?

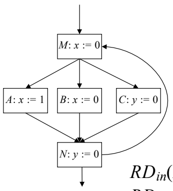
Example: Reaching Definitions (Must)

- Which set of definitions (assignments) reach (are valid in) a node A?
- Data flow values:
  - Subset of all definition (assignment) nodes $\{A_1 \ldots A_N\}$
  - Implementation: bit-vector $[\{false, true\}_1 \ldots \{false, true\}_N]$ where each position indicates if a node is in the subset
- We look at the forward - must version of the problem, hence:

$$RD_{in}(A) = \bigcap_{X \in Pre(A)} RD_{out}(X)$$
$$RD_{out}(A) = RD_{in}(A) - kill_{RD}(A) \cup gen_{RD}(A)$$

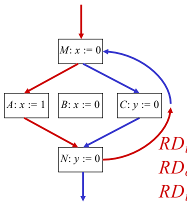
- Assume A contains assignment `x:=expr`, then
  - $gen_{RD}(A) = \{A\}$ and
  - $kill_{RD}(A) = \{A' \mid A'$ contains assignment `x:=expr'`$\}$
  - otherwise $gen_{RD}(A) = kill_{RD}(A) = \emptyset$
  - statically pre-calculated by checking the variables assigned in each node
- Initialization:
  - No definition reaches the start node:, i.e., $\iota = RD_{in}(A_1) = \emptyset$, but
  - All definitions reach each program point $RD_{in}(A_{i>1}) = RD_{out}(A_i) = \{A_1 \ldots A_N\}$

MFP



$RD_{in}(M)$	$= \emptyset \cap RD_{out}(N)$	$= \emptyset$
$RD_{out}(M)$	$= RD_{in}(M) - \{M,A,B\} \cup \{M\}$	$= \{M\}$
$RD_{in}(A)$	$= RD_{out}(M)$	$= \{M\}$
$RD_{out}(A)$	$= RD_{in}(A) - \{M,A,B\} \cup \{A\}$	$= \{A\}$
$RD_{in}(B)$	$= RD_{out}(M)$	$= \{M\}$
$RD_{out}(B)$	$= RD_{in}(B) - \{M,A,B\} \cup \{B\}$	$= \{B\}$
$RD_{in}(C)$	$= RD_{out}(M)$	$= \{M\}$
$RD_{out}(C)$	$= RD_{in}(C) - \{C,N\} \cup \{C\}$	$= \{M,C\}$
$RD_{in}(N)$	$= RD_{out}(A) \cap RD_{out}(B) \cap RD_{out}(C) = \emptyset$	
$RD_{out}(N)$	$= RD_{in}(N) - \{N,C\} \cup \{N\}$	$= \{N\}$

Example Run



$RD_{in}(M)$	$= \emptyset$	$= \emptyset$
$RD_{out}(M)$	$= RD_{in}(M) - \{M,A,B\} \cup \{M\}$	$= \{M\}$
$RD_{in}(A)$	$= RD_{out}(M)$	$= \{M\}$
$RD_{out}(A)$	$= RD_{in}(A) - \{M,A,B\} \cup \{A\}$	$= \{A\}$
$RD_{in}(N)$	$= RD_{out}(A)$	$= \{A\}$
$RD_{out}(N)$	$= RD_{in}(N) - \{C\} \cup \{N\}$	$= \{A,N\}$
$RD_{in}(M)$	$= RD_{out}(N)$	$= \{A,N\}$
$RD_{out}(M)$	$= RD_{in}(M) - \{M,A,B\} \cup \{M\}$	$= \{M,N\}$
$RD_{in}(C)$	$= RD_{out}(M)$	$= \{M,N\}$
$RD_{out}(C)$	$= RD_{in}(C) - \{N\} \cup \{C\}$	$= \{M,C\}$
$RD_{in}(N)$	$= RD_{out}(C)$	$= \{M,C\}$
$RD_{out}(N)$	$= RD_{in}(N) - \{C\} \cup \{N\}$	$= \{M,N\}$

Problem left open

- How to make sure RD computes the correct result?
  - As intended by the problem
  - Exact result or a conservative approximation
- Actually, in the example program and the specific run RD behaves correctly:
  - Static analysis: $RD_{out}(N) = \{N\}$
  - Example run: $RD_{out}(N) = \{A,N\}$, $RD_{out}(N) = \{M,N\}$
  - $\{A,N\} \sqsubseteq \{N\}$ and $\{M,N\} \sqsubseteq \{N\}$
    - Recall that RD was a must problem, ascending on the downwards CPO induced by the lattice power set lattice
    - Hence $\sqsubseteq$ relation is the inverse set inclusion $\supseteq$ on the label sets
- How does this generalize?
  - For all runs, all programs, and for all dataflow problems
  - We cannot test all (countable) paths of all (countable) programs and all (infinitely many) possible dataflow problems

Outline

- Summary of Data Flow Analysis
- Problems left open
- Abstract Interpretation idea

Abstract Interpretation Approach

- Relates semantics of a programming language
  - to a non-standard semantics defining the analysis question and
  - further to an abstract static analysis semantics that efficiently approximates a solution to this question
- Allows to compute or prove correct data flow equations (transfer functions)
- Idea even generalizes to other than dataflow analyses, as well (e.g., control flow analysis)
- Steps given the semantics of a programming languages:
  - Analysis question definition: Define an abstract execution semantics that correctly solves the analysis problem based on execution traces (in general, non-terminating as the traces may grow infinitely)
  - Analysis question solved with static analysis: Define a terminating abstraction of execution traces to (the finely many) program points (in general, maps infinitely many traces to a program point)
  - Show that they are correct abstractions indeed
  - Show that the static analysis terminates using the DFA framework

31

31

Program Traces

- Program traces are sequences of labels of statements
- Each program run corresponds to such a trace $tr \in Label^*$
- Program runs and, hence, traces are defined by the programming language semantics, e.g.,
  - tr[`stats; stat`] = tr[`stats`] $\oplus$ tr[`stat`]
  - tr[`assign`] := $label$(`assign`)
  - tr[`if expr then stats`$_1$ `else stats`$_2$] := $eval$[`expr`] = $true$? tr[`stats`$_1$] : tr[`stats`$_2$]
  - tr[`while expr do stats od`] := $eval$[`expr`] = $true$? tr[`stats`] $\oplus$ tr[`while … od`] : ε
- The actual program analysis questions, can be defined as a mapping Act: $Tr \rightarrow U$ of a trace to an analysis result
- E.g., the actual reaching definitions question RD_{act} can be defined as a mapping RD_{act}: $Tr \rightarrow \mathcal{P}^{Labels}$ i.e., for each trace (tr), what is the subset of definitions ($\subseteq \mathcal{P}^{Labels}$) that reaches the end of tr

32

32

Analysis Question Formalized

- Given a so-called standard semantics: a program's execution semantics is defined by the semantics of each programming (or intermediate) language computation statements and their composition in the program
  - Computation statements of kind K (*Alloc, Add, Load, Store, Call* …)
  - There are only finitely many such kinds
- The analysis question is formalized as a non-standard semantics
- Non-standard semantics: expected analysis results are defined for traces as an abstraction of the program's standard semantics wrt. the analysis problem
  - By giving each computation statements of kind K (*Alloc, Add, Load, Store, Call* …) a non-standard semantics answering that specific analysis question
  - Composed to an analysis execution semantics by/for each program

33

33

RD_{act} Execution Semantics

- Given a program $G = (N, E, n^1)$
- RD_{act} : $Tr \rightarrow \mathcal{P}^{Labels}$
- Basis for recursive definitions: empty trace
  - no definition reaches the end of the empty trace
  - $RD_{act}(\varepsilon) := \emptyset$
- Analysis execution semantics of $tr \oplus label$ (trace tr expanded by the next execution step $label$) is recursively defined on analysis execution semantics of trace tr and analysis execution semantics of the abstraction of the semantic of the computation (kind) at step $label$

if (S = "`x:=expr`") // computation kind is assignment to `x`
 $RD_{act}(tr \oplus label : S) := RD_{act}(tr) - \{l \mid (l : \texttt{x:=expr}') \in N\} \cup \{label\}$
else // any other computation kind
 $RD_{act}(tr \oplus label : S) := RD_{act}(tr)$

34

34

Observation

- Traces and semantics analysis values define a CPO $(U, \sqsubseteq)$
  - For RD_{act}, the universe U of analysis values can be defined by pairs of $Tr \rightarrow \mathcal{P}^{Labels}$
  - A partial order $\sqsubseteq$ can be defined as follows: elements are ordered iff
    - same program G, hence, $Labels$, and same traces
    - subset of $\mathcal{P}^{Labels}$
  - Smallest element $\varepsilon \rightarrow \emptyset$
- Universe U is not finite, since $Tr(G)$ is not
  - Even if the non-standard semantics (e.g., analysis function RD_{act}) was monotone, it is in general not continuous as universe not finite
- Then a solution to the analysis problem may exist, but cannot computed iteratively by applying the analysis function on the smallest element to fix point
  - Non-terminating program runs due to loops
  - Infinitely many possible different inputs that, in general, control the generation of traces and contribute to the analysis result

35

35

Solution

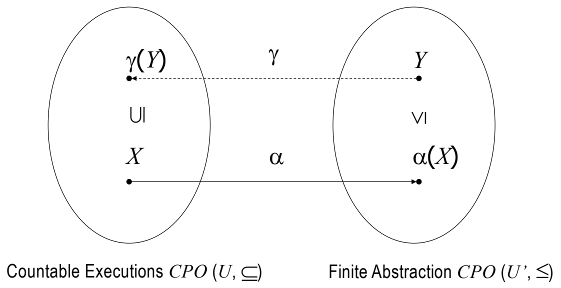
- Define an abstraction α of traces and analysis results to guarantee termination, e.g., by making universe U of analysis values finite
- Perform an abstract analysis on the abstraction of traces/values
- Define an inverse concretization function γ to map results back to the universe (traces and analysis results) of the analysis execution semantics
- α and γ should form a so-called adjunction, or Galois connection: $\alpha(X) \leq Y \Leftrightarrow X \subseteq \gamma(Y)$
  - Mind the different domains (universes) of α and γ
  - Consequently, there are different partial order relations
  - $\subseteq$ on non-standard analysis execution semantics domain, and
  - $\leq$ on abstract static analysis domain,
- Showing that abstraction and concretization form a Galois connection is one of our proof obligations to prove a static analysis correct

36

36

Galois Connections

$\alpha(X) \leq Y \Leftrightarrow X \subseteq \gamma(Y)$



37

How to define the static analysis?

- Choose an abstract analysis function F abstracting, i.e., giving larger or equal results than, $\alpha \bullet Act \bullet \gamma : U' \rightarrow U'$ where Act is the actual analysis execution semantics function
  - $\alpha \bullet Act \bullet \gamma : U' \rightarrow U'$ might be that function F
  - In general, function F requires a "widening", an explicit further abstraction of the results
- Analysis terminates if $(U', \leq)$ a finite CPO and F monotone
- Analysis is conservative if Act is monotone and (α, γ) a Galois connection
- Then conservative approximation is computable by fixed point iteration, and it holds for the minimum fix points MFP:

$$\alpha(MFP(Act)) \leq MFP(\alpha \bullet Act \bullet \gamma) \leq MFP(F)$$

38

38

Reaching Definitions (α)

- Let Tr_{label} be the set of all traces ending with program point $label$:
  $Tr_{label} = \{ tr \mid tr \in Tr \wedge tr = tr' \oplus label \}$
- We abstract a set $Tr_{label} \in \mathcal{P}^{Tr}$ with that program point $label \in Label$

  $\alpha: \mathcal{P}^{Tr} \rightarrow Label$
  $\alpha(Tr_{label}) = label$
- Concrete and abstract analysis value domains $\mathcal{P}^{Label}$ are the same:
  - Let $RD_{act}(tr' \oplus label) \in \mathcal{P}^{Label}$ be the set of definitions reaching the end $label$ of trace $tr' \oplus label$
  - Let $RD(label) \in \mathcal{P}^{Label}$ be the set of reaching definitions analyzed for the program point $label$
- We abstract the analysis execution semantics $RD(tr)$ of a trace $tr \in Tr_{label}$ with the abstract analysis results $RD(label)$ of the program point $label$
  $\alpha: \mathcal{P}^{Label} \rightarrow \mathcal{P}^{Label}$
  $\alpha(RD_{act}(tr)) = RD(label)$ *iff* $tr \in Tr_{label}$

39

39

Reaching Definitions (γ)

- Conversely, we concretize each program point $label$ with the set of all traces ending in $label$
- The concretization function on labels is
  $\gamma: Label \rightarrow \mathcal{P}^{Tr}$
  $\gamma(label) = Tr_{label}$
- Consequently, we concretize the abstract analysis results $RD(label)$ of a program point $label$ by assuming it is a conservative abstraction for any of the traces $tr \in Tr_{label}$:
  $\gamma: \mathcal{P}^{Label} \rightarrow \mathcal{P}^{Label}$
  $\gamma(RD(label)) = (tr \rightarrow RD(label)), \forall\, tr \in Tr_{label}$

40

40

RD Static Analysis Semantics

- Given a program $G = (N, E, n^1)$
- $RD: Label \rightarrow \mathcal{P}^{Labels}$
- Basis for recursive definitions:
  - Empty trace abstraction: starting point of the program n^1
  - no definition reaches n^1
  - $RD_{in}(n^1) := \varnothing$
- Static analysis semantics at $label$ is a conservative abstraction of $\alpha \bullet RD_{act} \bullet \gamma$
- It is recursively defined
  - on the static analysis result at the predecessors of $label$ (using the supremum) and
  - on and abstraction of the analysis execution semantics at the computation (kind) at $label$ (defining the transfer function)

41

41

RD Static Analysis Semantics

$\alpha \bullet RD_{act} \bullet \gamma$

$RD_{out}(label : S) :=$
  if (S = "`x:=expr`")
    $\bigcap_{(\ldots \oplus p \oplus label) \in Tr} : RD_{out}(p) - \{l \mid (l : \texttt{x:=expr}') \in N\} \cup \{label\}$
  else
    $\bigcap_{(\ldots \oplus p \oplus label) \in Tr} : RD_{out}(p)$

$\leq$ (more concrete than, abstracted by)

$RD_{in}(label : S) := \bigcap_{p \in \mathrm{Pre}(label)} RD_{out}(p)$

$RD_{out}(label : S) :=$
  if (S = "`x:=expr`")
    $RD_{in}(label) - \{l \mid (l : \texttt{x:=expr}') \in N\} \cup \{label\}$
  else
    $RD_{in}(label)$

42

42

Correctness of Analysis Abstraction

- By structural induction over all programs
- Compare analysis execution semantics and static analysis semantics (transfer functions) of program constructs
- Basis:
  - Claim holds for the empty trace: each program's starting point is abstracted correctly: $RD_{in}(n^1) = \emptyset$, $RD_{act}(\varepsilon) = \emptyset$
- Step:
  - Given a trace $tr \oplus label$ and its abstraction $label$
  - Provided $RD_{in}(label : S)$ is a correct abstraction of $RD_{act}(tr)$
  - Then $RD_{out}(label : S)$ is a correct abstraction of $RD_{act}(tr \oplus label)$:

    $\forall\ tr \in \gamma(label)$: $\alpha(RD_{act}(\gamma(RD_{in}(label)))) \leq RD_{out}(label)$
  - Distinguish cases of each program construct and the corresponding transfer function
  - Here trivial as RD_{act} and RD are identical (and monotone)

44

44

RD Proof of Correctness

- To show (i): (α, γ) is a Galois connection
- To show (ii): $\alpha \bullet RD_{act} \bullet \gamma$ is abstracted with RD i.e., $\alpha \bullet RD_{act} \bullet \gamma \leq RD$
- Proof (sketch): for each node n of G
  - By our definition of γ, $\gamma(label)=Tr_{label}$ corresponds to path graph of G in $n = (label{:}S)$
  - By our definition of RD_{act}, $RD_{act} \bullet \gamma$ in a node n is MFP of RD in the path graph of G in n
  - By our definition of α, $\alpha \bullet RD_{act} \bullet \gamma$ is the supremum of MFP of RD of the path graph of G in n
  - Hence, it is the MOP of RD in G in n
  - MOP of $RD \leq MFP$ of RD

45

45

General Proof Obligations

- To show (i): (α, γ) is a Galois connection
- To show (ii): $\alpha \bullet Act \bullet \gamma$ is abstracted with F i.e., $\alpha \bullet Act \bullet \gamma \leq F$
- Proof (sketch): for each node n of G
  - By our definition of γ, $\gamma(label)= Tr_{label}$ of corresponds to path graph of G in $n = (label{:}S)$
  - By our definition of Act and F, $\alpha \bullet Act \bullet \gamma(n) \leq F(n)$ in every node n (sufficient to show this for every $f_K(n)$)
  - Then $\alpha \bullet Act \bullet \gamma$ in a node n is MFP of F 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

46

46

Outline

Part 1: Data Flow Analysis and Abstract Interpretation

Part 2: Inter-procedural and Points-to analysis

Part 3: Static Single Assignment (SSA) form

Part 4: SSA based optimizations

47

47