

- Summary of Data Flow Analysis (previous lecture)
- Problems left open
- Abstract interpretation idea

Abstract Interpretation

Welf Löwe
Welf.Lowe@lnu.se

2

Complete Partial Order (CPO)

- Partially ordered sets (U, \sqsubseteq) over a universe U
 - Smallest element $\perp \in U$
 - Partial order relation \sqsubseteq
- Ascending chain $C = [c_1, c_2, \dots] \subseteq U$
 - Smallest element c_1
 - $c_i \sqsubseteq c_{i+1}$
 - Maybe finite or countable: constructor for next element $c_i = \text{next}([c_1, c_2, \dots, c_{i-1}])$
- Unique largest element s of chain $C = [c_1, c_2, \dots]$
 - $c_i \sqsubseteq s$ (larger than all chain elements c_i)
 - s called supremum $s = \sqcup(C)$
- 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 $c_j \sqsubseteq c_i$ and
 - for all elements $c_{j+1} = c_j$,
 then $c_i = \sqcup(C)$
- Example: $(\mathcal{P}^N, \subseteq)$ and $C = [\emptyset, \{1\}, \{1,2\}, \{1,2,3\}, \dots]$, $c_i = \{\max(c_{i-1})+1\} \cup c_{i-1}$
 $s = \cup(C) = \mathbb{N}$ but ascending chain property does not hold
 3

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 kinds of CPOs (U, \sqsubseteq)
 - upwards:
 - $a \sqsubseteq b \iff a \sqcup b = b$, smallest \perp ,
 - finite heights \implies ascending chain property holds ($c_i = \top$)
 - downwards:
 - $b \sqsubseteq a \iff a \sqcup b = b$ ($\iff a \sqcap b = a$), smallest \top ,
 - finite heights \implies ascending chain property holds ($c_i = \perp$)

4

Special lattices of importance

- Boolean Lattice over $U = \{true, false\}$
 - $\perp = true, \top = false, true \sqsubseteq false, \sqcup(a,b) = a \vee b, \sqcap(a,b) = a \wedge b$
 - 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)
 - $\perp = \emptyset, \top = S, \sqsubseteq = \subseteq, \sqcup(a,b) = a \cup b, \sqcap(a,b) = a \cap b$ or the dual lattice
 - $\perp = S, \top = \emptyset, \sqsubseteq = \supseteq, \sqcup(a,b) = a \cap b, \sqcap(a,b) = a \cup b$
 - Finite heights if S is finite

5

Functions on CPOs

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

6

Example

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

7

Fixed Point Theorem (Knaster-Tarski)

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

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

- For $CPO (U, \sqsubseteq)$ with smallest element \perp and **continuous** functions $f: U \rightarrow U$
- Minimum fixed point $X = \sqcup f^n(\perp)$
 - 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.

8

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**
 - $f_K: U \rightarrow U$,
 - Set F of transfer functions is closed under composition and (obviously) the composed functions are monotone as well
- Contribution of predecessors Pre of A is modeled by **supremum** \sqcup of predecessor properties (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

9

Monotone DFA Framework (cont'd)

- **Monotone DFA Framework:** $(U, \sqsubseteq, F, \iota)$
 - (U, \sqsubseteq) a CPO of analysis values fulfilling the ascending chain property
 - $F = \{f_K: U \rightarrow U, f_C: U \rightarrow U, \dots\}$ 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 $n^1 \in N$
- $((N \times U \times U)^{\mathbb{N}}, \sqsubseteq_{\text{Vector}})$ 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 $m = [a^1, a^2, \dots, a^{\mathbb{N}}], n = [b^1, b^2, \dots, b^{\mathbb{N}}], m, n \in (N \times U \times U)^{\mathbb{N}}$
 - $m \sqsubseteq_{\text{Vector}} n \Leftrightarrow a^1 \sqsubseteq_{\text{Triple}} b^1 \wedge a^2 \sqsubseteq_{\text{Triple}} b^2 \wedge \dots \wedge a^{\mathbb{N}} \sqsubseteq_{\text{Triple}} b^{\mathbb{N}}$
 - Smallest element is vector $[(n^1, \iota, \perp), (n^2, \perp, \perp), \dots, (n^{\mathbb{N}}, \perp, \perp)]$

10

Monotone DFA Framework (cont'd)

- Data flow equations define **monotone** functions in $(N \times U \times U, \sqsubseteq)$:
 $P_{in}(A) = \sqcup_{X \in Pre(A)} (P_{out}(X))$
 $P_{out}(A) = f_{Kind(A)}(P_{in}(A))$ 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 $(N \times U \times U)^{\mathbb{N}}$ completely partially ordered and fulfill the ascending chain property
 - System of equations defines monotone function in $(N \times U \times U)^{\mathbb{N}}$
- **Data flow analysis algorithm:**
 - Start with the smallest element: $[(n^1, \iota, \perp), (n^2, \perp, \perp), \dots, (n^{\mathbb{N}}, \perp, \perp)]$
 - Apply equations in any (fair) order
 - Until no $P_{in}(A)$ nor $P_{out}(A)$ changes

11

4 DFA Equations Schemata

- forward and must:

$$P_{in}(A) = \sup_{X \in Pre(A)} P_{out}(X)$$

$$P_{out}(A) = P_{in}(A) - kill(A) \cup gen(A)$$
- backward and must:

$$P_{out}(A) = \sqcup_{X \in Succ(A)} P_{in}(X)$$

$$P_{in}(A) = P_{out}(A) - kill(A) \cup gen(A)$$
- forward and may:

$$P_{in}(A) = \sqcup_{X \in Pre(A)} P_{out}(X)$$

$$P_{out}(A) = P_{in}(A) - kill(A) \cup gen(A)$$
- backward and may:

$$P_{out}(A) = \sqcup_{X \in Succ(A)} P_{in}(X)$$

$$P_{in}(A) = P_{out}(A) - kill(A) \cup gen(A)$$

12

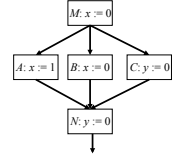
Initialization

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

13

Example I

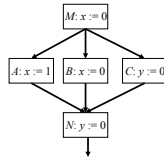
- Property P : $x = 1$ **guaranteed**?
- Universe Boolean, CPO Boolean Lattice
- Transfer functions: $true, false, id$
 - Statement A : $f_A = true$
 - Statement B : $f_B = false$
 - Statement C : $f_C = id$ i.e. does not change
- Let P_A, P_B, P_C, P_N be values of P after statements $A, B, C, (P_{end})$ and Let $\underline{P}_A, \underline{P}_B, \underline{P}_C, \underline{P}_N$ be values of P before statements $A, B, C, (P_{in})$
- 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 $\underline{P}_N = false$ before statement M is $x \neq 1$
 - Iteration leads to fixed point $P_N = false$
- $x := 1$ - **more difficult**:
 - Obviously, a naive transfer function *not* 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



14

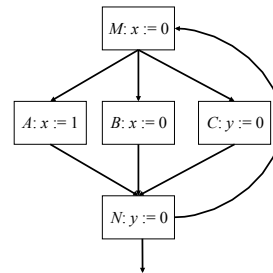
Example II

- 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 $\underline{P}_N = true$
 - 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



15

What does Data Flow Analysis?



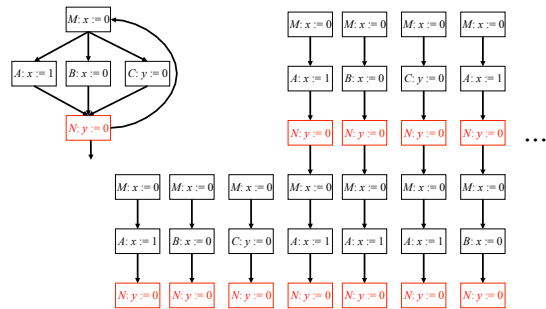
16

Path Graph

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

17

Example: Path Graph



18

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 minimum fix point MFP of $DFE'(n)$ in node n .
- MFP is equivalent with MOP , if f are distributive over \sqcap in U ,
- MFP is a conservative approximation of the MOP otherwise

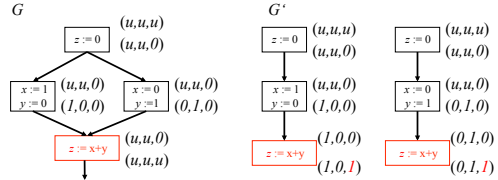
Attention:

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

19

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

Constant propagation: value vector: $(x,y,z) \in \{0,1,unknown\}^3$



20

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 \rightarrow procedure \rightarrow another call
- Data flow graph of a procedure:
 - Nodes – Statements (Expressions), Edges – (initial or essential) dependencies between them
 - Application of a monotone DFA framework computes MFP not MOP

21

Outline

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

22

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
 - $MOP \sqsubseteq MFP$?

23

Example: Reaching Definitions (Must)

- Which „Definitions“ (assignments) are guaranteed to reach a node A ?
- Solution: Subset of all nodes $\{A_1, \dots, A_N\}$ containing an assignment to a variable reaching a access from this variable in node A
- Universe: e.g. a bit-vector representation $\{ \{false, true\}_1 \dots \{false, true\}_N \}$ of all possible subsets of nodes for each variable
- Forward, Must
- Schema: $RD_{in}(A) = \bigcap_{X \in Pre(A)} RD_{out}(X)$
 $RD_{out}(A) = RD_{in}(A) - kill(A) \cup gen(A)$
- Initialization:
 - Universe, i.e. all definitions $\{A_1, \dots, A_N\}$ reach each program point
 - Start node: no definition reaches, i.e. $RD_{in}(A_1) = \emptyset$ for all variables
- A definition is generated ($gen(A)$) by assignment $x := expr$
- A definition is removed ($kill(A)$) by an assignment $x := expr'$ to the same variable in another node

24

Analysis Execution Semantics

- Non-standard semantics: actually expected analysis results are defined for traces as an abstraction of the program's execution semantics wrt. the analysis problem
- Standard semantics: a program's execution semantics is defined by the semantics of each programming language construct and their composition in the program
- There are only finitely many such constructs

31

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
 - $RD_{act}(\epsilon) := \emptyset$
- Analysis execution semantics of $tr \oplus label$ (trace tr expanded by the next dynamic step $label$)
 - recursively defined on analysis execution semantics of trace tr and analysis execution semantics of the static programming language construct of step $label$
 - $RD_{act}(tr \oplus label : S) :=$
 - if $S = "x := expr"$ then $RD_{act}(tr) \cup \{l : x := expr'\} \in N\} \cup \{label\}$
 - else $RD_{act}(tr)$

32

Observation

- Traces and semantics analysis values define a CPO (U, \sqsubseteq)
 - Universe U defined by $Tr \rightarrow \mathcal{P}Labels$
 - Partial order \sqsubseteq : same program G , hence $Labels$, and same traces and subset of $\mathcal{P}Labels$
 - Smallest element $\epsilon \rightarrow \emptyset$
- Universe U is countable not finite
- Even if the semantic analysis function (e.g., RD_{act}) is **monotone** it is in general **not continuous** as universe **not finite** since $Tr(G)$ is not
- Then a solution to the analysis problem may exist, but cannot be computed iteratively by applying the analysis function on the smallest element to fix point
 - Non-terminating program runs
 - Infinitely many different inputs

33

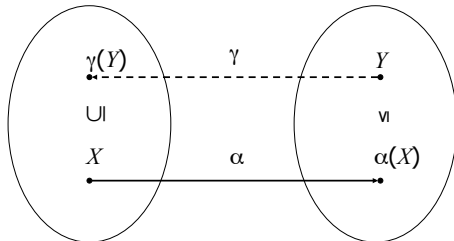
Solution

- Define an **abstraction** α of traces to make universe finite
- Perform an abstract analysis on the abstraction of traces
- Define an inverse **concretization** function γ to map results back to the semantic domain of the programming language
- α and γ 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 α and γ
 - Consequently, there are different partial order relations \sqsubseteq
 - \sqsubseteq on analysis execution semantics domain, countable and
 - \leq on abstract analysis domain, finite
- Showing that abstraction and concretization form a Galois connection is one of our proof obligations to prove the analysis correct

34

Galois Connections

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



Countable Execution CPO (U, \subseteq)

Finite Abstraction CPO (U', \leq)

36

How to define the analysis?

- Take (any) abstract analysis F function **abstracting** (i.e., **gives larger results that**) $\alpha \cdot Act \cdot \gamma : U' \rightarrow U'$ where Act is the actual analysis execution semantics function
- Analysis terminates if (U', \leq) a CPO and F monotone
- Analysis is conservative if Act is monotone and (α, γ) a Galois connection
- Then conservative approximation computable by fixed point iteration
- It holds for the minimum fix points MFP :
 - $\alpha(MFP(Act)) \leq MFP(\alpha \cdot Act \cdot \gamma) \leq MFP(F)$

Reaching Definitions (α)

- Let Tr_{label} be the set of all traces ending with program point $label$
 $Tr_{label} = \{ tr \oplus label \mid label \in Label \wedge tr \oplus label \text{ is an admissible trace of } G \}$
- 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 $label$
 - Let $RD(label) \in \mathcal{P}^{Label}$ be the set of reaching definitions analyzed for $label$
- We abstract the analysis execution semantics of the trace $tr \in Tr_{label}$:
 $RD(tr)$ 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 $tr \in Tr_{label}$

37

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 holds for any of the traces $tr \in Tr_{label}$:
 $\gamma: \mathcal{P}^{Label} \rightarrow \mathcal{P}^{Label}$
 $\gamma(RD(label)) = \bigvee tr \in Tr_{label} : RD(tr) = RD(label)$

38

RD 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
 - $RD_{in}(n^1) := \emptyset$
- Analysis semantics at $label$ which is actually $\alpha \cdot RD_{act} \cdot \gamma$
 - recursively defined on analysis semantics of abstractions of predecessor traces tr (predecessor labels)
 - $RD_{in}(label: S) := \bigcap_{p \in \text{pre}(label)} RD_{act}(p)$
 - analysis abstraction of the execution semantics of the static programming language construct of step $label$ (transfer function)
 - $RD_{out}(label: S) :=$
if $(S = "x := exp E")$ then
 $RD_{in}(label) - \{l \mid (l : x := exp E') \in N\} \cup \{label\}$
else
 $RD_{in}(label)$

39

Correctness of Analysis Abstraction

- Use structural induction over all programs
- Compare execution semantics and analysis semantics (transfer functions) of program constructs
- Basis:
 - Claim holds for the empty trace: each program's starting point is abstracted correctly: $RD_{in}(n^1) = \emptyset, RD_{act}(n^1) = \emptyset$
- 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 $RD_{act}(tr \oplus label)$:
 $\forall tr \in \gamma(label): \alpha(RD_{act}(\gamma(RD_{in}(label)))) \subseteq RD_{out}(label)$
 - Distinguish cases of each program construct and transfer function
 - Here trivial as RD_{act} and RD are identical (and monotone)

40

RD Proof Obligations

- To show (i): (α, γ) is a Galois connection (obvious)
- To show (ii): $\alpha \cdot RD_{act} \cdot \gamma$ is abstracted with RD i.e.,
 $\alpha \cdot RD_{act} \cdot \gamma \leq RD$
- 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 RD_{act} , $\alpha \cdot RD_{act} \cdot \gamma$ 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

41

General Proof Obligations

- To show (i): (α, γ) is a Galois connection
- To show (ii): $\alpha \cdot Act \cdot \gamma$ is abstracted with F i.e.,
 $\alpha \cdot Act \cdot \gamma \leq F$
- 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 , $\alpha \cdot Act \cdot \gamma(n) \subseteq F(n)$ in every node n (sufficient to show this for every $f_k(n)$)
 - Then $\alpha \cdot Act \cdot \gamma$ 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

42