Constraints

Express relationships between sets of facts at different points in the program

- Equational approach allowed *equalities* to be expressed
- We'll express sub-set (inclusion) relationships

Results look like a re-statement of the equational approach but methodology is different

- more than two classes of constraints (flow and transfer)
- for RD we have flow, generating and killing constraints

Reaching Definition Constraints

Constraints are introduced based on nature of data flow facts and the program statements

For assignments `[x := a]`^{l}

- $\mathsf{RD}_{exit}(l) \supseteq \mathsf{RD}_{entry}(l) \setminus \{(x, l') | l' \in \mathsf{Lab}_*\}$
- $\mathsf{RD}_{exit}(l) \supseteq \{(x, l)\}$

For other labeled statements `[...]`^{l}

- $\mathsf{RD}_{exit}(l) \supseteq \mathsf{RD}_{entry}(l)$

For all labeled statements, l, and their predecessors, l'

- $\mathsf{RD}_{entry}(l) \supseteq \mathsf{RD}_{exit}(l')$
- $\mathsf{RD}_{entry}(l) \supseteq \{(x, ?) | x \in \mathsf{Var}_*\}$ if l is the initial state

Constraint System Solutions

The fixed point of the RD equation system is a fixed point of the constraint system

- $\vec{\mathsf{RD}} = F(\vec{\mathsf{RD}}) \Rightarrow \vec{\mathsf{RD}} \sqsupseteq F(\vec{\mathsf{RD}})$

Fixed point of constraint system is least solution of equation system

- let $\vec{\mathsf{RD}}$ be a solution to the constraint system, i.e., $\vec{\mathsf{RD}} \sqsupseteq F(\vec{\mathsf{RD}})$
- $\vec{\emptyset} \sqsubseteq \vec{\mathsf{RD}}$
- then, $F(\vec{\emptyset}) \sqsubseteq F(\vec{\mathsf{RD}})$ (monotonicity)
- $F(\vec{\emptyset}) \sqsubseteq \vec{\mathsf{RD}}$
- $F(F(\vec{\emptyset})) \sqsubseteq F(\vec{\mathsf{RD}}) = \vec{\mathsf{RD}}$ (monotonicity)
- ...
- for some n, $F^n(\vec{\emptyset})) \sqsubseteq F(\vec{\mathsf{RD}}) = \vec{\mathsf{RD}}$ (monotonicity)

Type Systems

What is a type?

Type Systems

Usually types describe the possible values that a variable can take on during program execution?

We can use *annotated* types to characterize *meta* information about the program

- dead code
- which definitions reach a point
- potential data races

We'll read Freund & Flanagan's PLDI'00 paper later in the semester

Annotated Type System

The basic idea is to use types to encode properties of program states

- $S : \Sigma_1 \rightarrow \Sigma_2$

Properties, Σ_i, could be that

- a value is within a certain range
- an invariant holds

Effects System

Instead of talking about properties of states, we could annotate the transitions

- $S : \Sigma \xrightarrow{\varphi} \Sigma$

Effect annotations, φ, can encode

- the exceptions that might be raised
- information about the execution of the statement

Effects System from PPA section 1.6