Datalog

• Recall Datalog evaluation:
  • \( \text{Head}(x, y) \leftarrow \text{Body1}(x, y, z), \text{Body2}(z, y) \).  
  • Keep adding tuples matching head (monotonically) based on conjunction of body predicates
    • implemented by joining the database tables of body predicates
• Negation stratified
Consider a “next” relation on instructions

- \( \text{Next}(i, j) \)

Implement:

- \( \text{Reachable}(i, j) \)
- \( \text{ReachableBypassing}(i, j, k) \)
- \( \text{ReachableFromEntry}(i) \), assuming an \( \text{Entry}(i) \)
- \( \text{CanReachReturn}(i) \), assuming \( \text{ReturnInstruction}(i) \)

How about:

- \( \text{CanReachAllReturns}(i) \)
- \( \text{AllPredecessorsReachableFromEntry}(i) \)
Propositional Logic

• A language (framework) with:
  • propositions: P, Q, R, ...
  • logical connectives:
    • $\rightarrow$ (implies)
    • $\wedge$ (and)
    • $\vee$ (or)
    • $\neg$ (not)
    • $\leftrightarrow$ (equivalent/equivales)
  • constants: t, f
Propositional Logic Warmup

• What is the truth table of → ? Of ↔ ?
• Can derive all logical connectives from one of them and ¬
  • or all of them just from → and f
  • how?
• Basics: P → P ∨ Q, P ∧ Q → P
• Most important identity to remember:
  • P → Q ≡ ¬P ∨ Q
  • ≡ is the extra-logical “equivalent”, but ↔ also works
Other Useful Properties

- \( P \land (Q \lor R) = \)
- \( P \lor (Q \land R) = \)
- \( \neg (P \land Q) = \)
- \( \neg (P \lor Q) = \)
  - distributivity, DeMorgan

- Generally lots of cool properties
  - \( P \land Q \iff P \iff Q \iff P \lor Q \)
    - \( \iff \) associative, lower binding power
    - “Golden rule”
First-Order Logic
(aka first-order predicate/functional calculus)

- Another language framework with:
  - vars: x, y, …
  - predicates: P(x,…), Q(x,…), …
  - functions f(x,…), g(x,…)
  - logical connectives, constants as in propositional
  - quantifiers: ∀ (forall), ∃ (exists)

- Quantifiers introduce variable scopes
  - Example
    \[ ∀x,y,z: \text{Path}(x,y) \land \text{Path}(y,z) \rightarrow \text{Path}(x,z) \]
First-Order Logic Properties

- \((\forall x: F(x)) \rightarrow F(r)\)
  - \(F\) any formula, \(r\) replaces all occurrences of \(x\)
- \(F(r) \rightarrow (\exists x: F(x))\)
- \(\exists\) associates with \(\exists\), \(\forall\) with \(\forall\), but neither with each other
- Terms that do not reference the bound variable can move outside quantifier
- \(\forall\) is a big \(\wedge\): distributes over it
- \(\exists\) is a big \(\vee\): distributes over it
Properties and Exercises

- \((\forall x: P(x)) \leftrightarrow (\exists x: \neg P(x))\)
- \((\exists x: P(x)) \leftrightarrow (\forall x: \neg P(x))\)
- What happens with \(\rightarrow\)?
  - \((\forall x: P(x) \rightarrow Q(x)) \leftrightarrow ((\forall x: P(x)) \rightarrow (\forall x: Q(x)))\)
  - \((\exists x: P(x) \rightarrow Q(x)) \leftrightarrow ((\exists x: P(x)) \rightarrow (\exists x: Q(x)))\)
    - stronger, weaker, equivalent, or none?

- How about
  - \((\exists x: P(x) \rightarrow Q(x)) \leftrightarrow ((\forall x: P(x)) \rightarrow (\exists x: Q(x)))\)
Datalog and First-Order Logic

• These are exactly the logical properties we use to do forall emulations!
  • more complex for recursive relations—see code!
• Generally, relationship of Datalog to f.o. logic:
  • $P(x,y) \leftarrow Q(x,z), R(z,y)$
    means
    $$\forall x,y,z: Q(x,z) \land R(z,y) \rightarrow P(x,y)$$
    but also, if this is the only rule deriving $P$,
    $$\forall x,y: \exists z: P(x,y) \rightarrow Q(x,z) \land R(z,y)$$
• What if there are other rules deriving $P$?
Datalog Exercise

• We saw forall emulations (CanReachAllReturns(i))

• Let’s see a more complex one:
  • consider a flow-sensitive VarPointsTo relation:
    • VarPointsTo(instr, var, heap)
  • write the logical rule “a variable points to an abstract object at instruction \( i \), if it points to that same object at all predecessors of \( i \)”
    • in practice there will need to be more conditions, e.g., that \( i \) doesn’t assign the variable, but that’s easy
More Datalog Exercises

- Consider an intermediate language represented as Datalog relations
  - Instruction(method_name, i_counter, instruction)
  - Var(method_name, variable)
  - Next(method_name, i_counter, j_counter)
  - VarMove(method_name, i_counter, var1, var2)
  - ConstMove(method_name, i_counter, variable, const)
  - VarUse(method_name, i_counter, variable)
  - VarDef(method_name, i_counter, variable)

- Compute live ranges, basic blocks, constant propagation, copy propagation
  - A variable is live from the point of its use all the way back to the point of its last def