Datalog + Logic Tutorial

European Research Council
Established by the European Commission

European Union
European Social Fund
OPERATIONAL PROGRAMME
EDUCATION AND LIFELONG LEARNING
investing in knowledge society
MINISTRY OF EDUCATION & RELIGIOUS AFFAIRS, CULTURE & SPORTS
MANAGING AUTHORITY
NSRF 2007-2013
programme for development
EUROPEAN SOCIAL FUND
Co-financed by Greece and the European Union

Datalog

- Recall Datalog evaluation:
  - `Head(x,y) <- Body1(x,y,z), 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

Datalog Exercises

- Consider a “next” relation on instructions
  - `Next(i, j)`
- Implement:
  - `Reachable(i,j)`
  - `ReachableBypassing(i,j,k)`
  - `ReachableFromEntry(i)`, assuming an `Entry(i)`
  - `CanReachReturn(i)`, assuming `ReturnInstruction(i)`
- How about:
  - `CanReachAllReturns(i)`
  - `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: $P \rightarrow P \vee Q$, $P \wedge Q \rightarrow P$
- Most important identity to remember:
  - $P \rightarrow Q \equiv \neg P \vee Q$
    - ≡ is the extra-logical "equivalent", but ↔ also works

Other Useful Properties

- $P \wedge (Q \vee R) =$
- $P \vee (Q \wedge R) =$
- $\neg (P \wedge Q) =$
- $\neg (P \vee Q) =$
  - distributivity, DeMorgan
- Generally lots of cool properties
  - $P \wedge Q \leftrightarrow P \leftrightarrow Q \leftrightarrow P \vee Q$
    - $\leftrightarrow$ 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
    $\forall x,y,z$: Path(x,y) $\wedge$ Path(y,z) $\rightarrow$ 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

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

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) <- Q(x,z), R(z,y)
    means
    $\forall x,y,z: Q(x,z) \wedge R(z,y) \rightarrow P(x,y)$
    but also, if this is the only rule deriving P,
    $\forall x,y: \exists z: P(x,y) \rightarrow Q(x,z) \wedge R(z,y)$
  - 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