COS320 Dataflow Optimization III

What lies ahead

next week: register allocation, Control flow
next next week: loop optimizations, high-level languages I
next^3 week: high-level languages II, wrap-up.

Dataflow Analysis, Cont.

Recall generic worklist algorithm

Input:
- Abstract domain L (space of program properties, partial order)
  $(L, \sqsubseteq, \sqcup, \bot, \top)$
- transfer function post: $BB \times L \to L$

Output: least annotation IN, OUT

$IN[s] = \top$
$\forall n \in N \quad post(n, IN(n)) \sqsubseteq OUT[n]$
$\forall (p \to n) \in E \quad OUT[p] \sqsubseteq IN(n)$

Algorithm:

Start w/ least annotation satisfying first constraint.

$IN(s) = \top, \quad OUT(s) = \bot;$
$IN(n) = OUT(n) = \bot;$
$work = N;$
while $work \neq \emptyset$ do
  Pick n from work;
  $work = work \setminus \{n\}; \quad old = OUT(n);$
  $IN(n) = IN(n) \sqcup \bigsqcup_{p \to n \in E} OUT(p);$
  $OUT(n) = post(n, IN(n))$
  if $(old \neq OUT(n))$ update $work = work \cup succ(n)$

return IN, OUT.

Partial Correctness

Invariants

1) $IN[s] = T$

Pf by Contrapositive

2) $\forall n \in N$ $post_c(n, IN[n]) \not\sqsubseteq OUT[n]$ $\Longrightarrow$ $n \in$ work

3) $\forall p \to n \in E$ $OUT[p] \not\sqsubseteq IN[n]$ $\Longrightarrow$ $n \in$ work

☆ Property: Annotations ~~strict~~ increasing.
→ Can't ever decrease ($In(n) = In(n) \sqcup_{p \to n \in E} OUT(p)$)

② is true initially true. Also true at the _end_ of each loop iteration: invariant is maintained at end of loop.

③ is initially true. $IN[n] \leftarrow IN[n] \sqcup \bigsqcup_{p \to n} OUT(p)$ maintains this, but $OUT[n] = Post_c(n, IN[n])$, might break constraints of type (n, p).

edge ... src

But then we add these constraints that might be broken back to the worklist.

Proof of optimality

(Induction)

Claim Worklist algorithm gives least solution.

Pf Let $\overline{IN}$, $\overline{OUT}$ be any upper bound. We prove that at every step, $IN \sqsubseteq^* \overline{IN}$, $OUT \sqsubseteq^* \overline{OUT}$.

- $\sqsubseteq^*$ is a pointwise order on function space $N \to L$

- Invariant holds initially; we sends $IN[\varepsilon] \to \top$ and everything to $\bot$. Thing smaller holds.

Argument: Let IN_i OUT_i be the sets in ith iteration

Invariant I

$$\underbrace{IN_{i+1}[n_i] = IN_i[n_i] \sqcup \bigsqcup_{(p,n)} OUT_i[p_i]}_{\sqsubseteq\, IN_i[n_i] \,\sqcup\, \underbrace{\bigsqcup_{p,n_i} \overline{OUT[p]} \sqsubseteq \overline{IN[n_i]}}_{}}$$

Pf Show that $\overline{IN}[n_i] \sqsupseteq \overline{OUT[p]}$ for $p \to n_i$.
Why? by properties of Solution: $\overline{IN}(n_i) \sqsupseteq \overline{OUT}(p)$ since we have to satisfy edge constraints

Invariant II

$$\underbrace{OUT_{i+1}(n_i) = Post_L(n_i, IN_{i+1}[n_i])}_{\sqsubseteq\, Post_L(n_i, \overline{IN}[n_i]) \,\sqsubseteq\, \overline{OUT}[n_i]}$$

Inductive hypothesis $OUT_i(n_i) \sqsubseteq \overline{OUT_i}(n_i)$ $IN_i(n_i) \sqsubseteq \overline{IN_i}(n_i)$

for next step on proving $IN_{i+1}[n_i]$

Invariant II :

by monotonicity of POST, since $IN_i[n_i] \sqsubseteq \overline{IN}_{i+1}[n_{i+1}]$

$Post(n_i, IN_{i+1}[n_i]) \sqsubseteq Post(n_i, \overline{IN}_i[n_i])$

by def, $Post(\ldots) \sqsubseteq \overline{OUT}[n_{i+1}]$

Termination

Ascending chain condition

A Poset $\sqsubseteq$ satisfies this condition if any ~~as~~- ascending sequence

$$x_1 \sqsubseteq x_2 \sqsubseteq x_3 \sqsubseteq \ldots$$

eventually stabilizes: $\exists\, i$ such that $x_j = x_i$ & for $j > i$

ex: X is finite $\Rightarrow$ $(2^X, \subseteq)$ and $(2^X, \supseteq)$ satisfies the ACC.

For available expressions X is set of expressions in program.

Reason: Every time I go up the chain I increase the cardinality of set but I can only do so n times.

Another ex if X is finite and $(L, \sqsubseteq)$ satisfies Acc
then $(X \to L, \sqsubseteq^*)$ also satisfies ACC.

Const prop. : Asc. C. C. satisfied because can't go up more than twice.

$\Longrightarrow$ Function space also satisfies ACC.

Argument : Each chain stabilized $\to$ Chain of functions stabilized.

Termination argument

if $(L, \sqsubseteq)$ satisfies Acc, so does $(N \to L, \underbrace{\sqsubseteq^*}_{\text{space of annotations}})$

so $OUT_0 \sqsubseteq OUT_1 \sqsubseteq^* \ldots OUT_n$ Assume Algorithm doesn't terminate $\Longrightarrow$ Chain doesn't stabilize
$\Longrightarrow$ algorithm terminates.

Compilers: Reverse postorder (fact)
Verification: ~~Weak~~ topological order. (gives more info about loop structure).

Local vs. Global constraints

Two specifications for available expressions:

Global: e available at entry of n

$\Updownarrow$

for every path from s to n in G
1. expr e is evaluated along each path
2. after last evaluation no vars are overwritten.

Why are they equivalent?

local: ae smallest function s.t.

$ae(s) = \emptyset$

$\forall\, p \to n,\ post(p, ae(p)) \supseteq ae(n)$

Coincidence

"Global": Join over paths:

$$\mathrm{JOP}[n] = \bigsqcup_{\pi \in \mathrm{Path}(s,n)} \mathrm{post}_L(\pi, T)$$

Extend post_L to Paths by taking

$$\mathrm{post}(n_1 \dots n_k, T) = \mathrm{post}(n_{k-1} \cdots \mathrm{post}_k(n_1, T))$$

Coincidence thm (Kildall-Kam-Ullman)

If post is a distributive function then $\mathrm{JOP}[n] = \mathrm{N}[n]$ for any abstract domain.

Caveat. Reason in terms of global properties, but it translates into a local constraint.

Post distributivity

☆ $\mathrm{post}(n, x \sqcup y) = \mathrm{post}(n, x) \sqcup \mathrm{post}(n, y)$.

<u>example</u>: Avail. exprs.

$$\text{Post}_{AE}(x = e, E) = \{e' \in E \cup \{e\} : x \notin e'\}$$

$$\text{post}_{AE}(x = e, E_1 \cap E_2)$$

$$= \text{Post}_{AE}\ \{e' \in (E_1 \cap E_2) \cup \{e\} : x \notin e'\}$$

$$= \{e' \in E_1 \cup \{e\} : x \notin e'\} \cap \{e' \in E_2 \cup \{e\} : x \notin e'\}$$

[by De Morgan]

$$= \text{post}_{AE}(x = e, E_1) \cap \text{post}(x = e, E_2)$$

... but not for Post_{cp} of const. prop:

$$\text{Post}_{cp}(x := x+y, \{x \mapsto 0, y \mapsto 1\} \sqcup \{x \mapsto 1, y \mapsto 0\})$$

$$= \text{Post}_{cp}(x := x+y, \{x \to \top, y \to \top\})$$

but...

$$\text{Post}_{cp}(x := x+y, \{x \mapsto 0, y \mapsto 1\}) = \{y \to 1, x = 1\}$$

$$\sqcup\ \text{Post}_{cp}(x := x+y, \{x \mapsto 1, y \mapsto 0\}) = \{x = 1, y = 0\}$$

$$\text{Join}: \{x \mapsto 1, y \mapsto \top\} \neq \text{Post}_{cp} \text{ of Joins.}$$

In fact if post is monotone,

post of joins is always worse than join of posts

So in const. prop., the local condition ≠ global condition, but since we're joining conservatively it's still a conservative approximation.

Gen/kill Analysis

If we formulate an analysis as gen/kill then we have some nice properties.

- Suppose a finite set of of data flow "facts" e.g. available expressions, (also gen/kill)
- elements of Abstract domain are sets of facts
- For each BB n, associate set of generated facts gen(n) and killed facts kill(n).
- Define $post_c(n, F) = (F \setminus kill(n)) \cup gen(n)$

e.g. anything ~~in~~ in available exprs involving x on the & kill anything involving x on rhs. gen is x=e.

Ordering for G/K:

① $\subseteq$ for existential analyses
   a fact holds at n if it holds some paths to n.

② $\supseteq$ for universal analyses
   a fact holds at n if it holds along all paths to n.

ex ①: Variable possibly uninitialized at n
   if it is possibly uninitialized along paths to n

②: ~~Variable~~ Available expressions.

In either case Post is monotone and distributive
 - directly follows from set operation properties
 - set coincidence thm for gen.

(Not as expressive as generic dataflow analysis.
 But have some nice properties).

Possibly Uninitialized Variables Analysis

- A variable x is possibly uninitialized at a location n if $\exists$ some path from start to n along which x is never written to.

As Gen/kill analysis:

- Abstract domain contains facts
- existential analysis
  → $\perp$ least element $= \emptyset$
  → set of all vars → $\top$ element
  → $\cup = \sqcup$

$$Post_{UV}\,(x = e, E) \rightarrow \begin{array}{l} ① \text{ kill } x \\ ② \text{ gen } \emptyset \end{array}$$

kill everything involving x. if we want uninitialized expr. but for now, only x.

Reaching defs analysis

- def is a pair (n, x) with node n and var x such that n contains assignment to x.
- def reaches a node m if $\exists$ path from start to m such that latest def of x along path is at n.

reachg defs ~~&~~ Also existential analysis

$$\text{Post}_{rd}((x = e), E) \left\{ \begin{array}{l} \text{kill}_d \{ (m, x) : m \in N, (x = e) \text{ in } n \} \\ \text{gen}_d(n, x) : x = e \text{ in } n \} \end{array} \right.$$

Abstract domain: $2^{N \times Var}$

Wrap-up

Program analysis is used to inform optimization.

- Solving $\underbrace{\text{constraint systems}}_{\text{fixed point algorithms.}}$ over ordered sets appear in many areas of CS
  - Parsing - first, follow, nullable
  - Networking - shortest paths
  - Automated planning - dist-to-goal estimation.

Duality : If gen/kill analysis is existential ⟷ Universal dual exists.

"dual analysis" — reachability,

(in existential reachable set if ∃... preceding) ∀ node, what are the nodes that can reach this node.

graph-theoretic characterization of what a loop is in compiled.

dual — Dominance, v dominated v if all path from start → v goes to u, Universal.