Formalization of DFA using lattices

Recall worklist algorithm

```plaintext
let m: map from edge to computed value at edge
let worklist: work list of nodes

for each edge e in CFG do
  m(e) := Ø

for each node n do
  worklist.add(n)

while (worklist.empty.not) do
  let n := worklist.remove_any;
  let info_in := m(n.incoming_edges);
  let info_out := F(n, info_in);
  for i := 0 .. info_out.length do
    let new_info := m(n.outgoing_edges[i])
    if (m(n.outgoing_edges[i]) ≠ new_info)
      m(n.outgoing_edges[i]) := new_info;
    worklist.add(n.outgoing_edges[i].dst);
```

Using lattices

• We formalize our domain with a powerset lattice
• What should be top and what should be bottom?

Using lattices

• We formalize our domain with a powerset lattice
• What should be top and what should be bottom?
• Does it matter?
  – It matters because, as we’ve seen, there is a notion of approximation, and this notion shows up in the lattice

Using lattices

• Unfortunately:
  – dataflow analysis community has picked one direction
  – abstract interpretation community has picked the other
• We will work with the abstract interpretation direction
• Bottom is the most precise (optimistic) answer, Top the most imprecise (conservative)

Direction of lattice

• Always safe to go up in the lattice
• Can always set the result to T
• Hard to go down in the lattice
• So ... Bottom will be the empty set in reaching defs
Worklist algorithm using lattices

let m: map from edge to computed value at edge
let worklist: work list of nodes

for each edge e in CFG do
  m(e) := ⊥
for each node n do
  worklist.add(n)

while (worklist.empty.not) do
  let n := worklist.remove_any;
  let info_in := m(n.incoming_edges);
  let info_out := F(n, info_in);
  for i := 0 .. info_out.length do
    let new_info := m(n.outgoing_edges[i])
    info_out[i];
    if (m(n.outgoing_edges[i])  new_info)
      m(n.outgoing_edges[i]) := new_info;
    worklist.add(n.outgoing_edges[i].dst);

Termination of this algorithm?

• For reaching definitions, it terminates...
• Why?
  – lattice is finite
• Can we loosen this requirement?
  – Yes, we only require the lattice to have a finite height
• Height of a lattice: length of the longest ascending or descending chain
• Height of lattice (2^S, ⊆) = | S |

Even more formal

• To reason more formally about termination and precision, we re-express our worklist algorithm mathematically

• We will use fixed points to formalize our algorithm

Termination

• Still, it’s annoying to have to perform a join in the worklist algorithm

• It would be nice to get rid of it, if there is a property of the flow functions that would allow us to do so

Fixed points

• Recall, we are computing m, a map from edges to dataflow information

• Define a global flow function F as follows: F takes a map m as a parameter and returns a new map m’, in which individual local flow functions have been applied
Fixed points

• We want to find a fixed point of F, that is to say a map m such that m = F(m)
• Approach to doing this?
• Define \( \tilde{\bot} \), which is \( \bot \) lifted to be a map:
  \( \tilde{\bot} = \lambda e. \bot \)
• Compute \( F(\tilde{\bot}) \), then \( F(F(\tilde{\bot})) \), then \( F(F(F(\tilde{\bot}))) \), ... until the result doesn’t change anymore

Fixed points

• Formally:
  \[
  \text{Seq} = \bigcup_{i=0}^{\infty} F^i(\tilde{\bot})
  \]

• We would like the sequence \( F^i(\tilde{\bot}) \) for \( i = 0, 1, 2 \) ... to be increasing, so we can get rid of the outer join
• Require that \( F \) be monotonic:
  \( \forall a, b \in \text{dom} \) \( a \subseteq b \Rightarrow F(a) \subseteq F(b) \)

Fixed points

Back to termination

• So if \( F \) is monotonic, we have what we want: finite height \Rightarrow \text{termination}, without the outer join
• Also, if the local flow functions are monotonic, then global flow function \( F \) is monotonic

Another benefit of monotonicity

• Suppose Marsians came to earth, and miraculously give you a fixed point of \( F \), call it \( \text{fp} \).
• Then:
Another benefit of monotonicity

• Suppose Marsians came to earth, and miraculously give you a fixed point of $F$, call it $fp$.

Then:

\[\begin{align*}
\emptyset \subseteq fp \\
F(\emptyset) \subseteq F(fp) \\
F(\emptyset) \subseteq fp \\
\emptyset \subseteq F(fp)
\end{align*}\]

Recap

• Let’s do a recap of what we’ve seen so far

• Started with worklist algorithm for reaching definitions

Worklist algorithm for reaching defns

let $m$: map from edge to computed value at edge
let worklist: work list of nodes
for each edge $e$ in CFG do
  $m(e) := \emptyset$
for each node $n$ do
  worklist.add($n$)
while (worklist.empty.not) do
  let $n :=$ worklist.remove_any;
  let info_in := $m(n$.incoming_edges$)$;
  let info_out := $F(n, info_in)$;
  for $i := 0 .. info_out$.length do
    let new_info := $m(n$.outgoing_edges$[i]).dst$;
    if ($m(n$.outgoing_edges$[i]) \neq new_info$)
      $m(n$.outgoing_edges$[i]) := new_info$;
    worklist.add($n$.outgoing_edges$[i].dst$);

Generalized algorithm using lattices

let $m$: map from edge to computed value at edge
let worklist: work list of nodes
for each edge $e$ in CFG do
  $m(e) := \emptyset$
for each node $n$ do
  worklist.add($n$)
while (worklist.empty.not) do
  let $n :=$ worklist.remove_any;
  let info_in := $m(n$.incoming_edges$)$;
  let info_out := $F(n, info_in)$;
  for $i := 0 .. info_out$.length do
    let new_info := $m(n$.outgoing_edges$[i]).dst$;
    if ($m(n$.outgoing_edges$[i]) \neq new_info$)
      $m(n$.outgoing_edges$[i]) := new_info$;
    worklist.add($n$.outgoing_edges$[i].dst$);

Next step: removed outer join

• Wanted to remove the outer join, while still providing termination guarantee

• To do this, we re-expressed our algorithm more formally

• We first defined a "global" flow function $F$, and then expressed our algorithm as a fixed point computation
Guarantees

• If $F$ is monotonic, don’t need outer join
• If $F$ is monotonic and height of lattice is finite: iterative algorithm terminates
• If $F$ is monotonic, the fixed point we find is the least fixed point.

What about if we start at top?

• What if we start with $\top: F(\top), F(F(\top)), F(F(F(\top)))$

• We get the greatest fixed point
• Why do we prefer the least fixed point?
  – More precise
Graphically, another way