Formalization of DFA using lattices

1

Getting help

Are you getting enough help and support?

A. Yes

B. No because I didn't realize there were office hours

C. No because the office hours are at a time that I can't make

D. No because I'm embarrassed to ask for help

E. No because of some other reason

2

Project

How is the project going?

A. Easy, getting it all done quickly and easily

B. Challenging but doable

C. Very challenging, I'm having a hard time

D. Have no clue where to start

3

Recall worklist algorithm

```
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);
```

4

Using lattices

- We formalize our domain with a powerset lattice
- But more generally ANY lattice
- What should be top and what should be bottom?

5

Using lattices

- We formalize our domain with a powerset lattice
- But more generally ANY 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

6

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)

7

Direction of lattice

- Always safe to go up in the lattice
- Can always set the result to $\top$
- Hard to go down in the lattice
- Bottom will be the empty set in reaching defs

8

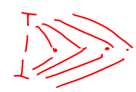
Worklist algorithm using lattices

```
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);
```

9

Termination of this algorithm?

- For reaching definitions, it terminates...
- Why?
  - lattice is finite
- Can we loosen this requirement?



10

Termination of this algorithm?

S {1,2,3}

{1,2}

{1}

∅

- 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 $(2^S, \subseteq)$ = ??

A. | S | - 1
B. | S |
C. | S | + 1
D. None of the above

11

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 $(2^S, \subseteq)$ = | S |

12

Termination

- Still, it's annoying to have to perform a join in the worklist algorithm

```
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);
```

- 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

13

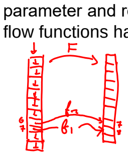
Even more formal

- To reason more formally about termination and precision, we re-express our worklist algorithm mathematically
- We will use fixed points to formalize our algorithm

14

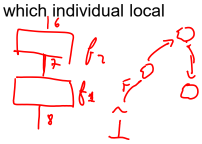
Fixed points

- 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



$m = F(m)$

$F(F(A\ell\ell_{\perp}))$



15

Fixed points

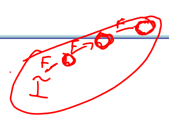
- We want to find a fixed point of F, that is to say a map m such that m = F(m)
- Approach to doing this?
- Define $\tilde{\perp}$, which is $\perp$ lifted to be a map:
  $\tilde{\perp} = \lambda$ e. $\perp$
- Compute $F(\tilde{\perp})$, then $F(F(\tilde{\perp}))$, then $F(F(F(\tilde{\perp})))$, ...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

16

Fixed points

- Formally:

$Soln = \bigsqcup_{i=0}^{\infty} F^i(\tilde{\perp})$



- Outer join has same role here as in worklist algorithm: guarantee that results keep increasing
- BUT: if the sequence $F^i(\tilde{\perp})$ for i = 0, 1, 2 ... is increasing, we can get rid of the outer join!

17

Fixed points

- Formally:

$Soln = \cancel{\bigsqcup_{i=0}^{\infty}} F^i(\tilde{\perp})$

- Outer join has same role here as in worklist algorithm: guarantee that results keep increasing
- BUT: if the sequence $F^i(\tilde{\perp})$ for i = 0, 1, 2 ... is increasing, we can get rid of the outer join!
- How? Require that F be monotonic:
  - $\forall$ a, b . a $\sqsubseteq$ b $\Rightarrow$ F(a) $\sqsubseteq$ F(b)

18

Little bit more about monotonicity

- Definition: F is monotonic if and only if:
  - $\forall$ a, b . a $\sqsubseteq$ b $\Rightarrow$ F(a) $\sqsubseteq$ F(b)
- Which of the following is true:
  - A. If F is monotonic then $\forall$ a . F(a) $\sqsubseteq$ a
  - B. If F is monotonic then $\forall$ a . a $\sqsubseteq$ F(a)
  - C. If $\forall$ a . F(a) $\sqsubseteq$ a then F is monotonic
  - D. If $\forall$ a . a $\sqsubseteq$ F(a) then F is monotonic
  - E. None of the above or more than one of the above

19

Fixed points

$F(\tilde{\bot}) \sqsubseteq FF(\bot)$

$\tilde{\bot} \sqsubseteq F(\tilde{\bot})$

20

Fixed points

$\tilde{\bot} \sqsubseteq F(\tilde{\bot})$

$F(\tilde{\bot}) \sqsubseteq F(F(\tilde{\bot}))$

$F^{k}(\tilde{\bot}) \sqsubseteq F^{k+1}(\tilde{\bot})$

$F^{k+1}(\tilde{\bot}) \sqsubseteq F^{k+2}(\tilde{\bot})$

21

Back to termination

- So if F is monotonic, we have what we want: finite height $\Rightarrow$ termination, without the outer join
- Also, if the local flow functions are monotonic, then global flow function F is monotonic

22

Another benefit of monotonicity

- Suppose Marsians came to earth, and miraculously give you a fixed point of F, call it fp.
- Then:

$ofp \sqsubseteq fp$

$FF(\bot) \sqsubseteq fp$

$F(\bot) \sqsubseteq fp$

$\bot \sqsubseteq fp$

$\{a\}$ $\{b\}$

23

Another benefit of monotonicity

- Suppose Marsians came to earth, and miraculously give you a fixed point of F, call it fp.
- Then:

$\tilde{\bot} \sqsubseteq fp$

$F(\tilde{\bot}) \sqsubseteq F(fp)$

$F(\tilde{\bot}) \sqsubseteq fp$

$F^{2}(\tilde{\bot}) \sqsubseteq fp$

$\vdots$

$ofp \sqsubseteq fp$

24

Another benefit of monotonicity

- We are computing the least fixed point...

25

Recap

- Let's do a recap of what we've seen so far
- Started with worklist algorithm for reaching definitions

26

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) := ∅

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);
```

27

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) := ⊥

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);
```

28

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

29

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.

30

What about if we start at top?

- What if we start with $\widetilde{\top}$: $F(\widetilde{\top})$, $F(F(\widetilde{\top}))$, $F(F(F(\widetilde{\top})))$

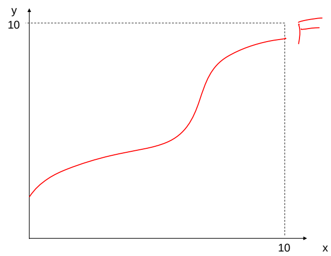
31

What about if we start at top?

- What if we start with $\widetilde{\top}$: $F(\widetilde{\top})$, $F(F(\widetilde{\top}))$, $F(F(F(\widetilde{\top})))$
- We get the greatest fixed point
- Why do we prefer the least fixed point?
  - More precise

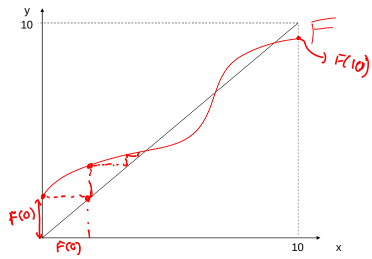
32

Graphically



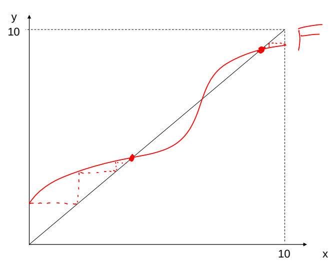
33

Graphically



34

Graphically



35

Graphically, another way

36