웹 소프트웨어 신뢰성

- ✓ **Instructor: Gregg Rothermel**
- ✓ **Institution:** 한국과학기술원
- ✓ **Dictated: OES** 봉사단

We talked about data flow analysis.

We talked about the algorithm for iterative data flow analysis on two different applications: reaching definitions, reachable uses.

We talked about different algorithm for computing definition use: pairs or reaching definitions or whatever you want based on a work list.

Then I got two about there.

And I wanted to finish that up.

Before I do that, there are answers.

I actually asked to do some problems.

I just need to turn on access to the tables that are online that shows the definitions of pairs tabular form rather than graph form.

So I'm not going to go through this in class, it's going to take way too long.

So, but hope you have a chance to go through and calculate definition each pairs, that's the only way to learn this algorithm to apply them.

Everyone have some time to do that? Okay. Yes, some time, wow, colored pens.

So what I'll ask you to do is take those answers that you have, between now and next class check them against which here, online I'll enable after I get back to my office.

Correct them and turn them in next class. Okay? You have chance to fix it if it's wrong.

But I do want to make sure you've all done that. Okay?

CC BY NC ND

So to finish up data flow analysis, just a few things we wanted to do.

Oh, any questions before I go on? I should've asked you in that assignment you might had questions. All okay? You can all… On the midterm which we'll not going to have, I can ask you to compute dataflow, to do some dataflow analysis, on the fictional midterm.

Okay, just a few other things that I want you to know about.

Couple other terms we may run into.

Definition use chain otherwise known as DU-chain. It connects definitions up to uses that they reach.

When we do reaching definitions, we would take a definition at some point, x equals something, and propagated around the graph to all the points it reached, whether or not, y=z, a=x, and we propagated to all the insets.

And that's the, we would know that this definition reached there. x=2 reaches there, x=2 I'm not trying to write everything out.

Reached definitions reach every point, but usually we're going to be interested in connecting things up.

And so a DU-chain connects definitions with the uses they reach.

So the only use this reach is this one.

So a DU-chain for this definition consists of this definition and this use.

Use definition chain goes the opposite direction.

I don't know why they call these chains because chain usually applies chain, they're really just sets.

But…

So, what's the DU-chain? This notation means definition variable 'x' in block 2. So what's the DU-chain for the definition of 'x' in block 2? You got to find the uses of 'x' that can be reached from there.

BY NC ND

And without doing dataflow analysis, we can just look at this graph and figure them out. What uses of ‘x’ can be reached from here? B3? B3 ‘x’.So we’re going to start putting DU-chain, and I’ll going to say {(x, 3),(x, 5)} (x,6)? Oh wait, there’s a kill. I almost got it too, missed it too. (x,4) what things does it reach?

There’s 4. {(x, 5)}and again there’s a kill, don’t get any farther. Okay?

(x, 5) where does it reach? We understand to be the definition of exit okay?

B6, yeah. (y, 3) nothing, null set. (z, 5) null set. (z, 6) the same thing.

So this is just small example but those are the DU-chains.

Okay. And that matches what I have there.

Use definition chains for each use which definition reach it.

Let’s work out way up. For the use of ‘x’ and 6, which definitions reach it? B5. What about B4?

Nope, because it’s killed, just the same looking from the other direction.

Use definition chain for (x, 5). And that means, again, the use in this case, B4 and B2. Oops, forgetting my sets. (x, 3) just 2. (z,2)nothing. And (z,1) is the same.

Okay? So it’s not difficult concept.

You find with the definition you get to, you got your insets, and then you go through each node, and at each, let me say this better, you can calculate your insets, reaching definitions, this is for DU-chains at each node, and at each node you look for use that occurs in the node, and each use gets paired with every definition that variable that reaches.

And then you just read through the nodes if wants to do this. Okay?

A data dependence graph, graphs, essentially these definition use, I call them DU-chains, they’re also sometimes, in testing we’ll find them call definition use pairs which is often abbreviated to def-use pairs, ordef- use associations, you’ll see them call various things in papers. Okay?

CC BY NC ND

So def is definition. And a data dependence graph, graphs, these, or can graph of other types of data interactions, and here I've got a control flow graph, and the data dependencies connect nodes, based on types of definitions.

And the type we're usually interested in our word dataflow edges or flow edges that connect definitions to use.

And those edges just correspond to the DU pairs we calculated.

In some circumstances, anti-dependencies from use to def. are used such as node they exist.

I don't think we'll run into those.

And sometimes output dependencies, from definition to definition are used, that can matter in concurrent programs, when you might have a right after a right and then you wanted have them in right order before some other thread uses of the variables, something like that.

But you can graph them, this is graphic containszest of datadepends the edges, you can add them to the control flow graph.

As some of you, at least one of you did on your papers with your answers.

Last, just to show you that there are other types.

I'm not going to do this last one.

I don't know why it's there.

There are some other types of data flow analysis or other problems, there's actually many many problems that can be solved with different analyses.

I wanted to talk about at least two others. I will have going to see reaching definitions, and reachable uses.

Live variables, is a different concept. It's from the compiler word, which I started off the last lecture by mentioning these, in a compiler situation, when I'm generating code, I'm putting some variables into the register for fast access, and I certainly don't

BY NC ND

want to put the variables aren't live into registers, that means the variables don't have any further use in the program.

You want to put the variables that do have further uses, into registers.

So variables live if there is a path, at point p in the program, if there is a path, from 'p' to a use on which 'p' is not redefined, it sounds a lot like reaching definitions.

Doesn't it? It's just a different, what is live, It's verb? Adjective? Well, Whatever. Okay.

With all of these data flow problems, there're defined in terms of essentially these items.

The members of sets, let's back up to reaching definitions for a minute.

The members of the sets in reaching definitions was definitions.

The local information was the stuff coming in and out of the block and it was defined by gen and kill, coming in or not getting through a block.

The propagated sets, or the in and out sets, then we had formulas for those, and you'll calculate in based on something coming out of predecessors or maybe turning in around if you are backward flow, direction of the flow, forward, backward, in the graph. And reaching definition was forward, reachable use was backward.

Confluence operator.

In reaching definitions, when you get to a node, the inset here, is calculated as the union of things come out.

And that's because we're interested in "does the definition reached by some path" So if it comes in any path, I wanted to be here.

And that's why, it could jets here, and I get it and it could jets here and that's by union.

There're some problems, where, if this comes in every path, I'll show you one of those in a minute and the then the conference operator would be intersection.

The transfer function essentially that's a definitions of in and out, the way information

BY NC ND

move through, and the algorithm, well It's pretty much a template for an algorithm, is what I've been showing reaching defs and reachable uses, once you substitute it in the things here.

Officially that's IN=U OUT(P) for all predecessors.

P of there has to be node here IN(N)=U OUT(P) for all predecessors p of n. that's transfer function.

OUT(N)=GEN(N) U IN(N) – KILL(N) those are transfer functions. Okay?

So, in this case, the confluence operator, part of transfer functions, and if I'm doing intersection, I have that. Okay? Otherquestions?

Now live variables.

Well, one of the members of the sets, what do we propagating around the graph?

From point P if there is the path, in the CFG from P to a use of V.

So variable V, so live at point p, if there's a path from P.

Some path, pretty multiple paths, to a use of V.

So what things are we going to propagate around to the graph?

Let's see. Let's put up more of an example.

I don't know. Not a useful computation. But, this is point P.

I want to know actually if V is alive here.

So, is V alive here? According to definition? It is, because the path from here to here.

There's path in the CFG to a use.

So it can be used.

Now if I want to find, if I want sets, let's say I want to know sets, of variables, that are alive at this point, how can I get that set there? Is it going to be more like reaching definition or reachable uses? What do you think? When I started out it looked like

BY NC ND

this, so how did V get here?

If I did an algorithm, I followed V reachable from there.

And the use of V is here. What would get it up to there? Go ahead.

I'm just interested in V at the moment.

Cause it's the example. Yeah, not the whole algorithm, I started with one of the members of the sets, but that wasn't working so I thought ask, I'm going to tell you what the members of the sets are.

Members of the sets are variables. Okay?

Because, we're asking whether the variables are alive.

So the members of the sets are the variables.

And that means that's what we'll going to propagate around the graph.

Now I want to know which variables are alive at this point.

That means at this point for which variables does exist is path from here that will reach those variables.

And you said V because here it is.

And there could be another one down there, and plus do that.

They're essentially two live variables here.

Well 'b' and 'c' also are all alive because from here, you can get to uses and they're not killed.

So those are all the variables at that point.

So what I want in the end is these sets that list the live variables.

This one will list, I think will list the same one.

And this X also. B,C,X.V.Q. That's the insets.

CC BY NC ND

The outset here will list v and q. Okay? So things are propagating variables.

In an algorithm, you saw propagating definitions around or propagating uses.

So what would, what kind of algorithm would let me build this set? What would I have to do to get V in the set? Iterate. Iterated from where to where? In which direction what I'm trying to say?

Yes. That's right.

So is that going back wardor forward in flow? Backward. That's right, okay?

So we're going to propagate the variables backward.

If they're killed they won't propagate any further, okay?

So at local information, just the equal, actually just variables really, the transfer functions, well you can say that the GEN and the KILL sets.

The GENset is set of variables upward exposed this is just like reachable uses.

Set of variable supper is exposed the KILL set is any variables in the program that are redefined. Just like reachable uses.

Propagated sets are variables. Direction of flow which are backwards.

The confluence operator.

If I got multiple edges here.

This is, I'll put Q equals something else in there.

Q equals R. I know that's engine we will still put v and the set of reachable uses here.

When there two edgs out, why do you say yes? You're right.

Why do you say it? At least one path.

Very good, okay? I don't say that to be every path at least one.

So there is at least one.

BY NC ND

And therefore that's in the set there.

So what, is it union orinter section? That I used to decide to whether it's in the outset here.

Isn't union wanted to coming out one. Okay?

That's the confluence operator.

And then the transfer function is going to be same as reachable uses, okay? So really the only differences reachable uses is that are propagating variables around.

And now, technically speaking, oh, never mind.

I don't know what I was to say.

So that's that pretty much like reachable uses of algorithm analysis.

Now let's look at one little different.

And that's expressions.

And I have mentioned this couple times.

An expression.

This is use for register allocation.

There may be other uses, but that's the one I know about.

So If you've got your code generated, at some point in the code generation phase, that our statements but your mapping into machine language binary code, whatever, and you're choosing some variables and registers, and if registers and you don't need to do a load operation on it.

So you save some time.

But the ones you put in registers other ones get used a lot for they're alive.

The what? But this is done only in compilation not during run time, okay?

But what happens is we end up taking these pieces of code, and generating, well, I

CC BY NC ND

could do it in assembly language LOAD; R1, B: ADD; R1, C Maybe I have to load or loaded to I've forgotten.

A STORE, I am who writes assembly language anymore; R1, A right, okay? So the compiler is generating the code and that gets assemble into binary, okay? So if we're able to just if we know that the b is a live variable somewhere I could choose I may be loaded it already. And I already know it's in our one and I don't need extra LOAD, okay? So I get to decide which ones to keep in registers.

Such as LOOP increment variables, it might stay in the registers.

And so there's only during the compilation, so doesn't slow down the program in fact speed program run time.

Does it slow down computation? Of course.

Because you're doing extra, you got to compute live variables forth e code generation phase.

But that's compilation we rather, we're willing this more time of compilation to get a faster program. Make sense? Okay. Good question.

An expression.

Well expression means something on the right side.

That's an expression.

That's an expression.

That's an expression essentially.

Is available at a point p at a point p if on every path in CFG to P the expression in computed and there is no redefinition of variables between the computation and P.

So b plus c is available at this point because on every path two p there is only one path the expression was computed and b and c were not redefined.

And why do I care about that? Well because if it's available I don't have to do this computation again.

CC BY NC ND

I don't do have all that.

I can instead change this to TMP equals A, A equals TMP, Z equals TMP, and theV equals TMP.

TMPs might be variable or live registers something like that.

So and use this things then you can build other optimizations on the top of them.

So that's why we're interested in available expressions.

So in this case we are saying, I asked questions.

I can ask, Let me ask a different question.

What expressions are available at point P?

Let's just concern ourselves with these two each expressions.

So we already saw that this is available here, because on every path from to here variables are re defined which of course means I couldn't, I had to have them recalculated A.

If the variables are redefined, I can reuse the old b plus c.

Is this expression available down here? How come?

That's right, very good, okay?

So now the question is how do we get well what are the dataflow propagates things around?

What do we need propagate around the graph in this case?

If I want to know the expressions that are available here, and what I've got this set of expressions are available there are just b plus c and the set of expressions are available here b plus c andin this case actually sets.

{(b+c), (f(x)+g)}, okay? When this algorithm in order to do this optimization, I want to have these set of available expressions at each node.

BY NC ND

So how do I get that set to the top of the nodes? What do I propagate around?

A notion? What did I put here?

An expression.

If I'm interested in the available expressions that a point it's the expressions that I want to propagate around just as I'm interested in the definitions that reach a point is definitions that I propagated around.

Are the variables that are point, it's the variables, okay? So the expressions.

Local information.

What's the, for a known, what's the GEN set? What does the node generate?

What comes out of a node?

Think back to reaching definitions. What does this node generate? Oh, okay, you're talking about expressions, and you're right.

F(x) + g. So on reaching definitions; it generates V.

On available expressions; it generates the expression.

That's what comes out it, okay? So the GEN set equals the set of expressions computed in the node.

Now if it's a basic box there might be several such expressions, okay?

What's the KILL set? Definition variable.

The things that are in KILL sets.

In reaching definitions, The GEN set produces variables that are defined.

And the KILL set kills variables.

And in available expressions the GEN sets are expressions. And the KILL sets are going to be expressions, but which expressions? So the KILL set, let me back up to reaching definitions again.

CC BY NC ND

The kill set for this is any definition of x in the program.

Now you said to me that this kills this expression.

Why does it kill the expression? It's in the definition.

Perfect.

KILL set is going to be the expressions that contain a variable that we redefined here.

That's from the definition, okay? And any expression in the program essentially, because you remember, we compute at locally without knowing what might reach what.

GEN=Expressions created in node.

KILL=all expressions in program such that a variable.

I might say the expressions e, in ‘e’ is redefined in the node, okay? Propagated sets are the expressions.

Direction of flow.

What were we doing here? Was I pushing expressions down or pulling them up?

Pushing them down.

Expression comes out here I'm C see where it get to, so directions of flow is forward.

What is the confluence operator? They both come from the entry of the program.

Does f(x) + g, does that expression reach here? Check the definition out.

There’s four key words in this definition that say something, related to what the operator has to be. Okay. How come?

On every path.

That's a key. This expression is not computed on every path to here.

That means you can’t put into a temporary variable and assume that variable would be available.

cc BY NC ND

Cause the program comes from here, it's not, has to be on every path.

So, how do I calculate the inset here? I have to do with an intersection.

I have to say.

It has to be on every, you have to correct out that don't come in by some paths. When things come together on multiple paths, when you've got a data flowing on around the program.

Sometimes you're interested in whether something comes in some path.

So when I'm calculating what reaches here.

Let’s say it's the reaching variable, so it doesn't matter what's here.

So there is a definition v there v equals 2, then definition of Y here.

Both of these get here because they come in some paths.

So IN=U of the things coming in on the path to come in. If I'm interested in something that, if I'm insisting of something that has been on every path, I can't do union.

And so if you look up here, If I say the available expressions are all those coming on any path.

I would say both of these are available.

And that's not true, only the truth is coming on every path.

So confluence, the word confluence means, it’s about coming together.

There's the confluence of two rivers what North East of Seoul, the two branches of Han River.

That's the confluence the word is that come together, okay?

And the confluence operator is the thing that you use to determine which things come in okay?

The confluence operator’s intersection.

CC BY NC ND

So the things coming in.

So the in equals the intersection over, I mean .

The inset of node N equals the intersection across the predecessor's P of N of the outset of P.

And the sets are expressions okay? So this is saying take a look at all the predecessors see intersection across those of the expressions that come in the inset.

And then the outset is going to be of the gen set together with in minus KILL.

So the expressions that come out are just the ones that get produced together with an In came in and didn't get killed.

That's the same as these other things that we do. Okay.

And you could plug those formulas in to the algorithm that's in the paper for reaching definitions and you have the.

So if there are times where you want to, if you're looking at some problem and you realize that it involves flow of something around a program, you can come up with a data flow algorithm for it.

Like thinking of these things.

A little more complexity you want it to converge you want the sets to always increase or decrease so that it terminates.

So its whole class of algorithms and if you were to do things on mistake you'd learn other data flow algorithms.

That ends the data flow stuff.

And begins the next topic, dominance post dominance.

I asked you to read, I asked you to read 4 and 5 but 5 is on the dominance stuff.

I don't know.

I hope I asked you to read.

Yeah, 5 is dominance and post dominance.

For next time, next time we'll start one more topic which is control dependence.

We've done data dependence, there's another thing called control dependence.

That's section 6 of this paper.

BY NC ND

You may find it useful.

So look at another paper that also defines it.

That's what's listed here.

It's not really pages 1-10, it's pages well it is pages, pieces of paper 1-10, it's pages 319-329 on the paper but the first ten pages are about control dependence competition.

They don't really give a good version of the algorithm I find a little confusing and so they describe it work and they give you an example and so between this and this you should be able to get a good view of what the control dependence competition algorithm is.

Dominators and Post dominators.

It turns out that how to calculate control dependence you need to calculate post dominators.

There's some other things that are used.

First, definition.

It's again based on control flow graphs.

Suppose you have a control flow graph with node D and node N.

D dominates N if every path from the initial node of Program let me put in E, the entry node every path from E to N goes through D.

Another way to put it is the only way you can get to N is by going through D.

Okay? If there were another path here, then D does not dominate N.

There are other ways to get there.

What dominance is telling you are things that must happen before things that necessarily will happen before.

That turns out to be important.

Now it's been to find a couple ways in the literature.

In one definition we'll say every node by definition dominates itself.

And sometimes that's not needed.

It depends on what your analysis is.

BY NC ND

I think in this paper I'd say that's not true, but you could go either way.

And the second thing is the initial node dominates all others.

E necessarily dominates all because the only way to get to any of them is through E.

Okay? Now, let's just fill in an example.

In trying to get intuition for this (43:31)

Well, one's a little bit off the screen but you can see that one is there and one is essentially the entry node.

So remember the definition. I'm going to say one, well one dominates everything.

That's the entry node that dominates everything.

So node 1 dominates 1,2,3,4,5,6,7,8,9 and 10, okay? Now, we're going to start with 2.

Does 2 dominate 3?

No. And if you want to prove that to yourself, cite the definition.

2 dominates 3 if every path from E to 3 goes through 2.

And that's not true. Okay?

Just reason about this.

Does 2 dominate anything? Yeah, there's always a path around it except itself, if we're going to include the reflexive thing for the moment, it dominates itself.

Ah 3. Does 3 dominate 2? No. Node 3 dominates 2 if every path from entry to 2 goes through 3.

And that's not true. Okay? Likewise 3 doesn't dominate entry.

Does 3 dominate 4? Every path from E to 4 has to go through 3.

So it does. 3 dominates 4.

Does 3 dominate 5? Every path from entry to 5 has to go through 3.

6. Yes? 7.

In fact I you can begin to see that 3 is going to get all of these.

Cause they're all..

BY NC ND

You've got to go through 3 to get to any of those, okay?

What does 4 dominate? Does 4 dominate 3? No.

Does every path from entry to 3 go through 4?

No. There is a path from entry to 3 that goes through 4 but not every path. Similarly doesn't dominate 2 or 1.

What about the other nodes? Does it dominate 5? Yeah, okay? 6, yeah.

In fact, everything down there.

4 dominates.. oh I should have put 3.

I always forget the reflexive ones.

Okay? 5.

Does 5 dominate 4? No. You can get to 4 without going through 5.

Does 5 dominate 7?

No.

Does 5 dominate anything other than itself? You can get to anything else without going through 5 basically so it's just itself. 6?

Similarly just itself. 7? What does 7 dominate? Besides itself?

8,9, and 10. 8. Yeah and itself. 9. Just itself.

And 10 okay.

So that's, you've just computed the dominator information.

Let's see the answers what I have here.

1-10, 3-10,4-10,5,6,7,8,9,10 yep.

Now, an algorithm for that.

Well, you can start to reason about it by thinking about these

properties.

Most immediately a dominates b, dom.

If a equals b, that's easy. Of if a is the unique

immediate predecessor. A dominates b.

CC BY NC ND

Certainly if a is the unique immediate predecessor, I can

see it dominates.

So, okay? So 4 dominates 8.

So I can calculate and say well 8 dominates itself, 4 dominates it.

5 is a unique immediate predecessor of 6. So 5 dominates 6.

7 doesn't have a unique immediate predecessor.

So nothing gets added by that.

But now the last one.

A dominates b if b has more than one predecessor.

And for all immediate predecessors, c of b, a dominates c.

So there's a transitivity thing here.

So 4 dominates 5 and 6. Therefore 4 will dominate 7.

And you can kind of. So this dom relation is reflexive transitive and anti symmetric.

So if a dominates b, b will dominate a.

Turns out you can solve this with a different problem.

An iterative data-flow problem.

And what we're passing around in this case is just the node itself.

And we're going to propagate the node to places that it dominates so that at the start of each node we'll have the set of nodes that dominate that node.

So the gen set for a node is itself.

So you can think of this as generating potential dominators.

So when you come out of. Let me put this a different way.

When you come out of entry, an entry is a potential dominator for things.

And when you get to 1 it's still a potential dominator.

And when you get to 2 it's still a potential dominator.

The KILL set is going to be. I think I'll put these answers up. Okay.

CC BY NC ND

The KILL set is empty because there's a, once the dominator comes through it doesn't get killed off, but it might seize the dominator.

And then sets by propagation.

In is the sets of dominators that are predecessors of b.

So the inset at 1 is the set of dominators at E, the inset of E is the set of dominators at 1, okay?

The outset is the set.

Oh these are the. .I'm sorry that's the initialization.

I'm getting confused on this myself. The initialization is to this.

The iteration is going to compute in as the intersection.

And the reason is for something that we saw transitive rule said the immediate ones dominate it or things that reach all of it's predecessors as dominators of all of it's predecessors dominate it.

So it's like the intersection of the dominators 5 and 6 gives you the dominators of 7.

And that's why we compute it with intersection

And then the out just becomes Gen and in.

And so we can apply the algorithm ; Well here's the algorithm.

N is the set of nodes. DN is the set of nodes that dominate N.

That's what we're calculating.

We'll start initializing each node to dominate itself.

And then we're going to do something a little different than before.

We're going to initialize DN to everything.

We're going to end up subtracting those from the dominator sets in this case.

By using intersection.

So let's see go through the algorithm.

I think that happens on the next slide.

So the algorithm since I had to take it off the screen, is on page 10. of the reputation analysis part on the bottom, okay?

CC BY NC ND

So you should be looking at that.

And good we've got the nodes and the dominator sets.

At the end. And first off, the dominator for node zero, which is the entry node of the graph, equals itself. Okay? And for every other node, we're going to let the dominators equal everything in the graph.

This may seem a little strange at first, but you'll see how it works.

We're going to let equal every graph every node 2, I'm not going to use commas.

Because these are all single digit.

Just restart like that. For each node other than the entry node let it's dominator set equal all the nodes .

And now we're going to start the y loop.

It iterates to the nodes. And __ to specify the order, I can use any order, but as in the data flow one going trying to go depth speed fastest order.

And that's what we'll do. And well changes to any DN occur.

So and we're never going to bother with the entry node, because the only thing that's going to be on the set is already there.

So, for each of the other ones.

The dominators of this node equal itself 1, together with the intersection of the dominators on all its immediate predecessors.

What is the immediate predecessor of node 1? There's that single 1. Okay?

It's this, together with the intersection of all the dominators of node E.

What's. Well the intersection is the set, intersection 1 node.

So what are the dominators at node E? Just E, okay? So the only one that ends up getting there is 1 and E.

So we quickly subtracted all of those.

Node 2.

We are going to keep 2 there.

Other than that, it's the intersection of

All the things coming in from it is immediate predecessor.

BY NC ND

Should happen to be node1.

So what are the dominators at node1?

E and 1.

So we are going to have E, 1 and 2 there.

Node 3.

We are going to keep 3.

And we are going to take intersection of things on Node 1.

So what's that?

E and 1 again. So it's E, 1 and 3.

Node 4 is a little more interesting.

We are going to include itself.

Now there are multiple predecessors. The predecessors are 2 and 3.

So we are going to take the intersection of the dominators of 2 and 3.

What are those? E and 1. Okay?

Set back from (in it then) ask yourself if that seems right.

Just looking at it. What things dominate for?

Well E does, 1 does because the only way to 4 is through 1.

2 doesn't because there's another way to get to 4.

So this algorithm subtracted out 2 and 3 because 2 came in only by this route and 3 came in only by that route. And to include them they'd have to come in by every route.

So that gave me 4.

Node 5.

Who are the dominators of Node 5?

Yes.

(...)

I'm sorry. You are right!

CC BY NC ND

Yeah, I should not admit that should be included there. Thank you very much.

And it gives the same result. Because of the intersection.

But yes I should have included 7 in that and intersected with that.

Thank you.

So at node5?

Predecessors 4 just one of them.

E, 1, 4 and 5 by that fault.

6.

5 is the only predecessor. 1 is there.

E, 1, 4, 5 and 6.

It's got a single predecessor. So the predecessors are going to dominate it and anything that dominated that predecessor is.

Node 7 is a little more interesting.

Not a ton, but little.

Now we need to get the intersection of 5 and 6.

So what's that?

Right.

So we are removing 6 because it's only on one path that's what effectively is happening here.

And 8.

E, 1, 4 to get a relate.

None of these dominate because you can go straight through.

Okay?

And a X..is ..

Just will be E, 1, 4, 8, X.

Now it only took me one path through the algorithm.

I'm going to another one and nothing will change and I will be done.

BY NC ND

That's not always the case.

If there were a more complex loops one might have to do it again in order to get in order to intersect things out properly.

And also I could have these in any order will take me a long time.

I can consider X first and then initially get everything on 8 which is still the whole set. And then 8, initially get everything on 4 and but eventually, as I went around and around and around, it will stabilize because every time through I'll subtract something, some sets, somewhere.

So there's that.

Now a dominator tree is a graphical representation of dominator information.

The initial node is a CFG is the roof of the tree.

The parent of a node is its immediate dominator.

So what's.. without calculating the dom thing we can probably figure out the tree on this.

The root of the node. The dominator tree is one. And then back of minute.

The parent of node N.

The initial node is N, the parent of node N is a .. is its immediate dominator should be ancestor. The ancestor of node N is graph is its immediate dominator.

I know where to put that is. Oops.

What things does 1 immediately dominate?

To be immediate dominating means it has to be its children.

Not something farther down.

So does node 1 dominate 2? Yes.

Does it dominate 3? Yes.

So the dom tree will go edges to 2 and 3.

The children of 2 are going to be things immediately dominates.

What things are those?

It doesn't dominate anything.

BY NC ND

So that's a dead end on that.

So what things does 3 immediately dominate? 4.

How about 4?

5 and 6 and there's another one. 7. Okay?

It's the closest dominator to 7.

5 and 6 don't dominate anything.

7 dominates 8.

8 dominates 9 and 10.

That's a dominator tree.

Here is an algorithm for it.

This is in your paper.

It involves making some sets.

So N is the set of nodes in CFG.

D(n) is a set of nodes that dominate N.

And you are going to build a Q called queue.

It's hard for me to go through this algorithm without all displaying the dominator information which I don't have back there anymore.

Well, you are able to build one by hand.

And we are able to look at one and build one by hand.

Let's see here.

I think I will leave this as an exercise that you will try in your homework.

And you can probably look at the graph and build the dominator tree but.. anything else there.

You've got to pull things of the Q's.

And then the ..I think its straight forward to try it.

You just go through slowly.

You should be able to make this algorithm work on your own and calculate a tree.

CC BY NC ND

The tree would be the same one that we came up with my hand in just a minute ago. So, you will come up with this.

Post dominates is kind of the universal dominates.

If dominates means that the node must come before in any path from the entry, post dominates means the node must come after in any path to the exit. Okay?

Given CFG with nodes PD and N

PD post dominates N if every path from N to the exit node go through PD.

So every path go through PD.

It post dominates.

If there's another path? It does not.

And it turns out you will see in the control the dependence discussion

Next time, we will need to confute post dominators.

There are other analyses to use dominators.

So this is a simple example.

Let's calculate this one too.

Okay.

Well, we could say every node post dominates itself.

Well, it's okay.

Back to the definition.

The 7 post dominates 6.

And 7.. 7 is the exit.

Why shouldn't I draw an exit in there?

Let's do that. Let's put an exit in there too just reform.

The 7 post dominates 6.

Every path from 6 to exit from 6 to exit go through 7.

You can't get the exit 6 without going through 7.

So don't let the fails that there's a multiple path to 7 confuse you.

CC BY NC ND

So 7 T 6.

Does it post dominate 5?

Every path from 5 the exit go through 7.

And you can begin to see what everything has the go through 7 to get to X.

So it is going to post dominate everything in here.

What about 6?

Do 6 post dominates 7?

There's not even a path from a 7 to X.

That goes through 6.

So no.

Do 6 post dominate 3?

No.

You can get a exit by different route.

Which one does six post dominate?

4 and 5 for sure because the only way to exit from here is through 6.

2? Only way is through there.

1? No. Okay?

So 2, 4 and 5.

6 post dominates 2, 4 and 5.

5. What does it dominate?

Does it post dominate 2?

No? 3? 1? And nothing else.

Just itself.

So I'm not putting in the reflexible ones here.

4? what is it post dominate?

It's like 5.

BY NC ND

There's always way around it to get to the exit.

3. Same deal.

2. Same deal.

1. So there are all just themselves.

Another way. Look at this is.

6 and 7 in this graph are the only nodes that you always have to go through from at least somewhere in the graph.

From these you always have to go through 6 to get to exit.

So it post dominates those. They always have to go to this route everywhere.

There's the other ways to get to the exit.

Post dominate tree.

Like the dominator tree but it's gotten from postdominance.

Maybe I want to put them back up.

The initial node is the exit node.

Try blue maybe it's better.

The initial node is the exit node.

Then you start finding things that are post dominate.

Let's back up for this for moment.

What things does exit immediately post dominate?

7.

X post dominate 7.

7 immediately post dominates 6 and 3.

And 6 post dominates 2, 4 and 5.

Where is 1 go?

There's 7.

This is saying 7 post dominates 1.

BY NC ND

What does it saying.

So this is the graph that exit post dominates 7.

Immediately post dominates 7.

7 immediately post dominates 6.

But basically there's nothing else between 1 and ..

The only way.. Okay.

Let's look at this way.

The only 1 as we had there's the only thing post dominates 1 in this graph is 7.

The only thing that's always between 1 and exit is 7.

So that's kind of immediate post dominate for latter, the closest post dominate if you will. None of this post dominate cause it can always take another path.

How do you make post dominates?

Pretty simple.

You take that flow graph and reverse all the arrows.

Now you got a reverse control for your graph.

And now, you just apply the dominators algorithm on reverse graph.

Tree in the exit node is the entry and entry node is the exit.

Dominators on reverse graph are the post dominates on original graph.

And you can build dominator-post dominate tree by applying the dominator tree algorithm to the set of post dominates.

So just from those path out in original how to do it if you can reverse a flow graph.

That ends those slides.

Questions?

So let's see backing up again.

It's not written here but next time you check out your data flow answers.

Against the answers that are online.

Correct them. Make sure that you understand data flow analysis algorithm and why

BY NC ND

it comes up with those answers.

And then read about the dominator stuff.

I mean.. excuse me.. read about control the dependence.

And analysis in this representation paper.

Read the 10 pages of Ferrante.

And start to understand how we build control the dependence graphs.

And that's we will talk about next time.

I wasn't sure how long this would take.

So online I did make.. you might of seen post another set of slides called testing fundamentals obviously we didn't get to those today.

Might be couple of classes before we get to them know.

So I'm just going to take them off and moving down later.

If you printed or kept them.. forget them.

BY NC ND