NTNU – Trondheim
Norwegian University of
Science and Technology

Fixed points

Some key observations from last time

- Denotational semantics describe the function that a program computes
- Finding that function is structured by attaching rules about how to combine functions to all the syntactic elements
- Unlike all the other rules we have attached to syntactic elements in this course, these ones are not instructions for a machine to follow
  - A machine can't work out the function of a possibly infinite process, *cf.* Turing

Some key observations from last time

- They are rather instructions for an Eager Theorist to follow, so that we can work out what the function of a program *should* be
- A machine can then run an implementation, and if it doesn't evaluate the function that the Eager Theorist prescribed, people can have arguments about whose fault it is
- They could always have such arguments anyway, but in this manner, it will at least be a rigorously structured argument

Where we stopped

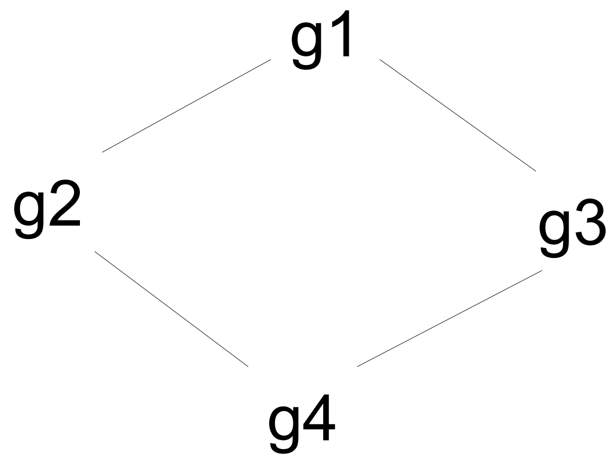
- The semantic function of a loop is a fixed point of the functional given by the loop
- A whole bunch of functions can fit that description
- They can be compared with each other according to how much stuff they specify
- The semantic function of a loop can be made unique by picking the smallest fixed point in this ordering

The ordering relation

- With fixed points g, g', the relation $g \preccurlyeq g'$ means that g shares its effects with g'

  in the sense that if g1 s = s', then g2 s = s'
- Take, for instance
  - g1 s = s
  - g2 s = s if x>=0, undef otherwise
  - g3 s = s if x<=0, undef otherwise
  - g4 s = s if x=0, undef otherwise
- $g4 \preccurlyeq g4$ (it has the same effect as itself)
- $g4 \preccurlyeq g2$ and $g4 \preccurlyeq g3$ (they're the same in x=0)
- g2, g3 aren't related (they're defined in different places)
- $g2 \preccurlyeq g1$ and $g3 \preccurlyeq g1$ (and it follows that $g4 \preccurlyeq g1$)

We can draw this



- The line between g4 and g1 is omitted, because it follows by tracing a path through g2 or g3
- This is called a *Hasse diagram*, it depicts a *partially ordered set*, you can find such diagrams in
  - Algebra books
  - Dragon, chapter 9 (when we get there)

Partially ordered sets

- These are defined by
  - A pile of things (the set)
  - Some badly disfigured comparison operator that looks like $\preccurlyeq$, $\subseteq$, or similar, to clarify that it's special (the ordering relation)
- They're *partially* ordered because the relation doesn't have to specify orders between all pairs of things (such as g2, g3)
- *Total* orders (like the usual comparison of real numbers) puts all things in relations to each other, so the Hasse diagram just becomes a long line

The ordering relation

For an order like this to work out, the chosen relation has to be

- Reflexive (things relate to themselves)
  - $x \preccurlyeq x$
- Transitive (relation preserved across in-betweens)
  - $x \preccurlyeq y$ and $y \preccurlyeq z$ means that $x \preccurlyeq z$
- Anti-symmetric (things relate in 1 direction only)
  - $x \preccurlyeq y$ and $y \preccurlyeq x$ means that $x = y$

It's time to stop now

- By continuing to vigorously wave my hands, I could attempt to convince you that when a loop's functional has multiple fixed points, the *shares-effect* relation always creates an order with a unique minimum in it
  - Thereby arguing that the D.S. for *While* is, in fact, a complete specification of the programs that have values
- We have already seen all the parts I wanted to show you, so we can return to Earth and spend our time on looking at them instead

Some properties are computable

It's the list we can do department

- Even though we can't automate the whole business of figuring out what a program computes, there are many semantic properties that match functions we *can* evaluate automatically
  - You've probably noticed that the VSL compiler perfectly well can be programmed to detect uses of undeclared variables, for instance
- Some of those can reveal ways to rewrite the program without altering its meaning

What if, say, x is assigned exactly once?

foo(z)	→	foo(z)
x = 1		
y = x	→	y = 1
z = x+y	→	z = 1+y
bar(z)	→	bar(z)
etc.		

Same program, less space and time

The constant-ness of x

Statement	Is x a constant?
foo(x)	
x=1	
y=x	
x=2	
z=x+y	
bar(z)	

The constant-ness of x

Statement	Is x a constant?
foo(x)	**x is undefined**
x=1	
y=x	
x=2	
z=x+y	
bar(z)	

The constant-ness of x

Statement	Is x a constant?
foo(x)	x is undefined
x=1	**so far, it's 1**
y=x	
x=2	
z=x+y	
bar(z)	

NTNU – Trondheim
Norwegian University of
Science and Technology

The constant-ness of x

Statement	Is x a constant?
foo(x)	x is undefined
x=1	so far, it's 1
y=x	**so far, it's 1**
x=2	
z=x+y	
bar(z)	

The constant-ness of x

Statement	Is x a constant?
foo(x)	x is undefined
x=1	so far, it's 1
y=x	so far, it's 1
x=2	**no, it can't be**
z=x+y	
bar(z)	

The constant-ness of x

Statement	Is x a constant?
foo(x)	x is undefined
x=1	so far, it's 1
y=x	so far, it's 1
x=2	no, it can't be
z=x+y	**no**
bar(z)	

The constant-ness of x

Statement	Is x a constant?
foo(x)	x is undefined
x=1	so far, it's 1
y=x	so far, it's 1
x=2	no, it can't be
z=x+y	no
bar(z)	**no**

The constant-ness of x

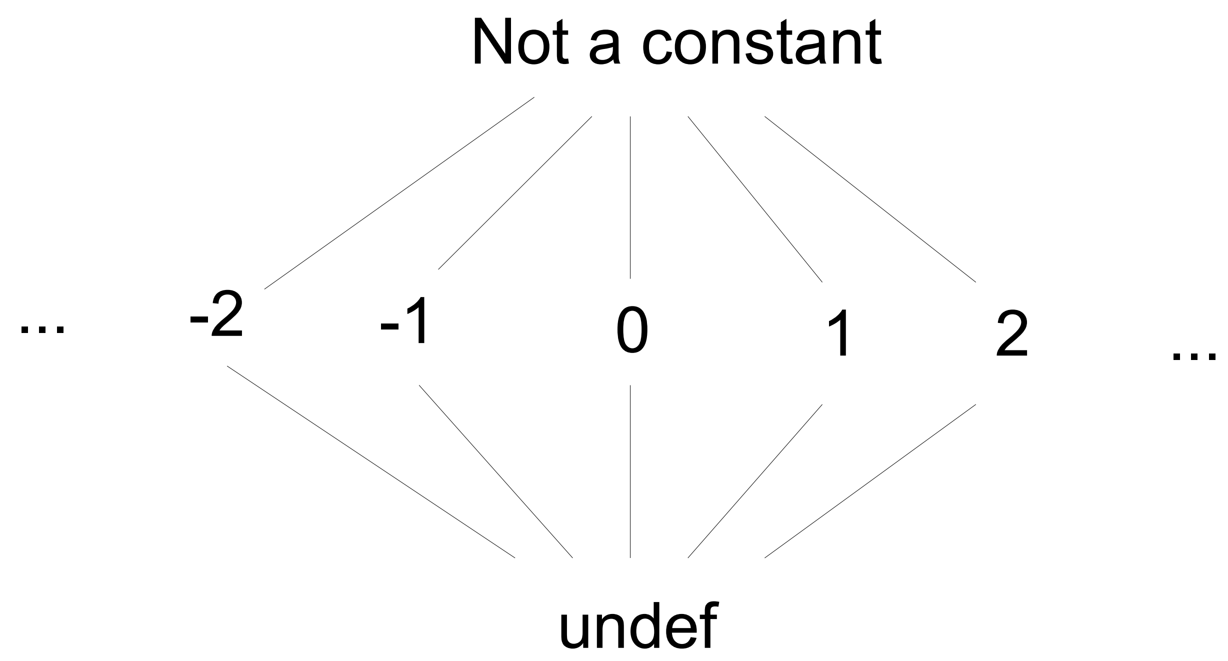
Statement	Is x a constant?
foo(x)	x is undefined
x=1	so far, it's 1
y=x	so far, it's 1
x=2	no, it can't be
z=x+y	no
bar(z)	no
• *Etc.*	**no forever after**

NTNU – Trondheim
Norwegian University of
Science and Technology

An order of constant-ness points

- A) While x is not defined, it is unknown
- B) When x is first defined to 1, it is possibly constant
- C) When x is redefined to 2, it is certainly not constant
- That is...
  - undef $\preccurlyeq$ 1 $\preccurlyeq$ not-a-constant
- If x were first defined to 36, it's also true that...
  - undef $\preccurlyeq$ 36 $\preccurlyeq$ not-a-constant

The Hasse diagram



Monotonicity

- When we evaluate the function of a statement in the constant-ness sense, x either stays where it is, or moves up the order
- It can't be determined as variable and then later return to having a constant value
- A function f of points x, y in the order is *monotone* if it preserves the order of points, that is,
  - If $x \preccurlyeq y$, then $f(x) \preccurlyeq f(y)$ also

NTNU – Trondheim
Norwegian University of Science and Technology

Peeking into the crystal ball

- We're going to capture a few different semantic properties by defining the effect of statements as a function between states that represent
  - Sets of variables that may (or will certainly not) be used again,
  - Sets which map uses to definitions they may (or certainly) match,
  - Sets of expressions that were already evaluated and may (or certainly) still hold the same value,
  - ...and such things

Pleasant properties to look for

- We will define those functions so that they reach a fixed point when they describe the program as accurately as they can
- They will be monotone wrt. a partial ordering of the states, so that they always land somewhere along a path from one end to the other
- The partial orders will have a similar structure to the one we just saw, with a top and a bottom that all paths lead between
  - This guarantees that a monotone function always reaches a fixed point sooner or later – at the top, there is nowhere left to go

This was an attempt at providing “perspective”

- The actual functions and their associated meanings will have to be covered when everyone can be here
- I'll say everything about orders and relations and diagrams over again, with a bit more texture
- Still, it may be easier to recognize them after having thought about it, than to meet them for the first time
- I hope you (will) feel it was worth your time

- Have a peaceful Easter