Compilers Problem Set

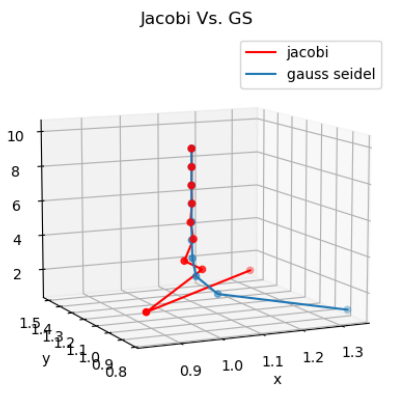
Jeffrey Gordon — jtg2595

February 2023

1 Iterative Solutions of Linear Systems

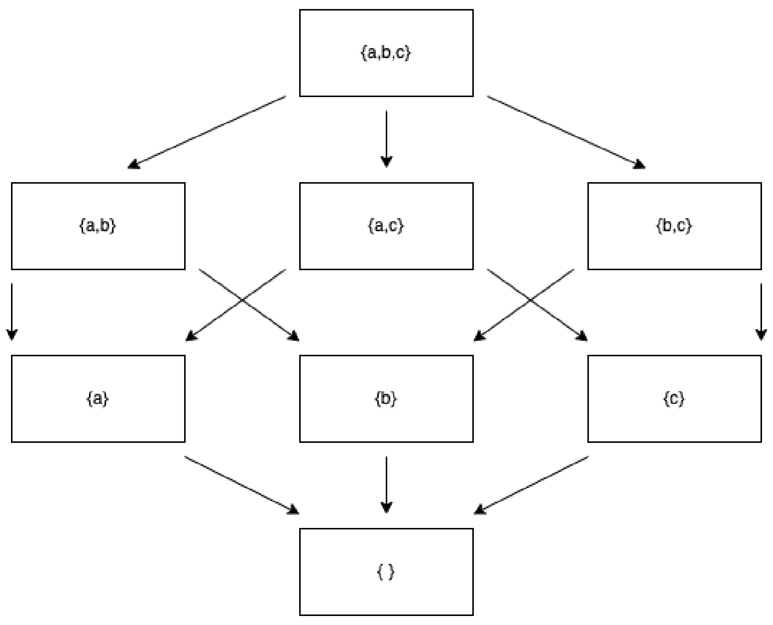
A The Jacobi Method equation is $x^{k+1} = D^{-1}(-Tx^k + b)$. D^{-1} is the inverse of the diagonal matrix of A (the coefficients), $-T$ is the A matrix with the diagonal zeroed out and multiplied by -1, and b is the solution vector. Since the initial estimates are 0, the vector $[0, 0]$ would be x^k when $k = 1$. Then, x^k is now x^{k+1}, and you repeat until convergence. The equation for Gauss-Seidel is $x^{k+1} = (L + D)^{-1}[-Ux^k + b]$. L is the strictly lower triangle of A, D is the diagonal of A, U is the strict upper triangle of A, and b is the solution vector as before. Gauss-Seidel typically converges faster because instead of updating one variable at a time, it always used the most recently updated estimate at all times.

B The plot shows the estimated solution point at each iteration, then the point of convergence happens when the lines start going vertical, or the slope approaches infinity/undefined. As you can see, Jacobi converges at around 7 iterations, but Gauss-Seidel converges at around 4. This is because Gauss-Seidel always uses the most updated estimates of each variable, instead of waiting for another iteration to occur.



2 Powersets

A This is the Hasse Diagram of the partially ordered set.



B Monotonic, Not Extensive: f(x) = x - $\{a\}$

C Extensive, Not Monotonic: if x is empty, f(x) = x'. Else, f(x) = x

D Extensive, Monotonic: f(x) = x

E f(x) = $\{a\}$ - x has no fixpoints. The output will always be $\{a\}$ or {}, and using either of those as input will output the other one.

F f(x) = x has multiple fixpoints. Matter of fact, every input is a fixpoint for this function.

3 Set of Fixpoints

$(L, \leq)$ being a domain is proven by Knaster Tarski. Let $\perp$ denote the least element in the domain. $max\perp, f(\perp), f(f(\perp)), f(f(f(\perp)))$ is proven to be the least fixpoint of the function. By monotonicity, $\perp \leq f(\perp)$. The chain grows until the largest element matches to itself since D is finite. Therefore, the largest element in the chain is a fixpoint of D. Let p be any fixpoint in L. By definition, $\perp \leq$ p. Therefore, $f(\perp) \leq f(p) = p$ due to monotonicity. So the largest element in the chain $\leq p$. The largest element is a fixpoint, so $= p$. So the largest element of the chain is the least fixpoint, proving there's a least element in L. So, $(L, \leq)$ is a valid domain.

4 Monotonicity

Proof that f is monotonic, by contradiction: Let's say that f isn't monotonic. Therefore there is some $T_2 \geq T_1$ such that $f(T_2) < f(T_1)$. Lets say that $T_1 = a, T_2 = a \cup b$. So it holds that $T_2 \geq T_1$. $f(T_1) = T_2$. $f(T_2) = T_2$. Therefore, $f(T_1) = f(T_2)$, which is a contradiction to non monotonicity. Therefore, f is monotonic.

To prove that $g(T) = T - \{a\}$ is monotonic: $f(T_1) = \{\}$. $f(T_2) = \{b\}$. $T_2 \geq T_1, f(T_2) > f(T_1)$. Therefore the function is monotonic. This can also be seen intuitively. If the set doesn't contain a, then the output is the same size as the input. If you add a to the set, the input size increases, but the output size remains the same. This satisfies monotonicity.

5 h(x) Monotonicity

To show that h(x) = f(g(x)) is monotonic: Lets say that $x_1 \leq x_2$. Therefore $g(x_1) \leq g(x_2)$. Lets call $g(x_1) = x_1'$, and $g(x_2) = x_2'$. Therefore $x_1' \leq x_2'$. So $f(x_1') \leq f(x_2')$. So h(x) is monotonic.

6 Monotonicity Extensitivity

A To show that any solution to the system is a solution to x = f(g(x)) and vice versa: Lets say that x_1 is a solution to x = f(x), x = g(x). This means that x_1 is a common fixpoint. Also $f(x_1) = x_1$ and $g(x_1) = x_1$. Therefore $f(g(x_1)) = f(x_1) = x_1$. So $x_1 = x_1$. To show vice versa, let's say that x_1' satisfies x = f(g(x)). Which means $x_1' = f(g(x_1'))$. Lets call $g(x_1')$ x_1''. So $x_1' = f(x_1'')$. $x_1'' \geq x_1'$ by extensitivity. $f(x_1'') \geq x_1''$ by extensitivity. And $f(x_1'') = x_1'$, so $x_1' \geq x_1''$. Therefore $x_1' \geq x_1'' \geq x_1'$. So $x_1' = x_1''$. Therefore x remains unchanged through f and g = fixpoints of f and g, satisfying the system.

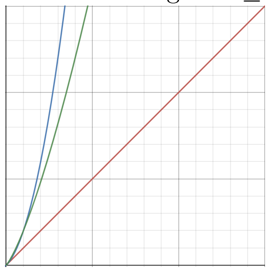
B The previous gives us the following information: h(x) = f(g(x)) is monotonic if f and g are monotonic, and the solution to x = f(x) and x = g(x) are common fixpoints of f and g and is a solution to x = f(g(x)). So, x is a fixpoint of h,f and g. And h,f, and g are monotonic. So by knaster, theres a least fixpoint, which is the least solution to the system. You can also prove this with the chain theorem.

C It depends on if the solution to the system, x, is extensive or not. In order for the property to hold, the solution HAS to be extensive, or else there's no solution.

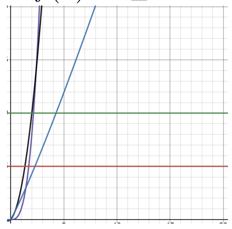
D A would hold, B wouldn't.

7 Real Numbers

A Extensive means that every x is associated with a y value $\geq x$. So, any function that grows $\geq y = x$ is considered extensive.



B Monotonic (Positively) means that the derivative of the function, which is $f(x)'$ is ≥ 0 at every point.



C The fixpoints of f, if they exist, must lie on the line $y = x$.

D $f(x) = x^2 + 5$ has no fixpoints. $f(x) = x^2$ has two fixpoints.

E The hairy ball theorem states that there is no way to comb the hair on a spherical ball without creating at least one point on the ball where the hair sticks up straight. More generally, the theorem applies to any continuous vector field on a sphere, which can be thought of as an infinite number of tiny hairs on the surface of the sphere. The theorem is a consequence of the topology of the sphere and implies that there must always be at least one point on the sphere where the vector field is zero.

F No, monotonicity of a function f is not enough to guarantee that it has a least fixpoint. A counterexample can be given by the function f(x) = x + 1, which is monotonic but does not have a least fixpoint. In general, additional conditions on f or on the domain of f may be needed to guarantee the existence of a least fixpoint. One common condition is completeness of the domain, which means that every non-empty subset of the domain that is bounded above has a least upper bound. If the domain is complete and f is monotonic, then it can be shown that f has a least fixpoint. if D is a complete lattice (a partially ordered set in which every non-empty subset has a least upper bound and a greatest lower bound) and f:D→D is a monotonic function, then f has a least fixpoint and a greatest fixpoint. Moreover, the least fixpoint can be obtained as the least upper bound of the set of all iterates of f, starting from the bottom element of D. So you also need domain completeness.

8 Grammars

1. A language is the set of symbols/strings over an alphabet, and grammar for a language are the rules for generating the strings in a language.

2. Ambiguous grammar is grammar in which there are multiple leftmost derivations for some word. So it could lead to multiple valid parse trees. An example would be:

   S - AB | CB
   A - a
   B - b
   C - a

   Lets say we want to derive the string ”ab”. We can do AB - a - b, or we can do CB - a - b. Since there are two valid paths to take to the string, this is considered ambiguous grammar.

3. No, not all context-free grammars can be parsed using recursive-descent parsing. A simple context-free grammar that cannot be parsed using recursive-descent parsing is:

   S - SS
   S - a|b

   Recursive descent parsing would be stuck in a loop.

4. The CYK algorithm can be used to parse general context free grammar. The CYK algorithm is a dynamic programming algorithm that uses a table to store partial parse results, and builds up the full parse tree by combining smaller parse trees. It works by iteratively applying a set of rules to the table, in a bottom-up fashion, to fill in the entries of the table with the non terminals that can derive the corresponding sub string of the input. The algorithm is based on a modification of the Chomsky normal form (CNF), which is a way of rewriting a context-free grammar so that every production rule has either a single terminal symbol on the right-hand side, or two non terminal symbols on the right-hand side.

9 FIRST, FOLLOW

i. FIRST - If a string is broken down to its form with all terminals, then FIRST is the set of all the possible first k terminals in the string. For example if its FIRST1, and the possible broken down strings are abc and bca, the set would be (a,b), since either a or b could be first. FOLLOW - All the possible first k terminals that could be right after the non terminal in question. For example if the grammar is S-AaB, B-Ad, FOLLOW1(A) = (a,d). To do FOLLOW2(A), youd have to break down aB, then take the first two terminals. Then union that with d. The intuition behind the equations is that you are computing FIRST and FOLLOW for the possible productions (separated by |) of the non terminal, then unioning them. Computing the NULLABLE set can be computationally expensive. The FIRST and FOLLOW sets can help reduce the need for computing the NULLABLE set by providing information about which symbols can appear at the beginning and end of nonterminal symbols, respectively. Consider a nonterminal symbol A and a production rule of the form A - BC, where B and C are nonterminal symbols. We want to determine whether this production rule can derive the empty string. To do this, we can look at the FOLLOW set of B and the FIRST set of C. If the empty string is in the FOLLOW set of B and the empty string is in the FIRST set of C, then the production rule A - BC can derive the empty string, and we can add A to the NULLABLE set. By using the FOLLOW and FIRST sets in this way, we can determine which production rules can derive the empty string without explicitly computing the NULLABLE set.

ii.
FIRST2(S) = FIRST2(yLab) ∪ FIRST2(yLbc) ∪ FIRST2(M)
FIRST2(L) = FIRST2(a) ∪ FIRST2(ϵ)
FIRST2(M) = FIRST2(MM) ∪ FIRST2(x)

FOLLOW2(S) = \$
FOLLOW2(L) = FIRST2(ab) + FOLLOW2(S) ∪ FIRST2(bc) + FOLLOW2(S)
FOLLOW2(M) = FOLLOW2(S) ∪ FIRST2(M) + FOLLOW2(M) ∪ FOLLOW2(M)

Solutions:
FIRST2(S):
FIRST2(yLab) = ya, yϵ
FIRST2(yLbc) = ya, yϵ
FIRST2(M) = x, xx

FIRST2(S) = ya, yϵ, xx, x

FIRST2(L):
FIRST2(a) = a
FIRST2(ϵ) = ϵ

FIRST2(L) = a, ϵ

FIRST2(M):
FIRST2(MM) = xx
FIRST(x) = x

FIRST2(M) = x, xx

FOLLOW2(S): $$

FOLLOW2(L): ab, bc

FOLLOW2(M) = $, x$, xx