Evaluation/Interpolation (I)

Lecture 6

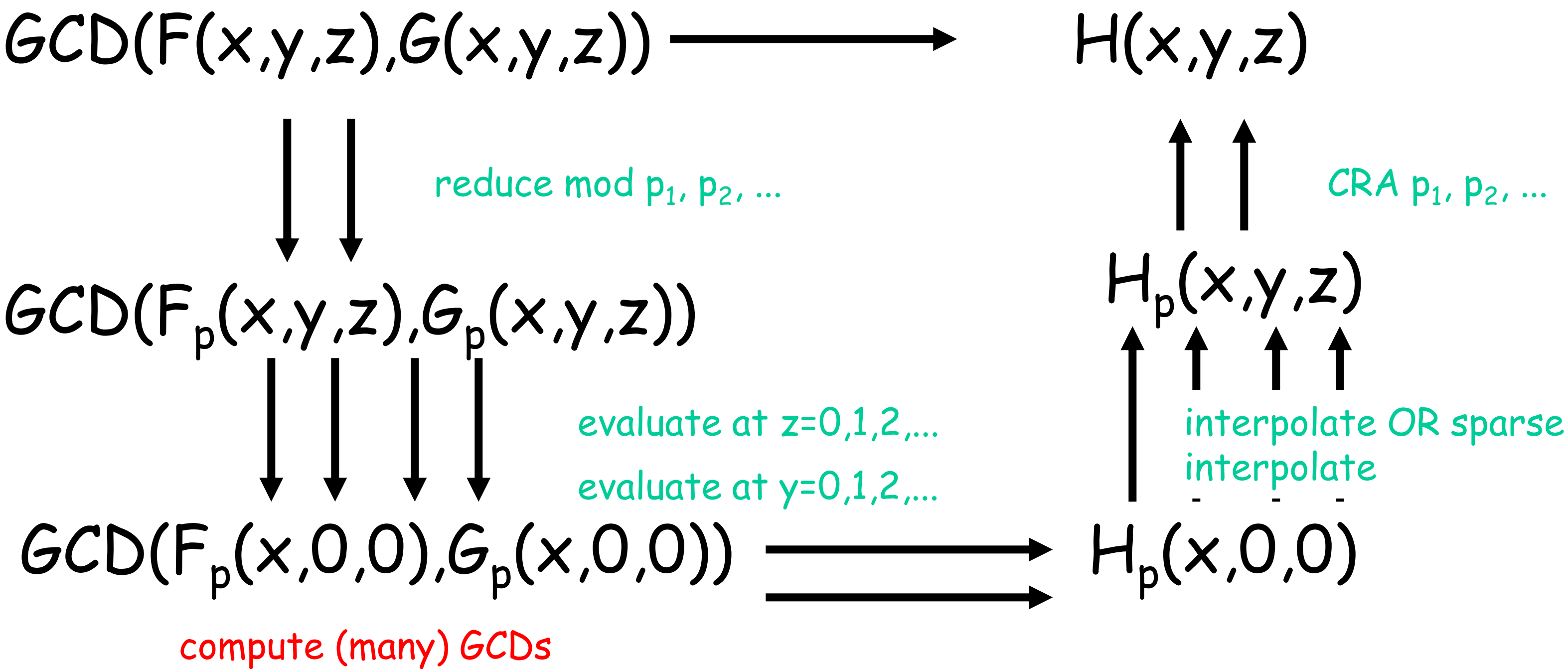
Backtrack from the GCD ideas a bit

- We can do some operations faster in a simpler domain, even if we need to do them repeatedly

$$
\begin{tikzcd}
\mathrm{GCD}(F(x,y,z),G(x,y,z)) \arrow[rr] \arrow[d, shift right] \arrow[d, shift left] & & H(x,y,z) \\
\mathrm{GCD}(F_p(x,y,z),G_p(x,y,z)) \arrow[d, shift right=3] \arrow[d, shift right=1] \arrow[d, shift left=1] \arrow[d, shift left=3] & & H_p(x,y,z) \arrow[u, shift right] \arrow[u, shift left] \\
\mathrm{GCD}(F_p(x,0,0),G_p(x,0,0)) \arrow[rr, shift left] \arrow[rr, shift right] & & H_p(x,0,0) \arrow[u, shift right=3] \arrow[u, shift right=1] \arrow[u, shift left=1] \arrow[u, shift left=3]
\end{tikzcd}
$$

Backtrack from the GCD ideas a bit

- Some of the details



How does this work in some more detail

- How many primes p_1, p_2, ...?
  - Bound the coeffs by $p_1*p_2*...p_n$? (or +/- half that)
    - bad idea, the bounds are poor. given ||f|| what is ||h|| where h is factor of ||f|| ?
    - Try an answer and test to see if it divides?
    - See if WHAT divides?
      - compute cofactors A, B, A*H=F, B*H=G, and when A*H= F or B*H=G, you are done.
  - The process doesn't recover the leading coefficient since F modulo p etc might as well be monic.
  - The inputs F and G are assumed primitive; restore the contents.
  - There may be unlucky primes or evaluation points.

Chinese Remainder Theorem

- (Integer) Chinese Remainder Theorem:
- We can represent a number x by its remainders modulo some collection
- of relatively prime integers n_1 n_2 n_3...
- Let $N = n_0 * n_1 * \ldots * n_k$. Then the Chinese Remainder Thm. tells us that we can represent any number x in the range $-(N-1)/2$ to $+(N-1)/2$ by its residues modulo $n_0, n_1, n_2, \ldots, n_k$. {or some other similar sized range, 0 to N-1 would do}

Chinese Remainder Example

Example $n_0=3$, $n_2=5$, $N=3*5=15$

x	x mod 3	x mod 5
-7	-1	-2 note: if you hate balanced notation -7+15=8. mod 3 is 2->-1
-6	0	-1
-5	1	0
-4	-1	1
-3	0	2
-2	1	-2 note: x mod 5 agrees with x, for small x 2 [-2,2], +-(n-1)/2
-1	-1	-1 note:
0	0	0 note:
1	1	1 note:
2	-1	2
3	0	-2
4	1	-1
5	-1	0 note: symmetry with -5
6	0	1
7	1	2 note: also 22, 37, ... and -8, ...

Conversion to modular CRA form

Converting from normal notation to modular representation is easy in principle;

you do remainder computations (one instruction if x is small, otherwise a software routine simulating long division)

Conversion to Normal Form (Garner's Alg.)

Converting to normal rep. takes k^2 steps.

Beforehand, compute

inverse of n_1 mod n_0, inverse of n_2 mod n_0*n_1, and also the products n_0*n_1, etc.

Aside: how to compute these inverses:

These can be done by using the Extended Euclidean Algorithm.

Given $r= n_0$, $s= n_1$, or any 2 relatively prime numbers, EEA computes a, b such that $a*r+b*s=1 = \gcd(r,s)$

Look at this equation mod s: $b*s$ is 0 (s is 0 mod s) and so we have a solution $a*r=1$ and hence a = inverse of r mod s. That's what we want. It's not too expensive since in practice we can precompute all we need, and computing these is modest anyway. (Proof of this has occupied quite a few people..)

Conversion to Normal Form (Garner's Alg.)

Here is Garner's algorithm:

Input: x as a list of residues $\{u_i\}$: $u_0 = x \bmod n_0$, $u_1 = x \bmod n_1$, ...

Output: x as an integer in $[-(N-1)/2,(N-1)/2]$. (Other possibilities include x in another range, also x as a rational fraction!)

Consider the mixed radix representation

$$x = v_0 + v_1*n_0 + v_2*(n_0*n_1)+ \ldots +v_k*(n_0*\ldots n_{k-1}) \qquad [G]$$

if we find the v_i, we are done, after k more mults.

Conversion to Normal Form (Garner's Alg.)

$x = v_0 + v_1{*}n_0 + v_2{*}(n_0{*}n_1) + \ldots + v_k{*}(n_0{*}\ldots n_{k-1})$ [G]

It should be clear that

$v_0 = u_0$, each being x mod n_0

Next, computing remainder mod n_1 of each side of [G]

$u_1 = v_0 + v_1{*}n_0$ mod n_1, with everything else dropping out

so $v_0 = (u_1 - v_0){*}\ n_0^{-1}$ all mod n_1

in general,

$v_k = (\ u_k - [v_0 + v_1{*}n_0 + \ldots + v_{k-1}{*}\ (n_0 \ldots n_{k-2})\ ])\ {*}\ (n_0{*}\ldots n_{k-1})^{-1}$ mod n_k. mod n_k

Note that all the v_k are "small" numbers and the items in green are precomputed.

Conversion to Normal Form (Garner's Alg.)

Note that all the v_k are "small" numbers and the items in green are precomputed.

Cost: if we find all these values in sequence, we have k^2 multiplication operations, where k is the number of primes needed. In practice we pick the k largest single-precision primes, and so k is about 2* log (SomeBound/2^{31})

Interpolation

- Abstractly THE SAME AS CRA
  - change primes $p_1, p_2, \ldots$ to $(x-x_1) \ldots$
  - change residues $u_1 = x \bmod p_1$ for some integer x to $y_k=F(x_k)$ for a polynomial F in one variable
  - eh, we don't usually program it that way, though.

To factor a polynomial h(x)

- Evaluate h(0); find all factors. $h_{0,0}$, $h_{0,1}$ …
  - E.g. if h(0)=8, factors are –8,-4,-2,-1,1,2,4,8
- Repeat … until
- Factor h(n)
- Find a factor: What polynomial f(x) assumes the value $h_{0,k}$ at 0, $h_{1,j}$ at 1, …. ?
- Questions:
  - Does this always work?
  - What details need to be resolved?
  - How much does this cost?