Resolving Zero-Divisors using Hensel Lifting

John Kluesner and Michael Monagan

Department of Mathematics
Simon Fraser University
Burnaby, Canada

September 23 2017

Motivation From Algebraic Number Theory

Consider an algebraic number field $\mathbb{Q}(\alpha)$ with minimal polynomial $t(x)$. Let $a, b \in \mathbb{Q}(\alpha)[x]$. How should be compute $\gcd(a, b)$?

1. Use the Euclidean algorithm.
   - The polynomials in the remainder sequence have large integer coefficients.
   - Inefficient!
2. Researchers developed modular algorithms.
   - This is a divide and conquer algorithm.
   - Langemyr and McCallum solved the algebraic integer case.
   - Encarnacion solved the algebraic number field case.
   - Monagan and van Hoeij solved the multiple extension case.

Encarnacion's Algorithm

Input: $a, b \in \mathbb{Q}(\alpha)[x]$ with $t(\alpha) = 0$.
Output: $\gcd(a, b)$.

1. Pick a prime p that satisfies
   - $\text{lc}(a)\text{lc}(b)\text{den}(a)\text{den}(n)\text{den}(t) \not\equiv 0 \pmod{p}$
   - $t(x)$ is square-free modulo p
2. Compute $g_p := \gcd(a, b) \pmod{p}$ using EA
   - If a zero-divisor is encountered, pick a new prime.
3. Combine gcds
   - Combine all g_p of lowest degree using CRT and RR and store in $g \in \mathbb{Q}(\alpha)[x]$.
4. Division Test
   - If $g \mid a$ and $g \mid b$, output g.

Triangular Sets

- Encarnacion's problem solves the problem of computing gcds over $\mathbb{Q}[z]/\langle t(z)\rangle[x]$ where t is irreducible.
- What if t were possible reducible? Or more generally, we had multiple extensions?
- $T = \{t_1(z_1), t_2(z_1, z_2), \ldots, t_n(z_1, \ldots, z_n)\}$ where each t_i is possibly reducible.
- T is called a triangular set.
- This is the problem we solved.

Troublesome Example: Why is this hard?

- $T = \{z_1^2 + 1, z_2^2 + 1\}$ and $R = \mathbb{Q}[z_1, z_2]/T$.
- Note that $z_1 - z_2$ and $z_1 + z_2$ are zero-divisors in R.
- $a = x^4 + (z_1 + 18\,z_2)\,x^3 + (-z_2 + 3\,z_1)\,x^2 + 324\,x + 323$
- $b = x^3 + (z_1 + 18\,z_2)\,x^2 + (-19\,z_2 + 2\,z_1)\,x + 324$

EA over $\mathbb{Q}$

- $r_0 := a$, $r_1 := b$
- $r_2 = (z_1 + 18z_2)x^2 + 323$
- $r_3 = (z_1 - z_2)x + 1$
- Terminate with an error since $z_1 - z_2$ is a zero-divisor.

EA mod primes

- In $\mathbb{Z}_{11}$, we terminate with the modular image of $z_1 - z_2$
- In $\mathbb{Z}_{17}$, we terminate earlier since $z_1 + 18z_2$ is a zero-divisor
- The CRT and RR can NOT combine these into a zero-divisor over $\mathbb{Q}$

Objective

- Let T be a radical triangular set of $\mathbb{Q}[z_1, \ldots, z_n]$ and $R = \mathbb{Q}[z_1, \ldots, z_n]/T$.
  - That is, $T = \{t_1(z_1), t_2(z_1, z_2), \ldots, t_n(z_1, \ldots, z_n)\}$ where t_i is monic in z_i, and $\langle T \rangle$ is a radical ideal.
- Our goal is to create a technique for resolving zero-divisors so that we may create efficient modular algorithms for computation modulo T.
- This paper uses Hensel lifting to resolve zero-divisors for a modular gcd algorithm in $R[x]$.

Known algorithms

Please see the article "Computations Modulo Regular Chains" by Xin Li, Marc Moreno Maza, and Wei Pan in Proceedings of ISSAC '09, pp. 239–246, 2009.

- The `RegularChains` package uses subresultant-based algorithms.
- The last non-zero subresultant of a and b will be a $\gcd(a, b)$.
- This isn't a fully modular algorithms. They don't have to worry about zero-divisors modulo primes.

Notation

- $T := \{t_1, t_2, \ldots, t_n\}$ is a radical triangular set over $\mathbb{Q}$.
- $R := \mathbb{Q}[z_1, \ldots, z_n]/T$.
- $T_k := \{t_1, \ldots, t_k\}$.
- p will be a prime number.

Definitions

Definition

A prime number p is a radical prime if T mod p remains radical.

Definition

Let $a, b \in R[x]$ and $g = \gcd(a, b)$. A prime number p is an *unlucky prime* if $g \neq \gcd(a, b) \pmod{p}$. A prime number p is *bad* if $\mathrm{den}(a)\mathrm{den}(b)\mathrm{lc}(a)\mathrm{lc}(b) \equiv 0 \pmod{p}$.

Theorems 5, 8

Only finitely many primes are unlucky, nonradical, or bad.

Hensel Lifting

- We will be resolving zero-divisors using Hensel lifting.
- This is used to compute factorizations of polynomials.
- **Input:** $f \in R[x]$ and $a_0, b_0 \in R/p[x]$ where p is a radical prime and $f = a_0 b_0 \pmod{p}$ and $\gcd(a_0, b_0) = 1 \pmod{p}$.
- The Hensel construction computes $a_k, b_k \in R/p^{k+1}[x]$ where $f \equiv a_k b_k \pmod{p^{k+1}}$, $a_k \equiv a_0 \pmod{p}$, and $b_k \equiv b_0 \pmod{p}$.
    - Use rational reconstruction on a_k. If succesful, store result as u.
    - If $u \mid f$ over R, terminate and output u, f/u.
- Continue until $p^{k+1} \geq B$ where B is a bound on integer coefficients of any monic factorization of f.
    - If the bound is reached, output with failure.

Modular Algorithm Framework

- Let ALGO be a modular algorithm that may encounter a zero-divisor modulo a prime p.
  - ALGO could be a modular gcd algorithm, an inversion algorithm, or a matrix inversion algorithm, for instance.
- If ALGO encounters a zero-divisor mod a prime, lift it from $\mathbb{Z}_p$ to $\mathbb{Q}$ using Hensel lifting.
  - Hensel lifting succeeds $\Longrightarrow$ gives a factorization $t_k = uv \pmod{T_{k-1}}$, so split T into T_u, T_v where t_k is replaced by u and v, respectively.
  - Hensel lifting fails $\Longrightarrow$ pick a new prime.
  - Hensel lifting encounters a new zero-divisor $\Longrightarrow$ resolve that instead.
- If ALGO doesn't encounter a zero-divisor, let ALGO continue.

ModularC-GCD

Input: Polynomials $a, b \in R[x]$. **Output:** $\gcd(a, b)$.

- Pick a new prime p that is not bad.
- Test if p is a radical prime.
  - If a zero-divisor is encountered, resolve it using Hensel lifting.
  - If p is not radical, pick a new prime. Otherwise, continue as p is a radical prime.
- Use the monic Euclidean algorithm to compute $\gcd(a, b) \pmod{p}$.
  - If a zero-divisor is encountered, resolve it using Hensel lifting.
  - Combine all gcds computed modulo primes of lowest degree using Chinese remaindering and rational reconstruction into a polynomial h over $\mathbb{Q}$.
  - Test if $h \mid a$ and $h \mid b$ over $\mathbb{Q}$. If the division test succeeds, return h. Otherwise, we need more primes, so pick a new one.

Proof of Correctness for ModularC-GCD

Proposition 9

A finite number of zero-divisors are encountered when running ModularC-GCD(a, b).

Lemma 11

Suppose $f, u \in R[x]$ are monic such that $u \mid f$. Let

$$S = \{\text{prime numbers } p \in \mathbb{Z} : p \text{ isn't radical or divides } \mathrm{den}(f)\}.$$

Then, $u \in \mathbb{Z}_S[z_1, \ldots, z_n, x]/T$.

Theorem 12

ModularC-GCD(a, b) returns a gcd(a, b).

Implementation

- We have implemented all algorithms in Maple's RECDEN package.
- We compare our implementation with the procedure `RegularGcd` from the `RegularChains` package in Maple.

Time Complexity

- Let M be the number of primes needed by ModularC-GCD.
- The gcd computation modulo primes costs an expected $O(M\deg(a)\deg(b))$ operations in $\mathbb{Z}_p[z_1,\ldots,z_n]/T$.
- The division test over $\mathbb{Q}[z_1,\ldots,z_n]/T$ costs an expected $O(\deg(a)\deg(b))$ operations in $\mathbb{Q}[z_1,\ldots,z_n]/T$.
- Note: we've done a more proper time complexity analysis, but there's not enough room in the paper.

Timings for GCDs

Input: Polynomials a, b, g with degrees $6, 5, 4$. We compute $\gcd(ag, ab)$. All polynomials and extensions are dense.

extension		ModularC-GCD			RegularGcd		
n	degrees	time	divide	#primes	time real	cpu	#terms
1	[4]	0.013	0.006	3	0.064	0.064	170
2	[2, 2]	0.029	0.022	3	0.241	0.346	720
2	[3, 3]	0.184	0.138	17	1.73	4.433	2645
3	[2, 2, 2]	0.218	0.204	9	10.372	29.357	8640
2	[4, 4]	0.512	0.391	33	12.349	40.705	5780
4	[2, 2, 2, 2]	1.403	1.132	33	401.439	758.942	103680
3	[3, 3, 3]	2.755	1.893	65	413.54	1307.46	60835
3	[4, 2, 4]	1.695	1.233	33	39.327	86.088	19860
1	[64]	6.738	5.607	65	43.963	160.021	3470
2	[8, 8]	13.321	11.386	129	1437.76	5251.05	30420
3	[4, 4, 4]	17.065	14.093	129	7185.85	22591.4	196520

All timings are in seconds.

Timings for GCDs, continued

Input: Polynomials a, b, g with degrees $6, 5, 4$. We compute $\gcd(ag, ab)$. All polynomials and extensions are dense, but g is monic.

extension		ModularC-GCD			RegularGcd		
n	degrees	time	divide	#primes	time real	cpu	#terms
1	[4]	0.01	0.006	2	0.065	0.065	170
2	[2, 2]	0.02	0.016	2	0.238	0.329	715
2	[3, 3]	0.048	0.041	2	1.771	4.412	2630
3	[2, 2, 2]	0.05	0.041	2	11.293	31.766	8465
2	[4, 4]	0.077	0.068	2	11.521	36.854	5750
4	[2, 2, 2, 2]	0.117	0.097	2	321.859	431.368	99670
3	[3, 3, 3]	0.222	0.201	2	508.465	1615.28	57645
3	[4, 2, 4]	0.05	0.032	2	34.358	71.351	16230
1	[64]	0.304	0.282	2	27.55	98.354	3450
2	[8, 8]	0.482	0.455	2	1628.7	5979.51	29505
3	[4, 4, 4]	0.525	0.477	2	2989.18	4751.04	192825

All timings are in seconds.

Input: Polynomials a, b, g with degrees $9, 8, 4$. We compute $\gcd(ag, ab)$. All polynomials and extensions are dense.

extension		ModularC-GCD			RegularGcd		
n	degrees	time	divide	#primes	time real	cpu	#terms
1	[4]	0.021	0.011	5	0.124	0.13	260
2	[2, 2]	0.043	0.031	5	0.968	1.912	1620
2	[3, 3]	0.214	0.163	17	10.517	34.513	6125
3	[2, 2, 2]	0.287	0.204	9	64.997	173.53	29160
2	[4, 4]	0.638	0.427	33	67.413	245.789	13520
4	[2, 2, 2, 2]	2.05	1.613	33	2725.13	3528.41	524880
3	[3, 3, 3]	3.35	2.731	33	3704.61	11924.0	214375
3	[4, 2, 4]	2.399	1.793	33	334.201	869.116	68940
1	[64]	10.097	8.584	65	171.726	658.518	5360
2	[8, 8]	21.890	18.086	129	10418.4	38554.9	72000
3	[4, 4, 4]	37.007	31.369	129	$>$ 50000	–	–

All timings are in seconds.

Summary

In summary, we

- Developed a new technique for handling zero-divisors modulo triangular sets.
- Used this technique to create an efficient modular algorithm.
- Implemented and analyzed the algorithms.
- Compared them with procedures implemented in Maple.
- Concluded that our new algorithms are much more efficient.

Conclusion

What still needs to be done?

- Multivariate polynomial gcd computation in $\mathbb{Q}[x_1, \ldots, x_m][z_1, \ldots, z_n]/T$ where T is a triangular set of $\mathbb{Q}[z_1, \ldots, z_n]$.
- A bound on monic factors for the Hensel lifting.
  - Currently, we use an "engineering"-esque approach with a bound B that grows at every function call of the Hensel lifting procedure.
- Optimizations to the division test.

Conclusion

We have shown that the technique of using Hensel lifting for modular algorithms is one worth pursuing and should allow for more modular algorithms to be created.

Examples of more ideas for modular algorithms:

- Inversion in R,
- Triangular Decomposition of T,
- Resultant computation modulo T using a Euclidean algorithm,
- Matrix Inversion,
- Linear System Solver.

Thank you!