Karatsuba's algorithm for polynomials over rings without decidable equality

Max Zeuner

Stockholm University, Stockholm, Sweden
zeuner@math.su.se

Karatsuba's algorithm, a divide and conquer algorithm for fast multiplication [5], can also be used to compute the product of two polynomials. Formally verified versions of this algorithm for polynomials have been implemented in several proof assistants by now. This includes an implementation in `Coq`'s `CoqEAL` library [3], as well as a more recent implementation in `Agda`'s `DoCon-A` library [7]. In both of these libraries, polynomials are essentially implemented as lists of coefficients with a proof that the last coefficient is non-zero. Such an implementation, however, can only be equipped with a ring structure if the base ring has decidable equality, since one needs to be able to decide whether an element in the list of coefficients is zero or not.[1]

We want to demonstrate that with the help of higher-inductive types (HIT) we can give a formally verified implementation of Karatsuba's algorithm that works for polynomials over arbitrary rings.[2] To this end we use `Cubical Agda`, a recent extension of `Agda` with fully computational support for HITs. `Cubical Agda`'s library has several implementations of polynomials that do not require the base ring to have a decidable equality. One of these uses a simple HIT to represent polynomials as lists of coefficients:

```
data R[X] : Type where
  []    : R[X]
  _::_  : R → R[X] → R[X]
  drop0 : 0r :: [] ≡ []
```

Here 0r is the zero element of the base ring R. Points of this HIT are thus lists of coefficients and the higher constructor drop0 equates the singleton list [0r] with the empty list []. It turns out that this is enough to equate lists up to trailing zeros (representing the same polynomial), allowing one to construct the ring structure on R[X] and prove its universal property [9].

Karatsuba's algorithm *can* actually be done for polynomials over arbitrary rings (with or without decidable equality) [1, Sec. 6.1]. Given a polynomial $p(X) = a_0 + a_1X + \cdots + a_nX^n$, we can split it into coefficients of even and odd degree as polynomials $p_e(X) = a_0 + a_2X + \ldots$ and $p_o(X) = a_1 + a_3X + \ldots$ and rewrite our polynomial as $p(X) = p_e(X^2) + X \cdot p_o(X^2)$. With this we can write the product of two polynomials as

$$pq = p_eq_e(X^2) + X \cdot \big((p_e + p_o)(q_e + q_o) - p_eq_e - p_oq_o\big)(X^2) + X^2 \cdot p_oq_o(X^2)$$

Writing the multiplication this way, we only have to evaluate three multiplications recursively, namely p_eq_e, p_oq_o and $(p_e + p_o)(q_e + q_o)$. Note that all of these multiplications take inputs of roughly half the size of the original p and q. Where the naïve algorithm needs four multiplications and runs in $\mathcal{O}(n^2)$ time (n being degree of the polynomials involved), Karatsuba's algorithm thus runs in $\mathcal{O}(n^{\log_2 3}) \approx \mathcal{O}(n^{1.585})$ time.

[1] Naturally, one would like to work constructively to ensure computational meaning. In this constructive setting one might also be able to consider rings with an inequality relation [8].

[2] The formalization can be found here: https://github.com/mzeuner/cubical/blob/Karatsuba/Cubical/Algebra/Polynomials/UnivariateList/Karatsuba.agda

As the splitting of the polynomials is non-trivial, `Agda`'s termination-checker will not recognize the arguments in the recursive calls to be structurally smaller than the original input. To this end, one can introduce an additional natural number argument, often called the *fuel*. In each recursive call the fuel is decreased by one until it reaches zero, at which point one just performs naïve multiplication. Now, we can prove the algorithm correct regardless of the fuel level. In the case of our HIT polynomials we even need recursive calls of this correctness proof to fill in paths arising from the presence of drop0 in the definition of the algorithm. We thus arrive at a *mutually recursive* definition of the algorithm with fuel and its correctness proof:

```
karatsubaRec : ℕ → R[X] → R[X] → R[X]
karatsubaRec≡ : ∀ (n : ℕ) (p q : R[X]) → karatsubaRec n p q ≡ p · q
```

The next step is to find a lower bound for the fuel, so that all the recursive calls can be made before the fuel reaches zero.[3] One such bound is the maximum of the degrees of the polynomials that are being multiplied. Unfortunately, if the base ring doesn't have decidable equality we cannot define the degree of a polynomial. In our HIT case we cannot even speak of the "length" of a list of coefficients as drop0 requires [0r] and [] to be assigned the same value (up to path). We can however define a "truncated length" function mapping into the *propositional truncation* of naturals:

```
truncLength : R[X] → ‖ ℕ ‖
truncLength [] = | 0 |
truncLength (a :: p) = map suc (truncLength p)
truncLength (drop0 i) = squash | 1 | | 0 | i
```

Note that the value of truncLength still carries computational information, even if the codomain is a proposition. To leverage this information we can observe that the correctness proof karatsubaRec≡ implies that karatsubaRec n is the same operation (namely multiplication) for any $n : \mathbb{N}$. The type of operations R[X] → R[X] → R[X] is a set and by a result due to Kraus [6] this lets us factor karatsubaRec through the canonical map $\mathbb{N} \to \| \mathbb{N} \|$. We thus get a map

```
karatsubaTruncRec : ‖ ℕ ‖ → R[X] → R[X] → R[X]
```

such that karatsubaTruncRec $|n|$ p q = karatsubaRec n p q definitionally for all $n : \mathbb{N}$ and $p\ q$: R[X]. With this we can define our algorithm with an appropriate bound for the fuel that will compute properly in the case of lists of coefficients not containing drop0:

```
karatsuba : R[X] → R[X] → R[X]
karatsuba p q = let fuelBound = map2 max (truncLength p) (truncLength q)
                in karatsubaTruncRec fuelBound p q
```

As mentioned before, this algorithm works for any ring R and the methods described here could be useful more generally for a library of verified computer algebra algorithms for rings without decidable equality. A natural next step would be to compile this code to a runable program. This can be done using `Agda`'s `--erased-cubical` flag [4]. One would like to make sure that the rather complicated paths used in the correctness proof do not interfere with the program at runtime. To this end all HITs involved would need to have erased higher constructors [2], which will result in quite a bit of rewriting of the original code.

[3] Observe that we can not prove a bound for the fuel in this setting. This would essentially require complexity theoretic reasoning, really extending the scope of the simple algebraic correctness proof presented here.

References

[1] Jounaidi Abdeljaoued and Henri Lombardi. *Méthodes matricielles - Introduction à la complexité algébrique*. Springer Science & Business Media, 2004.

[2] Andreas Abel, Nils Anders Danielsson, and Andrea Vezzosi. Compiling Programs with Erased Univalence. unpublished preprint, 2023. https://www.cse.chalmers.se/~nad/publications/abel-danielsson-vezzosi-erased-univalence.pdf.

[3] Maxime Dénès, Anders Mörtberg, and Vincent Siles. A Refinement-Based Approach to Computational Algebra in Coq. In Lennart Beringer and Amy Felty, editors, *Interactive Theorem Proving*, volume 7406 of *Lecture Notes in Computer Science*, pages 83–98. Springer Berlin Heidelberg, 2012.

[4] Agda developers. Agda documentation. https://agda.readthedocs.io/en/latest/, 2023.

[5] Anatolii Alekseevich Karatsuba and Yu P Ofman. Multiplication of many-digital numbers by automatic computers. In *Doklady Akademii Nauk*, volume 145, pages 293–294. Russian Academy of Sciences, 1962.

[6] Nicolai Kraus. The General Universal Property of the Propositional Truncation. In Hugo Herbelin, Pierre Letouzey, and Matthieu Sozeau, editors, *20th International Conference on Types for Proofs and Programs (TYPES 2014)*, volume 39 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 111–145, Dagstuhl, Germany, 2015.

[7] Sergei D Meshveliani. A Certified Program for the Karatsuba Method to Multiply Polynomials. *Programming and Computer Software*, 48(1):1–18, 2022.

[8] Ray Mines, Fred Richman, and Wim Ruitenburg. *A course in constructive algebra*. Springer Science & Business Media, 2012.

[9] Carl Åkerman Rydbeck. Formalisation of Polynomials in Cubical Type Theory Using Cubical Agda. Bachelor's thesis, Stockholm University, 2022. http://urn.kb.se/resolve?urn=urn:nbn:se:su:diva-207780.