A Macaulay2-Lean interface for proofs in Lean

Matthew Ballard[1][0000-0001-5819-0159], Anton Leykin[2][0000-0002-9216-3514], Michael E. Stillman[3][0000-0003-0078-3116], Damiano Testa[4][0009-0004-5949-6861], Douglas A. Torrance[5,2][0000-0003-3999-4973], and Jay Yang[6][0000-0001-8366-423X]

[1] University of South Carolina, Columbia SC 29208, USA ballard@math.sc.edu
[2] Georgia Institute of Technology, Atlanta GA 30332, USA leykin@math.gatech.edu dtorrance9@gatech.edu
[3] Cornell University, Ithaca NY 14850, USA mes15@cornell.edu
[4] University of Warwick, Coventry CV4 7AL, UK D.Testa@warwick.ac.uk
[5] Piedmont University, Demorest GA 30535
[6] Vanderbilt University, Nashville TN 37235 jay.k.yang@vanderbilt.edu

Abstract. Our project aims to build tools that ease the use of Macaulay2, and potentially other computer algebra systems, with Lean. Our current work focuses on proving statements over commutative rings by using Macaulay2 to construct certificates. This initial work focus on proving vanishing of polynomial expressions using ideal membership computations in Macaulay2. This extended abstract discusses our approach and strategy along with current progress and future goals.

Keywords: Lean · Macaulay2 · Commutative Algebra · Computer Algebra System.

1 Introduction

There has been a long history of efforts to build bridges between computer algebra systems and theorem provers. This work has included many different theorem provers and computer algebra systems [7,4]. Additionally, there have been attempts to build frameworks for specifying interfaces between computer algebra systems and theorem provers [1]. Attempts specific to Lean include the now-defunct `polyrith` tactic [2], as well as more recent efforts in [13]. We also point out previous work in formalizing Gröbner bases in Isabelle/HOL [9].

Inspired by such connections and the increasing relevance of automated theorem proving, we aim to build a bridge between Macaulay2 [6] and Lean [11] with a particular emphasis on reasoning about algebraic expressions via polynomial computations. The work represented in this extended abstract is ongoing. However, this paper presents the current state of the implementation as well as future goals.

Our communications protocol is built on well developed interfaces including a variation of the protocol used by LSP [10], and the use of the MRDI format [5] to store serialized mathematical objects. Our current implementation is built for ease of implementation and so reuses significant components of the `grind` tactic infrastructure. However, as we move towards a more mature implementation, we are re-implementing those parts for which optimizations specific to our goals are required.

2 Design

As compared to the recent work in [13], our goal is not to represent the full Macaulay2 language in Lean, but rather to build an interface and framework to allow passing specific computational problems and certificates to and from Macaulay2. Moreover, as we serialize and de-serialize problems rather than Macaulay2 expressions, we do not need to carefully specify either the Macaulay2 language in Lean or vice-versa. Instead, the serialization and de-serialization steps

handle the interface between the languages. While our interface requires each individual computational problem of interest to be specified, there are advantages in the relative robustness of the interface between the languages to changes on either side.

The types of expressions that we aim to consider are what we will call "polynomial expressions". In this case this means expressions in Lean that are a combination of addition, multiplication, and positive integer exponents over a commutative ring. Equivalently, each of the expressions we consider should be polynomials over some set of variables in the context of the expression. Phrasing in terms of polynomial expressions means that we do not require the user to manually convert their expressions to polynomials for processing.

Example 1. As a simple example of the types of statements we aim to prove, consider the following Lean proposition:

```
example {x y : Rat} (f : 2*x= 0) (g : 3*y = 0) : (x + y)^4 = 0
```

This example represents the mathematical statement that for variables $x, y \in \mathbb{Q}$, if $2x = 0$ and $3y = 0$ then $(x + y)^4 = 0$. While this problem is simple, it is a good example of the format of proposition that our current code aims to prove.

Given a problem like that of Example 1, we must broadly speaking do the following steps, which we illustrate in Figure 1:

1. Extract a computational problem from the proposition and hypotheses.
2. Send the problem to Macaulay2, including serialization and de-serialization by Lean and Macaulay2 respectively.
3. Solve the problem using Macaulay2.
4. Send the solution and the certificate to Lean, including again serialization and de-serialization.
5. Convert the solution and certificate to a proof of the original proposition.

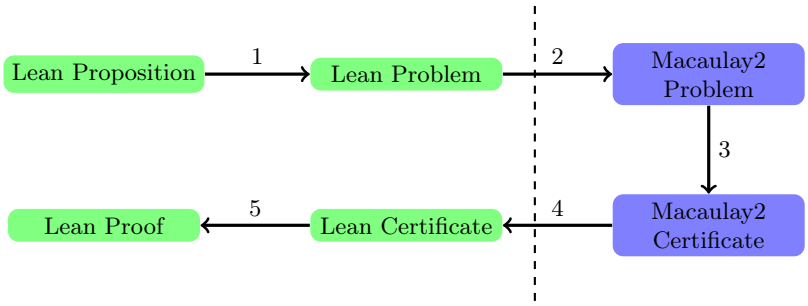

Fig. 1. The general steps by which a Lean proposition is translated to a computation in Macaulay2 and then returned to construct a proof in Lean

We emphasize here that the underlying computational problem is distinct from the proposition to be proven and the certificate is distinct from the proof. The problem consists of the mathematical data needed to construct the certificate and the certificate consists of the mathematical data needed to construct the proof. This separation of the problem and the proposition along with the certificate and the proof is important, because there are instances where two different classes of proposition may be solvable using the same underlying problem and certificate. This decouples the proof goal from the computational goal.

The Ideal Membership Tactic

The user-visible interface for our code is primarily in the form of Lean tactics to allow us to easily read and modify the proof state. We currently implement two tactics, `m2idealmem` and

`m2remainder`, both of which are based on the `quotientRemainder` operation in Macaulay2. The Macaulay2 operation computes the quotient and remainder of a polynomial relative to an ideal. As both tactics share most of the underlying code, we focus here on the `m2idealmem` tactic.

Example 2. Given the example in Example 1, seen above, our tactic can solve it using Macaulay2

```
example {x y : Rat} (f : 2*x= 0) (g : 3*y = 0) : (x + y)^4 = 0 := by
  m2idealmem -grind [f,g]
  clear f g
  grind
```

The tactic we implement is `m2idealmem`. Here it takes as parameters the two propositions `f` and `g`, each of which represents the vanishing of a polynomial over x and y. Here the addition of `-grind` supresses the step where we use grind the prove a polynomial equality. After sending the polynomial $(x+y)^4$ and the ideal generators $2x$ and $3y$ to Macaulay2, the result of the `m2idealmem` tactic then is a goal of the form

```
(x+y)^4=((1/2)*x^3+2*x^2*y+3*x*y^2+2*y^3)*(2 * x)+((1/3)*y^3)*(3*y)
```

which we use `grind` to show. The call to `clear` simply removes the two hypotheses from the context so that grind cannot use them.

We remark that the polynomials used as coefficients, in this case $\frac{1}{2}x^3+2x^2y+3xy^2+2y^3$ and $\frac{1}{3}y^3$, are highly non-unique, and different choices will have performance implications. Second, while the implementation of quotient and remainder in Macaulay2 uses Gröbner bases, we do not return the Gröbner bases to Lean at any phase of this computation.

Finally, we currently outsource the final polynomial equality step to `grind`, however ongoing work is to give a more efficient implementation of this step, tailored towards the polynomial equality statements that we require. For more on this ongoing work see Section 4.

Example 3. As another slightly larger example, we consider the following larger example. This particular example was constructed by taking the 2nd power of a element of a Gröbner basis of the ideal

$$I = \langle (x-uz)^2 + y^2 - r^2z^2, a^2+b^2-c^2, xa+yb-zc, k^2-(u+r)^2+1 \rangle.$$

```
example
  (u r k x y z a b c : Rat)
  (ho : (x - u * z) ^ 2 + y ^ 2 - r ^ 2 * z ^ 2 = 0)
  (hi : a ^ 2 + b ^ 2 - c ^ 2 = 0)
  (hpq : x * a + y * b - z * c = 0)
  (hk : k ^ 2 - (u + r) ^ 2 + 1 = 0) :
    (u*k^2*z^2-r*k^2*z^2+2*u*r*x*z+2*r^2*x*z-2*k^2*x*z+u*x^2+
    r*x^2+u*y^2+r*y^2+u*z^2-r*z^2-2*x*z)^2 = 0 := by
    m2idealmem -grind [ho, hi, hpq, hk]
    grind
```

While this represents a larger example that better demonstrates the capabilities of our work, We do not at this point have an example where `m2idealmem` succeeds but `grind` does not.

While we have focused on the `m2idealmem` tactic, we also give the following example of the usage of the `m2remainder` tactic.

Example 4. We consider the non-radical ideal $I = \langle xy^2 - x^2, y^4 - x^2 \rangle$. The ideal I does not contain $xy^3 - x^2y - y^3 + xy$, however the radical ideal $\sqrt{I}$ does. If we wish to show $xy^3 - x^2y - y^3 + xy = 0$ subject to the constraints $xy^2 - x^2 = 0$ and $y^4 - x^2 = 0$, the `m2idealmem` tactic would not produce the desired result as the polynomial is not contained in the ideal I. Here we can use `m2remainder` to make progress.

```
example
  {x y : Rat} (f : x*y^2-x^2 = 0) (g : y^4-x^2 = 0) :
  x*y^3-x^2*y-y^3+x*y = 0 := by
    m2remainder [f, g]
    ...
```

With an appropriate choice of Gröbner basis, we see that the remainder of $xy^3-x^2y-y^3+xy$ after polynomial long division is $-y^3+xy$. Then the tactic `m2remainder` can use this fact to produce the goals `-1*y^3+x*y = 0` and `x*y^3-x^2*y-y^3+x*y = y*(x*y^2-x^2)+(-1*y^3+x*y)`.

Universalization to $\mathbb{Z}$

There is one final feature of our current work worth noting. Not every ring has an easy representation that allows serialization to Macaulay2. However, even over general rings, many problems can be lifted to the integers especially if the relevant expressions have coefficients over the integers. As such, instead of providing an error when encountering an unknown ring, we first attempt to rephrase the question over the integers. We can see this in the following example:

```
example [CommRing R] {x y z : R} (f : x = 0) (g : y = 0) (h : y+z=0)
  : (x + y + z)^4 = 0 := by
  m2idealmem -grind [f, g, h]
  clear f g h
  grind
```

3 Implementation

In the discussion in Section 2, we have ignored the details of the representation of the mathematical objects on both sides as well as the details of the communication protocol between them. Our current efforts attempt to maximize reuse of existing code for this by employing standard formats and protocols.

Communication

Due to its simplicity and ubiquity, it was decided that JSON (JavaScript Object Notation) would be used to facilitate communication between Macaulay2 and Lean. In particular, the top-level communication is handled by the JSON-RPC protocol [8] and the mathematical objects are serialized using the MRDI file format [5].

The "RPC" in JSON-RPC stands for *remote procedure call*. It is a protocol that defines a set of JSON objects for clients to make requests and servers to send responses. It is the foundation of the Language Server Protocol [10], used by a wide variety of editors, and which is fundamental to the user experience in Lean. Since both Lean and Macaulay2 already support JSON-RPC, it was a natural choice upon which to build the interface.

The MRDI file format is used for serializing mathematical objects in the OSCAR computer algebra system [3,12], but it was designed with the idea that it could be used for the same purpose in other systems. Each MRDI file contains a single JSON object with at least one key, `_type`, for specifying the particular type of the object. Additionally, it may have a `_ns` key for specifying the namespace (such as the particular computer algebra system), a `data` key for information about the object. Some of the values under the `_type` and `data` keys may contain *universally unique identifiers* (or UUID's) for additional objects, and further information about these objects is stored under the `_refs` key.

Consider, for example, the polynomial $3x^2 + \frac{1}{2} \in \mathbb{Q}[x]$. We store this as $3y_0^2 + y_1 \in \mathbb{Z}[y_0, y_1]$ together with the map $y_1 \mapsto \frac{1}{2}$. See the example below.

```
{"_type": {"params": "Rat", "name": "ConcretePoly"},
 "data": {"poly": "a8ea4eb2-4393-45b0-ba9a-0c91716a55d3",
          "coefficients": [["1", ["1", "2"]]]},
 "_ns": {"Lean": ["https://github.com/leanprover/lean4", "4.26.0-rc1"]},
 "_refs": {"a8ea4eb2-4393-45b0-ba9a-0c91716a55d3":
            {"_type": "LeanGrindCommRingPoly",
             "data": [["3", [["0", "2"]]], ["1", [["1", "1"]]]]}}}
```


Putting these pieces together, when we delagate a computation from Lean to Macaulay2, then we may serialize any associated mathematical objects using MRDI, pass these as arguments to an appropriate method and send the corresponding JSON-RPC request to a server running in Macaulay2, de-serialize the mathematical objects in Macaulay2, run the requested computation, serialize any mathematical objects produced by this computation, pass the result in a JSON-RPC response back to Lean, and finally de-serialize the result in Lean.

Polynomial Representation

For convenience, our current efforts reuse parts of the infrastructure of the `grind` tactic. One such reuse is our polynomial representation. We use the `Lean.Grind.CommRing.Poly` type for this purpose. The limitation of this type is that the it is designed to model polynomials with integer coefficients. To handle non-integer coefficients, we add auxiliary variables to the polynomial which are then assigned the individual non-integer coefficients.

4 Ongoing Work and Future Directions

As this is a project in progress, there are some key points where the implementation is still progressing.

Stored Computations

One goal is to make it easier to separate the calls to Macaulay2 and the actual proof in Lean. While the current structure is very useful for interactive work or smaller proofs, there are a few disadvantages with requiring this approach. First is the requirement that a copy of Macaulay2 has to be installed whenever the proof is to be checked. But more significantly, for problems where the computation may require non-trivial resources, the ability to load solutions from a file allows for the separation of the proof stage and the computational stage.

Our use of the MRDI format makes it easier to achieve this objective as MRDI is a format already designed with storage and export/import in mind. In particular its UUID-based design allows for an implementation where the de-serialization of the data is not dependent on the particular state of the computation from when it was serialized.

Polynomial Format

While the current polynomial representation maximizes reuse of `grind`-related code, it is not clear that it is optimal for our purposes. The challenge of this polynomial format is that it should be simple to de-serialize and serialize and have a performant polynomial equality in the kernel. It is comparatively easy to achieve the first objective with a variety of different methods, but to carefully structure the underlying operations to be performant in the kernel requires care.

As an example of the potential scale of computations and why our current approach does not suffice, consider the example from https://github.com/leanprover/lean4/issues/11861, reproduced in the Appendix as Example 5. The computation of the quotient and remainder step is essentially immediate in Macaulay2, completing in a few seconds on a desktop computer. However, the computation results in large polynomials for the quotient, with the largest of them having 16024 terms. While the serialization and de-serialization of the terms proceeds without

difficulty, the size of the term means that we are unable to prove the required polynomial equality using our current approach. The resulting Lean expression is so large as to cause difficulties even when the polynomial equality step is replaced by a call to `sorry`.

Our aim for the new polynomial format is by passing to polynomials in the proof rather than directly manipulating the expressions over the underlying ring, the proof terms will be considerably more manageable.

Acknowledgments. The work in this paper was funded by the Renaissance Philanthropy AI for Math Grant "Bridging Proof and Computation". The fifth author was also supported by a grant from the Simons Foundation International (SFI-MPS-SSRFA-00012695).

The authors would also like to thank two anonymous referees of this paper for their feedback.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Bertoli, P.G., Calmet, J., Giunchiglia, F., Homann, K., Calmet, J., Plaza, J.: Specification and integration of theorem provers and computer algebra systems. In: Artificial Intelligence and Symbolic Computation. pp. 94–106. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
2. Bhatia, D., Wieser, E., Carneiro, M., Zhu, T.: Mathlib documentation Mathlib.Tactic.Polyrith, https://web.archive.org/web/20250313233446/https://leanprover-community.github.io/mathlib4_docs/Mathlib/Tactic/Polyrith.html
3. Decker, W., Eder, C., Fieker, C., Horn, M., Joswig, M. (eds.): The Computer Algebra System OSCAR: Algorithms and Examples, Algorithms and Computation in Mathematics, vol. 32. Springer, 1 edn. (2025). https://doi.org/10.1007/978-3-031-62127-7, https://link.springer.com/book/9783031621260
4. Delahaye, D., Mayero, M.: Quantifier elimination over algebraically closed fields in a proof assistant using a computer algebra system. Electron. Notes Theor. Comput. Sci. **151**(1), 57–73 (Mar 2006). https://doi.org/10.1016/j.entcs.2005.11.023, https://doi.org/10.1016/j.entcs.2005.11.023
5. Della Vecchia, A., Joswig, M., Lorenz, B.: A FAIR file format for mathematical software. In: Mathematical software—ICMS 2024, Lecture Notes in Comput. Sci., vol. 14749, pp. 234–244. Springer, Cham ([2024] ©2024). https://doi.org/10.1007/978-3-031-64529-7_25, https://doi.org/10.1007/978-3-031-64529-7_25
6. Grayson, D.R., Stillman, M.E.: Macaulay2, a software system for research in algebraic geometry. Available at https://macaulay2.com/
7. Harrison, J., Théry, L.: A skeptic's approach to combining HOL and Maple. Journal of Automated Reasoning **21**, 279–294 (1998). https://doi.org/https://doi.org/10.1023/A:1006023127567
8. JSON-RPC Working Group: JSON-RPC 2.0 Specification. http://www.jsonrpc.org (2010)
9. Maletzky, A., Immler, F.: Gröbner bases of modules and Faugère's f_4 algorithm in Isabelle/HOL. In: Rabe, F., Farmer, W.M., Passmore, G.O., Youssef, A. (eds.) Intelligent Computer Mathematics. pp. 178–193. Springer International Publishing, Cham (2018)
10. Microsoft: Language server protocol specification - 3.17. https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/ (2022)
11. de Moura, L., Kong, S., Avigad, J., van Doorn, F., von Raumer, J.: The lean theorem prover (system description). In: Automated deduction—CADE 25, Lecture Notes in Comput. Sci., vol. 9195, pp. 378–388. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21401-6_26, https://doi.org/10.1007/978-3-319-21401-6_26
12. OSCAR – Open Source Computer Algebra Research system, Version 1.7.1 (2026), https://www.oscar-system.org
13. Rowney, T., Ahuja, R., Avigad, J., Welleck, S.: DSLean: A framework for type-correct interoperability between Lean 4 and external DSLs (2026), https://arxiv.org/abs/2602.18657

Appendix

Example 5.

```
example
  (u r k x y z a b c : Rat)
  (ho : (x - u * z) ^ 2 + y ^ 2 - r ^ 2 * z ^ 2 = 0)
  (hi : a ^ 2 + b ^ 2 - c ^ 2 = 0)
  (hpq : x * a + y * b - z * c = 0)
  (hk : k ^ 2 - (u + r) ^ 2 + 1 = 0) :
  (r *
           ((k * x + ((u + r) ^ 2 - 1) * y) * c ^ 2 +
             (2 * u * k * a ^ 2 + u * ((u + r) ^ 2 - 2) * a * b +
                    (r * (u + r) - 2) * k * a * c +
                  (2 - (u + r) * (u + 2 * r)) * b * c -
                u * k * c ^ 2) *
              z) +
         r * ((u + r) * a - c) * ((u + r) * b + k * c) * z * u) ^
       4 *
     y ^ 2 -
   r ^ 2 * 2 ^ 2 * k ^ 2 * z ^ 2 *
     (r ^ 2 *
          (((k * x + ((u + r) ^ 2 - 1) * y) * c ^ 2 +
               (2 * u * k * a ^ 2 + u * ((u + r) ^ 2 - 2) * a * b +
                      (r * (u + r) - 2) * k * a * c +
                    (2 - (u + r) * (u + 2 * r)) * b * c -
                  u * k * c ^ 2) *
                z) ^
            3 *
            (r * ((u + r) * a - c) * ((u + r) * b + k * c) * z)) +
        (1 - u ^ 2 - r ^ 2) *
          (((k * x + ((u + r) ^ 2 - 1) * y) * c ^ 2 +
               (2 * u * k * a ^ 2 + u * ((u + r) ^ 2 - 2) * a * b +
                      (r * (u + r) - 2) * k * a * c +
                    (2 - (u + r) * (u + 2 * r)) * b * c -
                  u * k * c ^ 2) *
                z) ^
            2 *
            (r * ((u + r) * a - c) * ((u + r) * b + k * c) * z) ^ 2) +
       u ^ 2 *
           ((k * x + ((u + r) ^ 2 - 1) * y) * c ^ 2 +
             (2 * u * k * a ^ 2 + u * ((u + r) ^ 2 - 2) * a * b +
                    (r * (u + r) - 2) * k * a * c +
                  (2 - (u + r) * (u + 2 * r)) * b * c -
                u * k * c ^ 2) *
              z) *
         (r * ((u + r) * a - c) * ((u + r) * b + k * c) * z) ^ 3) = 0
```