Reconstructing MetiTarski Proofs in Isabelle/HOL

End-of-internship Presentation

Cristina Matache

AESTHETIC
INTEGRATION

September 22, 2017

Outline

1. MetiTarski
2. Sledgehammer

MetiTarski

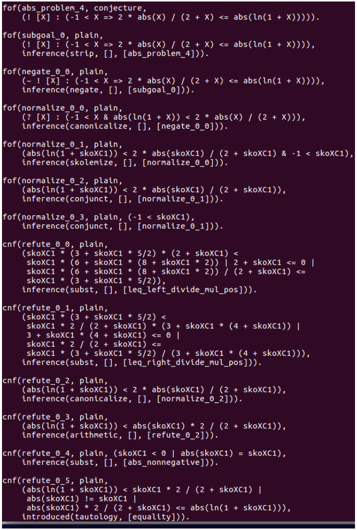
- Automatic theorem prover (ATP)
- Proves universally quantified inequalites involving:
  - polynomials
  - real-valued special functions: *log*, *exp*, *sin*, *cos*, *sqrt* etc.
- Using:
  - resolution
  - a decision procedure for the theory of real closed fields (RCF)
- The special functions are *approximated* by polynomials. So Metitarski is incomplete. (Without the approximations, the problem is undecidable.)

Motivation

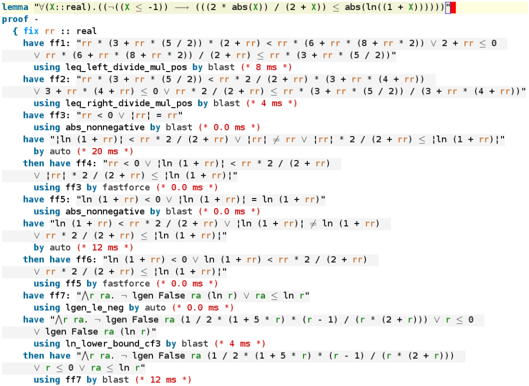
- Long-term goal: use MetiTarski inside Imandra to solve geometric problems
- Why translate MetiTarski proofs to Isabelle proofs?
  - No formal guarantee that the MetiTarski proofs are correct.
  - Isabelle is more trustworthy than MetiTarski
  - If proof reconstruction is available, MetiTarski can be included as an automated tool in Isabelle

1. These can be encoded as inequalities that MT understands.
2. If we want to use MT to verify things, we need very high confidence in its proofs.
3. MetiTarski was coded in Standard ML and there is no guarantee the code o the output is correct. Although it has of course been tested.

The Problem



```
fof(abs_problem_4, conjecture,
    (! [X] : (-1 < X => 2 * abs(X) / (2 + X) <= abs(ln(1 + X))))).

fof(subgoal_0, plain,
    (! [X] : (-1 < X => 2 * abs(X) / (2 + X) <= abs(ln(1 + X)))),
    inference(strip, [], [abs_problem_4])).

fof(negate_0_0, plain,
    (~ ! [X] : (-1 < X => 2 * abs(X) / (2 + X) <= abs(ln(1 + X)))),
    inference(negate, [], [subgoal_0])).

fof(normalize_0_0, plain,
    (? [X] : (-1 < X & abs(ln(1 + X)) < 2 * abs(X) / (2 + X))),
    inference(canonicalize, [], [negate_0_0])).

fof(normalize_0_1, plain,
    (abs(ln(1 + skoXC1)) < 2 * abs(skoXC1) / (2 + skoXC1) & -1 < skoXC1),
    inference(skolemize, [], [normalize_0_0])).

fof(normalize_0_2, plain,
    (abs(ln(1 + skoXC1)) < 2 * abs(skoXC1) / (2 + skoXC1)),
    inference(conjunct, [], [normalize_0_1])).

fof(normalize_0_3, plain, (-1 < skoXC1),
    inference(conjunct, [], [normalize_0_1])).

cnf(refute_0_0, plain,
    (skoXC1 * (3 + skoXC1 * 5/2) * (2 + skoXC1) <
     skoXC1 * (6 + skoXC1 * (8 + skoXC1 * 2)) | 2 + skoXC1 <= 0 |
     skoXC1 * (6 + skoXC1 * (8 + skoXC1 * 2)) / (2 + skoXC1) <=
     skoXC1 * (3 + skoXC1 * 5/2)),
    inference(subst, [], [leq_left_divide_mul_pos])).

cnf(refute_0_1, plain,
    (skoXC1 * (3 + skoXC1 * 5/2) <
     skoXC1 * 2 / (2 + skoXC1) * (3 + skoXC1 * (4 + skoXC1)) |
     3 + skoXC1 * (4 + skoXC1) <= 0 |
     skoXC1 * 2 / (2 + skoXC1) <=
     skoXC1 * (3 + skoXC1 * 5/2) / (3 + skoXC1 * (4 + skoXC1))),
    inference(subst, [], [leq_right_divide_mul_pos])).

cnf(refute_0_2, plain,
    (abs(ln(1 + skoXC1)) < 2 * abs(skoXC1) / (2 + skoXC1)),
    inference(canonicalize, [], [normalize_0_2])).

cnf(refute_0_3, plain,
    (abs(ln(1 + skoXC1)) < abs(skoXC1) * 2 / (2 + skoXC1)),
    inference(arithmetic, [], [refute_0_2])).

cnf(refute_0_4, plain, (skoXC1 < 0 | abs(skoXC1) = skoXC1),
    inference(subst, [], [abs_nonnegative])).

cnf(refute_0_5, plain,
    (abs(ln(1 + skoXC1)) < skoXC1 * 2 / (2 + skoXC1) |
     abs(skoXC1) != skoXC1 |
     abs(skoXC1) * 2 / (2 + skoXC1) <= abs(ln(1 + skoXC1))),
    introduced(tautology, [equality])).
```

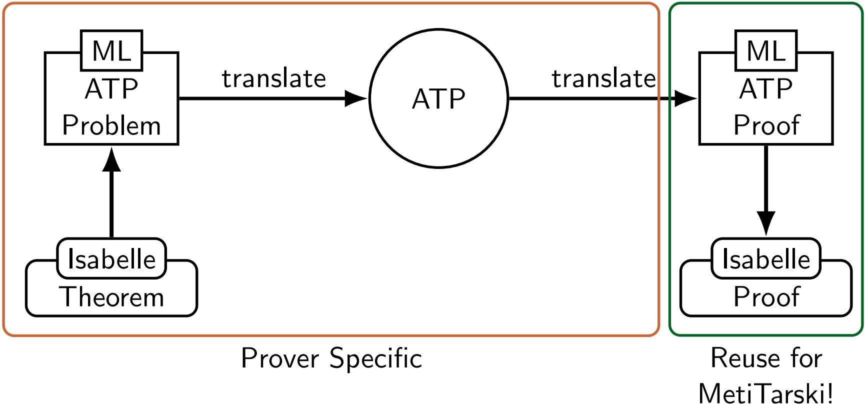


```
lemma "∀(X::real).((¬((X ≤ -1)) ⟶ (((2 * abs(X)) / (2 + X)) ≤ abs(ln((1 + X))))))"
proof -
  { fix rr :: real
    have ff1: "rr * (3 + rr * (5 / 2)) * (2 + rr) < rr * (6 + rr * (8 + rr * 2)) ∨ 2 + rr ≤ 0
      ∨ rr * (6 + rr * (8 + rr * 2)) / (2 + rr) ≤ rr * (3 + rr * (5 / 2))"
      using leq_left_divide_mul_pos by blast (* 8 ms *)
    have ff2: "rr * (3 + rr * (5 / 2)) < rr * 2 / (2 + rr) * (3 + rr * (4 + rr))
      ∨ 3 + rr * (4 + rr) ≤ 0 ∨ rr * 2 / (2 + rr) ≤ rr * (3 + rr * (5 / 2)) / (3 + rr * (4 + rr))"
      using leq_right_divide_mul_pos by blast (* 4 ms *)
    have ff3: "rr < 0 ∨ ¦rr¦ = rr"
      using abs_nonnegative by blast (* 0.0 ms *)
    have "¦ln (1 + rr)¦ < rr * 2 / (2 + rr) ∨ ¦rr¦ ≠ rr ∨ ¦rr¦ * 2 / (2 + rr) ≤ ¦ln (1 + rr)¦"
      by auto (* 20 ms *)
    then have ff4: "rr < 0 ∨ ¦ln (1 + rr)¦ < rr * 2 / (2 + rr)
      ∨ ¦rr¦ * 2 / (2 + rr) ≤ ¦ln (1 + rr)¦"
      using ff3 by fastforce (* 0.0 ms *)
    have ff5: "ln (1 + rr) < 0 ∨ ¦ln (1 + rr)¦ = ln (1 + rr)"
      using abs_nonnegative by blast (* 0.0 ms *)
    have "ln (1 + rr) < rr * 2 / (2 + rr) ∨ ¦ln (1 + rr)¦ ≠ ln (1 + rr)
      ∨ rr * 2 / (2 + rr) ≤ ¦ln (1 + rr)¦"
      by auto (* 12 ms *)
    then have ff6: "ln (1 + rr) < 0 ∨ ln (1 + rr) < rr * 2 / (2 + rr)
      ∨ rr * 2 / (2 + rr) ≤ ¦ln (1 + rr)¦"
      using ff5 by fastforce (* 0.0 ms *)
    have ff7: "⋀r ra. ¬ lgen False ra (ln r) ∨ ra ≤ ln r"
      using lgen_le_neg by auto (* 0.0 ms *)
    have "⋀r ra. ¬ lgen False ra (1 / 2 * (1 + 5 * r) * (r - 1) / (r * (2 + r))) ∨ r ≤ 0
      ∨ lgen False ra (ln r)"
      using ln_lower_bound_cf3 by blast (* 4 ms *)
    then have "⋀r ra. ¬ lgen False ra (1 / 2 * (1 + 5 * r) * (r - 1) / (r * (2 + r)))
      ∨ r ≤ 0 ∨ ra ≤ ln r"
      using ff7 by blast (* 12 ms *)
```

1. We need to turn this MT proof into an Isabelle proof. You probably can't read this because the proofs look horrible, and that's exactly the point. You want to be able to do this translation automatically. Roughly what happens is that every step in the MT proof corresponds to a "have" in Isabelle and proof method.
2. In Isabelle, the way you do proofs is you can specify intermmediate steps, prove them, and then use them later to prove your conclusion. To prove things you need to specify a proof method, there are maybe 10-15 that are commonly used. And sometimes you have to give Isabelle hints about what lemmas to use.

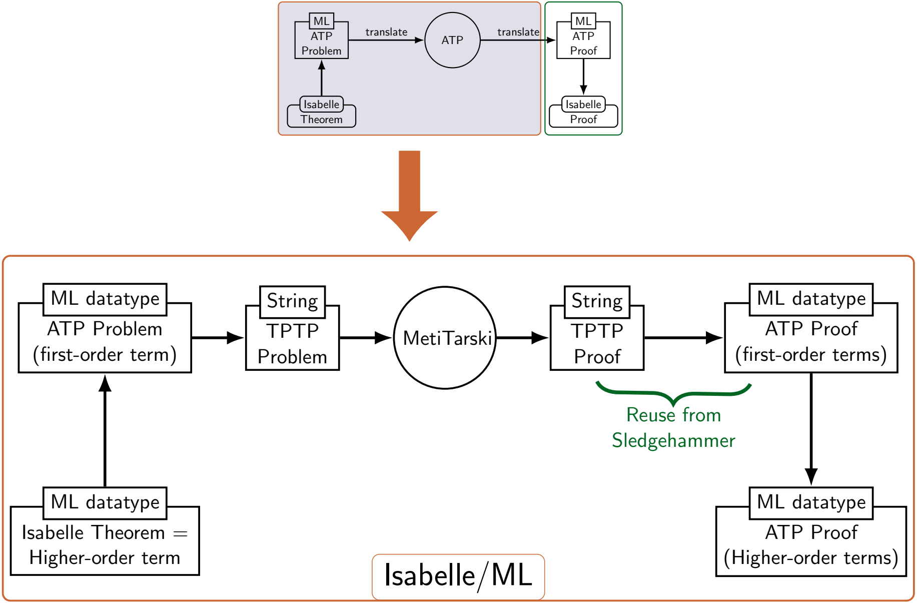
Sledgehammer

- Automatic proof tool in Isabelle.
- Sledgehammer operation:



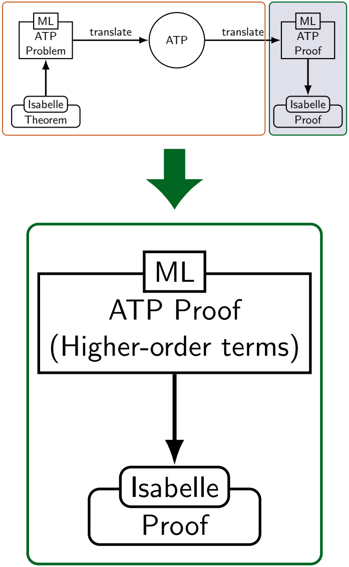
1. It turns out there is a tool in Isabelle that does something similar. Isabelle an *interactive* theorem prover so it normally requires a lot of user input to do a proof, the user has to specify what steps should be taken. Sledgehammer aims to improve automation.
2. Sledgehammer tries to automatically find a proof for the given lemma without any user input. What it does is it calls a bunch of ATPs and tries t translate their proofs into Isabelle proofs. Which is exactly what I need to do.
3. When the user invokes Sledgehammer the current goal is translated into some intermediate form, this ATP problem, which is an ML datastructure. This is given to the ATP, it comes up with a proof, which you can put into this standard format of an ATP proof, and then translate it to an Isabelle proof. This is of course a simplified diagram of what happens but you get the idea.
4. The left part of the diagram is specific to every prover, because to get the ATP proof we need some information about how the initial Isabelle lemma was encoded. And this of course depends on each prover. But the right par of the diagram does not depend on the prover. And it is actually the more involved part. So the idea was to somehow reuse this part when reconstructing Metitarski proofs.

Translations



1. I'm going to focus first on the left side of the diagram before, which involve multiple translations of the problem and the proof. The part I said was prover specific.
2. Because Isabelle is written in SML, any code that wants to interface with it must access the ML internals. So all of the code for this diagram is in SML So for example there is an internal representation of theorems as an ML datatype and you need to use that to manipulate the statement of theorem
3. Explain difference between higher-order syntax and first-order using +
4. TPTP format is a standard format that ATPs use and MT also uses. So al the Isabelle names and symbols have to be translated to the tptp symbols.
5. The

Generating Isabelle Proofs



ML

ATP Problem

translate

ATP

translate

ML

ATP Proof

Isabelle

Theorem

Isabelle

Proof

ML

ATP Proof

(Higher-order terms)

Isabelle

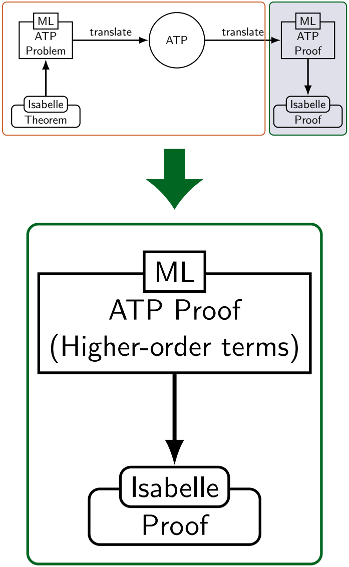
Proof

Existing functionality:

- Proof redirection;
- Proof minimization;
- Proof preplay;
- Type annotations.

1. This is the part of Sledgehammer we decided to reuse, the right of the diagram. From the ATP proof with higher-order terms to the actual Isabell proof text. Sledgehammer already has a lot of sophisticated functionality implemented for this, apart from the textual translation. Explain each of it.
2. But what is difficult is choosing the appropriate proof method for each step in the proof. Because this depends on what steps the prover made. So I ha to change the Sledgehammer code to for the MT proofs to work.
3. In particular MetiTarski does some algebraic simplifications so a custom Isabelle method is needed to mirror it. So lately I've been building this method but it still needs more experimentation.
4. I mentioned that MT uses a decision procedure for real closed fields. This needs to be formalised in Isabelle as a proof method. At the moment,one c Larry's students has a formalisation for the univariate case that we can use But ideally we would extend it to multivariate.
5. In the beginning of the talk I also said that special functions are approximated with polynomials. These approximations are inequalities that MT considers to be axioms. So in Isabelle you would need to prove these inequalities before using them. Some of these have already been formalised but they need to be brought to the exact form that MT uses. While some still need to be proved.

Generating Isabelle Proofs

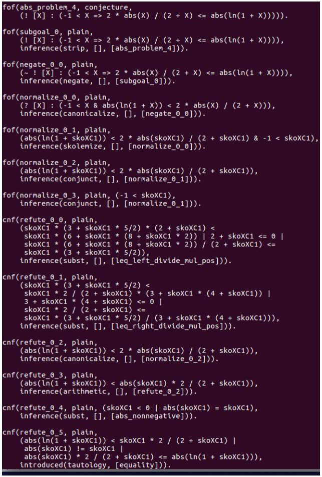


MetiTarski requirements:

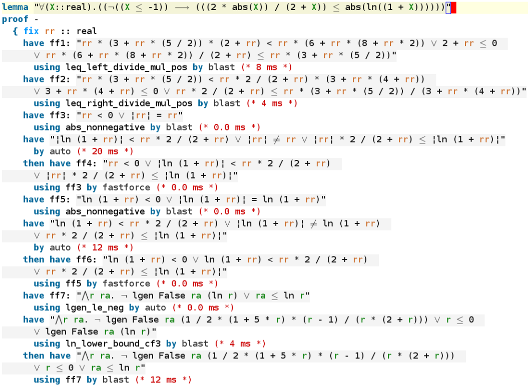
- Proof methods for:
  - algebraic simplification;
  - the decision procedure.
- Correctness of special function bounds;

1. This is the part of Sledgehammer we decided to reuse, the right of the diagram. From the ATP proof with higher-order terms to the actual Isabell proof text. Sledgehammer already has a lot of sophisticated functionality implemented for this, apart from the textual translation. Explain each of it.
2. But what is difficult is choosing the appropriate proof method for each step in the proof. Because this depends on what steps the prover made. So I ha to change the Sledgehammer code to for the MT proofs to work.
3. In particular MetiTarski does some algebraic simplifications so a custom Isabelle method is needed to mirror it. So lately I've been building this method but it still needs more experimentation.
4. I mentioned that MT uses a decision procedure for real closed fields. This needs to be formalised in Isabelle as a proof method. At the moment,one o Larry's students has a formalisation for the univariate case that we can use But ideally we would extend it to multivariate.
5. In the beginning of the talk I also said that special functions are approximated with polynomials. These approximations are inequalities that MT considers to be axioms. So in Isabelle you would need to prove these inequalities before using them. Some of these have already been formalised but they need to be brought to the exact form that MT uses. While some still need to be proved.

What we have so far



```
fof(abs_problem_4, conjecture,
    (! [X] : (-1 < X => 2 * abs(X) / (2 + X) <= abs(ln(1 + X))))).

fof(subgoal_0, plain,
    (! [X] : (-1 < X => 2 * abs(X) / (2 + X) <= abs(ln(1 + X)))),
    inference(strip, [], [abs_problem_4])).

fof(negate_0_0, plain,
    (~ ! [X] : (-1 < X => 2 * abs(X) / (2 + X) <= abs(ln(1 + X)))),
    inference(negate, [], [subgoal_0])).

fof(normalize_0_0, plain,
    (? [X] : (-1 < X & abs(ln(1 + X)) < 2 * abs(X) / (2 + X))),
    inference(canonicalize, [], [negate_0_0])).

fof(normalize_0_1, plain,
    (abs(ln(1 + skoXC1)) < 2 * abs(skoXC1) / (2 + skoXC1) & -1 < skoXC1),
    inference(skolemize, [], [normalize_0_0])).

fof(normalize_0_2, plain,
    (abs(ln(1 + skoXC1)) < 2 * abs(skoXC1) / (2 + skoXC1)),
    inference(conjunct, [], [normalize_0_1])).

fof(normalize_0_3, plain, (-1 < skoXC1),
    inference(conjunct, [], [normalize_0_1])).

cnf(refute_0_0, plain,
    (skoXC1 * (3 + skoXC1 * 5/2) * (2 + skoXC1) <
     skoXC1 * (6 + skoXC1 * (8 + skoXC1 * 2)) | 2 + skoXC1 <= 0 |
     skoXC1 * (6 + skoXC1 * (8 + skoXC1 * 2)) / (2 + skoXC1) <=
     skoXC1 * (3 + skoXC1 * 5/2)),
    inference(subst, [], [leq_left_divide_mul_pos])).

cnf(refute_0_1, plain,
    (skoXC1 * (3 + skoXC1 * 5/2) <
     skoXC1 * 2 / (2 + skoXC1) * (3 + skoXC1 * (4 + skoXC1)) |
     3 + skoXC1 * (4 + skoXC1) <= 0 |
     skoXC1 * 2 / (2 + skoXC1) <=
     skoXC1 * (3 + skoXC1 * 5/2) / (3 + skoXC1 * (4 + skoXC1))),
    inference(subst, [], [leq_right_divide_mul_pos])).

cnf(refute_0_2, plain,
    (abs(ln(1 + skoXC1)) < 2 * abs(skoXC1) / (2 + skoXC1)),
    inference(canonicalize, [], [normalize_0_2])).

cnf(refute_0_3, plain,
    (abs(ln(1 + skoXC1)) < abs(skoXC1) * 2 / (2 + skoXC1)),
    inference(arithmetic, [], [refute_0_2])).

cnf(refute_0_4, plain, (skoXC1 < 0 | abs(skoXC1) = skoXC1),
    inference(subst, [], [abs_nonnegative])).

cnf(refute_0_5, plain,
    (abs(ln(1 + skoXC1)) < skoXC1 * 2 / (2 + skoXC1) |
     abs(skoXC1) != skoXC1 |
     abs(skoXC1) * 2 / (2 + skoXC1) <= abs(ln(1 + skoXC1))),
    introduced(tautology, [equality])).
```



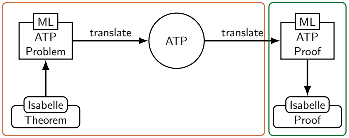
```
lemma "∀(X::real).((¬((X ≤ -1)) ⟶ (((2 * abs(X)) / (2 + X)) ≤ abs(ln((1 + X))))))"
proof -
  { fix rr :: real
    have ff1: "rr * (3 + rr * (5 / 2)) * (2 + rr) < rr * (6 + rr * (8 + rr * 2)) ∨ 2 + rr ≤ 0
      ∨ rr * (6 + rr * (8 + rr * 2)) / (2 + rr) ≤ rr * (3 + rr * (5 / 2))"
      using leq_left_divide_mul_pos by blast (* 8 ms *)
    have ff2: "rr * (3 + rr * (5 / 2)) < rr * 2 / (2 + rr) * (3 + rr * (4 + rr))
      ∨ 3 + rr * (4 + rr) ≤ 0 ∨ rr * 2 / (2 + rr) ≤ rr * (3 + rr * (5 / 2)) / (3 + rr * (4 + rr))"
      using leq_right_divide_mul_pos by blast (* 4 ms *)
    have ff3: "rr < 0 ∨ ¦rr¦ = rr"
      using abs_nonnegative by blast (* 0.0 ms *)
    have "¦ln (1 + rr)¦ < rr * 2 / (2 + rr) ∨ ¦rr¦ ≠ rr ∨ ¦rr¦ * 2 / (2 + rr) ≤ ¦ln (1 + rr)¦"
      by auto (* 20 ms *)
    then have ff4: "rr < 0 ∨ ¦ln (1 + rr)¦ < rr * 2 / (2 + rr)
      ∨ ¦rr¦ * 2 / (2 + rr) ≤ ¦ln (1 + rr)¦"
      using ff3 by fastforce (* 0.0 ms *)
    have ff5: "ln (1 + rr) < 0 ∨ ¦ln (1 + rr)¦ = ln (1 + rr)"
      using abs_nonnegative by blast (* 0.0 ms *)
    have "ln (1 + rr) < rr * 2 / (2 + rr) ∨ ¦ln (1 + rr)¦ ≠ ln (1 + rr)
      ∨ rr * 2 / (2 + rr) ≤ ¦ln (1 + rr)¦"
      by auto (* 12 ms *)
    then have ff6: "ln (1 + rr) < 0 ∨ ln (1 + rr) < rr * 2 / (2 + rr)
      ∨ rr * 2 / (2 + rr) ≤ ¦ln (1 + rr)¦"
      using ff5 by fastforce (* 0.0 ms *)
    have ff7: "⋀r ra. ¬ lgen False ra (ln r) ∨ ra ≤ ln r"
      using lgen_le_neg by auto (* 0.0 ms *)
    have "⋀r ra. ¬ lgen False ra (1 / 2 * (1 + 5 * r) * (r - 1) / (r * (2 + r))) ∨ r ≤ 0
      ∨ lgen False ra (ln r)"
      using ln_lower_bound_cf3 by blast (* 4 ms *)
    then have "⋀r ra. ¬ lgen False ra (1 / 2 * (1 + 5 * r) * (r - 1) / (r * (2 + r)))
      ∨ r ≤ 0 ∨ ra ≤ ln r"
      using ff7 by blast (* 12 ms *)
```

But some proof methods are still missing!

1. So this is the same picture as in the beginning. We got the translation that we wanted from a MT proof to an Isabelle proof. But what is missing is appropriate proof methods for some of the steps.

Summary

- Translate MetiTarski proofs to Isabelle.
- Reuse part of the Sledgehammer code.
- Implement the prover-specific part.



- Provide appropriate proof methods.

1. So we noticed that this tool in Isabelle does something similar for other automitic theorem provers and decided to use part of this functionality. Bu I had to reimplement the part that was specific to each prover. There is th picture about how Sledgehammer works that I kept showing you.
2. In the Sldgehammer code I had to customise the code for choosing proof methods. And provide some new proof methods needed in the MT proofs.

Still to do

- Finish proof method for algebraic simplification.
- Use the formalisation of the decision procedure.
- Prove more bounds.
- Thoroughly test the proof reconstruction.

1. There is still a certain shape of goals it can't prove
2. At the moment for the steps that require the RCF decision procedure I use simpler method that is also available in Isabelle and that's easier to use. Bu this method is less powerful so we wouldn't want to use it in the long-run. But instead we would want to use the formalisation that Wenda has.
3. I mentioned that some of the approximations for special functions still neec to be proved in Isabelle.
4. MetiTarski comes with a few hundreds of sample problems and ideally we would like to reconstruct the proofs for all of these in Isabelle. So seeing how many we can do now and where they fail would be a helpful in determining what to focus on next.