The tactic rfl means reflexivity

Project: LEAN Programming Language
Document: Document 1.1
Author: mujirin
Verifier: Not verified
Downloaded: June 14, 2026 05:12 KST
Status: Working

Why rfl Proves Reflexive Equalities in Lean

The highlighted sentence says: "The tactic rfl means reflexivity." This is a good first approximation, but it hides a central subtlety of Lean. In ordinary mathematical language, reflexivity of equality means that every object is equal to itself: x = x. In Lean, rfl is the proof tool that exploits this principle, but it does so through Lean's notion of definitional equality. That is why it can prove not only goals that look like 4 = 4, but also goals such as 2 + 2 = 4, when Lean can compute both sides to the same canonical expression.

The parent document's example is:

```
theorem two_plus_two : 2 + 2 = 4 := by
rfl
```

The theorem being declared has type 2 + 2 = 4. Under the propositions-as-types interpretation, this type is a proposition, and the proof must be a term inhabiting that proposition [Avigad et al.]. The phrase := by begins a tactic proof. Inside that proof, the tactic rfl asks Lean to close the current goal by reflexivity.

The interesting question is: how can reflexivity prove 2 + 2 = 4, when the two sides are not textually identical?

Equality, Reflexivity, and the Constructor Eq.refl

Lean's equality type is usually written with the infix notation =. If a and b are terms of the same type, then a = b is a proposition asserting that they are equal. At the foundation of this equality type is a constructor often described as reflexivity. In Lean, the canonical proof that a term is equal to itself is Eq.refl.

Informally, one may think of:

```
Eq.refl a : a = a
```

as saying: for any term a, there is a direct proof that a equals itself. The tactic rfl uses this reflexive proof principle. If the current goal is, or can be seen by Lean as, a reflexive equality, then rfl succeeds.

For example, the following theorem is as direct as possible:

```
theorem four_eq_four : 4 = 4 := by
rfl
```

Here the left and right sides are visibly the same. Reflexivity applies without any arithmetic insight. The goal is already of the form a = a.

But Lean's equality checker is not limited to visible textual sameness. It also knows when two expressions are the same after unfolding definitions and evaluating computation. This is where the parent document's explanation becomes important: Lean can reduce 2 + 2 to 4, so both sides are definitionally the same.

Definitional Equality: Sameness by Computation

A proof assistant such as Lean distinguishes between at least two notions that informal mathematics often blends together.

The first is propositional equality. This is an explicit proposition of the form:

$$a = b.$$

To use such an equality, one often needs a proof term or theorem. For example, a theorem may prove that addition is commutative:

$$m + n = n + m.$$

That is not usually true merely by computation from the definitions when m and n are variables; it requires a mathematical proof.

The second is definitional equality, sometimes called convertibility. Two expressions are definitionally equal when Lean's kernel treats them as the same because they reduce to the same expression by computation, unfolding definitions, simplifying pattern matches, or performing other built-in reductions of the type theory [de Moura and Ullrich 2021]. Definitional equality is not something the user proves as a theorem each time. It is part of the type checker's judgment.

The example 2 + 2 = 4 depends on this second notion. Numerals such as 2 and 4 are represented internally as natural numbers. Addition on natural numbers is a recursive definition. When Lean sees 2 + 2, it can evaluate that expression according to the definition of addition. The result is 4. Therefore, from Lean's point of view, the goal

```
2 + 2 = 4
```

can be converted into something like:

```
4 = 4
```

At that point, reflexivity applies. The tactic rfl does not prove a general arithmetic theorem such as "addition works correctly" in a broad informal sense. Rather, in this particular case, the computation built into the definitions reduces the left-hand side to the right-hand side, and reflexivity finishes the goal.

This is why the sentence "rfl means reflexivity" is directly supported by the parent document, but should be read with care. In Lean practice, rfl means more than "the two sides are textually identical." It means that the goal can be solved by reflexivity after Lean accounts for definitional equality.

Why rfl Is Stronger Than It First Looks

Suppose one defines a function:

```
def square (n : Nat) : Nat :=
n * n
```

Then the theorem

```
theorem square_zero : square 0 = 0 := by
rfl
```

may be accepted because Lean unfolds square 0 to 0 0, evaluates the multiplication, and sees the result as 0. Again, the proof is reflexive at the kernel level after computation.

This pattern is common in Lean. A theorem proved by rfl often says: "once the definitions are unfolded and computed, the two sides are the same." Such proofs are especially useful for checking that a definition has the expected behavior on concrete inputs.

But this power has limits. Consider a general statement such as:

```
theorem add_comm_example (m n : Nat) : m + n = n + m := by
rfl
```

This will not work. The equality is mathematically true for natural numbers, but the two sides do not reduce to the same expression for arbitrary variables m and n. Lean cannot compute m + n to a numeral, because m and n are unspecified. A proof of commutativity requires induction or an existing theorem such as Nat.add_comm.

So rfl proves equalities that are true "by definition," not equalities that require mathematical reasoning beyond definitional computation. The distinction is crucial when verifying Lean proofs. If rfl succeeds, Lean has found that the goal is reducible to a reflexive fact. If it fails, the statement may still be true, but it is not definitionally obvious to Lean.

Tactic Proofs and Kernel Checking

The parent document correctly places rfl inside a tactic proof:

```
:= by
rfl
```

Tactics are a user-friendly layer. They transform proof goals, search for proof terms, apply theorems, unfold definitions, or invoke automation. But Lean's trust does not ultimately rest on the English meaning of a tactic. It rests on the kernel checking the proof object produced by elaboration [Avigad et al.; de Moura and Ullrich 2021].

In the case of rfl, the produced proof is essentially a reflexivity proof, such as an instance of Eq.refl, accepted because the target proposition is definitionally equal to a reflexive equality. The kernel checks that this proof term really has the theorem's claimed type. Thus, if the theorem is accepted without sorry or unsafe axioms, Lean has verified that the proof object is valid relative to Lean's foundations and definitions.

This matters for the highlighted passage because "means reflexivity" should not be interpreted as a mere mnemonic. It is connected to the formal constructor for equality and to the type checker's ability to identify definitional sameness.

What Must Be Verified Around This Claim

The parent document's explanation is sound as an introductory account: rfl is indeed Lean's reflexivity tactic, and 2 + 2 = 4 is accepted because Lean can compute the left side to the right side. The main assumption behind this explanation is that the reader understands "the same" in Lean's technical sense, not merely in a visual or informal sense.

A careful reader should verify three things when applying this idea elsewhere. First, whether the target relation is equality or another relation for which Lean's reflexivity mechanism applies. Second, whether the two sides are definitionally equal after Lean unfolds definitions and performs computation. Third, whether the proof is relying only on computation and reflexivity, rather than on a hidden theorem, axiom, or omitted proof. In small examples like 2 + 2 = 4, these issues are straightforward. In larger formalizations, especially with custom definitions, typeclass-driven notation, or imported libraries, the question "will rfl see these as the same?" can become a meaningful diagnostic of how the definitions were designed.

References

[Avigad et al.] Jeremy Avigad, Leonardo de Moura, Soonho Kong, and Sebastian Ullrich, Theorem Proving in Lean 4, online book. https://leanprover.github.io/theoremprovinginlean4/

[Lean Reference Manual] The Lean Language Reference, official Lean documentation. https://lean-lang.org/doc/reference/latest/

[de Moura and Ullrich 2021] Leonardo de Moura and Sebastian Ullrich, "The Lean 4 Theorem Prover and Programming Language," in Automated Deduction – CADE 28, Lecture Notes in Computer Science, vol. 12699, Springer, 2021. https://lean-lang.org/papers/lean4.pdf

[Lean Website] Lean Programming Language, official project website. https://lean-lang.org/