A Lean tactic for normalising ring expressions with exponents

Anne Baanen

Lean Forward
Vrije Universiteit Amsterdam

IJCAR, 2 July 2020

Background

Lean is a proof assistant based on the calculus of constructions. It has a simple kernel for proof checking and an elaborator with powerful tactic support.
The Lean community is developing `mathlib`, a repository of formalised classical mathematics proofs and proof automation.

The Lean Forward project aims to make proof assistants accessible to mathematicians by developing proof automation informed by users' needs.

Background

To a mathematician, the following is obvious:

$$2^{n+1} - 1 = 2 * 2^n - 1$$

Lean should do it automatically.

The `ring` tactic (in Lean and Coq) proves equations using Horner normal form: efficient for the semiring operators $+$ and $*$, but it doesn't support exponentiation ($^\wedge$) by variables.

Background

To a mathematician, the following is obvious:

$$2^{n+1} - 1 = 2 * 2^n - 1$$

Lean should do it automatically.

The `ring` tactic (in Lean and Coq) proves equations using Horner normal form: efficient for the semiring operators $+$ and $*$, but it doesn't support exponentiation ($^\wedge$) by variables.

It's not too hard to solve this manually: rewrite $a^{n+1} = a^n * a$ and `ring` can finish by applying commutativity.

Such rules don't work unconditionally: x^{100} should not become 100 multiplications of x.

Background

More examples from `mathlib`:

$$
\begin{aligned}
2^p * 2 &= 2^{p+1} \\
4^m * 4^{m+1} &= 4^{2m+1} \\
x^{k+2} - y^{k+2} &= x * (x^{k+1} - y^{k+1}) + (x * y^{k+1} - y^{k+2}) \\
(x+y)(x^{n+1} + (n+1)x^{n+1-1}y + zy^2) &= x^{n+2} + (n+2)x^{n+1}y + \\
&\quad (xz + (n+1) * x^n + zy)y^2
\end{aligned}
$$

Design overview

Goal: a practical normalising tactic `ring_exp` for expressions with $+$, $*$ and $^\wedge$, numerals (in $\mathbb{Q}$) and variables. It should solve all goals that `ring` can and be approximately as fast.

Design overview

Goal: a practical normalising tactic `ring_exp` for expressions with $+$, $*$ and $^\wedge$, numerals (in $\mathbb{Q}$) and variables. It should solve all goals that `ring` can and be approximately as fast.

To prove $a = b$, normalise a giving $p_a : a = a'$ and normalise b giving $p_b : b = b'$, then check a' is identical to b'. If identical, `eq.trans` p_a (`eq.symm` p_b) proves $a = b$.

Lean tactics typically don't use reflection since producing proof terms (in the VM) tends to be faster than kernel reduction.

The normal form is a syntax tree in the type family ex. The children for each node are restricted by a parameter ex_type:

```
inductive ex_type : Type
| sum | prod | exp | base
inductive ex : ex_type → Type
| zero  : ex_info →                          ex sum  -- 0
| sum   : ex_info → ex prod →  ex sum → ex sum  -- +
| coeff : ex_info →             coeff → ex prod -- rat
| prod  : ex_info → ex exp  → ex prod → ex prod -- *
| exp   : ex_info → ex base → ex prod → ex exp  -- ^
| var   : ex_info →              atom → ex base -- atom
| sum_b : ex_info →            ex sum → ex base -- (...)
```

- $(a+b)+c$ is not allowed: left argument to sum must be a product
- $a * (b+c)$ is not allowed: right argument to prod must be a product

Intricacies

Commutativity: pick a linear order $\prec$ on ex. Then sort $a + b + c + \cdots$ so that $a \prec b \prec c \prec \cdots$.

Intricacies

Commutativity: pick a linear order $\prec$ on `ex`. Then sort $a + b + c + \cdots$ so that $a \prec b \prec c \prec \cdots$.

To prevent exponential blowup, don't unfold $100 * a$ to $a + a + \cdots + a$. This means keeping track of coefficients.

The function `add_overlap` decides when to add coefficients:

$$\begin{aligned}
&\texttt{add_overlap}\ (3 * x^2)\ (7 * x^2) && = 10 * x^2 \\
&\texttt{add_overlap}\ (3 * x^2)\ (7 * y^2) && = 3 * x^2 + 7 * y^2 \\
&\texttt{add_overlap}\ (3 * x^2)\ (-3 * x^2) && = 0 \qquad \text{(not } 0 * x^2\text{)}
\end{aligned}$$

Intricacies

Commutativity: pick a linear order $\prec$ on `ex`. Then sort $a + b + c + \cdots$ so that $a \prec b \prec c \prec \cdots$.

To prevent exponential blowup, don't unfold $100 * a$ to $a + a + \cdots + a$. This means keeping track of coefficients.

The function `add_overlap` decides when to add coefficients:

$$\begin{aligned}
&\texttt{add_overlap}\ (3 * x^2)\ (7 * x^2) &&= 10 * x^2 \\
&\texttt{add_overlap}\ (3 * x^2)\ (7 * y^2) &&= 3 * x^2 + 7 * y^2 \\
&\texttt{add_overlap}\ (3 * x^2)\ (-3 * x^2) &&= 0 \qquad \text{(not } 0 * x^2\text{)}
\end{aligned}$$

In a general semiring R, exponentiation has type $^\wedge : R \to \mathbb{N} \to R$. During execution, `ring_exp` keeps track of the current type using a reader monad transformer.

Optimisations

To be practical, the `ring_exp` tactic must be fast and optimisation is needed to achieve acceptable running time. The Horner form used by the `ring` tactic is optimal for $+$ and $*$.

Optimisations

To be practical, the `ring_exp` tactic must be fast and optimisation is needed to achieve acceptable running time. The Horner form used by the `ring` tactic is optimal for $+$ and $*$.

Typeclass instances and implicit arguments cost time to infer,
so they are cached:
instances are stored with the current type in the reader monad,
implicit arguments and intermediate values in the `ex_info` field of `ex`.

Discussion

After optimisations, the running time of `ring` and `ring_exp` are in the same order of magnitude.

On problems with larger exponents, `ring_exp` is noticeably faster (20 times on $x^{50} * x^{50} = x^{100}$), also in practice for $(1 + x^2 + x^4 + x^6) * (1 + x) = 1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7$.

Since the `ex` type is an AST, extending `ring_exp` to other algebraic structures is relatively straightforward.