powers commute

Johan G. F. Belinfante  
2002 June 5

■ Introduction

It is shown in the notebook that all powers commute. What is novel here is that we look not at individual powers, which would be vertical sections of the relation power\([x]\), but rather at arbitrary images, that is, arbitrary unions of individual powers. The commutative law for powers presages the theorem that addition of natural numbers is commutative, but at this stage we have yet to define addition of natural numbers. The use of RIF allows a neat expression of commutativity which uses only one variable.

■ Preliminary simplification rules.

Our first step is to translate the theorem about iterated iteration into the language of powers. In general, each of the concepts iterate and power could be defined in terms of the other, and so any theorem about one implies something about the other. The trick is to do this cleanly. We begin with three temporary simplification rules which help to control the complexity of the formulas obtained.

\[
\text{Assoc}\left[\text{id}\left[\text{complement}\left[\text{cart}\left[\text{V}, \text{V}\right]\right]\right], \text{id}\left[\text{cart}\left[\text{V}, \text{V}\right]\right], \text{power}\left[\text{x}\right]\right]\right]
\]
\[
\text{composite}\left[\text{id}\left[\text{complement}\left[\text{cart}\left[\text{V}, \text{V}\right]\right]\right], \text{power}\left[\text{x}\right]\right]\right] = 0
\]
\[
\text{composite}\left[\text{id}\left[\text{complement}\left[\text{cart}\left[\text{V}, \text{V}\right]\right]\right], \text{power}\left[\text{x}_{\_}\right]\right]\right] := 0
\]

\[
\text{Assoc}\left[\begin{array}{c}
\text{id}\left[\text{intersection}\left[\text{complement}\left[\text{cart}\left[\text{V}, \text{V}\right]\right], \text{image}\left[\text{V}, \text{intersection}\left[\text{y}, \text{singleton}\left[0\right]\right]\right]\right]\right]\right], \\
\text{id}\left[\text{cart}\left[\text{V}, \text{V}\right]\right], \text{power}\left[\text{x}\right]\right]\right]\
\text{composite}\left[\begin{array}{c}
\text{id}\left[\text{intersection}\left[\text{complement}\left[\text{cart}\left[\text{V}, \text{V}\right]\right], \text{image}\left[\text{V}, \text{intersection}\left[\text{y}, \text{singleton}\left[0\right]\right]\right]\right]\right]\right], \\
\text{power}\left[\text{x}\right]\right]\right] = 0
\end{array}\right]
\]
\[
\text{composite}\left[\text{id}\left[\text{intersection}\left[\text{complement}\left[\text{cart}\left[\text{V}, \text{V}\right]\right], \text{image}\left[\text{V}, \text{intersection}\left[\text{y}_{\_}, \text{singleton}\left[0\right]\right]\right]\right]\right]\right], \\
\text{power}\left[\text{x}_{\_}\right]\right]\right] := 0
\]
composite[SWAP, SECOND, id[cart[x, V]], inverse[RIF]] // DoubleInverse

composite[SWAP, SECOND, id[cart[x, V]], inverse[RIF]] == cross[inverse[x], Id]

composite[SWAP, SECOND, id[cart[x_, V]], inverse[RIF]] := cross[inverse[x], Id]

■ Lemma

The idea is to translate the following rule for iterated iteration by eliminating iterate in favor of power.

\[
\text{iterate}[u, \text{image[iterate}[u, v], w]]
\]

\[
\text{composite[\text{image[power}[u], w], \text{iterate}[u, v]]}
\]

Recall the rule defining power in terms of iterate.

\[
\text{iterate[cross[Id, x], Id]}
\]

\[
\text{power}[x]
\]

The first step of the translation of the rule for iterated iteration uses this latter formula to eliminate the inner occurrence of iterate.

\[
\text{SubstTest[iterate, u, \text{image[iterate}[u, v], w], \{u \rightarrow \text{cross[Id, x]}, v \rightarrow \text{Id}\}]}
\]

\[
\text{iterate[cross[Id, x], \text{image[power}[x], w]] ==}
\]

\[
\text{composite[cross[Id, \text{image[power}[x], w]], power[x]]}
\]

This begins the process of eliminating iterate in favor of power, and we add it as a temporary rewrite rule:

\[
\text{iterate[cross[Id, x_], \text{image[power}[x_], w_]] :=}
\]

\[
\text{composite[cross[Id, \text{image[power}[x], w]], power[x]]}
\]

■ continuing the translation

To continue, we recall the rule that ordinarily defines iterate in terms of power:

\[
\text{composite[SECOND, id[cart[y, V]], power[x]]}
\]

\[
\text{iterate[x, y]}
\]

We now apply this rule to the special case what leads to the type of expression involving iterate that occurs in the temporary rule. This rule kicks in and converts the expression to something which involves only power.

\[
\text{SubstTest[composite, SECOND, id[cart[v, V]], power[u],}
\]

\[
\{u \rightarrow \text{cross[Id, x]}, v \rightarrow \text{image[power}[x], y]\})]
\]

\[
\text{composite[cross[\text{inverse[\text{image[power}[x], y]]}, \text{Id}], power[x]] ==}
\]

\[
\text{composite[cross[Id, \text{image[power}[x], y]], power[x]]}
\]

The final step is to map this with image, and would you believe it, the whole thing miraculously simplifies to the law that says that any two powers commute.
Map[img[#, z] &%]
  composite[img[power[x], z], img[power[x], y]] ==
  composite[img[power[x], y], img[power[x], z]]

We cannot make this into a simplification rule because it would lead to looping. We can however convert this to a statement of fact, and we can add this fact to the GOEDEL program.

Map[equal[#, composite[img[power[x], y], img[power[x], z]]] &%,
  equal[composite[img[power[x], y], img[power[x], z]],
  composite[img[power[x], z], img[power[x], y]]] == True

  equal[composite[img[power[x_, y_], img[power[x_, z_]]],
  composite[img[power[x_, z_], img[power[x_, y_]]] := True

## abstraction

Instead of introducing a third variable, we can apply abstraction to eliminate the variable y.

Map[composite[class[pair[y, z], member[z, #]], SINGLETON] &,
  composite[cross[inverse[img[power[x], y]], Id, power[x]] ==
  composite[cross[Id, img[power[x], y]], power[x]]]

  composite[SWAP, cross[SWAP, inverse[power[x]]], inverse[RIF], power[x]] ==
  composite[cross[inverse[power[x]], SWAP], inverse[RIF], power[x]]

This can be cleaned up:

Map[composite[SWAP, rotate[inverse[#]]] &%,
  composite[RIF, SWAP, cross[power[x], power[x]]] ==
  composite[RIF, cross[power[x], power[x]]]

This yields a pretty version of the statement that arbitrary powers commute:

  composite[RIF, SWAP, cross[power[x_], power[x_]]] :=
  composite[RIF, cross[power[x], power[x]]]