■ 3: The Prototype Prover

This prototype of a theorem prover by induction on polynomials was implemented to be able to test the application of the polynomial induction principle and the interaction between different provers of the *Theorema* system. It was *not* designed to create proofs as the hand typed ones that were presented in the last chapter, as it does not use sequence induction as an alternative to polynomial induction. Furthermore it assumes in advance that all variables are monomials or polynomials, and it can only be applied on equational goals. This prover was a first way of trying polynomial proofs, but its unsatisfying proof results showed the necessity to create and examine the hand typed proofs before continuing the program implementation.

■ 3.1: Description of the Output Language

The currently used language in the presentation of proofs still has many differences to the language we finally want to use. The optimum would be to generate proof notebooks in the the style we are already using in the hand typed proof notebooks, as for example in the hand typed proof of the closure of negation.

The natural English language phrases are still very fragmentaric, often only key words are used instead of full sentences, for example the header of the proof simply says "PROPOSITION:" . This will be generally quite easy to modify, since the texts are only produced in the proof postprocessor, which actually takes the proof object and produces the presented notebook. Having this post–processing procedure divided from the actual prover makes it also possible to present the proof in another language, e.g. in German, by simply translating the phrases and rearranging them, if necessary. However, it is not easy to always stick to the idea that by opening or closing subgroups the phrases stay complete. In out test examples we tried to accomplish this, but in the proofs generated by our prototype prover this does generally not hold. For us, of course, this was not a main goal, since the form of the intermediate texts do not influence proving.

The cell structure used by the prototype prover is also not yet the one we want to have, and which is shown in our manual example proofs. The current one is mainly based on what the auxiliary provers (simplification prover, case prover, predicate logic prover) deliver, which is mostly designed at an ad–hoc level. Since they all are in the process of development, this is not a main topic already, since first of all the proofs have to work.

The formulae occuring in the following proof are also printed using a slightly different language then the one we finally want to use. The main differences appear in the style the universal quantifiers are used, the labels, the names of newly generated constants, function symbols and the case statement.

The universal quantifiers are written like $\forall_{\langle p,q\rangle}$, the variables under the quantifier are displayed as subscript tuple. The final syntax of this will be $\underset{p,q}{\forall}$, where the variables are listed as underscript to the $\forall$ symbol, and the tuple constructor $\langle\ldots\rangle$ is not displayed any more.

The labels are all displayed. Finally we want to optionally hide the labels that are not used during the following proof. Similarly, human proof authors do not label each occuring formula, but only some important ones. This will also be important to immediately see whether some of the assumptions were not even needed for the proof. Currently the labels of the input formulae (assumptions and proposition) are selected by the person calling the prover interactively in a *Mathematica* session. Later on, the labels will be extracted from the corresponding definition, because the prover will not be called directly by the user any more. For example, in the following proof the label (⊕comm) of the proposition was provided by us in the argument to the call of the proof function. Finally, the string element "comm" forming this label should be taken from the label of the definition of commutativity. The labels of the function definitions amongst the assumptions have also currently been defined manually. This comes from the fact that the prototype prover does not yet extrakt these formulae from the corresponding functors, where they are initially defined.

During the proof the labels of the newly generated formulae are constructed by the prover (or the called subprover) by expanding the initial label with an appendix, separated by a dot. This ensures that all the labels are unique. These appendices are selected in a way that one can infer which proof rule was used to generate the formula, e.g. in

(⊕comm.V.IB2.IH) $\left\langle \dot{p}_0 \right\rangle \oplus \left\langle \ddot{q}_0 ,\ \overline{q_0} \right\rangle \equiv \left\langle \ddot{q}_0 ,\ \overline{q_0} \right\rangle \oplus \left\langle \dot{p}_0 \right\rangle$

the label (⊕comm.V.IB2.IH) gives the information that the formula appears in the proof of commutativity of ⊕, after the polynomial variables have been converted to their representation using sequence variables (denoted by V). Furthermore, it is the induction hypothesis (IH) in the proof of the second base case of the outermost induction (IB2).

The only deviation of this method generating labels is the way a simplification proofs looks like. It does not deal with formulae, but with terms. One could also introduce labels for terms (e.g. by appending LHS to the label of an equation to label its left hand side term). On the other hand it would not be useful to expand the term label after each of the possibly many simplification steps. The labels would simply become to large. Therefore, only the final formula is labeled, using an S as appendix. However, the labels of the applied equalities are displayed in the simplification. By this, one still can refer to a certain term, e.g. by speaking of "the term yielded by the third application of equation (⊕assoc)".

Two different proof rules need new, unique constants to be generated:
– proving a universally quantified formula by inserting an arbitrary but fixed new constant and
– proving by polynomial induction.

In the first case, the prototype prover constructs the name of the new constant by adding a natural number as subscript to the variable name. This number is then increased such that it is never used twice for this specific variable name. This ensures that never the same constant is generated twice, although the same variable can occur twice.

The second case is treated by the prototype in the following way: To create the second induction base case, a new monomial constant has to be generated. To create the induction hypothesis and conclusion, the prover has to generate two new monomial constants and a new sequence constant. For this, it takes the name of the induction variable and adds one or two dots over it in addition to adding a natural number subscript. The sequence constant is again simply composed by adding a subscript. An example for a second base case and an induction step formula generated in this way is

(⊕comm.V.IB2) $\forall_{\langle \bar{q} \rangle} \left\langle \dot{p}_0 \right\rangle \oplus \langle \bar{q} \rangle \equiv \langle \bar{q} \rangle \oplus \left\langle \dot{p}_0 \right\rangle$ and

(⊕comm.V.IS) $\forall_{\langle \bar{q} \rangle} \left\langle \dot{p}_0 ,\ \ddot{p}_0 ,\ \overline{p_0} \right\rangle \oplus \langle \bar{q} \rangle \equiv \langle \bar{q} \rangle \oplus \left\langle \dot{p}_0 ,\ \ddot{p}_0 ,\ \overline{p_0} \right\rangle$.

Here the induction variable was p, the new monomial constants are $\dot{p}_0$ and $\ddot{p}_0$ and the new sequence constant is $\overline{p_0}$. The two formulae can share one of the monomial constants, $\dot{p}_0$, since their proofs are completely independent. If one of these constants would have already existed in the proof, the new one would have got subscript one instead of zero. Note that the overbar of the sequence constant is not part of the name, but only the indicator that this is a sequence, therefore we do not write $\bar{p}_0$.

In a human textbook proof this dotted and double–dotted notation is of course not used. There, the author would have probably written

$$\forall_{\langle \bar{q} \rangle} \langle m_0 , m_1 , \overline{p_0} \rangle \oplus \langle \bar{q} \rangle \equiv \langle \bar{q} \rangle \oplus \langle m_0 , m_1 , \overline{p_0} \rangle$$

or

$$\forall_{\langle \bar{q} \rangle} \langle m_0 , n_0 , \overline{p_0} \rangle \oplus \langle \bar{q} \rangle \equiv \langle \bar{q} \rangle \oplus \langle m_0 , n_0 , \overline{p_0} \rangle$$

instead. For small proofs with few variables and constants these are surely good solutions. In our hand typed proofs we use the dotted notation, because in larger proofs each of these versions above can give rise to problems:

Using simply the next two elements of the sequence m_0, m_1, … for naming the monomial constants has the disadvantage that all the constants can be distinguished not by their symbol, but only by the subscript. And using both m and n even complicates reading, because e.g. when after double induction m_0, n_0, m_1 and n_1 are all appearing in a formula, the reader cannot easily infer any more whether the constants with same letters or the ones with the same indices were generated in one induction step. The main argument for using the dotted and double–dotted notation is that the reader always knows where the constant was introduced first.

The prototype prover also uses a function notation different to the one in the functor. This comes from the fact that the current testing environment does not yet use all elements of the Theorema language. This means that we are not allowed to use Mathematica–interpretable functions (e.g. +) any more. Instead, we use function symbols undefined in Mathematica, namely ⊕, ⊖, ⊗, ≡, ⊳ for +, −, ⋆, =, >. Furthermore we have to omit all the underscripts. This is only possible by denoting the monomial operations by different symbols. We replace $\underset{M}{+}$ by ⋄ ("Diamond") and $\underset{M}{\star}$ by · ("CenterDot"). The original > is only used for monomials, as $\underset{M}{>}$, therefore we can keep the symbol ⊳, only omitting the underscript M. And we also provisionally use the "not" function, since there is no good symbol for this that is not already used by *Mathematica,* like the ¬ .

The case construct, that will finally be written in the same syntax as it was presented in the polynomial functor, is still used in a different notation by the prototype. Here the term corresponding to the case term of $(\underset{P}{+})$ is

$$\text{case}[\langle m \rhd n,\ m \smile (\langle \bar{p} \rangle \oplus \langle n,\ \bar{q} \rangle) \rangle,\ \langle n \rhd m,\ n \smile (\langle m,\ \bar{p} \rangle \oplus \langle \bar{q} \rangle) \rangle,$$
$$\langle m \diamond n \equiv \bigcirc,\ \langle \bar{p} \rangle \oplus \langle \bar{q} \rangle \rangle,\ \langle \text{not}[m \rhd n] \wedge \text{not}[n \rhd m] \wedge \text{not}[m \diamond n \equiv \bigcirc],$$
$$m \diamond n \smile (\langle \bar{p} \rangle \oplus \langle \bar{q} \rangle) \rangle] \quad .$$

This compressed style is of course unsatisfactory, because the functor version can be read and understood easier. The main reason for us using this form in a first approach was, that there is no good way to print nested case statements in the tabular form from above, and that the prototype prover, when dealing with cases, produces very large terms of nested case statements, that can only be presented in the compressed way. This is one of the main weaknesses of the current prover, since a human prover would not generate these large case statements and show them as parts of formulae, but he would immediately do case distinction whenever such case statement occurs during proving, thus never printing the case statement itself in the proof. This is what we will show in the manual example proof later.

In the above case statement used in the prototype, each case is represented as a tuple consisting of the condition and the term value under this condition, so there only one semantic deviation to the final version: The "otherwise" condition is replaced by the condition that holds in the "otherwise" case, which is the conjunction of the negation of all previous conditions. This has to be done explicitely because the prover cannot yet do this by itself. Note that in each of the cases the semantics of the condition is, that not only this condition should hold, but also all others before should not hold. So when one goes into case distinction, one in fact would have to expand the conditions by building the conjunction with the negation of the preceding ones, to generate the correct case assumption. For example when we assume $n \underset{M}{>} m$, we also have to assume not[$m \underset{M}{>} n$]. However, for proving it is often enough to assume the one condition which is special to this case, as for example in the following commutativity proof.

A minor difference to the final language is, that definitions do not yet have the colon to distinguish between definiendum and definiens. That means, we use the normal equality instead of the combined symbol := .

The presentation of the proof notebook in this paper will be black and white, although the actual proof notebooks are generated using colors to faciliate reading:

– English language texts are printed in black, except when speaking about variables, which are printed in blue.

– Labels are colored blue, in front of the formula and when occuring in text.

– Assumptions and local hypotheses are printed in green.

– The proposition to be proved and the subgoals are printed in red.

– Terms during a simplification process are printed in purple.

■ 3.2: Details of an Example Prover Output

In the following we show the essential parts of a proof of commutativity of polynomial plus, as it is produced by the prototype prover. In this section grey background indicates parts of the automatically generated proof notebook.

■ 3.2.1: Assumptions

The proof notebook, with only the assumptions cell group opened, looks like this:

PROPOSITION:

(⊕comm) $\forall_{\langle p,q\rangle}\ p \oplus q \equiv q \oplus p$

under the assumptions

(⌣) $\forall_{\langle x,\bar{y}\rangle}\ x \smile \langle \bar{y}\rangle \equiv \langle x,\ \bar{y}\rangle$,

(⟨m⟩⊕⟨n⟩)

$\forall_{\langle m,n\rangle}\ \langle m\rangle \oplus \langle n\rangle \equiv \text{case}[\langle m \rhd n,\ \langle m,\ n\rangle\rangle,\ \langle n \rhd m,\ \langle n,\ m\rangle\rangle,\ \langle m \diamond n \equiv \bigcirc,\ \langle\rangle\rangle,,$
$\langle \text{not}[m \rhd n] \wedge \text{not}[n \rhd m] \wedge \text{not}[m \diamond n \equiv \bigcirc],\ \langle m \diamond n\rangle\rangle]$

(⊕0) $\forall_{\langle p\rangle}\ p \oplus \langle\rangle \equiv p$,

(0⊕) $\forall_{\langle p\rangle}\ \langle\rangle \oplus p \equiv p$,

(⊕)

$\forall_{\langle m,n,\bar{p},\bar{q}\rangle}\ \langle m,\bar{p}\rangle \oplus \langle n,\bar{q}\rangle \equiv \mathrm{case}[\langle m \rhd n,\ m \smile (\langle\bar{p}\rangle \oplus \langle n,\bar{q}\rangle)\rangle,\ \langle n \rhd m,\ n \smile (\langle m,\bar{p}\rangle \oplus \langle\bar{q}\rangle)\rangle,, \langle m \diamond n \equiv \bigcirc,\ \langle\bar{p}\rangle \oplus \langle\bar{q}\rangle\rangle,\ \langle \mathrm{not}[m \rhd n] \wedge \mathrm{not}[n \rhd m] \wedge \mathrm{not}[m \diamond n \equiv \bigcirc],\ m \diamond n \smile (\langle\bar{p}\rangle \oplus \langle\bar{q}\rangle)\rangle]$

(⋄comm) $\forall_{\langle m,n\rangle}\ m \diamond n \equiv n \diamond m$.

PROOF:

The prototype prover assumes that all the variables are monomials, polynomials or sequences of monomials. This means that it has to be made sure before calling it, that the input formulae hold for these types. The prover finally decides the types of the variables in the following way: In knows the type of each occuring function symbol, therefore it knows which type the function arguments must have. If one of them is a variable, this variable is assumed to have the matching type. The type of occuring sequence variables is always assumed to be "sequence of monomials".

As expected, the only polynomial and monomial assumptions needed to prove commutativity of polynomial plus are the definitions of ⊕, (⊕0), (0⊕, (⊕), and the commutativity of the monomial plus (⋄comm). As explained later, the lemma (⟨m⟩⊕⟨n⟩) is not really necessary for the proof, since it follows from the definition of ⊕. However, it makes the proof shorter, which is a main area of applications for lemmata.

The general case of ⊕ is defined a little different to how it was done in the functor. We use $m \diamond n \equiv \bigcirc$, with $\bigcirc$ for the "zero monomial", instead of $m = \underset{M}{-}n$. The reason is, that we wanted to use only equations as assumptions (and as few assumptions as possible) to test the prover. If we had used the monomial negation, we would have needed to find a new uninterpreted symbol for $\underset{M}{-}$, and to prove

$$\underset{m,n\in M}{\forall}\ m = \underset{M}{-}n \implies n = \underset{M}{-}m$$

(written in functor language here) as a lemma first.

However, introducing a zero monomial $\bigcirc$ is problematic, because then the definition of polynomials has to be changed, otherwise we get $\langle\bigcirc\rangle \in P$, which we do not want. Therefore the functor does not assume that the argument domain M has a $\underset{M}{0}$, hence the definition of $\underset{P}{+}$ cannot use it. The final prover will have to stick exactly to the definitions from the functor.

For the prototype prover, the order of the given assumptions has important influence on the proof, mainly on the simplification subproofs. This is why we cannot order the assumptions into monomial assumptions (in this case: (⋄comm)), polynomial function definitions and polynomial lemmata (here: (⟨m⟩⊕⟨n⟩)), as we would finally like to do.

The order we use here is exactly the order in which the simplifier will use the formulae. Therefore, the rule (⌣), which defines prepending an element to a tuple, comes first, since this is part of the semantics of tuples. Finally, we will not even list this equation any more, because it is not an assumption coming from a certain functor, but a built–in part of the first order predicate logic with sequence variables that we use as underlying logical language.

The lemma (⟨m⟩⊕⟨n⟩) has to come before the definitions of ⊕, because its only use is to abbreviate long simplifications. Therefore the simplifier has to try it first, before applying the definitions, otherwise the lemma would never be used.

The commutativity of monomial plus (⋄comm) has to be listed at the end, otherwise it would be used much too often during the simplifications, swapping the arguments of the ⋄ back and forth.

■ 3.2.2: Sequence Representation

As described above, the prover knows that the arguments to the $\oplus$ function have to be polynomials. By this it derives that in the proposition $\forall_{\langle p,q\rangle}\ p \oplus q \equiv q \oplus p$ the variables p and q have to be of this type. The first line of the proof use this knowledge immediately, by saying:

We first replace all list variables by their definition.

In fact, this line should say: "We represent polynomials in sequence notation.". In the formulae appearing in our proofs the "list variables" are exactly the polynomials. The new goal generated, with the label extended by "V", is

($\oplus$comm.V) $\forall_{\langle \bar{p},\bar{q}\rangle}\ \langle \bar{p}\rangle \oplus \langle \bar{q}\rangle \equiv \langle \bar{q}\rangle \oplus \langle \bar{p}\rangle$,

where all polynomial variables are replaced by tuples. The names of the sequence variables denoting the contents of the tuples are derived directly by the names of the variables, by only marking them as sequences with an overbar. So far, the prover does not check during this step whether a sequence variable already occured in the initial formula. This can lead to unexpected behaviour, for example if we want to prove

$\forall_{\langle p,p\rangle}\ p \oplus \langle \bar{p}\rangle \equiv \langle \bar{p}\rangle \oplus p$.

Actually, the prover interprets this formula like

$\forall_{\langle p\rangle}\ p \oplus p \equiv p \oplus p$,

which is not what we want. Therefore, the final prover version will have to rename variables first, in order to avoid clashes.

■ 3.2.3: Simplification

The next step of proving is trying to simplify the goal. In the proof of ($\oplus$comm) this does not change the formula, but it is anyways documented in the output. But since the simplifier only deals with ground formulae, we have to chose arbitrary but fixed constants first. The label of the generated goal gets a final "F" attached, meaning "fixed". In the proof notebook, this step adds

For this, we take all variables arbitrary, but fixed and prove

($\oplus$comm.V.F) $\langle \overline{p_0}\rangle \oplus \langle \overline{q_0}\rangle \equiv \langle \overline{q_0}\rangle \oplus \langle \overline{p_0}\rangle$.

Now the simplification prover is called, which tries to prove an equation by simplifying the terms of both sides according to the equalities in the assumptions, getting the same terms. In our case, this fails, because no simplification is possible. This result is shown in the proof as

An attempt for proving this by simplification fails.

Simplification of the lhs term:

$\langle \overline{p_0}\rangle \oplus \langle \overline{q_0}\rangle \rfloor$

Simplification of the rhs term:

$\langle \overline{q_0} \rangle \oplus \langle \overline{p_0} \rangle \rfloor$

The symbol $\rfloor$ indicates the last step of the simplification.

Although simplification did not prove the theorem, the formula still could have been changed by simplifying subterms. Therefore, we do not continue with the old goal (⊕comm.V.F), but we use the final terms of the simplification to constuct our new goal. Since in the current proof the goal is not affected by this, we will give an example to describe this step:

Assume we want to prove

(simplification example) $\forall_{\langle p,q \rangle}\ (p \oplus q) \oplus \langle\, \rangle \equiv q \oplus p$.

After the steps of inserting requence variables and taking arbitrary but fixed constants, the simplification call would look like

An attempt for proving this by simplification fails.

Simplification of the lhs term:

$(\langle \overline{p_0} \rangle \oplus \langle \overline{q_0} \rangle) \oplus \langle\, \rangle =$ by (⊕0)

$\langle \overline{p_0} \rangle \oplus \langle \overline{q_0} \rangle \rfloor$

Simplification of the rhs term:

$\langle \overline{q_0} \rangle \oplus \langle \overline{p_0} \rangle \rfloor$

The new goal is now generated by using the simplified terms and changing from constants back to variables. This yields as a new goal

(simplification example.V.S) $\forall_{\langle \bar{p},\bar{q} \rangle}\ \langle \bar{p} \rangle \oplus \langle \bar{q} \rangle \equiv \langle \bar{q} \rangle \oplus \langle \bar{p} \rangle$.

Here simplification was useful in order to make the following induction proof shorter.

■ 3.2.4: Starting the Induction on $\bar{p}$

Now, since it was not possible to do the proof by simplification, polynomial induction is applied. Since $\bar{p}$ is the first variable under the quantifier, induction is done on $\bar{p}$. The induction cell group looks like this (with all subproofs closed):

Induction base 1 for $\bar{p}$:

We prove

(⊕comm.V.IB1) $\forall_{\langle\bar{q}\rangle}\ \langle\,\rangle \oplus \langle\bar{q}\rangle \equiv \langle\bar{q}\rangle \oplus \langle\,\rangle.$

Induction base 2 for $\bar{p}$:

We try to prove by simplification:

(⊕comm.V.IB2) $\forall_{\langle\bar{q}\rangle}\ \left\langle \dot{p}_0 \right\rangle \oplus \langle\bar{q}\rangle \equiv \langle\bar{q}\rangle \oplus \left\langle \dot{p}_0 \right\rangle$

Induction hypothesis for $\bar{p}$:

(⊕comm.V.IH) $\forall_{\langle\bar{q}\rangle}\ \left\langle \ddot{p}_0,\ \overline{p_0} \right\rangle \oplus \langle\bar{q}\rangle \equiv \langle\bar{q}\rangle \oplus \left\langle \ddot{p}_0,\ \overline{p_0} \right\rangle.$

Induction step for $\bar{p}$:

We try to prove by simplification:

(⊕comm.V.IS) $\forall_{\langle\bar{q}\rangle}\ \left\langle \dot{p}_0,\ \ddot{p}_0,\ \overline{p_0} \right\rangle \oplus \langle\bar{q}\rangle \equiv \langle\bar{q}\rangle \oplus \left\langle \dot{p}_0,\ \ddot{p}_0,\ \overline{p_0} \right\rangle$

The two lines saying "We try to prove by simplification:" indicate that a simplification subproof does not work for the formula. Finally, we do not want such information on this level, only inside the subproofs, but for the moment it is good to immediately see that one of the induction goals, (⊕comm.V.IB1), can be proved by simplification only.

The proof of the first base case looks like this (with all cells opened):

For this, we take in (⊕comm.V.IB1) all variables arbitrary, but fixed and prove

(⊕comm.V.IB1.F) $\langle\,\rangle \oplus \langle\overline{q_0}\rangle \equiv \langle\overline{q_0}\rangle \oplus \langle\,\rangle.$

A proof by simplification works.

Simplification of the lhs term:

$\langle\,\rangle \oplus \langle\overline{q_0}\rangle =$ by (0⊕)

$\langle\overline{q_0}\rangle \rfloor$

Simplification of the rhs term:

$\langle\overline{q_0}\rangle \oplus \langle\,\rangle =$ by (⊕0)

$\langle\overline{q_0}\rangle \rfloor$

When the reader opens only the top level cell group, he just gets the information that the proof by simplification works. By opening the subgroups, he can examine the simplification steps and see the labels of the used assumptions.

■ 3.2.5: Second Base Case of $\bar{p}$: Induction on $\bar{q}$

The proof of the second induction base case starts like this:

Induction base 2 for $\bar{p}$:

We try to prove by simplification:

$$(\oplus\text{comm.V.IB2})\quad \forall_{\langle \bar{q}\rangle}\ \langle \dot{p}_0 \rangle \oplus \langle \bar{q} \rangle \equiv \langle \bar{q} \rangle \oplus \langle \dot{p}_0 \rangle$$

Now we prove the simplified formula

$$(\oplus\text{comm.V.IB2.S})\quad \forall_{\langle \bar{q}\rangle}\ \langle \dot{p}_0 \rangle \oplus \langle \bar{q} \rangle \equiv \langle \bar{q} \rangle \oplus \langle \dot{p}_0 \rangle$$

by induction on $\bar{q}$.

Here the unsuccessful simplification cell group is closed. Since there is still a sequence variable left under the quantifier, the prover now calls itself recursively, creating a subproof by induction over $\bar{q}$. The induction cell group looks like this (all subgroups closed):

Induction base 1 for $\bar{q}$:

$$(\oplus\text{comm.V.IB2.IB1})\quad \langle \dot{p}_0 \rangle \oplus \langle\,\rangle \equiv \langle\,\rangle \oplus \langle \dot{p}_0 \rangle.$$

Induction base 2 for $\bar{q}$:

$$(\oplus\text{comm.V.IB2.IB2})\quad \langle \dot{p}_0 \rangle \oplus \langle \dot{q}_0 \rangle \equiv \langle \dot{q}_0 \rangle \oplus \langle \dot{p}_0 \rangle.$$

Induction hypothesis for $\bar{q}$:

$$(\oplus\text{comm.V.IB2.IH})\quad \langle \dot{p}_0 \rangle \oplus \langle \ddot{q}_0, \overline{q_0} \rangle \equiv \langle \ddot{q}_0, \overline{q_0} \rangle \oplus \langle \dot{p}_0 \rangle.$$

Induction step for $\bar{q}$:

$$(\oplus\text{comm.V.IB2.IS})\quad \langle \dot{p}_0 \rangle \oplus \langle \dot{q}_0, \ddot{q}_0, \overline{q_0} \rangle \equiv \langle \dot{q}_0, \ddot{q}_0, \overline{q_0} \rangle \oplus \langle \dot{p}_0 \rangle.$$

Now we have arrived at a level where all of the formulae are ground. Therefore the subproofs will only use simplification and case distinction.

The first base case is proved by simplification, using ($\oplus$0) and (0$\oplus$) to make both sides equal to $\langle \dot{p}_0 \rangle$.

■ 3.2.6: Case Distinction

The second base case proof of the inner induction also starts with simplification, for the first time applying the general definition of $\oplus$. However, it does not use ($\oplus$) but the lemma ($\langle m\rangle\oplus\langle n\rangle$) to immediately generate

Simplification of the lhs term:

$\langle \dot{p}_0 \rangle \oplus \langle \dot{q}_0 \rangle =$ by (⟨m⟩⊕⟨n⟩)

$\mathtt{case}\Big[\big\langle \dot{p}_0 \triangleright \dot{q}_0,\ \langle \dot{p}_0,\ \dot{q}_0 \rangle\big\rangle,\ \big\langle \dot{q}_0 \triangleright \dot{p}_0,\ \langle \dot{q}_0,\ \dot{p}_0 \rangle\big\rangle,\ \big\langle \dot{p}_0 \diamond \dot{q}_0 \equiv \bigcirc,\ \langle\,\rangle\big\rangle, \rfloor$
$\big\langle \mathtt{not}[\dot{p}_0 \triangleright \dot{q}_0] \wedge \mathtt{not}[\dot{q}_0 \triangleright \dot{p}_0] \wedge \mathtt{not}[\dot{p}_0 \diamond \dot{q}_0 \equiv \bigcirc],\ \langle \dot{p}_0 \diamond \dot{q}_0 \rangle\big\rangle\Big]$

This step was now the abbreviation of applying (⊕), (⊕0), (0⊕) and (⌣) several times. In the proof of the lemma one can see which simplification steps are replaced by this one.

(A problem of presentation is that currently *Mathematica* prints the punctuation at wrong places, e.g. the ⌋ should be printed in the second line.)

Here it becomes visible that the current prototype prover deals with case statements in a way a human would never do. A human proof author would never introduce a case term as part of a formula, as the simplifier just did. Instead, he would immediately do case distinction on the different conditions inside the case term. This means, he would go through all cases, everytime assuming the corresponding condition and using the corresponding value for the case term. In our manual example proof we will show how the final prover should proceed according to this human way of proving. To achive this, we will hide all intermediate formulae with case terms from the proof notebook, since the basic "case distinction" proof rule of generating subproofs with additional assumptions is already used, as we will show below.

Simplification of the right hand side of (⊕comm.V.IB2.IB2) reveals another unwanted behaviour:

Simplification of the rhs term:

$\langle \dot{q}_0 \rangle \oplus \langle \dot{p}_0 \rangle =$ by (⟨m⟩⊕⟨n⟩)

$\mathtt{case}\Big[\big\langle \dot{q}_0 \triangleright \dot{p}_0,\ \langle \dot{q}_0,\ \dot{p}_0 \rangle\big\rangle,\ \big\langle \dot{p}_0 \triangleright \dot{q}_0,\ \langle \dot{p}_0,\ \dot{q}_0 \rangle\big\rangle,\ \big\langle \dot{q}_0 \diamond \dot{p}_0 \equiv \bigcirc,\ \langle\,\rangle\big\rangle, =$ by (⋄comm.1)
$\big\langle \mathtt{not}[\dot{q}_0 \triangleright \dot{p}_0] \wedge \mathtt{not}[\dot{p}_0 \triangleright \dot{q}_0] \wedge \mathtt{not}[\dot{q}_0 \diamond \dot{p}_0 \equiv \bigcirc],\ \langle \dot{q}_0 \diamond \dot{p}_0 \rangle\big\rangle\Big]$

⋮ (another two times of using (⋄comm.1))

$\mathtt{case}\Big[\big\langle \dot{q}_0 \triangleright \dot{p}_0,\ \langle \dot{q}_0,\ \dot{p}_0 \rangle\big\rangle,\ \big\langle \dot{p}_0 \triangleright \dot{q}_0,\ \langle \dot{p}_0,\ \dot{q}_0 \rangle\big\rangle,\ \big\langle \dot{p}_0 \diamond \dot{q}_0 \equiv \bigcirc,\ \langle\,\rangle\big\rangle, \rfloor$
$\big\langle \mathtt{not}[\dot{q}_0 \triangleright \dot{p}_0] \wedge \mathtt{not}[\dot{p}_0 \triangleright \dot{q}_0] \wedge \mathtt{not}[\dot{p}_0 \diamond \dot{q}_0 \equiv \bigcirc],\ \langle \dot{p}_0 \diamond \dot{q}_0 \rangle\big\rangle\Big]$

(Again, the = sign is printed in the wrong line.)

Here the case term is not only introduced, but its subterms are further simplified. We do not want this, because a human prover deals with these subterms only when he is already in the corresponding case subproof. In the current proof, this even leads to the situation that a human reader would expect no more work left to be done for the case distinction, as the result of simplification is

The new goal is:

(⊕comm.V.IB2.IB2.S)

$$\mathtt{case}\left[\left\langle \dot{p}_0 \triangleright \dot{q}_0,\ \left\langle \dot{p}_0,\ \dot{q}_0 \right\rangle\right\rangle,\ \left\langle \dot{q}_0 \triangleright \dot{p}_0,\ \left\langle \dot{q}_0,\ \dot{p}_0 \right\rangle\right\rangle,\right.$$
$$\left.\left\langle \dot{p}_0 \diamond \dot{q}_0 \equiv \bigcirc,\ \langle\rangle \right\rangle,\ \left\langle \mathtt{not}\left[\dot{p}_0 \triangleright \dot{q}_0\right] \wedge \mathtt{not}\left[\dot{q}_0 \triangleright \dot{p}_0\right] \wedge \mathtt{not}\left[\dot{p}_0 \diamond \dot{q}_0 \equiv \bigcirc\right],\ \left\langle \dot{p}_0 \diamond \dot{q}_0 \right\rangle\right\rangle\right] \equiv$$
$$\mathtt{case}\left[\left\langle \dot{q}_0 \triangleright \dot{p}_0,\ \left\langle \dot{q}_0,\ \dot{p}_0 \right\rangle\right\rangle,\ \left\langle \dot{p}_0 \triangleright \dot{q}_0,\ \left\langle \dot{p}_0,\ \dot{q}_0 \right\rangle\right\rangle,\ \left\langle \dot{p}_0 \diamond \dot{q}_0 \equiv \bigcirc,\ \langle\rangle \right\rangle,\right.$$
$$\left.\left\langle \mathtt{not}\left[\dot{q}_0 \triangleright \dot{p}_0\right] \wedge \mathtt{not}\left[\dot{p}_0 \triangleright \dot{q}_0\right] \wedge \mathtt{not}\left[\dot{p}_0 \diamond \dot{q}_0 \equiv \bigcirc\right],\ \left\langle \dot{p}_0 \diamond \dot{q}_0 \right\rangle\right\rangle\right]$$

By comparing both sides of the equality one detects that reordering subterms is all that has to be done to make the case terms syntactically equal, which is necessary for proving the equality. The prototype however does not yet do this syntactic analysis, but enters case distinction at this point, which looks like this (with closed subgroups):

A proof by case distinction works.

Case (⊕comm.V.IB2.IB2.S.c1) $\dot{q}_0 \triangleright \dot{p}_0$:

Case (⊕comm.V.IB2.IB2.S.c2) $\dot{p}_0 \triangleright \dot{q}_0$:

Case (⊕comm.V.IB2.IB2.S.c3) $\dot{p}_0 \diamond \dot{q}_0 \equiv \bigcirc$:

Case (⊕comm.V.IB2.IB2.S.c4) $\mathtt{not}\left[\dot{q}_0 \triangleright \dot{p}_0\right] \wedge \mathtt{not}\left[\dot{p}_0 \triangleright \dot{q}_0\right] \wedge \mathtt{not}\left[\dot{p}_0 \diamond \dot{q}_0 \equiv \bigcirc\right]$:

In these case prove headers, the locally added assumption is shown. Its label is created by appending ".c" and a serial number for each case.

Now, of course, the generated subgoals are all immediately true, for example

Case (⊕comm.V.IB2.IB2.S.c1) $\dot{q}_0 \triangleright \dot{p}_0$:

The subgoal is:

(⊕comm.V.IB2.IB2.S.G1) $\left\langle \dot{q}_0,\ \dot{p}_0 \right\rangle \equiv \left\langle \dot{q}_0,\ \dot{p}_0 \right\rangle$.

In this case the subgoal (labeled by appending ".G" and the case number) was created by searching for the case $\dot{q}_0 \triangleright \dot{p}_0$ in both case terms and taking the corresponding values. Since the prover does not yet handle the case where both sides of an equality are syntactically equal, the simplifier is called (of course successfully) to prove this. We do not show this simplification call here.

The case prover however does not see that the conditions

$$\mathtt{not}\left[\dot{p}_0 \triangleright \dot{q}_0\right] \wedge \mathtt{not}\left[\dot{q}_0 \triangleright \dot{p}_0\right] \wedge \mathtt{not}\left[\dot{p}_0 \diamond \dot{q}_0 \equiv \bigcirc\right]$$

and

$$\mathtt{not}\left[\dot{q}_0 \triangleright \dot{p}_0\right] \wedge \mathtt{not}\left[\dot{p}_0 \triangleright \dot{q}_0\right] \wedge \mathtt{not}\left[\dot{p}_0 \diamond \dot{q}_0 \equiv \bigcirc\right]$$

are equivalent, since it does not call the predicate logic prover here. Therefore in the fourth case (⊕comm.V.IB2.IB2.V.c4) only the case term at the right hand side of the equation can be evaluated, and we get:

The subgoal is:

(⊕comm.V.IB2.IB2.S.G4)

$$\mathtt{case}\Big[\big\langle \dot{p}_0 \rhd \dot{q}_0,\ \langle \dot{p}_0,\ \dot{q}_0 \rangle\big\rangle,\ \big\langle \dot{q}_0 \rhd \dot{p}_0,\ \langle \dot{q}_0,\ \dot{p}_0 \rangle\big\rangle,\ \big\langle \dot{p}_0 \diamond \dot{q}_0 \equiv \bigcirc,\ \langle\rangle\big\rangle,\ \big\langle \mathtt{not}[\dot{p}_0 \rhd \dot{q}_0] \wedge \mathtt{not}[\dot{q}_0 \rhd \dot{p}_0] \wedge \mathtt{not}[\dot{p}_0 \diamond \dot{q}_0 \equiv \bigcirc],\ \langle \dot{p}_0 \diamond \dot{q}_0 \rangle\big\rangle\Big] \equiv \langle \dot{p}_0 \diamond \dot{q}_0 \rangle$$

At this point the case prover calls itself, because there is still a case term left. Now the predicate logic prover is called, which is printed (with closed proof) as:

We show that

(⊕comm.V.IB2.IB2.S.G4.c4) $\mathtt{not}[\dot{p}_0 \rhd \dot{q}_0] \wedge \mathtt{not}[\dot{q}_0 \rhd \dot{p}_0] \wedge \mathtt{not}[\dot{p}_0 \diamond \dot{q}_0 \equiv \bigcirc]$

holds.

Only then, (⊕comm.V.IB2.IB2.V.G4) can be reduced to

(⊕comm.V.IB2.IB2.S.G4.G4) $\langle \dot{p}_0 \diamond \dot{q}_0 \rangle \equiv \langle \dot{p}_0 \diamond \dot{q}_0 \rangle$.

This now (again by calling the simplifier) proves (⊕comm.V.IB2.IB2.V.G4), the last case in the case distinction. By this the second induction base case for $\bar{q}$ is proved.

■ 3.2.7: Induction Step for $\bar{q}$: a Two Level Case Distinction

The proof of (⊕comm.V.IB2.IS) works mainly in the same way as the one of (⊕comm.V.IB2.IB2), by applying simplification and case distinction. We only want to mention the following differences:

During the first simplification, the induction hypothesis (⊕comm.V.IB2.IH) is used. The corresponding simplification sequence starts using the following sequence of formulae: (⊕), (⊕comm.V.IB2.IH), (⊕0), (⌣), (⊕0), … The induction hypothesis is used as early as possible, because otherwise it could happen that the simplifier cannot use her any more.

This simplification generates an equation with nested case terms on both sides, since (⊕) was used two times during each side's simplification. We only show part of the right hand side term:

$$\mathtt{case}\Big[\big\langle \dot{q}_0 \rhd \dot{p}_0,\ \dot{q}_0 \smile \mathtt{case}\big[\langle \dot{p}_0 \rhd \ddot{q}_0,\ \langle \dot{p}_0,\ \ddot{q}_0,\ \overline{q_0}\rangle\rangle,\ \langle \ddot{q}_0 \rhd \dot{p}_0,\ \ddot{q}_0 \smile (\langle \dot{p}_0 \rangle \oplus \langle \overline{q_0} \rangle)\rangle,\ \langle \dot{p}_0 \diamond \ddot{q}_0 \equiv \bigcirc,\ \langle \overline{q_0} \rangle\rangle,\ \langle \mathtt{not}[\dot{p}_0 \rhd \ddot{q}_0] \wedge \mathtt{not}[\ddot{q}_0 \rhd \dot{p}_0] \wedge \mathtt{not}[\dot{p}_0 \diamond \ddot{q}_0 \equiv \bigcirc],\ \langle \dot{p}_0 \diamond \ddot{q}_0,\ \overline{q_0}\rangle\rangle\big]\big\rangle,\ \ldots\Big]$$

This now leads to a case distinction where the proof of the first case is again a case distinction. This second case distinction only proves the equality of two syntactically equal case terms. A human would see this immediately, and this is also what we want from the final prover. Even using the simplifier to prove the equality of two syntactically equal terms is done very unnaturally so far, since the two terms first are reduced by the same sequence of applicable rules, and only the simplified terms are compared.

■ 3.2.8: Induction Step for $\bar{p}$: again Induction on $\bar{q}$

The proof of the induction step for $\bar{p}$ starts in the same way as the proof on the second base case: by trying a simplification proof. Since this does not work, the resulting formula

$$(\oplus\text{comm.V.IS.S}) \quad \forall_{\langle \bar{q} \rangle} \left\langle \dot{p}_0 , \ddot{p}_0 , \overline{p_0} \right\rangle \oplus \langle \bar{q} \rangle \equiv \langle \bar{q} \rangle \oplus \left\langle \dot{p}_0 , \ddot{p}_0 , \overline{p_0} \right\rangle$$

is proved by induction on $\bar{q}$. We only show the induction cell group, with closed subproofs, since the subproofs work in the same way as in the proof of the second base case for $\bar{p}$, (⊕comm.V.IB2):

Induction base 1 for $\bar{q}$:

$$(\oplus\text{comm.V.IS.IB1}) \quad \left\langle \dot{p}_0 , \ddot{p}_0 , \overline{p_0} \right\rangle \oplus \langle \, \rangle \equiv \langle \, \rangle \oplus \left\langle \dot{p}_0 , \ddot{p}_0 , \overline{p_0} \right\rangle .$$

Induction base 2 for $\bar{q}$:

$$(\oplus\text{comm.V.IS.IB2}) \quad \left\langle \dot{p}_0 , \ddot{p}_0 , \overline{p_0} \right\rangle \oplus \left\langle \dot{q}_0 \right\rangle \equiv \left\langle \dot{q}_0 \right\rangle \oplus \left\langle \dot{p}_0 , \ddot{p}_0 , \overline{p_0} \right\rangle .$$

Induction hypothesis for $\bar{q}$:

$$(\oplus\text{comm.V.IS.IH}) \quad \left\langle \dot{p}_0 , \ddot{p}_0 , \overline{p_0} \right\rangle \oplus \left\langle \ddot{q}_0 , \overline{q_0} \right\rangle \equiv \left\langle \ddot{q}_0 , \overline{q_0} \right\rangle \oplus \left\langle \dot{p}_0 , \ddot{p}_0 , \overline{p_0} \right\rangle .$$

Induction step for $\bar{q}$:

$$(\oplus\text{comm.V.IS.IS}) \quad \left\langle \dot{p}_0 , \ddot{p}_0 , \overline{p_0} \right\rangle \oplus \left\langle \dot{q}_0 , \ddot{q}_0 , \overline{q_0} \right\rangle \equiv \left\langle \dot{q}_0 , \ddot{q}_0 , \overline{q_0} \right\rangle \oplus \left\langle \dot{p}_0 , \ddot{p}_0 , \overline{p_0} \right\rangle .$$

The proof of (⊕comm.V.IS.IB2) uses the hypothesis (⊕comm.V.IH) for simplification, and the proof of (⊕comm.V.IS.IS) even uses both induction hypotheses, (⊕comm.V.IS.IH) and (⊕comm.V.IH). In this last induction step proof the disadvantage of generating all possible case subterms as soon as possible becomes obvious: After the initial simplification, each side of the equation contains seven nested case subterms, so the formula becomes unnecessarily huge. This could be avoided by generating not more than one case term on each side of the formula during one call of the simplification.