■ 4: Conclusion

■ 4.1: Review of Discussed Topics

We provided manually composed proofs from the theory of polynomials, and we discussed various aspects of proving and presentation. The following list gives an overview of the appearing concepts and topics that have been discussed for the individual proofs:

– "Zero is a Polynomial" (2.5.1): the basic notebook structure (initial declarations, goal, assumptions, proof), term simplification, ‘assumption is conclusion’

– "One is a Polynomial" (2.5.2): a more general initial declaration of domain constants

– "Negative is Closed" (2.5.3): abbreviation by omitting underscripts, type declarations, assumption groups, introducing sequence notation, applying polynomial induction, labelling, relation to inference rule of polynomial induction, linear subproofs, modus ponens, internal specialization and typechecking, hidden versus explicit proof steps

– "Lemma: Leading Monomial of a Polynomial constructed by Prepending" (2.5.4): difference of sequence and polynomial induction, proof by taking an arbitrary but fixed constant, proof by simplification

– "Lemma: Conditions for Prepending to give a Polynomial" (2.5.5): initial term simplification, generalization, undefined terms, evaluating equality, abbreviating labels

– "Plus is Closed" (2.5.6): nested induction, case distinction on definition of plus, using the case assumption, avoiding dependent assumptions, goal driven rewriting instead of pure forward simplification

– "Times is Closed" (2.5.7): different definitions of times, changed induction order

– "Zero is Neutral Element" (2.6.1): no new topics

– "Negative is an Inverse of Plus" (2.6.2): conditional rewrite rules needing subproofs

– "Lemma: Prepending is Summation" (2.6.3): no new topics

– "Plus is Commutative" (2.6.4): supporting simplification by a lemma

– "Plus is Associative" (2.6.5): nested case distinctions, versions of using case assumptions to satisfy rewrite rule conditions, replacing case distinction by applying a lemma, deciding which case distinction to use, proof by contradiction, the ‘zero’ monomial, hiding impossible cases

– "One is Neutral element" (2.7.1): influence of definition on type of induction

– "Times is Commutative" (2.7.2): case distinction on definition of times, when to apply induction hypotheses

For each of the listed topics we also presented the needed presentation style.

■ 4.2: Open Problems and Further Work

In order to finally be able to implement a polynomial prover in *Theorema*, we used example proofs to find out common patterns in the sequences of applied proof steps. Such a pattern makes it possible to clearly define which of the proof steps are task of the polynomial prover and which ones should be carried out by calling responsible auxiliary provers, that are lower in the prover hierarchy. Of course the main task of the polynomial prover is to apply polynomial induction, but we do not want to delegate all other parts of a proof to the available general provers, such as the predicate logic prover (PLP) or the general simplifier, as it was done in the prover prototype. Instead, we want to build in as much proving knowledge as possible directly into the polynomial prover, in order to increase its efficiency. For this we have to use specialized versions of auxiliary provers.

One example would be to add a case distinction subprover that is specialized to work on goal formulae containing polynomial plus. It could avoid calling the predicate logic prover by using the case assumptions locally to derive additions to the knowledge base. The proof of associativity of polynomial plus shows that even the version using the PLP needs an additional feature that is not available in the current state of the system's provers: the ability of PLP and simplifier to interact by calling each other. This is necessary because the PLP has to make use of term simplification to use equational assumptions and the simplifier has to provide the PLP with goal conditions of needed rewrite rules.

Our presented proofs also show that – in contrast to the *Theorema* prover for equalities by list induction – the simplification prover, which is based on a term order, is not powerful enough to deal with all the equational subgoals appearing in our proofs. This could make it necessary to create a more general rewriting prover, or again a specialized version of it that can only deal with the limited set of rewrite rules occuring in the polynomial proofs.

The concept of hiding 'uninteresting' proof parts from the reader, such as used variable substitutions or type checks, has to be clarified further: Which of the abbreviated subproofs can really be left out by the prover and which ones have to be done but are only suppressed when generating the output notebook? Here it is important that the capabilities of the prover stay the same, i.e. the same set of formulae can be proved.