Theory Exploration with Theorema

Bruno Buchberger

Research Institute for Symbolic Computation
Johannes Kepler University
A4040 Linz, Austria
Buchberger@RISC.Uni-Linz.ac.at

■ Theory Exploration Versus Theorem Proving

■ Why Automated Theorem Proving Has Little Impact on Mathematics

Automated theorem proving is a well established subject in computer science. The advances made in the algorithmic techniques for automated theorem proving are impressive, see the survey paper [Loveland 1996]. Also, automated theorem proving has found various important applications in computer science. For example, Robinson's resolution proving method launched a whole new programming paradigm, namely logic programming. Amazingly, although proving is the essence of mathematics, the advances of automated theorem proving did not have any noticeable influence on the practice of mathematical research and teaching so far.

We do believe, however, that automated theorem proving should, could and will have an important impact on the future of mathematics, more precisely on the way how mathematics is "done". For this to happen we think that the basic paradigm must change from *automated theorem proving* towards *computer−supportedtheory exploration*. In fact, in our view, the paradigm of theory exploration is not only the natural paradigm for using automated or computer−supported proving systems but also for doing mathematics by paper and pencil.

In this paper, we will first clarify the paradigm of theory exploration and then give a couple of examples that should illustrate the paradigm. The examples are carried out in the frame of the *Theorema* system, a system designed to be a common frame for proving, solving, and simplifying mathematical formulae, see [Buchberger et al. 1997, 2000].

■ A Rough Distinction

We first give a rough distinction between the concept of (automated) *theorem proving* and the concept of *theory exploration.*

The two essential parameters for a *prover* are the *goal* and the *knowledge base* and the question is whether the goal follows from the knowledge base:

knowledge base $\overset{?}{\vdash}$ goal

The power of automated theorem provers is measured by the class of goal formulae and knowledge base formulae for which this question can be decided and / or proofs can be generated. Efficiency, naturalness of proofs generated etc. are other criteria for assessing the quality of provers.

In correspondence with the fundamental problem of theorem proving, in *Theorema*, the basic call has the form

Prove[goal, using → knowledge-base, by → proof-method]

(Similarly, in *Theorema*, we have the calls

Solve[goal, using → knowledge-base, by → solution-method]

Simplify[goal, using → knowledge-base, by → evaluation-method]

In mathematics, proving, solving, and computing interact intimately. We do not go into details about this interaction in the current paper, though.)

The main point which we want to make in this paper is that, usually, in mathematical research, teaching, and application, *we do not consider individual theorems* and, thus, the automated theorem proving paradigm does not well reflect the typical situation. Rather, *we normally explore an entire "theory":*

- We proceed by proving "layers" of more and more complex propositions.
- In each layer, we consider (try to prove or disprove) the propositions as an ensemble and not isolated.
- In each layer, we prove using the knowledge base(s) established in the (immediately) preceding layer(s) rather than proving from "first principles".
- In each layer, we apply specific proof method that are appropriate for the knowledge bases used, i.e. instead of using one proof method for all of mathematics, one applies special proof techniques that are determined by the knowledge bases available in the various exploration situations. In other words, the knowledge available in a given exploration situation migrates from the level of being knowledge to the level of constituting a special inference technique.
- If a specific proof method is fixed, the class of goals that can be proved can be increased if goals that are already proved are added to the knowledge base. Furthermore, the sequence in which goals are proved (and added to the knowledge base) may have a drastic influence on the class of goals provable. Good strategies for "completing" knowledge are therefore of utmost importance.

The thesis of this paper is:

- Organized theory exploration instead of isolated theorem proving has a drastic, positive, influence on the feasibility, power, naturalness and practical attractiveness of automated theorem proving as a tool for working mathematicians.
- Hence, future mathematical software systems that integrate theorem proving should support theory exploration in various ways. As a consequence, we try to provide such tools in *Theorema*.

■ Exploration Situations

■ The Parameters of an Exploration Situation

Mathematical theory exploration can be explained in terms of the notion of "exploration situation". In our view, an exploration situation is characterized by the following parameters:

- a collection *C* of known concepts
- a collection *K* of facts that (are thought to) "completely" describe the interactions (interrelations) among the known concepts

- a new concept *N*
- axioms (e.g. definitions) *A* that relate the new concept with the known concepts
- a collection *G* of goal propositions that (are thought to) "completely" explore the interaction between the new concept and the known concepts.

Note that, here, the notion of a "complete" knowledge on concepts is not meant to be a formal notion. In certain contexts, for example in rewriting, "complete" knowledge may well be conceived as a "completed system" in the technical sense, which entails that the truth of sentences can be decided w.r.t. to such a system. However, although we do not give a formal definition of "completeness", the intuitive ingredient of the notion of "complete knowledge" is that, as with formally complete systems, it should be "relatively easy" (i.e. by "symbolic computing" and not by handling quantifiers) to prove or disprove all other statements on the given notions as soon as we have "complete knowledge" about them. We will make this clearer in the examples below.

■ The Proof Method Appropriate for an Exploration Situation

We believe that

- the parameters *C* (known concepts), *K* (known facts on known concepts), *N* (new concept), *A* (axioms), and *G* (goals) of an exploration situation determine an "appropriate" proof method for proving the "routine" propositions in G and
- "often", the appropriate proof method can be derived from the syntactical structure of *C, K, N, A* , and *G*.

Accordingly, one can imagine increasing levels of sophistication in (present and future) computer−supported theory exploration systems:

- For many typical exploration situations, the system provides an appropriate proof method in a library of proof methods which the user may choose according this his analysis of the exploration situation. (This is, in fact, what we provide in the current version of *Theorema*.)
- For any exploration situation, the system allows the user to formulate, in an easy "prover programming language", using also basic proof methods from a library, his proposal for what he believes is an appropriate proof method for the given situation. (We intend to work on such a prover programming language in the next phase of the *Theorema* project.)
- The system provides a "proof method composer" that composes, for (m)any exploration situations, an appropriate proof method automatically from a library of basic proof methods. Although this approach, in its most general form, is unrealistic −both theoretically and practically −it is well possible to develop proof method composers for certain contexts. (An easy example: As soon as certain of the operations involved in the exploration situation satisfy the ring axioms − a fact that can be determined by a purely syntactical analysis −,polynomial simplification can be applied as a special inference method.)

■ An Example of an Exploration Situation

Here and in the subsequent examples we use *Theorema* syntax, see [Buchberger et al., 1997, 2000].

Known concepts: ’0’ (zero), ’ $\square^+$ ’ (successor function).

Fact(s) about known concepts: induction axiom(s)

$$\forall_A \left(\left(A[0] \bigwedge \forall_x \left(A[x] \Longrightarrow A[x^+] \right) \right) \Longrightarrow \forall_x A[x] \right)$$

New concept: ’+’ (addition).

Axioms (definition) that relate the new concept with the known concepts:

Definition["Addition", any[m, n],
$m + 0 = m$ " +0"
$m + n^+ = (m + n)^+$ " +succ"]

Goal propositions: all formulae of the form

$$\forall_{x_1,\ldots,x_n} L = R$$

where *L* and *R* are terms in ’0’, ’$\square^+$’, ’+’, and the variables ’x_1’, …, ’x_n’.

■ An Appropriate Proof Method

In principle, goal propositions in the above exploration situation could be proved from the known facts and the axioms by any complete predicate logic proof method. However, instead, more naturally and efficiently, the known facts can be "lifted" from their level of being knowledge expressed in terms of formulae onto the level of being a specific inference method and this method can be used for proving the goal propositions from the empty knowledge base. Here is a rough description of this special proof method:

> Take all x_1, …, x_n "arbitrary but fixed" and try to prove ’*L=R*’ by rewriting both *L* and *R* modulo the assumed equalities (in the definition).
>
> If this proof succeeds, report success (and show the proof).
>
> If this proof fails, organize induction on x_1:
>
> > The induction base is a proof situation of the same type but with one variable less.
> > Hence call the proof method recursively.
> >
> > The induction step is a proof situation of the same type but with one variable less.
> > Hence call the proof method recursively.

In *Theorema*, this proof method has the name ’NNEqIndProver’. (Note that, in fact, the power of this method, i.e. the class of formulae that can be proved by this method, heavily depends on the specific method used for rewriting *L* and *R*.)

■ Examples of Proofs

For proving the following proposition

Proposition["Addition of Zero from Left", any[n],
$0 + n = n$]

we execute the following *Theorema* call:

Prove[**Proposition**["Addition of Zero from Left"],
using → Definition["Addition"],
by → NNEqIndProver]

This call, without setting any additional options, applies an elementary leftmost outermost simplification technique for rewriting. It generates a proof of the proposition (in an easy−to−read "natural" style). Similarly, proofs for the following propositions can be generated automatically by the above method. The proof is shown in [Buchberger et al.1997].

Proposition["Addition from Left", any[m, n],
$m^+ + n = (m + n)^+$]

Prove[**Proposition**["Addition from Left"], using → Definition["Addition"], by → NNEqIndProver]

Proposition["Associativity of Addition", any[m, n, p],
$(m + n) + p = m + (n + p)$]

Prove[**Proposition**["Associativity of Addition"], using → Definition["Addition"], by → NNEqIndProver]

The following proposition cannot yet been proved by this proof method.

Proposition["Commutativity of Addition", any[m, n],
$m + n = n + m$]

Prove[**Proposition**["Commutativity of Addition"], using → Definition["Addition"], by → NNEqIndProver]

It can be proved, however, if the knowledge base is expanded by the propositions already proved, i.e. if the following call is executed:

```
Prove[Proposition["Commutativity of Addition"],
  using → ⟨Definition["Addition"],
      Proposition["Addition of Zero from Left"],
      Proposition["Addition from Left"]⟩,
  by → NNEqIndProver]
```

Expansion ("completion") of knowledge bases is discussed in the next sections. (Alternatively, the proof method could be made more powerful by applying a more sophisticated simplification method. However, this is not the issue in this paper. Rather, the issue is discussing strategies how *fixed* proof methods should be applied in the context of investigating entire theories instead of considering isolated theorems.)

■ Complete Exploration: Bottom−Up

■ Observation

- A fixed prover determines, of course, for every fixed knowledge base a fixed set of provable formulae.
- However, without changing the prover, the collection of provable formula can increase drastically if we add proved formulae to the knowledge base. Similarly, the length of proofs may decrease drastically by the same measure.
- Also, the order in which formulae are proved (and, if proved, are added to the knowledge base) may drastically influence the proving power of the prover.

Hence, we should add a mechanism to the proof system that automatically adds proved formulae to the knowledge if sequences of goal propositions are given to a prover of the system instead of isolated goal propositions. In *Theorema*, this mechanism is implemented as an option (’ExpandKnowledge’) that can be set in the Prove call.

■ An Example of Adding Proved Formulae to the Knowledge Base

Let us first formulate a couple of possible goals in the above exploration situation:

Propositions["Equational Properties of Addition", any[m, n, p],

$0 + n = n$	"0+ "
$m^+ + n = (m + n)^+$	"^++ "
$m + n = n + m$	" + "
$(m + n) + p = m + (n + p)$	"(+) + "

]

By setting the option 'ExpandKnowledge→True' (which forces every proved goal to be added to the knowledge base) in the Prove−call, for each of the above propositions the prover is called and, if a proof is generated successfully, the proposition is added to the knowledge base. In the above example, by this simple strategy, all propositions can be proved without increasing the power of the proof method. In particular, also commutativity can be proved:

Prove[Propositions["Equational Properties of Addition"],
 using → Definition["Addition"],
 by → NNEqIndProver,
 ExpandKnowledge → True]

■ Which and How Many Formulae should we Prove for a "Complete" Exploration?

Our answer to this question is heuristic:

- All propositions (in our case, equalities) that describe all interactions of the new concept with the known concepts (and with itself) are interesting goals.
- Interactions that can be described in the form of rewrite rules are particular useful because they "algorithmize" the branch of mathematics under exploration. (I think that, in fact, most part of mathematical knowledge is of this kind.)
- Only the "immediate" interactions need to be explored because the compound interactions are iterations of immediate interactions. The proof of compound interactions should be easy (and, consequently, the propositions formulating these interactions need not be stored in the knowledge base) as soon as the intermediate interactions are completely explored.

Note that the above heuristics, does not yet give any hint about how to *invent* propositions on the immediate interactions of the new concepts with the known ones. We will speak about invention in the next section.

In the above exploration situation, where '+' is the new concept and '0' and '$^+$ ' are the known concepts and all goals are equalities, we have the following possible immediate interactions:

$0 + 0 = ?$

$0 + n = ?$

$m + 0 = ?$

$m + m = ?$

$m^+ + n = ?$

$m + n^+ = ?$

$m + (n + p) = ?$

$(m + n) + p = ?$

All other interactions, for example,

$m + (0 + n^+) = ?$

are compound.

■ One Syntactical Class At A Time

In exploration situations where the axioms (definitions) that describe the dependence of the new concept on the known concepts, syntactically, have a more complex structure (e.g. involve nested quantifiers), we suggest to proceed by decomposing the exploration situation in a sequence of exploration situations reflecting the layered composition of the axioms. Roughly, this corresponds to the principle of "structured programming" in software engineering.

As an example, we consider the definition of "limit" in analysis:

Definition["limit", any[f],
$$\text{limit}[f] := \text{æ}_{a} \forall_{\epsilon>0} \exists_{N} \forall_{n\geq N} |f[n] - a| < \epsilon$$]

We also define the corresponding unary predicate 'has-limit' (is convergent):

Definition["has limit", any[f],
$$\text{has-limit}[f] := \exists_{a} \forall_{\epsilon>0} \exists_{N} \forall_{n\geq N} |f[n] - a| < \epsilon$$]

Exploration situation:

known concepts: '+', '−','*', '| |', '<', ... on numbers, '⊕', '⊖', ... on sequences,
known facts: "complete" knowledge about the mutual interaction of these operations,
new concepts: 'limit', 'has-limit',
axioms: the above definition,
goals: all interactions of 'limit' with '⊕', '⊖', ...

Example of a goal:

Proposition["limit of sums", any[f, g], with[has-limit[f], has-limit[g]],
$$\text{limit}[f \oplus g] = \text{limit}[f] + \text{limit}[g]$$]

According to the above suggestion, we first split the definition of the new notion so that, in each partial definition, we introduce only one quantifier at a time:

Definition["1−ary limit", any[f],
$$\text{limit}[f] := \text{æ}_{a}\, \text{limit}[f, a]$$]

Definition["2−ary limit", any[f, a],
$$\text{limit}[f, a] : \Longleftrightarrow \forall_{\epsilon>0}\, \text{limit}[f, a, \epsilon]$$]

Definition["3−ary limit", any[f, a, ϵ],
$$\text{limit}[f, a, \epsilon] : \Longleftrightarrow \exists_{N}\, \text{limit}[f, a, \epsilon, N]$$]

Definition["4−ary limit",
$$\text{limit}[f, a, \epsilon, N] : \Longleftrightarrow \forall_{n\geq N} |f[n] - a| < \epsilon$$]

Let us assume that the following formulae are part of the "complete" knowledge on the known notions:

Definition["sum of sequences", any[f, g, x],
$$((f \oplus g)[x]) = (f[x] + g[x])$$]

Proposition["distance of sum", any[x, y, a, b, δ, ϵ],
$(|x - a| < \delta \wedge |y - b| < \epsilon) \Longrightarrow (|(x + y) - (a + b)| < \delta + \epsilon)$]

Proposition["max", any[M, m, n],
$(M \geq \max[m, n]) \Longrightarrow (M \geq m \wedge M \geq n)$]

Now we explore each of the "layers" separately:

Explore Lowest Layer:

We first compose the appropriate knowledge base for the lowest exploration situation, i.e. the one in which the 4−ary predicate ’limit’ is the new concept. In *Theorema*, this can be done by the following instruction:

Theory["4−ary limit",
Definition["4−ary limit"]
Definition["sum of sequences"]
Proposition["distance of sum"]
Proposition["max"]]

Now, we formulate the proposition that corresponds to the final goal proposition in the lowest layer:

Proposition["4−ary limit of sum", any[f, a, δ, M, g, b, ϵ, N],
$((\text{limit}[f, a, \delta, M] \wedge \text{limit}[g, b, \epsilon, N])$
$\Longrightarrow \text{limit}[f \oplus g, a + b, \delta + \epsilon, \max[M, N]])$]

Prove[Proposition["4−ary limit of sum"], using → Theory["4−ary limit"], by → PredicateProver]

By this call, which refers to one of the *Theorema* provers for predicate logic, a proof is generated completely automatically. The proof and the subsequent proofs can be found in [Buchberger et al., 1997, 2000]. In the same way we could "completely" explore the interaction of the 4−arypredicate ’limit’ with all known concepts.

Explore Next Layer:

We now enter a new exploration stage, whose new concept is the ternary predicate ’limit’. The 4−arypredicate ’limit’ is now considered to be "known". The proposition "4−arylimit of sum" is now part of the "known facts". We may know proceed with exploring the ternary ’limit’:

Theory["3−ary limit",
Definition["3−ary limit"]
Proposition["4−ary limit of sum"]]

Proposition["3−ary limit of sum", any[f, a, δ, g, b, ϵ],
$((\text{limit}[f, a, \delta] \wedge \text{limit}[g, b, \epsilon])$
$\Longrightarrow \text{limit}[f \oplus g, a + b, \delta + \epsilon])$]

Prove[Proposition["3−ary limit of sum"], using → Theory["3−ary limit"], by → PredicateProver]

Explore Next Layer:

Let’s assume that the following variant of the proposition "3−arylimit of sum" is also part of the new knowledge base:

Proposition["3−ary limit of sum: variant", any[f, a, g, b, ϵ],
$((\text{limit}[f, a, \epsilon/2] \wedge \text{limit}[g, b, \epsilon/2])$
$\Longrightarrow \text{limit}[f \oplus g, a + b, \epsilon])$]

Theory["2−ary limit",
Definition["2−ary limit"]
Proposition["3−ary limit of sum: variant"]]

Proposition["2−ary limit of sum", any[f, a, g, b],
((limit[f, a] ∧ limit[g, b])
⟹ limit[f ⊕ g, a + b])]

Prove[Proposition["2−ary limit of sum"], using → Theory["2−ary limit"], by → PredicateProver]

Explore Next Layer:

Theory["1−ary limit",
Definition["1−ary limit"]
Definition["has-limit"]
Proposition["2−ary limit of sum"]]

Proposition["limit of sum", with[has-limit[f], has-limit[g]],
limit[f ⊕ g] = limit[f] + limit[g]]

Prove[Proposition["limit of sum"], using → Theory["1−ary limit"], by → PredicateProver]

■ Complete Exploration: Top−Down

■ Disadvantage of the Bottom−upApproach

The bottom−upapproach for "completing" knowledge for a new concept has the major drawback that the conjectures for facts about the new concept partly have to be invented by a human. For example, the right−handside of the interaction

m + (n + p) = ?

must be guessed by the user of the system. Also, in the example of the layered approach of proving facts about the limit concept, non−trivialinvention had to go into the formulation of the propositions on each of the levels. We could either try to come up with a more powerful proof method for the goal class at hand (in fact, in [Buchberger et al. 2000], we describe a powerful new prover for concepts, like the ’limit’ concept, involving "alternating quantifiers" that allows to prove facts about ’limit’ and similar concepts from analysis without decomposing the exploration situation in fine−grain layers) or we can try a different approach that, sometimes, may be able to prove new facts for a new concept without inventing a more powerful proof method. We call this approach the "cascade" approach.

■ The Cascade Method for Theory Generation

Given a prover *P*, a fixed knowledge base *K*, and a statement *G* which we want to prove, the function "Theory-Generation- Cascade" produces a theory consisting of statements that comprise all statements of *K* plus possibly some (hopefully many) other statements that are provable from *K* by *P* including the goal *G*. Roughly, the method (which is now implemented in *Theorema*) works as follows:

Theory-Generation-Cascade[G,K,P]:

Prove[*G*, using → *K*, by → *P*].

If proof succeeds
then resulting theory is $K \bigcup \{G\}$
else (i.e. if proof fails) *analyze the points where the proof fails*
and derive a conjecture C such that G might be provable by P using $K \bigcup \{C\}$.

T := Theory-Generation-Cascade[C, K, P].

If $T = K$ (i.e. no additional theorems were provable by P using K)
then resulting theory is K
else Prove[G, using $\rightarrow T$, by $\rightarrow P$].

If proof succeeds
then resulting theory is $T \bigcup \{G\}$
else resulting theory is T.

In other words, the cascade can be "challenged" by some conjecture. On the way to proving the conjecture it may detect that certain lemmata, which would make the proof possible, are not yet proved. It conjecture these lemmata automatically, proves them and tries to continue with the main proof.

If we have an algorithmic failure analyzer for P then the theory generation cascade turns the prover P into a more powerful prover P' which in fact, in addition to being a automated theorem prover, is also a automated theorem generator.

An example for the inductive theory of equalities on natural numbers is given in [Buchberger et al. 2000].

■ The Transfer of Knowledge to the Status of Inference Rule

■ A Subtle Point

We believe that, in many exploration situations, when we move from a given exploration situation to a next exploration situation that adds a new concept, it is very important to "stream−line" the prover in the following precise sense: The result of the application of a certain part of the knowledge of the previous phase leads to "predictable" results. This observation can be used in order to invent a new proof technique that should be added to the proof method used so far and should be applied whenever this particular part of the previous knowledge base is applicable in proving facts for the new concept. The new proof technique should be applied as one, coarse−grain, inference step in the next exploration situation. Only the result of these coarse−grain steps should be shown because showing the intermediate steps would be "boring", i.e. the are not any more interesting because the pertain to the "completely explored" previous exploration situation. In other words, with this technique, parts of the knowledge base of the previous exploration step is "lifted" into the status of a special proof method in the context of the new exploration situation. We tend to believe that exactly this technique is a major ingredient into successful and natural human proving and, thus, should also be implemented in automated theorem proving (or, in our view, theory exploration) methods.

This point is subtle. In this paper, we can only try to make it clear in an easy example. A more thorough and general formulation of this principle of "lifting knowledge into the status of a special proof method" will be given in a future paper.

■ Let's Analyze a Successful Proof

Consider the following scenario:

- Complete exploration of '+' over '0', '$\square^+$': Done.
- Next exploration situation: '*' inductively defined over '+'.

Let's prove, for example, associativity of '*', knowing already some facts about '+' and '*':

Propositions["Equational Properties of Addition", any[m, n, p],

$m + 0 = m$	" +0"
$m + n^+ = (m + n)^+$	" + ^+"
$0 + n = n$	"0+ "
$m^+ + n = (m + n)^+$	"^++ "
$m + n = n + m$	" + "
$(m + n) + p = m + (n + p)$	"(+) + "

]

Propositions["Some Equational Properties of Multiplication", any[m, n],

$m * 0 = 0$	" *0"
$m * n^+ = (m * n) + m$	" * ^+"
$0 * n = 0$	"0*"
$m^+ * n = (m * n) + n$	" ^+*"

]

Proposition["Associativity of Multiplication", any[m, n, p],
$(m * n) * p = m * (n * p)$ "(*) * "]

Prove[Proposition["Associativity of Multiplication"],
using →
⟨Propositions["Some Equational Properties of Multiplication"], Propositions["Equational Properties of Addition"]⟩,
by → NNEqIndProver]

The automatic proof, including explanatory natural language text, of the above proposition is a couple of pages long. Now we analyze the longest part of the proof:

Simplification of the lhs term:

$(n_3^+ + m_1 * n_3^+) * p_6^+ =$ by (* ^+)

$(n_3^+ + m_1 * n_3^+) * p_6 + (n_3^+ + m_1 * n_3^+) =$ by ((*) * .IS.S.IS.IH)

$(n_3^+ * p_6 + (m_1 * n_3^+) * p_6) + (n_3^+ + m_1 * n_3^+) =$ by (* ^+)

$(n_3^+ * p_6 + (m_1 * n_3 + m_1) * p_6) + (n_3^+ + m_1 * n_3^+) =$ by (* ^+)

$(n_3^+ * p_6 + (m_1 * n_3 + m_1) * p_6) + (n_3^+ + (m_1 * n_3 + m_1)) =$ by (^+*)

$((n_3 * p_6 + p_6) + (m_1 * n_3 + m_1) * p_6) + (n_3^+ + (m_1 * n_3 + m_1)) =$ by (^++)

$((n_3 * p_6 + p_6) + (m_1 * n_3 + m_1) * p_6) + (n_3 + (m_1 * n_3 + m_1))^+ =$ by (+ ^+)

$(((n_3 * p_6 + p_6) + (m_1 * n_3 + m_1) * p_6) + (n_3 + (m_1 * n_3 + m_1)))^+ =$ by (+)

$((n_3 + (m_1 * n_3 + m_1)) + ((n_3 * p_6 + p_6) + (m_1 * n_3 + m_1) * p_6))^+ =$ by (+)

$((n_3 + (m_1 + m_1 * n_3)) + ((n_3 * p_6 + p_6) + (m_1 * n_3 + m_1) * p_6))^+ =$ by (+)

$((n_3 + (m_1 + m_1 * n_3)) + ((p_6 + n_3 * p_6) + (m_1 * n_3 + m_1) * p_6))^+ =$ by (+)

$((n_3 + (m_1 + m_1 * n_3)) + ((p_6 + n_3 * p_6) + (m_1 + m_1 * n_3) * p_6))^+ =$ by ((+) +)

$(((n_3 + (m_1 + m_1 * n_3)) + (p_6 + n_3 * p_6)) + (m_1 + m_1 * n_3) * p_6)^+$ ⌋

Simplification of the rhs term:

$n_3^+ * p_6^+ + (m_1 * n_3^+) * p_6^+ =$ by (* ^+)

$(n_3{}^+ * p_6 + n_3{}^+) + (m_1 * n_3{}^+) * p_6{}^+ =$ by (* ^+)

$(n_3{}^+ * p_6 + n_3{}^+) + ((m_1 * n_3{}^+) * p_6 + m_1 * n_3{}^+) =$ by (* ^+)

$(n_3{}^+ * p_6 + n_3{}^+) + ((m_1 * n_3 + m_1) * p_6 + m_1 * n_3{}^+) =$ by (* ^+)

$(n_3{}^+ * p_6 + n_3{}^+) + ((m_1 * n_3 + m_1) * p_6 + (m_1 * n_3 + m_1)) =$ by (^+*)

$((n_3 * p_6 + p_6) + n_3{}^+) + ((m_1 * n_3 + m_1) * p_6 + (m_1 * n_3 + m_1)) =$ by (+ ^+)

$((n_3 * p_6 + p_6) + n_3)^+ + ((m_1 * n_3 + m_1) * p_6 + (m_1 * n_3 + m_1)) =$ by (^++)

$(((n_3 * p_6 + p_6) + n_3) + ((m_1 * n_3 + m_1) * p_6 + (m_1 * n_3 + m_1)))^+ =$ by (+)

$((n_3 + (n_3 * p_6 + p_6)) + ((m_1 * n_3 + m_1) * p_6 + (m_1 * n_3 + m_1)))^+ =$ by (+)

$((n_3 + (p_6 + n_3 * p_6)) + ((m_1 * n_3 + m_1) * p_6 + (m_1 * n_3 + m_1)))^+ =$ by (+)

$((n_3 + (p_6 + n_3 * p_6)) + ((m_1 * n_3 + m_1) + (m_1 * n_3 + m_1) * p_6))^+ =$ by (+)

$((n_3 + (p_6 + n_3 * p_6)) + ((m_1 + m_1 * n_3) + (m_1 * n_3 + m_1) * p_6))^+ =$ by (+)

$((n_3 + (p_6 + n_3 * p_6)) + ((m_1 + m_1 * n_3) + (m_1 + m_1 * n_3) * p_6))^+ =$ by ((+) +)

$(((n_3 + (p_6 + n_3 * p_6)) + (m_1 + m_1 * n_3)) + (m_1 + m_1 * n_3) * p_6)^+ =$ by (+)

$(((m_1 + m_1 * n_3) + (n_3 + (p_6 + n_3 * p_6))) + (m_1 + m_1 * n_3) * p_6)^+ =$ by ((+) +)

$((((m_1 + m_1 * n_3) + n_3) + (p_6 + n_3 * p_6)) + (m_1 + m_1 * n_3) * p_6)^+ =$ by (+)

$(((n_3 + (m_1 + m_1 * n_3)) + (p_6 + n_3 * p_6)) + (m_1 + m_1 * n_3) * p_6)^+ \rfloor$

We see that, in fact, most of the time is spent in applying associativity and commutativity of +. Furthermore, "it is clear" what the outcome of these AC−simplificationsteps is, i.e. we can predict the outcome. In other words, we can invent an easy algorithm that produces the outcome of these AC−simplificationsteps a priori:

- view the term as composed of ’+’ on the highest level and ’elementary terms’ that start with a symbol other than ’+’
- put all parentheses to the left
- order the elementary terms according to some fixed ordering.

Note that this algorithm is not just a reformulation of the AC−steps."Prediction algorithms" can be arbitrarily far away from the inference rules whose effect is predicted and may involve some high−levelnontrivial mathematics. (Examples: Groebner bases simplification for boolean combination of equalities over the complex numbers; Collins’ algorithm for arbitrary formulae of the theory of real closed fields; etc.)

■ Consequence: Transfer of Knowledge to the Status of an Inference Rule

Consequence: In this exploration situation, an appropriate prover should use a special "black box" inference rule for ’+’ applying the above prediction algorithm and not show details of applying this inference rule because this is "uninteresting" in the present exploration situation, in which our attention is on exploring ’*’. In contrast, ’+’ is already supposed to be "completely explored".

Hence, we cancel AC for '+' in the knowledge base

```
Propositions["Equational Properties of Addition", any[m, n, p],
    m + 0 = m                 " +0"
    m + n^+ = (m + n)^+       " + ^+"
    0 + n = n                 "0+ "
    m^+ + n = (m + n)^+       "^++ "]
```

and, instead, add AC of addition as a special inference rule:

```
Prove[Proposition["Associativity of Multiplication"],
      using →
    ⟨Propositions["Some Equational Properties of Multiplication"], Propositions["Equational Properties of Addition"]⟩,
      built-in –> ⟨Property[+ → {Associative, Commutative}]⟩,
      by → NNEqIndProver]
```

Of course, adding AC for addition as an inference rule needs extra programming work in the prover, it does not "come for free". The critical part of the proof now looks as follows:

Simplification of the lhs term:

$p_5^+ + (m_1 + (n_3 + m_1 * n_3)) * p_5^+ =$ by (Special Simpl)

$(m_1 + (n_3 + m_1 * n_3)) * p_5^+ + p_5^+ =$ by (* ^+)

$((m_1 + (n_3 + m_1 * n_3)) * p_5 + (m_1 + (n_3 + m_1 * n_3))) + p_5^+ =$ by (Special Simpl)

$(m_1 + (n_3 + (m_1 * n_3 + (m_1 + (n_3 + m_1 * n_3)) * p_5))) + p_5^+ =$ by (+ ^+)

$((m_1 + (n_3 + (m_1 * n_3 + (m_1 + (n_3 + m_1 * n_3)) * p_5))) + p_5)^+ =$ by (Special Simpl)

$(m_1 + (n_3 + ((p_5 + (m_1 + (n_3 + m_1 * n_3)) * p_5) + m_1 * n_3)))^+ =$ by ((*) * .IS.S.IS.S.IH)

$(m_1 + (n_3 + ((p_5 + (n_3 * p_5 + (m_1 + m_1 * n_3) * p_5)) + m_1 * n_3)))^+ =$ by (Special Simpl)

$(m_1 + (n_3 + (p_5 + (m_1 * n_3 + (n_3 * p_5 + (m_1 + m_1 * n_3) * p_5)))))^+$ ⌋

Simplification of the rhs term:

$p_5^+ + (n_3 * p_5^+ + (m_1 + m_1 * n_3) * p_5^+) =$ by (Special Simpl)

$n_3 * p_5^+ + ((m_1 + m_1 * n_3) * p_5^+ + p_5^+) =$ by (* ^+)

$(n_3 * p_5 + n_3) + ((m_1 + m_1 * n_3) * p_5^+ + p_5^+) =$ by (Special Simpl)

$n_3 + ((m_1 + m_1 * n_3) * p_5^+ + (p_5^+ + n_3 * p_5)) =$ by (* ^+)

$n_3 + (((m_1 + m_1 * n_3) * p_5 + (m_1 + m_1 * n_3)) + (p_5^+ + n_3 * p_5)) =$ by (Special Simpl)

$(m_1 + (n_3 + (m_1 * n_3 + (n_3 * p_5 + (m_1 + m_1 * n_3) * p_5)))) + p_5^+ =$ by (+ ^+)

$((m_1 + (n_3 + (m_1 * n_3 + (n_3 * p_5 + (m_1 + m_1 * n_3) * p_5)))) + p_5)^+ =$ by (Special Simpl)

$(m_1 + (n_3 + (p_5 + (m_1 * n_3 + (n_3 * p_5 + (m_1 + m_1 * n_3) * p_5)))))^+$ ⌋

One sees that long sequences of AC−steps for addition have been replaced by one invocation of the AC−simplifier for addition (marked by '(Special Simpl)'). The proof becomes faster, shorter, easier to read and "more interesting" because

it concentrates on the proof steps for the new concept ’*’ and omit details about using facts about the known concepts ’+’, ’$^{+}$ ’ and ’0’.

■ This Transfer Principle is Ubiquitous in Human Practical Proving

If one ananlyzes lots of "real−life"exploration situations, one detects that this lifting principle is not an exotic and special happening but, rather, the main principle by which human exploration of entire theories in the definition−proposition-proof style becomes practically feasible.

Other examples:

- In fact, we already tacitly applied this principle right at the beginning of this paper when we formulated induction as a proof method instead of putting induction axioms to the knowledge base and using a general predicate logic prover.
- Explore properties of ’Σ’ as a function, then introduce ’Σ’ as a quantifier and transfer the properties proved about the function ’Σ’ into special inference rules for the quantifier ’Σ’.
- Proceed further and develop an algebraic theory and a corresponding algorithm for checking the identity of ’Σ’−expressions, see the Zeilberger−Paule theory, see [Paule xxxx]. Call to this algorithm "whenever appropriate", see an example in [Buchberger et al. 2000].
- Explore rewrite properties of ’lim’, ’df/dx’, ’$\int$’ first for these functions. Then introduce the corresponding quantifiers and transfer the properties proved about the functions into special inference rules for the quantifiers.
- Proceed further and develop an algebraic theory and a corresponding algorithm for checking the integrability of certain classes of functions defined by certain classes of expressions, see [Risch xxx] . Call to this algorithm "whenever appropriate".
- Develop a theory of polynomial ideal membership and a membership (congruence, ...) algorithm based on it, see the author’s Groebner bases theory. Call this algorithm "whenever appropriate", see the demo of this method in [Buchberger et al. 2000].
- Develop a theory of "cylindric algebraic decompositions" of the real space w.r.t. systems of polynomials and develop an algorithm based on it for checking the truth of arbitrary formulae of the theory of real closed fields, see [Collins xxxx].Call this algorithm "whenever appropriate".

There is a seamless transition from near−at−hand special inference rules to gigantic black box decision procedures for special classes of formulae. However, the logics of integrating special inference mechanisms, which we briefly describe in the next subsection, is always the same.

■ A Logically Sound Integration of Special Inferencing

We start from *one* logic system, e.g. a variant of higher order predicate logic. For this system we have a semantical notion of logical consequence.

Whenever we establish a special prover *P*, we must specify two essential things:

- the class of knowledge bases *K* and goals *G* for which *P* can be applied
- the "tacit (implicit, built−in,...) theory" *T* under which correctness of *P* is guaranteed.

In more detail: For a correct special prover *P* for the theory *T* the following correctness theorem for *T* must be proved:

If *P* proves *G* from *K*, then *G* must be a logical consequence of *K* and *T*.

The correctness theorem for *T* is a meta−theorem.

For many important special provers (various special simplifiers, Collins' method, Groebner Bases method, Zeilberger method, ...) the correctness theorem is well established in the mathematical literature.

These proofs could also be given using a formalization of (parts of) logic within the proof system itself. (A challenge! Example: Formal proof of Groebner bases theory is one of the motivations for *Theorema*.)

However, note that the proof of the correctness theorem for the special prover *P* will normally be much harder then the proofs of the theorems that are automatically proved by *P*. ("A practical version of Goedel's second theorem".)

■ Avoiding Correctness Proofs for Special Provers

Formal proofs of the correctness theorem for provers (as for any other algorithms) −and their implementation ! − may be difficult.

Most people just "trust" the correctness of special proof technique proved "manually" in the literature and "carefully implemented" using the literature.

One can, however, also consider a special proof technique as a "conjecture generating black box". For each "proof" π generated by a prover relying on the special proof technique one would then have to check the correctness of each black box step by proving it by a prover that does not rely on the special proof technique but on a general proof technique with the corresponding implicit theory *T* in the knowledge base.

■ Conclusion

We argued that, for "real−life"application of proving / solving / simplifying systems, these systems should support the "theory exploration" paradigm rather than concentrate only on the "isolated theorem proving" paradigm. (Example: At the annual CADE conference, traditionally, a "single theorem proving" competition is organized. Instead, I think one should rather have a "Computer−supportedtheory exploration" competition!)

We discussed some strategies and approaches to facilitate theory exploration:

- Practical facilities for manipulating large amounts of formal text.
- Establishing provers that are "appropriate" for a given exploration situation.
- Complete exploration by bottom−up analysis of all possible interactions (hopefully in the form of algorithmic formulae) of new concepts with given concepts.
- Decomposition of goals with nested alternating quantifiers.
- Top−down conjectures generation by challenging provers and (automated) analysis of failing proofs (the "cascade").
- Transfer of knowledge to the status of proof methods as the main principle for "human−like"exploration and generation of short proofs.

We are currently working on a next version of *Theorema* that enables the user to apply these strategies for theory exploration.

■ References

[Buchberger et. al. 1997] B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, D. Vasaru, W. Windsteiger, 1997.
An Overview on the Theorema project. In: *Proceedings of ISSAC'97 (International Symposium on Symbolic and Algebraic Computation, Maui, Hawaii, July 21−23,1997)*, W. Kuechlin (ed.), ACM Press 1997, pp. 384−391.

[Buchberger et. al. 2000] B. Buchberger, C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, W. Windsteiger, 2000. The *Theorema* Project: A Progress Report. In: *8th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning*, St. Andrews (Scotland), August 6−7, 2000, M. Kerber, M. Kohlhase (eds.), available from Fachbereich Informatik, Univeristät des Saarlandes, pp. 100−115.

[Collins 1975] G. Collins, 1975.
Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. In: Second GI Conference on Automata Theory and Formal Languages, LNCS, Vol. 33, Springer Berlin, pp. 134−183.

[Paule, Schorn 1995] P. Paule, M. Schorn, 1995.
A Mathematica Version of Zeilberger's Algorithm for Proving Binomial Coefficient Identities. *J. Symbolic Computation*, Vol. 20, pp. 673−698.

[Loveland 1995] D. W. Loveland, 1996.
Automated Deduction: Some Achievements and Future Directions, *Report of a Workshop on the Future Directions of Automated Deduction*, Chicago, April 20−21,1996, http://www.cs.duke.edu/AutoDedFD.