Scheme–Based Systematic Exploration of Natural Numbers *

Mădălina Hodorog and Adrian Crăciun
Institute e-Austria
Timişoara, Romania
{mhodorog, acraciun}@ieat.ro

Abstract

In this paper, we report a case study of computer supported exploration of the theory of natural numbers, using a theory exploration model based on knowledge schemes, proposed by Bruno Buchberger.

We illustrate with examples from the exploration: (i) the invention of new concepts (functions, relations) in the theory, using knowledge schemes, (ii) the invention of new propositions, using proposition schemes, (iii) the invention of problems, using knowledge schemes, (iv) the introduction of new reasoning rules, by lifting knowledge to the inference level, after their correctness was proved.

1 Introduction

Mathematical knowledge is acquired through the exploration of theories. This should become clear as early as the training period of the mathematicians, when they go through various theories under the guidance of an instructor. Later on, the mathematicians will explore theories on their on, then move to develop new mathematics, based on the knowledge so far.

One of the goals of the recent field of Mathematical Knowledge Management (MKM) is to provide computer support for this type of activity. See [7], [2], [1] for an overview of the MKM community research. Recently, Bruno Buchberger proposed a model for theory exploration based on knowledge schemes, see [3, 9]. In this model, theories are developed in exploration rounds by:

- adding new concepts to the theory, using definition schemes,
- adding new conjectures about the notion introduced, using proposition schemes,
- inventing problems about concepts, using problem schemes and giving solutions, using algorithm schemes,
- inventing new inference rules, usually by lifting knowledge to the level of inference rules.

Remark. As it will turn out, the same scheme can be used, depending on the context, as a definition, proposition, problem, or algorithm scheme.

Examples of exploration cycles carried out with this exploration model, and focused on problem solving/algorithm synthesis can be seen in [10](synthesis in the theory of tuples), [6], [9].

In this paper we consider the exploration of natural numbers (Peano system). We chose this case study because it is a benchmark for any mathematical reasoning system, and it should illustrate the advantages of the scheme–based exploration approach. We compare this case study to that carried out by Manna and Waldinger in [18]. If we manage to produce similar content, then we believe the exploration model will have proved successful in this case.

The case study is carried out in the Theorema system, and we will use its notational convention. For more details concerning the system, see [11], [9].

In Section 2 we present the context of our case study: the logic, the language of natural numbers, our notion of a theory, then natural numbers expressed in the language in this setting, and describe the library of schemes available in the exploration process. Section 3 contains various examples from the exploration: adding notions, conjectures, problems using schemes, and lifting knowledge to the inference level. Section 4 is concerned with implementation issues: implementation of new provers needed to carry out the exploration, organization of the exploration. We consider related work in Section 5, and present our conclusions and future directions of research in Section 6.

*Work supported by EU Marie Curie Project MERG-CT-2004-012718: SysteMaThEx and by Romanian CNCSIS project 632/2006: SEPROPI.

2 Context

2.1 Language, Knowledge Base, Inference Rules, Theories

The language setting in which we express our exploration is that of predicate logic. Some considerations have to be made, however, about which predicate logic we use. This will be untyped, with type (sort) information being handled by the use of unary predicates.

To express a mathematical theory, we use a *first order predicate logic language with equality*, similar to those described in [20], [4], or [18]. The first order language $\mathcal{L}$ describing the theory is a triple: $\mathcal{L} = \left\langle \mathcal{P}, \mathcal{F}, \mathcal{C} \right\rangle$, where $\mathcal{P}$ is the set of predicate symbols (including the binary equality "=" predicate, and one or more unary predicates describing the "type" of objects in a theory), $\mathcal{F}$ is the set of function symbols (including the unary identity "id" function), $\mathcal{C}$ is the set of constants of the theory. For practical reasons, we distinguish between function symbols and constants, although the latter can be seen as 0–ary functions.

The *knowledge base* $\mathcal{KB}$ of the theory consists of a collection of first–order formulae, built over the language (axioms, properties).

The *inference mechanism* $\mathcal{IR}$ of the theory consists of the reasoning system corresponding to the theory. This will include, for any theory, the first order predicate logic calculus and rewriting (due to first order logic with equality being our language frame) and specific (i.e. dependent on axiomatization of a theory) inference rules.

To summarize, in our context a *theory* $\mathcal{T}$ is a triple $\mathcal{T} = \left\langle \mathcal{L}, \mathcal{KB}, \mathcal{IR} \right\rangle$.

2.2 The Theory of Natural Numbers

We now describe the theory of natural numbers, $\mathcal{T}_{\mathbb{N}}$.

The language of natural numbers,

$$\mathcal{L}_{\mathbb{N}} = \left\langle \langle \textit{is-nat}, = \rangle, \langle {}^{+}, id \rangle, \langle 0 \rangle \right\rangle,$$

where *is-nat* is the unary predicate symbol that characterizes natural numbers, $^{+}$ is an unary function symbol (the successor function), the identity and equality symbols (unary and binary, respectively).

The knowledge base $\mathcal{KB}_{\mathbb{N}}$ corresponding to the initial formulation of the theory consists of the equality axioms (available in any theory formulated in our setting), and the axioms characterizing the natural numbers (Peano):

$\texttt{Axioms}$["equality:naturals", $\texttt{any}[\textit{is-nat}[x, y, z]]$,
$x = x$ "reflexivity"
$(x = y) \Leftrightarrow (y = x)$ "symmetry"
$((x = y) \wedge (y = z)) \Rightarrow (x = z)$ "transitivity"],
$((x = y) \wedge \mathfrak{p}[x]) \Leftrightarrow \mathfrak{p}[y]$ "prd subst for $\mathfrak{p}$"
$(x = y) \Rightarrow (\mathfrak{f}[x] = \mathfrak{f}[y])$ "fnc subst for $\mathfrak{f}$"

$\texttt{Axioms}$["generation", $\texttt{any}[\textit{is-nat}[x]]$,
$\textit{is-nat}[0]$ "gen. zero"
$\textit{is-nat}[x^{+}]$ "gen. succ."],

$\texttt{Axioms}$["uniqueness", $\texttt{any}[\textit{is-nat}[x, y]]$,
$x^{+} \neq 0$ "zero"
$(x^{+} = y^{+}) \Leftrightarrow (x = y)$ "succ."],

$\texttt{Axioms}$["induction principle",
$(\mathfrak{F}[0] \wedge \underset{\textit{is-nat}[x]}{\forall}(\mathfrak{F}[x] \Rightarrow \mathfrak{F}[x^{+}]) \Rightarrow \underset{\textit{is-nat}[x]}{\forall}\mathfrak{F}[x]$.

Remark. In the above, Axiom["equality:naturals"], "prd subst for $\mathfrak{p}$", "fnc subst for $\mathfrak{f}$", and Axiom["induction principle"] are in fact *axiom schemes*. These can not be expressed in first order predicate logic. We have indicated in each case that certain symbols are *metavariables*, or *higher order variables* (depending on the choice of formalism) – $\mathfrak{f}$, $\mathfrak{p}$, $\mathfrak{F}$ stand respectively for any function symbol, predicate symbol, or formula where the argument is a free variable. We will not use these axioms schemes as such, but will **lift them to the inference level**, see below.

The inference mechanism $\mathcal{IR}_{\mathrm{N}}$ corresponding to the theory consists of *the structural induction rule*, *general predicate logic inference rules*, and *inference rules for equality* (simplification, rewriting).

The structural induction rule can be **lifted** from Axiom["induction principle"] in a straightforward manner. The sequent description of the inference rule is:

$$\frac{KB \vdash F_{x \leftarrow 0} \qquad \textit{is-nat}[x_0] \;\; \textit{abf}, KB \bigcup F_{x \leftarrow x_0} \vdash F_{x \leftarrow x_0{}^{+}}}{KB \vdash \underset{\textit{is-nat}[x]}{\forall} F},$$

i.e. in order to prove the universally quantified goal, prove the base case (substituting 0 for the variable), then assume the goal formula true for an arbitrary but fixed value (*abf*), and prove the goal for its successor.

In a similar fashion, the axiom schemes for equality are also lifted to the level of inference.

2.3 Knowledge Schemes

In our setting, *knowledge schemes* are formulae that capture "interesting" mathematical knowledge at various levels of abstraction (although arguably, the notion of what is interesting is hard to make precise).

These schemes are stored in libraries of schemes, and are used by instantiating them with symbols from the language of the theory being developed. We consider a global library, containing schemes at the highest level of abstraction (i.e. not depending on any theory), and other libraries that are dependent of the theory being developed (i.e. some of the higher order variables from schemes in the global library were instantiated).

To formulate knowledge schemes, we need higher order predicate logic. A scheme (formula) will contain higher order function and predicate variables. When using the schemes in the development of the theory, we have to make sure that the instantiation gets rid of all these higher order variables. The THEOREMA language allows higher order formulae, so schemes can be formulated.

Here we give some examples of knowledge schemes developed for our libraries of schemes.

Examples of schemes to be stored in the global library (i.e. independent of any theory) are those describing algebraic structures, such as:

$$\underset{p,op}{\forall}(\textit{is-semigroup}[p,op] \Leftrightarrow \underset{p[x,y,z]}{\forall} \bigwedge \left\{ \begin{array}{l} p[op[x,y]] \\ op[x,op[y,z]] = op[op[x,y],z] \end{array} \right.),$$

$$\underset{p,op,zero}{\forall}(\textit{is-monoid}[p,op,zero] \Leftrightarrow \underset{p[x,y,z]}{\forall} \bigwedge \left\{ \begin{array}{l} \textit{is-semigroup}[p,op] \\ op[x,zero] = x \end{array} \right.),$$

$$\underset{p,op,zero,inv}{\forall}(\textit{is-group}[p,op,zero,inv] \Leftrightarrow \underset{p[x]}{\forall} \bigwedge \left\{ \begin{array}{l} \textit{is-monoid}[p,op,zero,inv] \\ op[x,inv[x]] = zero \end{array} \right.),$$

or relational structures, such as:

$$\underset{p,r}{\forall}(\textit{is-preorder}[p,r] \Leftrightarrow \underset{p[x,y,z]}{\forall} \bigwedge \left\{ \begin{array}{l} r[x,x] \\ (r[x,y] \wedge r[y,z]) \Rightarrow r[x,z] \end{array} \right.).$$

$$\underset{p,r}{\forall}(\textit{is-partial-ordering}[p,r] \Leftrightarrow \underset{p[x,y,z]}{\forall} \bigwedge \left\{ \begin{array}{l} \textit{is-preorder}[p,r] \\ (r[x,y] \wedge r[y,x]) \Rightarrow x = y \end{array} \right.).$$

Examples of knowledge schemes that are dependent on the theory being developed - natural numbers in our case:

$$\underset{f,g,h}{\forall}(\textit{is-rec-nat-binary-fct-1r}[f,g,h] \Leftrightarrow \underset{is\text{-}nat[x,y]}{\forall} \bigwedge \left\{ \begin{array}{l} f[x,0] = g[x] \\ f[x,y^+] = h[f[x,y]] \end{array} \right.),$$

$$\underset{f,g,h}{\forall}(\textit{is-rec-nat-binary-fct-1l}[f,g,h] \Leftrightarrow \underset{is\text{-}nat[x,y]}{\forall} \bigwedge \left\{ \begin{array}{l} f[0,y] = g[y] \\ f[x^+,y] = h[f[x,y]] \end{array} \right.),$$

$$\underset{f,g,h,pred}{\forall}(\textit{is-nat-rec-binary-rel-2}[f,g,h] \Leftrightarrow \underset{is\text{-}nat[x,y,z]}{\forall} \bigwedge \left\{ \begin{array}{l} f[x,0] \Leftrightarrow g[x] \\ f[x,y^+] \Leftrightarrow (h[x,y] \vee f[x,y]) \end{array} \right.).$$

Remark. The recursive definitions schemes above all correspond to the (structural) induction axiom scheme. In fact, when reasoning about these schemes, we will use the structural induction rule.

Moreover, these recursive schemes are instances of a general recursive scheme, generated systematically. It is, therefore, not necessary to store all these, but they can be generated "just-in-time" when they are needed.

3 Exploration

3.1 Exploration Rounds in the Scheme–Based Model

For the case study we carried out, we used the following methodology to explore notions, consistent with [3]:

- *Introduce a new notion using a definition scheme*: from one of the libraries of schemes, choose one that is applicable (i.e. it can be instantiated with the function/predicate symbols from the language). Note that several instantiations could be possible, leading to the introduction of several corresponding notions.
- *Introduce equivalent definitions by exploring matching schemes*: Select schemes that "match" the one used to introduce the notion that is being explored (i.e. same arity, same auxiliary functions used in the definition). Then prove (or disprove) that the instantiation of this scheme is a consequence of the definition.
- *Explore structural properties of the notion, using structural schemes*: structural schemes (such as those presented in the previous section), when instantiated, represent properties that describe the interaction of the notion being explored with other notions from the theory.
- *Introduce problems, and solve them* (if this is possible): problems are introduced by instantiation of schemes, or introducing formulae where notions that are not in the language occur. These notions will have to be *synthesized*, for instance by an application of the *lazy thinking method*, see [5, 10, 8].

Carrying out these steps, in our experience, provides a fairly complete range of properties of the explored no-

tion (as compared to our benchmark example, [18]). Of course, the goal is to have a "complete" exploration of the notion (i.e. containing all "interesting" knowledge), yet such a completeness measure is hard to define formally. We will not make any further effort towards achieving this completeness, but propose the use of the lazy thinking method to "cover" the missing propositions, if they are needed in subsequent rounds of exploration (i.e. theorem synthesis, see [15] for further discussion of this, including its limitations).

3.2 Examples: Exploration of Natural Numbers

In this section, we give examples of exploration steps from the developement of the natural numbers theory using the scheme-based model.

Introducing a new notion

We start with the language described in Subsection 2.2.

For introducing a new notion in the language, we search in the scheme library for a definition scheme that can be instantiated with the symbols of the language. The first scheme that matches this description is the *is-rec-nat-binary-fct-1r* scheme. It introduces into the theory a new notion (in the scheme denoted by the variable f) - a binary function symbol, expressed in terms of the unary function symbols g, h. There are several possible instantiations:

$$\begin{array}{l}\{\{f \to \oplus, g \to id, h \to {}^{+}\}, \{f \to \boxplus, g \to {}^{+}, h \to {}^{+}\},\\ \{f \to \odot, g \to id, h \to id\}, \{f \to \boxdot, g \to {}^{+}, h \to id\}\}\end{array},$$

where the higher order variable f is in each case substituted with a new (constant) symbol ($\oplus, \boxplus, \odot, \boxdot$ respectively) and g, h are substituted with symbols from the language.

We invent the following new notions:

$$\textit{is-rec-nat-binary-fct-1r}[\oplus, id, {}^{+}] \Leftrightarrow \underset{\substack{\textit{is-nat}[x]\\ \textit{is-nat}[y]}}{\forall} \bigwedge \left\{ \begin{array}{l} x \oplus 0 = x \\ x \oplus y^{+} = (x \oplus y)^{+} \end{array} \right. ,$$

$$\textit{is-rec-nat-binary-fct-1r}[\boxplus, {}^{+}, {}^{+}] \Leftrightarrow \underset{\substack{\textit{is-nat}[x]\\ \textit{is-nat}[y]}}{\forall} \bigwedge \left\{ \begin{array}{l} x \boxplus 0 = x^{+} \\ x \boxplus y^{+} = (x \boxplus y)^{+} \end{array} \right. ,$$

$$\textit{is-rec-nat-binary-fct-1r}[\odot, id, id] \Leftrightarrow \underset{\substack{\textit{is-nat}[x]\\ \textit{is-nat}[y]}}{\forall} \bigwedge \left\{ \begin{array}{l} x \odot 0 = x \\ x \odot y^{+} = x \odot y \end{array} \right. ,$$

$$\textit{is-rec-nat-binary-fct-1r}[\boxdot, {}^{+}, id] \Leftrightarrow \underset{\substack{\textit{is-nat}[x]\\ \textit{is-nat}[y]}}{\forall} \bigwedge \left\{ \begin{array}{l} x \boxdot 0 = x^{+} \\ x \boxdot y^{+} = x \boxdot y \end{array} \right. .$$

We illustrate further exploration of the $\oplus$ function symbol.

Introducing propositions (I) - equivalent definitions

We now try to find potential equivalent definitions for the $\oplus$ function symbol. The scheme *is-rec-nat-binary-fct-1l*, is similar, in that it has the same arity with $\oplus$, and is expressed in terms of subfunctions that can be instantiated with symbols in the language. Therefore it is a candidate for the introduction of equivalent definitions. One of its instantiations ($\{f \to \oplus, g \to id, h \to {}^{+}\}$, now expressed as a THEOREMA proposition) gives:

```
Proposition["is-rec-nat-binary-fct-1l:id, +",
            any[is-nat[x, y]],
  0 ⊕ x = x                    "right zero"
  x⁺ ⊕ y = (x ⊕ y)⁺             "right succesor" ],
```

which can be proved invoking the following command in THEOREMA:

```
Prove[Proposition["is-rec-nat-binary-fct-1l:id, +",
      using → Theory["nat.1.2.0"],
      by → NatProverPC]
```

In the call above `Theory["nat.1.2.0"]` corresponds to the current knowledge base (axioms plus the definition of the $\oplus$ symbol), NatProverPC is a THEOREMA user prover combining structural induction with natural deduction and rewriting. For visualizing the proof, see [17].

The reader will note that no other schemes or instantiations give any other consequences.

Introducing propositions (II) - semigroup, monoid

To introduce other propositions, we look at the structural algebraic knowledge schemes that can be instantiated with the $\oplus$ function symbol and the other symbols from the language.

The first such scheme, *is-semigroup* with the substitution $\{p \to \textit{is-nat}, op \to \oplus\}$, yields the following:

$$\texttt{Proposition}[\text{“is-semigroup is-nat,}\oplus\text{”},\ \texttt{any}[\textit{is-nat}[x,y]],\ (x\oplus y)\oplus z = x\oplus(y\oplus z)\quad\text{“associativity”}\ ],$$

which is proved by the system, see [17].

The next scheme in the hierarchy of algebraic schemes, *is-monoid*, instantiated with $\{p \rightarrow \textit{is-nat}, \textit{op} \rightarrow \oplus, \textit{zero} \rightarrow 0\}$, yields:

$$\textit{is-monoid}[\textit{is-nat}, \oplus, 0] \Leftrightarrow \underset{\textit{is-nat}[x]}{\forall} \bigwedge \begin{cases} \textit{is-semigroup}[\textit{is-nat}, \oplus] \\ x \oplus 0 = x \end{cases},$$

for which we have to prove the neutral element formula, in addition to what we had previously. This holds, and is proved by the system.

Introducing propositions (III) - group

Going further in the hierarchy of schemes, we have *is-group*, which can be instantiated with:

$$\{p \rightarrow \textit{is-nat}, \textit{op} \rightarrow \oplus, \textit{zero} \rightarrow 0, \textit{inv} \rightarrow \textit{id}\}\ \text{, or}$$
$$\{p \rightarrow \textit{is-nat}, \textit{op} \rightarrow \oplus, \textit{zero} \rightarrow 0, \textit{inv} \rightarrow\ ^+\}.$$

This yields, in addition to what we had before:

$$\underset{\textit{is-nat}[x]}{\forall}(x \oplus x = 0), \text{ and}$$
$$\underset{\textit{is-nat}[x]}{\forall}(x \oplus x^+ = 0), \text{ respectively.}$$

None of the above formulae hold, i.e. neither the identity nor the successor are inverses for the natural numbers. The language does not contain any other unary function symbol. The only chance that natural numbers form a group is that there is some other unary function, not yet in the theory, that is an inverse. This means that if such function exists, we shall have to *invent* it and introduce it in the language.

Introducing and solving a problem

We want to invent a new function symbol, say $\ominus$, such that *is-group*$[\textit{is-nat}, \oplus, 0, \ominus]$, i.e. the following formula holds:

$$\underset{\textit{is-nat}[x]}{\forall}(x \oplus (\ominus x) = 0).$$

We now apply the *lazy thinking* method to synthesize the $\ominus$ function. If we are successfull, we add the new function to the theory. In order to synthesize the function, we attempt to prove the above formula, and if the proof fails, we analyze the proof and generate conjectures that will allow the proof to succeed. These conjectures are specifications for the unknown function symbol. In this case we will not use algorithm schemes.

Proof (inverse). Prove

$$\underset{\textit{is-nat}[x]}{\forall}(x \oplus (\ominus x) = 0),$$

using $\mathcal{KB}_{\mathbb{N}}$.

We try to prove by structural induction on x:
Base Case:

Prove $0 \oplus (\ominus 0) = 0$.

By $\texttt{Proposition}[\text{“is-rec-nat-binary-fct-1l:id, }^+\text{”}].1$, we have to prove: $\ominus 0 = 0$.
[*The proof fails, but the conjecture is straightforward:* $\ominus 0 = 0$.]
Induction Step:

Take x_0 arbitrary but fixed, such that $\textit{is-nat}[x_0]$.

Assume $x_0 \oplus \ominus x_0 = 0$.

Show ${x_0}^+ \oplus \ominus {x_0}^+ = 0$.

By $\texttt{Proposition}[\text{“is-rec-nat-binary-fct-1l:id, }^+\text{”}].2$, this is equivalent to proving: $(x_0 \oplus \ominus {x_0}^+)^+ = 0$. However, this is not true, because it contradicts $\texttt{Axiom}[\text{“uniqueness”}]:\text{“zero”}$.

Therefore, the goal is not valid.

□

Remark. The proof presented above is similar to the one generated by the THEOREMA system. $\mathcal{KB}_{\mathbb{N}}$ contains the knowledge in the theory. The part contained in the square brackets, in italics is not part of the proof, it represents the analysis and generation of the conjectures (which, in this case was straightforward). Because we can establish that our goal introduces a contradiction, our problem has no solution, i.e. there is no inverse function for the natural numbers, i.e. natural numbers cannot form a group.

Summary

We have now carried out some steps in the exploration of the notion $\oplus$. The reader will have recognized that this notion is the natural number addition.

Using the scheme-based model we have introduced all the propositions involving $\oplus$ that were present in the benchmark we considered, [18]. Some of these were not included in this presentation, see the forthcoming technical report [17] for all the details.

We continued the exploration, introducing in the same manner new concepts, like multiplication, $*$ (and related concepts, like 1), smaller-than relation (both weak, $\leq$, and strict, $<$), subtraction $-$, exponentiation.

The reader may wonder what happened to concepts introduced when $\oplus$ was introduced. These were all investigated, but $\oplus$ turned out to be the most “interesting”, in that it fulfills more structural properties than the others.

So far, we have illustrated almost all of the steps in the scheme-based exploration model. But not the addition of new inference rules. The general observation is that such exploration steps are rare, i.e. new inference rules do not get added as often as new concepts or propositions involving concepts. In the following, we illustrate how one could prove the correctness of such a new rule.

Introducing a new inference mechanism

Consider an extension of the initial theory whose language contains the symbol $<$, defined (in THEOREMA) as:

```
Definition["is-nat-rec-binary-rel-2:F,=",
        any[is-nat[x], is-nat[y]],
        (x ≠ 0)
        (x < y⁺) ⇔ ((x = y) ∨ (x < y))  ].
```

Note that this definition was introduced using the *is-nat-rec-binary-rel-2* scheme. The reader will note this is the usual strict less relation on natural numbers. We also consider the propositions related to $<$, including the fact that $\underset{is\text{-}nat[x]}{\forall}(x < x^+)$, see [17] for details.

We now introduce the following proposition which we prove using THEOREMA:

```
Proposition["Complete Induction,"
  ∀_{is-nat[x,z]} ((z < x ⇒ 𝔉[z]) ⇒ 𝔉[x]) ⇒ ∀_{is-nat[x]} 𝔉[x]  ].
```

Our proof follows the one in [18], i.e. we prove:

```
Proposition["Complete Induction.1",
  ∀_{is-nat[y,z]} (z < y ⇒ 𝔉[z]) ⇒ ∀_{is-nat[x]} 𝔉[x]  ],
```

and then, prove:

```
Proposition["Complete Induction.2",
  ∀_{is-nat[x,z]} ((z < x ⇒ 𝔉[z]) ⇒ 𝔉[x]) ⇒
                  ∀_{is-nat[y,z]} (z < y ⇒ 𝔉[z])  ].
```

For details of the proofs, see [17]. Not surprisingly, the proofs generated by THEOREMA are similar to those in [18].

The reader may notice that the formulae we proved are, in fact, not first order. However, if we consider $\mathfrak{F}[x]$ in the above as being an *arbitrary but fixed formula constant*, where x appears as a free variable, and taking into account the properties of substitutions, then the proof steps applied in our proofs remain correct. However, at the moment this argument is carried out outside of the laguage of THEOREMA. Mechanisms like reflection (see [3]) that are being developed will allow reasoning on the metalevel in the same language frame.

Since the process of lifting the formulae we proved to the inference level is, at least in this case, straightforward, similar to what we described in Subsection 2.2, the proof of these formulae constitutes the proof of correctness of the complete induction inference rule. Of course, the algorithm for lifting the knowledge to the level of inference has to also be proven correct (again, using reflection).

Remark. Once we have the $<$ (strict less than equal) relation symbol into the theory, we can introduce the above complete induction inference mechanism. Moreover, we can also add new knowledge schemes to the library of schemes, corresponding to the new induction.

The new recursive knowledge schemes are formulated using the $<$ relation symbol. Here we give examples of such schemes:

$$\underset{f,g,h}{\forall}(\textit{is–nat–step–recl–fct–1–1}[f,g,h] \Leftrightarrow \underset{\substack{is\text{–}nat[x,y]\\ y>0}}{\forall}(f[x,y] = \begin{cases} g[x] & \Leftarrow x < y \\ h[f[x-y,y]] & \Leftarrow \text{otherwise} \end{cases})).$$

$$\underset{r,q,s,const}{\forall}(\textit{is–nat–step–recr–rel–0–1–1}[r,q,s,const] \Leftrightarrow \underset{is\text{–}nat[x,y]}{\forall}(r[x,y] \Leftrightarrow \begin{cases} const & \Leftarrow y = 0 \\ q[x,y] & \Leftarrow x > y \\ s[r[x,y-x]] & \Leftarrow \text{otherwise} \end{cases})).$$

More exploration rounds (I) - solving a problem

In the following, we include some more examples of exploration that make use of the new recursive schemes introduced above. Due to the limited space available for this presentation, we only give an overview, and point the reader to the papers containing the details.

Consider now a new problem:

$$\underset{is\text{–}nat[x,y]}{\forall}(x = y * \textit{quot}[x,y]),$$

i.e., for any 2 natural numbers x, y, find a decomposition of x by multiplication, involving y.

We now try to solve this new problem, by lazy thinking, proposing as a solution for *quot*, an instantiation of *is–nat–step–recl–fct–1–1*, and using the complete induction principle to carry out the proofs.

The proof, however, does not work for the base case of the definition:

For arbitrary x_0, y_0 with $\textit{is–nat}[x_0, y_0], y_0 > 0$, in case

$$x_0 < y_0,$$

we have to prove

$$x_0 = y_0 * g[x_0].$$

However, since $\textit{is–nat}[g[x_0]]$, from the properties of $<$, this is impossible.

But analyzing the failure of this proof, we realize that a slight modification of the problem will avoid this failure. For the modified problem:

$$\underset{is\text{-}nat[x,y]}{\forall}(x = y * quot[x,y] + rem[x,y]),$$

apply again the lazy thinking method, with the *is–nat–step–recl–fct–1–1* as a candidate for both *quot* and *rem*, we obtain as solutions to our problem the well known quotient and remainder functions for natural numbers:

$$\texttt{Algorithm}[\text{“quotient”}, \texttt{any}[is\text{-}nat[x], is\text{-}nat[y]],$$
$$\text{quot}[x,y] = \left\{ \begin{array}{ll} 0 & \Leftarrow x < y \\ \text{quot}[x-y,y]+1 & \Leftarrow \text{otherwise} \end{array} \right.],$$

$$\texttt{Algorithm}[\text{“remainder”}, \texttt{any}[is\text{-}nat[x], is\text{-}nat[y]],$$
$$\text{rem}[x,y] = \left\{ \begin{array}{ll} x & \Leftarrow x < y \\ \text{rem}[x-y,y] & \Leftarrow \text{otherwise} \end{array} \right.].$$

For the complete details, we point the user to the technical report, [16].

More exploration rounds (II) - introducing a notion

We now introduce another notion in our theory: the scheme *is–nat–step–recr–rel–0–1–1* with the substitution $\{r \to |, const \to True, q \to False, s \to id_B\}$, yields the following:

$$\texttt{Definition}[\text{“divides relation symbol”},$$
$$\texttt{any}[is\text{-}nat[x], is\text{-}nat[y]],$$
$$x|y = \left\{ \begin{array}{ll} \texttt{True} & \Leftarrow y = 0 \\ \texttt{False} & \Leftarrow x > y \\ x|(y-x) & \Leftarrow otherwise \end{array} \right.],$$

where id_B is the identity function on the boolean domain (with truth values).

To introduce new propositions, we look at the relational knowledge schemes that can be instantiated with the | relation symbol.

The first such scheme, *is-preorder* instantianted with $\{p \to is\text{-}nat, r \to |\}$ yields:

$$\texttt{Proposition}[\text{“is-preorder is-nat,|”},$$
$$\texttt{any}[is\text{-}nat[x,y,z]],$$
$$\begin{array}{ll} (x|x) & \text{“reflexivity”} \\ (x|y \wedge y|z) \Rightarrow (x|z) & \text{“transitivity”} \end{array}],$$

which we can prove using Theorema, see [17].

The second scheme in the hierarchy of relational schemes, *is-partial-ordering* with the substitution $\{p \to is\text{-}nat, r \to |\}$, yields the following:

$$is\text{-}partial\text{-}ordering[is\text{-}nat, |] \Leftrightarrow$$
$$\underset{is\text{-}nat[x]}{\forall} \bigwedge \left\{ \begin{array}{l} is\text{-}preorder[is\text{-}nat, |] \\ (x|y \wedge y|x) \Rightarrow x = y \end{array} \right. ,$$

for which we have to prove the antisymmetry formula, in addition to what we had before. The proof is successful.

4 Implementation

We implemented the induction rules presented in this paper as new Theorema basic provers. These provers are combined with basic provers for natural deduction, simplification or generalized rewriting into user provers (such as `NatProverPC`) that correspond to the inference engine of the theory.

The structural induction basic prover implements the rule over natural numbers described in Subsection 2.2. The user can influence the behaviour of the prover by setting options:

- the *NNLang* option specifies the language to be used to formulate the theory of natural numbers (with the default language formed from the constant 0, the *is-nat* and = predicate symbols and the *id* and $^+$ function symbols);
- the *NNRepres* option specifies the used natural numbers representation. The natural numbers can either be constructed from 0 and $^+$ symbols or from 0 and +1 (in this case, we admit that the symbol 1 is already introduced in the language. The user can choose the notation $x+1$ instead of the notation x^+).

The second basic prover implements the complete induction principle over natural numbers. One option can be specified for this prover: the *NNLang* option where the user can specify the language components needed (default values are *is-nat* and < predicate symbols).

We have also implemented a new function `UseScheme[scheme, substitutions]`. This prototype function takes as arguments the knowledge scheme *scheme* and the list *substitutions* of all the possible combinations between all the symbols from the current language and generates the definitions of the new notions (i.e. function symbols, relations symbols or propositions) that can be constructed.

5 Related Work

The theory of natural numbers is present in any major system that aims at doing mathematical reasoning by computer. We will not, therefore, compare our implementation to all these systems. The provers

we wrote in order to carry out this research are based on the existing THEOREMA natural number induction provers, see [12]. However, our provers allow the user to use any language (s)he chooses, and we implemented more induction rules, such as the complete induction.

Our approach is not concerned with the formalization of natural numbers, but with the exploration of the theory, i.e. "invention". The HR system [13] was also proposed for the invention of mathematics, and was applied for number theory. However, the method is different. Mathematical concepts are formed by model checking, from examples, and proved by a theorem prover.

A more recent approach, MATHsAID, see [19], builds systematically a theory by proving consequences of the initial axioms, and tries to filter the interesting concepts and theorems.

Other approaches for theory invention were proposed, with various degrees of success. For an overview of various approaches and a comparison, see [14].

In comparison to the approaches mentioned, ours provides more control to the user (also helped by the natural style, textbook-like of the THEOREMA system), and we believe it is more suitable to the exploration of more complex theories. This remark is based both on results obtained so far, and on experience from the field of program synthesis, where it was shown that the use of algorithm schemes produces much better results.

Schemes, as concentrated mathematical knowledge, provide an indication of the interestingness of mathematical concepts, and help guide the exploration process.

6 Conclusion and Future Work

We reported on a case study in scheme based theory exploration: the natural numbers. We have shown that Buchberger's model can be successfully applied, and we have done so using the THEOREMA system. We have illustrated typical exploration steps, i.e. introducing a notion, introducing properties of this notion, introducing and solving problems (lazy thinking) and a possible approach to the problem of proving inference rules correct.

Although this is work in progress, we have already explored notions like addition, multiplication, the less than relation, exponentiation, subtraction, quotient, remainder. We are looking forward to the further exploration of notions like prime numbers, and set as one of our future goals the "invention" of the prime decomposition theorem. We measure our success by comparison to our benchmark, a well known textbook, [18], and so far we match this textbook. Our case study is not merely a formalization of this book, but the resulting theory is obtained by systematic use of knowledge schemes.

This approach, we believe, also has a strong didactic value. Regardless of the presence or not of a system to carry out these exploration steps, we have a methodology for theory exploration. In this paper, we described the basic steps to be carried out in an exploration round.

For the case study at hand, we implemented an updated version of the THEOREMA induction prover, that includes new induction rules introduced during the exploration. The purpose of this paper was not to describe how THEOREMA carries out the proofs, therefore we pointed the user, for this purpose, to the technical reports [17, 16]. However, all the proofs mentioned were carried out using the system.

We have also implemented prototype functions that should help in the exploration process (such as `UseScheme`).

So far, we did the exploration (i.e. introducing notions and propositions, problems, etc. theory formation, in fact) basically by hand, with only some of the steps supported by the system (like the `UseScheme` function). It is clear, however, that to carry out interesting case studies, the user will need tools to help the process of exploration: tools to carry out exploration steps, tools to query the libraries of schemes, to generate "just-in-time" schemes (including generation of unique names for schemes), to query the language and the knowledge base, and even to automate whole exploration rounds. We think that, in principle, exploration rounds such as the ones we illustrated in this paper could be carried out automatically by the system. THEOREMA already provides a uniform language frame, natural style deduction, easy to use interface, and such exploration tools would extend the system in a natural way. The implementation of such tools is one of the long-term future research points.

The experience in carrying out this case study, also illustrated in this paper, is that the richer the language of the theory is, the more possibilities to introduce notions we have. This leads to an explosion of possible exploration branches, as expected. Our opinion is that a system that provides support for theory exploration as described in this paper, should be used as a tool by mathematicians, who should be responsible for the development of the theory, including the control of the explosion of the exploration branches.

Moreover, we believe that practical deployment of such systems, at least after a few rounds of bottom–up theory formation, will be focused on problem solving (top–down exploration), making heavy use of the lazy

thinking method.

The first author has carried out the details of this case study, under the guidance of the second author. The implementation of the first versions of the induction provers, prototype exploration tools, are joint work.

We acknowledge the contribution of our colleagues in the SYSTEMATHEX project: Cristina Codreşi, Vlad Isac and Diana Pop, especially in the development of the knowledge scheme libraries, and helpful discussions.

References

[1] A. Asperti, G. Bancerek, and A. Trybulec, editors. *Mathematical Knowledge Management: Third International Conference MKM 2004 Bialowieza, Poland, September 19-21, 2004*, volume 3119 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2004.

[2] A. Asperti, B. Buchberger, and J. Davenport, editors. *Mathematical Knowledge Management: Second International Conference, MKM 2003, Bertinoro, Italy, February 16-18, 2003*, volume 2594 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2003.

[3] B. Buchberger. Algorithm–supported mathematical theory exploration: A personal view and strategy. *Lecture Notes in Artificial Intelligence, Springer*, 7th Conference on Artifiicial Intelligence and Symbolic Computation(Research Institute for Symbolic Computation, Hagenberg, Austria)(Proceedings of AISC 2004):16, September.

[4] B. Buchberger. Logic for computer science. lecture notes, 1991.

[5] B. Buchberger. Algorithm invention and verification by lazy thinking. In D. Petcu, V. Negru, D. Zaharie, and T. Jebelean, editors, *Proceedings of SYNASC 2003, 5th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing Timisoara*, pages 2–26, Timisoara, Romania, 1-4 October 2003. Copyright: Mirton Publisher.

[6] B. Buchberger. Towards the automated synthesis of a groebner bases algorithm. *RACSAM - Revista de la Real Academia de Ciencias (Review of the Spanish Royal Academy of Science), Serie A: Mathematicas*, 98(1):65–75, 2004.

[7] B. Buchberger and O. Caprotti, editors. *First International Workshop on Mathematical Knowledge Management (MKM 2001)*, September 2001.

[8] B. Buchberger and A. Craciun. Algorithm synthesis by lazy thinking: Using problem schemes. In D.Petcu, V.Negru, D.Zaharie, and T.Jebelean, editors, *Proceedings of SYNASC 2004, 6th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing Timisoara*, pages 90–106, Timisoara, Romania, 26-30 September 2004. Copyright: Mirton Publisher.

[9] B. Buchberger, A. Craciun, T. Jebelean, L. Kovacs, T. Kutsia, K. Nakagawa, F. Piroi, N. Popov, J. Robu, M. Rosenkranz, and W. Windsteiger. Theorema: Towards computer-aided mathematical theory exploration. *Journal of Applied Logic*, pages –, 2005. To appear.

[10] B. Buchberger and A. Crăciun. Algorithm synthesis by lazy thinking: Examples and implementation in theorema. In F. Kamareddine, editor, *Electronic Notes in Theoretical Computer Science*, volume 93, pages 24–59, 18 February 2004. Proc. of the Mathematical Knowledge Management Workshop, Edinburgh, Nov. 25, 2003.

[11] B. Buchberger, C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, and W.Windsteiger. Theorema: A progress report. Technical Report 99-42, RISC Report Series, University of Linz, Austria, December 1999. Also available as SFB Report 99-35, Johannes Kepler University Linz, Spezialforschungsbereich F013, December, 1999.

[12] B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, E. Tomuta, and D. Vasaru. A survey of the *theorema* project. Technical Report 97-15, RISC Report Series, University of Linz, Austria, March 1997.

[13] S. Colton. *Automated Theory Formation in Pure Mathematics*. Distinguished Dissertations. Springer Verlag, 2002.

[14] S. Colton, A. Bundy, and T. Walsh. On the notion of interestingness in automated mathematical discovery. *IJHCS: International Journal of Human-Computer Studies*, 53, 2000.

[15] A. Crăciun. *The Lazy Thinking Approach to Algorithm Synthesis: Implementation and Case Studies in Theorema*. PhD thesis, Research Instiute for Symbolic Computation, Johannes Kepler University, Linz, 2006, forthcoming.

[16] A. Crăciun and M. Hodorog. The quotient–remainder theorem for naturals: Discovery by lazy thinking. Technical Report no.09-06, I-eAT, 2006.

[17] M. Hodorog. Scheme–based systematic exploration of mathematical theories. case study: The natural numbers. Technical Report no.07-06, I-eAT, 2006.

[18] Z. Manna and R. Waldinger. *The Logical Basis for Computer Programming, Volume I, Deductive Reasoning*. Addison-Wesley Publishing Co, SUA, 1985.

[19] R. McCasland, A. Bundy, and P. Smith. Ascertaining mathematical theorems. *Electronic Notes in Theoretical Computer Science*, 151(1), 2006.

[20] J. R. Shoenfield. *Mathematical Logic*. Addison-Wesley Publishing Co, SUA, 1967.