Scheme-Based Synthesis of Inductive Theories

Scheme-Based Synthesis of Inductive Theories

Omar Montano-Rivas, Roy McCasland, Lucas Dixon, and Alan Bundy

School of Informatics, University of Edinburgh
{O.Montano-Rivas,rmccasla,ldixon,bundy}@inf.ed.ac.uk

Abstract. We describe an approach to automatically invent/explore new mathematical theories, with the goal of producing results comparable to those produced by humans, as represented, for example, in the libraries of the Isabelle proof assistant. Our approach is based on 'schemes', which are terms in higher-order logic. We show that it is possible to automate the instantiation process of schemes to generate conjectures and definitions. We also show how the new definitions and the lemmata discovered during the exploration of the theory can be used not only to help with the proof obligations during the exploration, but also to reduce redundancies inherent in most theory formation systems. We implemented our ideas in an automated tool, called IsaScheme, which employs Knuth-Bendix completion and recent automatic inductive proof tools. We have evaluated our system in a theory of natural numbers and a theory of lists.

Keywords: Mathematical theory exploration, schemes, theorem proving, term rewriting, termination.

1 Introduction

Mathematical theory exploration consists of inventing mathematical theorems from a set of axioms. It also includes the definition of new concepts. For example, in the theory of natural numbers we can define addition using successor, multiplication using addition, exponentiation using multiplication and so on. Once we have these new concepts of interest we can start conjecturing their properties and proving them.

A diversity of theory exploration computer programs have been implemented [12,5,13] and different approaches have been identified [16]. A recent approach, scheme-based mathematical theory exploration [2], has been proposed and its implementation is being undertaken within the *Theorema* project [3].

In [6], is described a case study of mathematical theory exploration in the theory of natural numbers using the scheme-based approach. However, apart from this paper there is, to our knowledge, no other case study of scheme-based mathematical theory exploration. In the Theorema system, which was used to carry out the aforementioned case study, the user had to provide the appropriate substitutions (Theorema cannot perform the possible instantiations automatically). The authors also pointed out that the implementation of some provers was still in progress and that the proof obligations were in part 'pen-and-paper'. From

G. Sidorov et al. (Eds.): MICAI 2010, Part I, LNAI 6437, pp. 348–361, 2010.

this observation, a natural question arises: whether the instantiation process of schemes and the proof obligations induced by the conjectures and definitions can be mechanized?

The main contribution of this paper is to give a positive answer to the above question. The scheme-based approach gives a basic facility to instantiate schemes (or rather higher-order variables inside schemes) with different 'pieces' of mathematics (terms built on top of constructor and function symbols) already known in the theory. In section 2 we discuss some motivating examples for the generation of conjectures and definitions using schemes. In order to soundly instantiate the schemes it is necessary to pay attention to the type of objects being instantiated. In sections 3 and 4 we show how this can be performed rigorously and with total automation on top of the simply typed lambda calculus of Isabelle/HOL [14]. To facilitate the process of proof construction, Isabelle provides a number of automatic proof tools. Tools such as the *Simplifier* [15] or *IsaPlanner* [8] can help with the proof obligations for conjectures in the process of theory exploration. Isabelle also has strong definitional packages such as the *function package* [11] that can prove termination automatically for many of the functions that occur in practice. Section 5 shows how new definitions and the lemmata discovered during the exploration of the theory can be used not only to strengthen the aforementioned tools, but also to reduce redundancies inherent in most theory formation systems (section 6). In section 7 we describe our theory exploration algorithms where the processes of theorem and definition discovery are linked together. The evaluation is described in section 8. The related and future work are discussed in sections 9 and 10 respectively and the conclusions in section 11.

2 Motivating Examples

The central idea of scheme-based mathematical theory exploration is that of a scheme; i.e. a higher-order formula intended to capture the accumulated experience of mathematicians for discovering new pieces of mathematics. The invention process is carried out through the instantiation[1] of variables within the scheme. As an example, let $\mathcal{T}_N$ be the theory of natural numbers in which we already have the constant function zero (0), the unary function successor (suc) and the binary function addition ($+$) and let s be the following scheme which captures the idea of a binary function defined recursively in terms of other functions.

$$\left(\begin{array}{l} def\text{-}scheme(\mathbf{g},\mathbf{h},\mathbf{i},\mathbf{j}) \equiv \\ \qquad \exists f.\ \forall x\, y.\ \bigwedge \left\{ \begin{array}{l} f(\mathbf{g},y) = \mathbf{h}(y) \\ f(\mathbf{i}(x),y) = \mathbf{j}(y, f(x,y)) \end{array} \right. \end{array}\right) \tag{1}$$

Here the existentially quantified variable f stands for the new function to be defined in terms of the variables $\mathbf{g}$, $\mathbf{h}$, $\mathbf{i}$ and $\mathbf{j}$. We can generate the definition of

[1] In Theorema, the instantiation process is limited to function, predicate or constant symbols already known in the theory. Additionally, IsaScheme can use any well-formed closed term of the theory including λ-terms such as $(\lambda x.\ x)$.

multiplication by allowing the theory $\mathcal{T}_{\mathcal{N}}$ to instantiate the scheme with $\sigma_1 = \{\mathbf{g} \mapsto 0, \mathbf{h} \mapsto (\lambda x.\, 0), \mathbf{i} \mapsto suc, \mathbf{j} \mapsto +\}$ (here $f \mapsto *$).

$$\begin{aligned} 0 * y &= 0 \\ suc(x) * y &= y + (x * y) \end{aligned}$$

which in turn can be used for the invention of the concept of exponentiation with the substitution $\sigma_2 = \{\mathbf{g} \mapsto 0, \mathbf{h} \mapsto \lambda x.\, suc(0), \mathbf{i} \mapsto suc, \mathbf{j} \mapsto *\}$ on scheme 1 (note that the exponent is the first argument in this case).

$$\begin{aligned} exp(0, y) &= suc(0) \\ exp(suc(x), y) &= y * exp(x, y) \end{aligned}$$

Schemes can be used not only for the invention of new mathematical concepts or definitions, they also can be used for the invention of new conjectures about those concepts. The scheme 2 creates conjectures about the *left-distributivity* property of two binary operators in a given theory (the variables $\mathbf{p}$ and $\mathbf{q}$ stand for the binary operators). Therefore if we are working w.r.t. $\mathcal{T}_{\mathcal{N}}$ extended with multiplication and exponentiation, we can conjecture the left-distributivity property of multiplication and addition and also between exponentiation and multiplication by using the substitutions $\sigma_3 = \{\mathbf{p} \mapsto +, \mathbf{q} \mapsto *\}$ and $\sigma_4 = \{\mathbf{p} \mapsto *, \mathbf{q} \mapsto exp\}$ respectively on the scheme (2).

$$\left(\begin{array}{l} left\text{-}distributivity(\mathbf{p}, \mathbf{q}) \equiv \\ \quad \forall x\, y\, z.\ \mathbf{q}(x, \mathbf{p}(y, z)) = \mathbf{p}(\mathbf{q}(x, y), \mathbf{q}(x, z)) \end{array} \right) \tag{2}$$

The aforementioned substitutions give the conjectures

$$\begin{aligned} x * (y + z) &= (x * y) + (x * z) \\ exp(x, y * z) &= exp(x, y) * exp(x, z) \end{aligned}$$

It is important to note that schemes could generate invalid definitions and false conjectures. For example, consider the substitution $\sigma_4 = \{\mathbf{g} \mapsto 0, \mathbf{h} \mapsto (\lambda x.\, 0), \mathbf{i} \mapsto (\lambda x.\, x), \mathbf{j} \mapsto +\}$[2] on scheme 1

$$\begin{aligned} f(0, y) &= 0 \\ f(x, y) &= y + f(x, y) \end{aligned}$$

This instantiation immediately leads to logical inconsistencies by subtracting $f(x, y)$ from the second equation producing $0 = y$. This definition is invalid because, contrary to the natural interpretation of $\mathbf{i}$ as a constructor symbol, schemes do not express such conditions on instantiations. Similarly, we can also obtain false conjectures from a substitution, e.g. $\sigma_4 = \{\mathbf{p} \mapsto *, \mathbf{q} \mapsto +\}$ on scheme 2 instantiates to $x + (y * z) = (x + y) * (x + z)$.

[2] Note that all theories considered depend upon the simply typed lambda calculus of Isabelle/HOL. Therefore, $(\lambda x.\, x)$ is a perfectly valid mathematical object. In fact, we can choose to have any (finite) set of well-formed closed terms as the initial theory elements for the exploration of the theory (see section 4 for details).

3 Representation of Schemes

A *scheme* is a higher-order formula intended to generate new *definitions* of the underlying theory and *conjectures* about them. However, not every higher-order formula is a scheme. Here, we formally define schemes.

Definition 1. *A* **scheme** *s is a (non-recursive) constant definition of a proposition in HOL which we write in the form $s_n(\overline{x}) \equiv t$.*

For the scheme $s_n(\overline{x}) \equiv t$, $\overline{x}$ are free variables and t does not contain s_n, does not refer to undefined symbols and does not introduce extra free variables. The scheme (where *dvd* means "divides") $prime(p) \equiv 1 < p \wedge (dvd(m,p) \Rightarrow m = 1 \vee m = p)$ is flawed because it introduces the extra free variable m on the right hand side. The correct version is $prime(p) \equiv 1 < p \wedge (\forall m.\, dvd(m,p) \Rightarrow m = 1 \vee m = p)$ assuming that all symbols are properly defined.

Definition 2. *Given a scheme $s := s_n(\overline{x}) \equiv t$ we say that s is a* **propositional scheme***. In case t has the form $\exists \overline{f}\ \forall \overline{y}\ \wedge_{i=1}^{m} l_i = r_i$ then we say that the propositional scheme s is a* **definitional scheme***, and $l_1 = r_1, \ldots, l_m = r_m$ are the* **defining equations** *of s.*

Examples of valid propositional schemes are listed below.

$$\begin{aligned} true &\equiv \top \\ comm(p) &\equiv (\forall x\, y.\ p(x,y) = p(y,x)) \\ assoc\_comm(p) &\equiv \forall x\, y\, z.\ p(p(x,y),z) = p(x,p(y,z)) \wedge comm(p) \end{aligned}$$

The following are examples of definitional schemes.

$$\left(\begin{array}{l} def\text{-}scheme(\mathbf{g},\mathbf{h},\mathbf{i},\mathbf{j}) \equiv \\ \quad \exists f.\ \forall x\, y\, z.\ \bigwedge \left\{ \begin{array}{l} f(\mathbf{g},y) = y \\ f(\mathbf{h}(z,x),y) = \mathbf{i}(\mathbf{j}(z,y), f(x,y)) \end{array} \right. \end{array} \right) \tag{3}$$

$$\left(\begin{array}{l} mutual\text{-}def\text{-}scheme(\mathbf{g},\mathbf{h},\mathbf{i},\mathbf{j},\mathbf{k},\mathbf{l}) \equiv \\ \quad \exists f_1\ f_2.\ \forall x\, y.\ \bigwedge \left\{ \begin{array}{l} f_1(\mathbf{g}) = \mathbf{h} \\ f_2(\mathbf{g}) = \mathbf{i} \\ f_1(\mathbf{j}(z,x)) = \mathbf{k}(z, f_2(x)) \\ f_2(\mathbf{j}(z,x)) = \mathbf{l}(z, f_1(x)) \end{array} \right. \end{array} \right) \tag{4}$$

The definitional scheme (4) captures the idea of two mutual functions defined recursively. Here the existentially quantified variables (f in scheme (3) and f_1 and f_2 in scheme (4)) stand for the new functions to be defined.

4 Generation of Instantiations

In this section we describe the technique used to instantiate schemes automatically. Here we define some preliminary concepts.

Definition 3. *For a scheme s, the set of* **schematic substitutions** *with respect to a (finite) set of closed terms $X \subset \mathcal{T}(\mathcal{F}, \mathcal{V})$ is defined by:*

$$Sub(s, X) := \{\sigma \mid Closed(s\sigma) \wedge (((v \mapsto x) \in \sigma) \Rightarrow x \in X)\}$$

where $Closed(t)$ is true when the term t contains no free variables.

Ensuring that $s\sigma$ is a closed term avoids overgeneralisations on conjectures or definitions, e.g. it is impossible to prove $\forall x\, y\, z.\ x * \mathbf{p}(y, z) = \mathbf{p}(x * y, x * z)$ where $\mathbf{p}$ is free. Definition 3 also bounds the possible substitutions such that free variables in the scheme are mapped to closed terms in X. The problem of finding the substitutions $Sub(s, X)$ of a scheme s given a set of terms X can be solved as follows. The free variables $\mathcal{V}(s) = \{v_1, \ldots, v_n\}$ in the scheme are associated with their initial domain $D_{0i} = \{x \in X |\ v_i$ and x can be unified$\}$ for $1 \leq i \leq n$. The typing information of the partially instantiated scheme is the only constraint during the instantiation of variables. Each time a variable v_i is instantiated to $x \in D_{ki}$ the domains $D_{(k+1)j}$ for $i < j \leq n$ of the remaining variables must be updated w.r.t the most general unifier σ_{mgu} of v_i and x. Variables are instantiated sequentially and if a partial instantiation leaves no possible values for a variable then backtracking is performed to the most recently instantiated variable that still has alternatives available. This process is repeated using backtracking to exhaust all possible schematic substitutions obtaining a complete algorithm.

Example 1. Let $\mathcal{F}$ be a signature consisting of $\mathcal{F} := \{+_{:nat \to nat \to nat}, *_{:nat \to nat \to nat}, @_{:`a\,list \to `a\,list \to `a\,list}, map_{:(`a \to `b) \to `a\,list \to `b\,list}\}$. Also let $X = \{+, *, @, map\}$ and s be the propositional scheme (2) of section 2 (here we assume the most general type infered for the scheme).

Figure (1) illustrates how $Sub(s, X)$ is evaluated following a sequential instantiation of the free variables of s. It is important to note that the asymptotic running time of the algorithm is $\Theta(|X|^{|\mathcal{V}(s)|})$ and the worst case is when we obtain $|X|^{|\mathcal{V}(s)|}$ valid substitutions.

For a scheme s, the generated schematic substitutions are used to produce instantiations of s; i.e. conjectures or definitions

Definition 4. *Given $\sigma \in Sub(s, X)$, the* **instantiation of the scheme** *$s := u \equiv v$ with σ is denoted by*

$$inst(u \equiv v, \sigma) := v\sigma$$

Definition 5. *For a scheme s, the* **set of instantiations** *$Insts(s, X)$ with respect to a (finite) set of closed terms $X \subset \mathcal{T}(\mathcal{F}, \mathcal{V})$ is denoted by*

$$Insts(s, X) := \{inst(s, \sigma) \mid \sigma \in Sub(s, X)\} \tag{5}$$

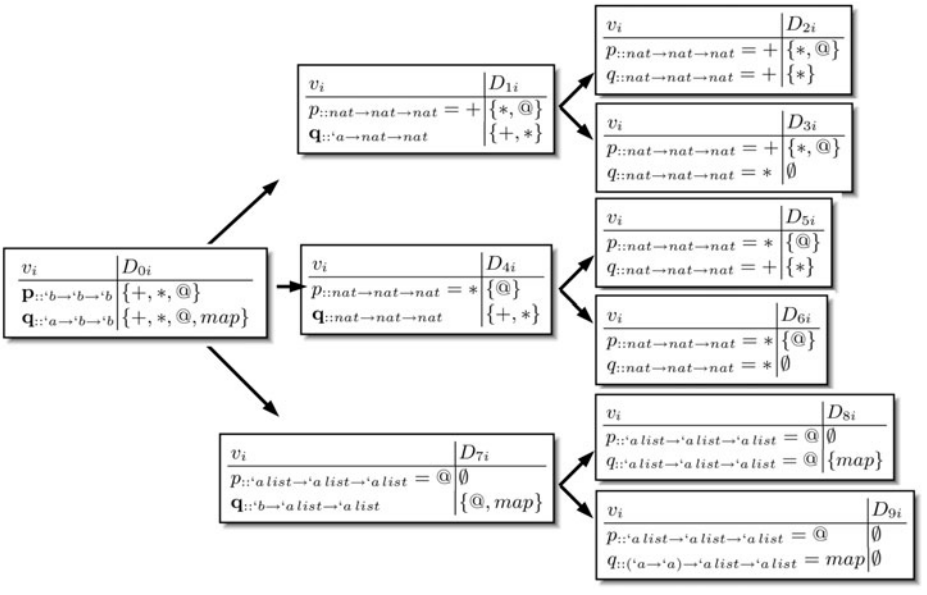

Fig. 1. Sequential evaluation of $Sub(s, X)$ where s is the propositional scheme (2) and $X = \{+_{::nat\to nat\to nat}, *_{::nat\to nat\to nat}, @_{::'a\,list\to'a\,list\to'a\,list}, map_{::('a\to'b)\to'a\,list\to'b\,list}\}$. Each box shows the unified and not-unified (in bold) variables and their domain during the evaluation. The output of the algorithm is the set of substitutions $\{\sigma_1 = \{p \mapsto +, q \mapsto +\}, \sigma_2 = \{p \mapsto +, q \mapsto *\}, \sigma_3 = \{p \mapsto *, q \mapsto +\}, \sigma_4 = \{p \mapsto *, q \mapsto *\}, \sigma_5 = \{p \mapsto @, q \mapsto @\}, \sigma_6 = \{p \mapsto @, q \mapsto map\}\}$. Note that a unified variable potentially changes the types of the rest of the variables, restricting their domain.

Example 2. The instantiations generated from scheme (2) and the set of terms $X = \{+, *, @, map\}$ are depicted in the following table.

$Sub(s, X)$	$Insts(s, X)$
$\sigma_1 = \{p \mapsto +, q \mapsto +\}$	$\forall x\, y\, z.\ x + (y + z) = (x + y) + (x + z)$
$\sigma_2 = \{p \mapsto +, q \mapsto *\}$	$\forall x\, y\, z.\ x * (y + z) = x * y + x * z$
$\sigma_3 = \{p \mapsto *, q \mapsto +\}$	$\forall x\, y\, z.\ x + y * z = (x + y) * (x + z)$
$\sigma_4 = \{p \mapsto *, q \mapsto *\}$	$\forall x\, y\, z.\ x * (y * z) = (x * y) * (x * z)$
$\sigma_5 = \{p \mapsto @, q \mapsto @\}$	$\forall x\, y\, z.\ x@(y@z) = (x@y)@(x@z)$
$\sigma_6 = \{p \mapsto @, q \mapsto map\}$	$\forall x\, y\, z.\ map(x, y@z) = map(x, y)@map(x, z)$

5 Identification of Equivalent Instantiations

Processing the instantiations (conjectures and definitions) of a scheme could be a demanding task. In the worst case, the number of substitutions $\sigma : V \to X$ is $|X|^{|V|}$. However, we can reduce the number of conjectures and definitions

by noticing that two different substitutions σ_1 and σ_2 could lead to equivalent instantiations.

Table 1 shows the set of instantiations $Insts(s, X)$ obtained from the following definitional scheme.

$$\left(\begin{array}{l} def\text{-}scheme(\mathbf{g}, \mathbf{h}, \mathbf{i}) \equiv \\ \quad \exists f.\ \forall x\, y.\ \bigwedge \left\{ \begin{array}{l} f(\mathbf{g}, y) = \mathbf{h}(\mathbf{g}, \mathbf{g}) \\ f(suc(x), y) = \mathbf{i}(y, f(x, y)) \end{array} \right. \end{array} \right) \tag{6}$$

In Table 1 $inst(s, \sigma_{N1})$ and $inst(s, \sigma_{N2})$ are clearly equivalent[3]. The key ingredient for automatically detecting equivalent instantiations is a term rewrite system (TRS) R which handles the normalization of the function symbols inside a term [10].

Table 1. Redundant definitions generated from the definitional scheme 6. Note that the instantiations $inst(s, \sigma_{N1})$ and $inst(s, \sigma_{N2})$ are equivalent as $0+0$ can be 'reduced' (within the theory) to 0. $inst(s, \sigma_{N3})$ and $inst(s, \sigma_{N4})$ are similarly equivalent.

$Sub(s, X)$	$Insts(s, X)$
$\sigma_{N1} = \left\{ \begin{array}{c} g \mapsto 0, h \mapsto + \\ i \mapsto + \end{array} \right\}$	$\exists f.\ \forall x\, y. \bigwedge \left\{ \begin{array}{l} f(0, y) = 0 + 0 \\ f(suc(x), y) = y + f(x, y) \end{array} \right.$
$\sigma_{N2} = \left\{ \begin{array}{c} g \mapsto 0, h \mapsto (\lambda x\, y.\, x) \\ i \mapsto + \end{array} \right\}$	$\exists f.\ \forall x\, y. \bigwedge \left\{ \begin{array}{l} f(0, y) = 0 \\ f(suc(x), y) = y + f(x, y) \end{array} \right.$
$\sigma_{N3} = \left\{ \begin{array}{c} g \mapsto 0, h \mapsto + \\ i \mapsto (\lambda x\, y.\, x) \end{array} \right\}$	$\exists f.\ \forall x\, y. \bigwedge \left\{ \begin{array}{l} f(0, y) = 0 + 0 \\ f(suc(x), y) = y \end{array} \right.$
$\sigma_{N4} = \left\{ \begin{array}{c} g \mapsto 0, h \mapsto (\lambda x\, y.\, x) \\ i \mapsto (\lambda x\, y.\, x) \end{array} \right\}$	$\exists f.\ \forall x\, y. \bigwedge \left\{ \begin{array}{l} f(0, y) = 0 \\ f(suc(x), y) = y \end{array} \right.$

However, for this idea to work, the constructed TRS R must have the property of being *terminating*. All functions in Isabelle/HOL are terminating to prevent inconsistencies. Therefore, the defining equations for a newly introduced function symbol can be used as a normalizing TRS. Furthermore, if we are to include a new equation e to the rewrite system $\mathcal{R}$ during the exploration of the theory then we must prove termination of the extended rewrite system $\mathcal{R} \cup \{e\}$. To this end, we use the termination checker AProVE [9] along with Knuth-Bendix completion to obtain a convergent rewrite system, if possible (using a similar approach to [17]). The following definition will help with the description of the algorithm for theory exploration of section 7.

Definition 6. *Given a terminating rewrite system $\mathcal{R}$ and an instantiation $i \in Insts(s, X)$ of the form $\forall \overline{x}.\, s = t$, the* **normalizing extension** *$ext(\mathcal{R}, i)$ of $\mathcal{R}$ with i is denoted by*

[3] Note that '+' denotes standard addition of naturals.

$$ext(\mathcal{R}, i) := \begin{cases} \mathcal{R}' & \text{if Knuth-Bendix completion succeeds for} \\ & \mathcal{R} \cup \{s = t\} \text{ with a convergent system } \mathcal{R}' \\ \mathcal{R} \cup \{r\} & \text{if termination succeeds for } \mathcal{R} \cup \{r\} \\ & \text{with } r \in \{s = t,\ t = s\} \\ \mathcal{R} & \text{otherwise} \end{cases}$$

6 Filtering of Conjectures and Definitions

As suggested by definition 6, IsaScheme updates the rewrite system $\mathcal{R}$ each time a new equational theorem is found. It is thus useful to consider the notion of equivalence of instantiations modulo $\mathcal{R}$.

Definition 7. *Let u and v be two instantiations and $\mathcal{R}$ a terminating rewrite system.* **Equivalence of instantiations** *modulo $\mathcal{R}$ is denoted as*

$$u \approx_R v := (\hat{u} =_\alpha \hat{v})$$

where $\hat{u}$ and $\hat{v}$ are normal forms (w.r.t. $\mathcal{R}$) of u and v respectively and $=_\alpha$ is term equivalence up to variable renaming.

Since the exploration process could generate a substantial number of definitions and each of them could potentially produce a multitude of conjectures it becomes necessary to restrict the search space in some way. We decided to filter out all functions whose values are independent of one of their arguments as they can always be defined with another function using fewer arguments and a λ-abstraction. For example, instead of generating $f_1(x, y) = x^2$ and $f_2(x, y) = y^2$ it would be better to just generate $f(x) = x^2$ and construct f_1 and f_2 on top of f, e.g. $(\lambda x\, y.\, f(x))$ and $(\lambda x\, y.\, f(y))$.

Definition 8. *An* **argument neglecting function** *$f \in \mathcal{F}$ with type $\tau_1 \to \ldots \to \tau_n \to \tau_0$, where τ_0 is a base type and $n > 0$, is a function such that*

$$f(x_1, \ldots, x_{k-1}, y, x_{k+1}, \ldots, x_n) = f(x_1, \ldots, x_{k-1}, z, x_{k+1}, \ldots, x_n)$$

for some k where $1 \leq k \leq n$.

7 Theory Exploration Algorithms

Scheme-based Conjecture Synthesis. The overall procedure for the generation of theorems is described by the pseudocode of `InventTheorems`. The algorithm receives as arguments a set of terms I (conjectures), a terminating rewrite system $\mathcal{R}$, a set of terms T (theorems), a set of propositional schemes S_p and a set of closed terms X_p from which schemes are to be instantiated. Note that initially, $I = T = \emptyset$.

```
   InventTheorems(I, R, T, S_p, X_p)
1  for each i ∈ ⋃_{s ∈ S_p} Insts(s, X_p)
2    î := a normal form of i w.r.t. R
3    if î is not subsumed by T ∪ {True} and
4    there is not a j ∈I such that j ≈_R î and
5    cannot find a counter-example of î then
6      if can prove î then
7        T := T ∪ {î}
8        if î is of the form ∀x̄. s = t then
9          R := ext(R, î)
10   I := I ∪ {î}
11 return <I, N, T>
```

The algorithm iterates through all instantiations obtained from any scheme $s \in S_p$ and the terms X_p. Line 4 can be implemented efficiently using discrimination nets and avoids counterexample checking equivalent instantiations modulo $\mathcal{R}$. Falsifiable instantiations are detected in line 5 to avoid any proof attempt on such conjectures. Isabelle/HOL provides the counter-example checker *quickcheck* [1] which is used to refute the false conjectures in the implementation of *IsaScheme*. In case the conjecture is not rejected by the inspection in lines 4, 5 or 6 then a proof attempt is performed in line 6. The prover used for the proof obligations in *IsaScheme* was the automatic inductive prover *IsaPlanner* [8] which implements the *rippling* heuristic [4].

Scheme-based Definition Synthesis. The generation of definitions is described by the pseudocode of `InventDefinitions`. The algorithm takes as input the same arguments received by the `InventTheorems` method. Additionally, it also takes a set of function symbols $\mathcal{F}$ in the current theory, a set of terms D (definitions), a set of definitional schemes S_d and a set of closed terms X_d from which definitional schemes are to be instantiated. Again, initially $D = \emptyset$.

The algorithm iterates through all instantiations obtained from any definitional scheme $s \in S_d$ and the terms X_d. Each instantiation d is reduced to a normal form $\hat{d}$ w.r.t. $\mathcal{R}$ in line 2. Since $\hat{d}$ is generated from a definitional scheme, it has the form $\exists f_1 \ldots f_n . \forall \overline{y} . e_1 \wedge \ldots e_m$ where $f_1, \ldots, f_n$ are variables standing for the new functions to be defined and $e_1, \ldots, e_m$ are the defining equations of the functions. In lines 4 and 5, new function symbols $f'_1, \ldots, f'_n$ (w.r.t. the signature $\mathcal{F}$) are created and a substitution σ is constructed to give a specific name to each of the new functions to be defined. This 'renaming' of functions is performed with the defining equations and $[e'_1, \ldots, e'_m]$ is obtained in line 6. Line 7 ensures that definitions that are equivalent modulo $\mathcal{R}$ to earlier generated ones, are ignored. Well-definedness properties such as termination or totality of the functions generated are proved in line 8. We used Isabelle/HOL's *function package* [11] for these proof obligations. Line 9 checks if the new functions created are not argument neglecting (AN). In practice, it is hard and expensive to prove conjectures of the form $f(x_1, \ldots, x_{k-1}, y, x_{k+1}, \ldots, x_n) = f(x_1, \ldots, x_{k-1}, z, x_{k+1}, \ldots, x_n)$ demanded by definition 8; instead, we produce counter-examples of that function

not being AN for each of its arguments. In case the instantiation $\hat{d}$ is not rejected by the inspection in lines 7, 8 or 9 then the context $\mathcal{F}$ and the theorems T are updated with the new function symbols $f'_1, \ldots, f'_n$ and the theorems $\{e'_1, \ldots, e'_m\}$ respectively (lines 10 and 11). Line 12 updates the rewrite system $\mathcal{R}$ with the newly introduced defining equations $e'_1, \ldots, e'_m$. A call to `InventTheorems` is performed in line 13 updating $I, \mathcal{N}$ and T. At the end of each iteration the instantiation $\hat{d}$ is added to the set of processed definitions D in line 14. Finally, when all instantiations $d \in \bigcup_{s \in S_d} Insts(s, X_d)$ have been processed the values $<I, \mathcal{N}, T, \mathcal{F}, D>$ are returned.

```
   InventDefinitions(I, R, T, S_p, X_p, F, D, S_d, X_d)
1  for each d ∈ ⋃_{s ∈ S_d} Insts(s, X_d)
2    d̂ := a normal form of d w.r.t. R
3    let ∃f_1 ... f_n.∀y. e_1∧...e_m = d̂
4    create function symbols f'_1, ..., f'_n such that f'_i ∉ F
5    σ := {f_1 ↦ f'_1, ..., f_n ↦ f'_n}
6    [e'_1, ..., e'_m] := [σ(e_1), ..., σ(e_m)]
7    if there is not a j ∈D such that j ≈_R d̂ and
8    [e'_1, ..., e'_m] is well-defined and
9    f'_1, ..., f'_n are not argument neglecting then
10       F := F ∪ {f'_1, ..., f'_n}
11       T := T ∪ {e'_1, ..., e'_m}
12       R := R ∪ {e'_1, ..., e'_m}
13       <I, N, T>:=InventTheorems(I, R, T, S_p, X_p ∪ {f'_1, ..., f'_n})
14   D := D ∪ {d̂}
15 return <I, R, T, F, D>
```

8 Evaluation

We conducted several case studies in a theory of natural numbers and a theory of lists to evaluate how similar were the results obtained with our method and implementation to those in the libraries of the Isabelle proof assistant. We performed a precision/recall analysis with Isabelle's libraries as reference to evaluate the quality of the theorems found by the `InventTheorems` algorithm and the following propositional scheme[4].

$$\left(\begin{array}{l} \mathit{prop\text{-}scheme}(\mathbf{p}, \mathbf{q}, \mathbf{r}, \mathbf{s}, \mathbf{t}, \mathbf{u}) \equiv \\ \qquad \forall x\, y\, z.\ \mathbf{p}(\mathbf{q}(x, y), \mathbf{r}(x, z)) = \mathbf{s}(\mathbf{t}(x, z), \mathbf{u}(y, z)) \end{array}\right)$$

IsaScheme produces a total of 23 theorems for the theory of naturals with 14 of them included in Isabelle's libraries. Isabelle contains 33 theorems about addition, multiplication and exponentiation giving a precision of 60% and a recall of 42%. The theorems discovered in the theory of natural numbers included,

[4] We also used another propositional scheme to handle ternary operators.

commutativity, associativity, distributivity of multiplication and addition, distributivity of exponentiation and multiplication, commuted versions of addition and multiplication, among others. It is important to say that 16 out of the 19 theorems not synthesised were subsumed (after normalization w.r.t. the resulting rewrite system $\mathcal{R}$) by more general synthesised ones and the rest fell out of the scope of the propositional scheme used as they contained 4 variables.

IsaScheme produces a total of 13 theorems for the theory of lists producing all 9 theorems about append, list reverse, map, right-fold and left-fold included in Isabelle's libraries. This gives a precision of 70% and a recall of 100% for this theory. In the theory of lists, there was a rather high number of unfalsified and unproved conjectures (279). The type information for these conjectures was more complex than Quickcheck could manage. A small random sample of these conjectures was taken, and in each case, a counterexample was quite easily found by hand. Table 2 summarises the statistics for the theories analysed.

Table 2. Precision and recall analysis with Isabelle's theory library as reference. The constructors are `zero`, `Suc`, `nil` and `cons` with labels Z, S, N and C respectively. The functions are `addition`, `multiplication`, `exponentiation`, `append`, `reverse`, `length`, `map`, `left-fold` and `right-fold` with labels +, *, ^, A, R, L, M, FL and FR respectively.

Precision-Recall	65%-50%	63%-100%	100%-100%	80%-100%
Constructors	Z, S	Z, S, N, C	N, C	N, C
Functions Symbols	+, *, ^	A, R, L	A, R, M	A, FL, FR
Elapsed Time (s)	1756	9237	9885	18179
Conjectures Synthesised	78957	175847	204950	13576
Conjectures Filtered	78934	175839	204944	13292
Proved-Not Proved	23-0	8-0	6-0	7-279

The definitions synthesised by IsaScheme (`InventDefinitions` algorithm) with the following definitional scheme are, among others, addition, multiplication, exponentiation, append, map, and (tail recursive) reverse.

$$\left(\begin{array}{l} def\text{-}scheme(\mathbf{g},\mathbf{h},\mathbf{i},\mathbf{j},\mathbf{k},\mathbf{l}) \equiv \\ \quad \exists f.\ \forall x\, y\, z.\ \bigwedge \left\{ \begin{array}{l} f(\mathbf{g},y) = \mathbf{h}(y) \\ f(\mathbf{i}(z,x),y) = \mathbf{j}(\mathbf{k}(x,y,z), f(x,\mathbf{l}(z,y))) \end{array} \right. \end{array} \right)$$

For the theory of natural numbers, IsaScheme obtained a precision of 6% and a recall of 100%. For the theory of lists, IsaScheme obtained a precision of 14% and a recall of 18%. The low recall for the list theory was caused because the definitional scheme used was only able to synthesise binary functions and not unary or ternary ones. This could have been addressed easily by considering definitional schemes producing unary and ternary functions at the expense of computational time (see section 4). This however, would have been detrimental for the precision of the `InventDefinitions` algorithm. Note that the scheme-based approach for the generation of definitions provides a free-form incremental construction of (potentially infinitely many) recursive functions. Overly general definitional

schemes provide a wide range of possible instantiations and thus, definitions. In fact, we believe this was the reason of the low precision in the evaluation of the algorithm for both theories. Strategies to assess the relevance of definitions are required given the big search space during exploration. Nevertheless, this is left as future work. For space reasons we can not give a presentation of the theories or the theorems and definitions found. Formal theory documents in human-readable Isabelle/Isar notation and all results described in this paper are available online[5]. For the evaluation we use a computer cluster where each theory exploration was run in a GNU/Linux node with 2 dual core CPUs and 4GB of RAM memory. We also use Isabelle/2009-2, IsaPlanner svn version 2614 and AProVE 1.2.

9 Related Work

Other than IsaScheme, Theorema is the only system performing the exploration of mathematical theories based on schemes [3]. However, the user needs to perform all schematic substitutions manually as Theorema does not instantiate the schemes automatically from a set of terms. The user also needs to conduct the proof obligations interactively [6]. Another important difference is that ensuring the soundness of definitions is left to the user in Theorema. In IsaScheme, which uses Isabelle's LCF-methodology, definitions are sound by construction [11].

The MATHsAiD program was intended for use of research mathematicians and was designed to produce interesting theorems from the mathematician's point of view [13]. MATHsAiD starts with an axiomatic description of a theory; hypotheses and terms of interest are then generated, forward reasoning is then applied to produce logical consequences of the hypotheses and then a filtering process is carried out according to a number of interestingness measures. MATHsAiD has been applied to the naturals, set theory and group theory.

Like IsaScheme, IsaCoSy is a theory exploration system for Isabelle/ IsaPlanner [10]. It generates conjectures in a bottom-up fashion from the signature of an inductive theory. The synthesis process is accompanied by automatic counter-example checking and a prove attempt in case no counter-example is found. All theorems found are then used as constraints for the synthesis process generating only irreducible terms w.r.t. the discovered theorems. The main difference between IsaScheme and IsaCoSy is that IsaCoSy considers all (irreducible) terms as candidate conjectures where IsaScheme considers only a restricted set (modulo $\mathcal{R}$) specified by the schemes. This restricted set of conjectures avoids the need for a sophisticated constraint language. The main advance made by IsaScheme is the use of Knuth-Bendix completion and termination checking to orient the resulting equational theorems to form a rewrite system. The empirical results show that for the theory of lists, these rewrite systems result in fewer theorems that prove all of the theorems in the theory produced by IsaCoSy.

HR is a theory exploration system which uses an example driven approach for theory formation[5]. It uses MACE to build models from examples and also to

[5] `http://dream.inf.ed.ac.uk/projects/isascheme/`

identify counter-examples. The resolution prover Otter is used for the proof obligations. The process of concept invention is carried out from old concepts starting with the concepts provided by MACE at the initial stage. These concepts, stored as data-tables of examples rather than definitions, are passed through a set of production rules whose purpose is to manipulate and generate new data-tables, thus generating new concepts. The conjecture synthesis process is built on top of concept formation. HR takes the concepts obtained by the production rules and forms conjectures about them. There are different types of conjectures HR can make, e.g. *equivalence* conjectures which amounts to finding two concepts and stating that their definitions are equivalent, *implication* conjectures are statements relating two concepts by stating that the first is a specialization of the second (all examples of the first will be examples of the second), etc. HR has been applied to the naturals, group theory and graph theory.

10 Limitations and Future Work

An important aspect of every theory exploration system is its applicability across different mathematical theories. The scheme-based approach used by IsaScheme, provides a generic mechanism for the exploration of any mathematical theory where the symbols (or closed terms built from them) in the theory's signature and the variables within the schemes could be unified. However, this free-form theory exploration could lead to a substantial number of instantiations that needs to be processed (see section 4) and it is particularly true with large numbers of constructors and function symbols. This is partially mediated with the lemmata discovered during the exploration of the theory. Nevertheless, this could be improved by also exploiting the intermediate lemmata needed to finish the proofs, e.g. with the lemma calculation critic used in rippling.

Another limitation is that termination (and thus confluence) of rewrite systems is in general undecidable and requires sophisticated technology to solve interesting cases. This problem is aggravated with rewrite systems with a large number of rewrite rules. In this situation, termination checking demanded by definition 6 would benefit from modular properties of rewrite systems such as *hierarchical termination*[7].

11 Conclusion

We have implemented the proposed scheme-based approach to mathematical theory exploration in Isabelle/HOL for the generation of conjectures and definitions. This interpretation is used to describe how the instantiation process of schemes can be automated. We have also described how we can make productive use of normalization in two ways: first to improve proof automation by maintaining a terminating and potentially convergent rewrite system and second to avoid numerous redundancies inherent in most theory exploration systems.

Acknowledgments. This work has been supported by Universidad Politécnica de San Luis Potosí, SEP-PROMEP, University of Edinburgh, the Edinburgh Compute and Data Facility, the RISC-Linz Transnational Access Programme (No. 026133), and EPSRC grants EP/F033559/1 and EP/E005713/1. The authors would like to thank the anonymous referees for their helpful comments.

References

1. Berghofer, S., Nipkow, T.: Random Testing in Isabelle/HOL. In: SEFM, pp. 230–239 (2004)
2. Buchberger, B.: Algorithm Supported Mathematical Theory Exploration: A Personal View and Stragegy. In: Buchberger, B., Campbell, J. (eds.) AISC 2004. LNCS (LNAI), vol. 3249, pp. 236–250. Springer, Heidelberg (2004)
3. Buchberger, B., Craciun, A., Jebelean, T., Kovács, L., Kutsia, T., Nakagawa, K., Piroi, F., Popov, N., Robu, J., Rosenkranz, M.: Theorema: Towards computer-aided mathematical theory exploration. J. Applied Logic 4(4), 470–504 (2006)
4. Bundy, A., Basin, D., Hutter, D., Ireland, A.: Rippling: Meta-level Guidance for Mathematical Reasoning. Cambridge Tracts in Theoretical Computer Science, vol. 56. Cambridge University Press, Cambridge (2005)
5. Colton, S.: Automated Theory Formation in Pure Mathematics. PhD thesis, Division of Informatics, University of Edinburgh (2001)
6. Craciun, A., Hodorog, M.: Decompositions of Natural Numbers: From A Case Study in Mathematical Theory Exploration. In: SYNASC 2007 (2007)
7. Dershowitz, N.: Hierachical Termination. In: Lindenstrauss, N., Dershowitz, N. (eds.) CTRS 1994. LNCS, vol. 968, pp. 89–105. Springer, Heidelberg (1995)
8. Dixon, L., Fleuriot, J.D.: IsaPlanner: A Prototype Proof Planner in Isabelle. In: Baader, F. (ed.) CADE 2003. LNCS (LNAI), vol. 2741, pp. 279–283. Springer, Heidelberg (2003)
9. Giesl, J., Schneider-kamp, P., Thiemann, R.: AProVE 1.2: Automatic Termination Proofs in the Dependency Pair Framework. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 281–286. Springer, Heidelberg (2006)
10. Johansson, M., Dixon, L., Bundy, A.: Conjecture Synthesis for Inductive Theories. Journal of Automated Reasoning (2010) (to appear)
11. Krauss, A.: Automating Recursive Definitions and Termination Proofs in Higher-Order Logic. PhD thesis, Dept. of Informatics, T. U. München (2009)
12. Lenat, D.B.: AM: An Artificial Intelligence approach to discovery in Mathematics as Heuristic Search. In: Knowledge-Based Systems in Artificial Intelligence (1982)
13. McCasland, R., Bundy, A., Smith, P.F.: Ascertaining Mathematical Theorems. Electr. Notes Theor. Comput. Sci. 151(1), 21–38 (2006)
14. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle's Logics: HOL (2000)
15. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002)
16. Sutcliffe, G., Gao, Y., Colton, S.: A Grand Challenge of Theorem Discovery (June 2003)
17. Wehrman, I., Stump, A., Westbrook, E.: Slothrop: KnuthBendix Completion with a Modern Termination Checker. In: Webster University, St. Louis, Missouri M.Sc. Computer Science, Washington University, pp. 268–279 (2006)