First Steps in Integrating {*log*} into Z/EVES

Maximiliano Cristiá[1], Gianfranco Rossi[2], and Claudia Frydman[3]

[1] CIFASIS and UNR, Rosario, Argentina
cristia@cifasis-conicet.gov.ar
[2] Università degli studi di Parma, Parma, Italy
gianfranco.rossi@unipr.it
[3] Aix Marseille Univ., CNRS, ENSAM, Univ. de Toulon, LSIS UMR 7296, France
claudia.frydman@lsis.org

Abstract. {*log*} is a Constraint Logic Programming language implementing a general theory of sets. As such it can determine the satisfiability of a large family of set formulas. Z/EVES is a proof assistant specifically tailored for the Z notation. In turn, the Z notation is based on a theory of sets. Hence, Z/EVES provides many facilities to prove theorems about set theory. We have observed that there are some such theorems that Z/EVES fails to prove automatically, while {*log*} is able to do it. Therefore, in this paper we propose to implement a new Z/EVES proof command by calling {*log*} after translating the goal to its input language. In this way it will be possible to prove many theorems automatically from Z/EVES.

1 Introduction

Interactive theorem proving is a difficult task. In general, it is more difficult than proving theorems by hand but, unless there are bugs in the provers, the proofs should not contain any mistake. Sometimes, this kind of proofs is needed in the software industry because developers must be certain of the absence of errors in their product,s as much as possible. Then, the more proofs automatically discharged, the better.

Several formal specification notations use sets as a key mathematical structure [27, 1, 18, 17, 20]. Many real-world entities are best represented as sets because repetitions of their elements and whether they are sorted in some way or not is irrelevant. Since specifications must describe only those properties that are interesting at some level of abstraction, sets must be used when *the entity to be modeled is essentially a set* (at that level of abstraction).

Sets can be manipulated, e.g. to prove theorems about sets, through general-purpose tools, such as the Coq proof assistant [6], the Z3 SMT solver [22], or Isabelle/HOL [23]. Alternatively, one can exploit tools specifically tailored to formal notations based on set theory, such as Z/EVES [26] for Z and Atelier B [4] for B. Tools for dealing with sets have been devised also by the constraint programming (CP) community, basically in the form of *Set Constraint Solvers* (SCS). SCS, however, usually put emphasis in efficiency rather than in completeness.

Therefore, many SCS are efficient in finding solutions for a relatively small class of formulas. For example, SCS a'la Gervet [15] support a quite restricted class of sets: those where all of their elements are integer constants. There are, however, SCS that support a very general theory of sets. Two good examples in the area of Constraint Logic Programming (CLP) are $\{log\}$ (pronounced 'setlog') [11, 24] and ProB [21]. In particular, $\{log\}$ presents some features that bring it closer to an automatic theorem prover or a SMT solver (for a set theory). In effect, it can determine the satisfiability of a large class of set formulas making it possible to prove that some formula is a theorem (when its negation is unsatisfiable).

The Test Template Framework (TTF) [28, 7] is a model-based testing method for the Z notation [27]. When the TTF is applied to a Z specification it generates a good number of unsatisfiable test specifications. Each test specification is defined by a Z set predicate. Then, to determine that a test specification is unsatisfiable means to decide whether a set predicate is unsatisfiable or not. In a previous paper we proposed a simple method to find unsatisfiable test specifications generated by the TTF [8]. This method requires from the user to introduce so-called elimination theorems. Each elimination theorem is a parametrized unsatisfiable set predicate. In order to be sure that each elimination theorem is indeed unsatisfiable we used the Z/EVES system to prove that. This meant to prove 85 (rather simple) theorems about set theory and integers. Thirty three of these theorems needed non trivial proofs (in the sense that Z/EVES most powerful proof command was not enough). Maybe surprisingly, we observed that 28 of these 33 theorems are automatically proved by $\{log\}$.

This fact suggests that $\{log\}$ can be a good complement to Z/EVES because it can augment the class of theorems that can be automatically proved. In this paper we show some of these theorems. We also propose a new Z/EVES proof command based on translating the goal to $\{log\}$, invoke it and wait for its answer.

The paper is structured as follows. In sections 2 and 3 we briefly introduce Z/EVES and $\{log\}$, respectively. Then, in Sect. 4 we present a motivating example consisting of a typical unsatisfiable test specification generated by the TTF, the theorem that should be proved by Z/EVES to eliminate it, and how the corresponding $\{log\}$ goal is found unsatisfiable. In Sect. 5 we show examples of the theorems that are automatically proved by each tool and those that are not automatically proved by at least one of them. How we plan to implement a new Z/EVES proof command by calling $\{log\}$ is described in Sect. 6. Finally, in Sect. 7 we present our conclusions.

2 Brief Introduction to Z/EVES

The Z notation is a widely know formal specification language based on first-order logic with equality and the Zermelo-Fraenkel set theory (ZFC) [27]. One of the key components of Z is called the Z mathematical toolkit (ZMT) [27, chapter 4]. The ZMT is a collection of definitions and laws verified by the various mathematical operators used in the notation. It includes sets, binary relations, partial functions, sequences, integer numbers and bags.

The Z/EVES system [26, 13] is a proof assistant for the Z notation. It was developed on top of the EVES system [19] which, as Z itself, is based on first-order predicate logic and ZFC. Laws in the ZMT were turned into theorems in Z/EVES [25]. Z/EVES provides a "linear" proof style—that is, each proof command transforms the current goal into an equivalent one. A proof finishes when the goal becomes *true*. Proof commands range from simple transformations of some predicate to heuristically-defined proof strategies. The idea behind this range of proof commands is to allow users to have full control of the proof while giving them the chance to automate large parts of it. The three main automatic proof strategies are described below.

Simplification. The tool implements a decision procedure for quantifier-free linear arithmetic and for equality. The simplifier also implements the one-point rule and removes quantified variables of formulas in prenex normal form. The simplifier can be extended by adding forward chaining rules and assumption rules to the ZMT.

Rewriting. If a theorem in the ZMT is labeled as an enabled rewrite rule, it is used automatically during rewriting. A rewrite rule must have a conclusion that is either an equality or an equivalence. There are around 500 rewriting rules in the ZMT and users can add their own. Many times these rules allow users to prove rather complex theorems in a few steps, but sometimes proofs end in unexpected goals that may be hard to prove. In these cases, rewriting rules must be manually applied by the user. For these reason, rewriting rules can be enabled or disabled for automatic application. Z/EVES designers have gone into great troubles to find an adequate balance between proof automation (i.e. more enabled rules) and proofs going awry (i.e. too many enabled rules).

Reduction. This strategy applies simplification and rewriting after schema expansion. This last feature is Z-specific given that schemas are the basic modularization construction of the Z notation. In Z, schemas can be combined in complex schema expressions with the aim to compose simpler ones. Hence, when a proof involving such an expression is attempted, reduction replaces the names of the schemas for their contents, thus generating a large predicate.

Regarding user-directed proof commands, the tool offers equality substitution, invocation of theorems, application of rewriting rules to specific parts of the goal, proof by cases, and so forth.

Example 1. Basically, there are two classes of theorems to be proved with Z/EVES: a) theorems about mathematics, particularly about set theory; and b) theorems about properties of Z specifications (for instance, state invariants). Since this paper deals with the first class, we give as an example such a theorem. Thus, the following is a typical Z/EVES theorem about set theory:

theorem PfunDomEmpty $[X, Y]$
$\forall R : X \pfun Y \bullet \operatorname{dom} R = \{\} \Rightarrow R = \{\}$

In this case none of the three automatic strategies works because in Z/EVES set equality can only be proved extensionally, which is a disabled rewriting rule. Furthermore, the definition of 'dom' is also needed during the proof, but this is another disabled rule. Therefore, the first step is to apply the definition of 'dom':

apply domDefinition;

domDefinition has a precondition asking the operand of 'dom' to be a binary relation. Then, a *simplify* command checks the precondition (in this case it is an hypothesis) and finishes the application of the rule. Now we have two set equalities ($\mathrm{dom}\, R = \{\}$ in the hypothesis and $R = \{\}$ in the conclusion) so we apply extensionality:

apply extensionality;

As expected the rule introduces two universal quantifications that can be eliminated by the rewrite command (which performs repeated rewriting and simplification on the current goal):

prove by rewrite;

These first four commands can be packed up in a single one:

with enabled (*domDefinition*, *extensionality*) *prove by rewrite*;

In fact, in either way, this command is able to discharge all the subgoals but the following:

$$\begin{array}{l} \quad R \in X \pfun Y \\ \quad \land \neg\, (\exists\, a : \{b : X;\ y : Y \mid (b, y) \in R \bullet b\} \bullet \mathit{true}) \\ \Rightarrow \neg\, x \in R \end{array}$$

In order to prove this goal we need to expand the expression $X \pfun Y$, a couple of instantiations and rewriting rules. The full proof script is the following one:

proof[*PfunDomEmpty*]
with enabled (*domDefinition*, *extensionality*) *prove by rewrite*;
invoke $X \pfun Y$;
prove by rewrite;
invoke $X \rel Y$;
apply inPower;
instantiate $e == x$;
split $x \in R$;
simplify;
instantiate $x\_\_1 == x.1$;
prove by rewrite;
instantiate $y == x.2$;
*apply inCross*2;
prove by rewrite;
■

As the example shows, a relatively simple property of partial functions needs a non trivial proof. We will come back to this example in Sect. 4.

From now on, we will say that Z/EVES *automatically* proves a theorem if it does that with the *proof by rewrite* command— because, in general, it is the most powerful command for theorems about set theory.

Sets are at the core of Z/EVES. For instance, according to the Z/EVES mathematical toolkit [25], powersets, cross products, singletons, set membership, set union, set comprehensions, and partial and total functions spaces are all predefined in Z/EVES—that is, they are not defined on top of other mathematical structures. In particular, functions, as in Z, are sets of ordered pairs—i.e. they are not first-class entities as in many tools such as Coq and Isabelle/HOL.

3 Brief Introduction to {*log*}

{*log*} [11, 24] is a Constraint Logic Programming (CLP) language implementing *hereditarily finite sets*—i.e., finitely nested sets that are finite at each level of nesting. As a CLP language and tool it can solve the satisfiability of a large class of set formulas. If it finds that a particular formula is satisfiable it can compute, one after the other, all of its solutions; if it finds the formula is unsatisfiable it simply returns 'no'. {*log*} is based on the theory of set unification [12]. Given that variables can occur within a set, representing either elements or sets, solving set equations means solving a set unification problem. Informally, the set unification problem is the problem of computing a binding of values to the variables occurring in two sets which makes them to denote the same set.

{*log*} supports all the classic set operators and allows users to define new data structures and operators. For example, partial functions can be represented as sets of ordered pairs. However, recently we have extended the core language to partial functions as we noted that a) they can be easily added, and b) the solver becomes more effective and efficient for goals including them[4].

Example 2. The theorem presented in Example 1 can be encoded in {*log*} as follows (implication has to be encoded as disjunction):

$$\mathsf{dom}(R, D)\,\&(D\ \mathsf{neq}\ \{\}\ \mathsf{or}\ R = \{\})$$

where $\mathsf{dom}(R, D)$ means that D is the domain of R; $D\ \mathsf{neq}\ \{\}$ means $\neg\ D = \{\}$; & means conjunction; and or, disjunction. Note that we avoid certain typing information (i.e. $R \in X \nrightarrow Y$) because {*log*} automatically adds what is really necessary (i.e. the fact that R is partial function between any two sets and that its domain is a set). When this goal is executed on the interpreter, it returns $R = \{\}, D = \{\}$.

However, perhaps a more interesting goal is the following:

$$\mathsf{dom}(R, D)\,\&\,D\ \mathsf{neq}\ \{\}\,\&\,R = \{\}$$

[4] See the paper "Adding Partial Functions to Constraint Logic Programming with Sets" by Cristiá and Rossi also accepted in this workshop.

in which case $\{log\}$ simply (and immediately) returns 'no', hence the goal is certainly unsatisfiable. And even this one:

$$\mathsf{dom}(R, D) \,\&\, D = \{\} \,\&\, R \;\mathsf{neq}\; \{\}$$

for which the answer is the same.

If the current version of $\{log\}$ answers 'no' for a goal containing partial functions then the formula is unsatisfiable; if it answers 'yes', along with an irreducible constraint C, then the validity of the answer depends on the contents of C. If C does not contain any primitive constraints for partial function management (namely, constraints dom, ran, comp and pfun) then it is surely satisfiable. Otherwise, to be sure that C is satisfiable one should add some form of variable labeling which forces unbound variables ranging on integers, as well as unbound variables ranging on partial functions, to get values from their domains. Of course this labeling process may cause the computation to become excessively long and to terminate with a time-out, in which case we are not able to get any precise answer on the satisfiability of C.

In a previous paper [10] we have shown how Z predicates can be translated into $\{log\}$ goals (the detailed translation rules are given in a technical report available on-line [9]). Given that Z is based on ZFC most of the translation rules are trivial because ZFC is one of the set theories that can be encoded in $\{log\}$. This translation turned $\{log\}$ into an efficient and effective test case generator for FASTEST, a tool implementing the TTF. This translation also implies that, in principle, it would be possible to use $\{log\}$ to prove Z/EVES theorems for which non-trivial proofs are necessary.

Example 3. This example is included for those readers unfamiliar with $\{log\}$ because we introduce some of its basic features by means of simple formulas. This presentation is intended to be informal; all the formal details can be found elsewhere [11, 24]. These features are needed to understand some formulas presented in coming sections, although most of them have an obvious meaning.

Set terms in $\{log\}$ are either $\{\}$, i.e. the empty set, or $\{X \mid s\}$ which is interpreted as $\{X\} \cup s$ where s must be another set term. Instead of writing, for instance, $\{1 \mid \{2 \mid \{\}\}\}$ it is admitted the simpler form $\{1, 2\}$.

The following are the $\{log\}$ set operators used in this paper: x in A means $x \in A$; A neq B means $\neg\, A = B$ for any two terms; $\mathsf{un}(A, B, C)$ means $C = A \cup B$; $\mathsf{subset}(A, B)$ means $A \subseteq B$; $\mathsf{ssubset}(A, B)$ means $A \subset B$; $\mathsf{inters}(A, B, C)$ means $C = A \cap B$; $\mathsf{disj}(A, B)$ means $A \cap B = \emptyset$; and $\mathsf{dres}(A, R, S)$ means $S = A \lhd R$, where $\lhd$ is the domain restriction of a binary relation to a set (i.e. if $R \in X \leftrightarrow Y$ and $A \in \mathbb{P}\, X$, then $A \lhd R = \{x : A \mid \exists\, y \in Y : (x, y) \in R\}$).

In $\{log\}$ ordered pairs can be written as lists of two elements. Then, if (a, b) is an ordered pair it becomes $[a, b]$ in $\{log\}$.

4 Motivating Example

As we have pointed out in the introduction, when the TTF is applied to Z operations it tends to generate many unsatisfiable test specifications. This is

so because test specifications are automatically generated by applying testing tactics, which are general forms for input domain partition. Each testing tactic conjoins an atomic predicate to the pre-condition of a Z operation thus introducing potential contradictions. Unsatisfiable test specifications must be eliminated because no test case can be derived from them. Eliminating such test specifications implies to be able to decide whether a first-order predicate over set theory (and integers) is satisfiable or not.

Example 4. A typical unsatisfiable test specification generated by the TTF is the following:

$$TS == [st : SYM \pfun VAL;\ s? : SYM \mid s? \notin \dom st \land st \neq \{\} \land \dom st \subset \{s?\}]$$

where SYM and VAL are two uninterpreted sorts (given sets, in Z terminology). The conjunction $st \neq \{\} \land \dom st \subset \{s?\}$ makes the test specification unsatisfiable because $\dom st \subset \{s?\}$ implies $\dom st = \{\}$, which contradicts the fact that $st \neq \{\}$ (cf. theorem PfunDomEmpty in Example 1).

Therefore, in order to decide whether TS is satisfiable or not, one option is to attempt to prove the following Z/EVES theorem:

theorem satTS
$$\forall st : SYM \pfun VAL;\ s? : SYM \bullet \neg (s? \notin \dom st \land st \neq \{\} \land \dom st \subset \{s?\})$$

If satTS is a theorem then TS (satTS's negation) is unsatisfiable. Nevertheless, proving satTS entails to prove $\neg\ st = \{\} \Rightarrow \neg\ \dom st = \{\}$, which is essentially theorem PfunDomEmpty, and it cannot be automatically proved with Z/EVES.

Another option would be to translate TS into $\{log\}$ as[5]:

$$\mathsf{dom}(st, D)\ \&\ s?\ \mathsf{nin}\ D\ \&\ st\ \mathsf{neq}\ \{\}\ \&\ \mathsf{ssubset}(D, \{s?\})$$

which is immediately found unsatisfiable by $\{log\}$. Even more, if a goal is satisfiable chances are that $\{log\}$ finds a solution for it—which becomes a test case.

5 $\{log\}$ as a Theorem Prover for Set Theory

It is fair to ask what if PfunDomEmpty is added to Z/EVES as an enabled rewrite rule, thus incrementing the chances for it to automatically prove satTS. The problems with this approach are that a) we do not know in advance all the predicates that might make a test specification unsatisfiable; and b) there is no guarantee that Z/EVES will automatically find all the unsatisfiable test

[5] In order to keep a more understandable relation between the two representations we use the same variable names in both; in practice, however, variables in $\{log\}$ formulas should be renamed according to the Prolog rules.

specifications as we explained when the rewriting strategy was introduced in Sect. 2.

During the course of our investigation about the TTF as a model-based testing method, we detected 85 predicates that make TTF test specifications to be unsatisfiable. Of these, 33 cannot be automatically proved by Z/EVES although $\{log\}$ automatically proves 28 of them. On the other hand, $\{log\}$ was able to automatically prove 46 of the 48 theorems automatically proved with Z/EVES.

Here we present some of them to give an idea of the kind of theorems we are talking about. *From now on, '**proved**' means '**automatically proved**' unless stated differently.*

Theorems proved by Z/EVES and not by $\{log\}$. As we have said above, Z/EVES proves only two theorems that $\{log\}$ does not. These are the following:

theorem Minus1
$$\forall\, n, m, k : \mathbb{N} \bullet \neg\, (n < m \land k < n - m)$$

theorem Minus2
$$\forall\, n, m, k : \mathbb{N} \bullet \neg\, (n < m \land k \leq n - m)$$

which, as can be seen, are theorems about arithmetic rather than set theory. Although $\{log\}$ handles goals including arithmetic constraints, it does it by resting on CLP(FD) [5] which is an incomplete solver, unless a finite domain is provided for all integer variables and labeling is forced for all of them. Using, for instance, CLP(Q,R) [16] as the arithmetic solver for $\{log\}$ would probably make it to prove these two theorems.

Theorems proved by Z/EVES and $\{log\}$. There are 46 theorems proved by both tools. One among the most interesting is the following one:

theorem DresEmpty $[X, Y]$
$$\forall\, A : \mathbb{P}\, X;\ R : X \nrightarrow Y \bullet \neg\, (A = \{\} \land A \lhd R \neq \{\})$$

which shows that the domain restriction ($\lhd$) of a binary relation to a set cannot be nonempty if the set is empty. Z/EVES proves it by substituting A by $\{\}$ and then automatically applying a rewrite rule called nullDres. In turn, $\{log\}$ proves it by rewriting $\mathsf{dres}(A, R, S)$ to the conjunction of primitive constraints $\mathsf{dom}(R, DR)$ & $\mathsf{inters}(A, DR, I)$ & $\mathsf{subset}(S, R)$ & $\mathsf{dom}(S, I)$ and from this, by applying the substitution $A = \{\}$, it easily obtains that $R = \{\}$ which contradicts $R \neq \{\}$.

Another theorem that is worth to be shown is the following:

theorem CapDomTotalFunctions $[X, Y]$
$$\forall\, f : X \to Y;\ a : X;\ b : Y \bullet \neg\, (\mathrm{dom}\, f \cap \mathrm{dom}\{a \mapsto b\} = \{\})$$

which states that the domain of a total function coincides with the domain of definition. Z/EVES proves it by reducing $\mathrm{dom}\{a \mapsto b\}$ to $\{a\}$, then $\mathrm{dom}\, f \cap \{a\}$ is rewritten to $a \in \mathrm{dom}\, f$ by capUnit, and finally domFunction is applied.

CapDomTotalFunctions is written in {*log*} as:

$$\mathsf{dom}(f, X)\ \&\ a\ \mathsf{in}\ X\ \&\ \mathsf{dom}(\{[a, b]\}, DS)\ \&\ \mathsf{inters}(X, DS, \{\})$$

where DS is a new unbound variable. When this goal is executed, {*log*} finds it unsatisfiable in a similar fashion: $\mathsf{dom}(\{[a, b]\}, DS)$ is rewritten to $DS = \{a\}$, while the constraint $\mathsf{dom}(f, X)\ \&\ a\ \mathsf{in}\ X$ is rewritten to $f = \{[a, Y] \mid R\}\ \&\ X = \{a \mid A\}\ \&\ \mathsf{dom}(R, A)$, where Y, R, and A are new unbound variables. Then, by applying substitution on DS and X in $\mathsf{inters}(X, DS, \{\})$, we get the constraint $\mathsf{inters}(\{a \mid A\}, \{a\}, \{\})$ which is immediately proved to be false.

Theorems proved by {log} and not by Z/EVES. Perhaps one of the most surprising theorems that Z/EVES is not able to prove is the following:

theorem CapSubsetEmpty $[X]$
$\forall A, B : \mathbb{P}\, X \bullet \neg\, (\neg\, A = \{\} \land A \cap B = \{\} \land A \subset B)$

which is translated into {*log*} as

$$A\ \mathsf{neq}\ \{\}\ \ \&\ \ \mathsf{inters}(A, B, \{\})\ \ \&\ \ \mathsf{ssubset}(A, B)$$

which is immediately proved to be unsatisfiable by rewriting $\mathsf{inters}(A, B, \{\})$ to $\mathsf{disj}(A, B)$; $\mathsf{ssubset}(A, B)$ to $\mathsf{un}(A, B, B)\ \ \&\ \ A\ \mathsf{neq}\ B$; and, using a rule that relate un and neq, constraining A in un to be the empty set.

A more complex theorem that {*log*} proves while Z/EVES does not is:

theorem DresSubsetEqual $[X, Y]$
$\forall A : \mathbb{P}\, X;\ R : X \nrightarrow Y \bullet \neg\, (\neg\, R = \{\} \land A = \mathrm{dom}\, R \land R \subset A \lhd R)$

which is translated into {*log*} as:

$$R\ \mathsf{neq}\ \{\}\ \&\ \mathsf{dom}(R, A)\ \&\ \mathsf{dres}(A, R, D)\ \&\ \mathsf{ssubset}(R, D)$$

In this case {*log*} basically rewrites $\mathsf{dres}(A, R, D)$ as:

$$\mathsf{dom}(R, DR)\ \&\ \mathsf{dom}(D, DD)\ \&\ \mathsf{inters}(A, DR, DD)\ \&\ \mathsf{subset}(D, R)$$

where DR and DD are new variables. Then it applies several inference rules until it finds an obvious contradiction.

Theorems not proved by neither. There are only five theorems in this category. The following one is interesting because it is similar to the previous one but {*log*} fails to find the contradiction.

theorem DresSubsetEqual2 $[X, Y]$
$\forall A : \mathbb{P}\, X;\ R : X \nrightarrow Y \bullet \neg\, (\neg\, R = \{\} \land A = \mathrm{dom}\, R \land A \lhd R \subset R)$

There are three theorems in this category that involve lists for which $\{log\}$ provides limited support resting most of the times on the standard Prolog facilities for lists—i.e. so far lists are not in the core of $\{log\}$ nor they are expressed as sets of ordered pairs. Furthermore, two of these involve intensional sets whose treatment is outside of the class of problems for which $\{log\}$ provides a complete solver.

If the current version of $\{log\}$ is provided with a goal including constraints on partial functions we can only trust its 'no' answers—and some of its 'yes'. This means that, in general and in this context, $\{log\}$ can be directly exploited as a prover for invalid formulas. So far we have used the power of $\{log\}$ to find unsatisfiable set predicates in order to eliminate unsatisfiable test specifications of a model-based testing method [10].

However, we think that $\{log\}$ can help Z/EVES users because many proof steps require to prove a contradiction among the hypothesis. And sometimes, as we have seen, even quite evident contradictions require non trivial proofs. So, if a Z/EVES user sees such a contradiction in the hypothesis of a goal, (s)he can send it to $\{log\}$ (after a suitable translation) and if it returns 'no' then (s)he can be sure that the goal is *true*. Thus, $\{log\}$ may reduce the number of proofs that need complex user directions. Note that, in this situation, the tool would translate into $\{log\}$ the predicates as they are, i.e. taking the Z/EVES encoding of predicates used either by the user or produced by Z/EVES itself.

6 Calling $\{log\}$ from Z/EVES

Maybe the major drawback of Z/EVES is that it is a project no longer active, although it is still used in many universities across the world. Its source code is not available neither. On the upside it is one of a handful of theorem provers for the Z notation—we can only mention two more: HOL-Z [3] and ProofPower-HOL [2], both of them with little activity in recent years and perhaps smaller user communities. For this reason we consider that it is worth to hack Z/EVES in order to try to extend or improve it. Hence, the mechanism we propose here, while far from the best, it is thought to make it work.

Z/EVES is a server program that uses the standard input and output streams to communicate with its environment. Then, it is rather easy to wrap it in the sense of the Decorator or Wrapper design pattern [14]. Let the set of user commands defined by Z/EVES be its input interface ($\mathcal{I}$), and the information printed on standard output as a result of their execution, its output interface ($\mathcal{O}$). Let {Z/EVES} be a program with input interface $\mathcal{I} \cup \{\texttt{prove by setlog}\}$, output interface $\mathcal{O} \cup \{\texttt{prove by setlog gives... true}\}$ and that behaves as follows[6]:

- It keeps track of three global lists of goals (as Z/EVES does): the proved goals, the untried goals, and the unfinished goals;

[6] We only describe the main behavior. Other interactions, such as discarding some proof steps (cf. the `back` command) behave in a similar way.

- If Z/EVES puts a goal in the proved or untried goals lists or removes a goal from the unfinished list, then so {Z/EVES} does;
- It keeps track of the list of subgoals of the current goal (these are the goals generated by a `cases` command);
- If the user issues a command in $\mathcal{I}\backslash\{$`print status`$\}$, it delegates its execution in Z/EVES and prints on standard output all the information received from Z/EVES;
- If the user issues `print status`, then {Z/EVES} calls Z/EVES but updates the lists according to the rules given here;
- If the user issues `prove by setlog`, then {Z/EVES} translates the hypothesis of the current goal into {*log*} according to the rules defined in [10] and sends it to {*log*}. If {*log*} answers 'no', then:
  - {Z/EVES} prints on standard output `prove by setlog gives... true`;
  - If the current goal is the only one in the list of subgoals, then the proof is finished, and the theorem is added to the list of proved goals;
  - If the subgoal list has more than one element, then the current subgoal is removed from that list.

  If the answer is other than 'no', then {Z/EVES} prints on standard output `Command had no effect` (i.e. Z/EVES' standard error message).

In summary, {Z/EVES} gives to the user the right impression about the state of proofs and theorems, uses Z/EVES for all the proof commands, and adds its own behavior for the new command, all with a minimal implementation. This design may be a good choice for other extensions or improvements of Z/EVES.

7 Conclusions

We have shown that {*log*} automatically proves theorems about set theory that cannot be automatically proved by Z/EVES. This observation makes us to think that {*log*} can be a good complement for Z/EVES. We have shown how {*log*} can be called from Z/EVES. This integration would make Z/EVES to automatically prove more theorems, thus improving its productivity. We believe that {*log*} is able to automatically prove these theorems because it is built on top of set theory rather than in taking sets as uninterpreted terms and reason about them with standard logical procedures. Thus, for instance, equality between set terms is not "syntactic equality" between uninterpreted terms, but "semantic equality" between set terms (i.e. equality modulo set theory). Conversely, if sets are represented as uninterpreted terms, predicates dealing with them must try to incorporate themselves the semantic properties of sets. This approach can lead to unsatisfactory behavior such as infinite computations.

The next question we would like to investigate is whether the observation made in this paper is true of other proof assistants. In particular, we would like to compare {*log*} with the automatic proof procedures implemented in Atelier-B and Isabelle/HOL because these are powerful, industrial-strength tools where theorems about set theory are among their main objectives. A fair comparison would not only include the number of theorems automatically proved by each

tool but also the number of proof rules included by each of them. In this sense and in the context of this paper, {*log*} not only proved more theorems than Z/EVES but does it implementing no more than 100 rules (including those for plain sets and those for partial functions) while Z/EVES includes around 500.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments which contributed to improve this paper. This work has been partially supported by ANPCyT PICT 2011-1002 (Argentina).

References

1. Abrial, J.R.: The B-book: Assigning Programs to Meanings. Cambridge University Press, New York, NY, USA (1996)
2. Arthan, R.D.: Some mathematical case studies in proofpower-hol. In: 17th International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2004, Emerging Trends). pp. 1–16 (2004)
3. Brucker, A.D., Rittinger, F., Wolff, B.: HOL-Z 2.0: A proof environment for z-specifications. J. UCS 9(2), 152–172 (2003), http://www.jucs.org/jucs_9_2/hol_z_2
4. Clearsy: Aterlier B home page, http://www.atelierb.eu/
5. Codognet, P., Diaz, D.: Compiling constraints in clp(FD). J. Log. Program. 27(3), 185–226 (1996)
6. Coq Development Team: The Coq Proof Assistant Reference Manual, Version 8.4pl6. LogiCal Project, Palaiseau, France (2014)
7. Cristiá, M., Albertengo, P., Frydman, C.S., Plüss, B., Rodríguez Monetti, P.: Tool support for the Test Template Framework. Softw. Test., Verif. Reliab. 24(1), 3–37 (2014)
8. Cristiá, M., Albertengo, P., Rodríguez Monetti, P.: Pruning testing trees in the Test Template Framework by detecting mathematical contradictions. In: Fiadeiro, J.L., Gnesi, S. (eds.) SEFM. pp. 268–277. IEEE Computer Society (2010)
9. Cristiá, M., Rossi, G.: Translation of ttf test specifications into {*log*} (2012), http://www.fceia.unr.edu.ar/~mcristia/publicaciones/encoding-ttf-setlog.pdf, last access: December 2012
10. Cristiá, M., Rossi, G., Frydman, C.S.: {log} as a test case generator for the Test Template Framework. In: Hierons, R.M., Merayo, M.G., Bravetti, M. (eds.) SEFM. Lecture Notes in Computer Science, vol. 8137, pp. 229–243. Springer (2013)
11. Dovier, A., Piazza, C., Pontelli, E., Rossi, G.: Sets and constraint logic programming. ACM Trans. Program. Lang. Syst. 22(5), 861–931 (2000)
12. Dovier, A., Pontelli, E., Rossi, G.: Set unification. Theory Pract. Log. Program. 6(6), 645–701 (Nov 2006), http://dx.doi.org/10.1017/S1471068406002730
13. Freitas, L.: Proving theorems with Z/EVES. Tech. rep., University of York (2004)
14. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1995)
15. Gervet, C.: Interval propagation to reason about sets: Definition and implementation of a practical language. Constraints 1(3), 191–244 (1997)

16. Holzbaur, C., Menezes, F., Barahona, P.: Defeasibility in clp(q) through generalized slack variables. In: Freuder, E.C. (ed.) CP. Lecture Notes in Computer Science, vol. 1118, pp. 209–223. Springer (1996)
17. Jackson, D.: Alloy: A logical modelling language. In: Bert, D., Bowen, J.P., King, S., Waldén, M.A. (eds.) ZB 2003: Formal Specification and Development in Z and B, Third International Conference of B and Z Users, Turku, Finland, June 4-6, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2651, p. 1. Springer (2003), http://dx.doi.org/10.1007/3-540-44880-2_1
18. Jones, C.B.: Systematic Software Development Using VDM (2Nd Ed.). Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1990)
19. Kromodimoeljo, S., Pase, B., Saaltink, M., Craigen, D., Meisels, I.: The EVES system. In: Lauer, P.E. (ed.) Functional Programming, Concurrency, Simulation and Automated Reasoning: International Lecture Series 1991-1992, McMaster University, Hamilton, Ontario, Canada. Lecture Notes in Computer Science, vol. 693, pp. 349–373. Springer (1993), http://dx.doi.org/10.1007/3-540-56883-2_16
20. Lamport, L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2002)
21. Leuschel, M., Butler, M.: ProB: A model checker for B. In: Keijiro, A., Gnesi, S., Mandrioli, D. (eds.) FME. Lecture Notes in Computer Science, vol. 2805, pp. 855–874. Springer-Verlag (2003)
22. de Moura, L.M., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS. Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer (2008)
23. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic, Lecture Notes in Computer Science, vol. 2283. Springer (2002), http://dx.doi.org/10.1007/3-540-45949-9
24. Rossi, G.: {*log*}. http://www.math.unipr.it/~gianfr/setlog.Home.html (2008), last access: December 2013
25. Saaltink, M.: The Z/EVES mathematical toolkit version 2.2 for Z/EVES version 1.5. Tech. rep., ORA Canada (1997)
26. Saaltink, M.: The Z/EVES system. In: Bowen, J.P., Hinchey, M.G., Till, D. (eds.) ZUM. Lecture Notes in Computer Science, vol. 1212, pp. 72–85. Springer (1997)
27. Spivey, J.M.: The Z notation: a reference manual. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK (1992)
28. Stocks, P., Carrington, D.: A Framework for Specification-Based Testing. IEEE Transactions on Software Engineering 22(11), 777–793 (Nov 1996)