Validating Reasoning Heuristics Using Next-Generation Theorem-Provers

Paul S. Steyn[1] and John A. van der Poll[2]

School of Computing, University of South Africa (UNISA)
`paulsteyn@gmail.com`[1] `vdpolja@unisa.ac.za`[2]

Abstract. The specification of enterprise information systems using formal specification languages enables the formal verification of these systems. Reasoning about the properties of a formal specification is a tedious task that can be facilitated much through the use of an automated reasoner. However, set theory is a corner stone of many formal specification languages and poses demanding challenges to automated reasoners. To this end a number of heuristics has been developed to aid the Otter theorem prover in finding short proofs for set theoretic problems. This paper investigates the applicability of these heuristics to a next generation theorem prover Vampire.

1 Introduction

Mathematical set theory is a building block of a number of formal specification languages, e.g. both Z [13] and B [1] are based on strongly-typed fragments of Zermelo-Fraenkel (ZF) [3] set theory. One of the advantages in using a formal notation for specifying an enterprise information system is that the specifier may formally reason about the properties of the system. In particular one may want to prove that the proposed system will behave in a certain way or that some unwanted behaviour will not occur. However, writing out such proofs is a tedious task as may be observed in [8]. Hence of particular interest to a specifier could be the feasibility of using an automated reasoning program [12, 17] to reason about such properties.

When reasoning about the properties of a specification language based on set theory, one inevitably has to move to the level of sets and the various operations defined on them. These operations in turn are based on the underlying axioms of the particular set theory in question.

1.2 Set-Theoretic Reasoning Heuristics

Set theoretic reasoning brings about a number of problems, especially if one opts for a resolution-based reasoner like Otter [6]. Much of the complexity arises from the fact that sets may be elements of other sets. Constructs in set theory are often strongly hierarchical and may lead to deeply nested structures that greatly increase a problem's search complexity [9]. In the following equality

$$\mathbb{P}(A) = \mathbb{P}(B) \leftrightarrow A = B$$

a reasoner has to transcend from the level of elements in set A to the level of elements in $\mathbb{P}(A)$ in its search for a proof, but should be prevented from transcending to the level of $\mathbb{P}(\mathbb{P}(A))$ which would greatly and unnecessarily enlarge the search space. It is generally accepted that heuristics are needed to guide reasoners, especially in the context of set-theoretic proofs [2]. One such set of heuristics for reasoning about set theory has been developed previously [15, 16], mainly through observing the behaviour of the resolution-based reasoner, Otter in its search for proofs. In total 14 heuristics, based on recognisable patterns, were developed and the question arises whether these heuristics have a wider applicability to other resolution-based reasoners, e.g. Vampire [12] and Gandalf [14]. This paper investigates the utility of the said heuristics for Vampire.

1.3 Layout of this Paper

Section 2 gives a brief introduction and justification of the use of the Vampire prover in this work. Section 3 presents the main results of our work, namely, the extent to which Vampire also needs the heuristics previously arrived at through the use of Otter. A case study in section 4 illustrates the utility of some of the heuristics on a small Z specification. We conclude with an analysis and pointers for future work.

2 The Vampire Theorem Prover

We chose Vampire [10, 12], a resolution-based automated reasoner for first-order logic with equality for evaluating the wider applicability of the 14 heuristics mentioned above for two reasons: The first is because of its consistent success at the annual CADE ATP System Competitions (CASC) [7]. The second reason stems from the fact that Vampire has solved more set-theoretic problems than any of the other competing provers in the period from 2002 to 2005 across all CASC divisions involving these problems. If we can show that Vampire benefits from the heuristics developed before, then it is plausible that other reasoners may benefit from these heuristics as well.

Vampire is a saturation-based reasoner and implements three different saturation algorithms that can be selected for its main loop for inferring and processing clauses. The three saturation algorithms are an Otter loop with or without the Limited Resource Strategy and the Discount loop. These algorithms belong to the class of given-clause algorithms. Vampire’s algorithm is a slight modification of the saturation algorithm used by Otter [6].

The Limited Resource Strategy [11] aims to improve the efficiency of the Otter algorithm when a time limit is imposed by identifying and discarding passive clauses that have little chance to be processed within the time limit. The Limited Resource Strategy is therefore not a complete proof procedure.

3 Evaluation of Set-Theoretic Reasoning Heuristics

In this section we measure the utility of some previously developed heuristics [15, 16] for Vampire. Fourteen heuristics were originally developed, but for reasons of space we evaluate 5 heuristics. Our experiments follow a pattern: First we present our sample problem and the ZF axiom(s) on which the problem is based. Then we report the performance of Otter in an attempt to solve the problem. From a failed proof attempt we define a heuristic that allows Otter to successfully solve the problem. Next Vampire is used on the original problem to determine its need for the particular heuristic. In some cases we increase the complexity of the problem as an additional test.

We used Vampire version 8.0 that was also used at the CADE ATP System Competition [7] in 2005 (CASC-20). A time limit of 30 minutes and a memory limit of 128MB were imposed which causes Vampire to use its limited resource strategy. No changes were made to Vampire's other default settings.

3.1 Equality versus Extensionality

Our first sample problem based on equality and the power set axiom is given by:

$$\mathbb{P}\{\{1\}\} = \{\varnothing, \{\{1\}\}\} \tag{1}$$

Currently neither Otter nor Vampire accept formulae in the highly evolved notation of ZF set theory, hence the user has to rewrite set-theoretic formulae like (1) above in a weaker first-order language. Therefore, proof obligation in (1) is rewritten as:

$$A = \{1\} \wedge B = \{A\} \wedge C = \mathbb{P}(B) \wedge D = \{\varnothing, B\} \rightarrow C = D \tag{2}$$

Further decomposition is required for $\mathbb{P}(B)$ as follows:

$$\forall x(x \in C \leftrightarrow \forall y(y \in x \rightarrow y \in B)) \tag{3}$$

Otter finds no proof for (2) in 20 minutes. Next, using the extensionality axiom we replace the consequent *(C=D)* by

$$\forall x(x \in C \leftrightarrow x \in D) \tag{4}$$

and this allows Otter to find a proof in 0.03 seconds. These findings lead to the following heuristic (for the sake of this paper we call it Heuristic #1):

Heuristic #1: Use the principle of extensionality to replace set equality with the condition under which two sets are equal, i.e., when their elements are the same.

When the same problem (2) is given to Vampire, it has no difficulty in finding a proof in 1.3 seconds. The application of the above extensionality heuristic leads to an equally fast proof in 0.1 seconds. These times are too short to determine the utility of the heuristic for Vampire. However, consider the following, more complex example involving a subset axiom of arbitrary intersection:

$$\bigcap \{\{1,2,3\}, \{2,3,4\}\} = \{2,3\} \tag{5}$$

As before formula (5) is rewritten to make the relevant constructions explicit:

$$A = \{1,2,3\} \wedge B = \{2,3,4\} \wedge C = \{A,B\} \wedge D = \{2,3\} \rightarrow \cap C = D \tag{6}$$

This time Vampire finds no proof within 30 minutes. When we however apply the principle of extensionality to the consequent of formula (6) as in

$$\forall x(x \in \cap C \leftrightarrow x \in D) \tag{7}$$

then Vampire finds a short proof in *0.4 seconds*. Therefore Heuristic #1 appears to be useful for Vampire as well, depending on the complexity of the problem.

3.2 Nested Functors

An effective heuristic is to give preference to deductions containing smaller clauses [5], i.e. clauses containing fewer literals or clauses of smaller term depth. The use of nested function symbols (called *functors*) leads to larger term depth and complicates unification. The nesting of function symbols occurs often, e.g.:

$$(A + B) + C = A + (B + C) \tag{8}$$

Formula (8) states that set-theoretic symmetric difference (denoted by '+') is associative. The symmetric difference of sets A and B is defined as $A + B = (A - B) \cup (B - A) = \{x \mid ((x \in A) \wedge (x \notin B)) \vee ((x \notin A) \wedge (x \in B))\}$. Therefore formula (8) employs equality as well as a ZF subset axiom as instantiated by set-theoretic difference. A first-order definition of the symmetric difference functor is:

$$\forall A \forall B \forall x(x \in \mathrm{symmdiff}(A,B) \leftrightarrow ((x \in A \wedge x \notin B) \vee (x \notin A \wedge x \in B))) \tag{9}$$

The conclusion of the proof is then stated as:

$$\begin{gathered}\forall x(x \in \mathrm{symmdiff}(\mathrm{symmdiff}(A,B), C) \leftrightarrow \\ x \in \mathrm{symmdiff}(A, \mathrm{symmdiff}(B,C)))\end{gathered} \tag{10}$$

With this formulation it takes Otter 4 minutes 3 seconds to find a proof of (10). Unfolding, and thereby effectively removing, the nested functors as

$$\begin{gathered}D = A + B \wedge E = D + C \wedge F = B + C \wedge G = A + F \rightarrow \\ \forall x(x \in E \leftrightarrow x \in G)\end{gathered} \tag{11}$$

allows Otter to find a proof in only 0.17 seconds, suggesting:

Heuristic #2: Avoid, if possible, the use of nested functor symbols in definitions.

Vampire quickly finds a proof of (10) in less than 0.1 seconds, both with or without the use of the nested functor heuristic. We therefore increase the complexity of the problem to further investigate the utility of Heuristic #2 for Vampire. Note that in both problem formulations the extensionality heuristic was already applied to problem conclusions. Rewriting (10) without using extensionality as

$$\text{symmdiff}(\text{symmdiff}(A,B), C) = \text{symmdiff}(A, \text{symmdiff}(B,C)) \tag{12}$$

results in Vampire finding no proof after 30 minutes. Next we apply the nested functor heuristic by rewriting our problem using Skolem constants:

$$D = A + B \wedge E = D + C \wedge F = B + C \wedge G = A + F \rightarrow E = G \tag{13}$$

Vampire now finds a proof after only *0.5 seconds*.

3.3 Divide-and-Conquer

The heuristic examined in this section is applicable to proofs where the consequence of the proof contains a set equality or an if-and-only-if formula. A set equality in the conclusion of a proof implies ‘if and only if’ via the axiom of extensionality. Owing to the if-and-only-if formula, a specifier can perform two separate proofs, one for the only-if part and another proof for the if part. Consider the following sample problem based on equality and the power set axiom:

$$\mathbb{P}\{0,1\} = \{\emptyset, \{0\}, \{1\}, \{0,1\}\} \tag{14}$$

The formula is rewritten to make the relevant constructions explicit:

$$A = \{0\} \wedge B = \{1\} \wedge C = \{0,1\} \wedge D = \mathbb{P}(C) \wedge E = \{\emptyset, A, B, C\} \rightarrow D = E \tag{15}$$

Otter terminates without finding a refutation after 30 minutes. We resort to our extensionality heuristic by changing the conclusion to:

$$\forall x(x \in D \leftrightarrow x \in E) \tag{16}$$

Otter now finds a proof in 3 minutes 23 seconds. An alternative approach is to perform two separate proofs, one for each half of (16) and in the two proofs specify the conclusions as in (17) and (18) below.

$$\forall x(x \in D \rightarrow x \in E) \tag{17}$$

$$\forall x(x \in E \rightarrow x \in D) \tag{18}$$

Otter proves (17) and (18) in 0.43 and 0.03 seconds respectively, leading to:

Heuristic #3: Perform two separate subset proofs whenever the problem at hand requires one to prove the equality of two sets.

Vampire is also unable to find a proof for (15) after 30 minutes. However for (16), (17) and (18) Vampire finds quick proofs in 0.8, 0.3 and 0.1 seconds respectively. These times are too short to affirm the utility of the divide-and-conquer heuristic for Vampire. As before we increase the complexity of the problem through the equality:

$$\mathbb{P}\{0,1,2\} = \{\emptyset, \{0\}, \{1\}, \{2\}, \{0,1\}, \{0,2\}, \{1,2\}, \{0,1,2\}\} \tag{19}$$

Formula (19) is again rewritten to make the relevant constructions explicit:

$$A = \{0\} \wedge B = \{1\} \wedge C = \{2\} \wedge D = \{0,1\} \wedge E = \{0,2\} \wedge F = \{1,2\} \wedge G = \{0,1,2\} \wedge H = \mathbb{P}(G) \wedge I=\{\emptyset, A, B, C, D, E, F, G\} \rightarrow H=I \qquad (\mathbf{20})$$

Vampire terminates without finding a refutation after 8 minutes 53 seconds with the message ‘no passive clauses left’. Note that this does not mean that a refutation does not exist. Since Vampire was run with both a time and memory limit, it uses the limited resource strategy [11], which is not a complete search strategy. Applying our extensionality heuristic by rewriting (H = I) above as

$$\forall x(x \in H \leftrightarrow x \in I) \qquad (\mathbf{21})$$

allows Vampire to find a proof after 8 minutes 40 seconds which is still too long. By applying divide-and-conquer to (21) in the usual way allows Vampire to find short proofs in 28 secs and 2 secs respectively, illustrating the utility of the heuristic.

3.4 Exemplification

When writing the contents of sets in list notation one naturally tends to define these sets using one or more levels of indirection by moving from the various elements to a symbol representing the collection of those elements. The sample problem used for the divide-and-conquer heuristic will be used here as well, viz:

$$\mathbb{P}\{0,1\} = \{\emptyset, \{0\}, \{1\}, \{0,1\}\} \qquad (\mathbf{22})$$

Recall that Otter failed to find a proof in 30 minutes for the initial unfolding in (15). Suppose we remove one level of indirection by eliminating symbol E, i.e.

$$A = \{0\} \wedge B = \{1\} \wedge C = \{0,1\} \wedge D = \mathbb{P}(C) \rightarrow D = \{\emptyset, A, B, C\} \qquad (\mathbf{23})$$

where D = {∅, A, B, C} is unfolded (repeatedly using the ZF pairing axiom) as

$$\forall x(x \in D \leftrightarrow (x = \emptyset \vee x = A \vee x = B \vee x = C)) \qquad (\mathbf{24})$$

in the proof conclusion. With this formulation Otter finds a proof in 4 minutes 5 seconds. These results lead us to the following heuristic:

Heuristic #4: Avoid unnecessary levels of elementhood in formulae by using the elements of sets directly.

The divide-and-conquer heuristic can be applied to this last proof attempt to yield proofs in 0.34 and 0.03 seconds for the ‘only-if’ and ‘if’ directions respectively. Vampire was also unable to find a proof for (15) within 30 minutes. However, for (23) Vampire finds a proof in *0.8 seconds*. In this example, therefore, it was not necessary to increase the complexity of the problem to illustrate the utility of the heuristic for Vampire. If we do increase the complexity of the problem by again using formula (19) as an example, but instead of unfolding it as in (20) we unfold it as

$$A = \{0\} \wedge B = \{1\} \wedge C = \{2\} \wedge D = \{0, 1\} \wedge E = \{0, 2\} \wedge F = \{1, 2\} \wedge G = \{0, 1 ,2\} \wedge H = \mathbb{P}(G) \rightarrow H = \{\emptyset, A, B, C, D, E, F, G\} \qquad (\mathbf{25})$$

then Vampire finds a proof in 5 minutes and 50 seconds. The divide-and-conquer heuristic can be applied to this last proof attempt to yield proofs in *31.5* and *1.6 seconds* for the 'only-if' and 'if' directions respectively.

3.5 Multivariate Functors

Functors containing variables as arguments lead to more unifications, which in turn lead to a larger search space. Functors are often introduced by Skolemisation [4], which occurs when first order formulae are clausified to serve as input to the resolution mechanism. If an existential quantifier occurs within the scope of any universal quantifiers, the existential quantifier is replaced by a Skolem functor taking each of the universally quantified variables as an argument.

The example problem (15) will be used again with the extensionality heuristic applied to the conclusion as in (16). First we define the term $D = \mathbb{P}(C)$ indirectly as

$$\forall x(x \in D \leftrightarrow x \subseteq C) \tag{26}$$

where the subset functor $\subseteq$ is defined as

$$\forall A \forall B(A \subseteq B \leftrightarrow \forall y(y \in A \rightarrow y \in B)) \tag{27}$$

With this formulation Otter finds no proof in 30 minutes. The clausification of (27) results in variable *y* being replaced by a Skolem function of the two variables *A* and *B*. The effect of Skolemisation may be reduced by eliminating one of the universally quantified variables in (27), e.g. replace variable *B* by the constant *C* in (26):

$$\forall A(A \subseteq C \leftrightarrow \forall y(y \in A \rightarrow y \in C)) \tag{28}$$

Now Otter finds a proof after 4 minutes 5 seconds. Variable *y* in the clausal form of (28) is now replaced by a Skolem functor of only one variable as opposed to a functor of two variables in (27). The possibility of irrelevant unifications with this Skolem functor has therefore been reduced. It should also be noted that the subset functor $\subseteq$ in both cases has an arity of two, but in (27) it contains two variables as opposed to one constant and one variable in (28). These results lead to:

Heuristic #5: Simplify terms in sets by either not involving functors, or else functors with the minimum number of argument positions taken up by variables.

Vampire finds quick proofs with or without the heuristic applied. With the subset functor formulated as in (27) it finds a proof in 21 seconds and for (28) in 0.1 seconds. The relative improvement in search time is significant. However, the search time for (27) may still be too low to seriously justify the use of the heuristic. We therefore increase the complexity of the problem to further test our heuristic. The example problem (20) that was also used in the divide-and-conquer heuristic has sufficient complexity and will be used again with the extensionality heuristic applied to the conclusion as in (21). As before, the term $H = \mathbb{P}(G)$ is unfolded as

$$\forall x(x \in H \leftrightarrow x \subseteq G) \tag{29}$$

where the subset functor $\subseteq$ is again defined as in (27). With this formulation Vampire finds no proof in 30 minutes. We next apply the multivariate functor heuristic by defining the subset functor with variable *B* replaced by the constant *G*:

$$\forall A(A \subseteq G \leftrightarrow \forall y(y \in A \rightarrow y \in G)) \qquad \mathbf{(30)}$$

Now Vampire finds a proof after 1 minute and 32 seconds. This result can further be improved through divide-and-conquer. The times for the two sub-proofs are 5.2 and 0.3 seconds respectively.

4 Case Study: Football Fan Register

The following case study serves as a very small example of the specification of an enterprise information system using Z and the subsequent reasoning about one of its properties using the heuristics of the previous section.
A Football Identity Scheme allocates each fan a single unique identity code. It also keeps a list of troublemakers who have been banned from attending matches. *PERSON* and *ID* are two given sets and represent the set of people and the set of all possible identity codes. The system state is recorded by *FIS* [8]*:*

FIS

members: *ID* ⤔ *PERSON;* *banned*: ℙ *ID*

---

banned $\subseteq$ dom *members*

The partial injective function *members* maps identity codes to fans. The set *banned* is a set of banned identity codes and is a subset of the domain of *members*.
Schema *AddMember* adds members to the system. It takes a person as input and returns a newly allocated identity code.

AddMember

Δ*FIS*
person?: *PERSON;* *id*!: *ID*

---

person? $\notin$ ran *members* $\wedge$ *id*! $\notin$ dom *members*
members′ = *members* $\cup$ {*id*! $\mapsto$ *person*?} $\wedge$ *banned*′ = *banned*

A Proof Obligation

Next we show how some of the above heuristics may be used to successfully discharge a proof obligation arising from the specification. We want to show that *members*′ is still an injective function. The following are given as input to Vampire:

$$\text{members} \in \text{rel(id,person)} \wedge \text{isSiv(members)} \wedge \text{isInjective(members)} \quad \textbf{(31)}$$

$$\text{banned} \in \mathbb{P}\text{(id)} \wedge \text{banned} \subseteq \text{dom(members)} \wedge \text{person?} \in \text{person} \wedge \text{id!} \in \text{id} \quad \textbf{(32)}$$

$$\text{person?} \notin \text{ran(members)} \quad \textbf{(33)}$$

$$\text{id!} \notin \text{dom(members)} \wedge \text{![M]: (M} \in \text{newMembers} \Leftrightarrow \text{M=ord(id!,person?))} \quad \textbf{(34)}$$

$$\text{members'} = \text{members} \cup \text{newMembers} \wedge \text{banned'} = \text{banned} \quad \textbf{(35)}$$

These facts represent the state *FIS* and operation *AddMember* Formula (31) states that *members* is a relation that is single valued and injective, i.e. a partial injective function [13]. The axioms for *rel*, *isSiv*, *isInjective*, *dom*, *ran*, *subset*, *union* etc. are not shown here but are part of the input to Vampire.

The proof obligation is stated as:

$$\text{members'} \in \text{rel(id,person)} \wedge \text{isSiv(members')} \wedge \text{isInjective(members')} \quad \textbf{(36)}$$

Vampire finds no proof for (36) in 30 minutes. The divide-and-conquer heuristic can be applied to (36), resulting in three separate sub-proofs with consequents:

$$\text{members'} \in \text{rel(id,person)} \quad \textbf{(37)}$$

$$\text{isSiv(members')} \quad \textbf{(38)}$$

$$\text{isInjective(members')} \quad \textbf{(39)}$$

Vampire finds proofs for (38) and (39) in 14 minutes 48 seconds and 14 minutes 24 seconds respectively, but fails to find a proof for (37) after 30 minutes. Next we apply the multivariate functor heuristic by removing axioms for union, domain, injectivity, single valued ness, power set, range, relation and subset and replace them by instances of the same axioms where some variables are replaced by constants. For example, (33) requires the following definition for the range of a relation:

$$\forall \text{R} \; \forall \text{Y[Y} \in \text{ran(R)} \Leftrightarrow \exists\text{(X)(ord(X,Y)} \in \text{R)]} \quad \textbf{(40)}$$

A replacement instance of (40) is therefore added to the proof attempt where variable *R* is replaced with constant *members*:

$$\forall \text{Y[Y} \in \text{ran(members)} \Leftrightarrow \exists\text{(X)(ord(X,Y)} \in \text{members)]} \quad \textbf{(41)}$$

Vampire now finds quick proofs for (38) and (39) in 4 and 7 seconds respectively. Vampire still cannot find a proof for (37) in 30 minutes. We finally apply the nested functor heuristic to all the introduced axiom instances like (41). For example, (41) contains the nested functors *el(Y,ran(members))* and is replaced by:

$$\text{ranMems} = \text{ran(members)} \wedge \quad \textbf{(42)}$$
$$\forall \text{Y[Y} \in \text{ranMems} \Leftrightarrow \exists\text{(X)(ord(X,Y)} \in \text{members)]}$$

Vampire finds a solution for sub-proof (37) in 9 minutes and 18 seconds. Solutions for (38) and (39) are also found slightly faster in 2 and 4 seconds respectively.

5 Conclusions and Future Work

In this paper we investigated to what extent a previously developed set of heuristics to facilitate proofs in set theory for a resolution-based automated reasoner are applicable to another reasoner with similar characteristics. The Vampire theorem prover was chosen for this task owing to its steadfast performance at recent CASC competitions. We evaluated 5 heuristics and found that all these heuristics are indeed needed, even though the original problem often had to be enlarged to illustrate the utility of the given heuristic using the new reasoner. Our heuristics appear to have an even larger support base since we also tested these on another reasoner, namely, Gandalf [14] and comparable results as reported on in this paper were witnessed.

Future work in this area may include an investigation into the applicability of the rest of our heuristics. Preliminary results indicate that at least 11 of the original 14 heuristics are useful, some addressing the challenge of tuples and functors with arity 6 or more [15].

References

1. Abriel, J-R. 1996. The B Book: Assigning Programs to Meanings. Cambridge University Press.
2. Bundy, A. 1999. A Survey of Automated Deduction. Tech. Rep. EDI-INF-RR-0001, Division of Informatics, University of Edinburgh. April.
3. Enderton, H. 1977. Elements of Set Theory. Academic Press, Inc.
4. Hamilton A. G. 1991. Logic for Mathematicians. Revised edition. Cambridge University Press.
5. Leitsch A. 1997. The resolution calculus. Springer-Verlag, New York.
6. McCune, W. W. 2003. OTTER 3.3 Reference Manual. Argonne National Laboratory, Argonne, Illinois. ANL/MCS-TM-263.
7. Pelletier, F. J., Sutcliffe, G., and Suttner, C. 2002. The development of CASC. AI Communications 15(2), 79-90.
8. Potter, B., Sinclair, J. and Till D., An Introduction to Formal Specification and Z, Prentice Hall, 1996.
9. Quaife, A. 1992. Automated Development of Fundamental Mathematical Theories. Automated Reasoning Series. Kluwer Academic Publishers.
10. Riazanov, A. 2003. Implementing an Efficient Theorem Prover. Ph.D. thesis, University of Manchester.
11. Riazanov, A. and Voronkov, A. 2003. Limited resource strategy in resolution theorem proving. Journal of Symbolic Computing 36(1-2), 101-115
12. Riazanov, A. and Voronkov, A. 2002. The design and implementation of VAMPIRE. AI Communications 15(2), 91-110.
13. Spivey, J. M. 1992. The Z Notation: A Reference Manual, 2nd ed. Prentice-Hall, London.
14. Tammet, T. 1997. Gandalf. Journal of Automated Reasoning 18(2), 199-204.
15. van der Poll, J. A. 2000. Automated Support for Set-Theoretic Specifications. Ph.D. thesis, University of South Africa.
16. van der Poll, J. A. and Labuschagne, W. A. 1999. Heuristics for Resolution-Based Set-Theoretic Proofs. South African Computer Journal Issue 23, 3 – 17.
17. Wos, L. 2006. Milestones for automated reasoning with OTTER. International Journal on Artificial Intelligence Tools, 15 (1): 3 – 19, February.